School of Engineering and Technology

PRACTICAL FILE

OF

Machine Learning Practical with Python, Scikit-learn, Matplotlib, TensorFlow

Lab Manual ENSP252 (2024-25)

Submitted To:
Mr. Kunal Rai
Samatrix
SOET

Submitted By:
Parth Bhardwaj
2301730289
B.Tech CSE (AIML) E

Experiments

Experiment No.	Experiment Title	Mapped CO/PO
1	Understanding Credit Card Fraud Detection Models	CO1
2	Evaluating and Improving Credit Card Fraud Detection Models	CO1, CO2
3	Understanding Baseball Player Salary Prediction Models	CO1
4	Evaluating and Optimizing Baseball Player Salary Prediction Models	CO1, CO3
5	Understanding Advertisement Budget Prediction Models	CO1
6	Evaluating and Optimizing Advertisement Budget Prediction Models	CO1, CO5
7	Understanding Diabetes Prediction Models	CO2
8	Evaluating and Optimizing Diabetes Prediction Models	CO2
9	Understanding Credit Card Default Prediction Models	CO5
10	Evaluating and Optimizing Credit Card Default Prediction Models	CO5

Experiment 1: Understanding Credit Card Fraud Detection Models

Objective:
To explore and understand how machine learning models can be used to detect fraudulent credit card transactions based on transaction data.

Description:
This experiment involves analyzing a dataset of credit card transactions, where the goal is to build and evaluate classification models capable of identifying fraudulent transactions. The data is typically highly imbalanced, with very few fraud cases compared to legitimate transactions, making it a classic example for applying techniques like:

- Data preprocessing (normalization, handling imbalance using SMOTE/undersampling)
- Model training (Logistic Regression, Decision Trees, Random Forest, XGBoost, etc.)
- Evaluation metrics (Precision, Recall, F1-score, ROC-AUC, Confusion Matrix)

Tools/Technologies Used:

- Python
- Scikit-learn
- Pandas, NumPy, Matplotlib/Seaborn
- Jupyter Notebook **Dataset:**
  A popular dataset used for this task is the **"Credit Card Fraud Detection"** dataset available on Kaggle, which contains anonymized features (V1-V28), along with Time, Amount, and Class (1 for fraud, 0 for non-fraud).

Expected Outcomes:

- Visualizations showing class imbalance.
- Performance comparison between different models.
- Understanding the trade-off between Precision and Recall in fraud detection.

Learning Outcomes

By completing this experiment, the following learning outcomes were achieved:

1. Understanding of Fraud Detection Challenges
2. Data Preprocessing Skills
3. Model Implementation and Evaluation
4. Critical Thinking in Model Selection
5. Practical Use of Python Libraries
6. Improved Data Visualization Skills

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE

# Load dataset
df = pd.read_csv('creditcard.csv')

# Check for missing values
print("Missing values per column:\n", df.isnull().sum())

# Impute missing numerical values with mean
imputer = SimpleImputer(strategy='mean')
df[df.select_dtypes(include=[np.number]).columns] =
imputer.fit_transform(df.select_dtypes(include=[np.number]))
```

```
# Remove outliers in 'Amount' column using IQR
Q1 = df['Amount'].quantile(0.25)
Q3 = df['Amount'].quantile(0.75)
IQR = Q3 - Q1
df = df[~((df['Amount'] < (Q1 - 1.5 * IQR)) | (df['Amount'] > (Q3 + 1.5 * IQR)))]

# Sort by time and create rolling average of 'Amount'
df = df.sort_values('Time')
df['rolling_avg_amount'] = df['Amount'].rolling(window=3, min_periods=1).mean()

# Select features and label
selected_features = ['Amount', 'Time', 'rolling_avg_amount'] + [f'V{i}' for i in range(1, 5)]
df = df[selected_features + ['Class']]  # 'Class' is the target

# Feature scaling
scaler = StandardScaler()
df[selected_features] = scaler.fit_transform(df[selected_features])

# Split into features and target
X = df.drop(columns=['Class'])
y = df['Class']

# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

```
# Train Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print(classification_report(y_test, y_pred))
```

Experiment 2: Evaluating and Improving Credit Card Fraud Detection Models

---

Introduction to Model Evaluation and Optimization

In this experiment, participants will evaluate the performance of their credit card fraud detection model. They will also learn techniques to improve the model's accuracy, handle imbalanced data, and ensure the model performs well in a real-world setting.

Key Concepts in Model Evaluation and Optimization:

- **Model Evaluation Metrics**: Precision, recall, accuracy, and F1-score.
- **Cross-Validation**: Using k-fold cross-validation to evaluate model performance.
- **Handling Imbalanced Data**: Techniques like oversampling, undersampling, and synthetic data generation.
- **Model Optimization**: Hyperparameter tuning and feature selection.

Objective

- **Evaluate the performance** of the fraud detection model.
- **Handle imbalanced datasets** using SMOTE or other techniques.
- **Optimize the model** for better performance using cross-validation and hyperparameter tuning.

Learning Outcomes

By the end of this experiment, participants will be able to:

1. **Evaluate the performance** of their fraud detection model using appropriate metrics.
2. **Apply techniques** to handle imbalanced datasets.
3. **Optimize the model** using cross-validation and hyperparameter tuning.

Instructions for Conducting the Experiment

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (

  accuracy_score, precision_score, recall_score, f1_score,

  classification_report, confusion_matrix

)

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

from imblearn.over_sampling import SMOTE


# Load dataset

df = pd.read_csv('creditcard.csv')


# Handle missing values

imputer = SimpleImputer(strategy='mean')

df[df.select_dtypes(include=[np.number]).columns] =
imputer.fit_transform(df.select_dtypes(include=[np.number]))


# Remove outliers from 'Amount'
```

```
Q1 = df['Amount'].quantile(0.25)
Q3 = df['Amount'].quantile(0.75)
IQR = Q3 - Q1
df = df[~((df['Amount'] < (Q1 - 1.5 * IQR)) | (df['Amount'] > (Q3 + 1.5 * IQR)))]

# Sort and create rolling average
df = df.sort_values('Time')
df['rolling_avg_amount'] = df['Amount'].rolling(window=3, min_periods=1).mean()

# Select features and target
selected_features = ['Amount', 'Time', 'rolling_avg_amount'] + [f'V{i}' for i in range(1, 5)]
df = df[selected_features + ['Class']]
scaler = StandardScaler()
df[selected_features] = scaler.fit_transform(df[selected_features])

X = df.drop(columns=['Class'])
y = df['Class']

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

# Initial model training
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluation
y_pred = model.predict(X_test)
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')
f1 = f1_score(y_test, y_pred, average='weighted')
```

```
print("=== Initial Evaluation ===")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
    xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Apply SMOTE to handle imbalance
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Re-train model on balanced data
model_smote = RandomForestClassifier(n_estimators=100, random_state=42)
model_smote.fit(X_resampled, y_resampled)

# Evaluate after SMOTE
y_pred_smote = model_smote.predict(X_test)
print("\n=== Evaluation After SMOTE ===")
print(classification_report(y_test, y_pred_smote))
```

```
# Cross-validation

cv_scores = cross_val_score(model_smote, X_resampled, y_resampled, cv=5,
scoring='accuracy')

print(f"Cross-Validation Accuracy Scores: {cv_scores}")

print(f"Mean Accuracy: {np.mean(cv_scores):.4f}")


# Grid Search for Hyperparameter Tuning

param_grid = {

  'n_estimators': [50, 100, 200],

  'max_depth': [None, 10, 20],

  'min_samples_split': [2, 5, 10]

