

```
from IPython.display import Image, display
display(Image(filename='Lecture 10.jpg'))
```



MIDOCEAN
UNIVERSITY

Applying Different Projects

Lecture 10

Prepared by

Dr. Hager Saleh

Import necessary libraries

```
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

```

Dataset linking

<https://www.kaggle.com/datasets/mahmoudelhema1y/students-grading-dataset>

Reading dataset

```

df = pd.read_csv("Students Performance Dataset.csv")
df.head()

```

	Student_ID	First_Name	Last_Name	Email	Gender
Age \					
0	S1000	Omar	Williams	student0@university.com	Female
22					
1	S1001	Maria	Brown	student1@university.com	Male
18					
2	S1002	Ahmed	Jones	student2@university.com	Male
24					
3	S1003	Omar	Williams	student3@university.com	Female
24					
4	S1004	John	Smith	student4@university.com	Female

23

	Department	Attendance (%)	Midterm_Score	Final_Score	...	\
0	Mathematics	97.36	40.61	59.61	...	
1	Business	97.71	57.27	74.00	...	
2	Engineering	99.52	41.84	63.85	...	
3	Engineering	90.38	45.65	44.44	...	
4	CS	59.41	53.13	61.77	...	

	Projects_Score	Total_Score	Grade	Study_Hours_per_Week	\
0	62.84	59.8865	F	10.3	
1	98.23	81.9170	B	27.1	
2	91.22	67.7170	D	12.4	
3	55.48	51.6535	F	25.5	
4	87.43	71.4030	C	13.3	

	Extracurricular_Activities	Internet_Access_at_Home	Parent_Education_Level	\
0		Yes	No	
1	Master's	No	No	High
2	School	Yes	No	High
3	School	No	Yes	High
4	Master's	Yes	No	

	Family_Income_Level	Stress_Level (1-10)	Sleep_Hours_per_Night
0	Medium	1	5.9
1	Low	4	4.3
2	Low	9	6.1
3	Low	8	4.9
4	Medium	6	4.5

[5 rows x 23 columns]

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

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

```
# Check for missing values
df.isnull().sum()
```

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

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

Filling missing values

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

Define features and target variable

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

```
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 \
count	5000.000000	5000.000000	5000.000000	5000.000000
mean	21.048400	75.356076	70.701924	69.546552
std	1.989786	14.392716	17.436325	17.108996
min	18.000000	50.010000	40.000000	40.010000
25%	19.000000	62.945000	55.707500	54.697500
50%	21.000000	75.670000	70.860000	69.485000
75%	23.000000	87.862500	85.760000	83.922500
max	24.000000	100.000000	99.990000	99.980000

	Assignments_Avg	Quizzes_Avg	Participation_Score
Projects_Score \			
count	5000.000000	5000.000000	5000.000000
5000.000000			
mean	74.956320	74.836214	49.963720
74.78305			
std	14.404287	14.423848	28.989785
14.54243			
min	50.000000	50.000000	0.000000
50.000000			

25%	62.340000	62.357500	25.075000
61.97000			
50%	75.090000	74.905000	49.600000
74.54000			
75%	87.352500	87.292500	75.500000
87.63000			
max	99.990000	99.990000	100.000000
100.00000			

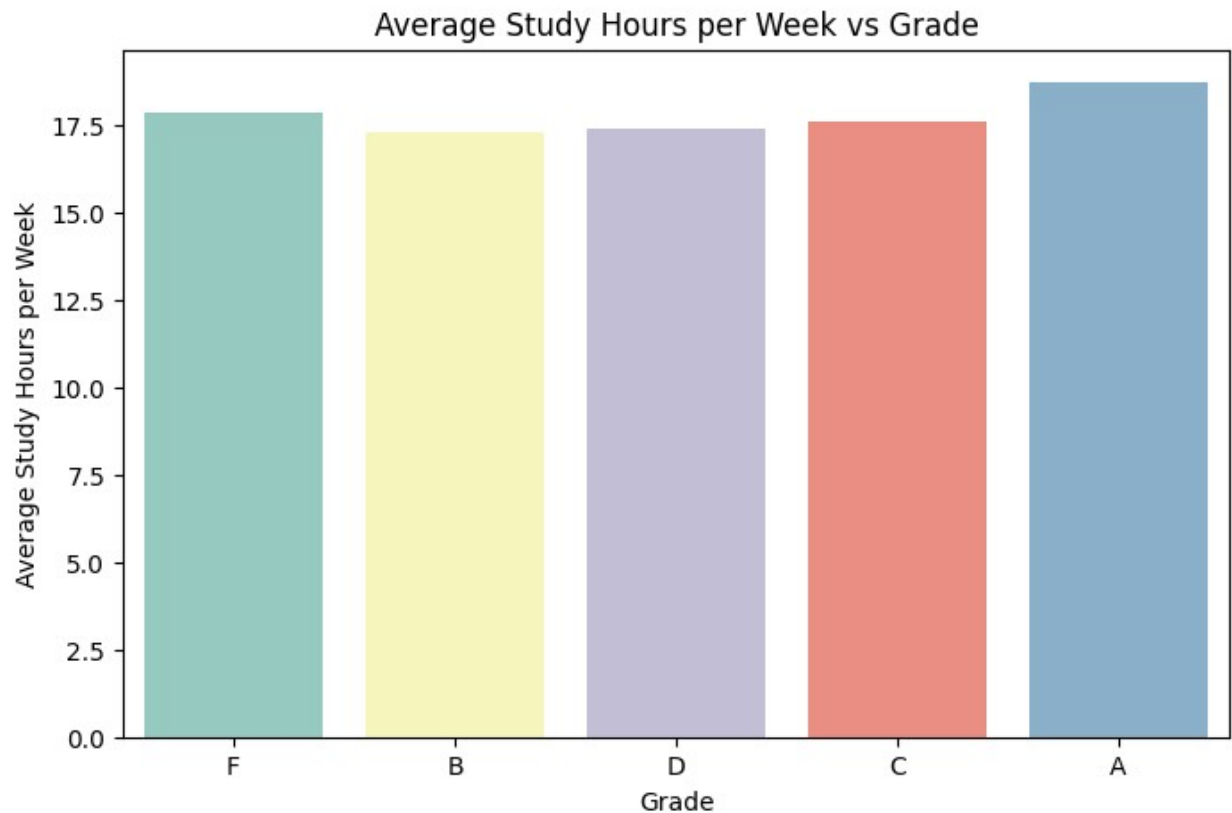
	Total_Score	Study_Hours_per_Week	Stress_Level (1-10) \
count	5000.000000	5000.000000	5000.000000
mean	71.652097	17.521140	5.507200
std	7.230097	7.193035	2.886662
min	50.602000	5.000000	1.000000
25%	66.533875	11.500000	3.000000
50%	71.696250	17.400000	6.000000
75%	76.711625	23.700000	8.000000
max	95.091500	30.000000	10.000000

	Sleep_Hours_per_Night
count	5000.000000
mean	6.514420
std	1.446155
min	4.000000
25%	5.300000
50%	6.500000
75%	7.800000
max	9.000000

Average Study Hours per Week vs Grade

