

```
In [2]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [3]: df=pd.read_csv("Future of Jobs AI Dataset.csv")
df.head()
```

Out[3]:

	job_title	country	experience_level	education_level	year	salary	ai_risk_score	primary_skill	skill_demand_score	job_openings	job_survival_class
0	Data Scientist	USA	Senior	Master	2022	61931.03	0.32	Python	88	39158	2
1	Software Engineer	India	Senior	Bachelor	2032	21330.84	0.52	Java	80	7265	1
2	Data Scientist	Canada	Mid	Bachelor	2015	24211.17	0.25	Python	70	22962	1
3	Data Scientist	India	Entry	PhD	2034	11794.86	0.44	Python	95	17023	1
4	Data Analyst	Canada	Entry	PhD	2035	17999.26	0.75	SQL	61	3433	0



```
In [5]: df.isnull().sum()
```

Out[5]:

job_title	0
country	0
experience_level	0
education_level	0
year	0
salary	0
ai_risk_score	0
primary_skill	0
skill_demand_score	0
job_openings	0
job_survival_class	0
salary_bucket	0
ai_risk_category	0
dtype:	int64

In [6]: `from sklearn.preprocessing import LabelEncoder` *#change categorical data to numerical data*

```
le = LabelEncoder()

categorical_cols = ['experience_level', 'education_level', 'salary_bucket', 'ai_risk_category']

for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

df.head()
```

Out[6]:

	job_title	country	experience_level	education_level	year	salary	ai_risk_score	primary_skill	skill_demand_score	job_openings	job_survival_class
--	-----------	---------	------------------	-----------------	------	--------	---------------	---------------	--------------------	--------------	--------------------

0	Data Scientist	USA	2	1	2022	61931.03	0.32	Python	88	39158	2
1	Software Engineer	India	2	0	2032	21330.84	0.52	Java	80	7265	1
2	Data Scientist	Canada	1	0	2015	24211.17	0.25	Python	70	22962	1
3	Data Scientist	India	0	2	2034	11794.86	0.44	Python	95	17023	1
4	Data Analyst	Canada	0	2	2035	17999.26	0.75	SQL	61	3433	0

In [7]: `from sklearn.preprocessing import LabelEncoder` *#explanation for numbers*

```
categorical_cols = ['experience_level', 'education_level', 'salary_bucket', 'ai_risk_category']

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

    print(f"\nColumn: {col}")
    print(dict(zip(le.classes_, le.transform(le.classes_))))
```

Column: experience_level
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

Column: education_level
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

Column: salary_bucket
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

Column: ai_risk_category
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

```
In [8]: df.describe()
```

Out[8]:

	experience_level	education_level	year	salary	ai_risk_score	skill_demand_score	job_openings	job_survival_class	salary_bucket
count	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000
mean	1.009722	1.000891	2025.054525	34553.729166	0.396008	79.453374	25223.652029	1.310054	0.393097
std	0.818520	0.814957	6.058714	18024.702961	0.187665	11.483535	14163.309081	0.658417	0.761169
min	0.000000	0.000000	2015.000000	3875.170000	0.050000	60.000000	1002.000000	0.000000	0.000000
25%	0.000000	0.000000	2020.000000	21331.405000	0.260000	70.000000	12993.000000	1.000000	0.000000
50%	1.000000	1.000000	2025.000000	31573.380000	0.360000	79.000000	24896.000000	1.000000	0.000000
75%	2.000000	2.000000	2030.000000	45307.845000	0.490000	89.000000	37440.500000	2.000000	0.000000
max	2.000000	2.000000	2035.000000	113589.260000	0.850000	99.000000	49998.000000	2.000000	2.000000

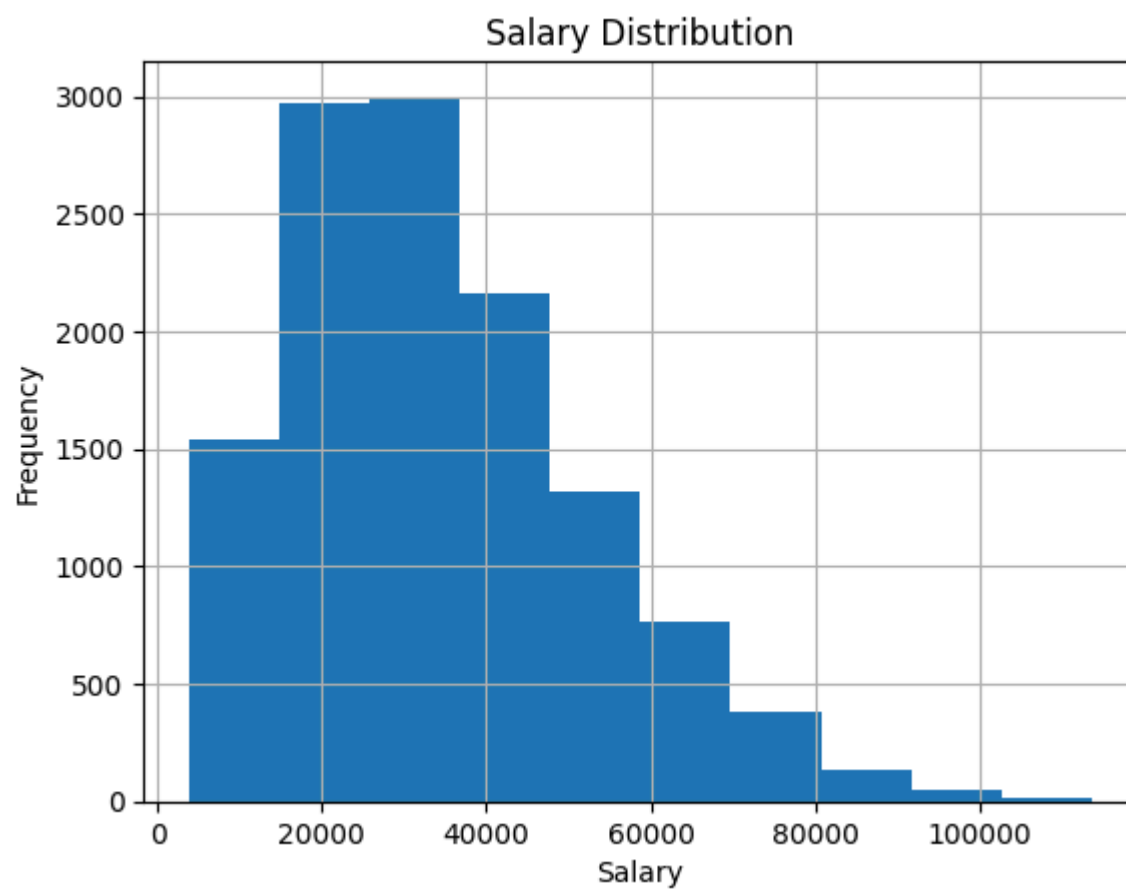


```
In [9]: df.info()
```

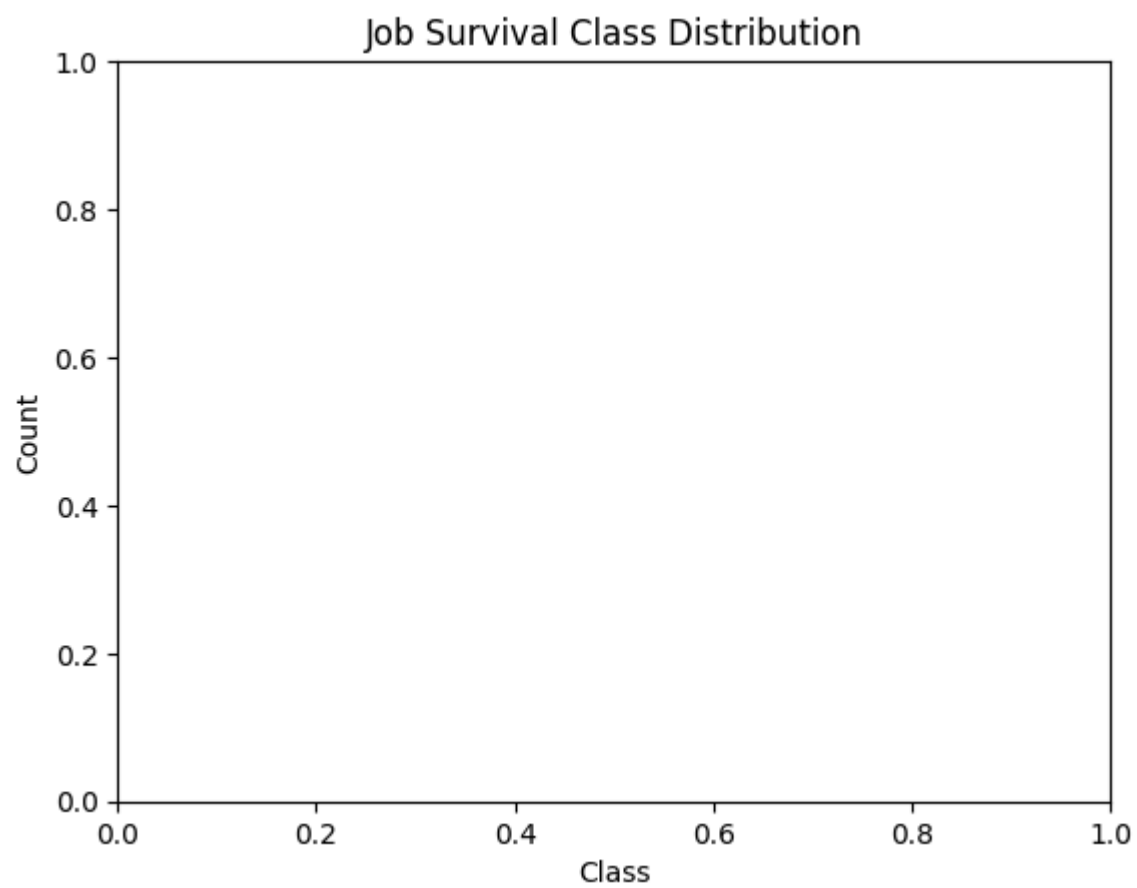
```
<class 'pandas.DataFrame'>
RangeIndex: 12343 entries, 0 to 12342
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   job_title             12343 non-null  str
 1   country               12343 non-null  str
 2   experience_level      12343 non-null  int64
 3   education_level       12343 non-null  int64
 4   year                  12343 non-null  int64
 5   salary                12343 non-null  float64
 6   ai_risk_score          12343 non-null  float64
 7   primary_skill         12343 non-null  str
 8   skill_demand_score    12343 non-null  int64
 9   job_openings          12343 non-null  int64
10   job_survival_class    12343 non-null  int64
11   salary_bucket         12343 non-null  int64
12   ai_risk_category      12343 non-null  int64
dtypes: float64(2), int64(8), str(3)
memory usage: 1.2 MB
```

```
In [10]: import matplotlib.pyplot as plt
```

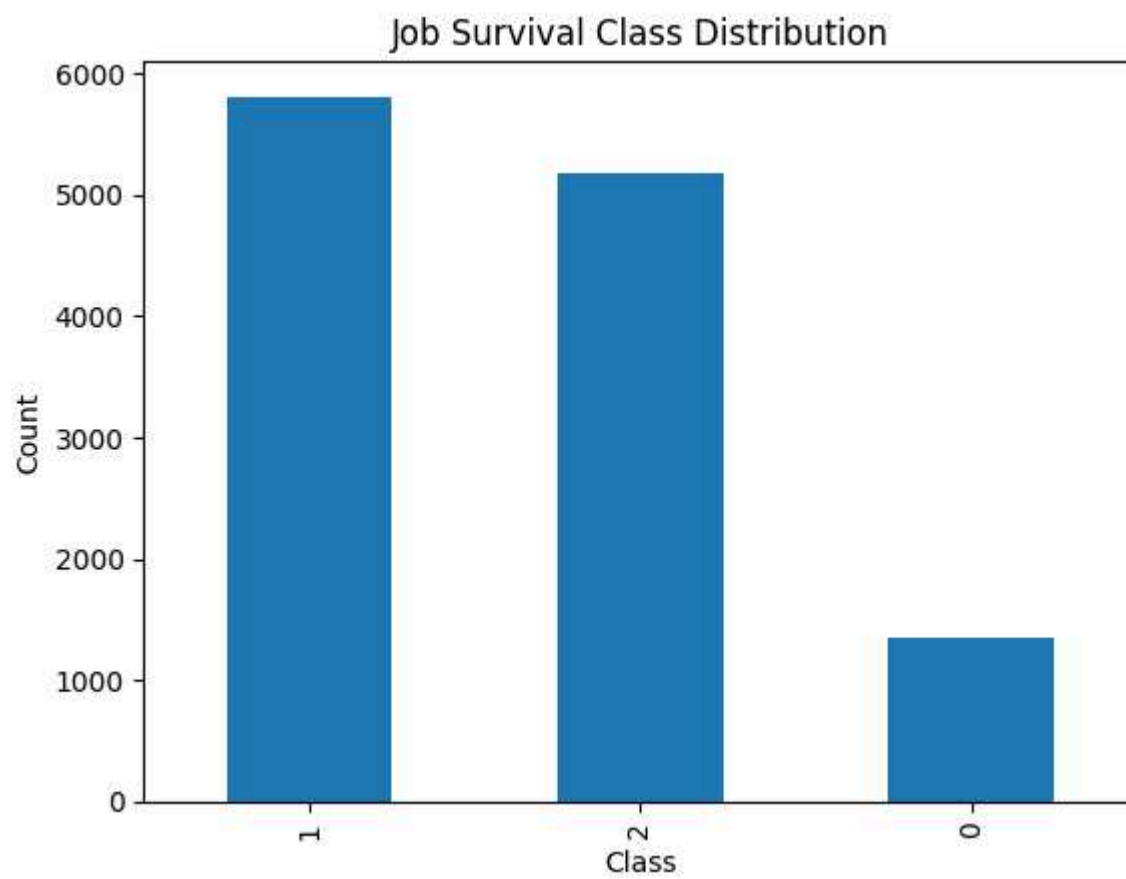
```
df['salary'].hist()
plt.title("Salary Distribution")
plt.xlabel("Salary")
plt.ylabel("Frequency")
plt.show()
```



```
In [11]: df['job_survival_class'].value_counts().plot
plt.title("Job Survival Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```

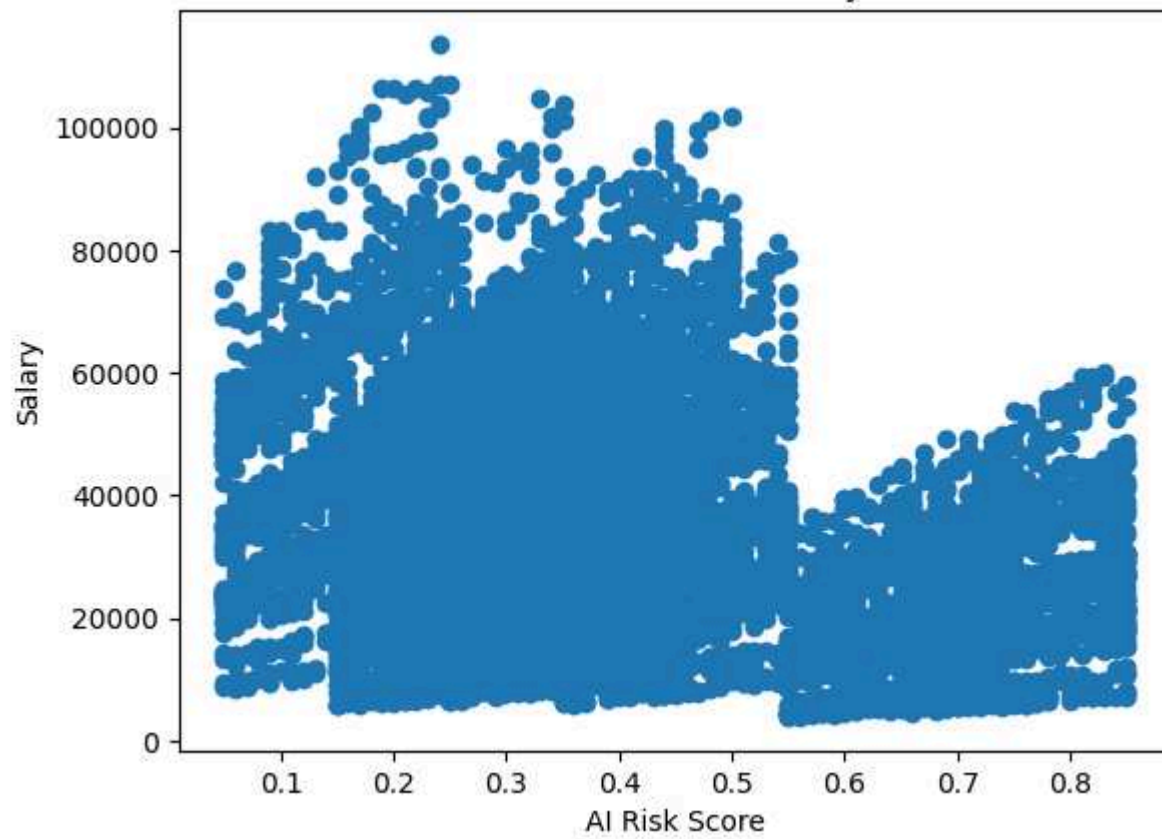


```
In [12]: df['job_survival_class'].value_counts().plot(kind='bar')
plt.title("Job Survival Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```

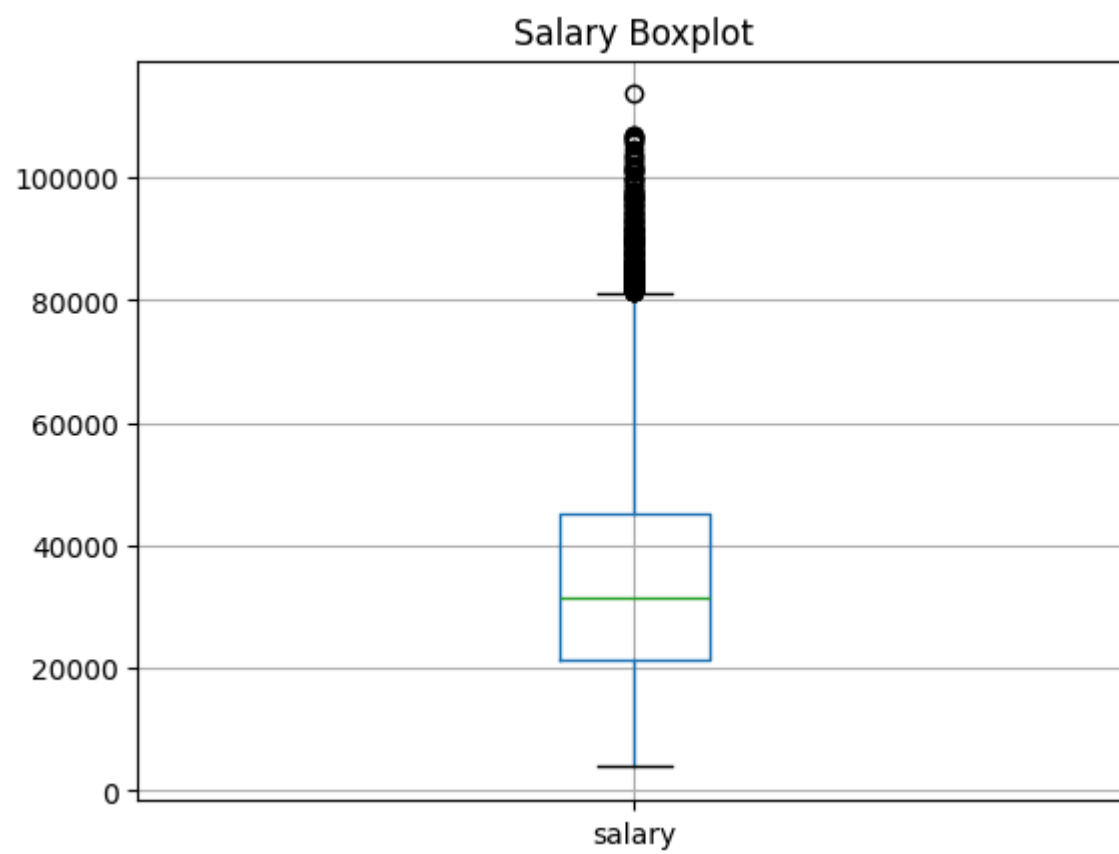


```
In [13]: plt.scatter(df['ai_risk_score'], df['salary'])
plt.title("AI Risk Score vs Salary")
plt.xlabel("AI Risk Score")
plt.ylabel("Salary")
plt.show()
```

AI Risk Score vs Salary

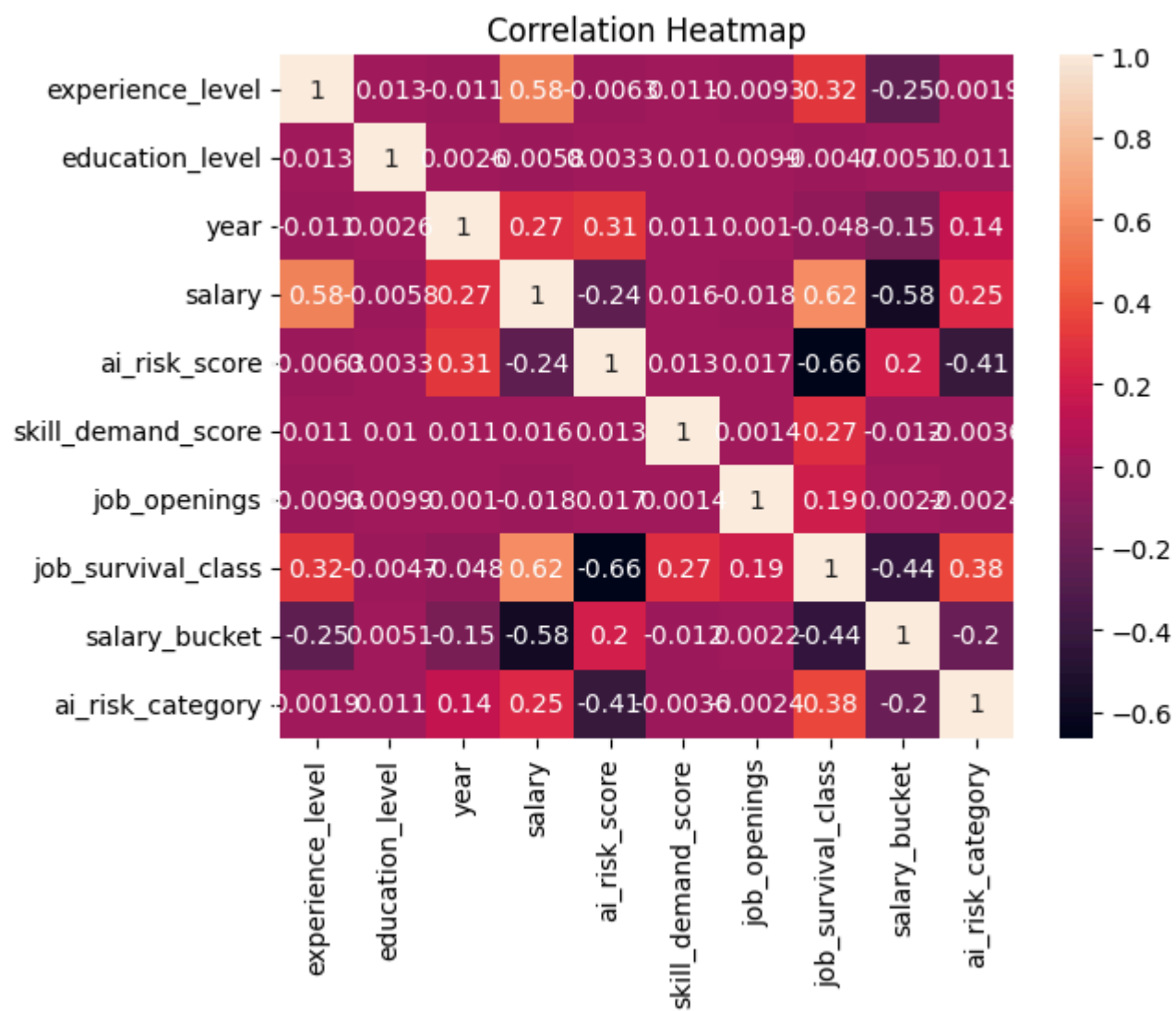


```
In [14]: df.boxplot(column='salary')
plt.title("Salary Boxplot")
plt.show()
```



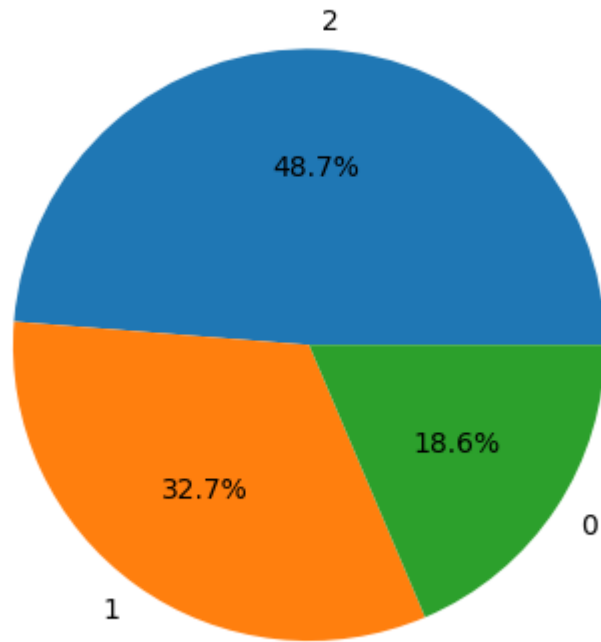
```
In [15]: import seaborn as sns

corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
plt.title("Correlation Heatmap")
plt.show()
```

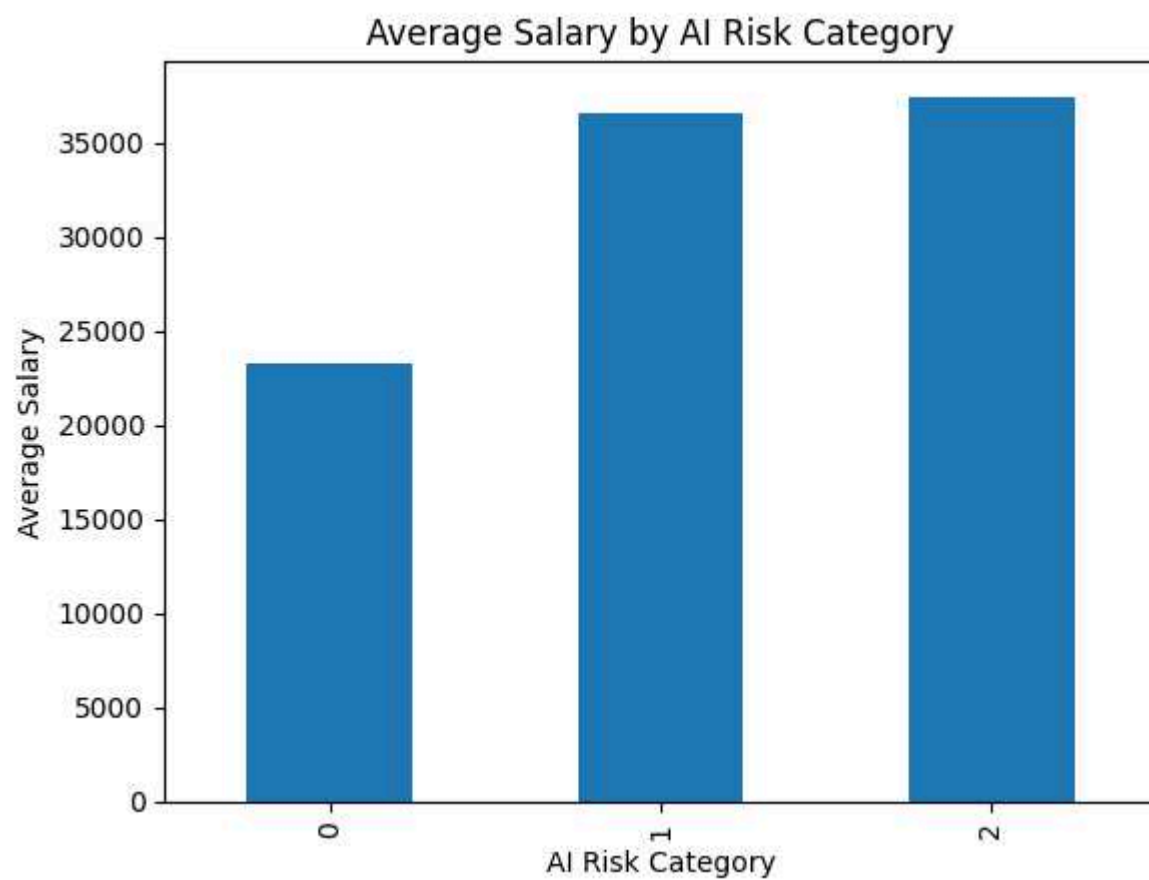


```
In [16]: df['ai_risk_category'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title("AI Risk Category Distribution")
plt.ylabel("")
plt.show()
```

AI Risk Category Distribution

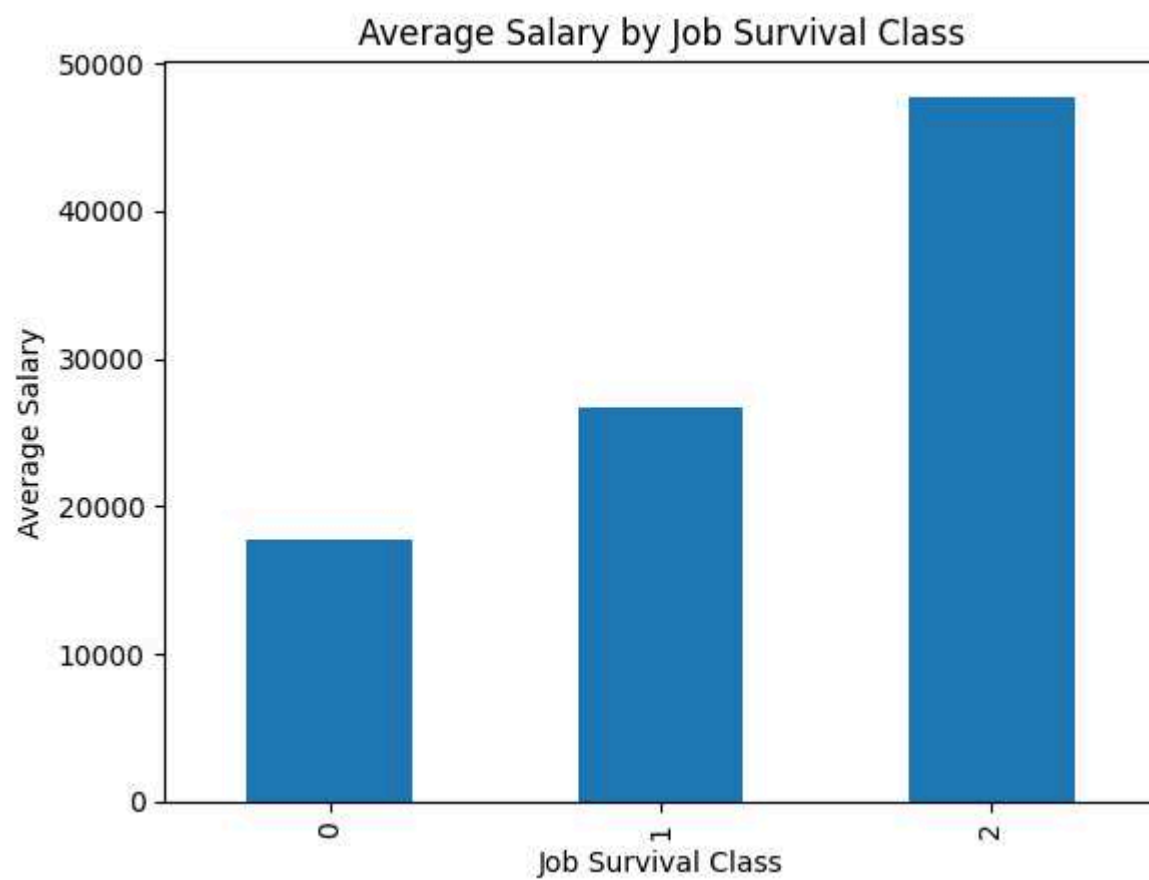


```
In [17]: df.groupby('ai_risk_category')['salary'].mean().plot(kind='bar')
plt.title("Average Salary by AI Risk Category")
plt.xlabel("AI Risk Category")
plt.ylabel("Average Salary")
plt.show()
```



```
In [25]: import matplotlib.pyplot as plt

df.groupby('job_survival_class')['salary'].mean().plot(kind='bar')
plt.title("Average Salary by Job Survival Class")
plt.xlabel("Job Survival Class")
plt.ylabel("Average Salary")
plt.show()
```