}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
cv=3,

                scoring='accuracy', n_jobs=-1)

grid_search.fit(X_resampled, y_resampled)


# Train optimized model

print("\nBest Hyperparameters:", grid_search.best_params_)

optimized_model = grid_search.best_estimator_

optimized_model.fit(X_resampled, y_resampled)


# Evaluate optimized model

y_pred_optimized = optimized_model.predict(X_test)

print("\n=== Evaluation After Hyperparameter Tuning ===")

print(classification_report(y_test, y_pred_optimized))
```

Experiment 3: Understanding Baseball Player Salary Prediction Models

---

Introduction to Salary Prediction in Sports Analytics

Predicting the salary of baseball players is a key application of machine learning in sports analytics. In this experiment, you'll build a regression model to forecast player salaries based on factors like performance stats, team success, and player demographics.

Key Concepts in Baseball Player Salary Prediction

- **Data Collection:** Historical datasets of player performance, salaries, and team metrics.
- **Feature Engineering:** Creating meaningful predictors (e.g., batting average, home runs, ERA).
- **Regression Models:** Algorithms such as Linear Regression, Decision Trees, and Random Forest Regression.
- **Data Splitting:** Dividing data into training and testing subsets for unbiased evaluation.

Objective

1. Understand the components of a baseball salary prediction model.
2. Explore which factors most influence player salaries.
3. Apply various regression techniques to predict salaries.

Learning Outcomes

By the end of this experiment, you will be able to:

1. Interpret how player and team statistics impact salary predictions.

2. Engineer new features to boost model performance.
3. Train and compare regression models for continuous-value prediction.
4. Evaluate model accuracy with metrics like MAE, MSE, and R^2.

Instructions for Conducting the Experiment

```
# Import libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.impute import SimpleImputer


# Step 1: Load dataset

df = pd.read_csv('baseball_data.csv')  # Replace with your actual dataset file


# Step 2: Handle missing values

print("Missing values per column:\n", df.isnull().sum())

imputer = SimpleImputer(strategy='mean')

df[df.select_dtypes(include=[np.number]).columns] =
imputer.fit_transform(df.select_dtypes(include=[np.number]))


# Drop irrelevant columns

df.drop(columns=['player_id', 'player_name'], errors='ignore', inplace=True)


# Step 3: Remove outliers using IQR

Q1 = df.quantile(0.25)
```

```
Q3 = df.quantile(0.75)

IQR = Q3 - Q1

df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]


# Step 4: Normalize numeric features

numeric_features = ['batting_average', 'home_runs', 'RBIs', 'OPS']  # Modify as per your
dataset

scaler = StandardScaler()

df[numeric_features] = scaler.fit_transform(df[numeric_features])


# Step 5: Feature engineering

selected_features = ['age', 'team_wins', 'batting_average', 'home_runs', 'RBIs']

df['recent_performance'] = df['batting_average'] * 0.5 + df['home_runs'] * 0.3 + df['RBIs']
* 0.2


# Step 6: Define target and features

target_variable = 'player_salary'  # Replace if your dataset uses a different column name

X = df[selected_features + ['recent_performance']]

y = df[target_variable]


# Step 7: Split data

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


# Step 8: Train model

model = RandomForestRegressor(n_estimators=100, random_state=42)

model.fit(X_train, y_train)


# Step 9: Predictions and evaluation

y_pred = model.predict(X_test)
```

```
mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print("=== Model Evaluation ===")

print(f"Mean Absolute Error: {mae:.4f}")

print(f"Mean Squared Error: {mse:.4f}")

print(f"R² Score: {r2:.4f}")
```

Experiment 4: Evaluating and Optimizing Baseball Player Salary Prediction Models

---

Introduction to Model Evaluation and Optimization

In this experiment, you will evaluate and improve your baseball salary prediction model using metrics, multicollinearity analysis, cross-validation, and hyperparameter tuning.

Key Concepts

- **Model Evaluation Metrics:** R^2, MSE, RMSE, and MAE.
- **Multicollinearity:** Removing correlated features to avoid redundancy.
- **Model Optimization:** Improving accuracy using cross-validation and grid search.

Objective

1. Evaluate model performance using standard metrics.
2. Identify and handle multicollinearity.
3. Use hyperparameter tuning to improve model accuracy.

Learning Outcomes

After completing this experiment, you will be able to:

1. Use key metrics to evaluate regression models.
2. Reduce model overfitting by removing multicollinearity.

3. Optimize regression models using grid search and cross-validation.

Instructions for Conducting the Experiment

```
# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import cross_val_score, GridSearchCV

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor


# Assuming model is trained from Experiment 3 and X_train, X_test, y_train, y_test are already available


# 2. Model Evaluation

y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

r2 = r2_score(y_test, y_pred)


print("=== Initial Model Evaluation ===")

print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"R² Score: {r2:.4f}")
```

```
# Residual Analysis
residuals = y_test - y_pred
plt.figure(figsize=(8, 5))
sns.histplot(residuals, kde=True, bins=30)
plt.title("Residual Distribution")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()

plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.title("Residual Plot")
plt.xlabel("Predicted Salary")
plt.ylabel("Residuals")
plt.show()

# 3. Handling Multicollinearity using VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in
range(X_train.shape[1])]
print("\n=== Variance Inflation Factor (VIF) ===")
print(vif_data)

# Drop features with VIF > 5
high_vif_features = vif_data[vif_data["VIF"] > 5]["Feature"].tolist()
```

```
X_train_reduced = X_train.drop(columns=high_vif_features)

X_test_reduced = X_test.drop(columns=high_vif_features)


# Retrain model

model_reduced = RandomForestRegressor(n_estimators=100, random_state=42)

model_reduced.fit(X_train_reduced, y_train)

y_pred_reduced = model_reduced.predict(X_test_reduced)

print("\n=== After Removing High VIF Features ===")

print(f"New R² Score: {r2_score(y_test, y_pred_reduced):.4f}")


# 4. Cross-Validation

cv_scores = cross_val_score(model_reduced, X_train_reduced, y_train, cv=5,
scoring='r2')

print("\n=== Cross-Validation ===")

print(f"R² Scores: {cv_scores}")

print(f"Mean R² Score: {np.mean(cv_scores):.4f}")


# 5. Hyperparameter Tuning with GridSearchCV

param_grid = {

  'n_estimators': [50, 100, 200],

  'max_depth': [None, 10, 20],

  'min_samples_split': [2, 5, 10]

}

grid_search = GridSearchCV(

  RandomForestRegressor(random_state=42),

  param_grid,

  cv=3,

  scoring='r2',
```

```
    n_jobs=-1
)
grid_search.fit(X_train_reduced, y_train)
print("\n=== Best Hyperparameters ===")
print(grid_search.best_params_)

