

```
import warnings
warnings.filterwarnings("ignore")
# =====
# Core libraries
# =====
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# =====
# Scikit-learn: data handling
# =====
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
# =====
# Scikit-learn: preprocessing & feature selection
# =====
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2, RFE
# =====
# Scikit-learn: models
# =====
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
# =====
# Scikit-learn: evaluation metrics
# =====
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)
# =====
# Imbalanced-learn: sampling methods
# =====
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek
```

```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/STDPER.csv")
df.head()
```

	Student_ID	First_Name	Last_Name	Email	Gender	Age	Department	Attendance (%)	Midterm_Score	Final_Score	...	Projects_Score
0	S1000	Omar	Williams	student0@university.com	Female	22	Mathematics	97.36	40.61	59.61	...	62.8
1	S1001	Maria	Brown	student1@university.com	Male	18	Business	97.71	57.27	74.00	...	98.2
2	S1002	Ahmed	Jones	student2@university.com	Male	24	Engineering	99.52	41.84	63.85	...	91.2
3	S1003	Omar	Williams	student3@university.com	Female	24	Engineering	90.38	45.65	44.44	...	55.4
4	S1004	John	Smith	student4@university.com	Female	23	CS	59.41	53.13	61.77	...	87.4

5 rows × 23 columns

```
df['Grade'].value_counts()
```

count	
Grade	
C	2307
D	1760
B	638
F	279
A	16
dtype: int64	

df.isnull().sum()	
	0
Student_ID	0
First_Name	0
Last_Name	0
Email	0
Gender	0
Age	0
Department	0
Attendance (%)	0
Midterm_Score	0
Final_Score	0
Assignments_Avg	0
Quizzes_Avg	0
Participation_Score	0
Projects_Score	0
Total_Score	0
Grade	0
Study_Hours_per_Week	0
Extracurricular_Activities	0
Internet_Access_at_Home	0
Parent_Education_Level	1025
Family_Income_Level	0
Stress_Level (1-10)	0
Sleep_Hours_per_Night	0
dtype: int64	

```
df.drop(['Student_ID', 'First_Name', 'Last_Name', 'Email'], axis=1, inplace=True)
```

```
df.dtypes
```

	0
Gender	object
Age	int64
Department	object
Attendance (%)	float64
Midterm_Score	float64
Final_Score	float64
Assignments_Avg	float64
Quizzes_Avg	float64
Participation_Score	float64
Projects_Score	float64
Total_Score	float64
Grade	object
Study_Hours_per_Week	float64
Extracurricular_Activities	object
Internet_Access_at_Home	object
Parent_Education_Level	object
Family_Income_Level	object
Stress_Level (1-10)	int64
Sleep_Hours_per_Night	float64

dtype: object

```
df['Parent_Education_Level'].fillna('Unknown', inplace=True)
```

```
df.isnull().sum()
```

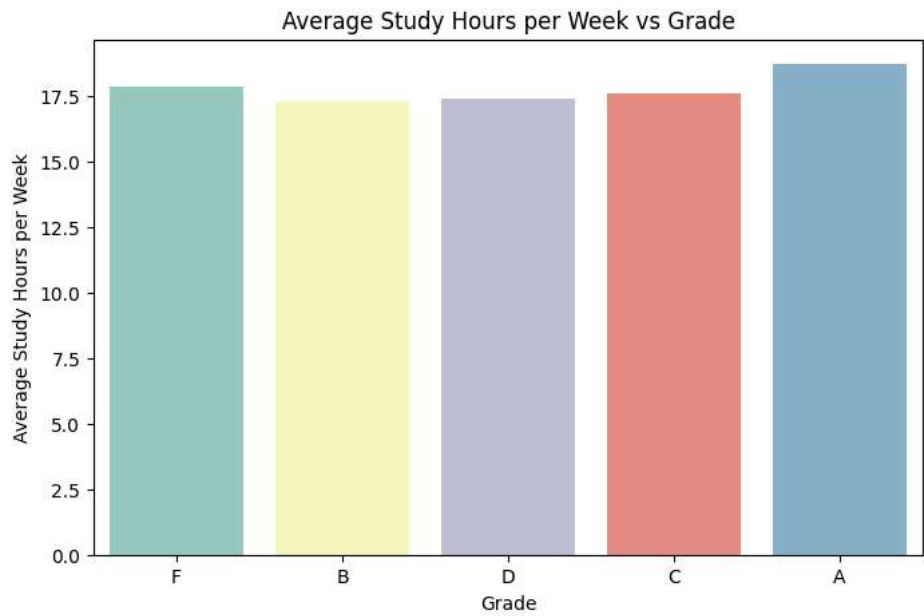
	0
Gender	0
Age	0
Department	0
Attendance (%)	0
Midterm_Score	0
Final_Score	0
Assignments_Avg	0
Quizzes_Avg	0
Participation_Score	0
Projects_Score	0
Total_Score	0
Grade	0
Study_Hours_per_Week	0
Extracurricular_Activities	0
Internet_Access_at_Home	0
Parent_Education_Level	0
Family_Income_Level	0
Stress_Level (1-10)	0
Sleep_Hours_per_Night	0

dtype: int64

```
df.describe()
```

	Age	Attendance (%)	Midterm_Score	Final_Score	Assignments_Avg	Quizzes_Avg	Participation_Score	Projects_Score	Total_Score	Stu
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	
mean	21.048400	75.356076	70.701924	69.546552	74.956320	74.836214	49.963720	74.78305	71.652097	
std	1.989786	14.392716	17.436325	17.108996	14.404287	14.423848	28.989785	14.54243	7.230097	
min	18.000000	50.010000	40.000000	40.010000	50.000000	50.000000	0.000000	50.00000	50.602000	
25%	19.000000	62.945000	55.707500	54.697500	62.340000	62.357500	25.075000	61.97000	66.533875	
50%	21.000000	75.670000	70.860000	69.485000	75.090000	74.905000	49.600000	74.54000	71.696250	
75%	23.000000	87.862500	85.760000	83.922500	87.352500	87.292500	75.500000	87.63000	76.711625	
max	24.000000	100.000000	99.990000	99.980000	99.990000	99.990000	100.000000	100.00000	95.091500	

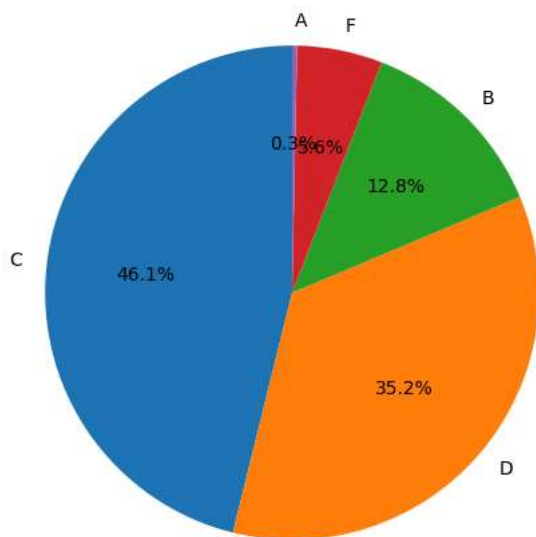
```
plt.figure(figsize=(8,5))
sns.barplot(data=df,
            x='Grade',
            y='Study_Hours_per_Week',
            estimator='mean', # default, but explicit is better
            palette='Set3',
            errorbar=None # removes confidence intervals (optional)
            )
plt.title("Average Study Hours per Week vs Grade")
plt.ylabel("Average Study Hours per Week")
plt.show()
```



