

```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder

from sklearn.feature_selection import SelectKBest, chi2, RFE

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

from sklearn.metrics import ( confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report)

from imblearn.over_sampling import RandomOverSampler, SMOTE

from imblearn.under_sampling import RandomUnderSampler

from imblearn.combine import SMOTETomek

import warnings

warnings.filterwarnings("ignore")

df = pd.read_csv('/content/drive/MyDrive/Project 3/Students Performance Dataset.csv')
```

df.head()

	Student_ID	First_Name	Last_Name	Email	Gender	Age	Department	Attendance (%)	Midterm_Score	Final_Score	...	Projects_Score	Total_Score	Grade	Study_Hours_per_Week	Extracurricular_Activities	Internet_Access_at_Home	Parent_Education_Level	Family_Income_Level	Stress_Level (1-10)	Sleep_Hours_per_Night
0	S1000	Omar	Williams	student0@university.com	Female	22	Mathematics	97.36	40.61	59.61	...	62.84	59.8865	F	10.3	Yes	No	Master's	Medium	1	5.9
1	S1001	Maria	Brown	student1@university.com	Male	18	Business	97.71	57.27	74.00	...	98.23	81.9170	B	27.1	No	No	High School	Low	4	4.3
2	S1002	Ahmed	Jones	student2@university.com	Male	24	Engineering	99.52	41.84	63.85	...	91.22	67.7170	D	12.4	Yes	No	High School	Low	9	6.1
3	S1003	Omar	Williams	student3@university.com	Female	24	Engineering	90.38	45.65	44.44	...	55.48	51.6535	F	25.5	No	Yes	High School	Low	8	4.9
4	S1004	John	Smith	student4@university.com	Female	23	CS	59.41	53.13	61.77	...	87.43	71.4030	C	13.3	Yes	No	Master's	Medium	6	4.5

5 rows × 23 columns

df['Grade'].value_counts() ## This is imbalanced dataset

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.dtypes
```

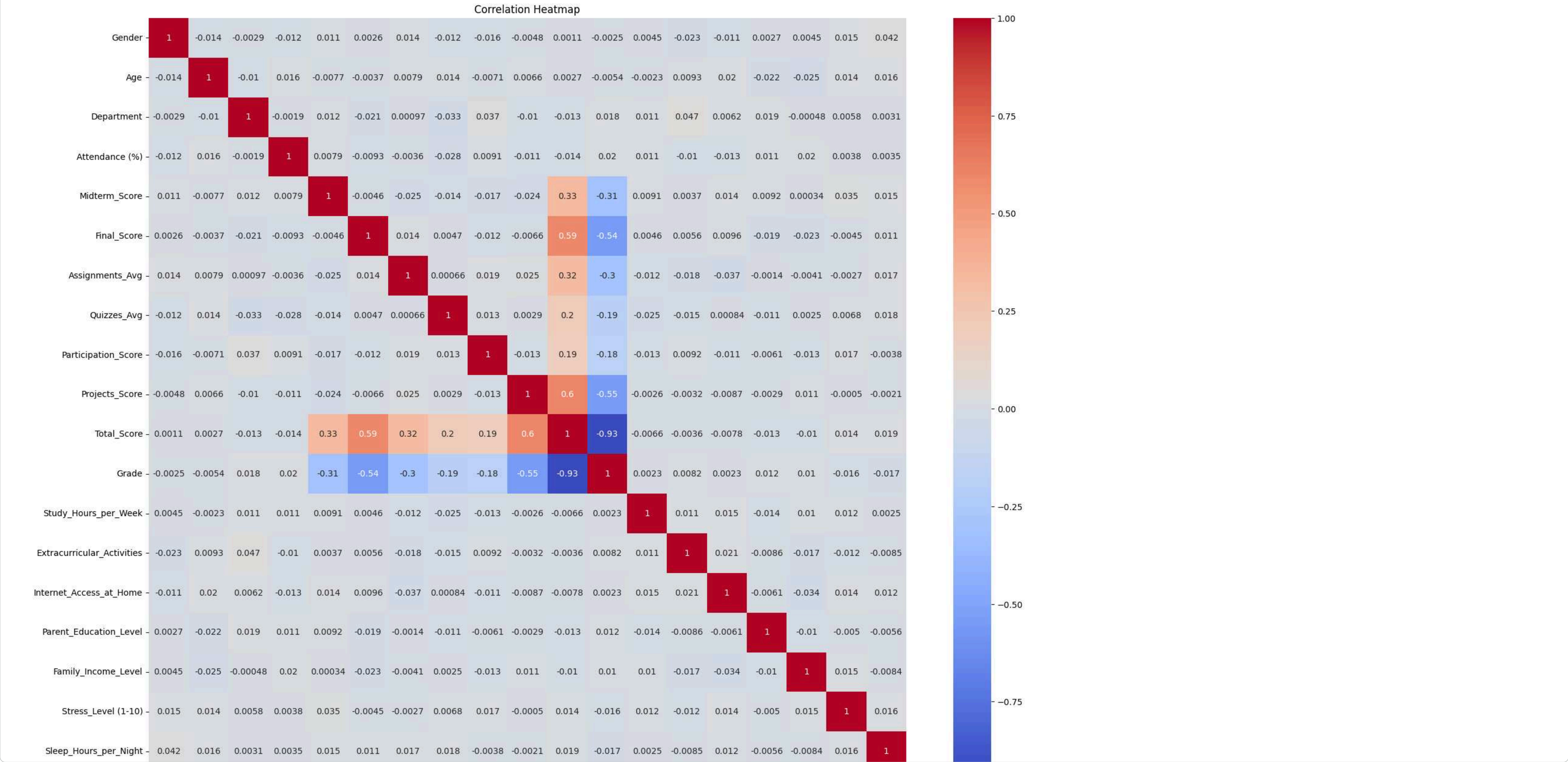
```
df['Parent_Education_Level'].fillna('Unknown', inplace=True)
```

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