# Train optimized model
optimized_model = grid_search.best_estimator_
optimized_model.fit(X_train_reduced, y_train)
y_pred_optimized = optimized_model.predict(X_test_reduced)

print("\n=== Optimized Model Evaluation ===")
print(f"Optimized R² Score: {r2_score(y_test, y_pred_optimized):.4f}")
```

Experiment 5: Understanding Advertisement Budget Prediction Models

---

Introduction to Advertisement Budget and Sales Prediction

In this experiment, participants will explore the relationship between advertisement budgets across various media—TV, Radio, and Newspaper—and their effect on product sales. By using historical data and regression models, the goal is to predict sales based on media spending and determine which platforms offer the highest return on investment.

Key Concepts in Advertisement Budget Prediction

- **Data Collection**: Gathering historical advertising budget and sales data.
- **Linear Regression**: Understanding how multiple inputs (TV, Radio, Newspaper) influence a single output (Sales).

- **Feature Engineering**: Creating new features like percentage contributions of each medium.
- **Multivariable Regression**: Building models with multiple independent variables.
- **Multicollinearity Detection**: Identifying highly correlated features using VIF (Variance Inflation Factor).

Objective

- Understand how media budget allocations affect sales.
- Apply multivariable regression to predict sales.
- Evaluate and improve model accuracy using standard metrics and VIF.

Learning Outcomes

By the end of this experiment, participants will be able to:

1. Analyze how advertisement budgets in different media influence sales outcomes.
2. Build and evaluate multiple linear regression models.
3. Implement feature engineering techniques for improved insights.
4. Assess model performance using R^2, MSE, RMSE, and MAE.
5. Detect and resolve multicollinearity to improve model robustness.

Instructions for Conducting the Experiment

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.stats.outliers_influence import variance_inflation_factor

# 1. Load and Inspect Dataset
df = pd.read_csv('advertising.csv')  # Replace with actual path
print("Missing values per column:\n", df.isnull().sum())

# 2. Handle Missing Values
df.fillna(df.mean(numeric_only=True), inplace=True)

# 3. Remove Outliers using IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]

# 4. Normalize Budget Features
scaler = StandardScaler()
budget_features = ['TV', 'Radio', 'Newspaper']
df[budget_features] = scaler.fit_transform(df[budget_features])

# 5. Feature Engineering: Budget Proportions
df['TV_pct'] = df['TV'] / (df['TV'] + df['Radio'] + df['Newspaper'])
df['Radio_pct'] = df['Radio'] / (df['TV'] + df['Radio'] + df['Newspaper'])
df['Newspaper_pct'] = df['Newspaper'] / (df['TV'] + df['Radio'] + df['Newspaper'])

# Optional: Encode time-based cyclic features if applicable
```

```
if 'Month' in df.columns:
    df['Month_sin'] = np.sin(2 * np.pi * df['Month'] / 12)
    df['Month_cos'] = np.cos(2 * np.pi * df['Month'] / 12)

# Define target and features
target_variable = 'Sales'
features = ['TV', 'Radio', 'Newspaper', 'TV_pct', 'Radio_pct', 'Newspaper_pct']
if 'Month_sin' in df.columns:
    features += ['Month_sin', 'Month_cos']

X = df[features]
y = df[target_variable]

# 6. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 7. Train Multiple Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# 8. Predictions and Evaluation
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("=== Initial Model Evaluation ===")
```

```
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R² Score: {r2:.4f}")

# 9. Multicollinearity Check with VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in
range(X_train.shape[1])]

print("\n=== Variance Inflation Factor (VIF) ===")
print(vif_data)

# 10. Remove High VIF Features (>5) and Retrain
high_vif_features = vif_data[vif_data["VIF"] > 5]["Feature"].tolist()
X_train_reduced = X_train.drop(columns=high_vif_features)
X_test_reduced = X_test.drop(columns=high_vif_features)

model_reduced = LinearRegression()
model_reduced.fit(X_train_reduced, y_train)
y_pred_reduced = model_reduced.predict(X_test_reduced)

print("\n=== Evaluation After Removing High VIF Features ===")
print(f"New R² Score: {r2_score(y_test, y_pred_reduced):.4f}")
```

Experiment 6: Evaluating and Optimizing Advertisement Budget Prediction Models

---

Introduction to Model Evaluation and Optimization

In this experiment, participants will evaluate and optimize their advertisement budget prediction models. The focus will be on understanding how to assess model performance, identify issues such as multicollinearity, and use optimization techniques to enhance prediction accuracy.

Key Concepts in Model Evaluation and Optimization

- **Model Evaluation Metrics**: Common metrics such as R-squared, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).
- **Multicollinearity**: Identifying and addressing high correlation between predictor variables (e.g., budgets for TV, Radio, and Newspaper).
- **Model Optimization**: Techniques like feature selection and hyperparameter tuning to improve the model's predictive power.

Objective

- To evaluate the performance of the advertisement budget prediction model.
- To address issues like multicollinearity that may affect the model.
- To optimize the model for more accurate and precise sales predictions.

Learning Outcomes

By the end of this experiment, participants will be able to:

1. Evaluate the model's performance using metrics such as R^2, MSE, and RMSE.
2. Identify and address multicollinearity in the regression model.
3. Optimize the model for better accuracy using techniques like feature selection and cross-validation.

Instructions for Conducting the Experiment

Required Libraries

import pandas as pd

import numpy as np

```
import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.feature_selection import RFE

from statsmodels.stats.outliers_influence import variance_inflation_factor


# 1. Assuming the DataFrame 'df' is preprocessed from previous experiment

# Select features and target variable

target_variable = 'Sales'

selected_features = ['TV', 'Radio', 'Newspaper', 'TV_pct', 'Radio_pct', 'Newspaper_pct']

X = df[selected_features]

y = df[target_variable]


# Train-Test Split

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


# Train Initial Model

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)


# 2. Model Evaluation

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)
```

```
r2 = r2_score(y_test, y_pred)

print("=== Initial Model Evaluation ===")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R² Score: {r2:.4f}")

# Residual Analysis
residuals = y_test - y_pred

plt.figure(figsize=(8, 5))
sns.histplot(residuals, kde=True, bins=30)
plt.xlabel("Residuals")
plt.title("Residual Distribution")
plt.show()

plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel("Predicted Sales")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()

# 3. Multicollinearity Check
plt.figure(figsize=(8, 6))
sns.heatmap(df[selected_features].corr(), annot=True, cmap="coolwarm", fmt=".2f")
```

```python
plt.title("Feature Correlation Matrix")
plt.show()

# Calculate VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
print("\n=== VIF Values ===")
print(vif_data)

# Remove features with VIF > 5
high_vif_features = vif_data[vif_data["VIF"] > 5]["Feature"].tolist()
X_train_reduced = X_train.drop(columns=high_vif_features)
X_test_reduced = X_test.drop(columns=high_vif_features)

# Retrain Reduced Model
model_reduced = LinearRegression()
model_reduced.fit(X_train_reduced, y_train)
y_pred_reduced = model_reduced.predict(X_test_reduced)

print("\n=== Evaluation After Removing High-VIF Features ===")
print(f"New R² Score: {r2_score(y_test, y_pred_reduced):.4f}")

# 4. Cross-Validation
cv_scores = cross_val_score(model_reduced, X_train_reduced, y_train, cv=5, scoring='r2')
print("\nCross-Validation R² Scores:", cv_scores)
print(f"Mean R² Score: {np.mean(cv_scores):.4f}")
```

```
# 5. Model Optimization using RFE
rfe = RFE(model_reduced, n_features_to_select=3)
rfe.fit(X_train_reduced, y_train)
rfe_selected_features = X_train_reduced.columns[rfe.support_]

print("\nSelected Features after RFE:", list(rfe_selected_features))

X_train_final = X_train_reduced[rfe_selected_features]
X_test_final = X_test_reduced[rfe_selected_features]

# Retrain Optimized Model
optimized_model = LinearRegression()
optimized_model.fit(X_train_final, y_train)
y_pred_optimized = optimized_model.predict(X_test_final)

# Final Evaluation
print("\n=== Final Optimized Model Evaluation ===")
print(f"Optimized R² Score: {r2_score(y_test, y_pred_optimized):.4f}")
```

Experiment 7: Understanding Diabetes Prediction Models

Introduction to Diabetes Prediction in Healthcare

In this experiment, participants will learn how to build predictive models to assess the likelihood of diabetes based on health metrics. The goal is to analyze various health factors (e.g., age, BMI, glucose levels, family history) and predict diabetes using machine learning models. This task involves creating models that predict whether a person is likely to develop diabetes, given their medical records.