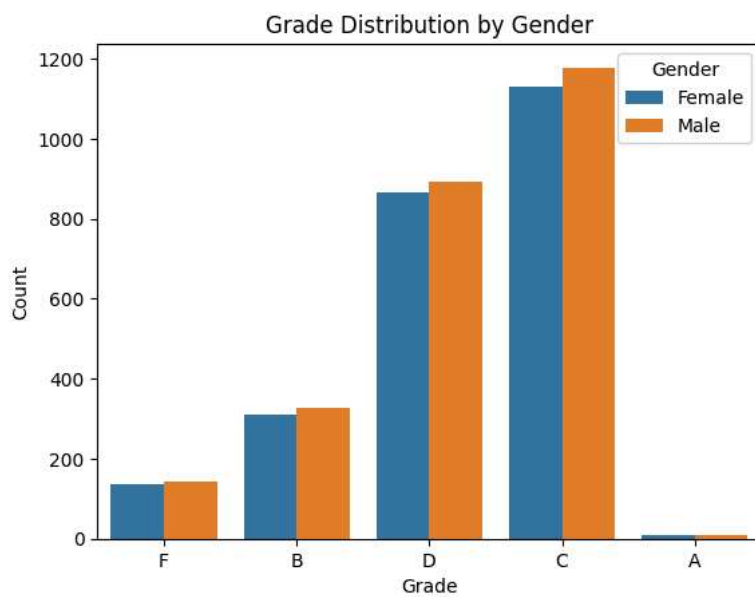
```
grade_counts = df['Grade'].value_counts()
```

```
plt.figure(figsize=(6,6))
plt.pie(
    grade_counts,
    labels=grade_counts.index,
    autopct='%1.1f%%',
    startangle=90
)
plt.title("Distribution of Students by Grade")
plt.show()
```

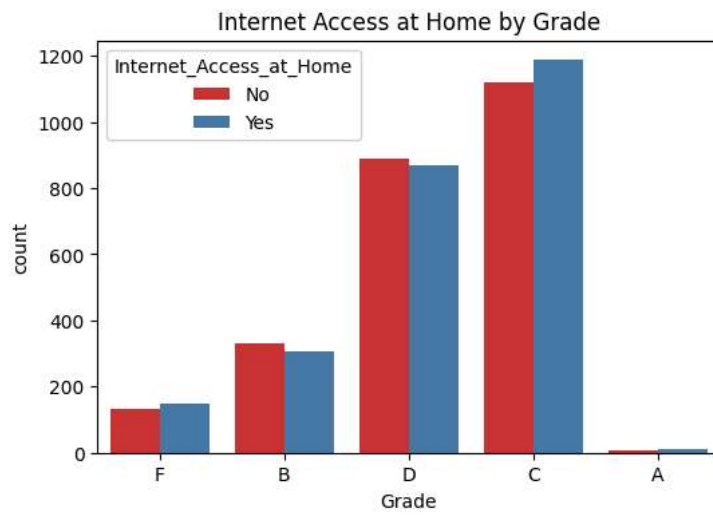
Distribution of Students by Grade



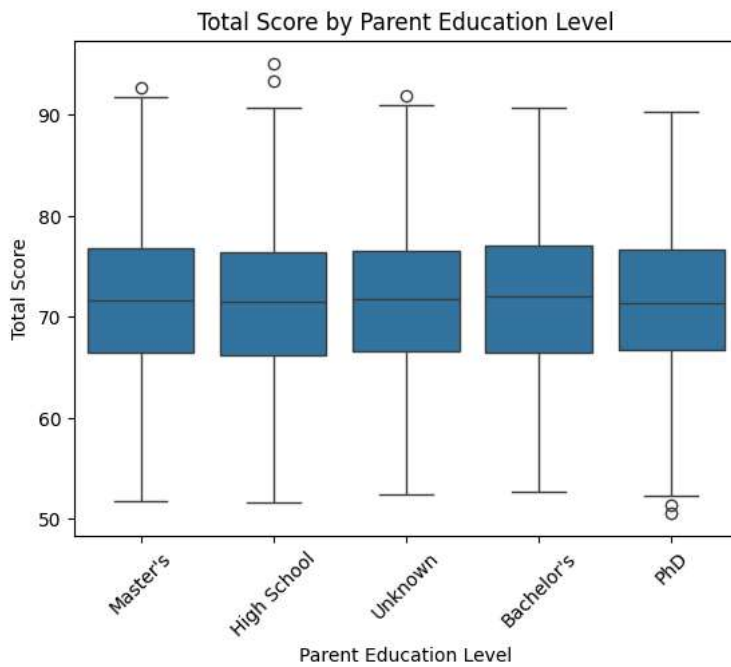
```
sns.countplot(x='Grade', hue='Gender', data=df)
plt.title('Grade Distribution by Gender')
plt.xlabel('Grade')
plt.ylabel('Count')
plt.show()
```



```
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='Grade', hue='Internet_Access_at_Home',
palette='Set1')
plt.title("Internet Access at Home by Grade")
plt.show()
```