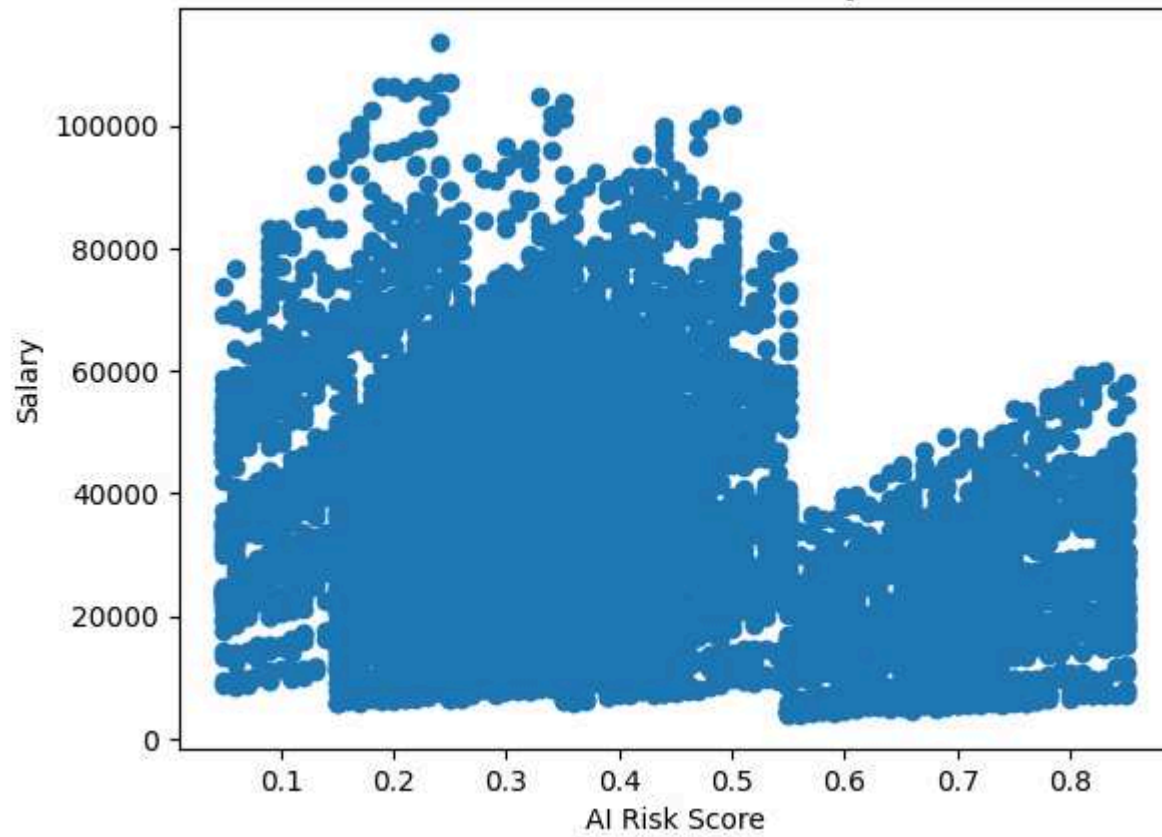
```
In [26]: import matplotlib.pyplot as plt

df.groupby('experience_level')['salary'].mean().plot(kind='bar')
plt.title("Average Salary by Experience Level")
plt.xlabel("Experience Level")
plt.ylabel("Average Salary")
plt.show()
```

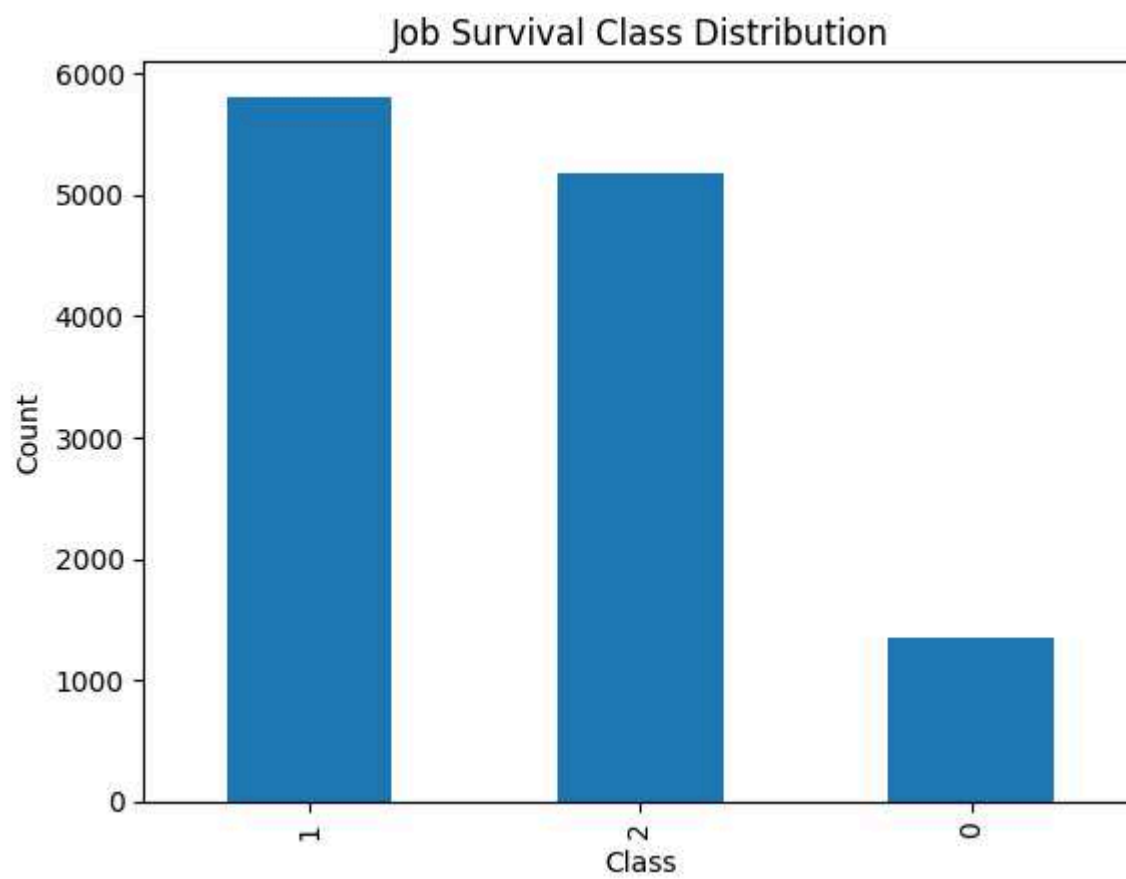


```
In [27]: plt.scatter(df['ai_risk_score'], df['salary'])  
plt.title("AI Risk Score vs Salary")  
plt.xlabel("AI Risk Score")  
plt.ylabel("Salary")  
plt.show()
```

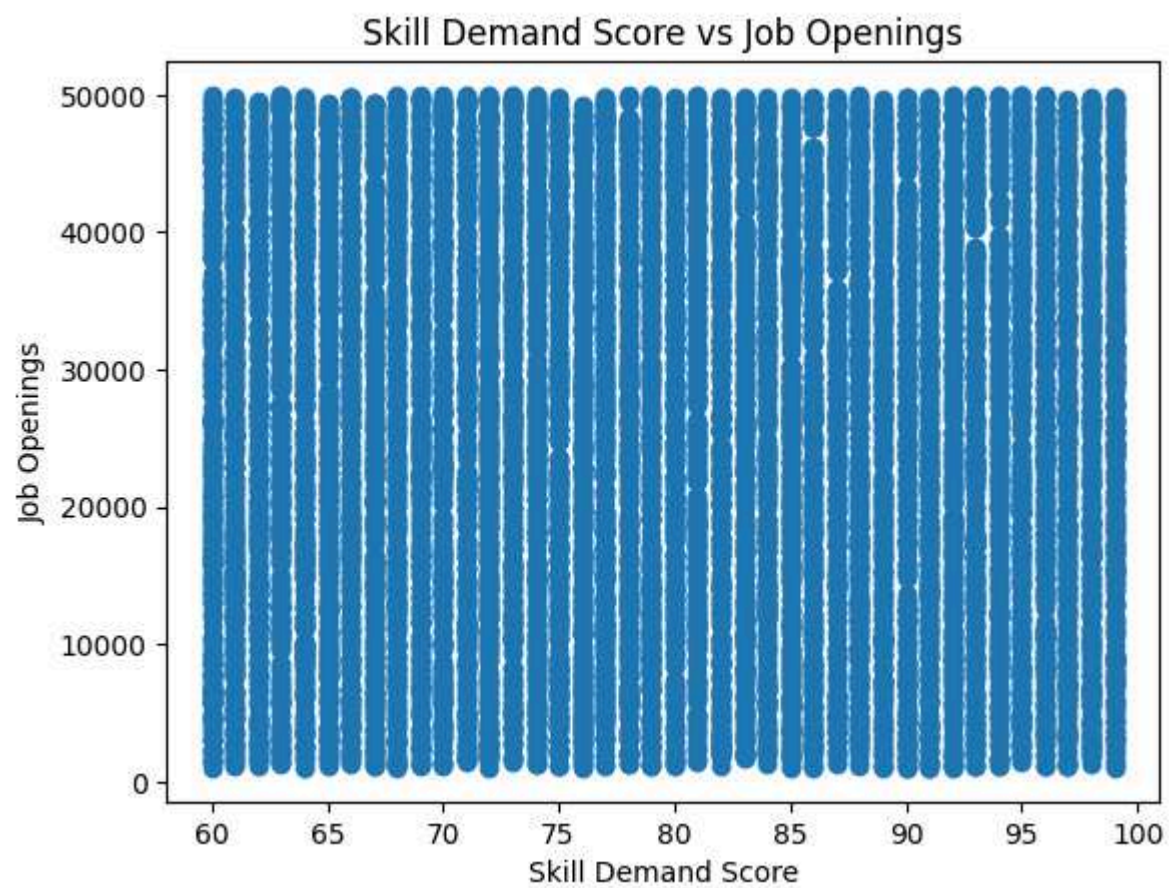
AI Risk Score vs Salary



```
In [28]: df['job_survival_class'].value_counts().plot(kind='bar')
plt.title("Job Survival Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```

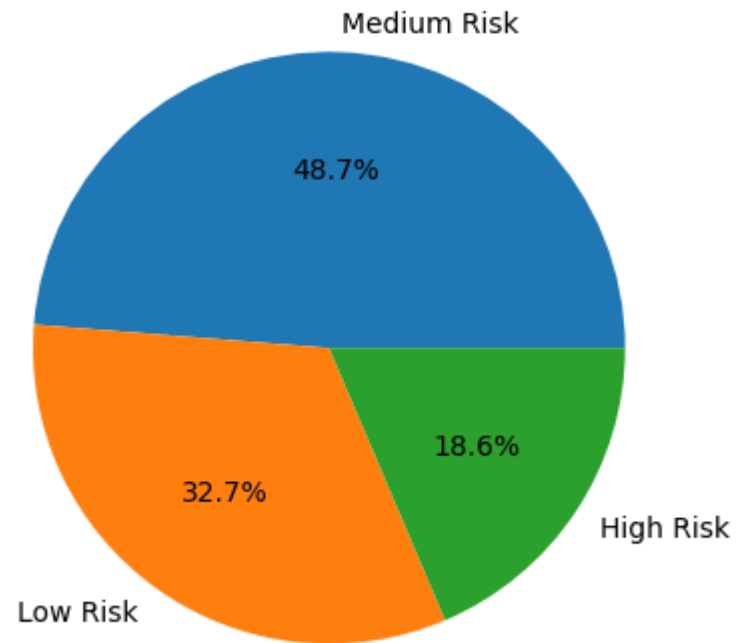


```
In [29]: plt.scatter(df['skill_demand_score'], df['job_openings'])  
plt.title("Skill Demand Score vs Job Openings")  
plt.xlabel("Skill Demand Score")  
plt.ylabel("Job Openings")  
plt.show()
```



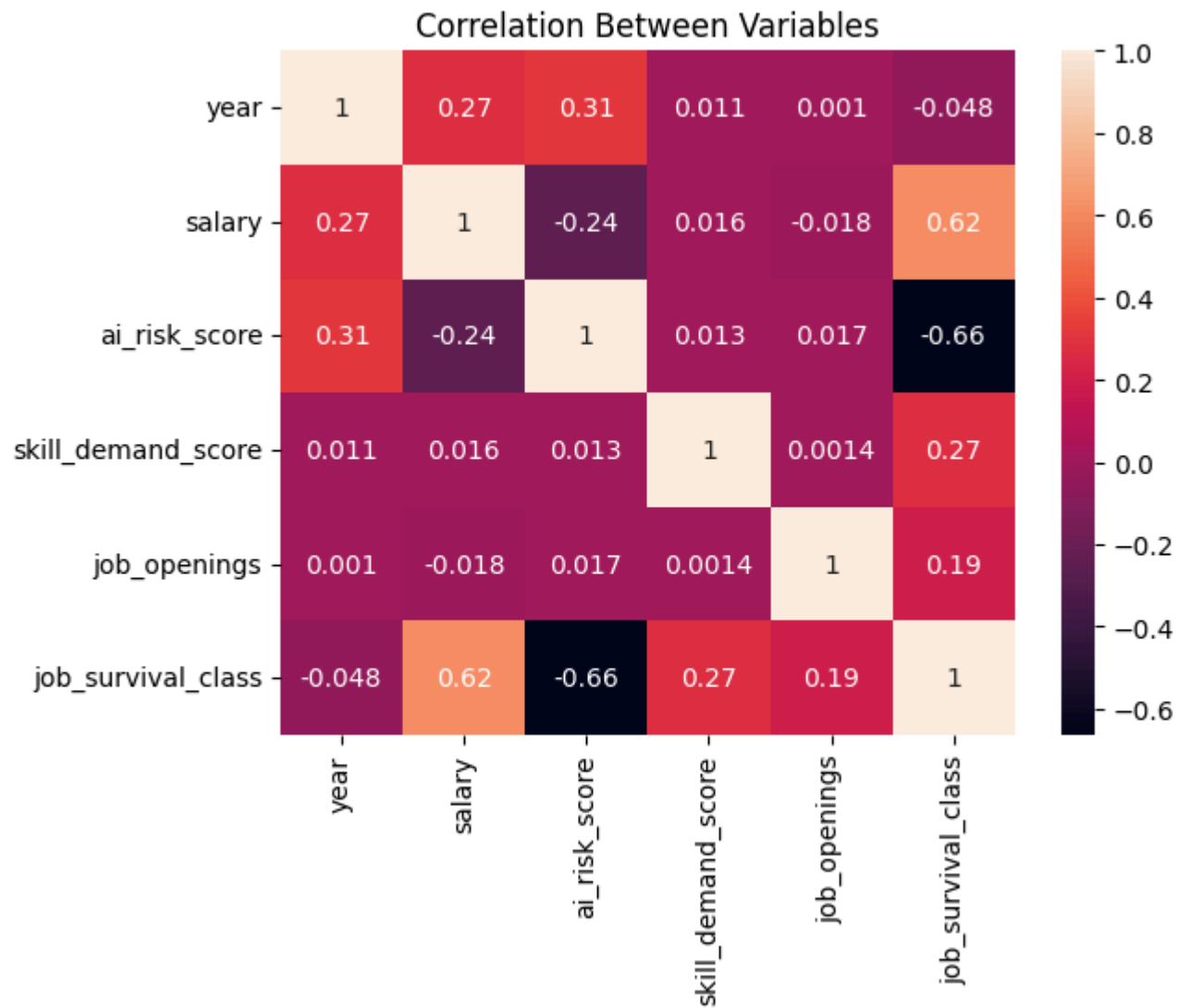
```
In [30]: df['ai_risk_category'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title("AI Risk Category Distribution")
plt.ylabel("")
plt.show()
```

AI Risk Category Distribution

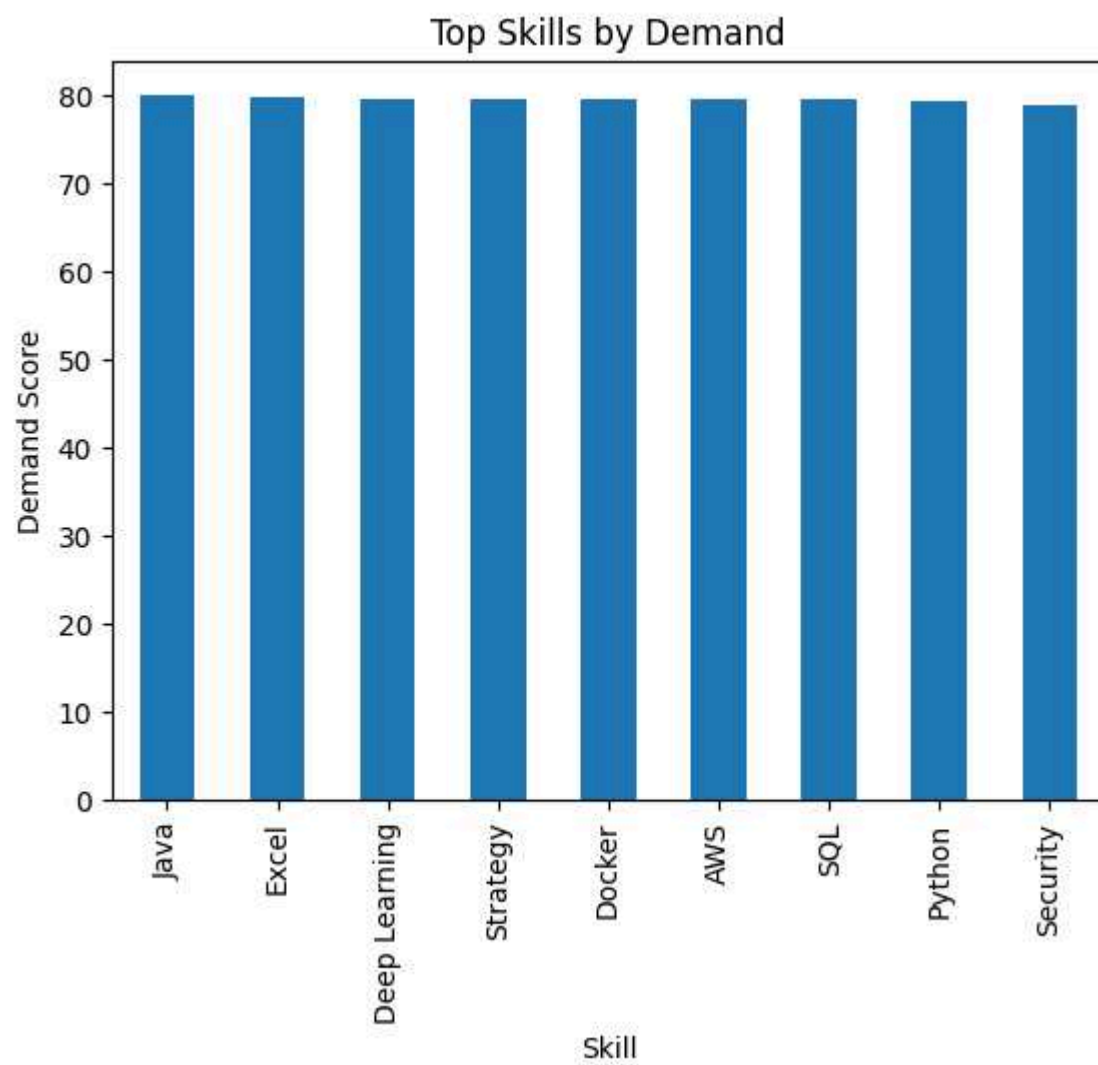


```
In [31]: import seaborn as sns

corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
plt.title("Correlation Between Variables")
plt.show()
```



```
In [32]: df.groupby('primary_skill')['skill_demand_score'].mean().sort_values(ascending=False).head(10).plot(kind='bar')
plt.title("Top Skills by Demand")
plt.xlabel("Skill")
plt.ylabel("Demand Score")
plt.show()
```



```
In [33]: from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X, y)

importances = pd.Series(model.feature_importances_, index=X.columns)
print(importances.sort_values(ascending=False))
```

```
-----  
NameError                                Traceback (most recent call last)
```

```
Cell In[33], line 4  
      1 from sklearn.ensemble import RandomForestClassifier  
      3 model = RandomForestClassifier()  
----> 4 model.fit(X, y)  
      6 importances = pd.Series(model.feature_importances_, index=X.columns)  
      7 print(importances.sort_values(ascending=False))
```

```
NameError: name 'X' is not defined
```

```
In [34]: import pandas as pd  
        from sklearn.preprocessing import LabelEncoder  
  
        df_encoded = df.copy()  
        le = LabelEncoder()  
  
        for col in df_encoded.select_dtypes(include='object'):  
            df_encoded[col] = le.fit_transform(df_encoded[col])
```

```
C:\Users\Maro\AppData\Local\Temp\ipykernel_8312\2691317883.py:7: Pandas4Warning: For backward compatibility, 'str' dtypes are included by select_dtypes when 'object' dtype is specified. This behavior is deprecated and will be removed in a future version. Explicitly pass 'str' to `include` to select them, or to `exclude` to remove them and silence this warning.  
See https://pandas.pydata.org/docs/user\_guide/migration-3-strings.html#string-migration-select-dtypes for details on how to write code that works with pandas 2 and 3.  
    for col in df_encoded.select_dtypes(include='object'):
```

```
In [35]: df_encoded = pd.get_dummies(df, drop_first=True)
```

```
In [36]: X = df_encoded.drop('job_survival_class', axis=1)  
        y = df_encoded['job_survival_class']
```

```
In [37]: from sklearn.ensemble import RandomForestClassifier  
  
        model = RandomForestClassifier()  
        model.fit(X, y)  
  
        import pandas as pd  
        importances = pd.Series(model.feature_importances_, index=X.columns)  
        print(importances.sort_values(ascending=False))
```

salary	0.218294
skill_demand_score	0.141811
job_openings	0.111042
ai_risk_score	0.106900
experience_level_Senior	0.049456
country_India	0.036068
year	0.035712
ai_risk_category_Low Risk	0.034959
job_title_Business Analyst	0.030705
salary_bucket_Medium	0.026457
primary_skill_Excel	0.022848
ai_risk_category_Medium Risk	0.020868
primary_skill_Deep Learning	0.020375
job_title_Data Analyst	0.017699
primary_skill_SQL	0.017603
experience_level_Mid	0.015030
country_USA	0.009974
job_title_Software Engineer	0.008851
salary_bucket_Low	0.008097
education_level_PhD	0.007913
primary_skill_Java	0.007719
education_level_Master	0.007683
country_Germany	0.005818
job_title_ML Engineer	0.005678
country_UK	0.005595
country_Canada	0.005510
primary_skill_Python	0.003706
job_title_Cloud Engineer	0.002976
job_title_Data Scientist	0.002889
job_title_Cybersecurity Analyst	0.002464
primary_skill_Security	0.002352
job_title_Product Manager	0.002130
primary_skill_Docker	0.001765
job_title_DevOps Engineer	0.001618
primary_skill_Strategy	0.001436

dtype: float64

```
In [38]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']
```

```

# 4) Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 5) Build the Decision Tree model
model = DecisionTreeClassifier(random_state=42, max_depth=5)
model.fit(X_train, y_train)

# 6) Make predictions
y_pred = model.predict(X_test)

# 7) Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# 8) Plot the tree
plt.figure(figsize=(20, 10))
plot_tree(
    model,
    feature_names=X.columns,
    class_names=[str(c) for c in sorted(y.unique())],
    filled=True,
    fontsize=8
)
plt.title("Decision Tree Classifier")
plt.show()

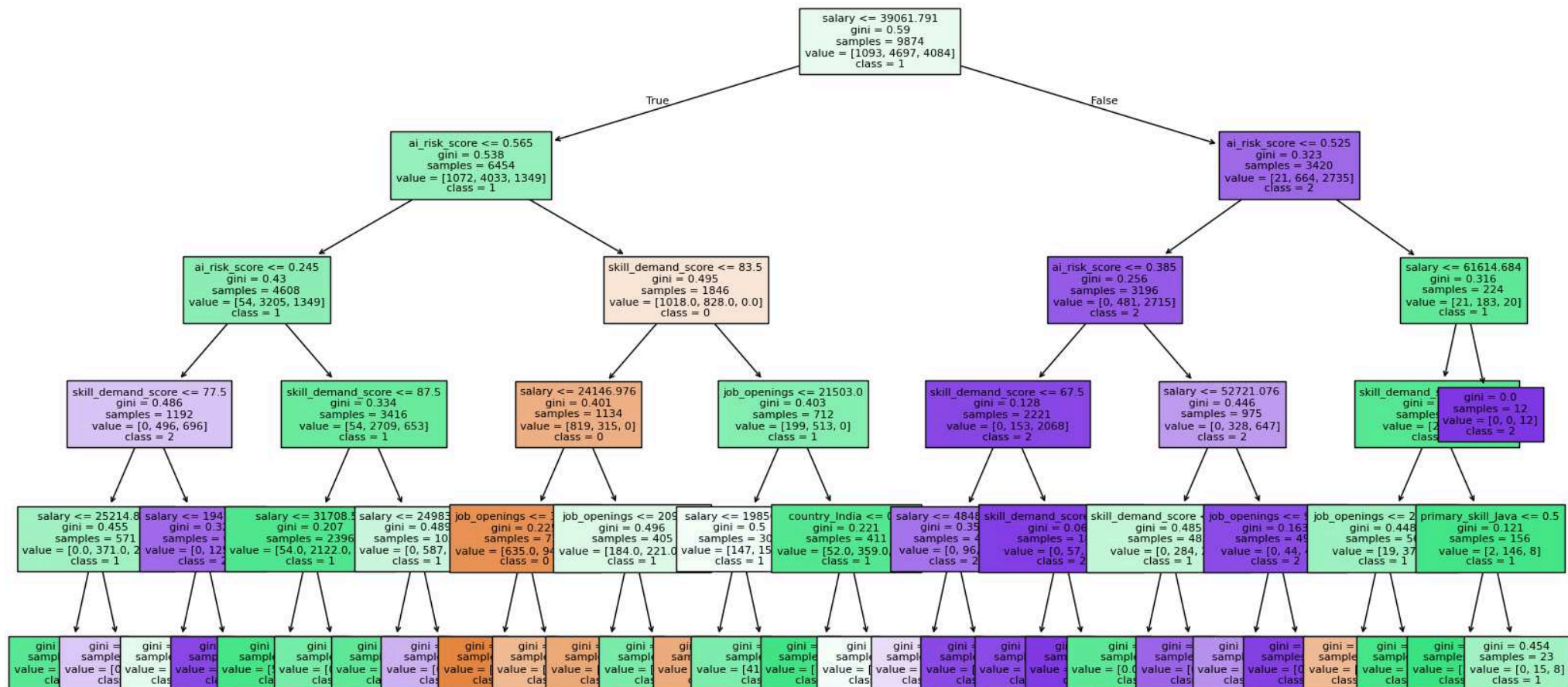
```

Accuracy: 0.8469015795868773

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.75	0.79	262
1	0.82	0.84	0.83	1109
2	0.88	0.88	0.88	1098
accuracy			0.85	2469
macro avg	0.84	0.82	0.83	2469
weighted avg	0.85	0.85	0.85	2469

Decision Tree Classifier



```
In [39]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_absolute_error, r2_score
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("Future of Jobs AI Dataset.csv")
df_encoded = pd.get_dummies(df, drop_first=True)
```

```
X = df_encoded.drop('salary', axis=1)
y = df_encoded['salary']
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```

model = DecisionTreeRegressor(random_state=42, max_depth=5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("MAE:", mean_absolute_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

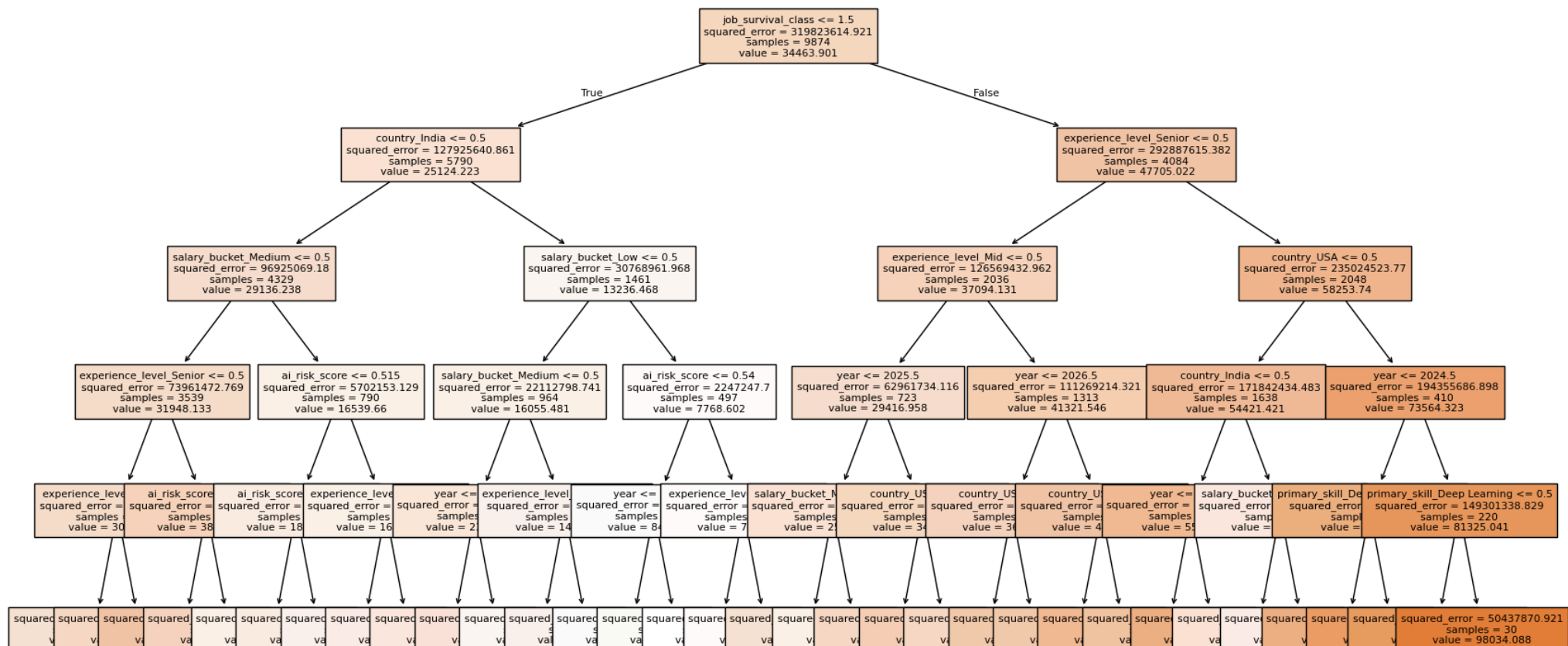
plt.figure(figsize=(20, 10))
plot_tree(model, feature_names=X.columns, filled=True, fontsize=8)
plt.title("Decision Tree Regressor")
plt.show()

```

MAE: 5196.102351178014

R2 Score: 0.8563911607430648

Decision Tree Regressor



```
In [40]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Encode categorical data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Build model
model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