Key Concepts in Diabetes Prediction

- **Data Collection**: Gathering patient data, including demographics, health metrics, and medical history related to diabetes.
- **Feature Engineering**: Selecting key features such as BMI, age, glucose levels, and family history of diabetes.
- **Classification Models**: Using classification algorithms like Logistic Regression, Decision Trees, and Random Forest to predict the presence of diabetes (binary classification).
- **Data Preprocessing**: Handling missing values, scaling numerical features, and encoding categorical variables.

Objective

- To understand the significance of health metrics in predicting diabetes.
- To explore classification models for diabetes prediction.
- To evaluate the performance of the diabetes prediction model.

Learning Outcomes

By the end of this experiment, participants will be able to:

1. Understand the relationship between health features and diabetes risk.
2. Apply classification models such as Logistic Regression and Decision Trees.
3. Implement preprocessing techniques to handle missing data and scale features.
4. Evaluate the model's performance using accuracy, precision, recall, and F1-score.

Instructions for Conducting the Experiment

```
# Required Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 1. Data Preprocessing
# Load dataset (replace 'diabetes.csv' with actual dataset)
df = pd.read_csv('diabetes.csv')

# Inspect for missing values
print("Missing values per column:\n", df.isnull().sum())

# Handle missing values (impute numerical features with mean)
num_imputer = SimpleImputer(strategy="mean")
numerical_features = ['Glucose', 'BMI', 'BloodPressure', 'Insulin']
df[numerical_features] = num_imputer.fit_transform(df[numerical_features])

# Detect and remove outliers using IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]

# Scale numerical features
scaler = StandardScaler()
```

```
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# 2. Feature Engineering
# Select relevant features
selected_features = ['Age', 'BMI', 'Glucose', 'BloodPressure', 'Insulin',
'DiabetesPedigreeFunction']

# Encode categorical variables (if present)
if 'Gender' in df.columns:
    df['Gender'] = LabelEncoder().fit_transform(df['Gender'])
    selected_features.append('Gender')

# Define target variable
target_variable = 'Outcome'  # Assuming "Outcome" is the label for diabetes (0 = No, 1 =
Yes)

# 3. Model Selection
# Split data into features and target
X = df[selected_features]
y = df[target_variable]

# Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Choose classification models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
```

```
    "Random Forest": RandomForestClassifier()
}

# 4. Training the Model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Model Evaluation
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\n{name} Accuracy: {accuracy:.4f}")
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues",
                xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"{name} - Confusion Matrix")
    plt.show()

# 5. Hyperparameter Tuning for Best Model (Example: Random Forest)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10]
}
```

```
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5,
                    scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

print("\nOptimized Random Forest Model Accuracy:", accuracy_score(y_test,
y_pred_best))
print(classification_report(y_test, y_pred_best))
```

Experiment 8: Evaluating and Optimizing Diabetes Prediction Models

---

Introduction to Model Evaluation and Optimization

In this experiment, participants will evaluate the performance of their diabetes prediction model and apply optimization techniques to improve it. They will learn about evaluation metrics, how to address overfitting and underfitting, and apply hyperparameter tuning to enhance the model's accuracy and generalization ability.

Key Concepts in Model Evaluation and Optimization

- **Model Evaluation Metrics**: Understand metrics like Accuracy, Precision, Recall, F1-score, and ROC-AUC to assess the model's performance.
- **Cross-Validation**: Implement k-fold cross-validation to check the model's ability to generalize.
- **Hyperparameter Tuning**: Optimize the model by fine-tuning its parameters using techniques like Grid Search.
- **Overfitting and Underfitting**: Learn how to identify and mitigate overfitting and underfitting issues in the model.

Objective

- Evaluate the performance of the diabetes prediction model.
- Optimize the model to improve accuracy and generalization.
- Address overfitting and underfitting by tuning hyperparameters.

Learning Outcomes

By the end of this experiment, participants will be able to:

1. Evaluate the model using performance metrics such as accuracy, precision, recall, and F1-score.
2. Address issues of overfitting and underfitting by observing and interpreting learning curves.
3. Optimize the model's performance through techniques like cross-validation and hyperparameter tuning.

Instructions for Conducting the Experiment

Required Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, learning_curve

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, roc_curve

from sklearn.ensemble import RandomForestClassifier

1. Preparation

Load dataset (replace 'diabetes.csv' with actual dataset)

df = pd.read_csv('diabetes.csv')

```
# Select features and target variable
selected_features = ['Age', 'BMI', 'Glucose', 'BloodPressure', 'Insulin', 'DiabetesPedigreeFunction']
target_variable = 'Outcome'  # Assuming "Outcome" is the label for diabetes (0 = No, 1 = Yes)

X = df[selected_features]
y = df[target_variable]

# Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initialize a Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 2. Model Evaluation
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]  # Probability scores for ROC-AUC

# Compute evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)
```

```
print(f"\nAccuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC Score: {roc_auc:.4f}")

# Confusion Matrix
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues",
    xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()

# 3. Cross-Validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
```

```
print(f"\nCross-Validation Accuracy Scores: {cv_scores}")

print(f"Mean CV Accuracy: {np.mean(cv_scores):.4f}")


# 4. Hyperparameter Tuning using GridSearchCV

param_grid = {

  'n_estimators': [50, 100, 200],

  'max_depth': [5, 10, 15],

  'min_samples_split': [2, 5, 10]

}


grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
cv=5,

             scoring='accuracy', n_jobs=-1)

grid_search.fit(X_train, y_train)


# Best model after tuning

best_model = grid_search.best_estimator_

y_pred_best = best_model.predict(X_test)


print("\nOptimized Model Performance:")

print(f"Optimized Accuracy: {accuracy_score(y_test, y_pred_best):.4f}")

print(classification_report(y_test, y_pred_best))


# 5. Addressing Overfitting/Underfitting - Learning Curve

train_sizes, train_scores, test_scores = learning_curve(best_model, X_train, y_train,
cv=5,

                          scoring="accuracy", train_sizes=np.linspace(0.1, 1.0, 5))

train_mean = np.mean(train_scores, axis=1)

test_mean = np.mean(test_scores, axis=1)
```

```
plt.figure(figsize=(7, 5))

plt.plot(train_sizes, train_mean, label="Training Accuracy", marker='o')

plt.plot(train_sizes, test_mean, label="Validation Accuracy", marker='s')

plt.xlabel("Training Set Size")

plt.ylabel("Accuracy")

plt.title("Learning Curve")

plt.legend()

plt.show()
```

Experiment 9: Understanding Credit Card Default Prediction Models

---

Introduction to Credit Card Default Prediction

In this experiment, participants will learn how to build and understand predictive models for credit card default prediction based on user behavior and financial data. The goal is to analyze the relationship between various features such as payment history, balance, credit limit, and the likelihood of defaulting on a credit card. The task involves using machine learning models to predict whether a credit card user will default based on their financial history.

Key Concepts in Credit Card Default Prediction

- **Data Collection**: Gathering data on credit card users' demographics, payment history, credit limits, etc.
- **Feature Engineering**: Selecting important features such as payment status, credit utilization, and previous defaults.
- **Classification Models**: Using classification algorithms (e.g., Logistic Regression, Random Forest) to predict whether a user will default or not (binary classification).
- **Data Preprocessing**: Handling missing values, encoding categorical data, and scaling features.