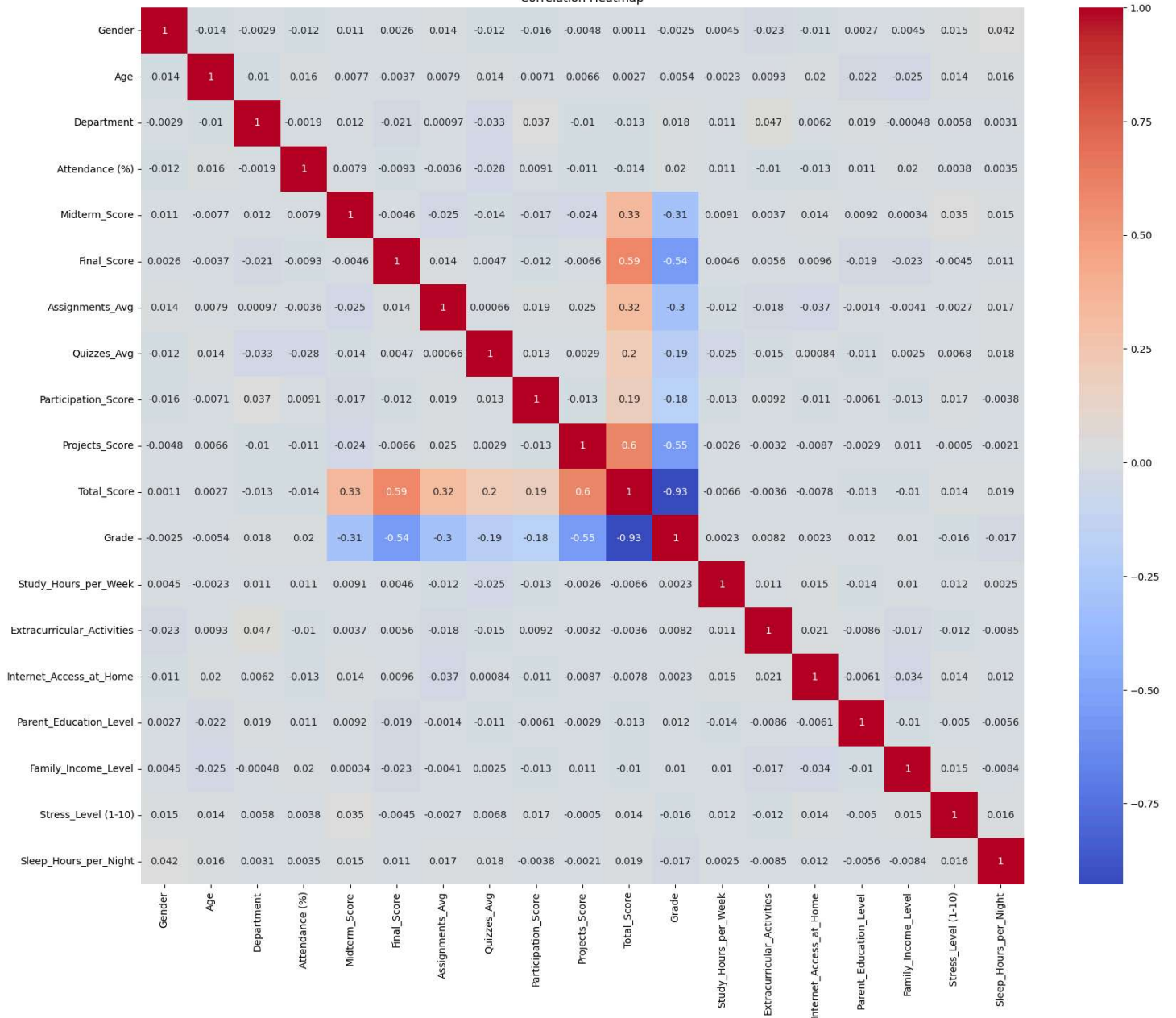


```
sns.boxplot(x='Parent_Education_Level', y='Total_Score', data=df)
plt.title('Total Score by Parent Education Level')
plt.xticks(rotation=45)
plt.xlabel('Parent Education Level')
plt.ylabel('Total Score')
plt.show()
```



```
from sklearn.preprocessing import LabelEncoder
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

Correlation Heatmap

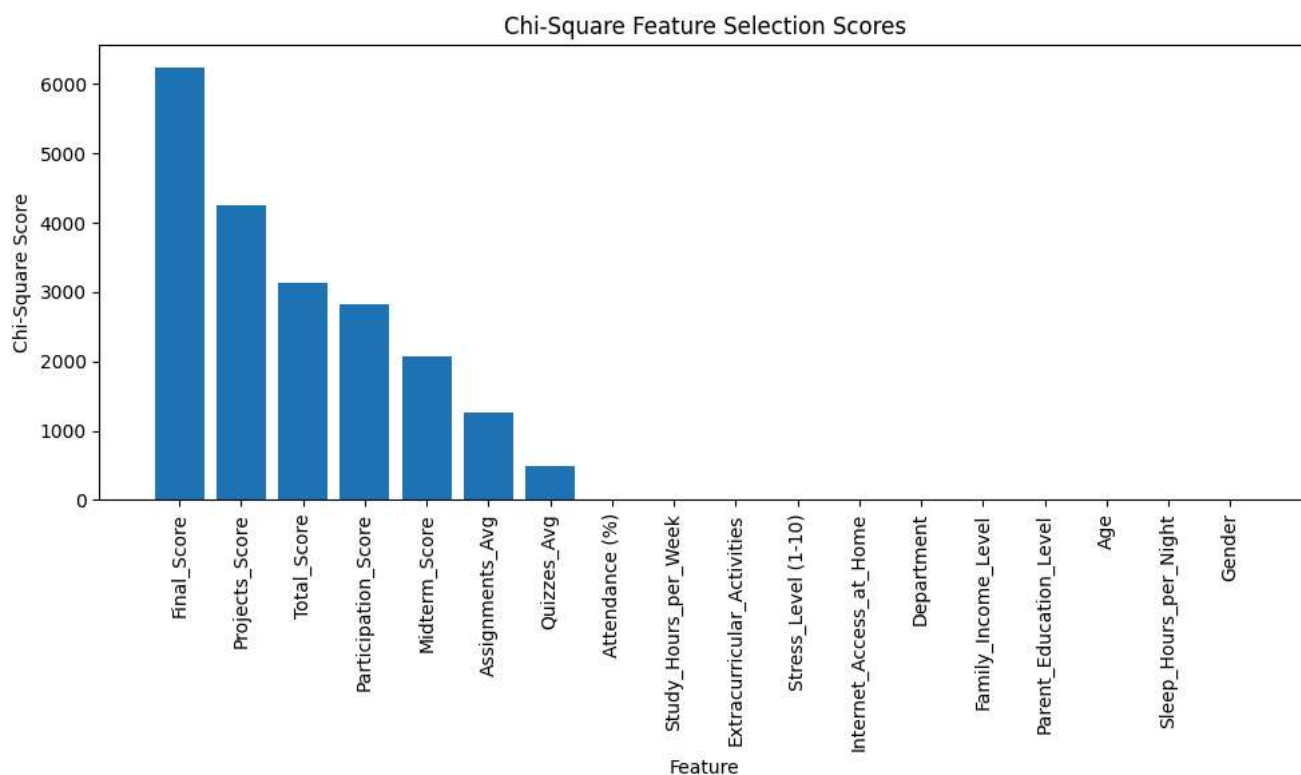


```
X = df.drop(['Grade'], axis=1)
y = df['Grade']
```

```

from sklearn.feature_selection import SelectKBest, chi2
# Create selector (choose number of features you want, e.g., k=5)
selector = SelectKBest(score_func=chi2, k=10)
# Fit and transform
selector.fit(X, y)
# Create DataFrame of features and chi-square scores
data = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
# Sort by score (optional, but makes plot clearer)
data = data.sort_values(by='Score', ascending=False)
# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("Chi-Square Score")
plt.title("Chi-Square Feature Selection Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```



```

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42
)

```

```

k = 10
selector = SelectKBest(score_func=chi2, k=k)