```
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()
```

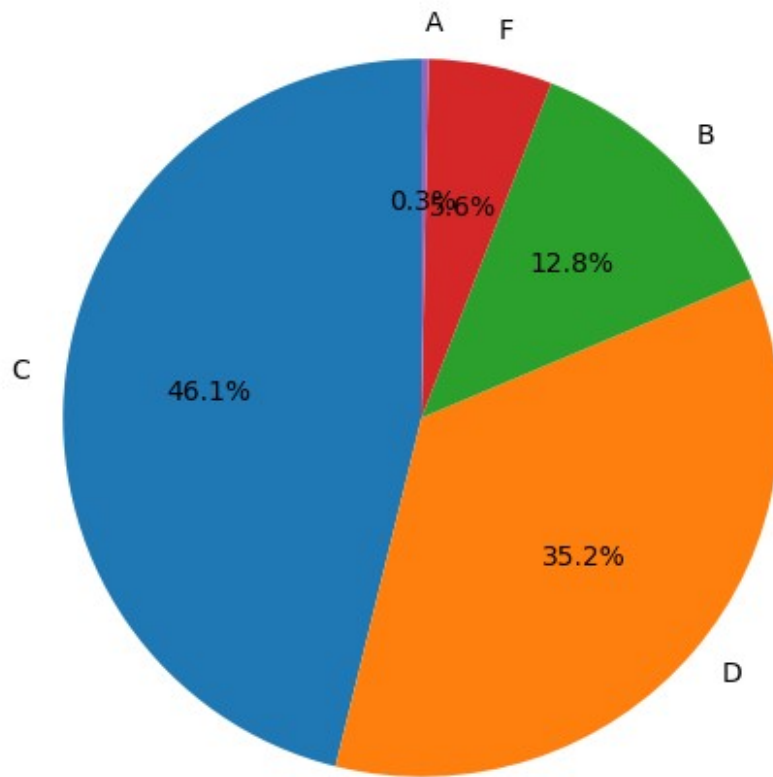


Distribution of Students by Grade

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

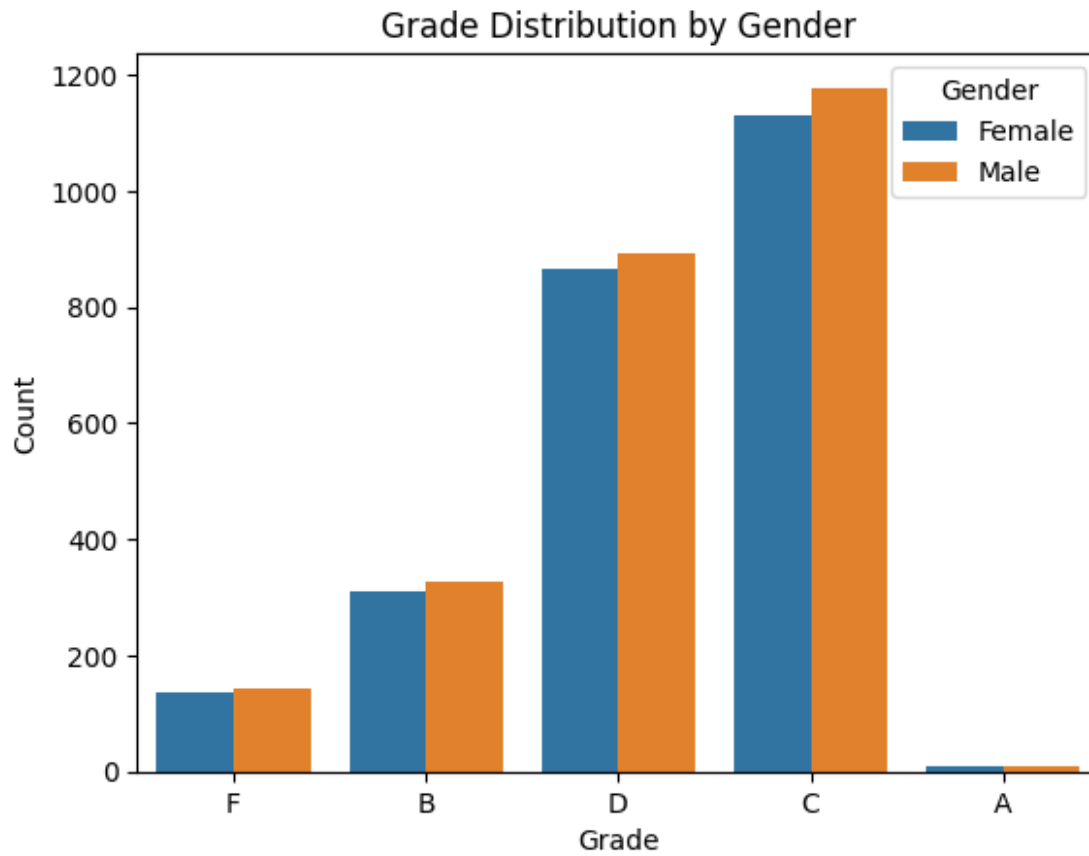
plt.figure(figsize=(6,6))
plt.pie(
    grade_counts,
    labels=grade_counts.index,
    autopct='%1.1f%%',
    startangle=90
)
plt.title("Distribution of Students by Grade")
plt.show()
```