```
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()
```



```
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 cleaner)
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()
```



```
# 2. TRAIN / TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

# 3. CHI-SQUARE FEATURE SELECTION
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))

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

# 5. TRAIN + EVALUATE
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')
    })
```

```
)
# 6. RESULTS AS DATAFRAME
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)

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

===== 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      0.96      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      1.00      1.00        1500
 macro avg          1.00      1.00      1.00        1500
weighted avg          1.00      1.00      1.00        1500

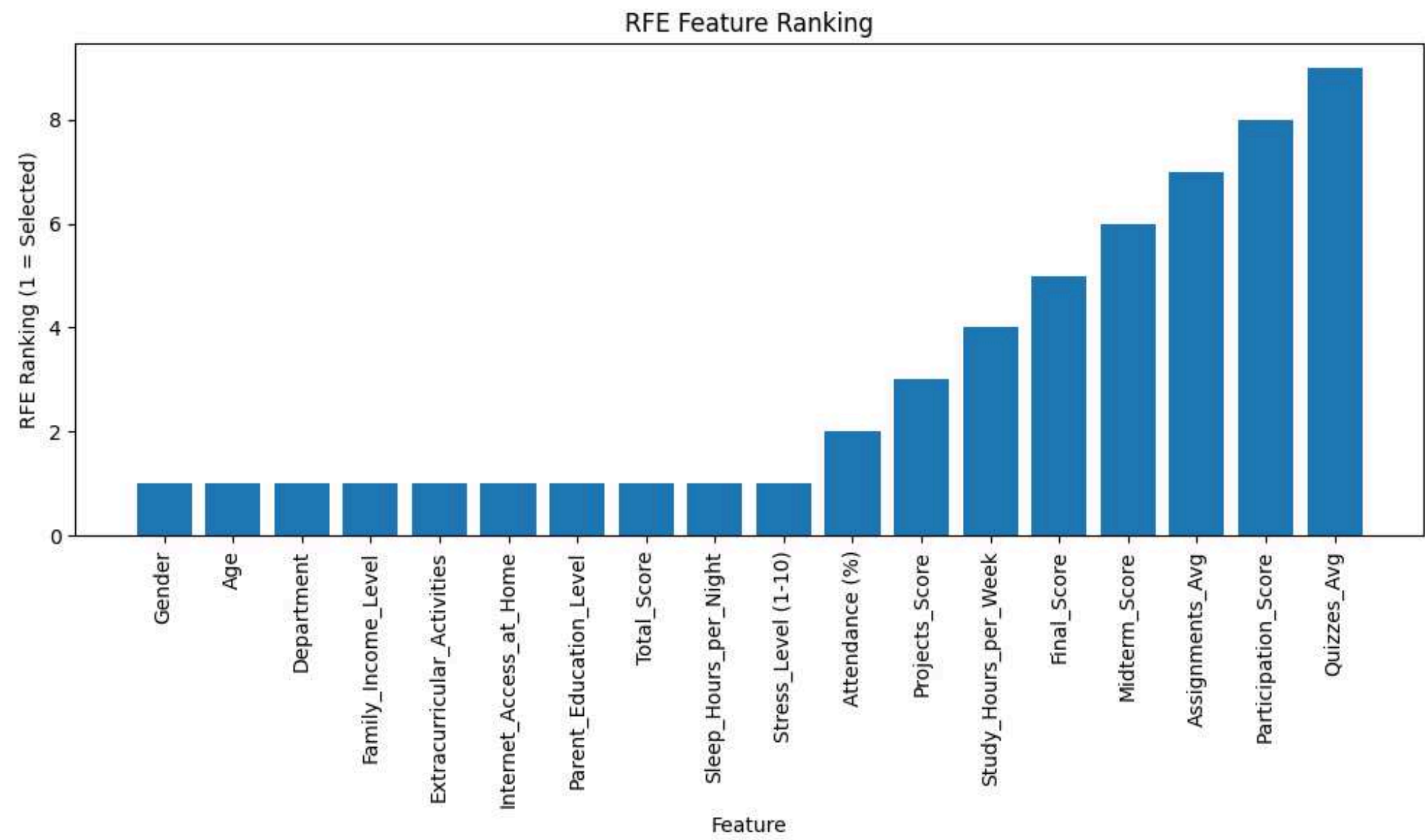
===== Classification Report: SVM =====
      precision    recall  f1-score   support