# Train
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

Accuracy: 0.9639530174159578

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.91	0.93	262
1	0.95	0.97	0.96	1109
2	0.98	0.97	0.98	1098
accuracy			0.96	2469
macro avg	0.96	0.95	0.96	2469
weighted avg	0.96	0.96	0.96	2469

```
In [41]: import matplotlib.pyplot as plt
import pandas as pd

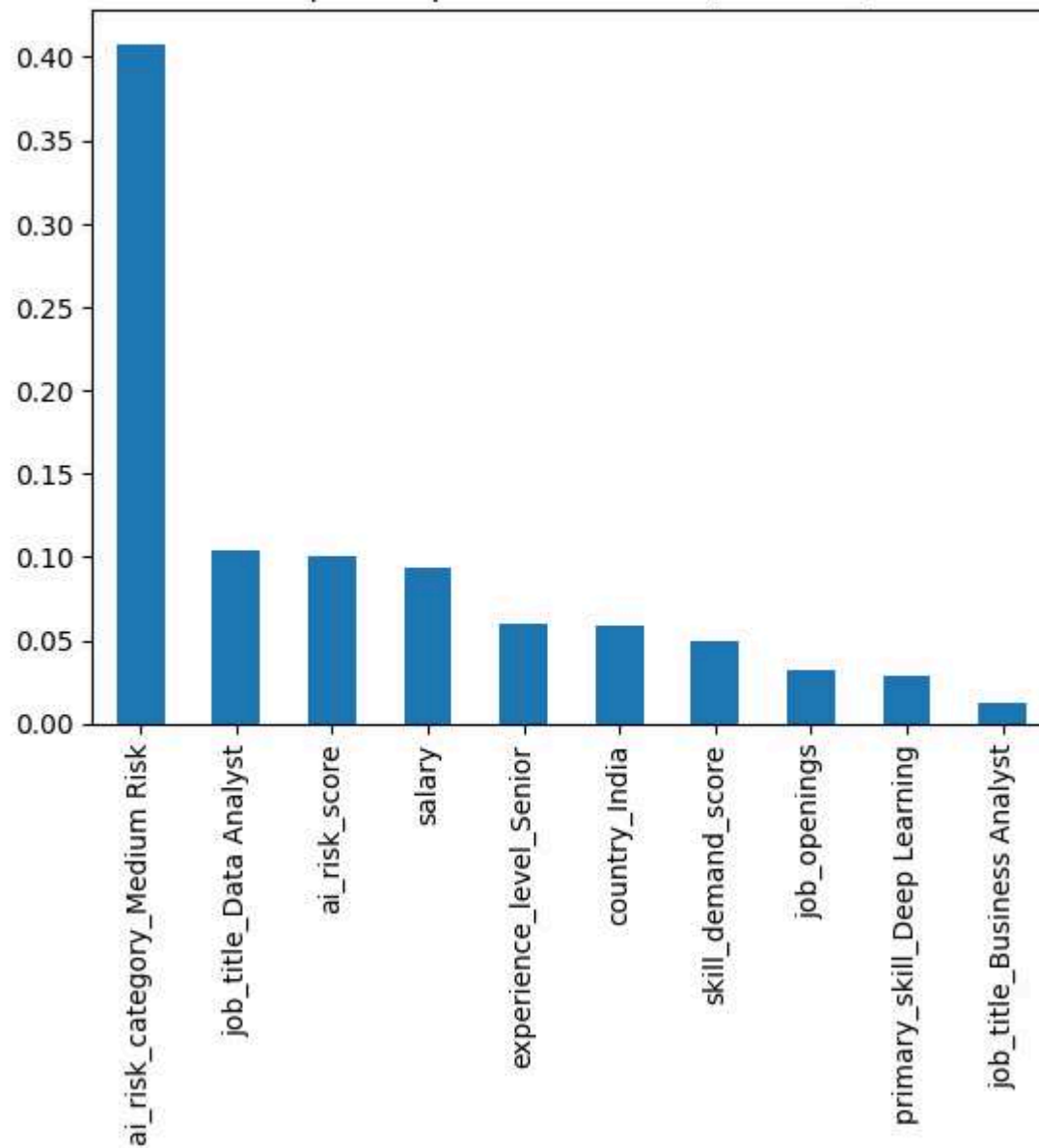
importance = pd.Series(model.feature_importances_, index=X.columns)
importance = importance.sort_values(ascending=False).head(10)

print(importance)

importance.plot(kind='bar')
plt.title("Top 10 Important Features (XGBoost)")
plt.show()
```

```
ai_risk_category_Medium Risk    0.407862
job_title_Data Analyst         0.104018
ai_risk_score                   0.100731
salary                         0.093438
experience_level_Senior        0.060164
country_India                  0.058497
skill_demand_score             0.049128
job_openings                   0.031589
primary_skill_Deep Learning    0.028336
job_title_Business Analyst     0.011890
dtype: float32
```

Top 10 Important Features (XGBoost)



```
In [42]: from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, r2_score

X = df_encoded.drop('salary', axis=1)
y = df_encoded['salary']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
```

```

)

model = XGBRegressor(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("MAE:", mean_absolute_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

```

MAE: 1924.391388641024

R2 Score: 0.9810595584134281

```

In [44]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# Define features and target (classification)
X = df_encoded.drop(['job_survival_class', 'salary_bucket', 'ai_risk_category'], axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Decision Tree
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)

dt_pred = dt_model.predict(X_test)
dt_acc = accuracy_score(y_test, dt_pred)

```

```
print("Decision Tree Accuracy:", dt_acc)

# XGBoost
xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

xgb_model.fit(X_train, y_train)

xgb_pred = xgb_model.predict(X_test)
xgb_acc = accuracy_score(y_test, xgb_pred)

print("XGBoost Accuracy:", xgb_acc)
```

KeyError Traceback (most recent call last)

Cell In[44], line 14

```
11 df_encoded = pd.get_dummies(df, drop_first=True)
13 # Define features and target (classification)
--> 14 X = df_encoded.drop([ , , ], axis=1)
15 y = df_encoded['job_survival_class']
17 # Split data
```

File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\frame.py:6288, in DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors)

```
6130 def drop(
6131     self,
6132     labels: IndexLabel | ListLike = None,
6133     errors: IgnoreRaise = "raise",
6134 ) -> DataFrame | None:
6141     """
6142     Drop specified labels from rows or columns.
6143
6144     (...) 6286         weight 1.0      0.8
6287     """
-> 6288     return super().drop(
6289         labels=labels,
6290         axis=axis,
6291         index=index,
6292         columns=columns,
6293         level=level,
6294         inplace=inplace,
6295         errors=errors,
6296     )
```

File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\generic.py:4644, in NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)

```
4642 for axis, labels in axes.items():
4643     if labels is not None:
-> 4644         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
4646 if inplace:
4647     self._update_inplace(obj)
```

File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\generic.py:4686, in NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)

```
4684     new_axis = axis.drop(labels, level=level, errors=errors)
4685     else:
-> 4686     new_axis = axis.drop(labels, errors=errors)
4687     indexer = axis.get_indexer(new_axis)
4689 # Case for non-unique axis
```

```
4690 else:
```

```
File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\indexes\base.py:7268, in Index.drop(self, labels, errors)
```

```
7266 if mask.any():
7267     if errors != "ignore":
-> 7268         raise KeyError(f"{labels[mask].tolist()} not found in axis")
7269     indexer = indexer[~mask]
7270 return self.delete(indexer)
```

```
KeyError: "['salary_bucket', 'ai_risk_category'] not found in axis"
```

```
In [45]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns BEFORE encoding
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical columns
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Decision Tree
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)
dt_acc = accuracy_score(y_test, dt_pred)

# XGBoost
xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
```

```

    random_state=42,
    eval_metric='mlogloss'
)
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)
xgb_acc = accuracy_score(y_test, xgb_pred)

print("Decision Tree Accuracy:", dt_acc)
print("XGBoost Accuracy:", xgb_acc)

print("\nDecision Tree Report:\n")
print(classification_report(y_test, dt_pred))

print("\nXGBoost Report:\n")
print(classification_report(y_test, xgb_pred))

```

Decision Tree Accuracy: 0.8469015795868773

XGBoost Accuracy: 0.9651680842446334

Decision Tree Report:

	precision	recall	f1-score	support
0	0.83	0.75	0.79	262
1	0.82	0.84	0.83	1109
2	0.88	0.88	0.88	1098
accuracy			0.85	2469
macro avg	0.84	0.82	0.83	2469
weighted avg	0.85	0.85	0.85	2469

XGBoost Report:

	precision	recall	f1-score	support
0	0.96	0.91	0.94	262
1	0.95	0.98	0.96	1109
2	0.98	0.97	0.98	1098
accuracy			0.97	2469
macro avg	0.96	0.95	0.96	2469
weighted avg	0.97	0.97	0.97	2469

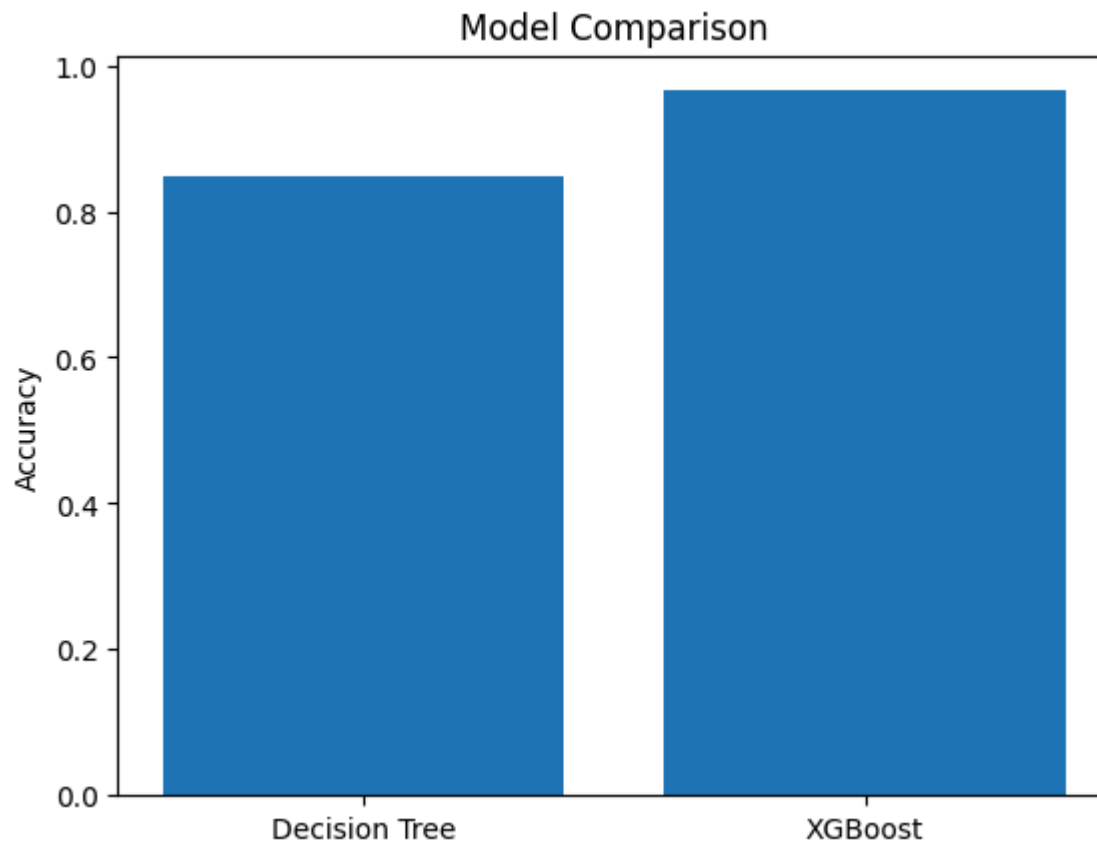
In [46]: `print(df_encoded.columns.tolist())`

```
['year', 'salary', 'ai_risk_score', 'skill_demand_score', 'job_openings', 'job_survival_class', 'job_title_Business Analyst', 'job_title_Cloud Engineer', 'job_title_Cybersecurity Analyst', 'job_title_Data Analyst', 'job_title_Data Scientist', 'job_title_DevOps Engineer', 'job_title_ML Engineer', 'job_title_Product Manager', 'job_title_Software Engineer', 'country_Canada', 'country_Germany', 'country_India', 'country_UK', 'country_USA', 'experience_level_Mid', 'experience_level_Senior', 'education_level_Master', 'education_level_PhD', 'primary_skill_Deep Learning', 'primary_skill_Docker', 'primary_skill_Excel', 'primary_skill_Java', 'primary_skill_Python', 'primary_skill_SQL', 'primary_skill_Security', 'primary_skill_Strategy']
```

```
In [47]: import matplotlib.pyplot as plt
```

```
models = ['Decision Tree', 'XGBoost']  
scores = [dt_acc, xgb_acc]
```

```
plt.bar(models, scores)  
plt.title("Model Comparison")  
plt.ylabel("Accuracy")  
plt.show()
```



```
In [48]: import pandas as pd  
from sklearn.model_selection import train_test_split  
from sklearn.feature_selection import RFE
```

```

from sklearn.tree import DecisionTreeClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create model
model = DecisionTreeClassifier(random_state=42)

# Apply RFE ( 5 features)
selector = RFE(model, n_features_to_select=5)
selector = selector.fit(X_train, y_train)

# Get selected features
selected_features = X.columns[selector.support_]

print("Selected Features:")
print(selected_features)

```

```

Selected Features:
Index(['year', 'salary', 'ai_risk_score', 'skill_demand_score',
      'job_openings'],
      dtype='str')

```

```

In [49]: ranking = pd.Series(selector.ranking_, index=X.columns)
print(ranking.sort_values())

```

year	1
salary	1
ai_risk_score	1
skill_demand_score	1
job_openings	1
country_India	2
experience_level_Senior	3
country_UK	4
country_Canada	5
experience_level_Mid	6
country_USA	7
country_Germany	8
education_level_PhD	9
job_title_Cloud Engineer	10
job_title_Data Scientist	11
job_title_Data Analyst	12
job_title_ML Engineer	13
primary_skill_Java	14
education_level_Master	15
primary_skill_Strategy	16
primary_skill_Security	17
job_title_Software Engineer	18
job_title_Cybersecurity Analyst	19
job_title_Product Manager	20
primary_skill_Docker	21
primary_skill_Python	22
primary_skill_Deep Learning	23
primary_skill_Excel	24
job_title_Business Analyst	25
job_title_DevOps Engineer	26
primary_skill_SQL	27
dtype:	int64

```
In [50]: X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

model.fit(X_train_selected, y_train)
y_pred = model.predict(X_test_selected)

from sklearn.metrics import accuracy_score
print("Accuracy after RFE:", accuracy_score(y_test, y_pred))
```

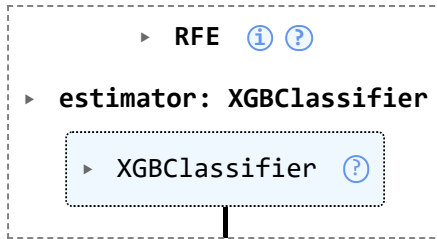
Accuracy after RFE: 0.9291211016605914

```
In [52]: from xgboost import XGBClassifier

model = XGBClassifier(random_state=42, eval_metric='mlogloss')
```

```
selector = RFE(model, n_features_to_select=5)
selector.fit(X_train, y_train)
```

Out[52]:



In [53]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import RFE
from sklearn.metrics import accuracy_score

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# ♦ BEFORE RFE
# -----
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred_before = model.predict(X_test)
acc_before = accuracy_score(y_test, y_pred_before)

print("Accuracy BEFORE RFE:", acc_before)
```

```

# -----
# ♦ APPLY RFE
# -----
selector = RFE(model, n_features_to_select=5)
selector.fit(X_train, y_train)

selected_features = X.columns[selector.support_]
print("\nSelected Features:", selected_features)

# -----
# ♦ AFTER RFE
# -----
X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

model.fit(X_train_selected, y_train)
y_pred_after = model.predict(X_test_selected)

acc_after = accuracy_score(y_test, y_pred_after)

print("\nAccuracy AFTER RFE:", acc_after)

```

Accuracy BEFORE RFE: 0.9218307006885379

Selected Features: Index(['year', 'salary', 'ai_risk_score', 'skill_demand_score',
'job_openings'],
dtype='str')

Accuracy AFTER RFE: 0.9291211016605914

In [54]:

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_selection import RFE
from xgboost import XGBClassifier

# 1. Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2. Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3. Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4. Define X and y
X = df_encoded.drop('job_survival_class', axis=1)

```

```
y = df_encoded['job_survival_class']

# 5. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6. Feature Selection (RFE)
base_model = XGBClassifier(random_state=42, eval_metric='mlogloss')
selector = RFE(base_model, n_features_to_select=5)
selector.fit(X_train, y_train)

selected_features = X.columns[selector.support_]
print("Selected Features:", selected_features)

# 7. Train final model using selected features
X_train_sel = X_train[selected_features]
X_test_sel = X_test[selected_features]

model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42,
    eval_metric='mlogloss'
)

model.fit(X_train_sel, y_train)

# 8. Predictions
y_pred = model.predict(X_test_sel)

# 9. Evaluation
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Selected Features: Index(['salary', 'ai_risk_score', 'skill_demand_score',  
      'job_title_Data Analyst', 'country_India'],  
      dtype='str')
```

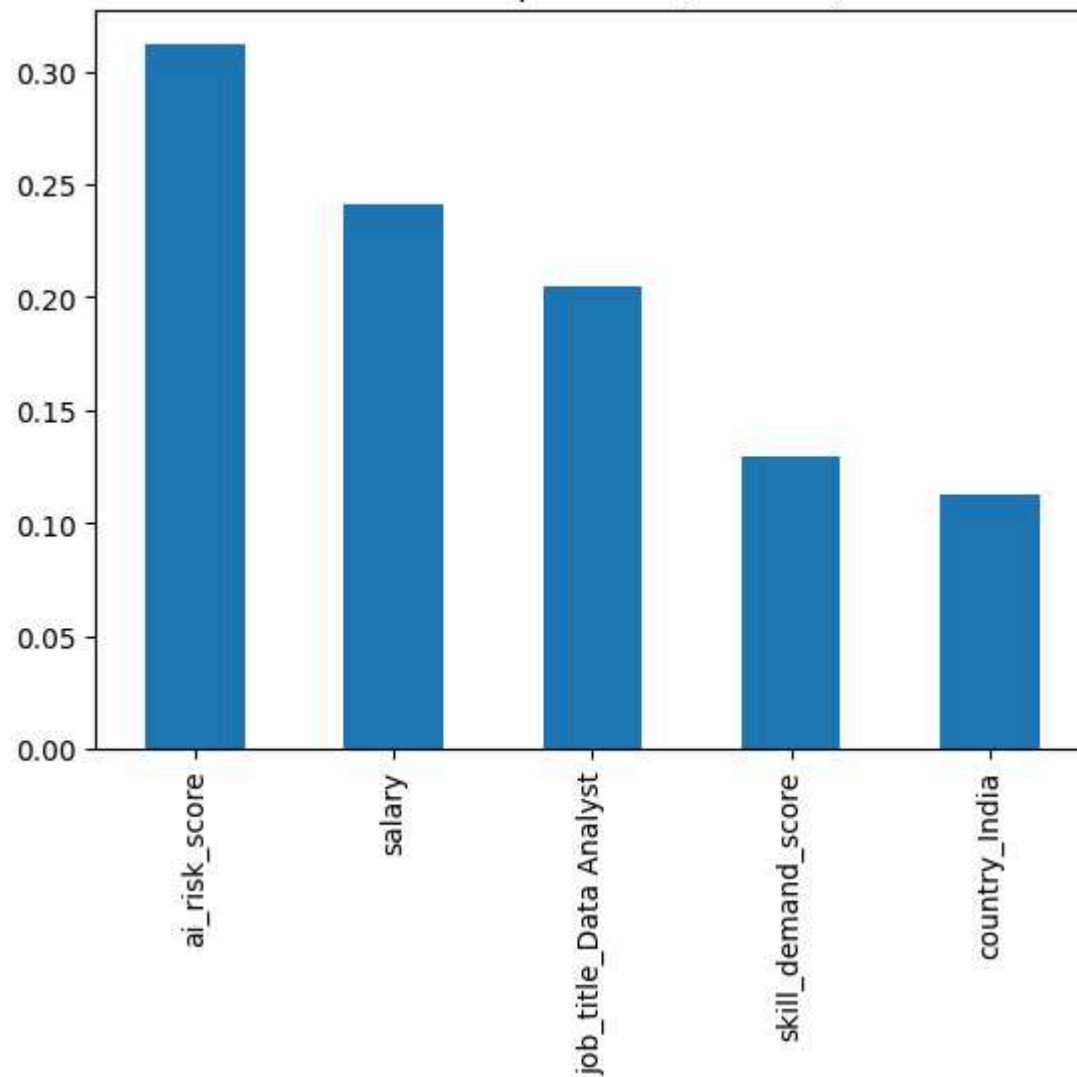
Accuracy: 0.8865937626569461

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.82	0.83	262
1	0.86	0.89	0.88	1109
2	0.92	0.90	0.91	1098
accuracy			0.89	2469
macro avg	0.88	0.87	0.87	2469
weighted avg	0.89	0.89	0.89	2469

```
In [55]: import matplotlib.pyplot as plt  
import pandas as pd  
  
importance = pd.Series(model.feature_importances_, index=selected_features)  
importance = importance.sort_values(ascending=False)  
  
importance.plot(kind='bar')  
plt.title("Feature Importance (XGBoost)")  
plt.show()
```

Feature Importance (XGBoost)



```
In [56]: import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Decision Tree metrics
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_precision = precision_score(y_test, dt_pred, average='weighted')
dt_recall = recall_score(y_test, dt_pred, average='weighted')
dt_f1 = f1_score(y_test, dt_pred, average='weighted')

# XGBoost metrics
xgb_accuracy = accuracy_score(y_test, xgb_pred)
```

```
xgb_precision = precision_score(y_test, xgb_pred, average='weighted')
xgb_recall = recall_score(y_test, xgb_pred, average='weighted')
xgb_f1 = f1_score(y_test, xgb_pred, average='weighted')
```

```
# Create table
```

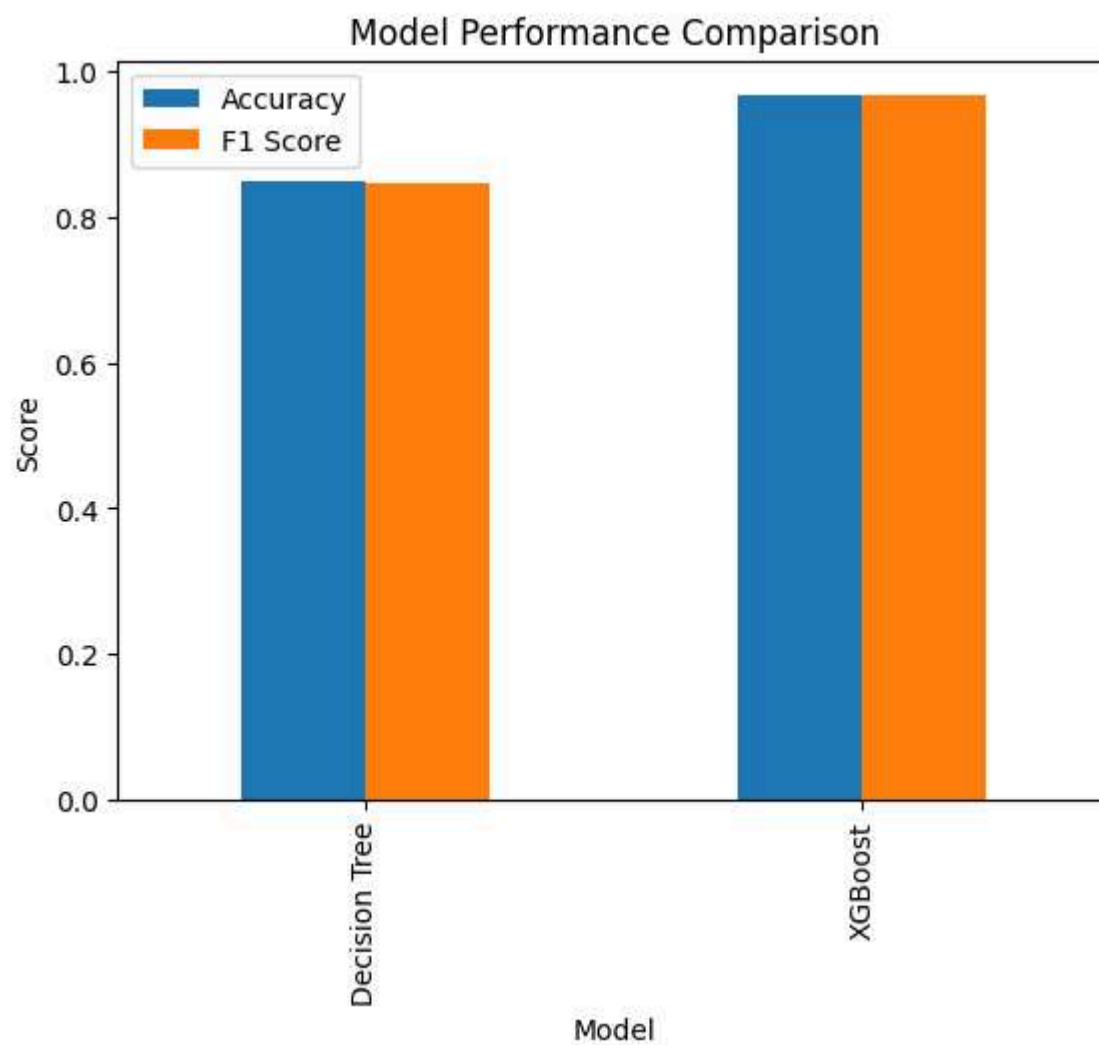
```
comparison = pd.DataFrame({
    "Model": ["Decision Tree", "XGBoost"],
    "Accuracy": [dt_accuracy, xgb_accuracy],
    "Precision": [dt_precision, xgb_precision],
    "Recall": [dt_recall, xgb_recall],
    "F1 Score": [dt_f1, xgb_f1]
})
```

```
print(comparison)
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.846902	0.846974	0.846902	0.846678
1	XGBoost	0.965168	0.965506	0.965168	0.965152

In [57]: `import matplotlib.pyplot as plt`

```
comparison.set_index('Model')[['Accuracy', 'F1 Score']].plot(kind='bar')
plt.title("Model Performance Comparison")
plt.ylabel("Score")
plt.show()
```



```
In [59]: import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical columns
df_encoded = pd.get_dummies(df, drop_first=True)
```

```

# Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Model
dt_model = DecisionTreeClassifier(random_state=42)

# Parameter grid
param_grid_dt = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# Grid Search
grid_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_dt.fit(X_train, y_train)

# Best model
best_dt = grid_dt.best_estimator_

# Prediction
dt_pred = best_dt.predict(X_test)

print("Best Decision Tree Parameters:", grid_dt.best_params_)
print("Best CV Score:", grid_dt.best_score_)
print("Test Accuracy:", accuracy_score(y_test, dt_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, dt_pred))

```

```
Best Decision Tree Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best CV Score: 0.923232606158542
Test Accuracy: 0.9400567031186715
```

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.89	0.91	262
1	0.93	0.94	0.93	1109
2	0.96	0.95	0.95	1098
accuracy			0.94	2469
macro avg	0.93	0.93	0.93	2469
weighted avg	0.94	0.94	0.94	2469

```
In [61]: import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical columns
df_encoded = pd.get_dummies(df, drop_first=True)

# Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Model
xgb_model = XGBClassifier(
    random_state=42,
    eval_metric='mlogloss'
)
```

```
# Parameter grid
param_grid_xgb = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Grid Search
grid_xgb = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid_xgb,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_xgb.fit(X_train, y_train)

# Best model
best_xgb = grid_xgb.best_estimator_

# Prediction
xgb_pred = best_xgb.predict(X_test)

print("Best XGBoost Parameters:", grid_xgb.best_params_)
print("Best CV Score:", grid_xgb.best_score_)
print("Test Accuracy:", accuracy_score(y_test, xgb_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, xgb_pred))
```

Best XGBoost Parameters: { 'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 150, 'subsample': 0.8}
Best CV Score: 0.9745790465930515
Test Accuracy: 0.9712434183880113

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.92	0.95	262
1	0.96	0.98	0.97	1109
2	0.99	0.98	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.96	2469
weighted avg	0.97	0.97	0.97	2469

In [74]: `from sklearn.linear_model import LogisticRegression`

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

D:\anaconda\envs\project2\Lib\site-packages\sklearn\linear_model_logistic.py:406: ConvergenceWarning: lbfgs failed to converge after 1000 iterations(s) (status=1):

STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

In [75]: `from sklearn.metrics import accuracy_score, classification_report`

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

Accuracy: 0.8991494532199271

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.82	0.83	262
1	0.88	0.90	0.89	1109
2	0.93	0.92	0.93	1098
accuracy			0.90	2469
macro avg	0.89	0.88	0.88	2469
weighted avg	0.90	0.90	0.90	2469

```
In [80]: from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
In [79]: model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train_scaled, y_train)
```

```
y_pred = model.predict(X_test_scaled)
```

```
In [81]: from sklearn.linear_model import LogisticRegression
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import accuracy_score, classification_report
```

```
# Scaling
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
# Model
```

```
model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train_scaled, y_train)
```

```
# Prediction
```

```
y_pred = model.predict(X_test_scaled)
```

```
# Evaluation
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

Accuracy: 0.9898744430943702

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.96	0.97	262
1	0.98	0.99	0.99	1109
2	0.99	0.99	0.99	1098
accuracy			0.99	2469
macro avg	0.99	0.98	0.99	2469
weighted avg	0.99	0.99	0.99	2469

```
In [82]: print(df['job_survival_class'].value_counts())
```

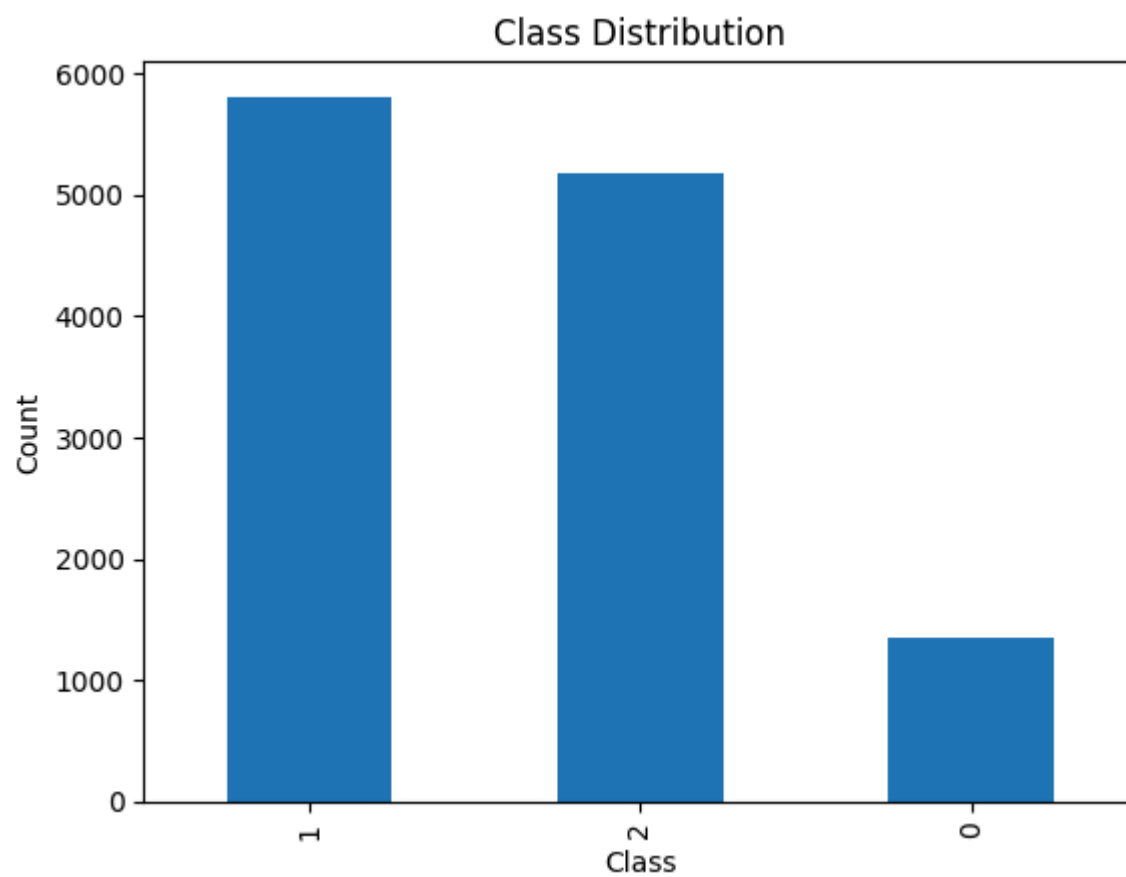
```
job_survival_class
1      5806
2      5182
0      1355
Name: count, dtype: int64
```

```
In [83]: print(df['job_survival_class'].value_counts(normalize=True) * 100)
```

```
job_survival_class
1      47.038807
2      41.983310
0      10.977882
Name: proportion, dtype: float64
```

```
In [84]: import matplotlib.pyplot as plt
```

```
df['job_survival_class'].value_counts().plot(kind='bar')
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```



```
In [86]: from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(class_weight='balanced')
```

```
In [87]: from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[87], line 1
----> 1 from imblearn.over_sampling import SMOTE
      3 smote = SMOTE(random_state=42)
      4 X_resampled, y_resampled = smote.fit_resample(X, y)

ModuleNotFoundError: No module named 'imblearn'
```

```
In [88]: import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Drop Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 🔥 Split FIRST
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 🔥 Apply SMOTE ONLY on training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[88], line 3
      1 import pandas as pd
      2 from sklearn.model_selection import train_test_split
----> 3 from imblearn.over_sampling import SMOTE
      4 # Load data
      5 df = pd.read_csv("Future of Jobs AI Dataset.csv")

ModuleNotFoundError: No module named 'imblearn'
```

```
In [89]: !pip install imbalanced-learn
```


[illegible]


```
# Encode data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 🔥 Split FIRST
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 🔥 Apply SMOTE ONLY on training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[90], line 3
      1 import pandas as pd
      2 from sklearn.model_selection import train_test_split
----> 3 from imblearn.over_sampling import SMOTE
      5 # Load data
      6 df = pd.read_csv("Future of Jobs AI Dataset.csv")

ModuleNotFoundError: No module named 'imblearn'
```

```
In [91]: import sys
         !{sys.executable} -m pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in D:\anaconda\envs\project2\Lib\site-packages (0.14.1)
Requirement already satisfied: numpy<3,>=1.25.2 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (2.4.3)
Requirement already satisfied: scipy<2,>=1.11.4 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (1.17.1)
Requirement already satisfied: scikit-learn<2,>=1.4.2 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (1.8.0)
Requirement already satisfied: sklearn-compat<0.2,>=0.1.5 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (0.1.5)
Requirement already satisfied: joblib<2,>=1.2.0 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (1.5.3)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (3.6.0)
```

```
In [92]: from imblearn.over_sampling import SMOTE
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[92], line 1
----> 1 from imblearn.over_sampling import SMOTE

ModuleNotFoundError: No module named 'imblearn'
```

```
In [93]: import sys
print(sys.executable)
```

D:\anaconda\envs\project2\python.exe

```
In [94]: import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Drop Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 🔥 Split FIRST
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 🔥 Apply SMOTE ONLY on training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[94], line 3
      1 import pandas as pd
      2 from sklearn.model_selection import train_test_split
----> 3 from imblearn.over_sampling import SMOTE
      4 # Load data
      5 df = pd.read_csv("Future of Jobs AI Dataset.csv")

ModuleNotFoundError: No module named 'imblearn'
```

```
In [100... from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [101...