Distribution of Students by Grade



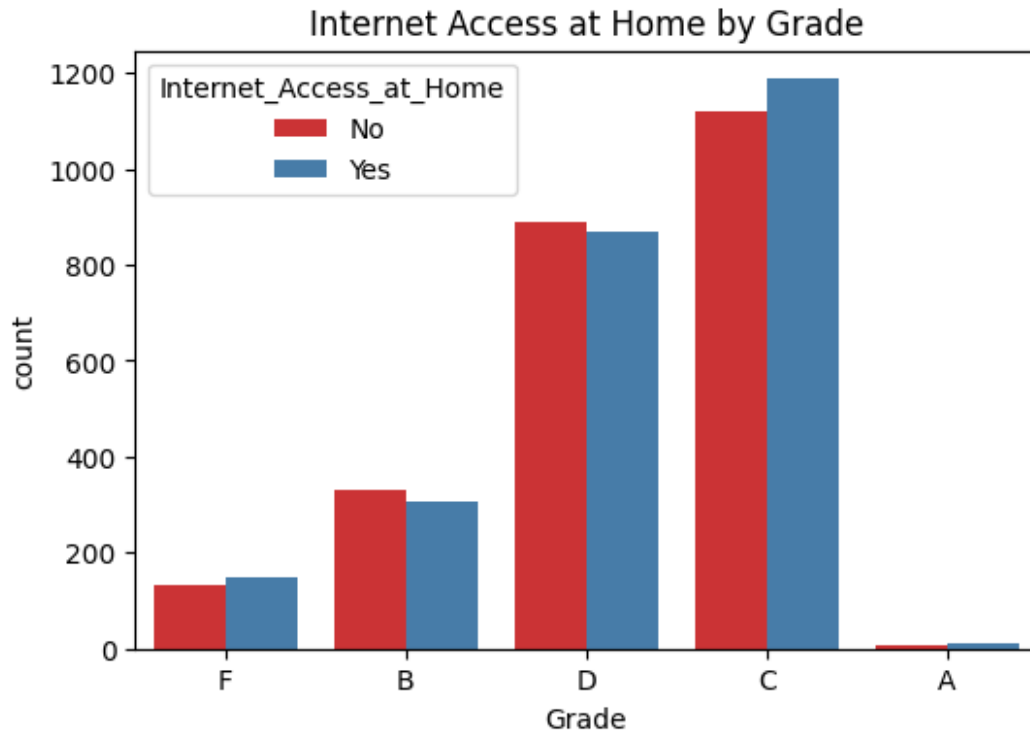
Grade Distribution by Gender

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



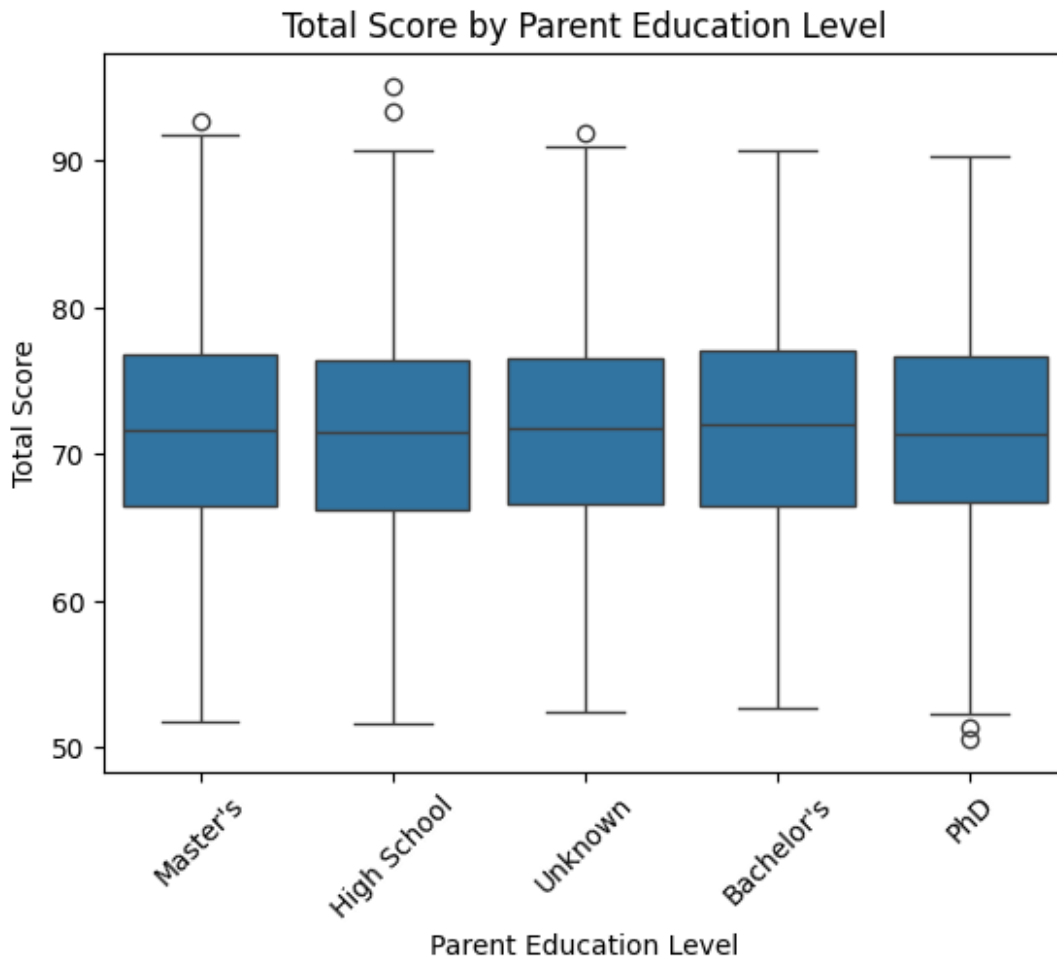
Internet Access at Home by Grade

```
# Internet Access
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()
```



Total Score by Parent Education Level

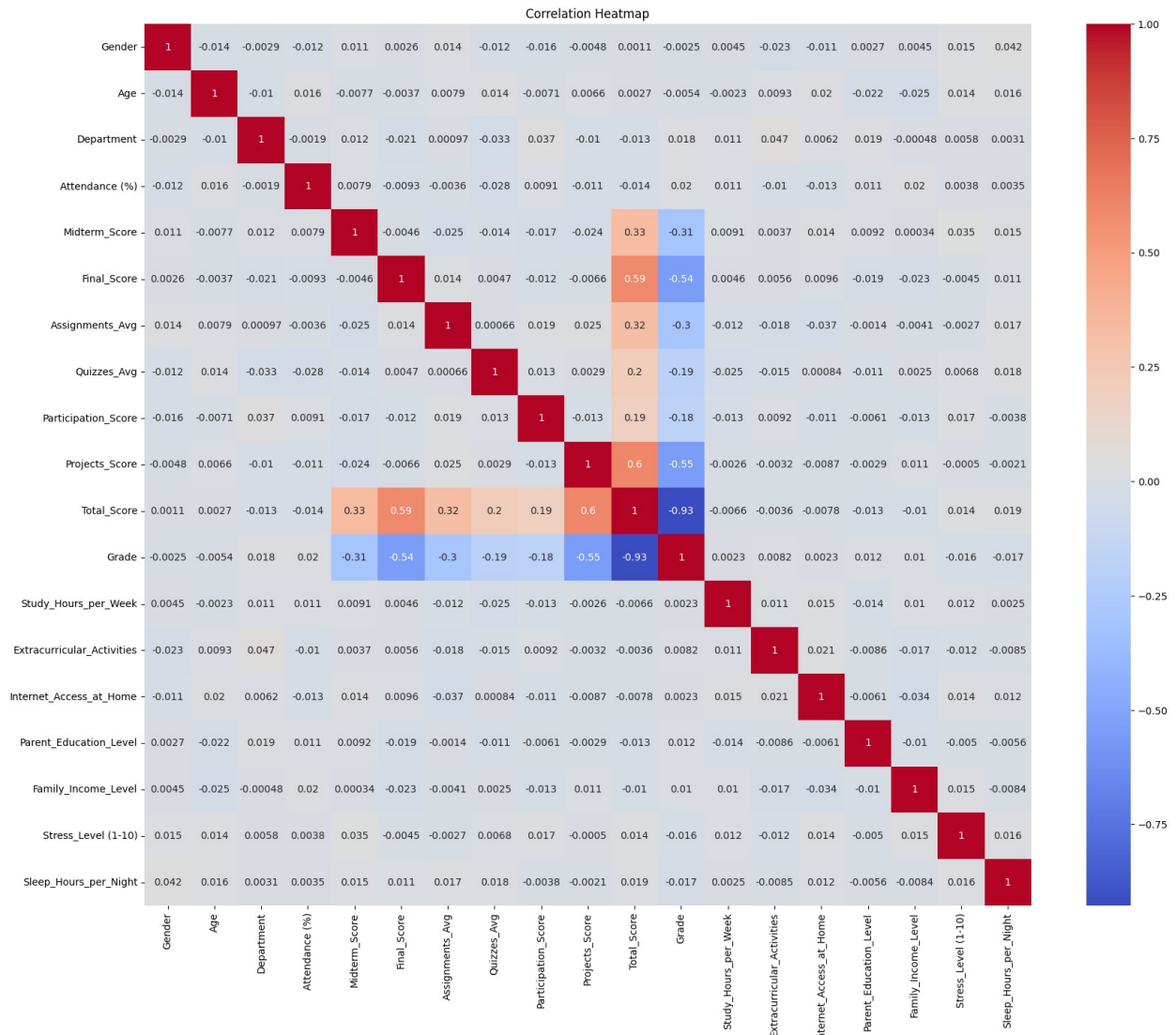
```
sns.boxplot(x='Parent_Education_Level', y='Total_Score', data=df)
plt.title('Total Score by Parent Education Level')
plt.xticks(rotation=45)
plt.xlabel('Parent Education Level')
plt.ylabel('Total Score')
plt.show()
```