    0       0.00      0.00      0.00         6
    1       0.96      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          0.77      0.70      0.95        1500
 macro avg          0.77      0.70      0.73        1500
weighted avg          0.94      0.95      0.94        1500

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()
```



```
# 2. TRAIN / TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

# 3. CHI-SQUARE FEATURE SELECTION
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))

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

# 5. TRAIN + EVALUATE
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')
    })

# 6. RESULTS AS DATAFRAME
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)
```

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

===== 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      0.96      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      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       0.96      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          0.77      0.70      0.95        1500
 macro avg          0.77      0.70      0.73        1500
weighted avg          0.94      0.95      0.94        1500

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
```

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

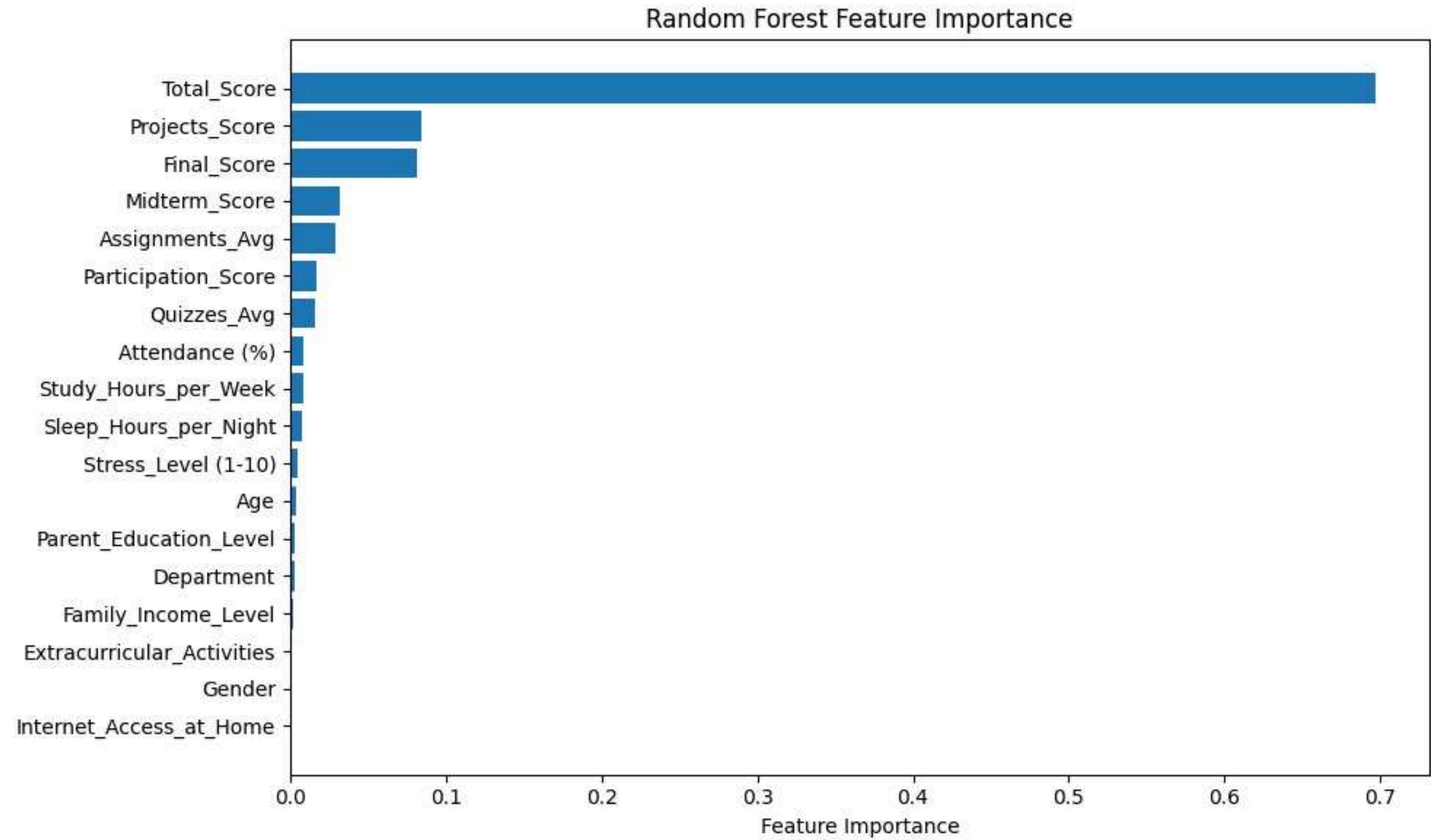
importances = rf.feature_importances_

feature_names = X.columns if hasattr(X, "columns") else [f"Feature {i}" for i in range(X.shape[1])]

df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})
```

```
df_importance = df_importance.sort_values(by='Importance', ascending=False)

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()
```



```
models = { "SVM": Pipeline([ ("scaler", StandardScaler()), ("clf", SVC(kernel="rbf", probability=True, random_state=42)) ]), "Random Forest": RandomForestClassifier( n_estimators=200, random_state=42, n_jobs=-1 ), "Decision Tree": DecisionTreeClassifier( random_state=42 ) }
```

```
def evaluate_models(Xt, yt, method_name):
    print(f"\n{'='*15} {method_name} {'='*15}")
    results = []
    for model_name, model in models.items():
        print(f"\n--- {model_name} ---")
        model.fit(Xt, yt)
        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))
        results.append({
            "Sampling": method_name,
            "Model": model_name,
            "Accuracy": acc,
            "Precision": prec,
            "Recall": rec,
            "F1-score": f1
        })
    return results
```

```
all_results = []

# Original (Imbalanced)
all_results += evaluate_models(
    X_train, y_train, "Original Imbalanced Data")

# Random Oversampling
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_train, y_train)
all_results += evaluate_models(
    X_ros, y_ros, "Random Oversampling")

# Random Undersampling
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_train, y_train)
all_results += evaluate_models(
    X_rus, y_rus, "Random Undersampling")

# SMOTE
smote = SMOTE(random_state=42)
X_sm, y_sm = smote.fit_resample(X_train, y_train)
all_results += evaluate_models(
    X_sm, y_sm, "SMOTE Oversampling")

# SMOTE + Tomek Links
smt = SMOTETomek(random_state=42)
X_smt, y_smt = smt.fit_resample(X_train, y_train)