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(class_weight='balanced', max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

D:\anaconda\envs\project2\Lib\site-packages\sklearn\linear_model_logistic.py:406: ConvergenceWarning: lbfgs failed to converge after 1000 iterations (status=1):

STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

n_iter_i = _check_optimize_result(

In [102...

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
```

Accuracy: 0.880113406237343

Classification Report:

	precision	recall	f1-score	support
0	0.72	0.92	0.81	262
1	0.89	0.83	0.86	1109
2	0.92	0.92	0.92	1098
accuracy			0.88	2469
macro avg	0.84	0.89	0.86	2469
weighted avg	0.89	0.88	0.88	2469

Confusion Matrix:

```
[[ 242   20    0]
 [  94  924   91]
 [    0   91 1007]]
```

In [103...

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# 1. Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2. Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3. Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4. Define features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 5. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6. Build XGBoost model (optimized parameters)
model = XGBClassifier()
```

```

n_estimators=200,      # more trees = better Learning
max_depth=6,           # deeper trees
learning_rate=0.1,
subsample=0.8,
colsample_bytree=0.8,
random_state=42,
eval_metric='mlogloss'
)

# 7. Train
model.fit(X_train, y_train)

# 8. Predict
y_pred = model.predict(X_test)

# 9. Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

```

Accuracy: 0.968408262454435

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.92	0.94	262
1	0.95	0.98	0.97	1109
2	0.98	0.97	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.96	2469
weighted avg	0.97	0.97	0.97	2469

In [104...

```

import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# 1) Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2) Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3) Encode categorical columns

```

```

df_encoded = pd.get_dummies(df, drop_first=True)

# 4) Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 5) Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6) Base model
xgb = XGBClassifier(
    objective='multi:softmax',
    num_class=3,
    random_state=42,
    eval_metric='mlogloss'
)

# 7) Parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [4, 6, 8],
    'learning_rate': [0.05, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# 8) Grid search
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)

# 9) Train grid search
grid_search.fit(X_train, y_train)

# 10) Best model
best_model = grid_search.best_estimator_

# 11) Predict
y_pred = best_model.predict(X_test)

```

```
# 12) Results
print("Best Parameters:")
print(grid_search.best_params_)

print("\nBest Cross-Validation Accuracy:")
print(grid_search.best_score_)

print("\nTest Accuracy:")
print(accuracy_score(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best Parameters:

```
{'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.8}
```

Best Cross-Validation Accuracy:

```
0.9745790978928603
```

Test Accuracy:

```
0.9736735520453625
```

Classification Report:

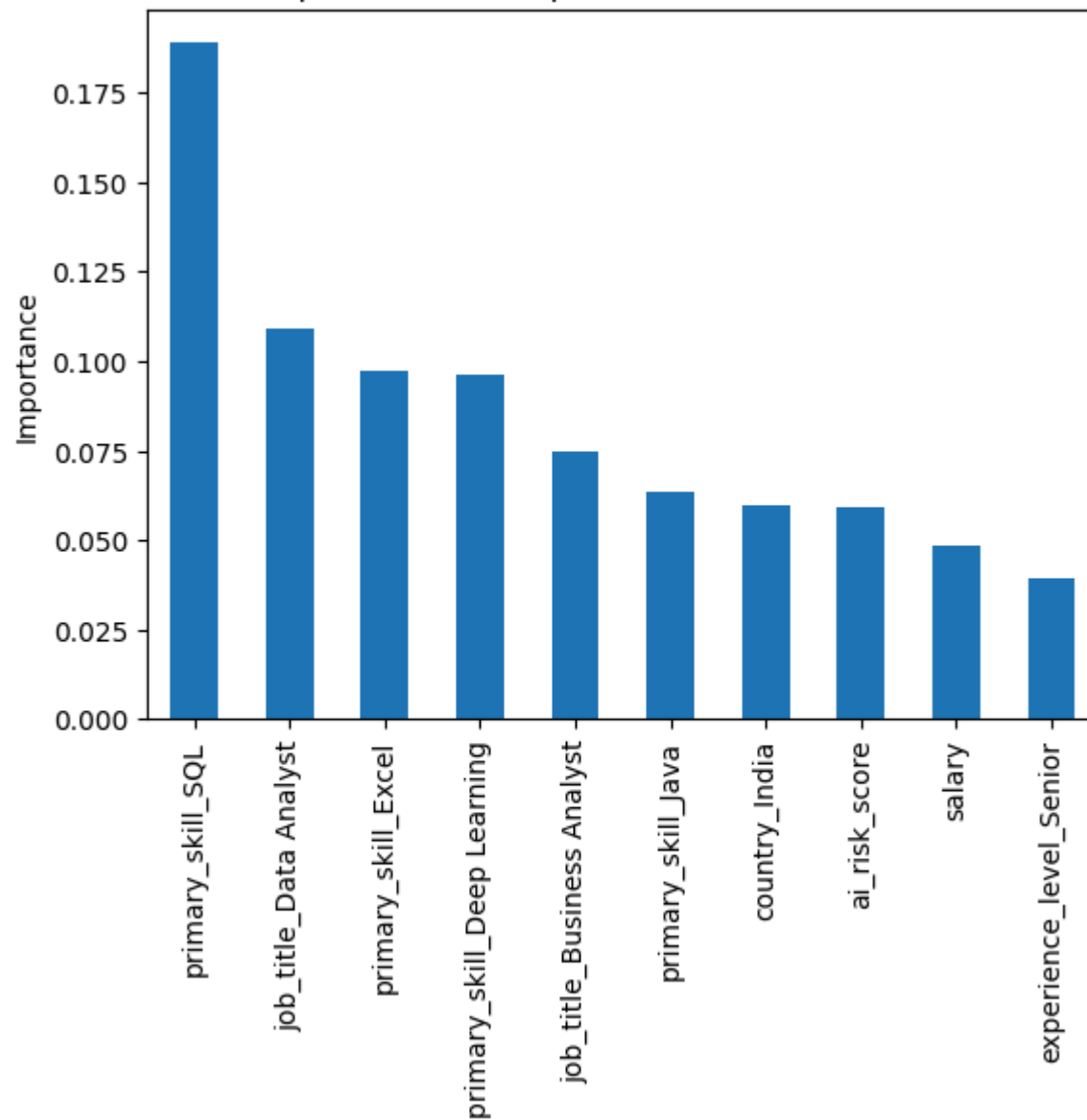
	precision	recall	f1-score	support
0	0.97	0.94	0.95	262
1	0.96	0.98	0.97	1109
2	0.99	0.98	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.97	2469
weighted avg	0.97	0.97	0.97	2469

In [108...

```
import matplotlib.pyplot as plt
import pandas as pd

importance = pd.Series(best_model.feature_importances_, index=X.columns)
importance.sort_values(ascending=False).head(10).plot(kind='bar')
plt.title("Top 10 Feature Importances - Tuned XGBoost")
plt.ylabel("Importance")
plt.show()
```

Top 10 Feature Importances - Tuned XGBoost



In [109... best_model = grid_search.bestbest_model = grid_search.best_estimator__estimator__

AttributeError

Traceback (most recent call last)

Cell In[109], line 1

```
----> 1 best_model = grid_search.bestbest_model = grid_search.best_estimator__estimator__
```

AttributeError: 'GridSearchCV' object has no attribute 'best_estimator__estimator__'

```
In [ ]: print(best_model)
```

```
In [110... plt.figure(figsize=(10,5))

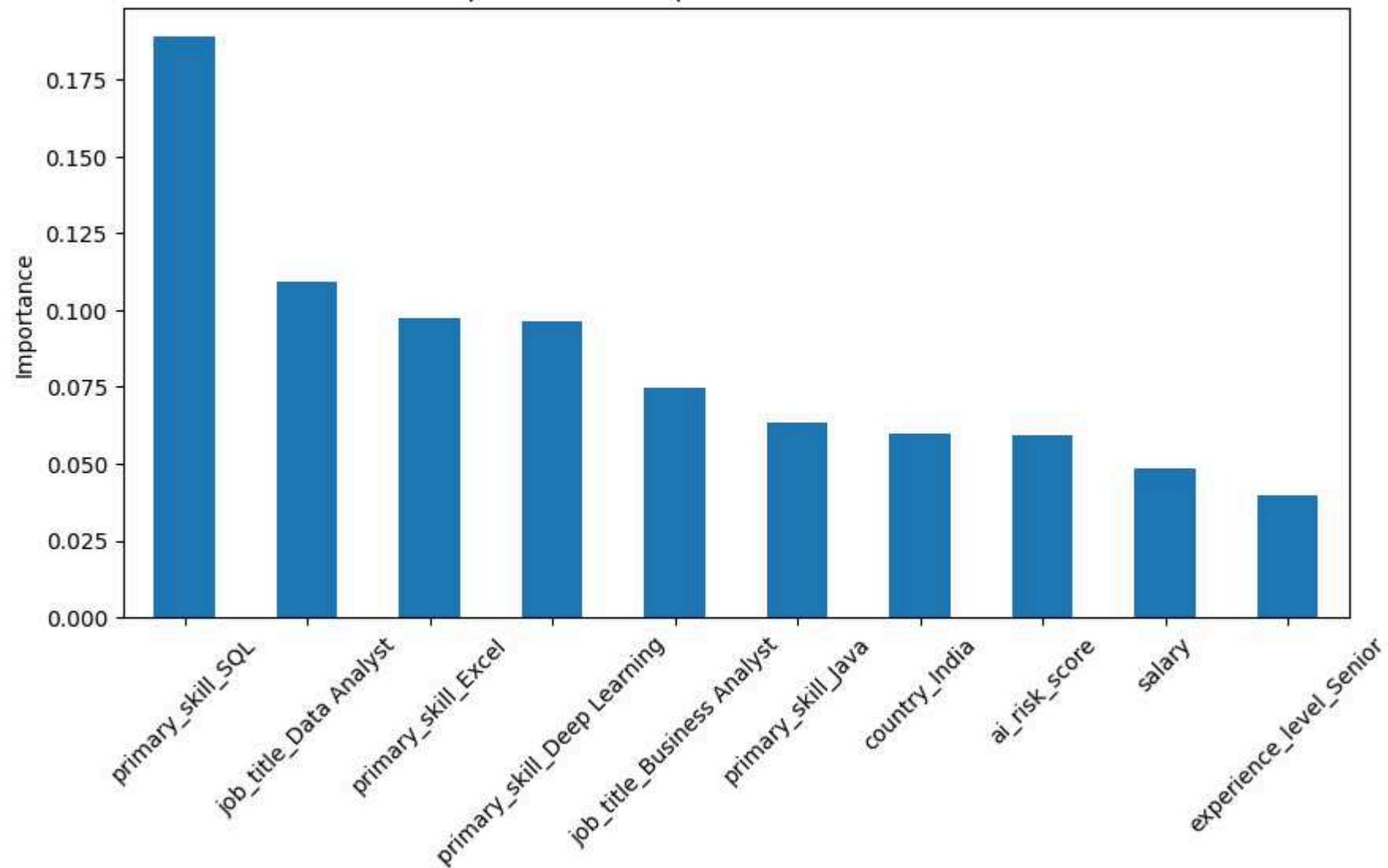
importance = pd.Series(best_model.feature_importances_, index=X.columns)
importance = importance.sort_values(ascending=False).head(10)

importance.plot(kind='bar')

plt.title("Top 10 Feature Importances - Tuned XGBoost")
plt.ylabel("Importance")
plt.xticks(rotation=45)

plt.show()
```

Top 10 Feature Importances - Tuned XGBoost



```
In [111]: print("Before tuning accuracy:", old_accuracy)
          print("After tuning accuracy:", new_accuracy)
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[111], line 1
----> 1 print("Before tuning accuracy:", old_accuracy)
      2 print("After tuning accuracy:", new_accuracy)

NameError: name 'old_accuracy' is not defined
```

```
In [112... from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

# Default model
model_before = XGBClassifier(random_state=42, eval_metric='mlogloss')
model_before.fit(X_train, y_train)

y_pred_before = model_before.predict(X_test)
old_accuracy = accuracy_score(y_test, y_pred_before)

print("Before tuning accuracy:", old_accuracy)
```

Before tuning accuracy: 0.9688132847306602

```
In [113... # Best model from GridSearch
best_model = grid_search.best_estimator_

y_pred_after = best_model.predict(X_test)
new_accuracy = accuracy_score(y_test, y_pred_after)

print("After tuning accuracy:", new_accuracy)
```

After tuning accuracy: 0.9736735520453625

```
In [114... print("Before tuning accuracy:", old_accuracy)
print("After tuning accuracy:", new_accuracy)
```

Before tuning accuracy: 0.9688132847306602

After tuning accuracy: 0.9736735520453625

```
In [115... import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from xgboost import XGBClassifier

# 1) Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2) Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3) Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4) Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']
```

```

# 5) Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6) Base XGBoost model
xgb = XGBClassifier(
    objective='multi:softmax',
    num_class=3,
    random_state=42,
    eval_metric='mlogloss'
)

# 7) Grid search parameters
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [4, 6, 8],
    'learning_rate': [0.05, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# 8) Grid search
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)

# 9) Train tuned model
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# 10) Predict
y_pred = best_model.predict(X_test)

# 11) Evaluation
print("Best Parameters:", grid_search.best_params_)
print("Best CV Accuracy:", grid_search.best_score_)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

```

```
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
```

Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.8}

Best CV Accuracy: 0.9745790978928603

Test Accuracy: 0.9736735520453625

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.94	0.95	262
1	0.96	0.98	0.97	1109
2	0.99	0.98	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.97	2469
weighted avg	0.97	0.97	0.97	2469

Confusion Matrix:

```
[[ 245   17    0]
 [   8 1085   16]
 [   0   24 1074]]
```

In [118...

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 1) Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2) Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3) Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4) Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 5) Split data
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6) Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 7) Logistic Regression with balanced class weights
lr_model = LogisticRegression(
    class_weight='balanced',
    max_iter=1000
)

# 8) Train model
lr_model.fit(X_train_scaled, y_train)

# 9) Predict
y_pred_lr = lr_model.predict(X_test_scaled)

# 10) Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred_lr))
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred_lr))
```

Accuracy: 0.9858242203321183

Classification Report:

	precision	recall	f1-score	support
0	0.93	1.00	0.96	262
1	0.99	0.97	0.98	1109
2	0.99	1.00	0.99	1098
accuracy			0.99	2469
macro avg	0.97	0.99	0.98	2469
weighted avg	0.99	0.99	0.99	2469

Confusion Matrix:

```
[[ 261    1    0]
 [  19 1080   10]
 [   0    5 1093]]
```

In [120...

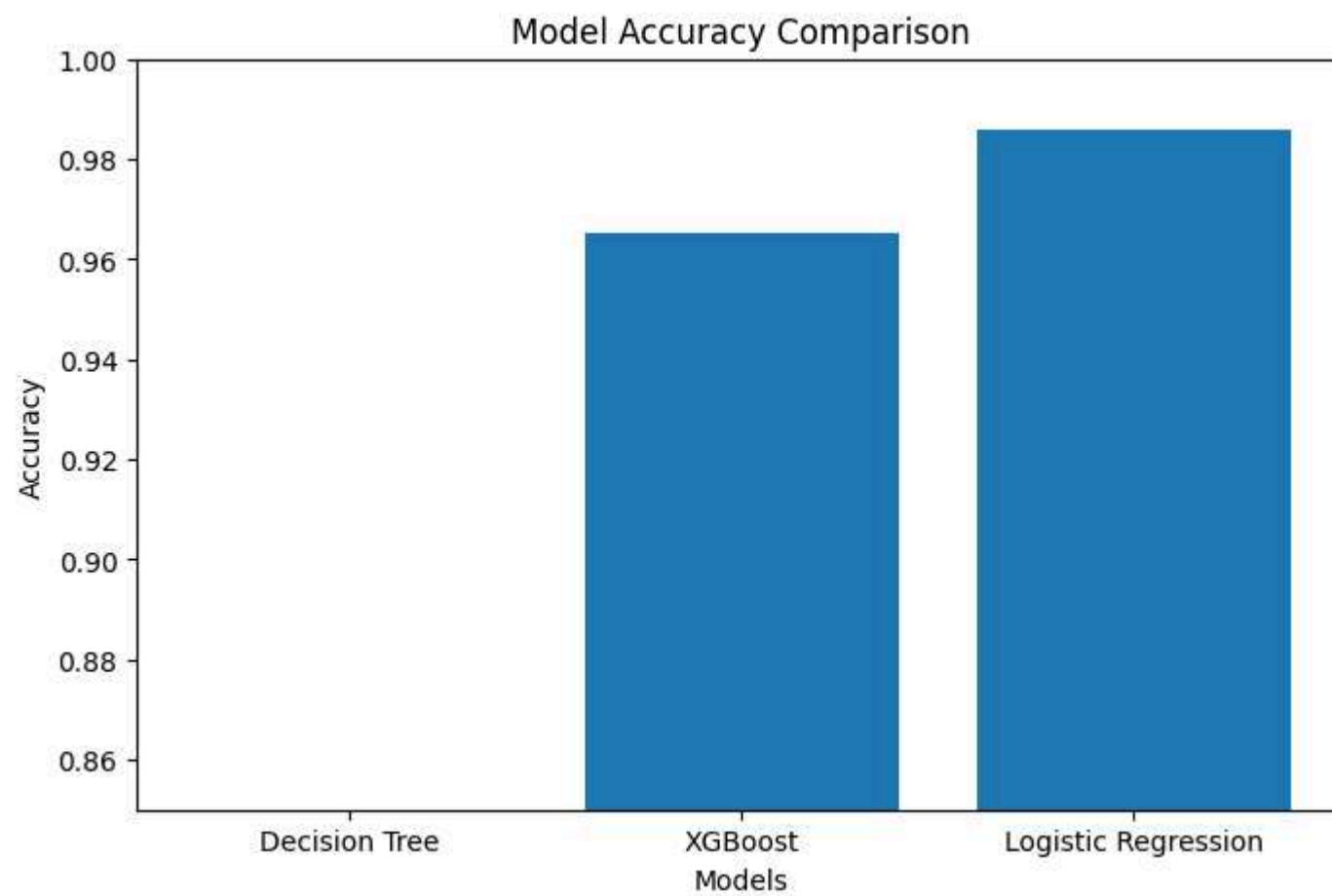
```
import matplotlib.pyplot as plt

models = ['Decision Tree', 'XGBoost', 'Logistic Regression']
accuracies = [dt_acc, xgb_acc, 0.986] # use your real values

plt.figure(figsize=(8,5))
plt.bar(models, accuracies)

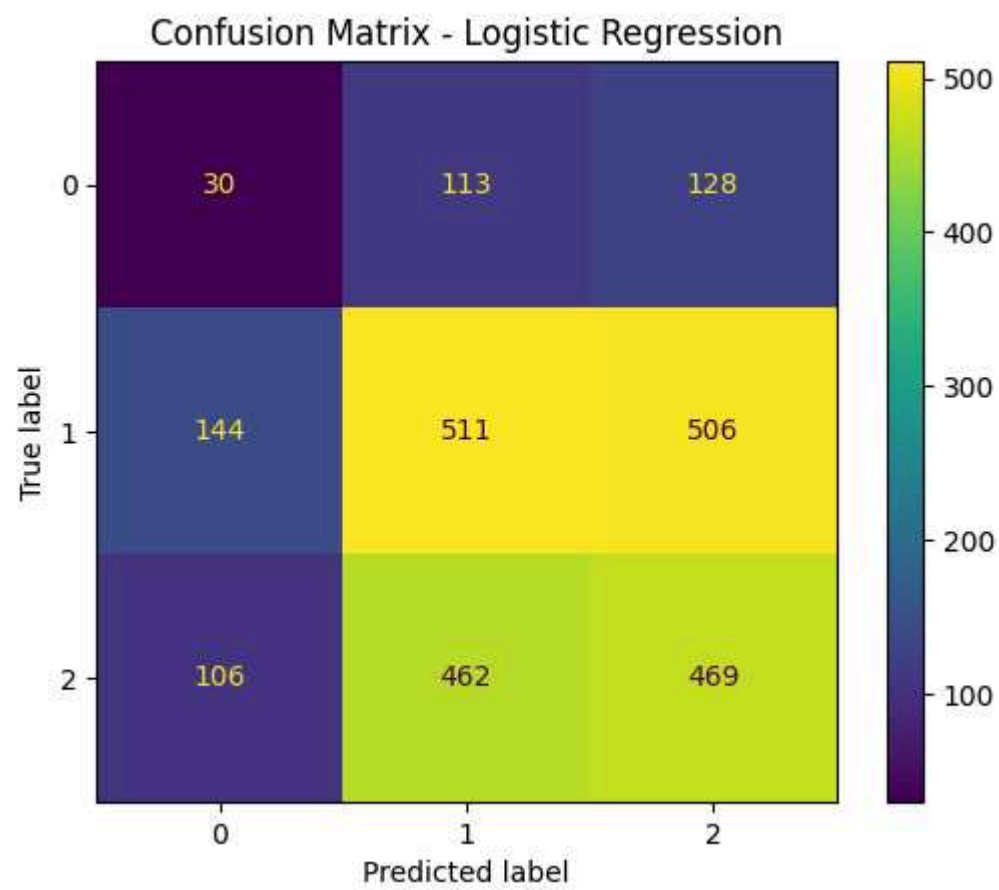
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.xlabel("Models")

plt.ylim(0.85, 1.0)
plt.show()
```



```
In [121... import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

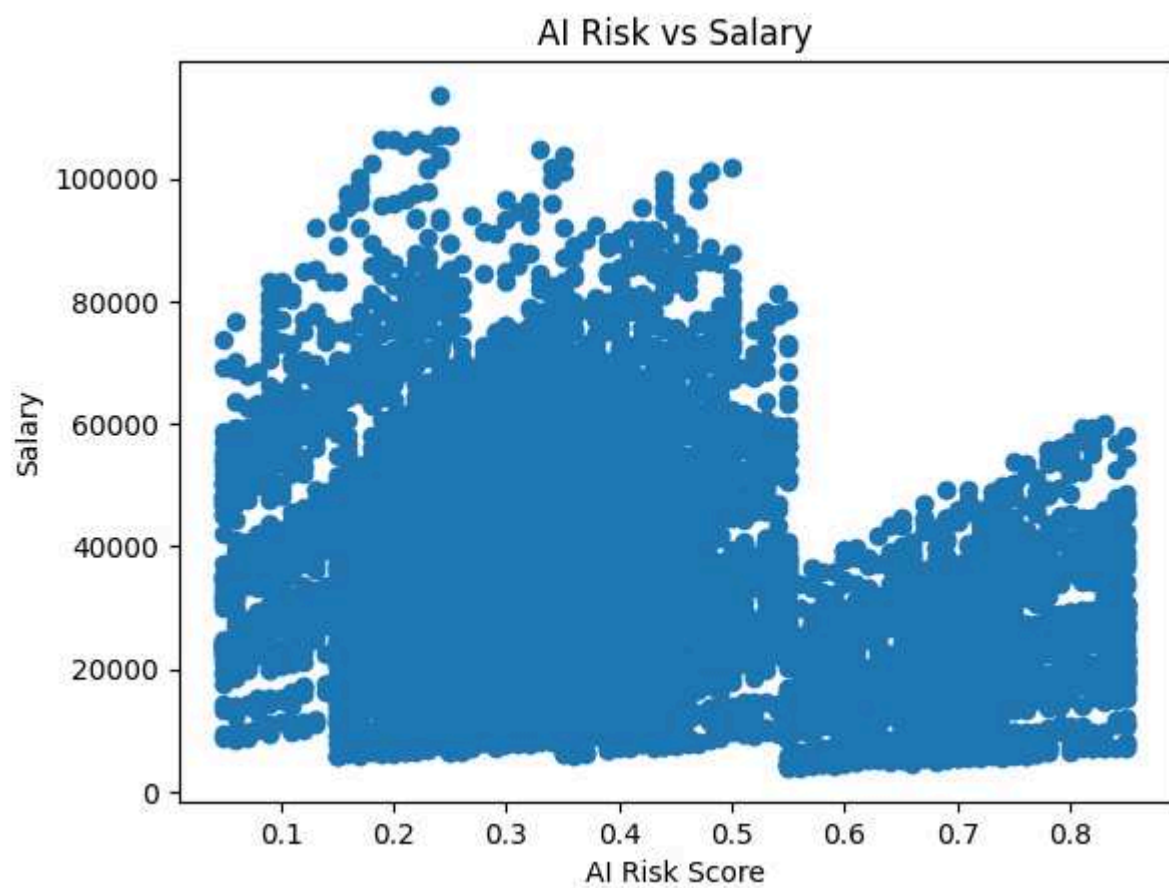
ConfusionMatrixDisplay.from_predictions(y_test, y_pred_lr)
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```



In [122...