LabelEncoder

```
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()
```



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

Chi-Square as feature selection

```
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_})

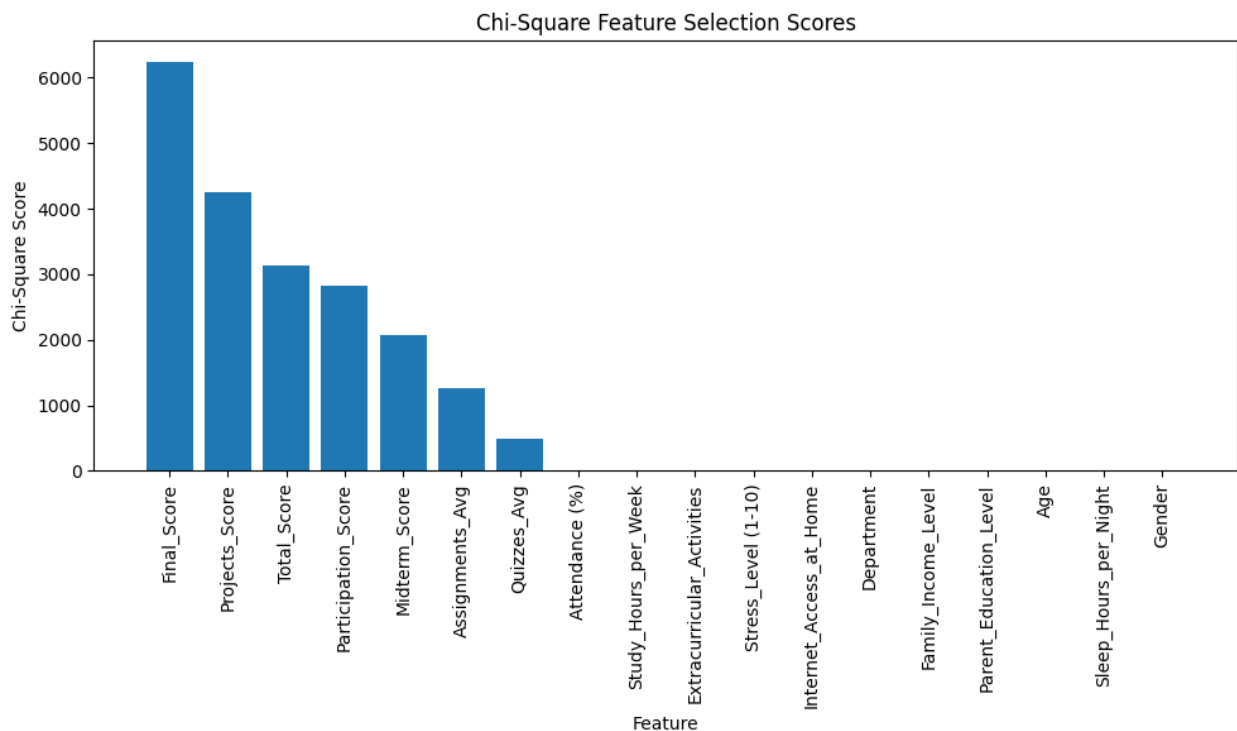
# Sort by score (optional, but makes plot clearer)
```

```

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()

```



Comparison of models using features selected by the Chi-Square test

```

# -----
# 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	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

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

Recursive Feature Elimination (RFE) with Logistic Regression

```
model = LogisticRegression(max_iter=500)
```

```

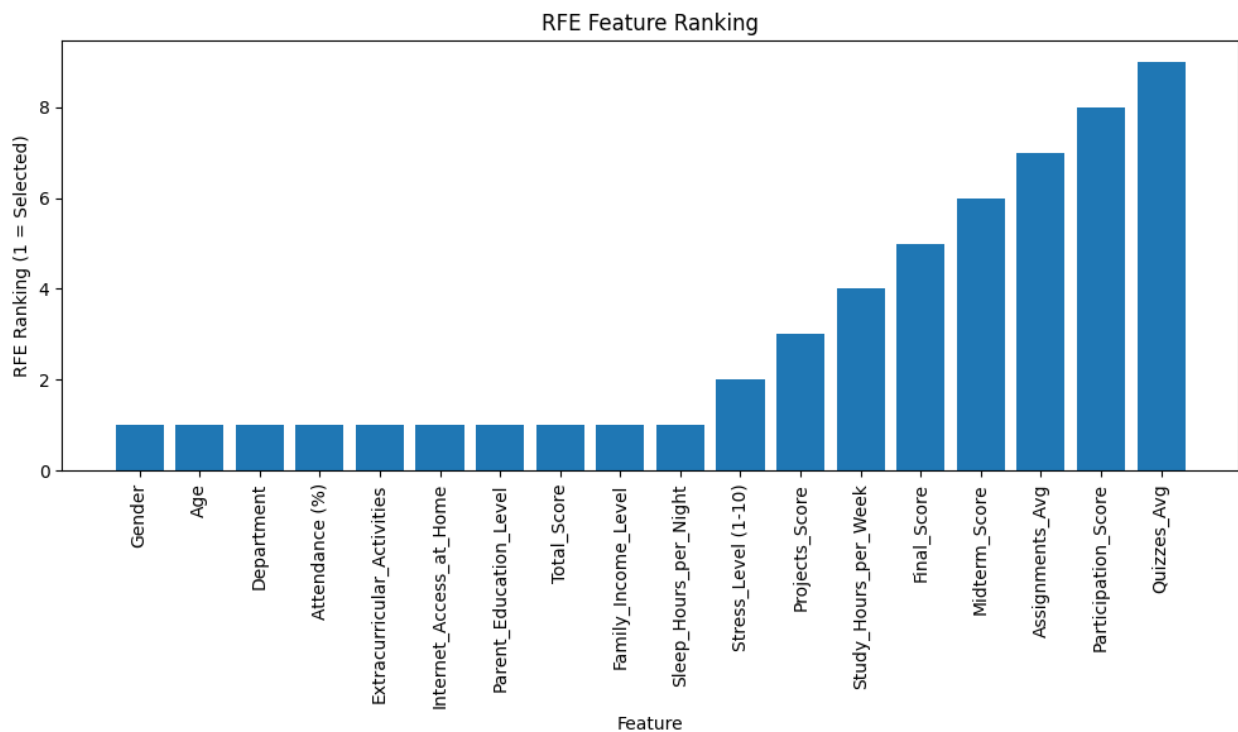
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X, y)

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()

```



Comparison of models using features selected by the REF

```

# -----
# 2. TRAIN / TEST SPLIT

```

```

# -----
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))

# -----
# 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	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			1.00	1500
macro avg	0.79	0.80	0.80	1500
weighted avg	0.99	1.00	0.99	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	0.97	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	1500
macro avg	0.76	0.63	0.67	1500
weighted avg	0.92	0.92	0.91	1500

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

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

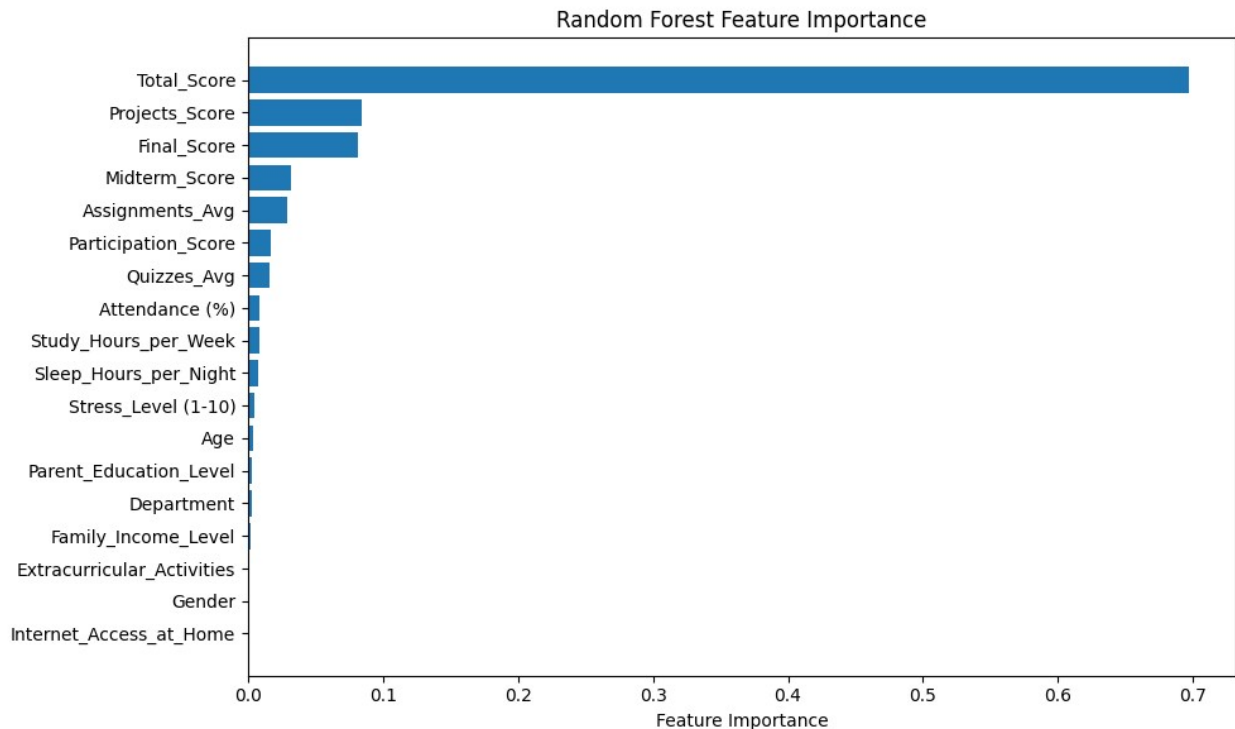
```
df_importance = pd.DataFrame({  
    'Feature': feature_names,  
    'Importance': importances  
})
```