all_results += evaluate_models(
    X_smt, y_smt, "SMOTE + Tomek Links")

results_df = pd.DataFrame(all_results)

summary = results_df.pivot_table(
    index="Sampling",
    columns="Model",
    values=["Accuracy", "Precision", "Recall", "F1-score"]
).round(4)
print("\n\n===== FINAL COMPARISON =====")
print(summary)
```

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

   0         1.00      1.00      1.00         5
   1         1.00      1.00      1.00        134
   2         1.00      1.00      1.00        450
   3         1.00      1.00      1.00        358
   4         1.00      1.00      1.00         53

 accuracy          1.00      1.00      1.00       1000
 macro avg          1.00      1.00      1.00       1000
weighted avg          1.00      1.00      1.00       1000

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

   0         1.00      1.00      1.00         5
   1         1.00      1.00      1.00        134
   2         1.00      1.00      1.00        450
   3         1.00      1.00      1.00        358
   4         1.00      1.00      1.00         53

 accuracy          1.00      1.00      1.00       1000
 macro avg          1.00      1.00      1.00       1000
weighted avg          1.00      1.00      1.00       1000
```

```
===== FINAL COMPARISON =====

Model          Accuracy          F1-score \
              Decision Tree Random Forest  SVM Decision Tree
Sampling
Original Imbalanced Data      1.000      0.994 0.936      1.0000
Random Oversampling           1.000      1.000 0.917      1.0000
Random Undersampling          0.978      0.853 0.458      0.9786
SMOTE + Tomek Links           1.000      0.999 0.916      1.0000
SMOTE Oversampling            1.000      1.000 0.917      1.0000

Model          Precision          \
              Random Forest  SVM Decision Tree Random Forest
Sampling
Original Imbalanced Data      0.9915 0.9326      1.00      0.9892
Random Oversampling           1.0000 0.9172      1.00      1.0000
Random Undersampling          0.8580 0.4660      0.98      0.8693
SMOTE + Tomek Links           0.9990 0.9152      1.00      0.9990
SMOTE Oversampling            1.0000 0.9161      1.00      1.0000

Model          Recall          SVM
Sampling
Original Imbalanced Data      0.9311      1.000      0.994 0.936
Random Oversampling           0.9214      1.000      1.000 0.917
Random Undersampling          0.5402      0.978      0.853 0.458
SMOTE + Tomek Links           0.9175      1.000      0.999 0.916
SMOTE Oversampling            0.9183      1.000      1.000 0.917
```

```
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)
```

```
importances = rf.feature_importances_
```

```
sorted_idx = np.argsort(importances)[::-1]
```

```
top_k_idx = sorted_idx[:k]
```

```
selected_features = X.columns[top_k_idx]
```

```
print("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', 'Quizzes_Avg', 'Attendance (%)', 'Study_Hours_per_Week', 'Sleep_Hours_per_Night', 'Stress_Level (1-10)', 'Age', 'Parent_Education_Level', 'Department', 'Family_Income_Level', 'Internet_Access_at_Home']
```

```
X_train_k = X_train[selected_features]
X_test_k = X_test[selected_features]
```

```
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    0.998180    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