```

```

X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))

```

```

Selected Features: ['Attendance (%)', 'Midterm_Score', 'Final_Score', 'Assignments_Avg', 'Quizzes_Avg', 'Participation_Score', 'Projects_Score']

```

```

models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf', random_state=42)
}
metrics = []

```

```

for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)
    print(f"\n==== Classification Report: {name} =====")
    print(classification_report(y_test, y_pred))

```

```
metrics.append({
    "Model": name,
    "Accuracy": accuracy_score(y_test, y_pred),
    "Precision": precision_score(y_test, y_pred,
average='weighted'),
    "Recall": recall_score(y_test, y_pred, average='weighted')
})
```

==== Classification Report: Random Forest ====

	precision	recall	f1-score	support
0	1.00	0.83	0.91	6
1	1.00	1.00	1.00	208
2	1.00	1.00	1.00	673
3	1.00	1.00	1.00	527
4	1.00	0.99	0.99	86
accuracy			1.00	1500
macro avg	1.00	0.96	0.98	1500
weighted avg	1.00	1.00	1.00	1500

==== Classification Report: Decision Tree ====

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	208
2	1.00	1.00	1.00	673
3	1.00	1.00	1.00	527
4	1.00	1.00	1.00	86
accuracy			1.00	1500
macro avg	1.00	1.00	1.00	1500
weighted avg	1.00	1.00	1.00	1500

==== Classification Report: SVM ====

	precision	recall	f1-score	support
0	0.00	0.00	0.00	6
1	0.96	0.91	0.94	208
2	0.95	0.99	0.97	673
3	0.93	0.97	0.95	527
4	1.00	0.63	0.77	86
accuracy			0.95	1500
macro avg	0.77	0.70	0.73	1500
weighted avg	0.94	0.95	0.94	1500

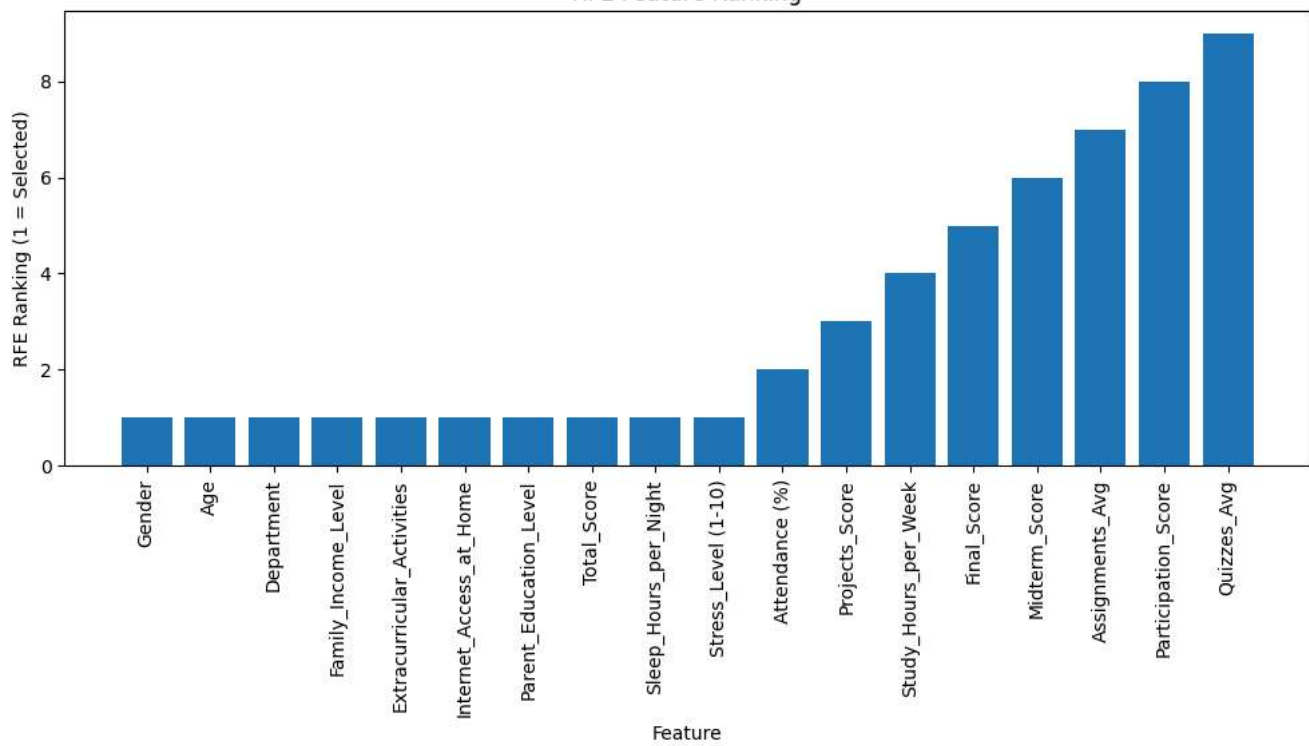
```
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)
```

Model Performance:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.998667	0.998671	0.998667
1	Decision Tree	1.000000	1.000000	1.000000
2	SVM	0.947333	0.944993	0.947333

```
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X, y)
feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
})
feature_ranking = feature_ranking.sort_values(by='Ranking',
ascending=True)
plt.figure(figsize=(10, 6))
plt.bar(feature_ranking['Feature'], feature_ranking['Ranking'])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

RFE Feature Ranking



```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42
)
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X, y)
X_train_k = rfe.fit_transform(X_train, y_train)
X_test_k = rfe.transform(X_test)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))
```

Selected Features: ['Attendance (%)', 'Midterm_Score', 'Final_Score', 'Assignments_Avg', 'Quizzes_Avg', 'Participation_Score', 'Projects_Score']

```
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf', random_state=42)
}
metrics = []
```

```
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)
    print(f"\n==== Classification Report: {name} =====")
    print(classification_report(y_test, y_pred))
    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred,
    average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted')
    })
```

```
==== Classification Report: Random Forest =====
              precision    recall  f1-score   support

     0           0.00       0.00       0.00         6
     1           0.97       1.00       0.99        208
     2           1.00       1.00       1.00        673
     3           1.00       1.00       1.00        527
     4           1.00       0.99       0.99         86

 accuracy          0.99
 macro avg          0.79
 weighted avg       0.99
```

```

===== Classification Report: Decision Tree =====
              precision    recall  f1-score   support

    0         1.00        1.00        1.00         6
    1         1.00        1.00        1.00        208
    2         1.00        1.00        1.00        673
    3         1.00        1.00        1.00        527
    4         1.00        1.00        1.00         86

 accuracy          1.00          1.00          1.00      1500
 macro avg         1.00          1.00          1.00      1500
 weighted avg      1.00          1.00          1.00      1500

```

```

===== Classification Report: SVM =====
              precision    recall  f1-score   support