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()
```



Applying imbalanced methods

```
# =====
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
    )
}

# =====
# Evaluation Function (MULTICLASS SAFE)
# =====
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

# =====
# Run Experiments
# =====
all_results = []

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

# 2 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"
)

# 3 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"
)

# 4 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"
)

# 5 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"
)

# =====
# Final Comparison Table
# =====
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)

```

===== 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

===== Random Oversampling =====

--- 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

===== Random Undersampling =====

--- SVM ---

	precision	recall	f1-score	support
0	0.09	0.83	0.17	6
1	0.42	0.50	0.45	208
2	0.62	0.55	0.59	673
3	0.58	0.46	0.51	527
4	0.37	0.80	0.51	86
accuracy			0.53	1500
macro avg	0.42	0.63	0.45	1500
weighted avg	0.56	0.53	0.54	1500

--- Random Forest ---

	precision	recall	f1-score	support
0	0.26	1.00	0.41	6
1	0.74	0.87	0.80	208
2	0.95	0.88	0.91	673
3	0.95	0.77	0.85	527
4	0.46	0.99	0.62	86

accuracy			0.85	1500
macro avg	0.67	0.90	0.72	1500
weighted avg	0.89	0.85	0.86	1500

--- Decision Tree ---

	precision	recall	f1-score	support
0	0.60	1.00	0.75	6
1	0.95	0.98	0.96	208
2	1.00	0.98	0.99	673
3	1.00	0.85	0.92	527
4	0.52	1.00	0.68	86

accuracy			0.94	1500
macro avg	0.81	0.96	0.86	1500
weighted avg	0.96	0.94	0.94	1500

===== SMOTE Oversampling =====

--- 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
4	0.78	0.78	0.78	86

accuracy			0.91	1500
macro avg	0.90	0.77	0.80	1500
weighted avg	0.91	0.91	0.91	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	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

--- 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

===== SMOTE + Tomek Links =====

--- SVM ---

	precision	recall	f1-score	support
0	1.00	0.33	0.50	6
1	0.88	0.88	0.88	208
2	0.92	0.93	0.93	673
3	0.91	0.91	0.91	527
4	0.78	0.78	0.78	86
accuracy			0.91	1500
macro avg	0.90	0.77	0.80	1500
weighted avg	0.91	0.91	0.90	1500

--- Random Forest ---

	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	0.99	0.99	86
accuracy			1.00	1500
macro avg	1.00	1.00	1.00	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

===== FINAL COMPARISON =====

score \	Accuracy	F1-
Model	Decision Tree	Random Forest
SVM Decision	Tree	Sampling

Original Imbalanced Data	1.0000	0.9960	0.9320
1.0000			
Random Oversampling	1.0000	0.9993	0.9213
1.0000			
Random Undersampling	0.9367	0.8453	0.5287
0.9433			
SMOTE + Tomek Links	1.0000	0.9980	0.9053
1.0000			
SMOTE Oversampling	1.0000	0.9987	0.9107
1.0000			

	Precision
\	
Model	Random Forest
SVM Decision	Tree
Random	Forest
Sampling	

Original Imbalanced Data	0.9948	0.9287	1.0000
0.9961			
Random Oversampling	0.9993	0.9211	1.0000
0.9993			
Random Undersampling	0.8569	0.5371	0.9637
0.8908			
SMOTE + Tomek Links	0.9980	0.9048	1.0000
0.9980			
SMOTE Oversampling	0.9986	0.9101	1.0000
0.9987			

	Recall			
Model	SVM Decision Tree			
Sampling	Random Forest			
	SVM			
Original Imbalanced Data	0.9288	1.0000	0.9960	0.9320
Random Oversampling	0.9229	1.0000	0.9993	0.9213
Random Undersampling	0.5649	0.9367	0.8453	0.5287
SMOTE + Tomek Links	0.9056	1.0000	0.9980	0.9053
SMOTE Oversampling	0.9109	1.0000	0.9987	0.9107

```

# -----
# 1. TRAIN/TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# 2. SELECT FEATURES USING RANDOM FOREST IMPORTANCE
# -----
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("Selected Features by RF Importance:", list(selected_features))

# Reduce datasets to selected features
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 = []

# -----
# 4. TRAIN + EVALUATE
# -----
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')
        })

# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RF-Selected Features:")
results_df

Selected Features by RF Importance: ['area_worst', 'concave
points_worst', 'perimeter_worst', 'concave points_mean',
'radius_worst', 'concavity_mean', 'radius_mean', 'area_mean',
'concavity_worst', 'perimeter_mean', 'area_se', 'texture_worst',
'compactness_worst', 'radius_se', 'compactness_mean', 'texture_mean',
'smoothness_worst', 'concavity_se']

Model Performance Using RF-Selected Features:

```

	Model	Accuracy	Precision	Recall
0	Random Forest	0.956140	0.956088	0.956140
1	Decision Tree	0.912281	0.912281	0.912281
2	SVM	0.947368	0.951470	0.947368

Import necessary libraries

```

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

# Select top k features (choose any k)
selector = SelectKBest(score_func=mutual_info_regression, k='all')

# Fit selector
selector.fit(X, y)

```

```

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

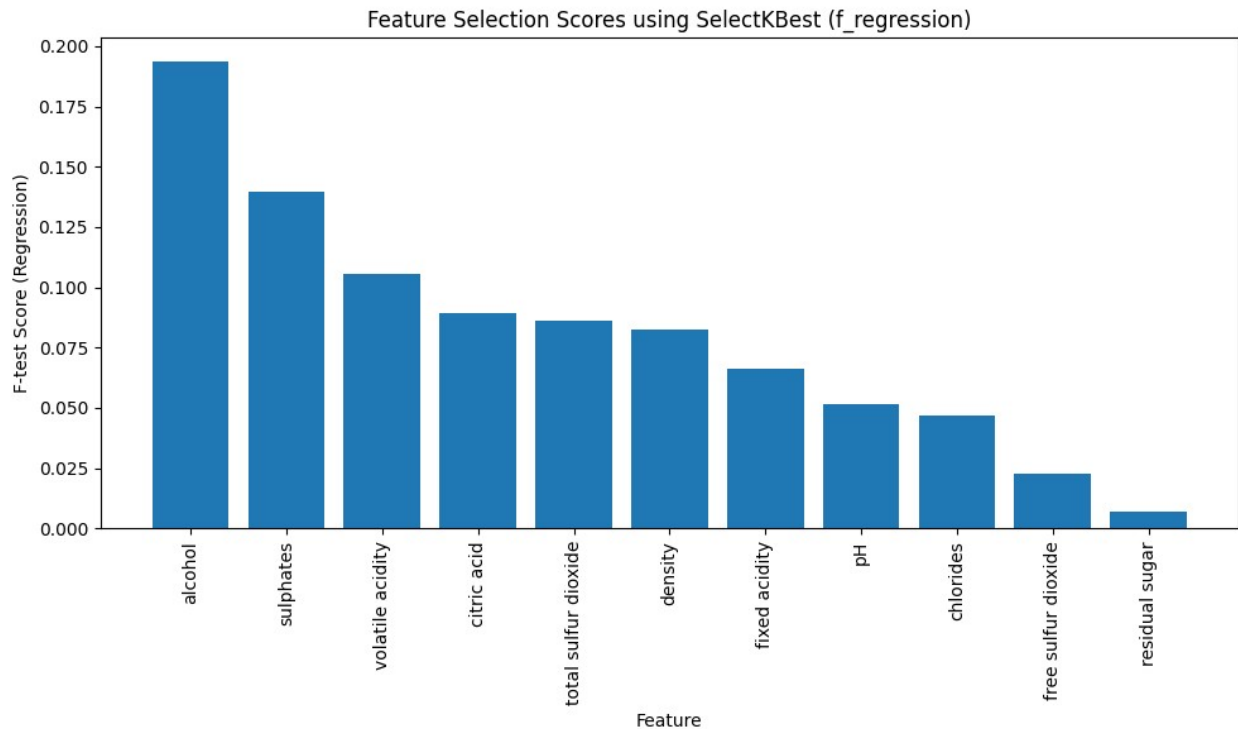
# Create DataFrame of features and F-test scores
data = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
})