```

```

from sklearn.feature_selection import SelectKBest, mutual_info_regression

```

```

from xgboost import XGBRegressor

```

```

from sklearn.tree import DecisionTreeRegressor

```

```

from sklearn.ensemble import RandomForestRegressor

```

```

from sklearn.svm import SVR

```

```

from sklearn.model_selection import train_test_split

```

```

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

```

```

selector = SelectKBest(score_func=mutual_info_regression, k='all')

```

```

selector.fit(X, y)

SelectKBest
SelectKBest(k='all',
            score_func=<function mutual_info_regression at 0x7affcd402c0>)

```

```

print("Selected Features:", selector.get_support())

```

```

Selected Features: [ True  True  True  True  True  True  True  True  True  True  True  True
                   True  True  True  True  True  True]

```

```

print("Scores:", selector.scores_)

```

```

Scores: [0.          0.0024743  0.          0.00120287  0.05735739  0.17408596
 0.04306764  0.02178457  0.01883515  0.17815453  1.16505341  0.00133115
 0.          0.          0.00274389  0.          0.00172902  0.01252745]

```

```

data = pd.DataFrame({ 'Feature': X.columns, 'Score': selector.scores_})

```

```

data = data.sort_values(by='Score', ascending=False)

```

```

plt.figure(figsize=(10, 6))

<Figure size 1000x600 with 0 Axes>
<Figure size 1000x600 with 0 Axes>

```

```

plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("F-test Score (Regression)")
plt.title("Feature Selection Scores using SelectKBest (f_regression)")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```



```

from sklearn.feature_selection import RFE

```

```

model = RandomForestClassifier() # Changed to Classifier

rfe = RFE(estimator=model, n_features_to_select=5) # choose k features
rfe.fit(X, y)

ranking = rfe.ranking_
support = rfe.support_
print("Selected Features:", X.columns[support])
print("Feature Ranking:", ranking)

data = pd.DataFrame({'Feature': X.columns, 'Rank': ranking})
data = data.sort_values(by='Rank', ascending=True)

plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Rank'])
plt.xlabel("Feature")
plt.ylabel("RFE Rank (1 = Selected)")
plt.title("RFE Feature Ranking for Classification") # Updated title
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```

```

Selected Features: Index(['Midterm_Score', 'Final_Score', 'Assignments_Avg', 'Projects_Score',
                        'Total_Score'],
                        dtype='object')
Feature Ranking: [14  8 10  4  1  1  1  3  2  1  1  5 13 12  9 11  7  6]