```
import matplotlib.pyplot as plt

plt.scatter(df['ai_risk_score'], df['salary'])
plt.title("AI Risk vs Salary")
plt.xlabel("AI Risk Score")
plt.ylabel("Salary")
plt.show()
```

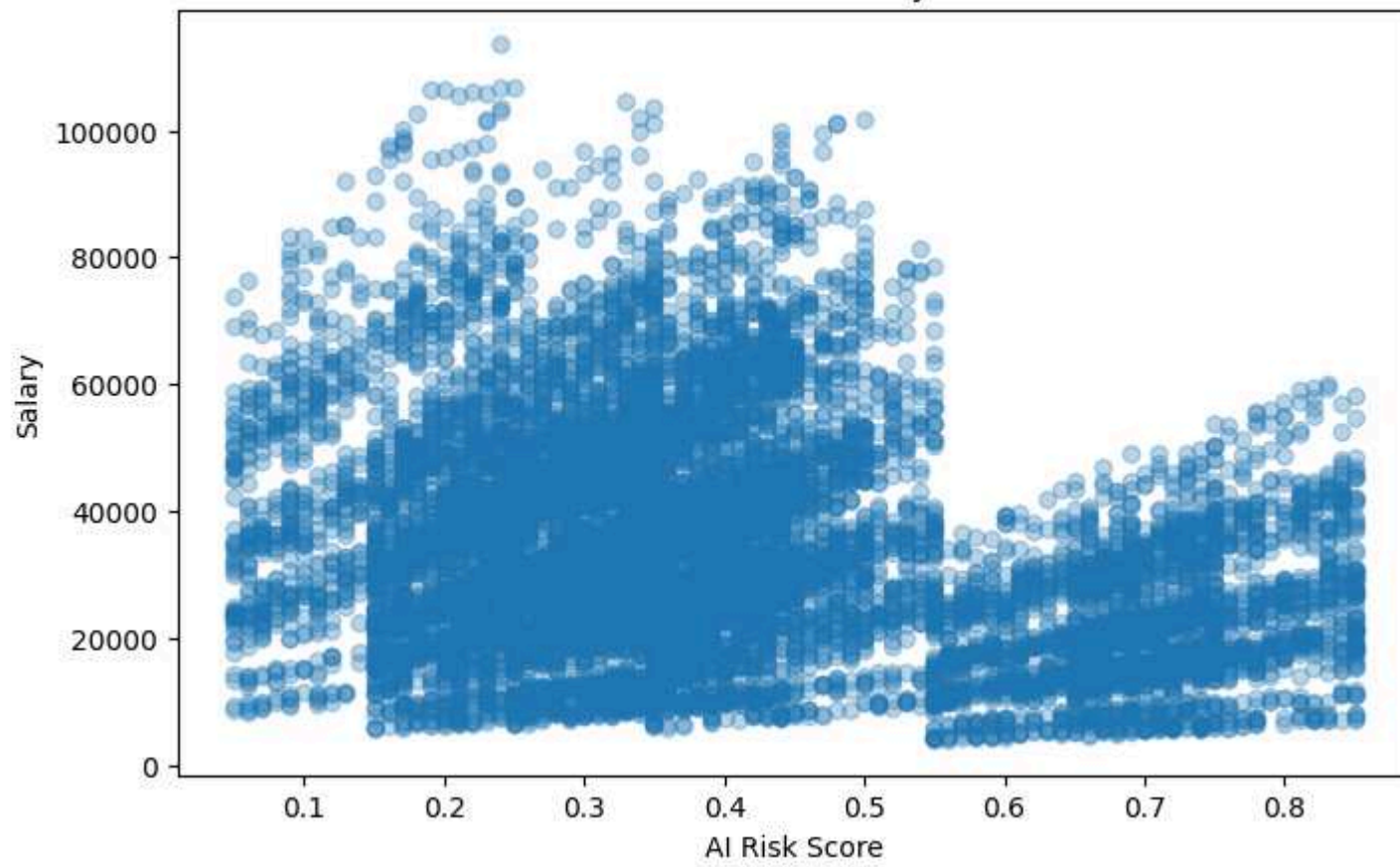


```
In [123... plt.figure(figsize=(8,5))
plt.scatter(df['ai_risk_score'], df['salary'], alpha=0.3)

plt.title("AI Risk vs Salary")
plt.xlabel("AI Risk Score")
plt.ylabel("Salary")

plt.show()
```

AI Risk vs Salary



In []:

```
In [2]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [3]: df=pd.read_csv("Future of Jobs AI Dataset.csv")
df.head()
```

Out[3]:

	job_title	country	experience_level	education_level	year	salary	ai_risk_score	primary_skill	skill_demand_score	job_openings	job_survival_class
0	Data Scientist	USA	Senior	Master	2022	61931.03	0.32	Python	88	39158	2
1	Software Engineer	India	Senior	Bachelor	2032	21330.84	0.52	Java	80	7265	1
2	Data Scientist	Canada	Mid	Bachelor	2015	24211.17	0.25	Python	70	22962	1
3	Data Scientist	India	Entry	PhD	2034	11794.86	0.44	Python	95	17023	1
4	Data Analyst	Canada	Entry	PhD	2035	17999.26	0.75	SQL	61	3433	0



```
In [5]: df.isnull().sum()
```

Out[5]:

job_title	0
country	0
experience_level	0
education_level	0
year	0
salary	0
ai_risk_score	0
primary_skill	0
skill_demand_score	0
job_openings	0
job_survival_class	0
salary_bucket	0
ai_risk_category	0
dtype:	int64

In [6]: `from sklearn.preprocessing import LabelEncoder` *#change categorical data to numerical data*

```
le = LabelEncoder()

categorical_cols = ['experience_level', 'education_level', 'salary_bucket', 'ai_risk_category']

for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

df.head()
```

Out[6]:

	job_title	country	experience_level	education_level	year	salary	ai_risk_score	primary_skill	skill_demand_score	job_openings	job_survival_class
--	-----------	---------	------------------	-----------------	------	--------	---------------	---------------	--------------------	--------------	--------------------

0	Data Scientist	USA	2	1	2022	61931.03	0.32	Python	88	39158	2
1	Software Engineer	India	2	0	2032	21330.84	0.52	Java	80	7265	1
2	Data Scientist	Canada	1	0	2015	24211.17	0.25	Python	70	22962	1
3	Data Scientist	India	0	2	2034	11794.86	0.44	Python	95	17023	1
4	Data Analyst	Canada	0	2	2035	17999.26	0.75	SQL	61	3433	0

In [7]: `from sklearn.preprocessing import LabelEncoder` *#explanation for numbers*

```
categorical_cols = ['experience_level', 'education_level', 'salary_bucket', 'ai_risk_category']

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

    print(f"\nColumn: {col}")
    print(dict(zip(le.classes_, le.transform(le.classes_))))
```

Column: experience_level
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

Column: education_level
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

Column: salary_bucket
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

Column: ai_risk_category
{np.int64(0): np.int64(0), np.int64(1): np.int64(1), np.int64(2): np.int64(2)}

```
In [8]: df.describe()
```

Out[8]:

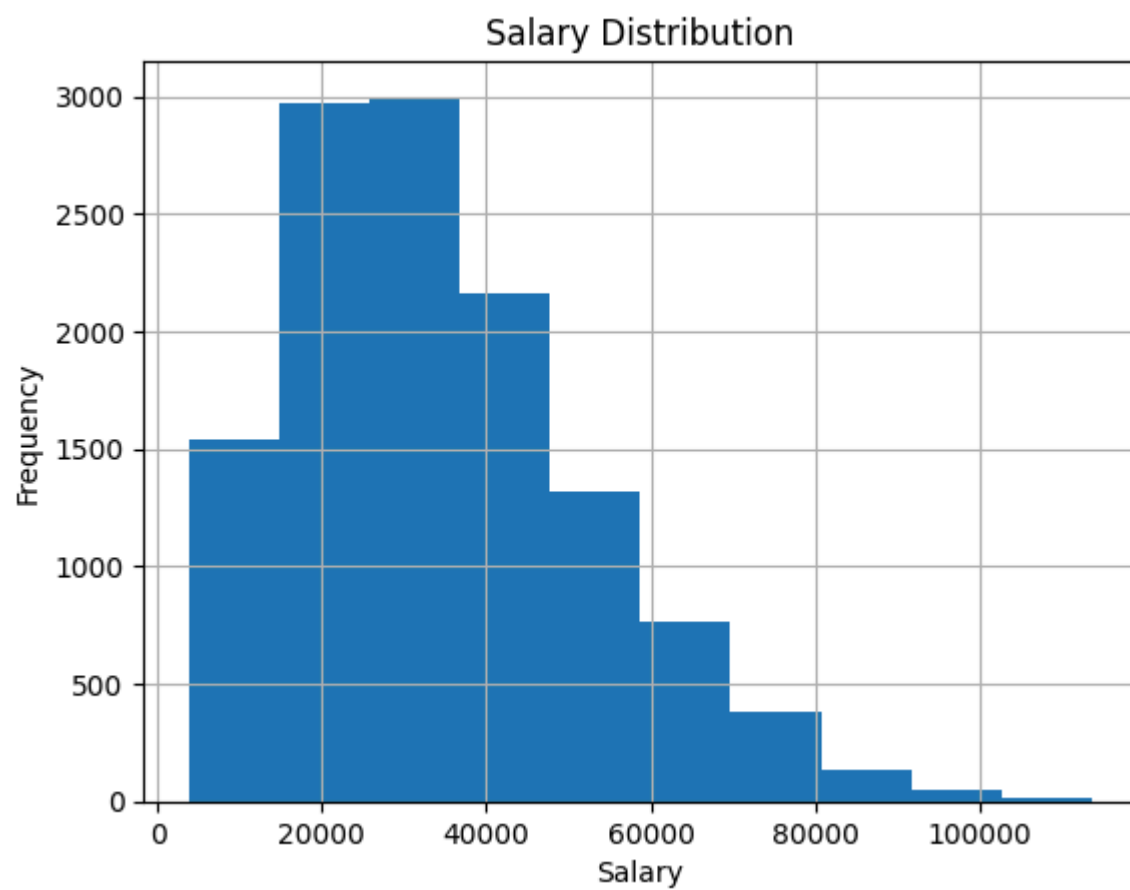
	experience_level	education_level	year	salary	ai_risk_score	skill_demand_score	job_openings	job_survival_class	salary_bucket
count	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000	12343.000000
mean	1.009722	1.000891	2025.054525	34553.729166	0.396008	79.453374	25223.652029	1.310054	0.393097
std	0.818520	0.814957	6.058714	18024.702961	0.187665	11.483535	14163.309081	0.658417	0.761169
min	0.000000	0.000000	2015.000000	3875.170000	0.050000	60.000000	1002.000000	0.000000	0.000000
25%	0.000000	0.000000	2020.000000	21331.405000	0.260000	70.000000	12993.000000	1.000000	0.000000
50%	1.000000	1.000000	2025.000000	31573.380000	0.360000	79.000000	24896.000000	1.000000	0.000000
75%	2.000000	2.000000	2030.000000	45307.845000	0.490000	89.000000	37440.500000	2.000000	0.000000
max	2.000000	2.000000	2035.000000	113589.260000	0.850000	99.000000	49998.000000	2.000000	2.000000

```
In [9]: df.info()
```

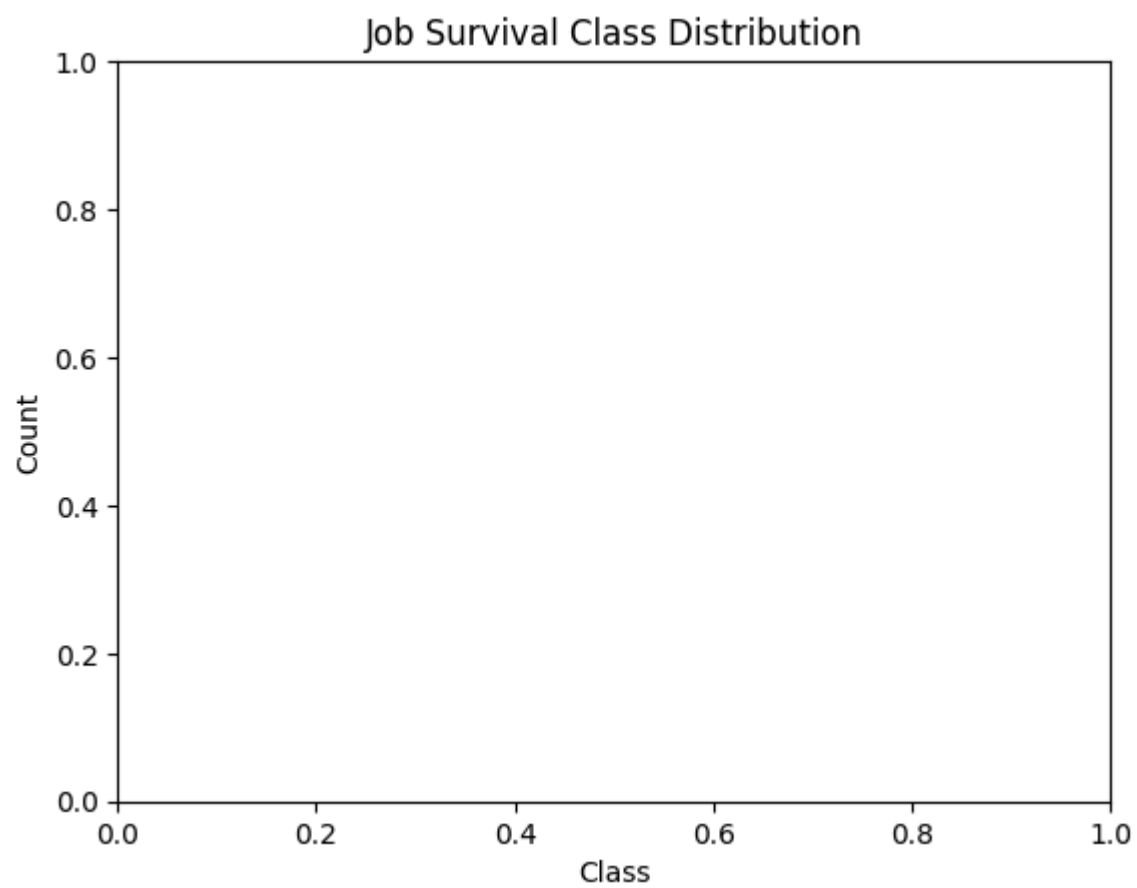
```
<class 'pandas.DataFrame'>
RangeIndex: 12343 entries, 0 to 12342
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   job_title             12343 non-null  str
 1   country               12343 non-null  str
 2   experience_level      12343 non-null  int64
 3   education_level       12343 non-null  int64
 4   year                 12343 non-null  int64
 5   salary               12343 non-null  float64
 6   ai_risk_score         12343 non-null  float64
 7   primary_skill         12343 non-null  str
 8   skill_demand_score    12343 non-null  int64
 9   job_openings          12343 non-null  int64
10   job_survival_class    12343 non-null  int64
11   salary_bucket         12343 non-null  int64
12   ai_risk_category      12343 non-null  int64
dtypes: float64(2), int64(8), str(3)
memory usage: 1.2 MB
```

```
In [10]: import matplotlib.pyplot as plt
```

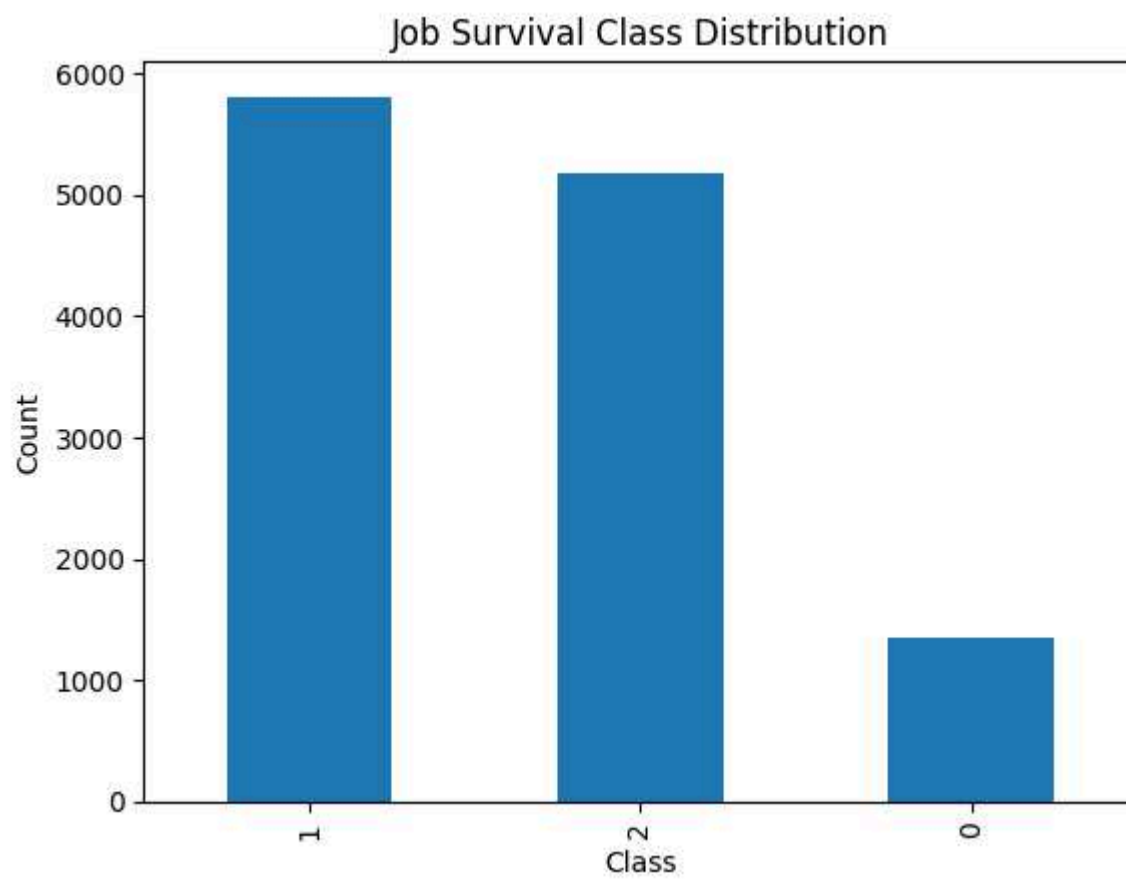
```
df['salary'].hist()
plt.title("Salary Distribution")
plt.xlabel("Salary")
plt.ylabel("Frequency")
plt.show()
```



```
In [11]: df['job_survival_class'].value_counts().plot
plt.title("Job Survival Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```

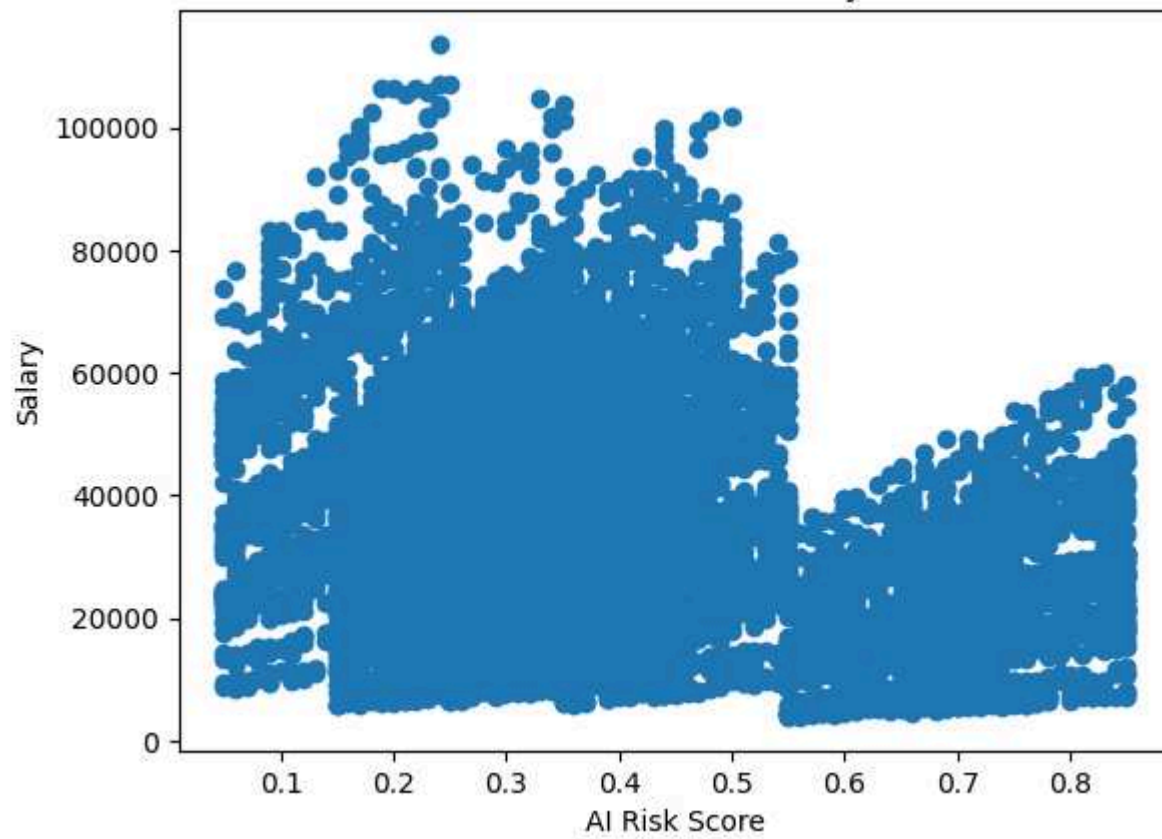


```
In [12]: df['job_survival_class'].value_counts().plot(kind='bar')
plt.title("Job Survival Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```

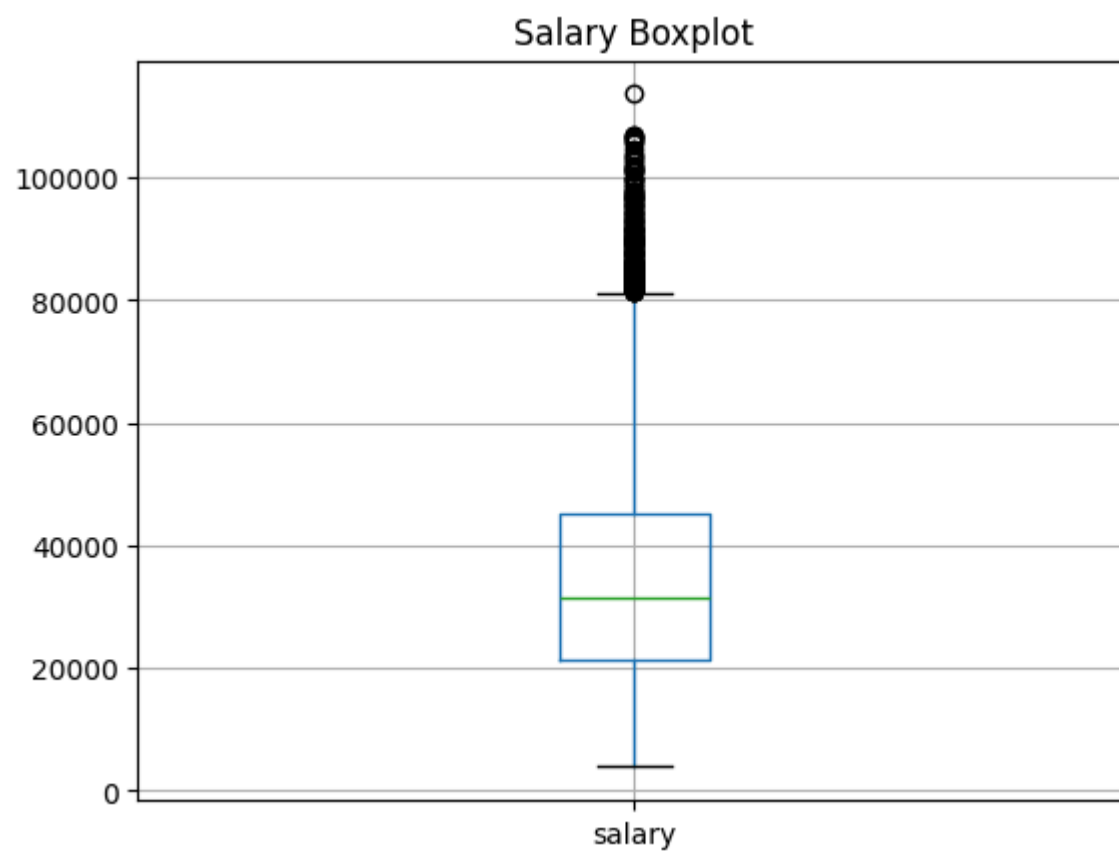


```
In [13]: plt.scatter(df['ai_risk_score'], df['salary'])
plt.title("AI Risk Score vs Salary")
plt.xlabel("AI Risk Score")
plt.ylabel("Salary")
plt.show()
```

AI Risk Score vs Salary

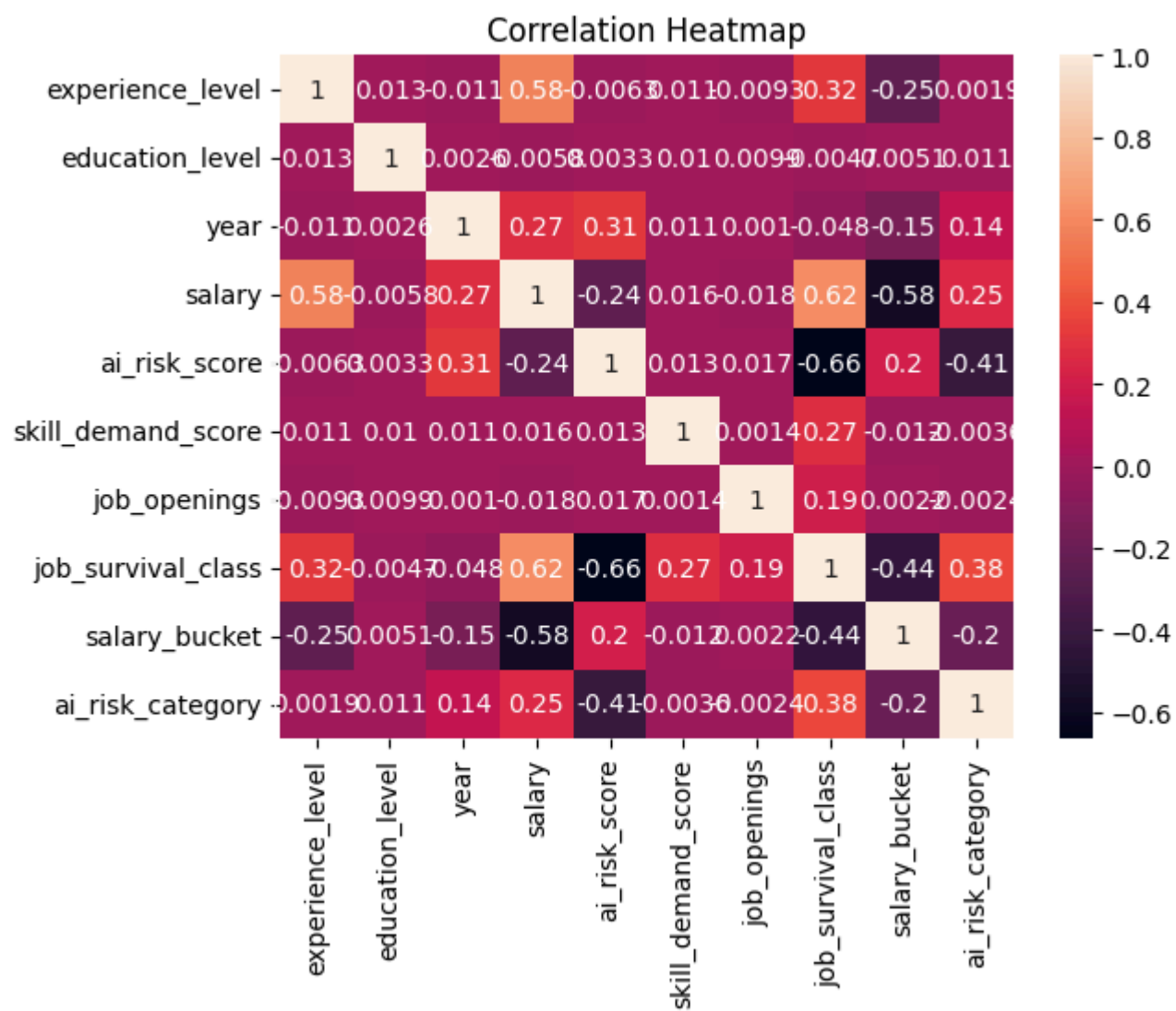


```
In [14]: df.boxplot(column='salary')
plt.title("Salary Boxplot")
plt.show()
```



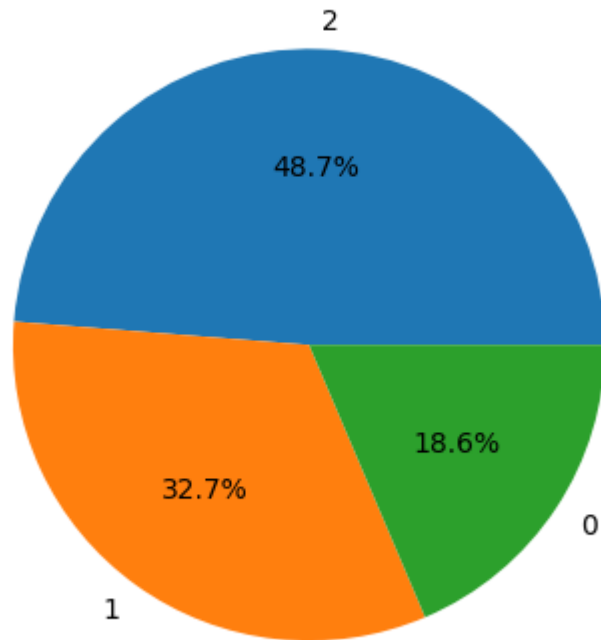
```
In [15]: import seaborn as sns

corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
plt.title("Correlation Heatmap")
plt.show()
```

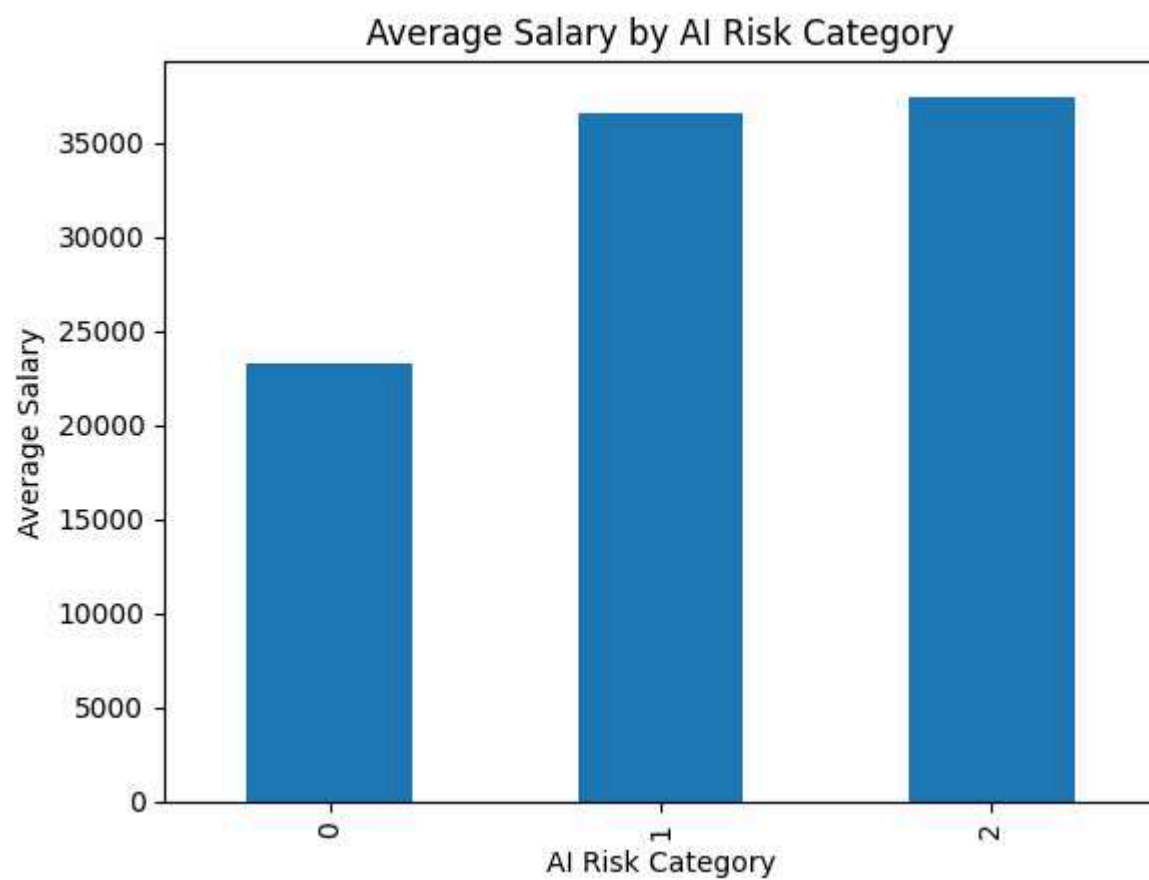


```
In [16]: df['ai_risk_category'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title("AI Risk Category Distribution")
plt.ylabel("")
plt.show()
```

AI Risk Category Distribution

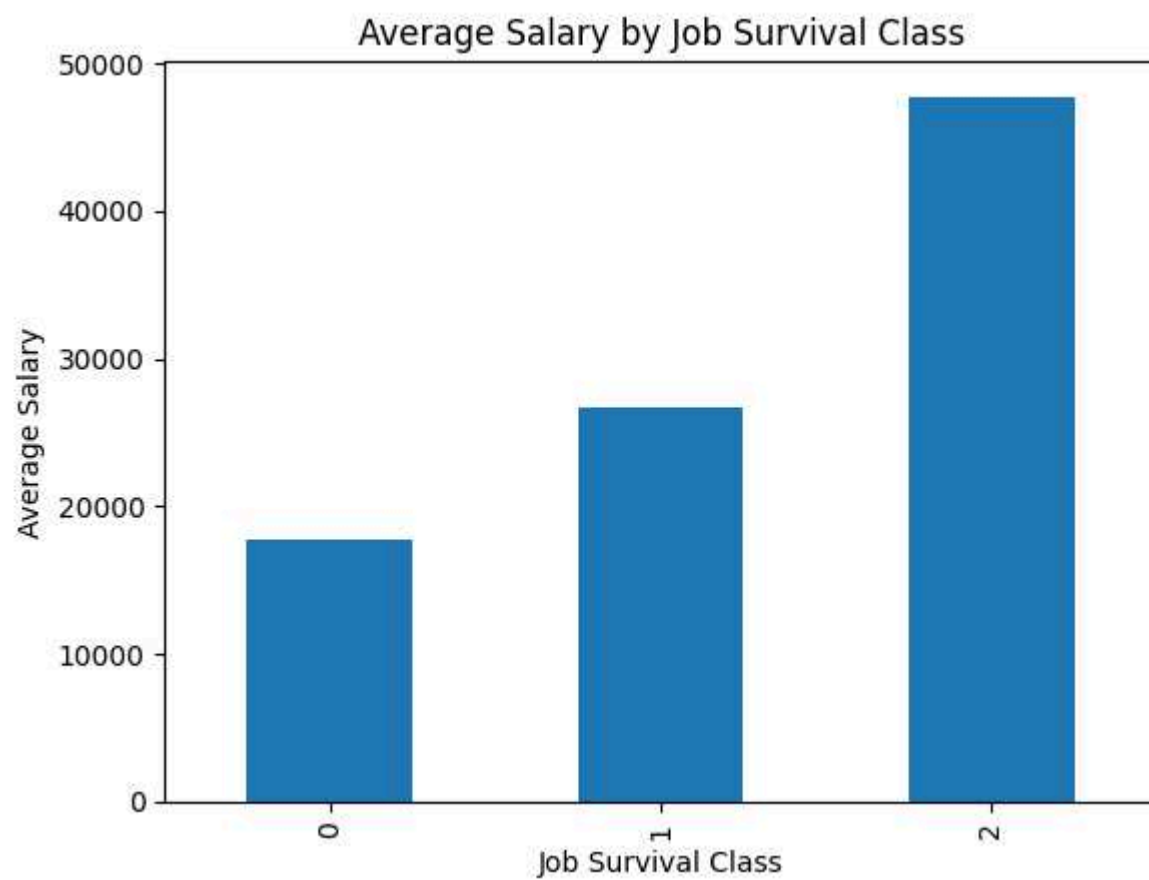


```
In [17]: df.groupby('ai_risk_category')['salary'].mean().plot(kind='bar')
plt.title("Average Salary by AI Risk Category")
plt.xlabel("AI Risk Category")
plt.ylabel("Average Salary")
plt.show()
```