# Sort by score for clearer plot
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("F-test Score (Regression)")
plt.title("Feature Selection Scores using SelectKBest (f_regression)")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```

Selected Features: [True True True True True True True True
 True True True]
 Scores: [0.06648509 0.10580416 0.08929389 0.00688396 0.04664157
 0.02252055
 0.08621084 0.08259931 0.0515941 0.13962082 0.19376264]



```
from sklearn.feature_selection import RFE

# Define regression model for RFE
model = RandomForestRegressor()

# RFE: choose number of features (or use all for ranking)
rfe = RFE(estimator=model, n_features_to_select=5) # choose k
features
rfe.fit(X, y)

# Get ranking (1 = selected, higher = less important)
ranking = rfe.ranking_
support = rfe.support_

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

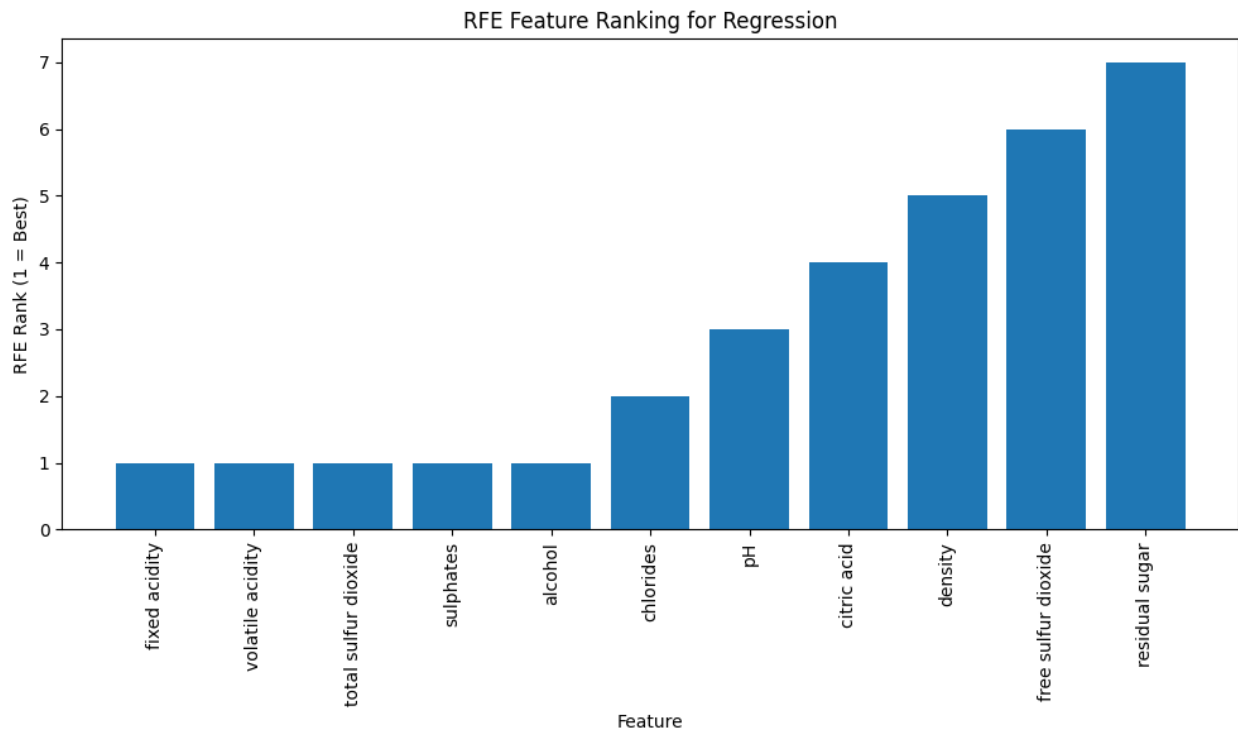
# Create DataFrame of features and RFE ranks
data = pd.DataFrame({
    'Feature': X.columns,
    'Rank': ranking
})

# Sort by rank (1 is best)
data = data.sort_values(by='Rank', ascending=True)

# Plot bar chart of feature ranks
```

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

```
Selected Features: Index(['fixed acidity', 'volatile acidity', 'total
sulphur dioxide',
                        'sulphates', 'alcohol'],
                        dtype='object')
Feature Ranking: [1 1 4 7 2 6 1 5 3 1 1]
```



Multi-task regression problem dataset

Sensor Based Aquaponics Fish Pond Datasets

<https://www.kaggle.com/dsv/3748790>

```
# =====
# Core Libraries
# =====
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("IoTpond1.csv")
df.head()

```

		created_at	Temperature	Turbidity	Dissolved Oxygen
PH \					
0	2021-06-19 00:00:05 CET	24.8750	100	4.505	
8.43365					
1	2021-06-19 00:01:02 CET	24.9375	100	6.601	
8.43818					
2	2021-06-19 00:01:22 CET	24.8750	100	15.797	
8.42457					
3	2021-06-19 00:01:44 CET	24.9375	100	5.046	
8.43365					
4	2021-06-19 00:02:07 CET	24.9375	100	38.407	
8.40641					
Ammonia	Nitrate	Fish_Length	Fish_Weight		