Objective

- To understand the importance of various financial factors in predicting credit card default.
- To explore classification models for predicting the likelihood of default.
- To evaluate the performance of a predictive model for credit card default prediction.

Learning Outcomes

By the end of this experiment, participants will be able to:

1. Understand how financial features (e.g., credit history, payment status) impact credit card default.
2. Apply classification models like Logistic Regression and Random Forest.
3. Implement preprocessing techniques such as handling missing values and scaling features.
4. Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1-score.

Instructions for Conducting the Experiment

```
# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.impute import SimpleImputer

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

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

```python
# Load dataset (replace 'credit_data.csv' with actual dataset)
df = pd.read_csv('credit_data.csv')

# Data Preprocessing
# Check for missing values
print("Missing values per column:\n", df.isnull().sum())

# Handle missing values (impute numerical features with mean)
num_imputer = SimpleImputer(strategy="mean")
numerical_features = ['Balance', 'Income', 'Credit_Limit']
df[numerical_features] = num_imputer.fit_transform(df[numerical_features])

# Detect and remove outliers using IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]

# Scale numerical features
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# Feature Engineering
# Select relevant features
selected_features = ['Payment_History', 'Age', 'Income', 'Balance', 'Credit_Limit',
'Previous_Defaults']
```

```
# Encode categorical variables (if present)
if 'Gender' in df.columns:
    df['Gender'] = LabelEncoder().fit_transform(df['Gender'])
    selected_features.append('Gender')
if 'Education_Level' in df.columns:
    df['Education_Level'] = LabelEncoder().fit_transform(df['Education_Level'])
    selected_features.append('Education_Level')

# Define target variable
target_variable = 'Default'  # Assuming "Default" is the label (0 = No Default, 1 = Default)

# Split data into features and target
X = df[selected_features]
y = df[target_variable]

# Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Choose classification models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier()
}

# Training and evaluation of models
for name, model in models.items():
```

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

    # Model Evaluation
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\n{name} Accuracy: {accuracy:.4f}")
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues",
                xticklabels=["No Default", "Default"], yticklabels=["No Default", "Default"])
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"{name} - Confusion Matrix")
    plt.show()

# Hyperparameter Tuning for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
# Best model after tuning

best_model = grid_search.best_estimator_

y_pred_best = best_model.predict(X_test)


print("\nOptimized Random Forest Model Accuracy:", accuracy_score(y_test, y_pred_best))

print(classification_report(y_test, y_pred_best))
```

Experiment 10: Evaluating and Optimizing Credit Card Default Prediction Models

---

Introduction to Model Evaluation and Optimization

In this experiment, participants will evaluate the performance of their credit card default prediction model and apply optimization techniques to improve it. They will learn about evaluation metrics, overfitting/underfitting issues, and hyperparameter tuning to enhance the model's predictive accuracy.

Key Concepts in Model Evaluation and Optimization

- **Model Evaluation Metrics**: Accuracy, Precision, Recall, F1-score, ROC-AUC.
- **Cross-Validation**: Using k-fold cross-validation to assess model generalizability.
- **Hyperparameter Tuning**: Optimizing the model by fine-tuning its parameters.
- **Overfitting and Underfitting**: Identifying and addressing overfitting or underfitting in the model.

Objective

- To evaluate the performance of the credit card default prediction model.
- To optimize the model for better accuracy and generalization.
- To prevent overfitting and underfitting by tuning hyperparameters.

Learning Outcomes

By the end of this experiment, participants will be able to:

1. Evaluate the model using various metrics like accuracy, precision, and recall.
2. Address issues related to overfitting and underfitting.
3. Optimize the model's performance using techniques like cross-validation and hyperparameter tuning.

Instructions for Conducting the Experiment

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, learning_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report, confusion_matrix, roc_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Data Preprocessing (assume the data is already cleaned as in previous steps)
# Load dataset (replace 'credit_data.csv' with actual dataset)
df = pd.read_csv('credit_data.csv')

# Define features and target variable
target_variable = 'Default'  # Assuming "Default" is the label (0 = No Default, 1 = Default)
selected_features = ['Payment_History', 'Age', 'Income', 'Balance', 'Credit_Limit', 'Previous_Defaults']

X = df[selected_features]
y = df[target_variable]
```

```
# Train-test split (80% training, 20% testing)

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


# 1. Model Evaluation

# Initialize a Random Forest model

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)


# Predictions and probability scores

y_pred = model.predict(X_test)

y_prob = model.predict_proba(X_test)[:, 1]  # Probability scores for ROC-AUC


# Compute evaluation metrics

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred)

roc_auc = roc_auc_score(y_test, y_prob)


# Print evaluation metrics

print(f"\nAccuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F1 Score: {f1:.4f}")

print(f"ROC-AUC Score: {roc_auc:.4f}")
```

```python
# Confusion Matrix
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues",
            xticklabels=["No Default", "Default"], yticklabels=["No Default", "Default"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.4f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()

# 2. Cross-Validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
print(f"\nCross-Validation Accuracy Scores: {cv_scores}")
print(f"Mean CV Accuracy: {np.mean(cv_scores):.4f}")

# 3. Hyperparameter Tuning using GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
```

```
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Best model after tuning
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

print("\nOptimized Model Performance:")
print(f"Optimized Accuracy: {accuracy_score(y_test, y_pred_best):.4f}")
print(classification_report(y_test, y_pred_best))

# 4. Addressing Overfitting/Underfitting - Learning Curve
train_sizes, train_scores, test_scores = learning_curve(best_model, X_train, y_train,
cv=5,
                                                        scoring="accuracy", train_sizes=np.linspace(0.1, 1.0, 5))
train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

plt.figure(figsize=(7, 5))
plt.plot(train_sizes, train_mean, label="Training Accuracy", marker='o')
plt.plot(train_sizes, test_mean, label="Validation Accuracy", marker='s')
plt.xlabel("Training Set Size")
plt.ylabel("Accuracy")
plt.title("Learning Curve")
```

```
plt.legend()

plt.show()
```

Project: Student Academic Performance Prediction

Parth Bhardwaj

2301730289

B.Tech CSE (AIML) - E

Problem Statement:

The task is to predict a student's final grades based on various factors like demographic details, family background, and school-related features. The objective is to build a model that can accurately predict a student's academic performance based on these inputs.

Dataset: Student Performance Dataset

The **Student Performance Dataset** typically contains features such as:

- **Demographic Information**: Age, gender, and study time.
- **Family Background**: Parent education, family relationship, and socioeconomic status.
- **School-related Features**: Attendance, previous grades, and extracurricular activities.

The target variable is the **final grade** or **academic performance** of the student, which could be either continuous (e.g., a grade score) or categorical (e.g., pass/fail).

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.axes import Axes
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

[1]

1. Understanding the Data

```
print('min age of student:', min(data['age']))
print('max age of student:',max(data['age']))
```

[ ]

```
min age of student: 15
max age of student: 22
```

```
# Checking the shape i.e number of rows & columns
df.shape
```

[ ]

```
(395, 33)
```

```
df['absences'].unique()
```

[ ]

```
array([ 6,  4, 10,  2,  0, 16, 14,  7,  8, 25, 12, 54, 18, 26, 20, 56, 24,
       28,  5, 13, 15, 22,  3, 21,  1, 75, 30, 19,  9, 11, 38, 40, 23, 17])
```