```
In [25]: import matplotlib.pyplot as plt

df.groupby('job_survival_class')['salary'].mean().plot(kind='bar')
plt.title("Average Salary by Job Survival Class")
plt.xlabel("Job Survival Class")
plt.ylabel("Average Salary")
plt.show()
```



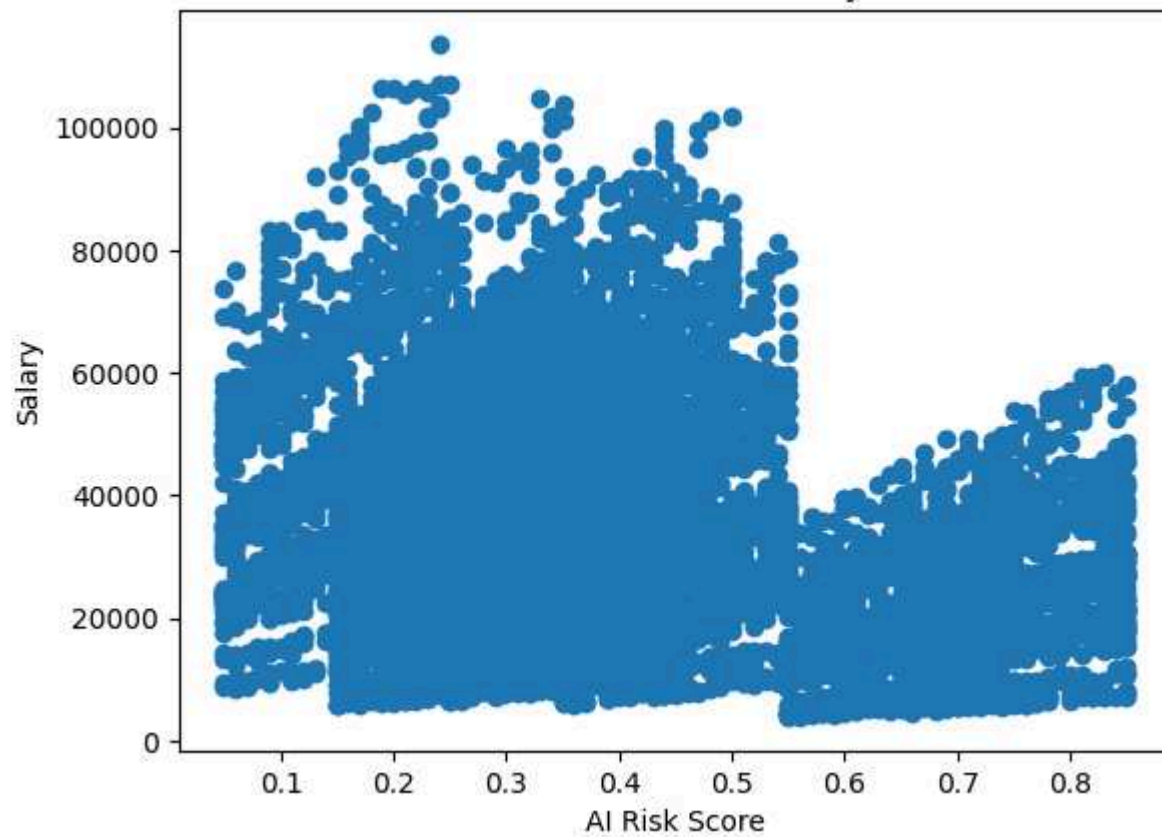
```
In [26]: import matplotlib.pyplot as plt

df.groupby('experience_level')['salary'].mean().plot(kind='bar')
plt.title("Average Salary by Experience Level")
plt.xlabel("Experience Level")
plt.ylabel("Average Salary")
plt.show()
```

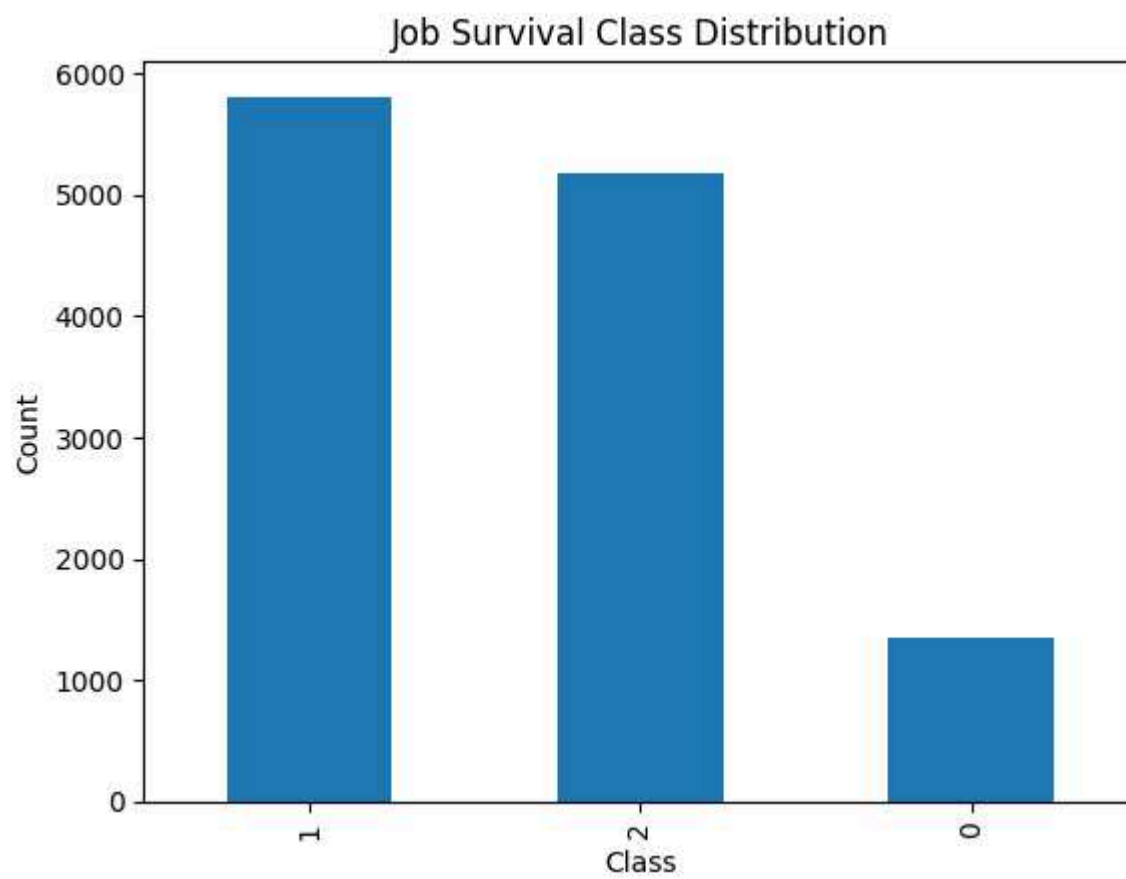


```
In [27]: plt.scatter(df['ai_risk_score'], df['salary'])
plt.title("AI Risk Score vs Salary")
plt.xlabel("AI Risk Score")
plt.ylabel("Salary")
plt.show()
```

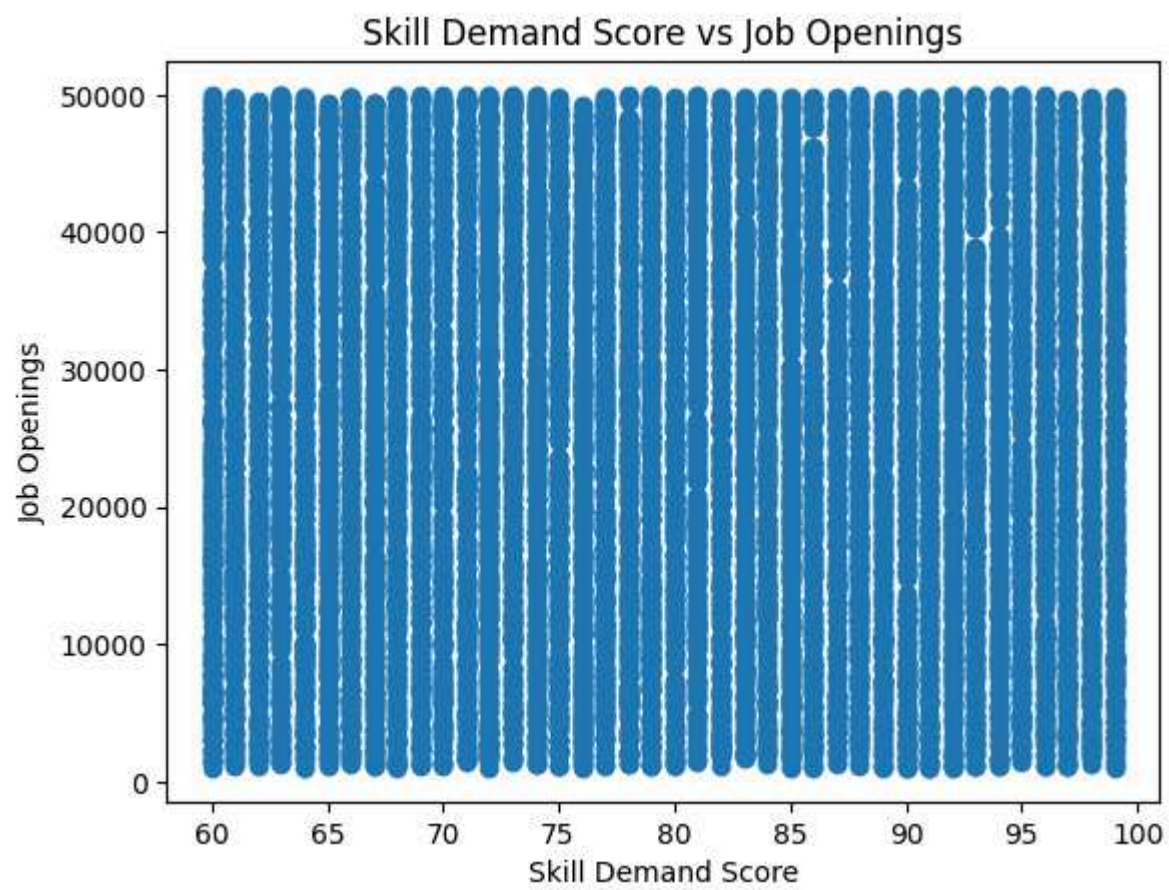
AI Risk Score vs Salary



```
In [28]: df['job_survival_class'].value_counts().plot(kind='bar')
plt.title("Job Survival Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```

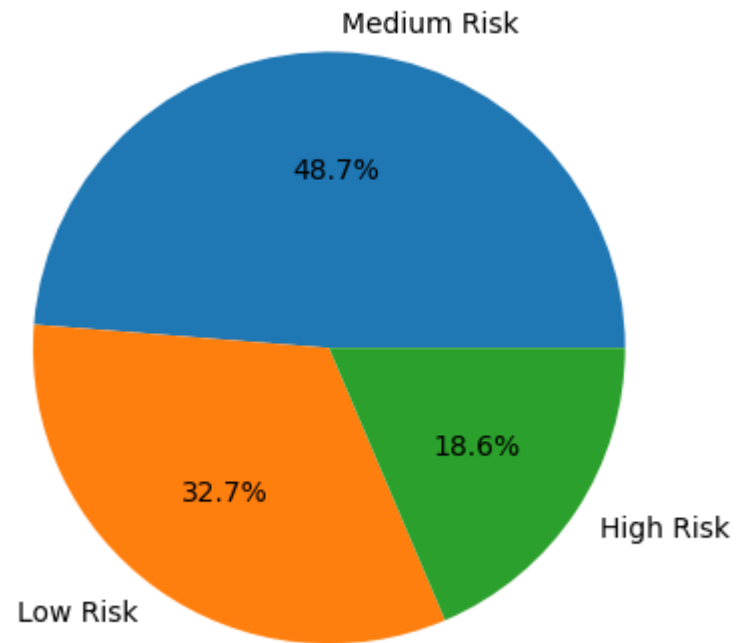


```
In [29]: plt.scatter(df['skill_demand_score'], df['job_openings'])  
plt.title("Skill Demand Score vs Job Openings")  
plt.xlabel("Skill Demand Score")  
plt.ylabel("Job Openings")  
plt.show()
```



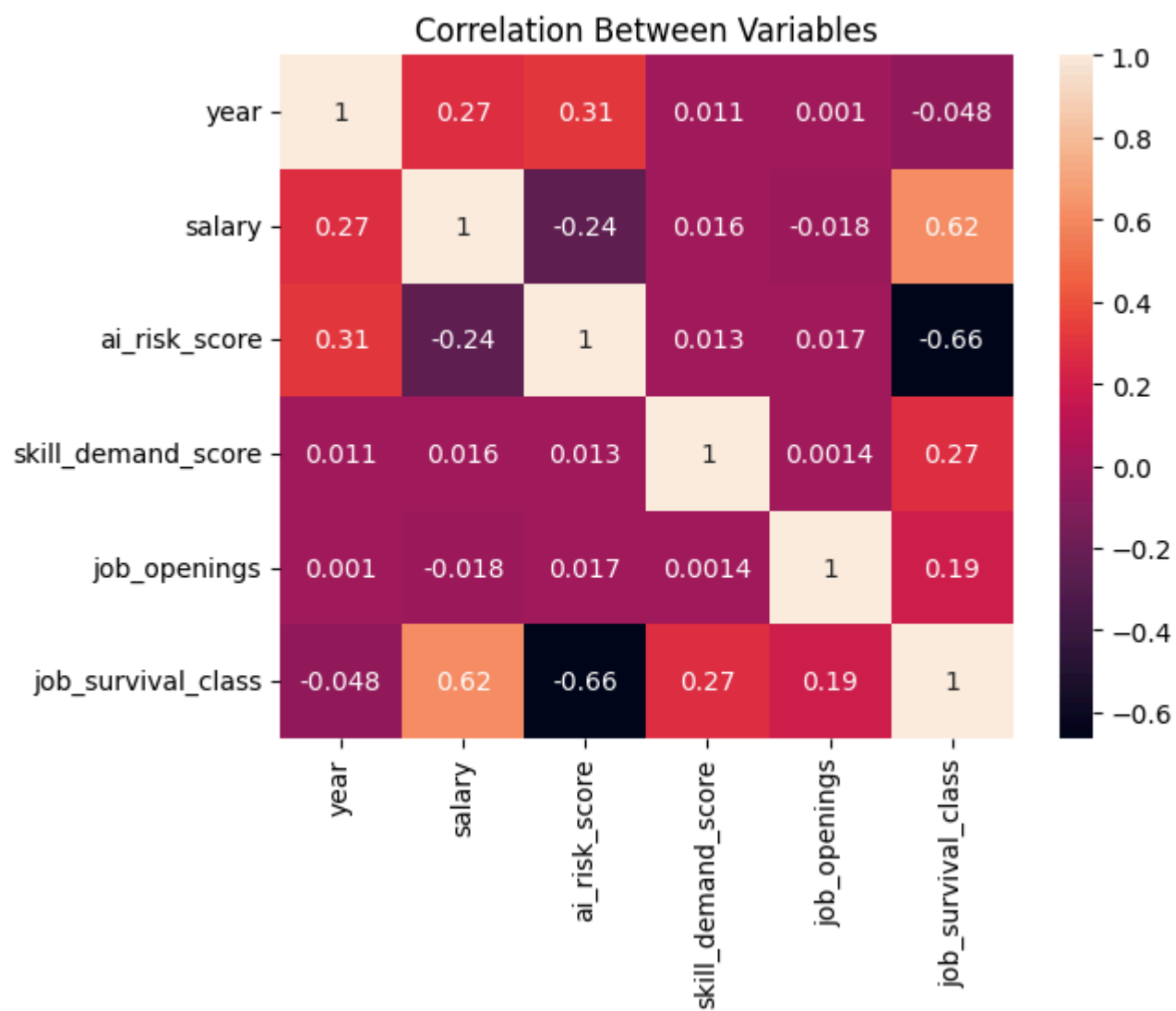
```
In [30]: df['ai_risk_category'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title("AI Risk Category Distribution")
plt.ylabel("")
plt.show()
```

AI Risk Category Distribution

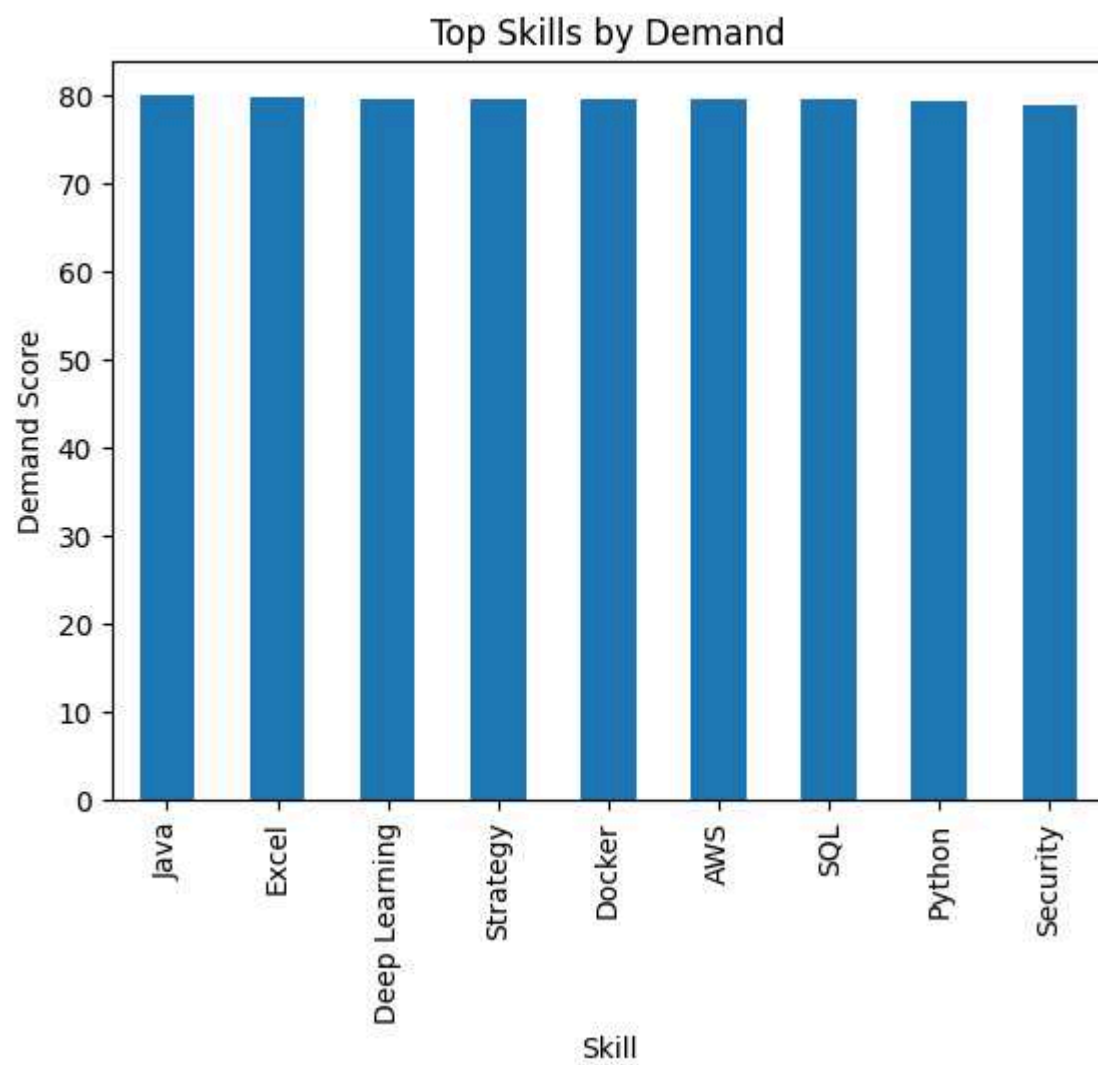


```
In [31]: import seaborn as sns

corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
plt.title("Correlation Between Variables")
plt.show()
```



```
In [32]: df.groupby('primary_skill')['skill_demand_score'].mean().sort_values(ascending=False).head(10).plot(kind='bar')
plt.title("Top Skills by Demand")
plt.xlabel("Skill")
plt.ylabel("Demand Score")
plt.show()
```



```
In [33]: from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X, y)

importances = pd.Series(model.feature_importances_, index=X.columns)
print(importances.sort_values(ascending=False))
```

```
-----  
NameError                                Traceback (most recent call last)
```

```
Cell In[33], line 4  
      1 from sklearn.ensemble import RandomForestClassifier  
      3 model = RandomForestClassifier()  
----> 4 model.fit(X, y)  
      6 importances = pd.Series(model.feature_importances_, index=X.columns)  
      7 print(importances.sort_values(ascending=False))
```

```
NameError: name 'X' is not defined
```

```
In [34]: import pandas as pd  
        from sklearn.preprocessing import LabelEncoder  
  
        df_encoded = df.copy()  
        le = LabelEncoder()  
  
        for col in df_encoded.select_dtypes(include='object'):  
            df_encoded[col] = le.fit_transform(df_encoded[col])
```

```
C:\Users\Maro\AppData\Local\Temp\ipykernel_8312\2691317883.py:7: Pandas4Warning: For backward compatibility, 'str' dtypes are included by select_dtypes when 'object' dtype is specified. This behavior is deprecated and will be removed in a future version. Explicitly pass 'str' to `include` to select them, or to `exclude` to remove them and silence this warning.  
See https://pandas.pydata.org/docs/user\_guide/migration-3-strings.html#string-migration-select-dtypes for details on how to write code that works with pandas 2 and 3.  
    for col in df_encoded.select_dtypes(include='object'):
```

```
In [35]: df_encoded = pd.get_dummies(df, drop_first=True)
```

```
In [36]: X = df_encoded.drop('job_survival_class', axis=1)  
        y = df_encoded['job_survival_class']
```

```
In [37]: from sklearn.ensemble import RandomForestClassifier  
  
        model = RandomForestClassifier()  
        model.fit(X, y)  
  
        import pandas as pd  
        importances = pd.Series(model.feature_importances_, index=X.columns)  
        print(importances.sort_values(ascending=False))
```

salary	0.218294
skill_demand_score	0.141811
job_openings	0.111042
ai_risk_score	0.106900
experience_level_Senior	0.049456
country_India	0.036068
year	0.035712
ai_risk_category_Low Risk	0.034959
job_title_Business Analyst	0.030705
salary_bucket_Medium	0.026457
primary_skill_Excel	0.022848
ai_risk_category_Medium Risk	0.020868
primary_skill_Deep Learning	0.020375
job_title_Data Analyst	0.017699
primary_skill_SQL	0.017603
experience_level_Mid	0.015030
country_USA	0.009974
job_title_Software Engineer	0.008851
salary_bucket_Low	0.008097
education_level_PhD	0.007913
primary_skill_Java	0.007719
education_level_Master	0.007683
country_Germany	0.005818
job_title_ML Engineer	0.005678
country_UK	0.005595
country_Canada	0.005510
primary_skill_Python	0.003706
job_title_Cloud Engineer	0.002976
job_title_Data Scientist	0.002889
job_title_Cybersecurity Analyst	0.002464
primary_skill_Security	0.002352
job_title_Product Manager	0.002130
primary_skill_Docker	0.001765
job_title_DevOps Engineer	0.001618
primary_skill_Strategy	0.001436

dtype: float64

```
In [38]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']
```

```

# 4) Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 5) Build the Decision Tree model
model = DecisionTreeClassifier(random_state=42, max_depth=5)
model.fit(X_train, y_train)

# 6) Make predictions
y_pred = model.predict(X_test)

# 7) Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# 8) Plot the tree
plt.figure(figsize=(20, 10))
plot_tree(
    model,
    feature_names=X.columns,
    class_names=[str(c) for c in sorted(y.unique())],
    filled=True,
    fontsize=8
)
plt.title("Decision Tree Classifier")
plt.show()

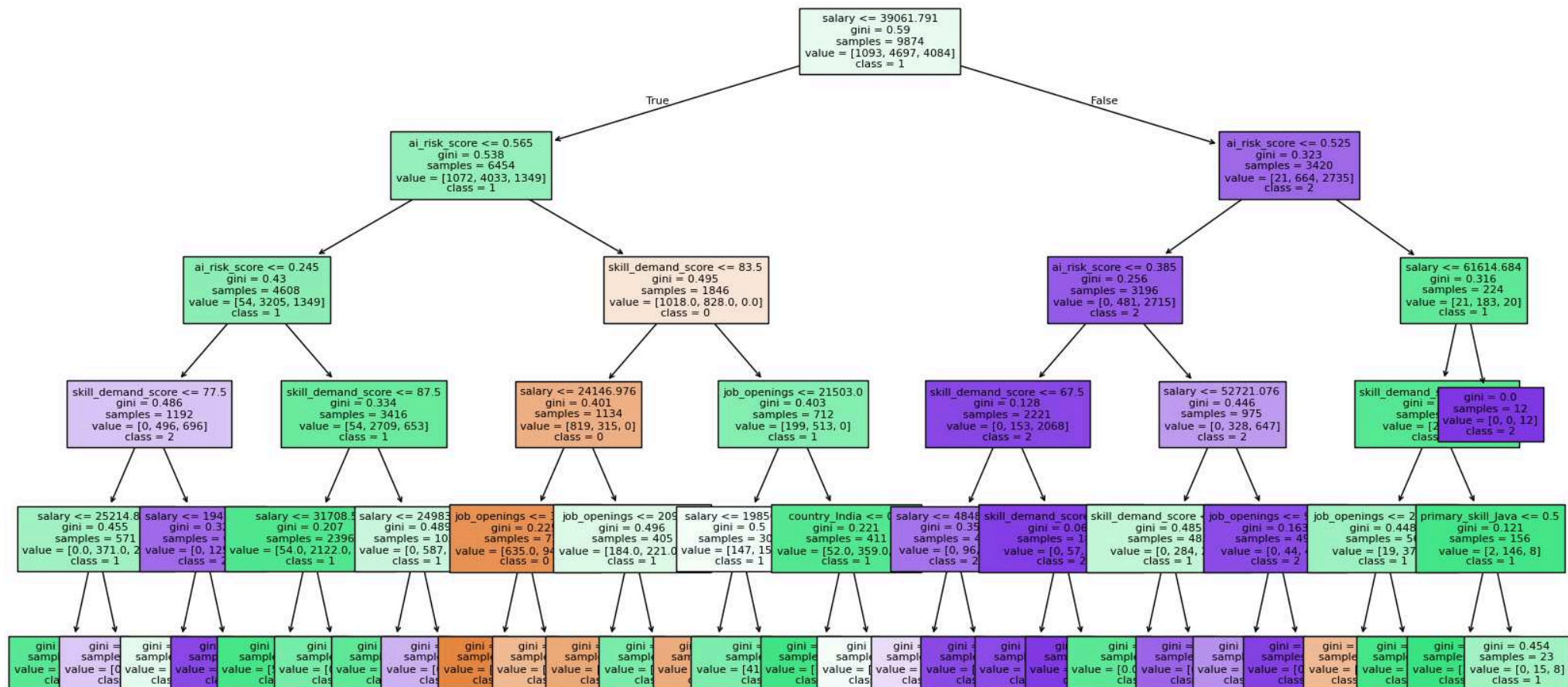
```

Accuracy: 0.8469015795868773

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.75	0.79	262
1	0.82	0.84	0.83	1109
2	0.88	0.88	0.88	1098
accuracy			0.85	2469
macro avg	0.84	0.82	0.83	2469
weighted avg	0.85	0.85	0.85	2469

Decision Tree Classifier



```
In [39]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_absolute_error, r2_score
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("Future of Jobs AI Dataset.csv")
df_encoded = pd.get_dummies(df, drop_first=True)
```

```
X = df_encoded.drop('salary', axis=1)
y = df_encoded['salary']
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```

model = DecisionTreeRegressor(random_state=42, max_depth=5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("MAE:", mean_absolute_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

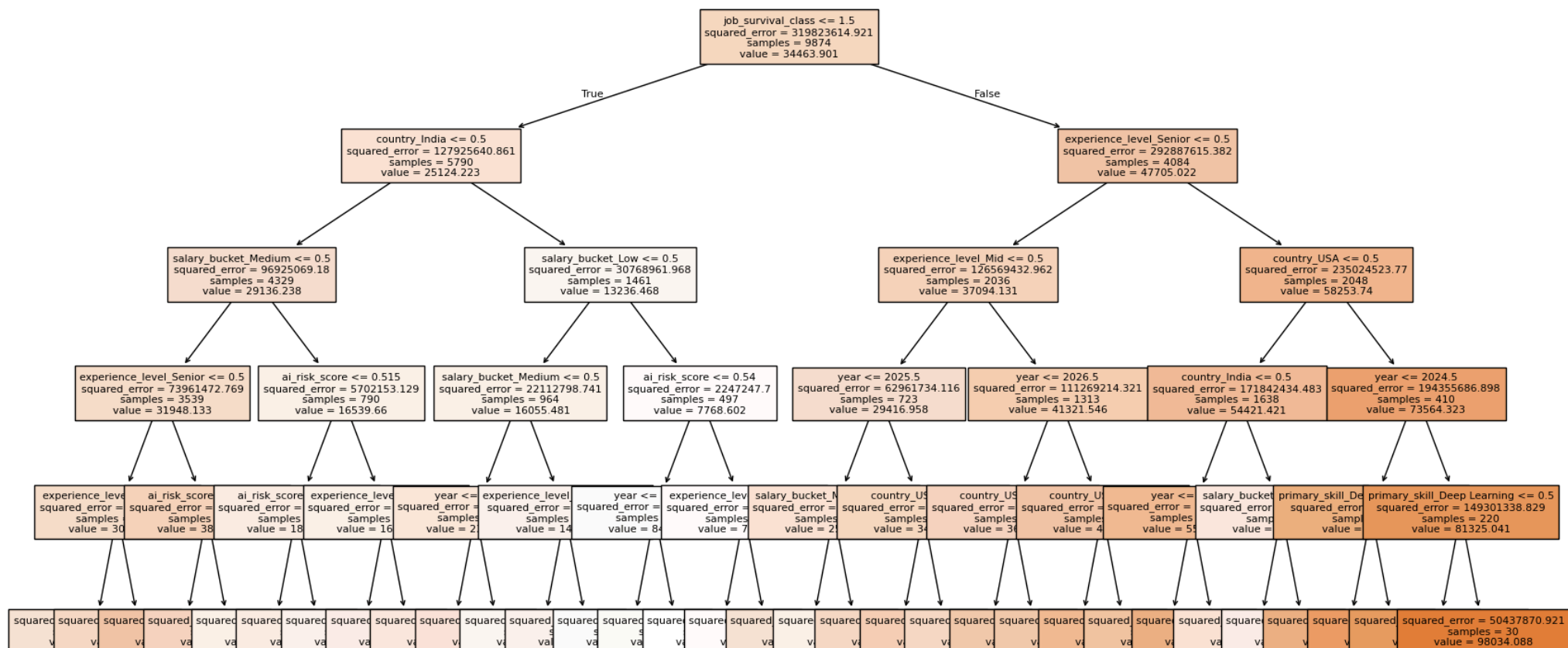
plt.figure(figsize=(20, 10))
plot_tree(model, feature_names=X.columns, filled=True, fontsize=8)
plt.title("Decision Tree Regressor")
plt.show()