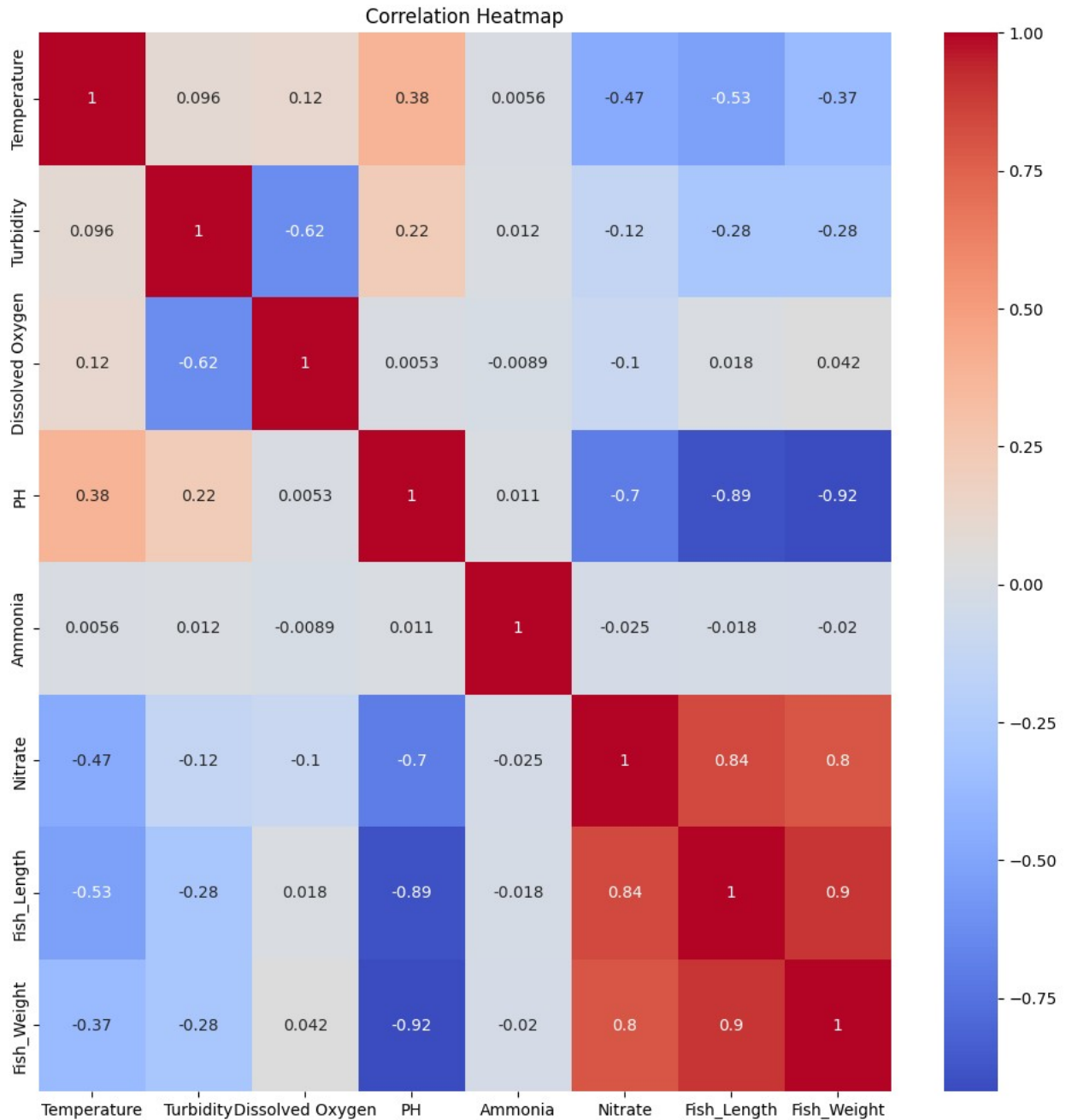
0	0.45842	193	7.11	2.91
1	0.45842	194	7.11	2.91
2	0.45842	192	7.11	2.91
3	0.45842	193	7.11	2.91
4	0.45842	192	7.11	2.91

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

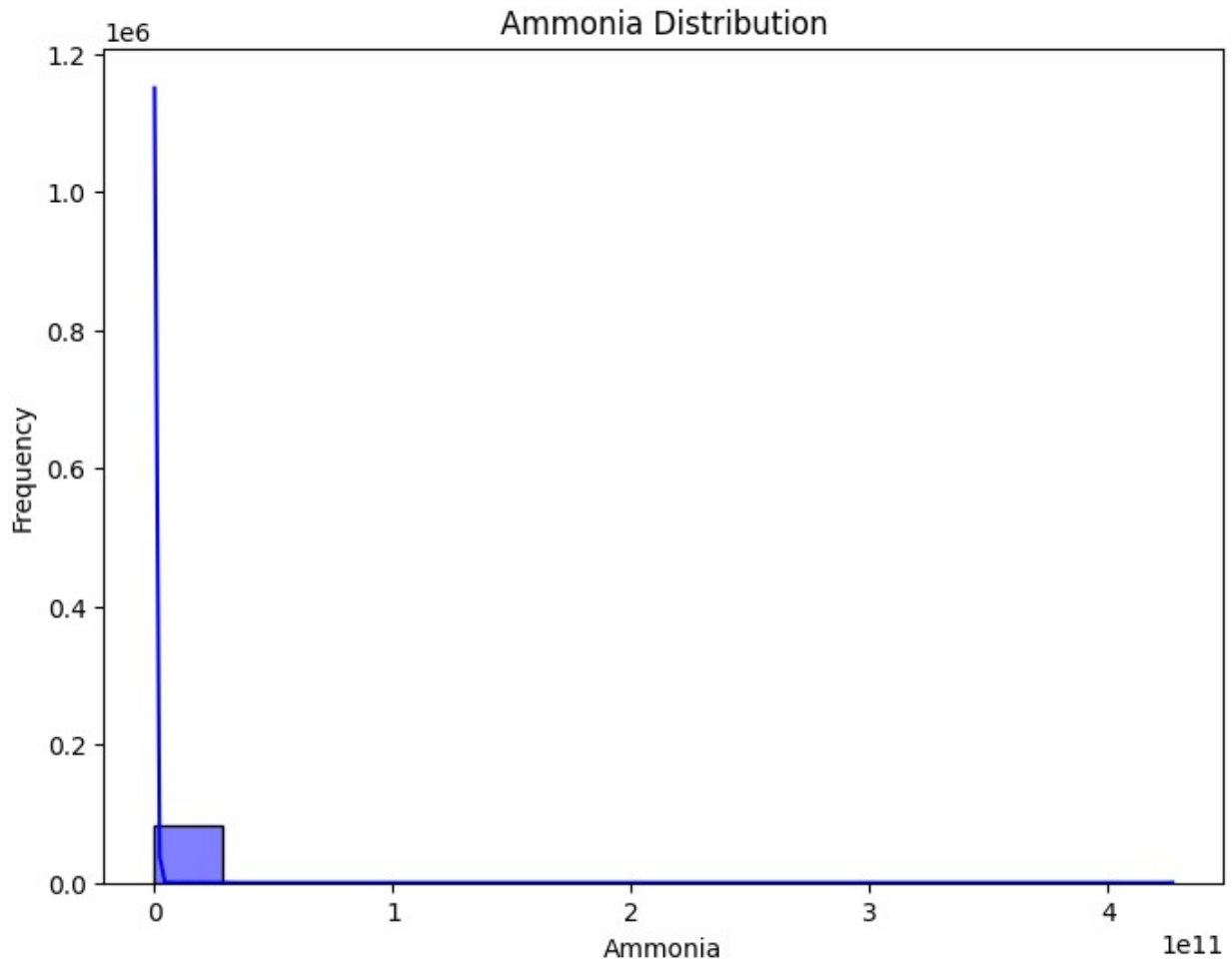
```
created_at      0
Temperature     0
Turbidity       0
Dissolved Oxygen 0
PH              0
Ammonia         52
Nitrate         0
Fish_Length     2
Fish_Weight     2
dtype: int64
```

```
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()
```



```
# 1. Distribution of Age
plt.figure(figsize=(8, 6))
sns.histplot(df['Ammonia'], 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"])
df[outliers.any(axis=1)]
```

\	Temperature	Turbidity	Dissolved Oxygen	PH	Ammonia
1801	26.2500	100	9.846	8.38371	157131.51560
1802	26.2500	100	18.915	8.38371	63147.98047
1803	26.2500	100	22.989	8.38371	1907.52869
1804	26.3125	100	10.691	8.38371	43343.46094
1805	26.3125	100	12.800	8.38825	636403.50000
...	...	...	...	...	...
82914	27.6875	100	0.086	5.79622	78.27628
82915	27.6875	100	0.311	5.79168	44.89249
82916	27.6875	100	0.832	5.79168	39.83778
82919	27.6875	100	0.224	5.69635	38.55425
82923	27.6875	100	1.584	4.33905	44.70709