```



```

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
```

```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/Project 3/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)
```

```
df.head()
```

```
-----
NameError                                Traceback (most recent call last)
/tmp/ipython-input-964094849.py in <cell line: 0>()
----> 1 df.head()

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

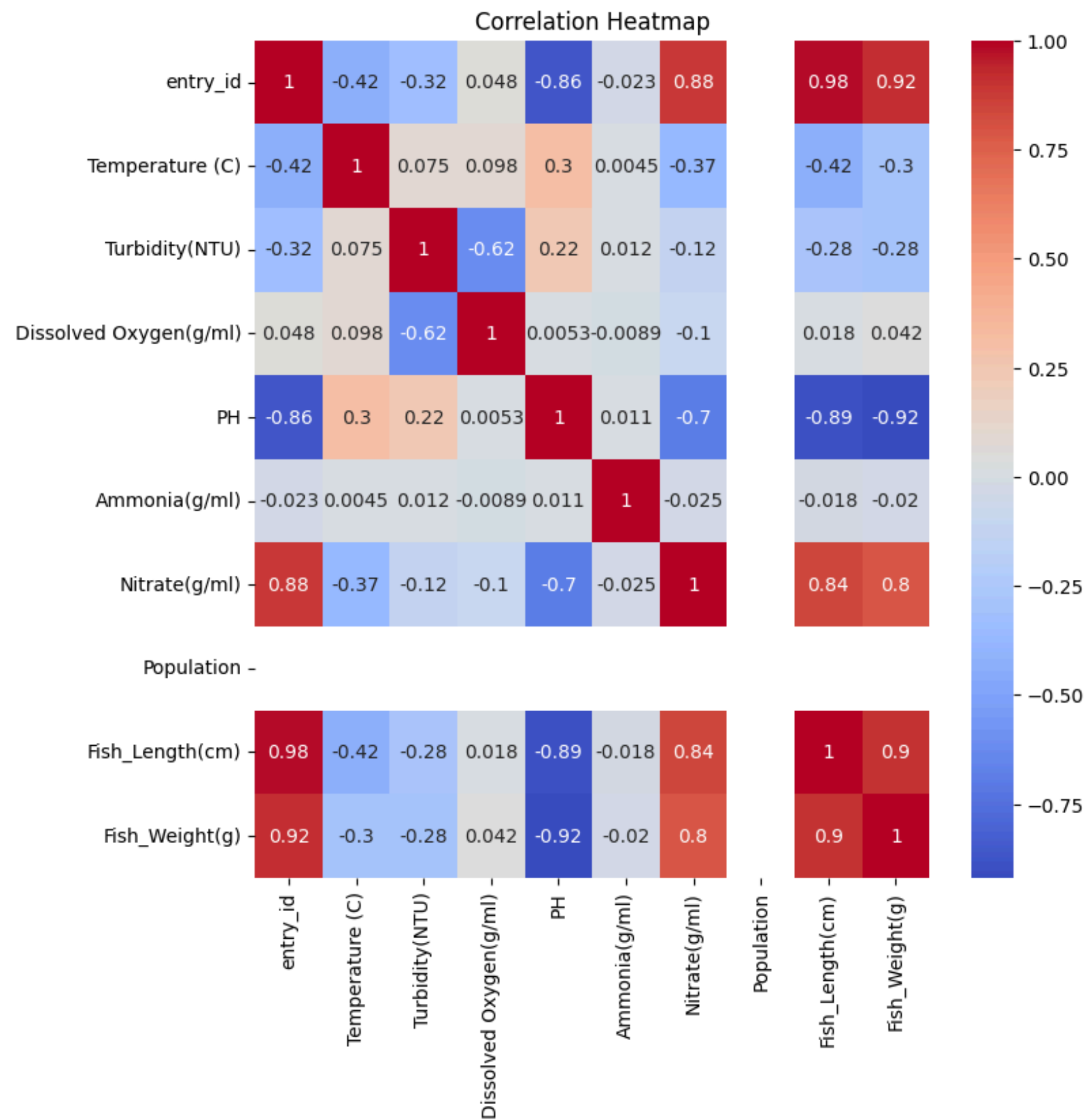
Next steps: [Explain error](#)

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

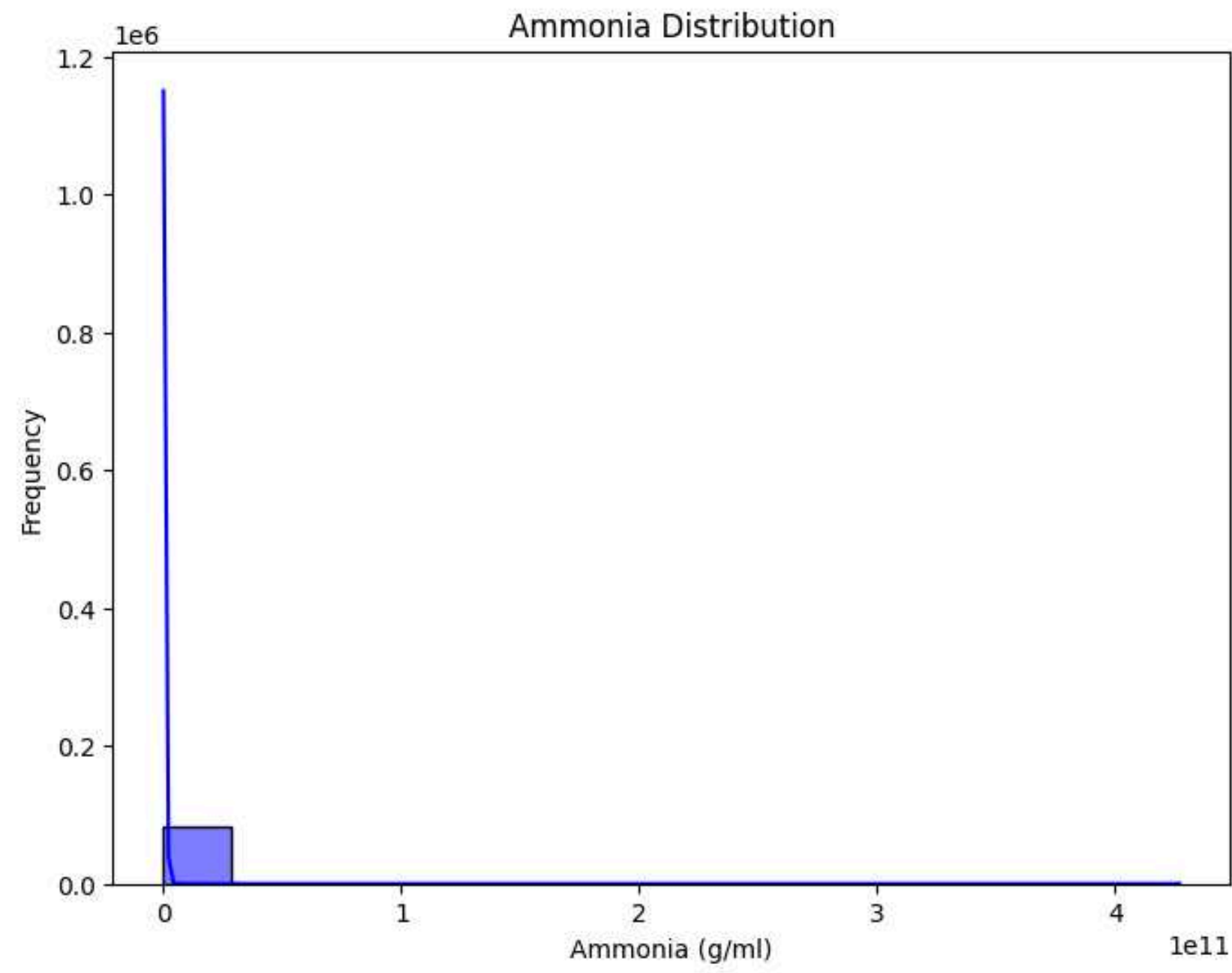
	0
created_at	0
entry_id	0
Temperature (C)	0
Turbidity(NTU)	0
Dissolved Oxygen(g/ml)	0
PH	0
Ammonia(g/ml)	52
Nitrate(g/ml)	0
Population	0
Fish_Length(cm)	2
Fish_Weight(g)	2
dtype:	int64