```python
# Checking the average scores
print('mean of G1:', data['G1'].mean())
print('mean of G2:', data['G2'].mean())
print('mean of G3:', data['G3'].mean())
```

[ ]

```
mean of G1: 10.90886075949367
mean of G2: 10.713924050632912
mean of G3: 10.415189873417722
```

```python
table = data.groupby('traveltime')['G3'].mean()
table
```

[142]

	G3
traveltime	
1	10.782101
2	9.906542
3	9.260870
4	8.750000

dtype: float64

2.1 Data Cleaning

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

	0
school	0
sex	0
age	0
address	0
famsize	0
Pstatus	0
Medu	0
Fedu	0
Mjob	0
Fjob	0
reason	0
guardian	0
traveltime	0
studytime	0
failures	0
schoolsup	0
famsup	0
paid	0

activities	0
nursery	0
higher	0
internet	0
romantic	0
famrel	0
freetime	0
goout	0
Dalc	0
Walc	0
health	0
absences	0
G1	0
G2	0
G3	0

```
# Information about the data types and the no. of entries in the columns
df['school'].info()
```

[ ]

```
<class 'pandas.core.series.Series'>
RangeIndex: 395 entries, 0 to 394
Series name: school
Non-Null Count  Dtype
--------------  -----
395 non-null    object
dtypes: object(1)
memory usage: 3.2+ KB
```

2.2 Categorizing Features

Categorical features

```
# Getting Categorical Features
categorical_features = data.select_dtypes(include=['object']).columns

# Getting Nominal Features
categorical_features_nominal = ['Mjob', 'Fjob', 'reason', 'guardian']

# Ordinal Features - Removing the nominal features from the categorical features
categorical_features_ordinal = list(categorical_features.drop(categorical_features_nominal))
```

[147]

3. Feature Engineering

3.1 Final Grades

COnverting marks into percentage and assigning grades -

- 16-20 : Excellent
- 14-15 : Good
- 12-13 : Satisfactory
- 10-11 : Poor
- 0-9 : Fail

```
df.loc[df['G3'] >= 16, 'final_grade'] = 'Excellent' # above 18
df.loc[df['G3'].between(13,16), 'final_grade'] = 'Good' # 15-17
df.loc[df['G3'].between(11,14), 'final_grade'] = 'Satisfactory' # 11-14
df.loc[df['G3'].between(9,12), 'final_grade'] = 'Poor' # 6-10
df.loc[df['G3'] <= 9, 'final_grade'] = 'Fail' # below 6
```

[ ]

Plotting function

```
def multiplot(x: list, y: str, data: pd.DataFrame, plot_type: str, palette= None, grid=False, dpi=100) -> Axes:

  # Checking the DataTypes of the arguments
  if not isinstance(x, list):
    raise TypeError('Input must be a list. Ensure it\'s a list of feature column names.')

  if not isinstance(y, str):
    raise TypeError('Input must be a string')

  if not isinstance(data, pd.DataFrame):
    raise TypeError('Input must be a DataFrame')

  if not isinstance(plot_type, str):
    raise TypeError('Input must be a string')

  if palette is None:
    palette = sns.color_palette('muted')

  if not isinstance(grid, bool):
    raise TypeError('Input must be a boolean')

  if not isinstance(dpi, int):
    raise TypeError('Input must be an integer')


  # Settings
  sns.set_style('white')
  if grid is True:
    sns.set_style('whitegrid')
```

```python
    if dpi is None:
        dpi = 100

    # creating the plot function from input
    plot_func = getattr(sns, plot_type, None)

    if plot_func is None or not callable(plot_func):
        raise ValueError(f'Invalid plot type: {plot_type}. Ensure it\'s a valid Seaborn plot type.')

    # Getting the number of features
    length = int(len(x))

    # Calculating the size of the plot
    rows = int(np.ceil(length/3)) # Such that we have 3 plot in each row

    # Dynamically adjusting the figure size
    figsize = (3 * 4.7, rows * 4.7)

    #creating the plot
    f, axs = plt.subplots(rows, 3, figsize=figsize, dpi=dpi)

    # Flatten axs for easier indexing if there is only one row or column
    axs = axs.flatten()

    # iterating through subplots
    for count, ax in enumerate(axs):
        if count < length:
            # Getting the feature to plot
            feature = x[count]

            # Plotting
            plot_func(x=data[feature], y=data[y], palette=palette, ax=ax)
            ax.set_title(f'{y} by {feature}')
```

```python
        else:
            # Deleting unused subplots
            ax.axis('off')

    # Adding title and finishing touches
    plt.suptitle('Bivariate Data Analysis', fontsize=16)
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

    return f
```

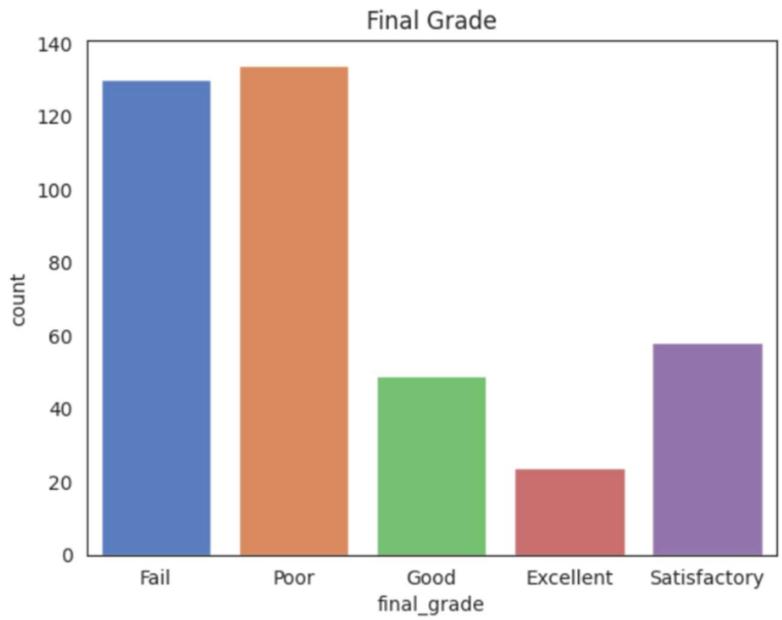
[ ]

4. EDA - Exploratory Data Analysis

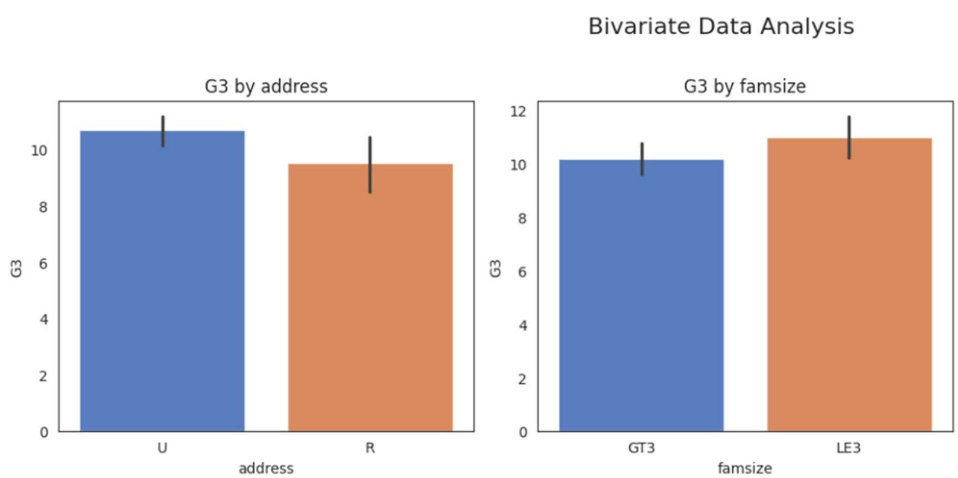
```python
# Setting a color palette for Visuals
color_palette = sns.color_palette('muted')
```

[151]

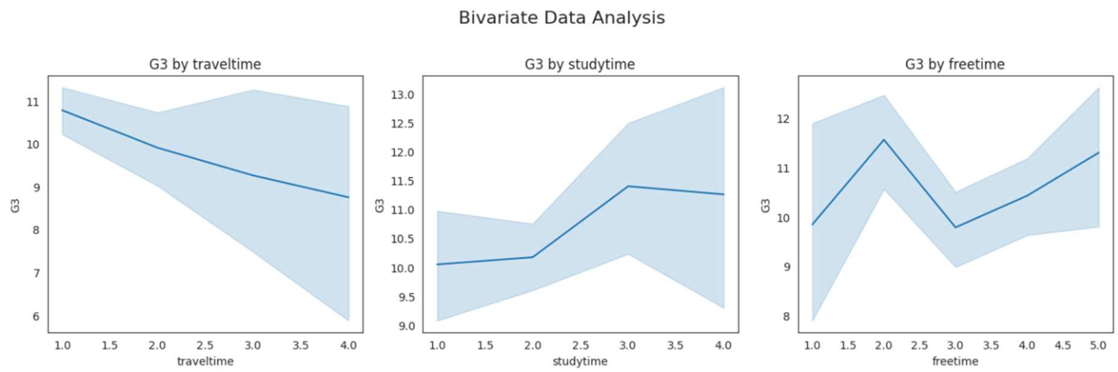
```
sns.countplot(x=df['final_grade'], palette=color_palette)
plt.title('Final Grade')
plt.savefig('final_grade.png')
```