	Nitrate	Fish_Length	Fish_Weight
1801	242	7.11	2.91
1802	246	7.11	2.91
1803	274	7.11	2.91
1804	273	7.11	2.91
1805	270	7.11	2.91
...	...	...	...
82914	911	33.31	313.72
82915	942	33.31	313.72
82916	945	33.31	313.72
82919	961	33.31	313.72
82923	940	33.31	313.72

```
[16818 rows x 8 columns]

df.drop(['Ammonia'], axis=1, inplace=True)

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

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

Temperature	0
Turbidity	0
Dissolved Oxygen	0
PH	0
Nitrate	0

```

Fish_Length      0
Fish_Weight      0
dtype: int64

X= df.drop(['Fish_Length', 'Fish_Weight'], axis=1)
y = df[['Fish_Length', 'Fish_Weight']]

# 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)

```

Multi-task using RandomForestRegressor

```

# Define multi-output RFR model
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()

```

Target: Fish_Length

MAE: 0.054

RMSE: 0.366

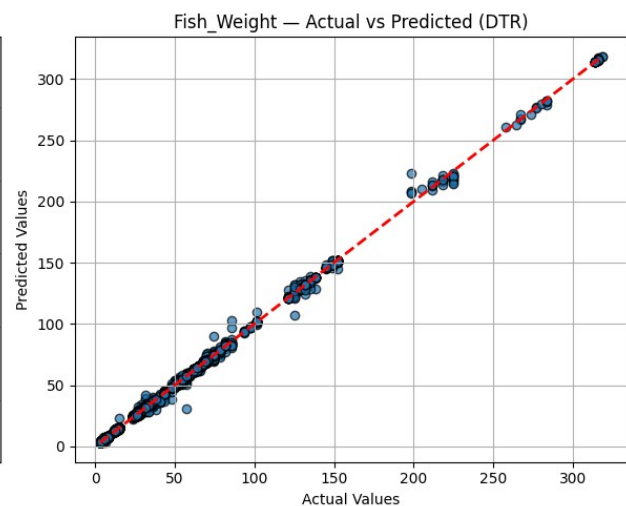
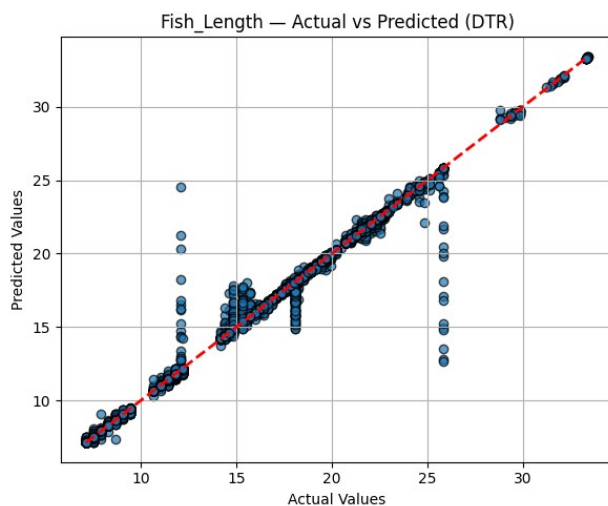
R^2 : 0.995

Target: Fish_Weight

MAE: 0.153

RMSE: 0.665

R^2 : 1.000



Multi-task using DecisionTreeRegressor

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()

```

```

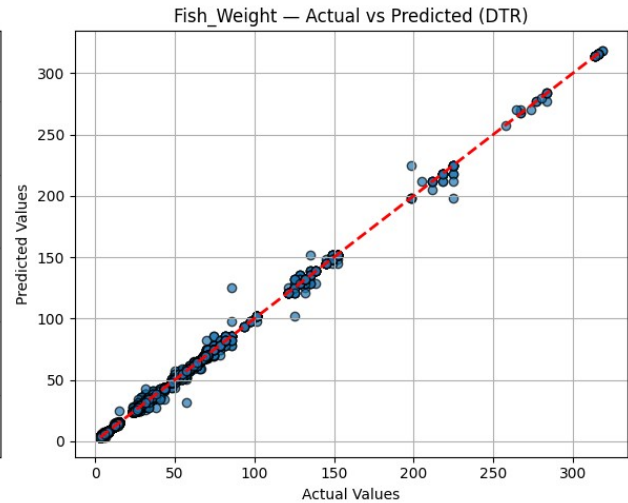
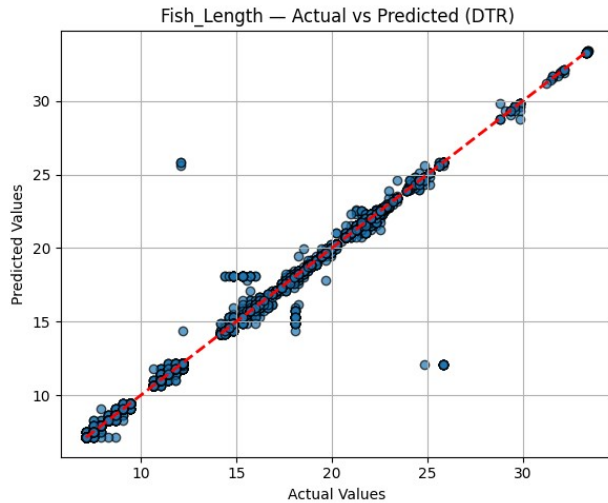
Target: Fish_Length
MAE: 0.053
RMSE: 0.467
R²: 0.992

```

```

Target: Fish_Weight
MAE: 0.143
RMSE: 0.851
R²: 0.999

```



Multi-task using SVR

```
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)

```

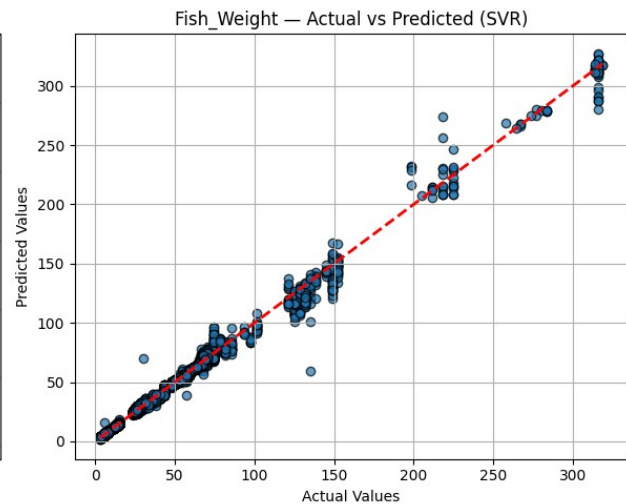
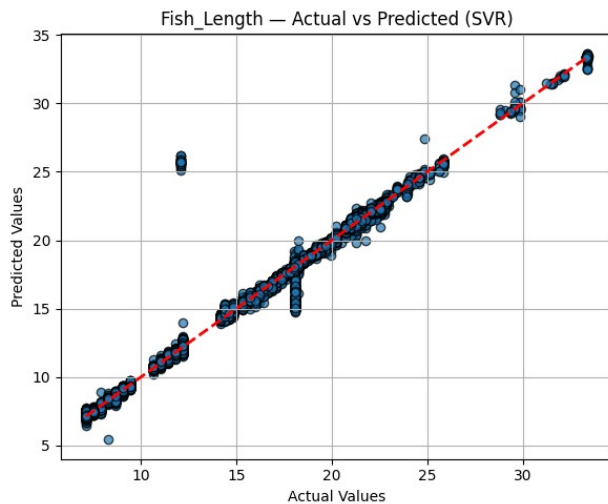
```

plt.tight_layout()
plt.show()

```

Target: Fish_Length
 MAE: 0.189
 RMSE: 0.636
 R^2 : 0.985

Target: Fish_Weight
 MAE: 0.992
 RMSE: 2.325
 R^2 : 0.995



```

# =====
# Define XGBoost base model
# =====
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

MAE : 0.122

RMSE: 0.401

R^2 : 0.994

Target: Fish_Weight

MAE : 0.462

RMSE: 1.589

R^2 : 0.998