    0         0.00        0.00        0.00         6
    1         0.97        0.81        0.88        208
    2         0.92        1.00        0.96        673
    3         0.91        0.97        0.94        527
    4         1.00        0.40        0.57         86

 accuracy          0.92          0.92          0.92      1500
 macro avg         0.76          0.63          0.67      1500
 weighted avg      0.92          0.92          0.91      1500

```

```

results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)
list(selected_features)

```

```

Model Performance:
   Model  Accuracy  Precision  Recall
0  Random Forest  0.995333  0.991447  0.995333
1  Decision Tree  1.000000  1.000000  1.000000
2         SVM      0.924000  0.924303  0.924000
['Attendance (%)',
 'Midterm_Score',
 'Final_Score',
 'Assignments_Avg',
 'Quizzes_Avg',
 'Participation_Score',
 'Projects_Score',
 'Total_Score',
 'Extracurricular_Activities',
 'Internet_Access_at_Home']

```

```

rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)

```

```

▼ RandomForestClassifier ⓘ ?
RandomForestClassifier(random_state=42)

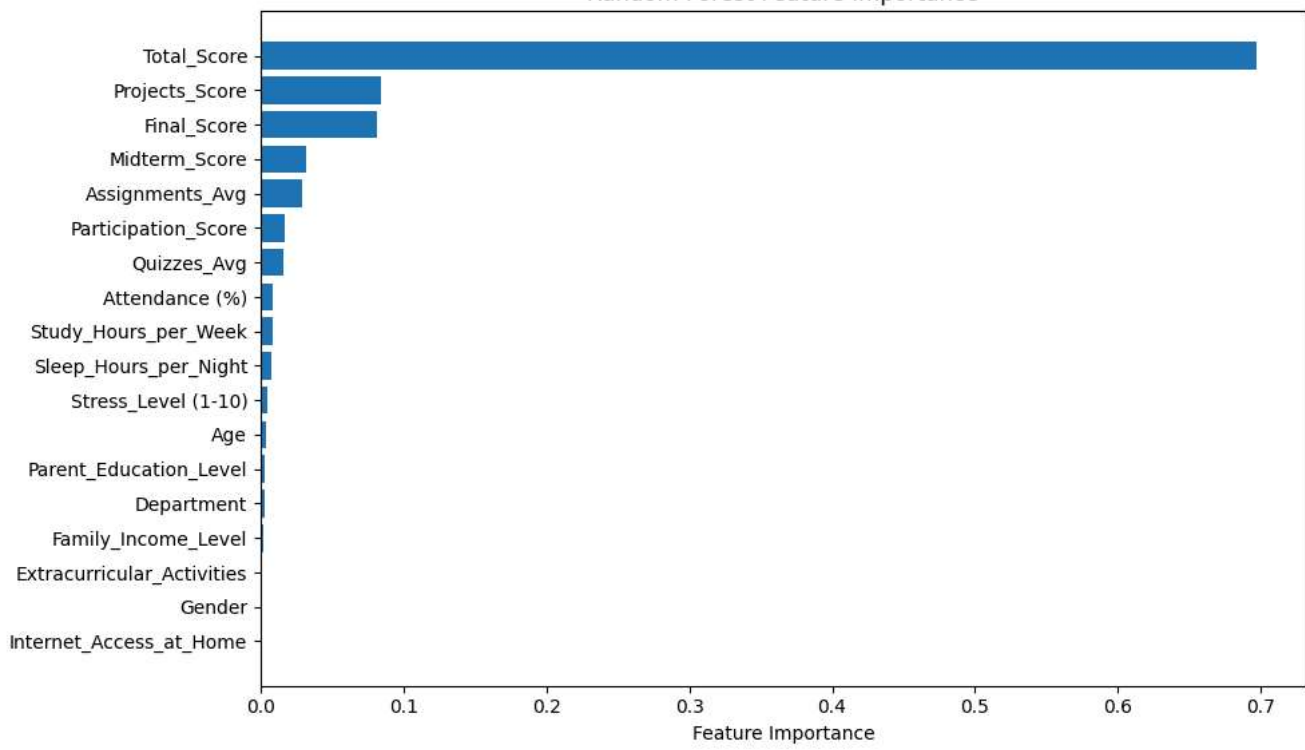
```

```

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)
# Extract importances
importances = rf.feature_importances_
# Use feature names from X (correct and safe)
feature_names = X.columns if hasattr(X, "columns") else [f"Feature{i}" for i in range(X.shape[1])]
# Create DataFrame
df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})
# Sort by importance
df_importance = df_importance.sort_values(by='Importance',
ascending=False)
# Plot
plt.figure(figsize=(10, 6))
plt.barh(df_importance['Feature'][::-1], df_importance['Importance']
[:::-1])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()

```

Random Forest Feature Importance



```
def evaluate_models(X_train, y_train, X_test, y_test, models, dataset_name):
    all_model_results = []
    print(f"\n===== {dataset_name} =====")
    for name, model in models.items():
        print(f"\n--- {name} ---")

        model.fit(X_train, y_train)
        preds = model.predict(X_test)

        acc = accuracy_score(y_test, preds)
        prec = precision_score(y_test, preds, average='weighted', zero_division=0)
        rec = recall_score(y_test, preds, average='weighted', zero_division=0)
        f1 = f1_score(y_test, preds, average='weighted', zero_division=0)

        print(classification_report(y_test, preds))

        all_model_results.append({
            'Sampling': dataset_name,
            'Model': name,
            'Accuracy': acc,
            'Precision': prec,
            'Recall': rec,
            'F1-score': f1
        })
    return all_model_results
```

```
all_results = []
all_results += evaluate_models(
    X_train, y_train, X_test, y_test,
    models, # Use the defined 'models' dictionary
    "Original Imbalanced Data"
)
```

===== Original Imbalanced Data =====

--- SVM ---

	precision	recall	f1-score	support
0	0.00	0.00	0.00	6
1	0.94	0.88	0.91	208
2	0.93	0.97	0.95	673
3	0.92	0.95	0.94	527
4	0.95	0.69	0.80	86
accuracy			0.93	1500
macro avg	0.75	0.70	0.72	1500
weighted avg	0.93	0.93	0.93	1500

```

--- Random Forest ---
      precision    recall  f1-score   support

     0       1.00      0.17      0.29         6
     1       0.98      1.00      0.99       208
     2       1.00      1.00      1.00      673
     3       1.00      1.00      1.00     527
     4       1.00      0.99      0.99        86

 accuracy          1.00          1500
 macro avg         0.99          0.83          0.85          1500
 weighted avg      1.00          1.00          0.99          1500

```

```

--- Decision Tree ---
      precision    recall  f1-score   support

     0       1.00      1.00      1.00         6
     1       1.00      1.00      1.00       208
     2       1.00      1.00      1.00      673
     3       1.00      1.00      1.00     527
     4       1.00      1.00      1.00        86

 accuracy          1.00          1500
 macro avg         1.00          1.00          1.00          1500
 weighted avg      1.00          1.00          1.00          1500

```