```
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 (g/ml)')
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)]
```

	entry_id	Temperature (C)	Turbidity(NTU)	Dissolved Oxygen(g/ml)	PH	Ammonia(g/ml)	Nitrate(g/ml)	Population	Fish_Length(cm)	Fish_Weight(g)
1801	4252	26.2500	100	9.846	8.38371	157131.51560	242	50	7.11	2.91
1802	4253	26.2500	100	18.915	8.38371	63147.98047	246	50	7.11	2.91
1803	4254	26.2500	100	22.989	8.38371	1907.52869	274	50	7.11	2.91
1804	4255	26.3125	100	10.691	8.38371	43343.46094	273	50	7.11	2.91
1805	4256	26.3125	100	12.800	8.38825	636403.50000	270	50	7.11	2.91
...	...	...	...	...	...	...	...	...	...	...
82915	242744	27.6875	100	0.086	5.79622	78.27628	911	50	33.31	313.72
82916	242756	27.6875	100	0.311	5.79168	44.89249	942	50	33.31	313.72
82917	242760	27.6875	100	0.832	5.79168	39.83778	945	50	33.31	313.72
82920	242804	27.6875	100	0.224	5.69635	38.55425	961	50	33.31	313.72
82924	242876	27.6875	100	1.584	4.33905	44.70709	940	50	33.31	313.72

16818 rows × 10 columns