```
X, Y = ['address', 'famsize'], 'G3'

plot = multiplot(x=X, y=Y ,data=df, plot_type='barplot', palette=color_palette)
plot.savefig('plot1')
```



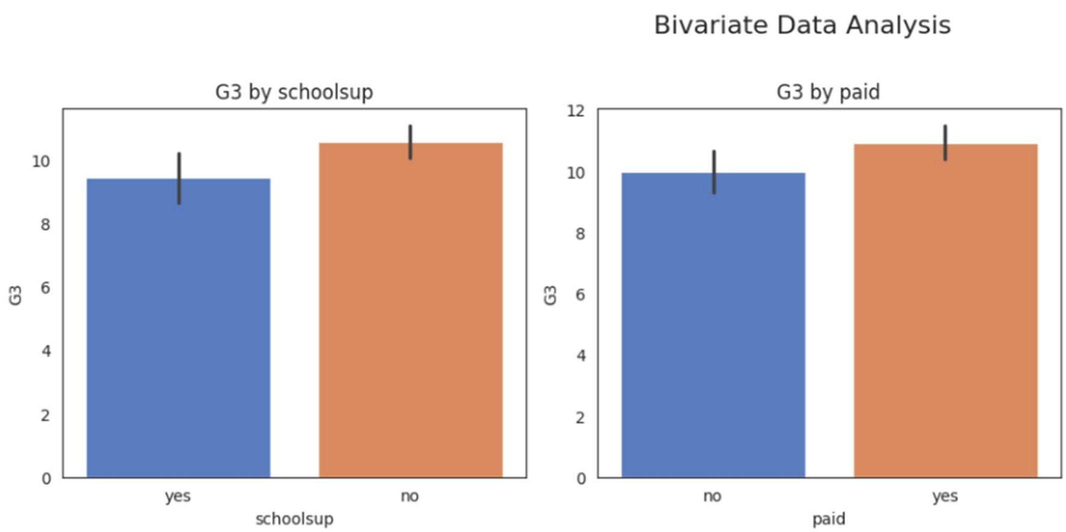

```
X, Y = ['traveltime', 'studytime', 'freetime'], 'G3'

plot = multiplot(x=X, y=Y ,data=df, plot_type='lineplot', palette=color_palette)
plot.savefig('plot2')
```




```python
X, Y = ['schoolsup', 'paid'], 'G3'

plot = multiplot(x=X, y=Y ,data=df, plot_type='barplot', palette=color_palette)
plot.savefig('plot3')
```

Python


```python
X, Y = ['schoolsup', 'paid'], 'G3'

plot = multiplot(x=X, y=Y ,data=df, plot_type='barplot', palette=color_palette)
plot.savefig('plot3')
```

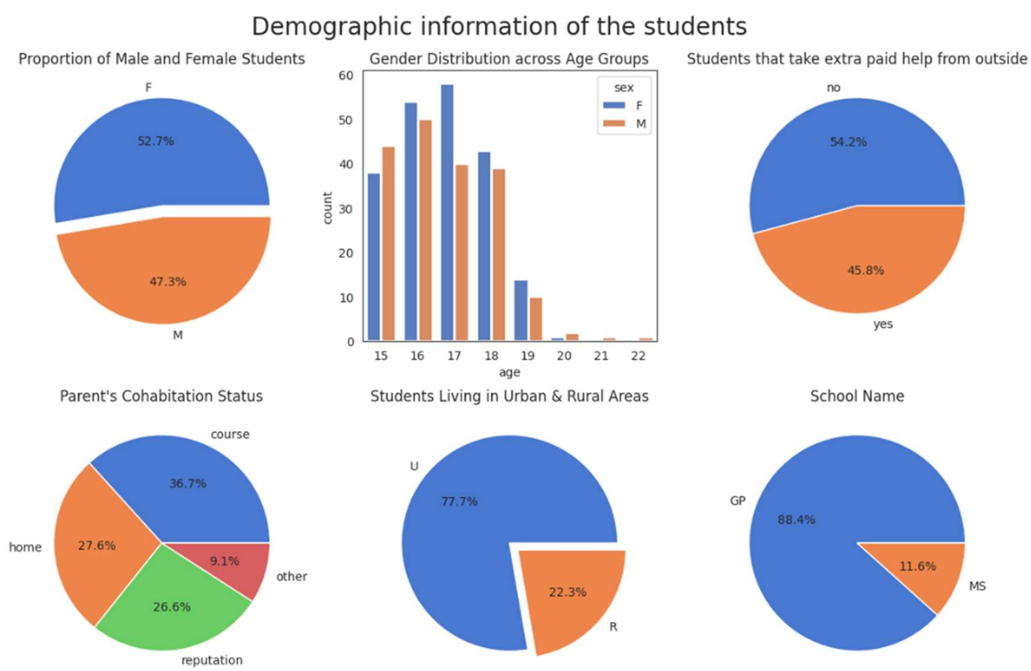



```python
ax = axs[0,2]
ax.pie(x=data['paid'].value_counts(),
       labels = data['paid'].value_counts().index,
       colors = color_palette,
       autopct='%1.1f%%')
ax.set_title('Students that take extra paid help from outside')

ax = axs[1,0]
ax.pie(x=data['reason'].value_counts(),
       labels = data['reason'].value_counts().index,
       colors = color_palette,
       autopct='%1.1f%%')
ax.set_title('Parent\'s Cohabitation Status')

ax = axs[1,1]
ax.pie(x=data['address'].value_counts(),
       labels=data['address'].value_counts().index,
       colors = color_palette, autopct= '%1.1f%%',
       explode = (0,0.1))
ax.set_title('Students Living in Urban & Rural Areas')

ax = axs[1,2]
ax.pie(x=data['school'].value_counts(),
       labels=data['school'].value_counts().index,
       colors = color_palette, autopct= '%1.1f%%')
ax.set_title('School Name')


plt.suptitle('Demographic information of the students', fontsize=20)


#plt.delaxes(ax=axs[1,2])
plt.tight_layout()
plt.show()
```

4.1 Univariate Data Analyis

```
# Creating basic plots about the Demographic information of students

f, axs = plt.subplots(2,3, figsize=(12,8))

ax = axs[0,0]
ax.pie(x=data['sex'].value_counts(), labels = data['sex'].value_counts().index,
       colors = color_palette, autopct='%1.1f%%', explode=(0,0.1))
ax.set_title('Proportion of Male and Female Students')

ax = axs[0,1]
sns.countplot(x=data['age'], hue=data['sex'],
              palette = color_palette, linewidth=2, ax=ax)
ax.set_title('Gender Distribution across Age Groups')

ax = axs[0,2]
ax.pie(x=data['paid'].value_counts(),
       labels = data['paid'].value_counts().index,
       colors = color_palette,
       autopct='%1.1f%%')
ax.set_title('Students that take extra paid help from outside')

ax = axs[1,0]
ax.pie(x=data['reason'].value_counts(),
       labels = data['reason'].value_counts().index,
       colors = color_palette,
       autopct='%1.1f%%')
ax.set_title('Parent\'s Cohabitation Status')
```

...



5. Data Pre-Processing

5.1 Pre-processing

```python
# Creating a backup
data_backup = df.copy()
```

Label encoder - For Ordinal Data (There is order of significance)
One Hot Encoder - For Nominal Data (No order of significance)

```python
# Scaling Categorical Ordinal Features

label = LabelEncoder()

# Going through and converting one column at a time
for col in categorical_features_ordinal:
    df[col] = label.fit_transform(df[col])
```

```python
# Scaling Categorical Nominal Features
one_hot = OneHotEncoder(sparse=False, drop='first')
```

```python
# Convert the columns
one_hot_encoded = one_hot.fit_transform(df[categorical_features_nominal])

# convert the above into a DataFrame
encoded_df = pd.DataFrame(one_hot_encoded, columns=one_hot.get_feature_names_out(categorical_features_nominal))

# Now add the new df in place of the old ones in the Data
data = pd.concat([data.drop(columns=categorical_features_nominal), encoded_df], axis=1)
```

```python
# Scaling Numerical Features

scaler = StandardScaler()
data[numerical_features] = scaler.fit_transform(data[numerical_features])
```

```python
data.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	traveltime	studytime	...	Mjob_teacher	Fjob_health	Fjob_other	Fjob_services
0	0	0	1.023046	1	0	0	1.143856	1.360371	0.792251	-0.042286	...	0.0	0.0	0.0	0.0
1	0	0	0.238380	1	0	1	-1.600009	-1.399970	-0.643249	-0.042286	...	0.0	0.0	1.0	0.0
2	0	0	-1.330954	1	1	1	-1.600009	-1.399970	-0.643249	-0.042286	...	0.0	0.0	1.0	0.0
3	0	0	-1.330954	1	0	1	1.143856	-0.479857	-0.643249	1.150779	...	0.0	0.0	0.0	1.0
4	0	0	-0.546287	1	0	1	0.229234	0.440257	-0.643249	-0.042286	...	0.0	0.0	1.0	0.0

5.2 Feature Selection

```
# dropping features
data=data.drop(['sex', 'G1', 'G2'], axis=1)
```

[166]

```
# Independent Variables - Feature table that will be used to predict Y
X = data.drop(columns='G3')
X = data.drop(columns='final_grade')
```

[167]

```
# Dependent Variable
Y = data['G3']
```

[168]

```
# Selecting the best Features using SelectKbest and f_regression
feature_selector = SelectKBest(score_func=f_regression, k='all')
X_new = feature_selector.fit_transform(X,Y)
```

[169]

```
# Saving the new features to variable X
X = X_new
```

[170]

5.3 Train Test Split

```python
# Splitting the dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
                                                    random_state=44)
```

[171]

Empty markdown cell, double-click or press enter to edit.

6. Modelling

6.1 Initialising the models

```python
# Initialising the models
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest Regressor': RandomForestRegressor(),
    'Support Vector Machine': SVR(),
    'Neural Network': MLPRegressor()
}
```

[172]

6. Modelling

6.1 Initialising the models

```
# Initialising the models
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest Regressor': RandomForestRegressor(),
    'Support Vector Machine': SVR(),
    'Neural Network': MLPRegressor()
}
```

[172]

6.2 Training

```
# Training the models
for name, model in models.items():
  model.fit(X_train, Y_train)
```

[173]

7 Model Evaluation

7.1 Calculating metrics

```python
# Evaluating the models
results = {}
overfit = {}

for name, model in models.items():
    Y_pred = model.predict(X_test)

    mae = mean_absolute_error(Y_test, Y_pred)
    mse = mean_squared_error(Y_test, Y_pred)
    rmse = mean_squared_error(Y_test, Y_pred, squared=False)
    r2 = r2_score(Y_test, Y_pred)

    # Storing results
    results[name] = {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'r2': r2}



    ### Calculating Overfitting ####

    # Predicting on training data and testing data (Seen and Unseen data)
    training_preds = model.predict(X_train)
    testing_preds = model.predict(X_test)
```

```python
    # Calculating the MSE
    train_MSE = mean_squared_error(Y_train, training_preds)
    test_MSE = mean_squared_error(Y_test, testing_preds)

    # overfitting values
    overfit[name] = {'Training MSE': train_MSE, 'Testing MSE':test_MSE}

#number of metrics
n = len(results[list(models.keys())[0]])

# Printing the results
results_df = pd.DataFrame.from_dict(results).T # .T to transpose it
overfit_df = pd.DataFrame.from_dict(overfit).T

# calucating difference to check overfitting
overfit_df['Difference'] = abs(overfit_df['Training MSE'] - overfit_df['Testing MSE'])

print(results_df)
print(f'\n{overfit_df}')
```

[174]

```
                                 MAE           MSE          RMSE        r2
Linear Regression       8.637605e-16  1.189023e-30  1.090423e-15  1.000000
Random Forest Regressor 4.204987e-03  6.495839e-04  2.548694e-02  0.999390
Support Vector Machine  1.751369e-01  6.000627e-02  2.449618e-01  0.943675
Neural Network          1.399901e-01  3.180245e-02  1.783324e-01  0.970148

                        Training MSE   Testing MSE    Difference
Linear Regression       1.248376e-30  1.189023e-30  5.935338e-32
Random Forest Regressor 7.864355e-05  6.495839e-04  5.709404e-04
Support Vector Machine  8.173978e-03  6.000627e-02  5.183230e-02
Neural Network          5.842573e-03  3.180245e-02  2.595987e-02
```

7.2 Comparing models

```
# creating a figure to compare the model metrics
f, axs = plt.subplots(1,n,figsize=(10,5))

for i in range(0,n):
  ax=axs[i]
  metric = results_df.columns[i]
  sns.barplot(x=results_df.index, y = results_df[metric], palette=color_palette, ax=ax)
  ax.set_title(metric)
  ax.set_xticklabels(labels=results_df.index, rotation=45, ha='right')
  ax.set_xlabel('')
  ax.set_ylabel(metric)

plt.suptitle('Model Comparison', fontsize=16)
plt.tight_layout()
plt.savefig('Model_performance')
plt.show()
```

[175]

...