```

MAE: 5196.102351178014

R2 Score: 0.8563911607430648

Decision Tree Regressor



```
In [40]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Encode categorical data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Build model
model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

# Train
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

Accuracy: 0.9639530174159578

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.91	0.93	262
1	0.95	0.97	0.96	1109
2	0.98	0.97	0.98	1098
accuracy			0.96	2469
macro avg	0.96	0.95	0.96	2469
weighted avg	0.96	0.96	0.96	2469

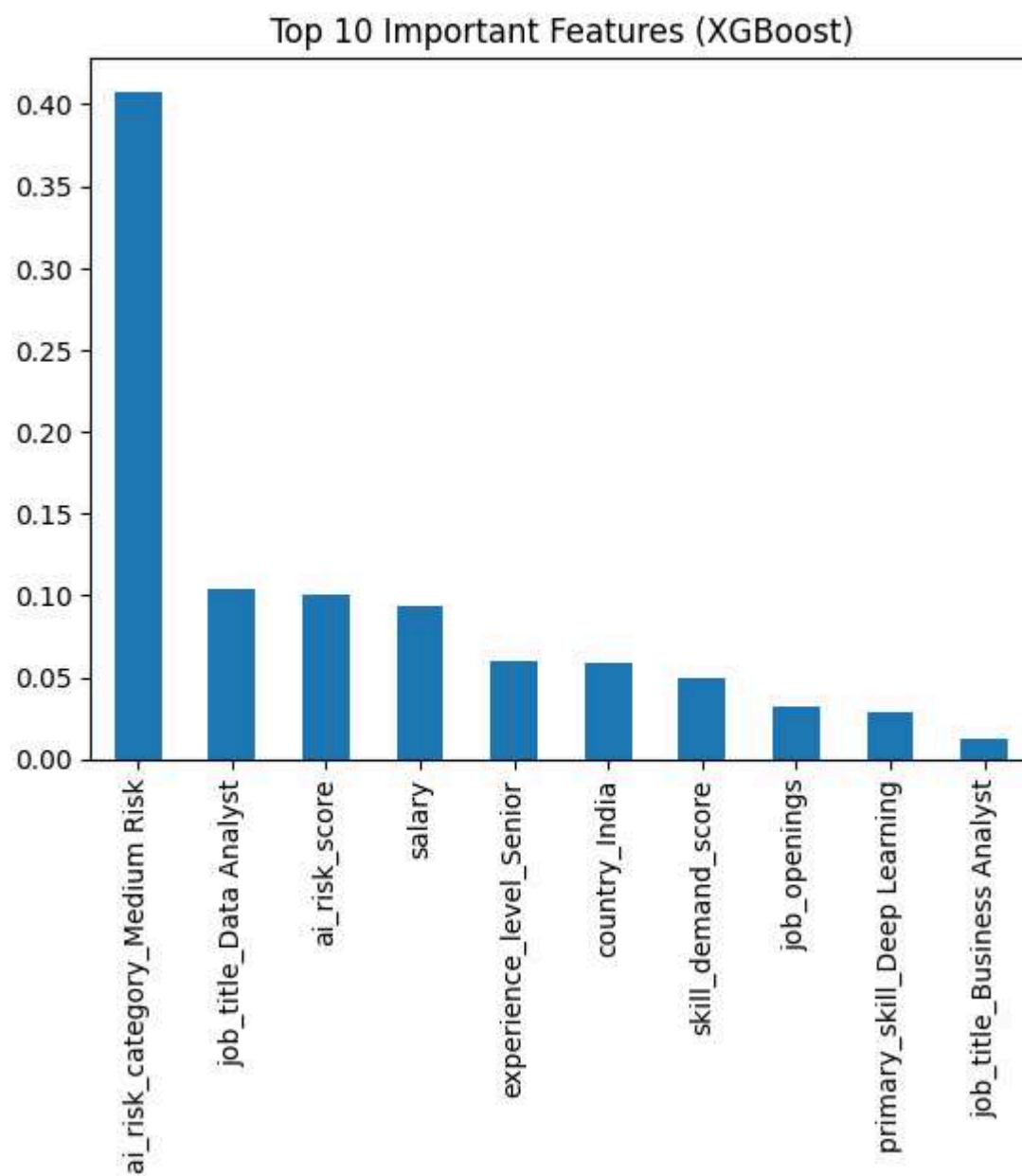
```
In [41]: import matplotlib.pyplot as plt
import pandas as pd

importance = pd.Series(model.feature_importances_, index=X.columns)
importance = importance.sort_values(ascending=False).head(10)

print(importance)

importance.plot(kind='bar')
plt.title("Top 10 Important Features (XGBoost)")
plt.show()
```

```
ai_risk_category_Medium Risk    0.407862
job_title_Data Analyst         0.104018
ai_risk_score                   0.100731
salary                         0.093438
experience_level_Senior         0.060164
country_India                   0.058497
skill_demand_score             0.049128
job_openings                   0.031589
primary_skill_Deep Learning    0.028336
job_title_Business Analyst     0.011890
dtype: float32
```



```
In [42]: from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, r2_score

X = df_encoded.drop('salary', axis=1)
y = df_encoded['salary']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
```

```

)

model = XGBRegressor(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("MAE:", mean_absolute_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

```

MAE: 1924.391388641024
R2 Score: 0.9810595584134281

```

In [44]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# Define features and target (classification)
X = df_encoded.drop(['job_survival_class', 'salary_bucket', 'ai_risk_category'], axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Decision Tree
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)

dt_pred = dt_model.predict(X_test)
dt_acc = accuracy_score(y_test, dt_pred)

```

```
print("Decision Tree Accuracy:", dt_acc)

# XGBoost
xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

xgb_model.fit(X_train, y_train)

xgb_pred = xgb_model.predict(X_test)
xgb_acc = accuracy_score(y_test, xgb_pred)

print("XGBoost Accuracy:", xgb_acc)
```

KeyError Traceback (most recent call last)

Cell In[44], line 14

```
11 df_encoded = pd.get_dummies(df, drop_first=True)
13 # Define features and target (classification)
---> 14 X = df_encoded.drop([ , , ], axis=1)
15 y = df_encoded['job_survival_class']
17 # Split data
```

File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\frame.py:6288, in DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors)

```
6130 def drop(
6131     self,
6132     labels: IndexLabel | ListLike = None,
6133     errors: IgnoreRaise = "raise",
6134 ) -> DataFrame | None:
6141     """
6142     Drop specified labels from rows or columns.
6143
6144     (...) 6286         weight 1.0      0.8
6287     """
-> 6288     return super().drop(
6289         labels=labels,
6290         axis=axis,
6291         index=index,
6292         columns=columns,
6293         level=level,
6294         inplace=inplace,
6295         errors=errors,
6296     )
```

File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\generic.py:4644, in NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)

```
4642 for axis, labels in axes.items():
4643     if labels is not None:
-> 4644         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
4646 if inplace:
4647     self._update_inplace(obj)
```

File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\generic.py:4686, in NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)

```
4684     new_axis = axis.drop(labels, level=level, errors=errors)
4685     else:
-> 4686     new_axis = axis.drop(labels, errors=errors)
4687     indexer = axis.get_indexer(new_axis)
4689 # Case for non-unique axis
```

```
4690 else:
```

```
File D:\anaconda\envs\project2\Lib\site-packages\pandas\core\indexes\base.py:7268, in Index.drop(self, labels, errors)
```

```
7266 if mask.any():
7267     if errors != "ignore":
-> 7268         raise KeyError(f"{labels[mask].tolist()} not found in axis")
7269     indexer = indexer[~mask]
7270 return self.delete(indexer)
```

```
KeyError: "['salary_bucket', 'ai_risk_category'] not found in axis"
```

```
In [45]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns BEFORE encoding
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical columns
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Decision Tree
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)
dt_acc = accuracy_score(y_test, dt_pred)

# XGBoost
xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
```

```

    random_state=42,
    eval_metric='mlogloss'
)
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)
xgb_acc = accuracy_score(y_test, xgb_pred)

print("Decision Tree Accuracy:", dt_acc)
print("XGBoost Accuracy:", xgb_acc)

print("\nDecision Tree Report:\n")
print(classification_report(y_test, dt_pred))

print("\nXGBoost Report:\n")
print(classification_report(y_test, xgb_pred))

```

Decision Tree Accuracy: 0.8469015795868773

XGBoost Accuracy: 0.9651680842446334

Decision Tree Report:

	precision	recall	f1-score	support
0	0.83	0.75	0.79	262
1	0.82	0.84	0.83	1109
2	0.88	0.88	0.88	1098
accuracy			0.85	2469
macro avg	0.84	0.82	0.83	2469
weighted avg	0.85	0.85	0.85	2469

XGBoost Report:

	precision	recall	f1-score	support
0	0.96	0.91	0.94	262
1	0.95	0.98	0.96	1109
2	0.98	0.97	0.98	1098
accuracy			0.97	2469
macro avg	0.96	0.95	0.96	2469
weighted avg	0.97	0.97	0.97	2469

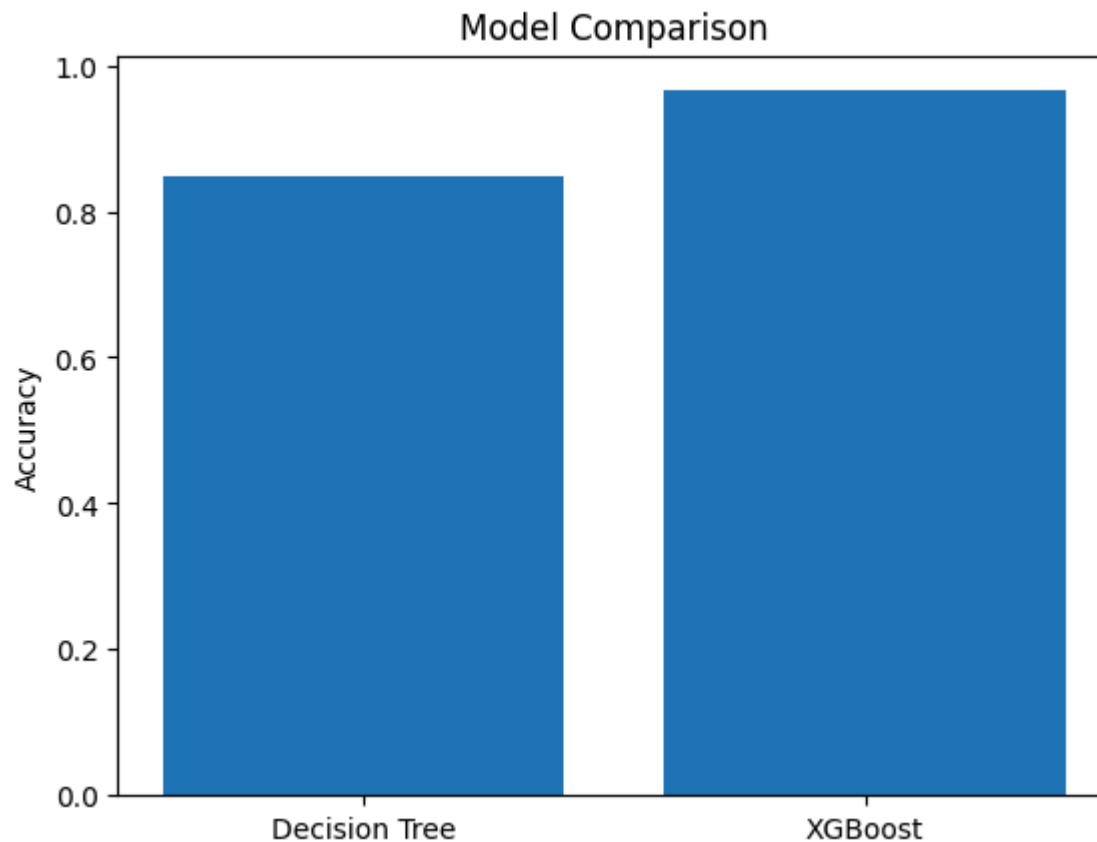
In [46]: `print(df_encoded.columns.tolist())`

```
['year', 'salary', 'ai_risk_score', 'skill_demand_score', 'job_openings', 'job_survival_class', 'job_title_Business Analyst', 'job_title_Cloud Engineer', 'job_title_Cybersecurity Analyst', 'job_title_Data Analyst', 'job_title_Data Scientist', 'job_title_DevOps Engineer', 'job_title_ML Engineer', 'job_title_Product Manager', 'job_title_Software Engineer', 'country_Canada', 'country_Germany', 'country_India', 'country_UK', 'country_USA', 'experience_level_Mid', 'experience_level_Senior', 'education_level_Master', 'education_level_PhD', 'primary_skill_Deep Learning', 'primary_skill_Docker', 'primary_skill_Excel', 'primary_skill_Java', 'primary_skill_Python', 'primary_skill_SQL', 'primary_skill_Security', 'primary_skill_Strategy']
```

```
In [47]: import matplotlib.pyplot as plt
```

```
models = ['Decision Tree', 'XGBoost']  
scores = [dt_acc, xgb_acc]
```

```
plt.bar(models, scores)  
plt.title("Model Comparison")  
plt.ylabel("Accuracy")  
plt.show()
```



```
In [48]: import pandas as pd  
from sklearn.model_selection import train_test_split  
from sklearn.feature_selection import RFE
```

```

from sklearn.tree import DecisionTreeClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create model
model = DecisionTreeClassifier(random_state=42)

# Apply RFE ( 5 features)
selector = RFE(model, n_features_to_select=5)
selector = selector.fit(X_train, y_train)

# Get selected features
selected_features = X.columns[selector.support_]

print("Selected Features:")
print(selected_features)

```

```

Selected Features:
Index(['year', 'salary', 'ai_risk_score', 'skill_demand_score',
      'job_openings'],
      dtype='str')

```

```

In [49]: ranking = pd.Series(selector.ranking_, index=X.columns)
print(ranking.sort_values())

```

year	1
salary	1
ai_risk_score	1
skill_demand_score	1
job_openings	1
country_India	2
experience_level_Senior	3
country_UK	4
country_Canada	5
experience_level_Mid	6
country_USA	7
country_Germany	8
education_level_PhD	9
job_title_Cloud Engineer	10
job_title_Data Scientist	11
job_title_Data Analyst	12
job_title_ML Engineer	13
primary_skill_Java	14
education_level_Master	15
primary_skill_Strategy	16
primary_skill_Security	17
job_title_Software Engineer	18
job_title_Cybersecurity Analyst	19
job_title_Product Manager	20
primary_skill_Docker	21
primary_skill_Python	22
primary_skill_Deep Learning	23
primary_skill_Excel	24
job_title_Business Analyst	25
job_title_DevOps Engineer	26
primary_skill_SQL	27
dtype:	int64

```
In [50]: X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

model.fit(X_train_selected, y_train)
y_pred = model.predict(X_test_selected)

from sklearn.metrics import accuracy_score
print("Accuracy after RFE:", accuracy_score(y_test, y_pred))
```

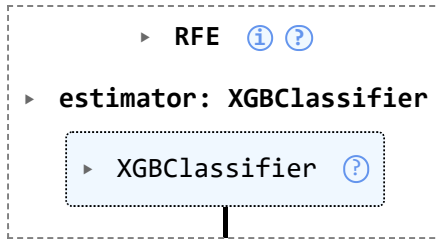
Accuracy after RFE: 0.9291211016605914

```
In [52]: from xgboost import XGBClassifier

model = XGBClassifier(random_state=42, eval_metric='mlogloss')
```

```
selector = RFE(model, n_features_to_select=5)
selector.fit(X_train, y_train)
```

Out[52]:



In [53]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import RFE
from sklearn.metrics import accuracy_score

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# ♦ BEFORE RFE
# -----
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred_before = model.predict(X_test)
acc_before = accuracy_score(y_test, y_pred_before)

print("Accuracy BEFORE RFE:", acc_before)
```

```

# -----
# ♦ APPLY RFE
# -----
selector = RFE(model, n_features_to_select=5)
selector.fit(X_train, y_train)

selected_features = X.columns[selector.support_]
print("\nSelected Features:", selected_features)

# -----
# ♦ AFTER RFE
# -----
X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

model.fit(X_train_selected, y_train)
y_pred_after = model.predict(X_test_selected)

acc_after = accuracy_score(y_test, y_pred_after)

print("\nAccuracy AFTER RFE:", acc_after)

```

Accuracy BEFORE RFE: 0.9218307006885379

Selected Features: Index(['year', 'salary', 'ai_risk_score', 'skill_demand_score',
'job_openings'],
dtype='str')

Accuracy AFTER RFE: 0.9291211016605914

```

In [54]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_selection import RFE
from xgboost import XGBClassifier

# 1. Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2. Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3. Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4. Define X and y
X = df_encoded.drop('job_survival_class', axis=1)

```

```

y = df_encoded['job_survival_class']

# 5. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6. Feature Selection (RFE)
base_model = XGBClassifier(random_state=42, eval_metric='mlogloss')
selector = RFE(base_model, n_features_to_select=5)
selector.fit(X_train, y_train)

selected_features = X.columns[selector.support_]
print("Selected Features:", selected_features)

# 7. Train final model using selected features
X_train_sel = X_train[selected_features]
X_test_sel = X_test[selected_features]

model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42,
    eval_metric='mlogloss'
)

model.fit(X_train_sel, y_train)

# 8. Predictions
y_pred = model.predict(X_test_sel)

# 9. Evaluation
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

```

```
Selected Features: Index(['salary', 'ai_risk_score', 'skill_demand_score',  
      'job_title_Data Analyst', 'country_India'],  
      dtype='str')
```

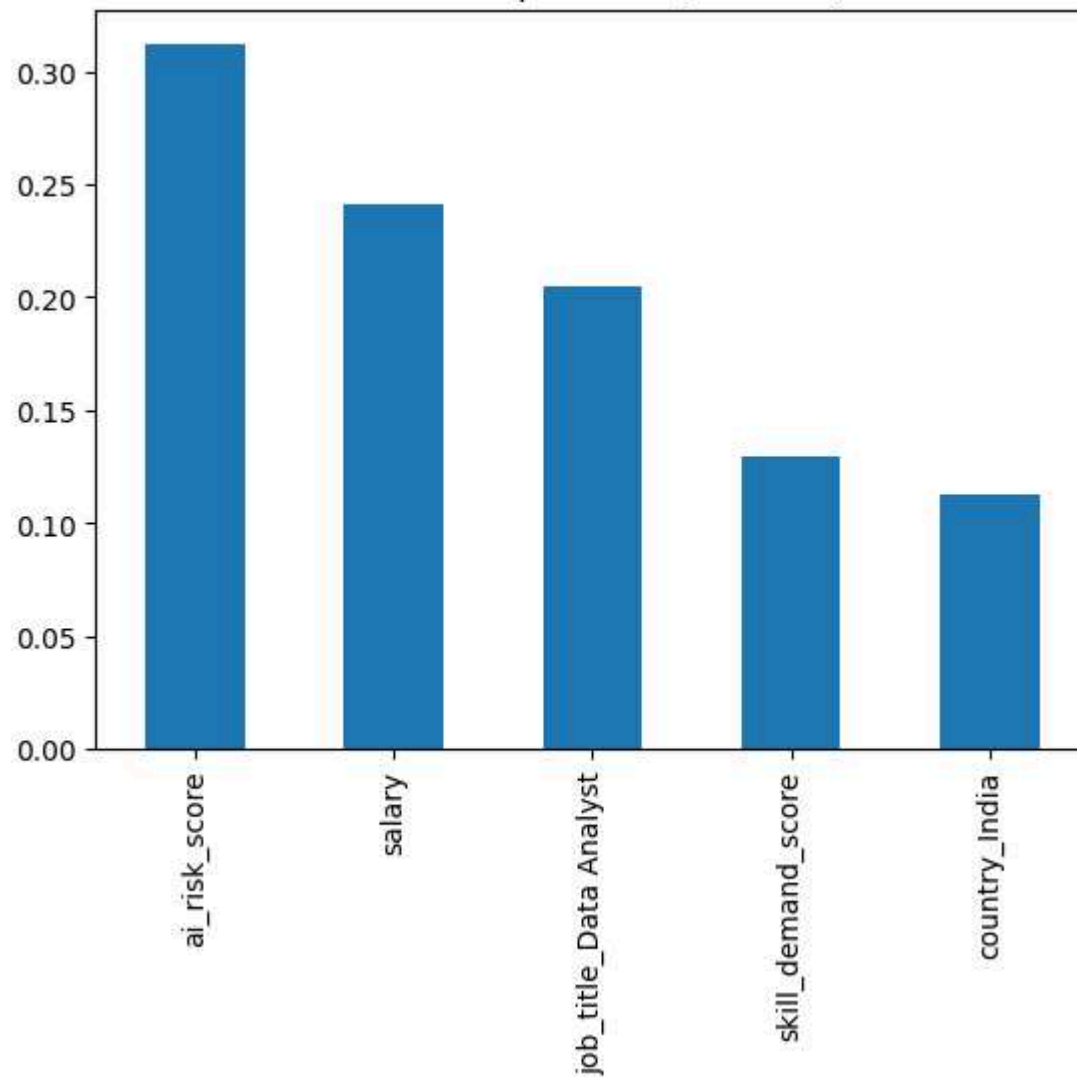
Accuracy: 0.8865937626569461

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.82	0.83	262
1	0.86	0.89	0.88	1109
2	0.92	0.90	0.91	1098
accuracy			0.89	2469
macro avg	0.88	0.87	0.87	2469
weighted avg	0.89	0.89	0.89	2469

```
In [55]: import matplotlib.pyplot as plt  
import pandas as pd  
  
importance = pd.Series(model.feature_importances_, index=selected_features)  
importance = importance.sort_values(ascending=False)  
  
importance.plot(kind='bar')  
plt.title("Feature Importance (XGBoost)")  
plt.show()
```

Feature Importance (XGBoost)



```
In [56]: import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Decision Tree metrics
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_precision = precision_score(y_test, dt_pred, average='weighted')
dt_recall = recall_score(y_test, dt_pred, average='weighted')
dt_f1 = f1_score(y_test, dt_pred, average='weighted')

# XGBoost metrics
xgb_accuracy = accuracy_score(y_test, xgb_pred)
```

```
xgb_precision = precision_score(y_test, xgb_pred, average='weighted')
xgb_recall = recall_score(y_test, xgb_pred, average='weighted')
xgb_f1 = f1_score(y_test, xgb_pred, average='weighted')
```

```
# Create table
```

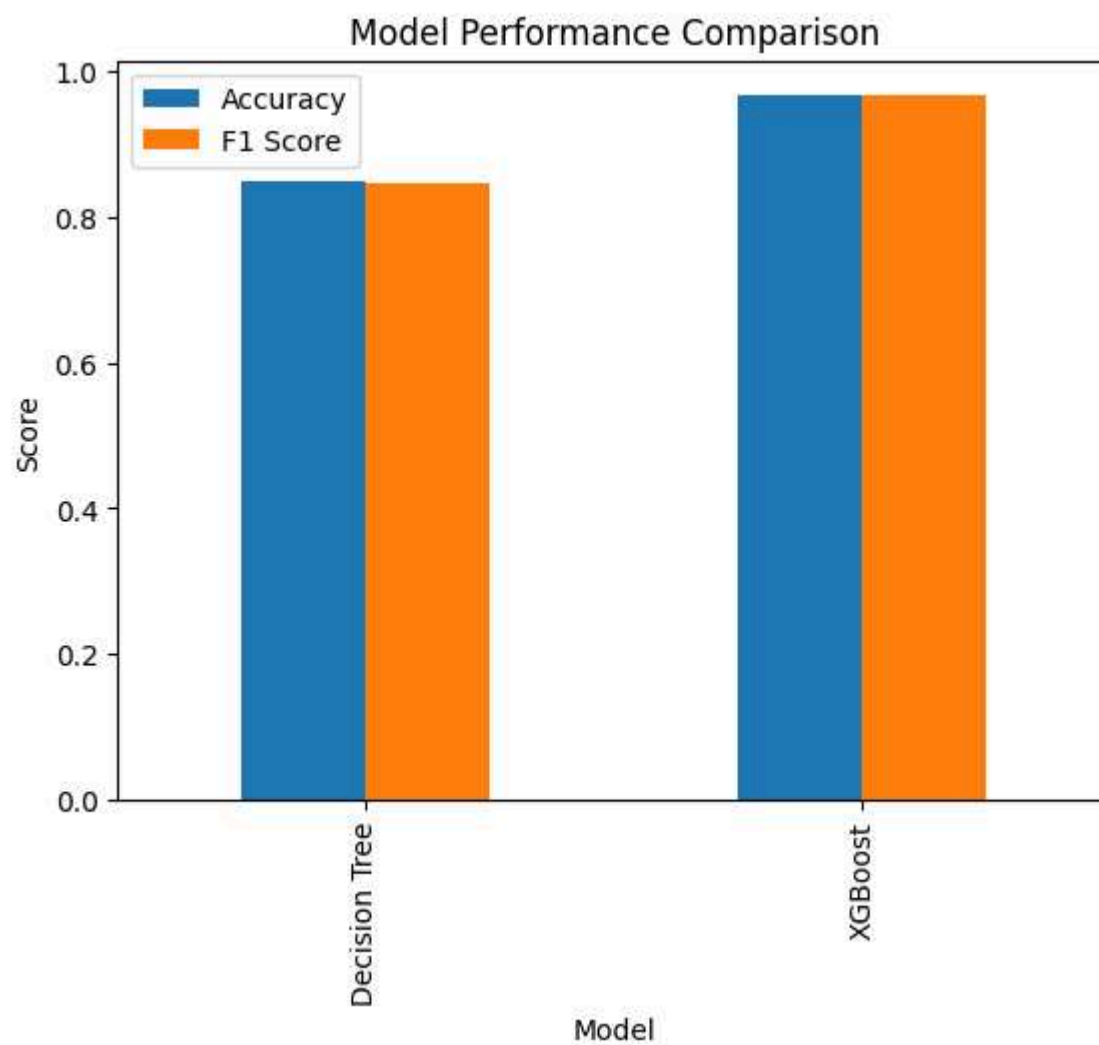
```
comparison = pd.DataFrame({
    "Model": ["Decision Tree", "XGBoost"],
    "Accuracy": [dt_accuracy, xgb_accuracy],
    "Precision": [dt_precision, xgb_precision],
    "Recall": [dt_recall, xgb_recall],
    "F1 Score": [dt_f1, xgb_f1]
})
```

```
print(comparison)
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.846902	0.846974	0.846902	0.846678
1	XGBoost	0.965168	0.965506	0.965168	0.965152

In [57]: `import matplotlib.pyplot as plt`

```
comparison.set_index('Model')[['Accuracy', 'F1 Score']].plot(kind='bar')
plt.title("Model Performance Comparison")
plt.ylabel("Score")
plt.show()
```



```
In [59]: import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical columns
df_encoded = pd.get_dummies(df, drop_first=True)
```

```

# Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Model
dt_model = DecisionTreeClassifier(random_state=42)

# Parameter grid
param_grid_dt = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# Grid Search
grid_dt = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid_dt,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_dt.fit(X_train, y_train)

# Best model
best_dt = grid_dt.best_estimator_

# Prediction
dt_pred = best_dt.predict(X_test)

print("Best Decision Tree Parameters:", grid_dt.best_params_)
print("Best CV Score:", grid_dt.best_score_)
print("Test Accuracy:", accuracy_score(y_test, dt_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, dt_pred))

```

```
Best Decision Tree Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best CV Score: 0.923232606158542
Test Accuracy: 0.9400567031186715
```

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.89	0.91	262
1	0.93	0.94	0.93	1109
2	0.96	0.95	0.95	1098
accuracy			0.94	2469
macro avg	0.93	0.93	0.93	2469
weighted avg	0.94	0.94	0.94	2469

```
In [61]: import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode categorical columns
df_encoded = pd.get_dummies(df, drop_first=True)

# Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Model
xgb_model = XGBClassifier(
    random_state=42,
    eval_metric='mlogloss'
)
```

```
# Parameter grid
param_grid_xgb = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Grid Search
grid_xgb = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid_xgb,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_xgb.fit(X_train, y_train)

# Best model
best_xgb = grid_xgb.best_estimator_

# Prediction
xgb_pred = best_xgb.predict(X_test)

print("Best XGBoost Parameters:", grid_xgb.best_params_)
print("Best CV Score:", grid_xgb.best_score_)
print("Test Accuracy:", accuracy_score(y_test, xgb_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, xgb_pred))
```

Best XGBoost Parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 150, 'subsample': 0.8}
Best CV Score: 0.9745790465930515
Test Accuracy: 0.9712434183880113

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.92	0.95	262
1	0.96	0.98	0.97	1109
2	0.99	0.98	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.96	2469
weighted avg	0.97	0.97	0.97	2469

In [74]: `from sklearn.linear_model import LogisticRegression`

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

D:\anaconda\envs\project2\Lib\site-packages\sklearn\linear_model_logistic.py:406: ConvergenceWarning: lbfgs failed to converge after 1000 iterations(s) (status=1):

STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

In [75]: `from sklearn.metrics import accuracy_score, classification_report`

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

Accuracy: 0.8991494532199271

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.82	0.83	262
1	0.88	0.90	0.89	1109
2	0.93	0.92	0.93	1098
accuracy			0.90	2469
macro avg	0.89	0.88	0.88	2469
weighted avg	0.90	0.90	0.90	2469

```
In [80]: from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
In [79]: model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train_scaled, y_train)
```

```
y_pred = model.predict(X_test_scaled)
```

```
In [81]: from sklearn.linear_model import LogisticRegression
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import accuracy_score, classification_report
```

```
# Scaling
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
# Model
```

```
model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train_scaled, y_train)
```

```
# Prediction
```

```
y_pred = model.predict(X_test_scaled)
```

```
# Evaluation
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

Accuracy: 0.9898744430943702

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.96	0.97	262
1	0.98	0.99	0.99	1109
2	0.99	0.99	0.99	1098
accuracy			0.99	2469
macro avg	0.99	0.98	0.99	2469
weighted avg	0.99	0.99	0.99	2469

```
In [82]: print(df['job_survival_class'].value_counts())
```

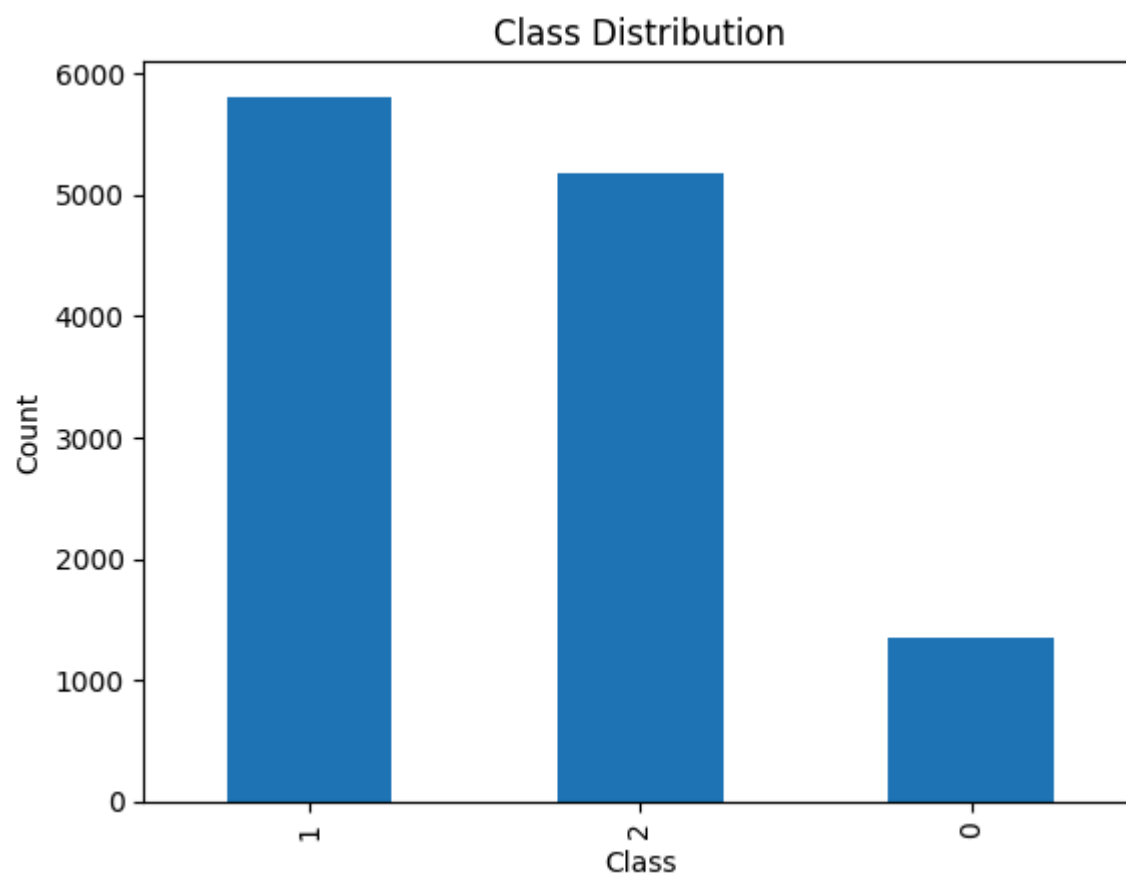
```
job_survival_class
1      5806
2      5182
0      1355
Name: count, dtype: int64
```

```
In [83]: print(df['job_survival_class'].value_counts(normalize=True) * 100)
```

```
job_survival_class
1      47.038807
2      41.983310
0      10.977882
Name: proportion, dtype: float64
```

```
In [84]: import matplotlib.pyplot as plt
```

```
df['job_survival_class'].value_counts().plot(kind='bar')
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```



```
In [86]: from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(class_weight='balanced')
```

```
In [87]: from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[87], line 1
----> 1 from imblearn.over_sampling import SMOTE
      3 smote = SMOTE(random_state=42)
      4 X_resampled, y_resampled = smote.fit_resample(X, y)

ModuleNotFoundError: No module named 'imblearn'
```

```
In [88]: import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Drop Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 🔥 Split FIRST
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 🔥 Apply SMOTE ONLY on training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[88], line 3
      1 import pandas as pd
      2 from sklearn.model_selection import train_test_split
----> 3 from imblearn.over_sampling import SMOTE
      4 # Load data
      5 df = pd.read_csv("Future of Jobs AI Dataset.csv")

ModuleNotFoundError: No module named 'imblearn'
```

```
In [89]: !pip install imbalanced-learn
```


[illegible]


```

# Encode data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 🔥 Split FIRST
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 🔥 Apply SMOTE ONLY on training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

```

```

-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[90], line 3
      1 import pandas as pd
      2 from sklearn.model_selection import train_test_split
----> 3 from imblearn.over_sampling import SMOTE
      5 # Load data
      6 df = pd.read_csv("Future of Jobs AI Dataset.csv")

ModuleNotFoundError: No module named 'imblearn'

```

```

In [91]: import sys
         !{sys.executable} -m pip install imbalanced-learn

```

```

Requirement already satisfied: imbalanced-learn in D:\anaconda\envs\project2\Lib\site-packages (0.14.1)
Requirement already satisfied: numpy<3,>=1.25.2 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (2.4.3)
Requirement already satisfied: scipy<2,>=1.11.4 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (1.17.1)
Requirement already satisfied: scikit-learn<2,>=1.4.2 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (1.8.0)
Requirement already satisfied: sklearn-compat<0.2,>=0.1.5 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (0.1.5)
Requirement already satisfied: joblib<2,>=1.2.0 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (1.5.3)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in D:\anaconda\envs\project2\Lib\site-packages (from imbalanced-learn) (3.6.0)

```

```

In [92]: from imblearn.over_sampling import SMOTE

```

```

-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[92], line 1
----> 1 from imblearn.over_sampling import SMOTE

ModuleNotFoundError: No module named 'imblearn'

```

```
In [93]: import sys
print(sys.executable)
```

D:\anaconda\envs\project2\python.exe

```
In [94]: import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# Drop Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# Encode data
df_encoded = pd.get_dummies(df, drop_first=True)

# Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 🔥 Split FIRST
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 🔥 Apply SMOTE ONLY on training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[94], line 3
      1 import pandas as pd
      2 from sklearn.model_selection import train_test_split
----> 3 from imblearn.over_sampling import SMOTE
      4 # Load data
      5 df = pd.read_csv("Future of Jobs AI Dataset.csv")

ModuleNotFoundError: No module named 'imblearn'
```

```
In [100... from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [101...