```
df.drop(['Ammonia(g/ml)'], axis=1, inplace=True)
```

```
df.fillna(df.mean(), inplace=True)
```

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

	0
entry_id	0
Temperature (C)	0
Turbidity(NTU)	0
Dissolved Oxygen(g/ml)	0
PH	0
Nitrate(g/ml)	0
Population	0
Fish_Length(cm)	0
Fish_Weight(g)	0
dtype: int64	

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

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

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)
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

rfr = RandomForestRegressor(n_estimators=200, random_state=42)
multi_rfr = MultiOutputRegressor(rfr)

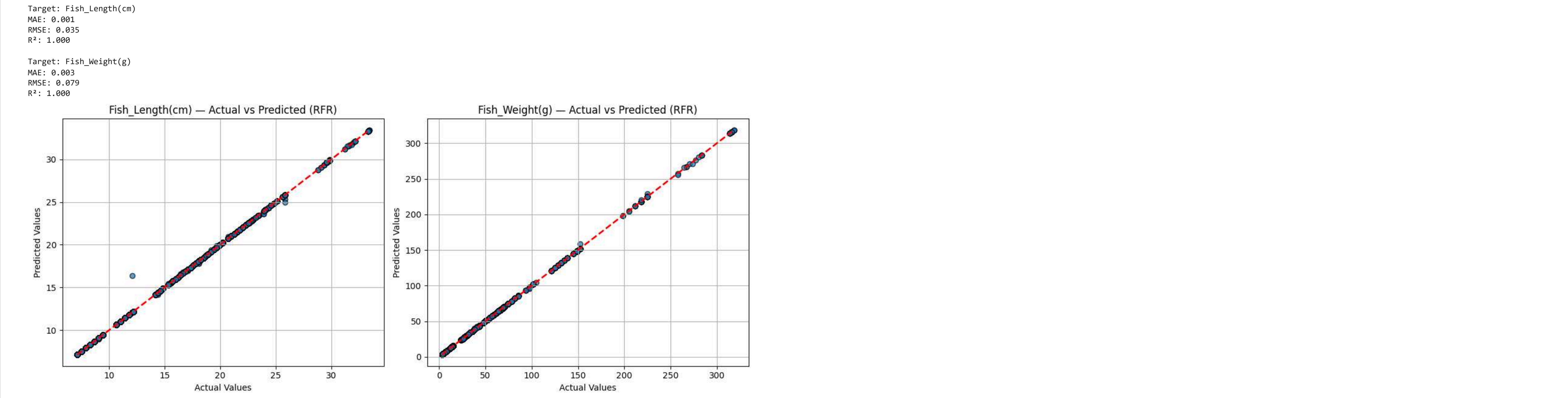
# Train
multi_rfr.fit(X_train, y_train)

# 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}")

# ----- 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 (RFR)')
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```



```
from sklearn.tree import DecisionTreeRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Define multi-output Decision Tree Regressor
dtr = DecisionTreeRegressor(random_state=42)
multi_dtr = MultiOutputRegressor(dtr)

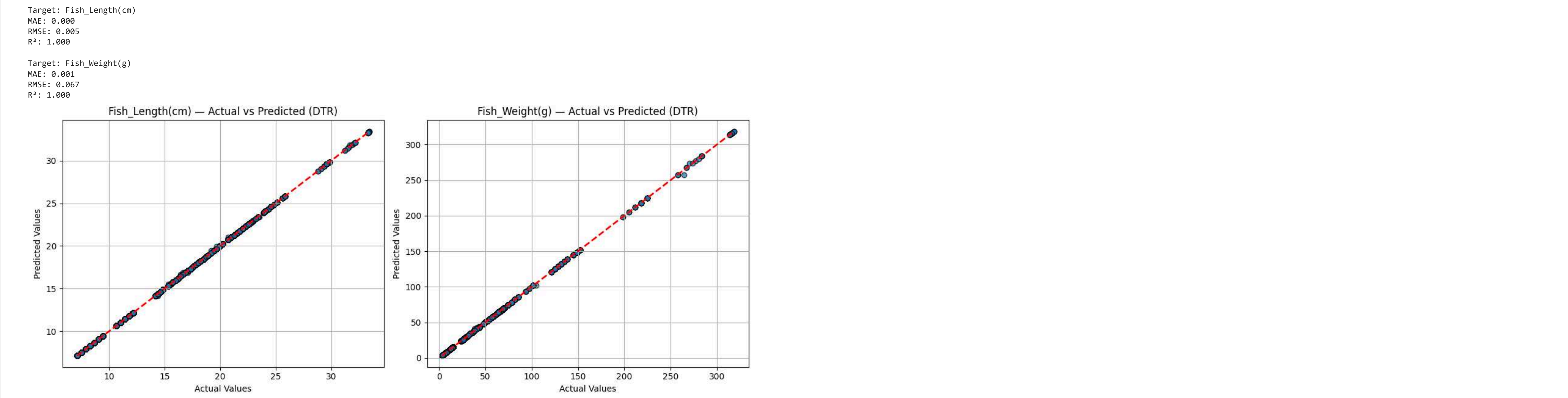
# Train model
multi_dtr.fit(X_train, y_train)

# 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.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt
```

```
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}")

# ----- 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 (SVR)')
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```

```
from xgboost import XGBRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

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
)

# Wrap in MultiOutputRegressor
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}")

# Scatter Plots: Actual vs Predicted
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()
```

Target: Fish_Length(cm)
MAE : 0.036
RMSE: 0.272
R² : 0.997

Target: Fish_Weight(g)
MAE : 0.145
RMSE: 0.639
R² : 1.000