```

all_results += evaluate_models(
    X_train, y_train, X_test, y_test,
    models,
    "Original Imbalanced Data"
)

```

===== Original Imbalanced Data =====

```

--- SVM ---
      precision    recall  f1-score   support

     0       0.00      0.00      0.00         6
     1       0.94      0.88      0.91       208
     2       0.93      0.97      0.95      673
     3       0.92      0.95      0.94     527
     4       0.95      0.69      0.80        86

 accuracy          0.93          1500
 macro avg         0.75          0.70          0.72          1500
 weighted avg      0.93          0.93          0.93          1500

```

```

--- Random Forest ---
      precision    recall  f1-score   support

     0       1.00      0.17      0.29         6
     1       0.98      1.00      0.99       208
     2       1.00      1.00      1.00      673
     3       1.00      1.00      1.00     527
     4       1.00      0.99      0.99        86

 accuracy          1.00          1500
 macro avg         0.99          0.83          0.85          1500
 weighted avg      1.00          1.00          0.99          1500

```

```

--- Decision Tree ---
      precision    recall  f1-score   support

     0       1.00      1.00      1.00         6
     1       1.00      1.00      1.00       208
     2       1.00      1.00      1.00      673
     3       1.00      1.00      1.00     527
     4       1.00      1.00      1.00        86

 accuracy          1.00          1500
 macro avg         1.00          1.00          1.00          1500
 weighted avg      1.00          1.00          1.00          1500

```

Start coding or [generate](#) with AI.

Visualizing Model Performance

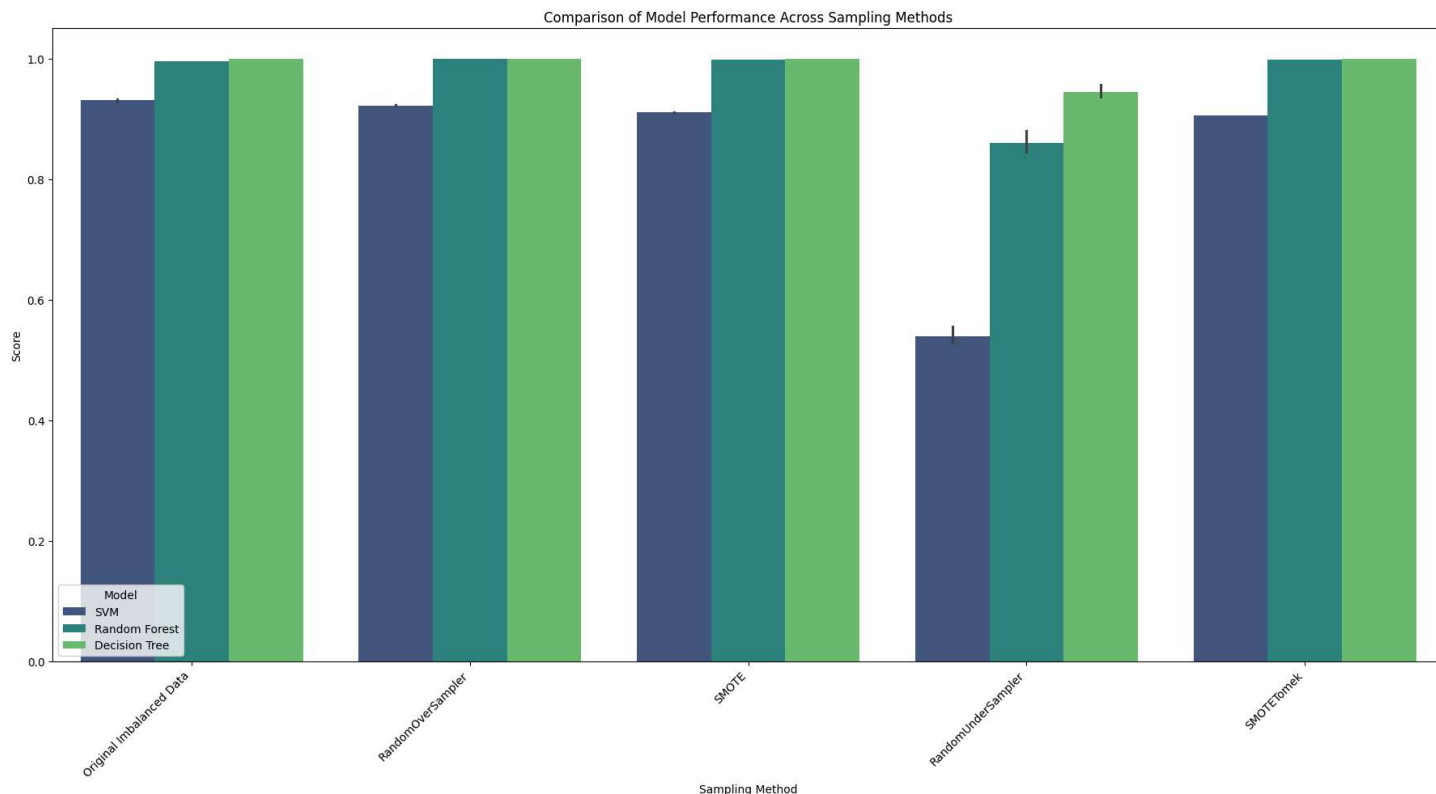
```

import matplotlib.pyplot as plt
import seaborn as sns

# Melt the DataFrame to prepare for plotting
results_melted = results_df_final.melt(id_vars=['Sampling', 'Model'],
                                       value_vars=['Accuracy', 'Precision', 'Recall', 'F1-score'],
                                       var_name='Metric',
                                       value_name='Score')

plt.figure(figsize=(18, 10))
sns.barplot(x='Sampling', y='Score', hue='Model', data=results_melted, palette='viridis')
plt.title('Comparison of Model Performance Across Sampling Methods')
plt.ylabel('Score')
plt.xlabel('Sampling Method')
plt.xticks(rotation=45, ha='right')
plt.ylim(0, 1.05) # Ensure scores are within a reasonable range for visualization
plt.legend(title='Model')
plt.tight_layout()
plt.show()

```



The above plot shows the Accuracy, Precision, Recall, and F1-score combined. To get a clearer view of each metric individually, we can create separate plots.

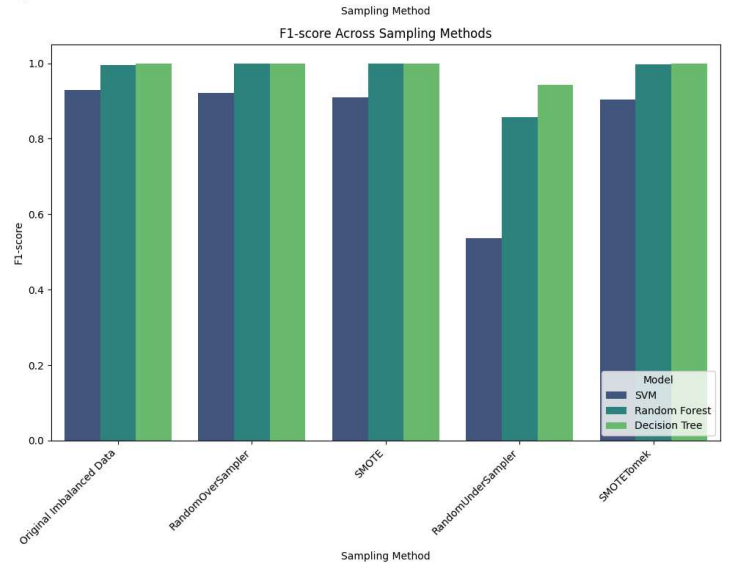
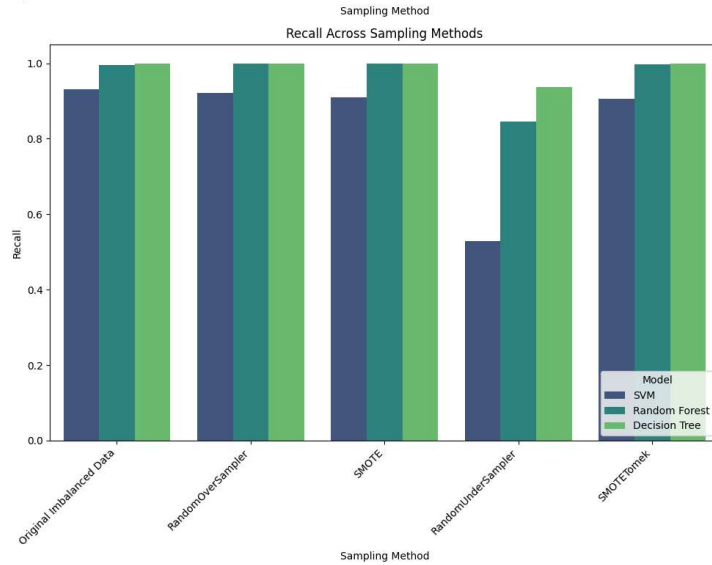
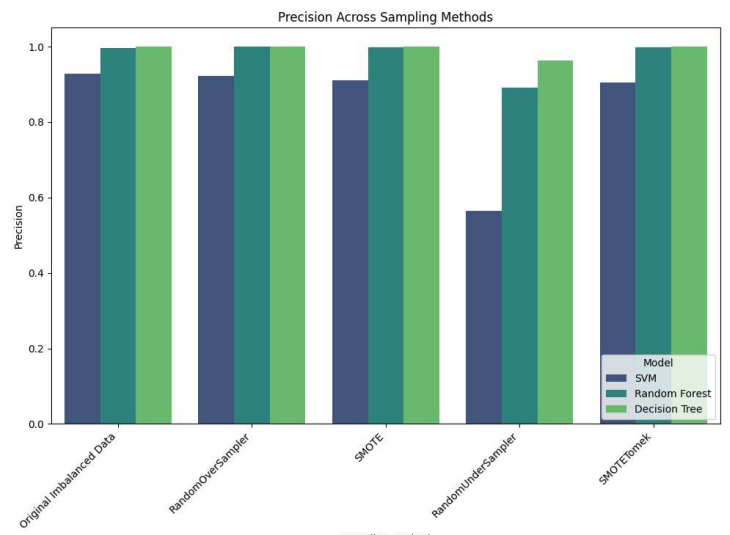
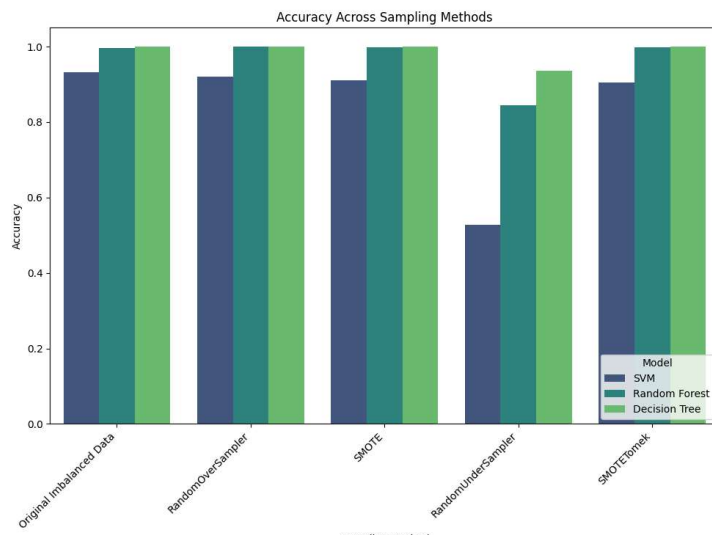
```

metrics_to_plot = ['Accuracy', 'Precision', 'Recall', 'F1-score']

plt.figure(figsize=(20, 15))
for i, metric in enumerate(metrics_to_plot):

```

```
plt.subplot(2, 2, i + 1) # Create a 2x2 grid of plots
sns.barplot(x='Sampling', y='Score', hue='Model', data=results_melted[results_melted['Metric'] == metric], palette='viridis')
plt.title(f'{metric} Across Sampling Methods')
plt.ylabel(metric)
plt.xlabel('Sampling Method')
plt.xticks(rotation=45, ha='right')
plt.ylim(0, 1.05)
plt.legend(title='Model', loc='lower right')
plt.tight_layout()
plt.show()
```



```
sampling_methods = {
    "RandomOverSampler": RandomOverSampler(random_state=42),
    "SMOTE": SMOTE(random_state=42),
    "RandomUnderSampler": RandomUnderSampler(random_state=42),
    "SMOTETomek": SMOTETomek(random_state=42)
}
```

```
for method_name, sampler in sampling_methods.items():
```

```

print(f"\nApplying {method_name}...")
X_resampled, y_resampled = sampler.fit_resample(X_train, y_train)

all_results += evaluate_models(
    X_resampled, y_resampled, X_test, y_test,
    models,
    method_name
)

results_df_final = pd.DataFrame(all_results)
print("\n\n--- All Model Performance Results ---")
print(results_df_final)

```

Applying RandomOverSampler...

===== RandomOverSampler =====

--- SVM ---

	precision	recall	f1-score	support
0	1.00	0.33	0.50	6
1	0.86	0.94	0.90	208
2	0.95	0.93	0.94	673
3	0.93	0.93	0.93	527
4	0.80	0.83	0.81	86
accuracy			0.92	1500
macro avg	0.91	0.79	0.82	1500
weighted avg	0.92	0.92	0.92	1500

--- Random Forest ---

	precision	recall	f1-score	support
0	1.00	0.83	0.91	6
1	1.00	1.00	1.00	208
2	1.00	1.00	1.00	673
3	1.00	1.00	1.00	527
4	1.00	1.00	1.00	86
accuracy			1.00	1500
macro avg	1.00	0.97	0.98	1500
weighted avg	1.00	1.00	1.00	1500

--- Decision Tree ---

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	208
2	1.00	1.00	1.00	673
3	1.00	1.00	1.00	527
4	1.00	1.00	1.00	86
accuracy			1.00	1500
macro avg	1.00	1.00	1.00	1500
weighted avg	1.00	1.00	1.00	1500

Applying SMOTE...

===== SMOTE =====

--- SVM ---

	precision	recall	f1-score	support
0	1.00	0.33	0.50	6
1	0.88	0.88	0.88	208
2	0.93	0.93	0.93	673
3	0.92	0.92	0.92	527

```

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

```

```

k = 18 # number of top important features to keep
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
# Get importance scores
importances = rf.feature_importances_
# Sort indices by importance (descending)
sorted_idx = np.argsort(importances)[::-1]
# Select top-k feature indices
top_k_idx = sorted_idx[:k]
# Get feature names

```

```
selected_features = X.columns[top_k_idx]
print(f"Selected Features by RF Importance: {list(selected_features)}")
```

```
Selected Features by RF Importance: ['Total_Score', 'Projects_Score', 'Final_Score', 'Midterm_Score', 'Assignments_Avg', 'Participation_Score',
```

```
X_train_k = X_train[selected_features]
X_test_k = X_test[selected_features]
# -----
# 3. MODELS TO COMPARE
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = []
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)
    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred,
average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted')
    })

results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RF-Selected Features:")
print(results_df)
```

Model Performance Using RF-Selected Features:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.995	0.990180	0.995
1	Decision Tree	1.000	1.000000	1.000
2	SVM	0.946	0.943325	0.946

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

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# =====
# Scikit-learn Models
# =====
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import (
    RandomForestRegressor,
    GradientBoostingRegressor
)
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.multioutput import MultiOutputRegressor
# =====
# Scikit-learn Utilities
# =====
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score
)
# =====
# XGBoost
# =====
from xgboost import XGBRegressor
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/IoTpond1.csv")
df.head()

```

```
df.isnull().sum()
```

```

df.drop(['created_at'], axis=1, inplace=True)
plt.figure(figsize=(8, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

```

```

plt.figure(figsize=(8, 6))
sns.histplot(df['Ammonia(g/ml)'], bins=15, kde=True, color='blue')
plt.title('Ammonia Distribution')
plt.xlabel('Ammonia')
plt.ylabel('Frequency')
plt.show()

```

```

def detect_outliers_iqr(df, columns, k=1.5):
    outliers = pd.DataFrame(False, index=df.index, columns=columns)
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - k * IQR
        upper = Q3 + k * IQR
        outliers[col] = (df[col] < lower) | (df[col] > upper)
    return outliers
# Example
outliers = detect_outliers_iqr(df, ["Ammonia(g/ml)"])
df[outliers.any(axis=1)]

```

```

df.drop(['Ammonia(g/ml)'], axis=1, inplace=True)
df.fillna(df.mean(), inplace=True)
df.isnull().sum()

```

```

X= df.drop(['Fish_Length(cm)', 'Fish_Weight(g)'], axis=1)
y = df[['Fish_Length(cm)', 'Fish_Weight(g)']]
# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

```

rfr = RandomForestRegressor(n_estimators=200, random_state=42)
multi_rfr = MultiOutputRegressor(rfr)
# Train
multi_rfr.fit(X_train, y_train)
# Predict
y_pred = multi_rfr.predict(X_test)
# Evaluate
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]
# Display results
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")

```

```

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (RFR)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()

```

```

dtr = DecisionTreeRegressor(random_state=42)
multi_dtr = MultiOutputRegressor(dtr)
# Train model
multi_dtr.fit(X_train, y_train)
# Predict
y_pred = multi_dtr.predict(X_test)
# Evaluate performance
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]
# Display metrics
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")
# ----- SCATTER PLOTS -----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (DTR)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()

```

```

from sklearn.svm import SVR
from sklearn.multioutput import MultiOutputRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import pandas as pd

# Load the dataset and perform preprocessing steps for df
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/IoTpond1.csv")
df.drop(['created_at'], axis=1, inplace=True)
df.drop(['Ammonia(g/ml)'], axis=1, inplace=True)
df.fillna(df.mean(), inplace=True)

```

```
# Re-defining X, y, X_train, y_train, X_test, y_test to ensure they are available
X = df.drop(['Fish_Length(cm)', 'Fish_Weight(g)'], axis=1)
y = df[['Fish_Length(cm)', 'Fish_Weight(g)']]
# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svr = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
multi_svr = MultiOutputRegressor(svr)
# Train model
multi_svr.fit(X_train, y_train)
# Predict
y_pred = multi_svr.predict(X_test)
# Evaluate
mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y.shape[1])]
# Display metrics
for i, col in enumerate(y.columns):
    print(f"\nTarget: {col}")
    print(f"MAE: {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R²: {r2[i]:.3f}")
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, col in enumerate(y.columns):
    axes[i].scatter(y_test.iloc[:, i], y_pred[:, i], alpha=0.7,
edgecolors='k')
    axes[i].plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
[y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
'r--', lw=2)
    axes[i].set_xlabel('Actual Values')
    axes[i].set_ylabel('Predicted Values')
    axes[i].set_title(f"{col} - Actual vs Predicted (SVR)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```

```
xgb = XGBRegressor(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="reg:squarederror",
    random_state=42,
    n_jobs=-1
```

```
model = MultiOutputRegressor(xgb)
# =====
# Train
# =====
model.fit(X_train, y_train)
# =====
# Predict
# =====
y_pred = model.predict(X_test)
# =====
# Evaluation
# =====
mae = mean_absolute_error(y_test, y_pred, multioutput="raw_values")
rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput="raw_values"))
r2 = [r2_score(y_test.iloc[:, i], y_pred[:, i]) for i in
range(y_test.shape[1])]
# =====
# Display Metrics
# =====
for i, col in enumerate(y_test.columns):
    print(f"\nTarget: {col}")
    print(f"MAE : {mae[i]:.3f}")
    print(f"RMSE: {rmse[i]:.3f}")
    print(f"R² : {r2[i]:.3f}")
```

```
fig, axes = plt.subplots(1, y_test.shape[1], figsize=(6 *
y_test.shape[1], 5))
if y_test.shape[1] == 1:
    axes = [axes]
for i, col in enumerate(y_test.columns):
    axes[i].scatter(
        y_test.iloc[:, i],
        y_pred[:, i],
        alpha=0.7,
        edgecolors='k'
    )
    axes[i].plot(
        [y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
        [y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
        'r--', lw=2
    )
    axes[i].set_xlabel("Actual Values")
    axes[i].set_ylabel("Predicted Values")
    axes[i].set_title(f"{col} - Actual vs Predicted (XGBoost)")
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```