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(class_weight='balanced', max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

D:\anaconda\envs\project2\Lib\site-packages\sklearn\linear_model_logistic.py:406: ConvergenceWarning: lbfgs failed to converge after 1000 iterations (status=1):

STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

n_iter_i = _check_optimize_result(

In [102...

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
```

Accuracy: 0.880113406237343

Classification Report:

	precision	recall	f1-score	support
0	0.72	0.92	0.81	262
1	0.89	0.83	0.86	1109
2	0.92	0.92	0.92	1098
accuracy			0.88	2469
macro avg	0.84	0.89	0.86	2469
weighted avg	0.89	0.88	0.88	2469

Confusion Matrix:

```
[[ 242   20    0]
 [  94  924   91]
 [    0   91 1007]]
```

In [103...

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# 1. Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2. Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3. Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4. Define features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 5. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6. Build XGBoost model (optimized parameters)
model = XGBClassifier()
```

```

n_estimators=200,      # more trees = better Learning
max_depth=6,           # deeper trees
learning_rate=0.1,
subsample=0.8,
colsample_bytree=0.8,
random_state=42,
eval_metric='mlogloss'
)

# 7. Train
model.fit(X_train, y_train)

# 8. Predict
y_pred = model.predict(X_test)

# 9. Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

```

Accuracy: 0.968408262454435

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.92	0.94	262
1	0.95	0.98	0.97	1109
2	0.98	0.97	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.96	2469
weighted avg	0.97	0.97	0.97	2469

In [104...

```

import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# 1) Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2) Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3) Encode categorical columns

```

```

df_encoded = pd.get_dummies(df, drop_first=True)

# 4) Define X and y
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 5) Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6) Base model
xgb = XGBClassifier(
    objective='multi:softmax',
    num_class=3,
    random_state=42,
    eval_metric='mlogloss'
)

# 7) Parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [4, 6, 8],
    'learning_rate': [0.05, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# 8) Grid search
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)

# 9) Train grid search
grid_search.fit(X_train, y_train)

# 10) Best model
best_model = grid_search.best_estimator_

# 11) Predict
y_pred = best_model.predict(X_test)

```

```
# 12) Results
print("Best Parameters:")
print(grid_search.best_params_)

print("\nBest Cross-Validation Accuracy:")
print(grid_search.best_score_)

print("\nTest Accuracy:")
print(accuracy_score(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best Parameters:

```
{'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.8}
```

Best Cross-Validation Accuracy:

```
0.9745790978928603
```

Test Accuracy:

```
0.9736735520453625
```

Classification Report:

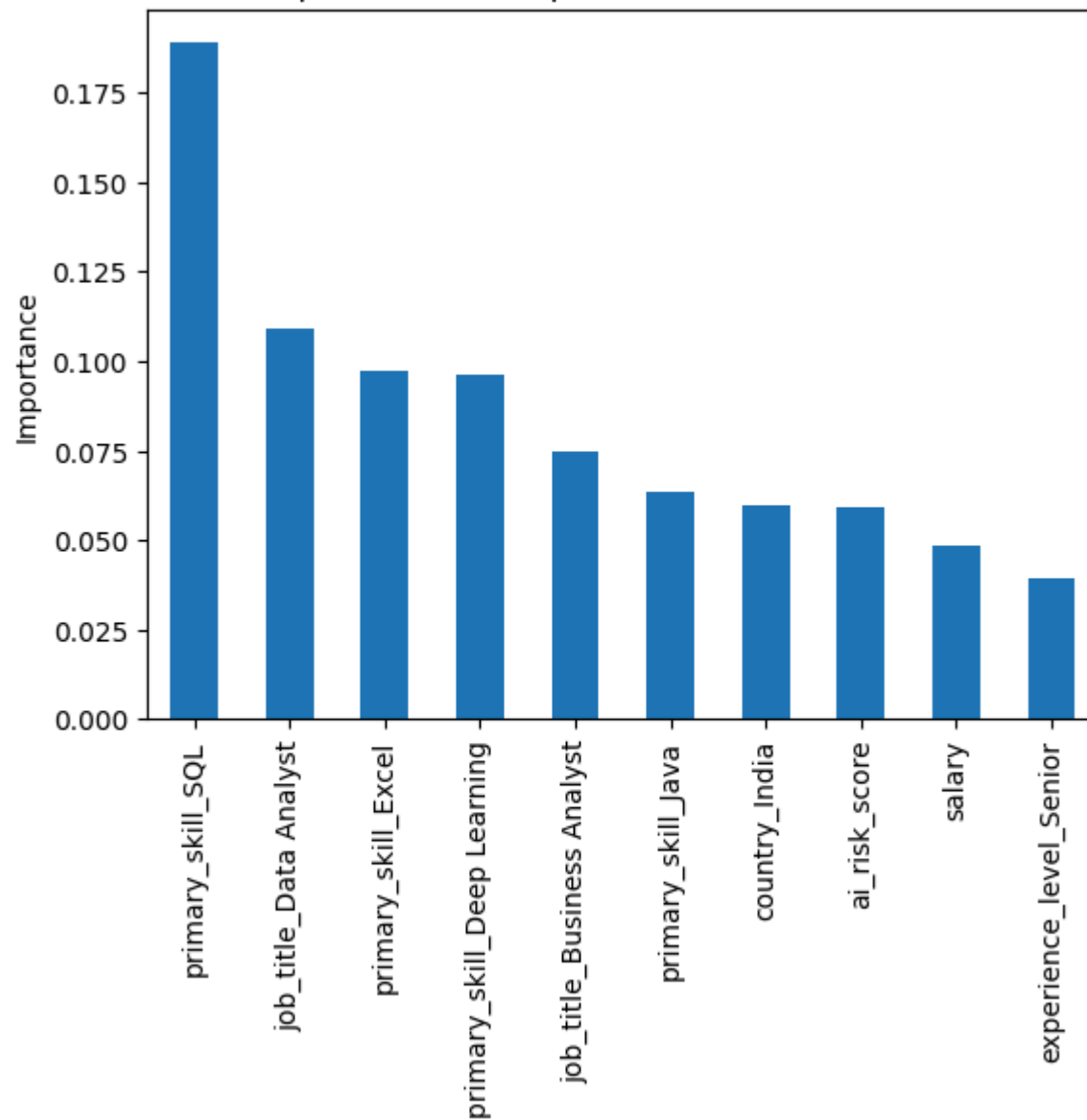
	precision	recall	f1-score	support
0	0.97	0.94	0.95	262
1	0.96	0.98	0.97	1109
2	0.99	0.98	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.97	2469
weighted avg	0.97	0.97	0.97	2469

In [108...

```
import matplotlib.pyplot as plt
import pandas as pd

importance = pd.Series(best_model.feature_importances_, index=X.columns)
importance.sort_values(ascending=False).head(10).plot(kind='bar')
plt.title("Top 10 Feature Importances - Tuned XGBoost")
plt.ylabel("Importance")
plt.show()
```

Top 10 Feature Importances - Tuned XGBoost



In [109... best_model = grid_search.bestbest_model = grid_search.best_estimator__estimator__

AttributeError

Traceback (most recent call last)

Cell In[109], line 1

```
----> 1 best_model = grid_search.bestbest_model = grid_search.best_estimator__estimator__
```

AttributeError: 'GridSearchCV' object has no attribute 'best_estimator__estimator__'

```
In [ ]: print(best_model)
```

```
In [110... plt.figure(figsize=(10,5))

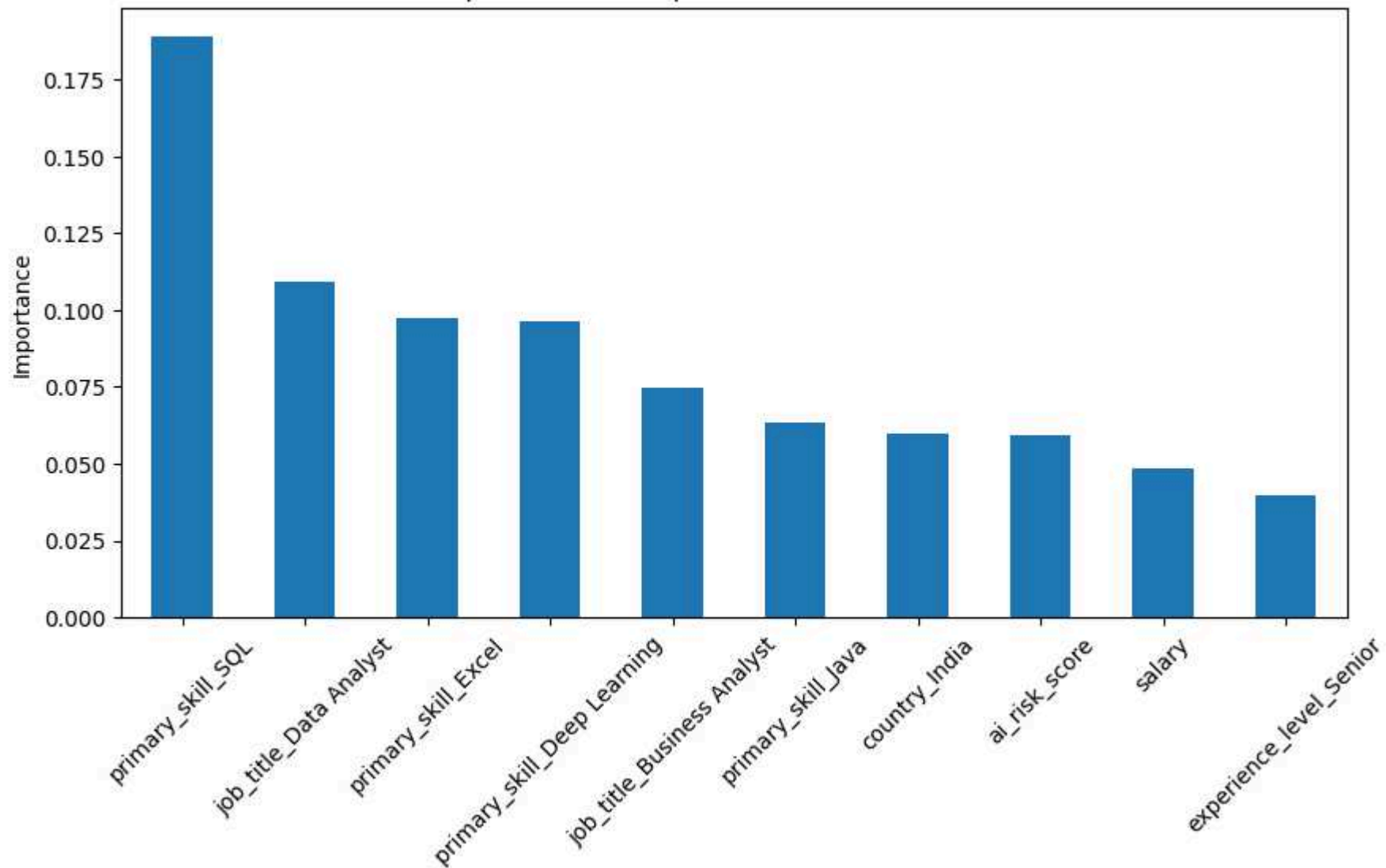
importance = pd.Series(best_model.feature_importances_, index=X.columns)
importance = importance.sort_values(ascending=False).head(10)

importance.plot(kind='bar')

plt.title("Top 10 Feature Importances - Tuned XGBoost")
plt.ylabel("Importance")
plt.xticks(rotation=45)

plt.show()
```

Top 10 Feature Importances - Tuned XGBoost



```
In [111]: print("Before tuning accuracy:", old_accuracy)
          print("After tuning accuracy:", new_accuracy)
```

NameError Traceback (most recent call last)

Cell In[111], line 1

```
----> 1 print("Before tuning accuracy:", old_accuracy)
      2 print("After tuning accuracy:", new_accuracy)
```

NameError: name 'old_accuracy' is not defined

```
In [112... from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

# Default model
model_before = XGBClassifier(random_state=42, eval_metric='mlogloss')
model_before.fit(X_train, y_train)

y_pred_before = model_before.predict(X_test)
old_accuracy = accuracy_score(y_test, y_pred_before)

print("Before tuning accuracy:", old_accuracy)
```

Before tuning accuracy: 0.9688132847306602

```
In [113... # Best model from GridSearch
best_model = grid_search.best_estimator_

y_pred_after = best_model.predict(X_test)
new_accuracy = accuracy_score(y_test, y_pred_after)

print("After tuning accuracy:", new_accuracy)
```

After tuning accuracy: 0.9736735520453625

```
In [114... print("Before tuning accuracy:", old_accuracy)
print("After tuning accuracy:", new_accuracy)
```

Before tuning accuracy: 0.9688132847306602

After tuning accuracy: 0.9736735520453625

```
In [115... import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from xgboost import XGBClassifier

# 1) Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2) Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3) Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4) Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']
```

```

# 5) Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6) Base XGBoost model
xgb = XGBClassifier(
    objective='multi:softmax',
    num_class=3,
    random_state=42,
    eval_metric='mlogloss'
)

# 7) Grid search parameters
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [4, 6, 8],
    'learning_rate': [0.05, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# 8) Grid search
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)

# 9) Train tuned model
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# 10) Predict
y_pred = best_model.predict(X_test)

# 11) Evaluation
print("Best Parameters:", grid_search.best_params_)
print("Best CV Accuracy:", grid_search.best_score_)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

```

```
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
```

Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.8}

Best CV Accuracy: 0.9745790978928603

Test Accuracy: 0.9736735520453625

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.94	0.95	262
1	0.96	0.98	0.97	1109
2	0.99	0.98	0.98	1098
accuracy			0.97	2469
macro avg	0.97	0.96	0.97	2469
weighted avg	0.97	0.97	0.97	2469

Confusion Matrix:

```
[[ 245   17    0]
 [   8 1085   16]
 [   0   24 1074]]
```

In [118...

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 1) Load data
df = pd.read_csv("Future of Jobs AI Dataset.csv")

# 2) Remove Leakage columns
df = df.drop(['salary_bucket', 'ai_risk_category'], axis=1)

# 3) Encode categorical variables
df_encoded = pd.get_dummies(df, drop_first=True)

# 4) Features and target
X = df_encoded.drop('job_survival_class', axis=1)
y = df_encoded['job_survival_class']

# 5) Split data
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 6) Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 7) Logistic Regression with balanced class weights
lr_model = LogisticRegression(
    class_weight='balanced',
    max_iter=1000
)

# 8) Train model
lr_model.fit(X_train_scaled, y_train)

# 9) Predict
y_pred_lr = lr_model.predict(X_test_scaled)

# 10) Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred_lr))
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred_lr))
```

Accuracy: 0.9858242203321183

Classification Report:

	precision	recall	f1-score	support
0	0.93	1.00	0.96	262
1	0.99	0.97	0.98	1109
2	0.99	1.00	0.99	1098
accuracy			0.99	2469
macro avg	0.97	0.99	0.98	2469
weighted avg	0.99	0.99	0.99	2469

Confusion Matrix:

```
[[ 261    1    0]
 [  19 1080   10]
 [   0    5 1093]]
```

In [120...

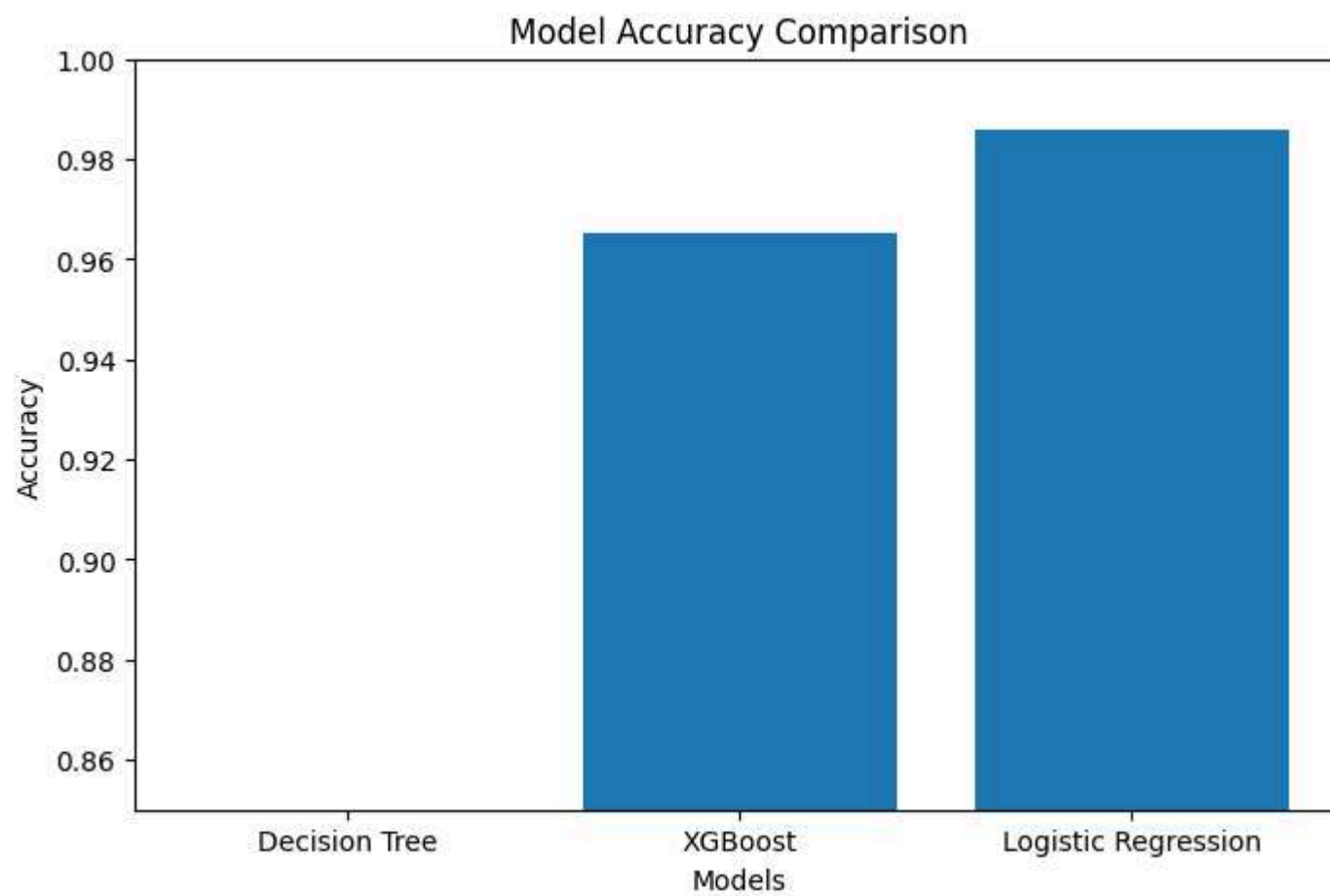
```
import matplotlib.pyplot as plt

models = ['Decision Tree', 'XGBoost', 'Logistic Regression']
accuracies = [dt_acc, xgb_acc, 0.986] # use your real values

plt.figure(figsize=(8,5))
plt.bar(models, accuracies)

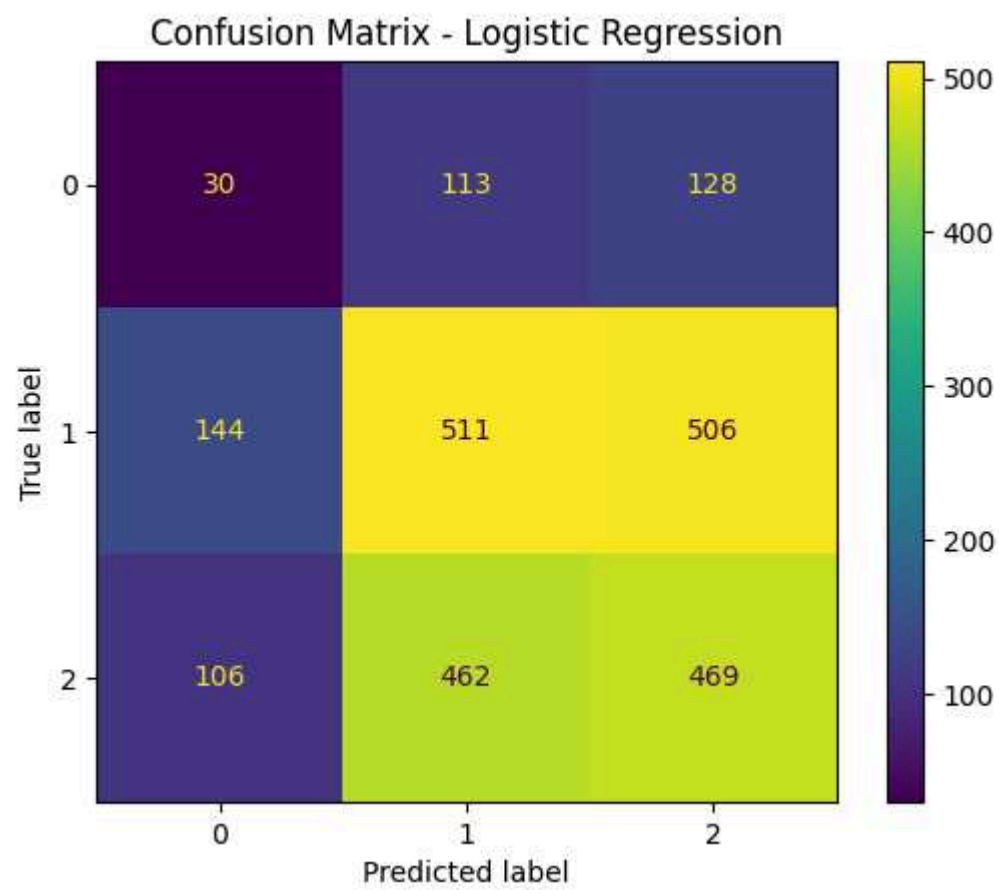
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.xlabel("Models")

plt.ylim(0.85, 1.0)
plt.show()
```



```
In [121... import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

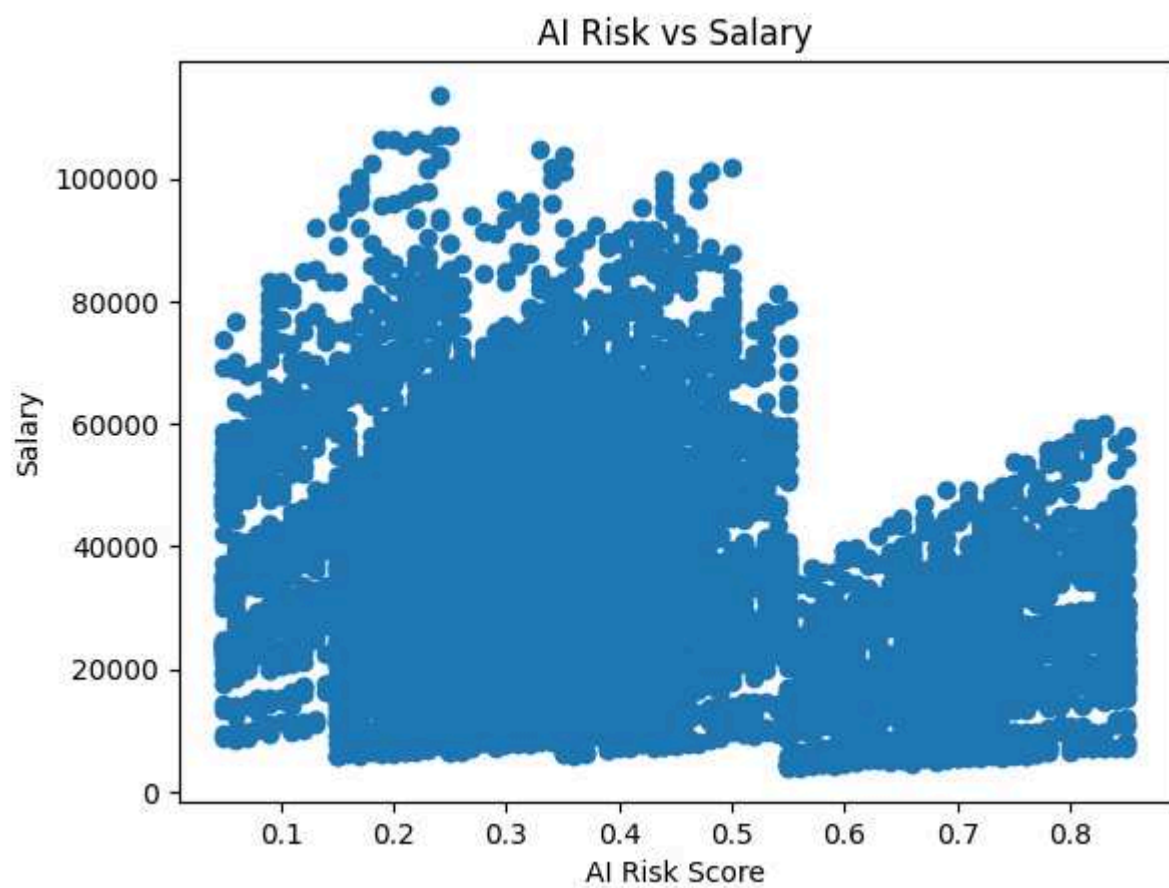
ConfusionMatrixDisplay.from_predictions(y_test, y_pred_lr)
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```



In [122...

```
import matplotlib.pyplot as plt

plt.scatter(df['ai_risk_score'], df['salary'])
plt.title("AI Risk vs Salary")
plt.xlabel("AI Risk Score")
plt.ylabel("Salary")
plt.show()
```

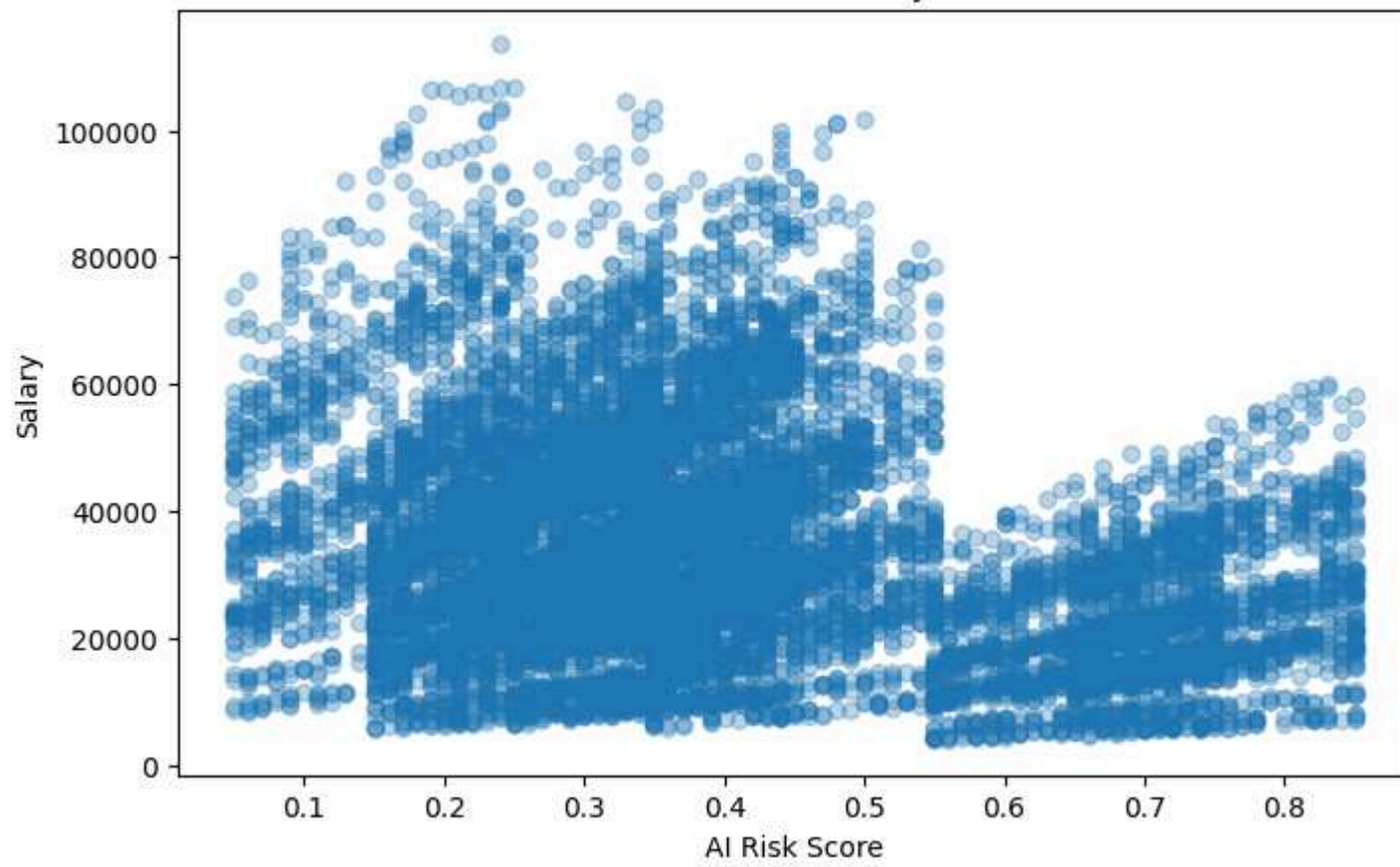


```
In [123... plt.figure(figsize=(8,5))
plt.scatter(df['ai_risk_score'], df['salary'], alpha=0.3)

plt.title("AI Risk vs Salary")
plt.xlabel("AI Risk Score")
plt.ylabel("Salary")

plt.show()
```

AI Risk vs Salary



In []: