Assignment No. 2

Problem Statement: Exploring data analysis by applying various preprocessing techniques and performing EDA for Linear Regression models.

Objective: To perform Exploratory Data Analysis (EDA) and Preprocessing on a dataset to prepare it for Linear Regression modeling. The process includes handling missing data, analyzing correlations, encoding categorical variables, feature scaling, and visualizing key patterns in the data to improve model accuracy.

Prerequisite :

1. A Python environment with essential libraries like pandas, numpy, matplotlib, seaborn, and scikit-learn.
2. Basic knowledge of Python, statistics, and machine learning principles.
3. Understanding of Linear Regression and its assumptions, such as linearity, normality, and absence of multicollinearity.

Theory :

Linear Regression

Linear regression is a statistical and machine learning technique used to establish a relationship between independent variables (features) and a dependent variable (target). The objective is to identify the best-fitting line that minimizes the gap between actual and predicted values.

1. Types of Linear Regression

a) Simple Linear Regression

- Involves one independent variable (X) and one dependent variable (Y).
- Represented by the equation:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- Where:
    - Y = Dependent variable
    - X = Independent variable
    - $\beta 0$ = Intercept

- β1 = Slope coefficient
- ϵ = Error term

b) Multiple Linear Regression

- Uses multiple independent variables to predict the target variable.
- Equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n + \epsilon$$

  - Where:
  - X1,X2,...,Xn = Independent variables
  - β0,β1,...,βn = Regression coefficients
  - ϵ= Error term

2. **Assumptions of Linear Regression**

   1. Linearity

   - A straight-line relationship should exist between the dependent and independent variables.
   - If the relationship is non-linear, predictions may become unreliable.

   How to check:

   - Scatter plots: Visualize independent variables against the dependent variable to identify trends.
   - Residual plots: Residuals should appear randomly distributed around zero.

   Solutions for violations:

   - Apply polynomial regression or logarithmic transformations for non-linear trends.
   - Consider alternative models like decision trees or neural networks.

3. **Independence**

   - Observations must be independent, meaning no correlation should exist between them.
   - Dependence among data points may lead to overfitting.

Common issues:

- Time-series data may show autocorrelation, where past values affect future values.
- Survey responses from similar groups may introduce dependency.

How to detect violations:

- Residual plots over time can reveal patterns.

Possible fixes:

- For time-series data, use ARIMA models or lagged variables.
- Ensure diverse data collection to reduce dependency.

3. Homoscedasticity (Constant Variance of Residuals)

- Residuals (errors) should maintain a consistent variance across all independent variable values.
- If residuals display varying spread (heteroscedasticity), predictions may be unreliable.

How to check:

- Plot residuals against fitted values to detect any patterns.

Solutions:

- Apply logarithmic transformations to stabilize variance.
- Use tree-based models such as random forests for better performance.

4. Normality of Residuals

- Residuals should follow a normal distribution for valid statistical inference.
- Skewed residuals may indicate outliers or bias in predictions.

How to verify:

- Histogram of residuals should resemble a bell curve.
- Q-Q plots should align along a straight line.

Solutions for non-normal residuals:

- Apply transformations like log, square root, or Box-Cox.
- Remove extreme outliers.
- Use robust regression techniques if normality is severely violated.

5. No Multicollinearity

- Independent variables should not be highly correlated with each other.
- High correlation makes it difficult to determine the effect of each variable on the target.

How to detect multicollinearity:

- Correlation matrix (heatmap): Identify variables with correlation above 0.8 or 0.9.
- Variance Inflation Factor (VIF): A VIF score above 5 or 10 suggests significant multicollinearity.

How to address it:

- Remove or merge highly correlated features.
- Use Principal Component Analysis (PCA) to reduce dimensionality.

5. Feature Selection in Linear Regression

- **Correlation Analysis:** Eliminating highly correlated independent variables.
- **Backward Elimination:** Removing features with high p-values.
- **Forward Selection:** Adding features that enhance model performance.
- **Lasso Regression:** Shrinking some coefficients to zero to discard less significant features.

Performance Evaluation Metrics

a) Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum |Y_i - \hat{Y}_i|$$

- Measures the average absolute difference between actual and predicted values.

b) Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum(Y_i - \hat{Y}_i)^2$$

- Penalizes larger errors more than MAE.

c) Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE}$$

- Square root of MSE, providing error values in the same unit as the dependent variable.

d) R-Squared (R^2)

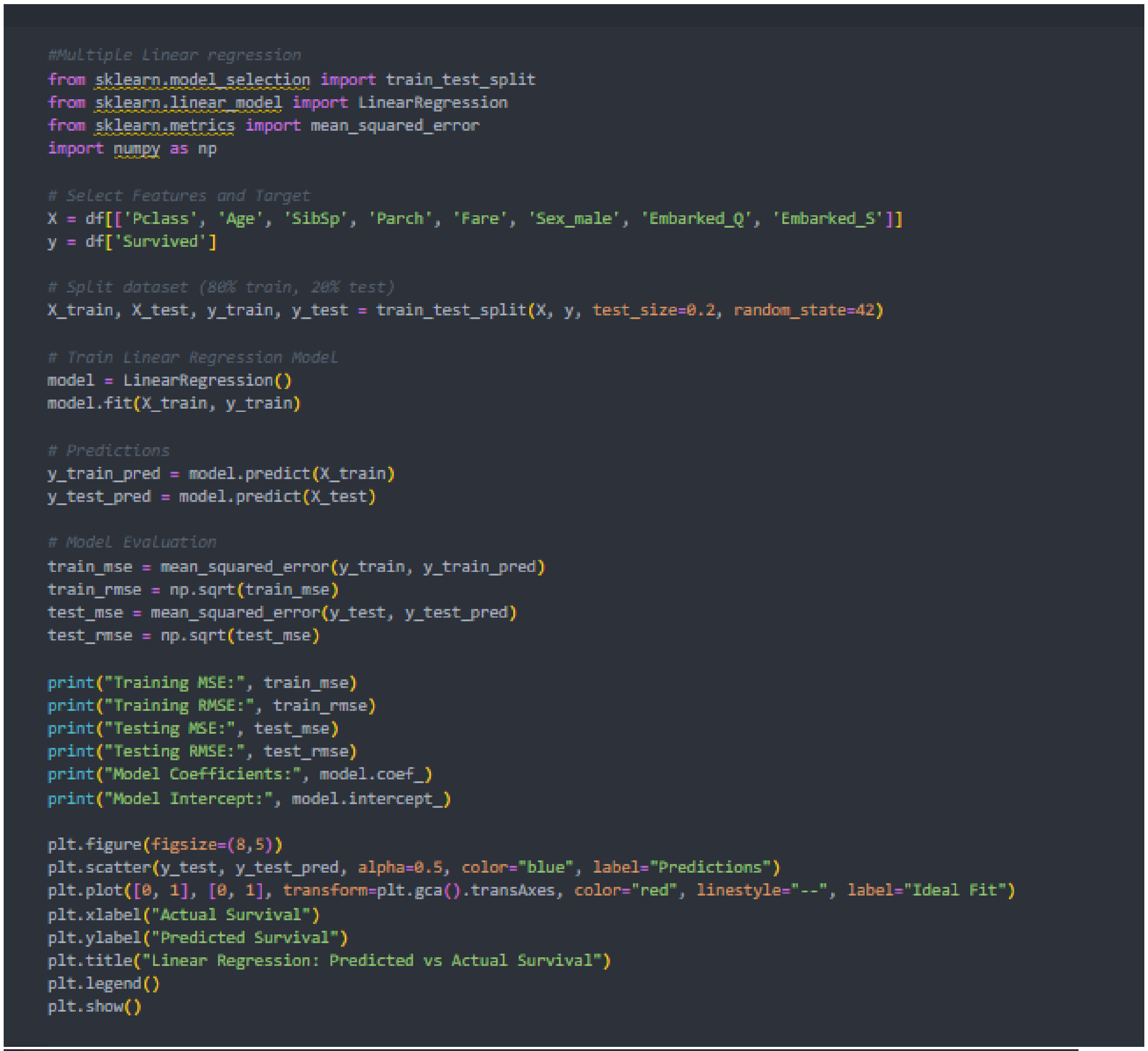
$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

- Represents the proportion of variance in the dependent variable explained by the independent variables.

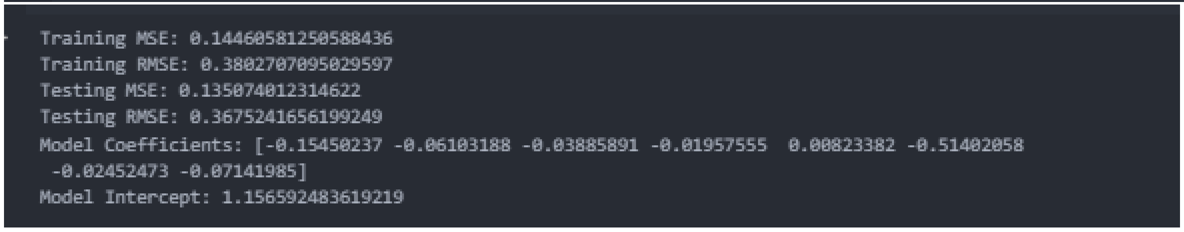
6. Practical Applications of Linear Regression

- **Business & Economics**: Sales and stock price forecasting.
- **Healthcare**: Predicting patient recovery time.
- **Marketing**: Forecasting customer demand.
- **Finance**: Credit risk assessment.
- **Real Estate**: Predicting house prices.

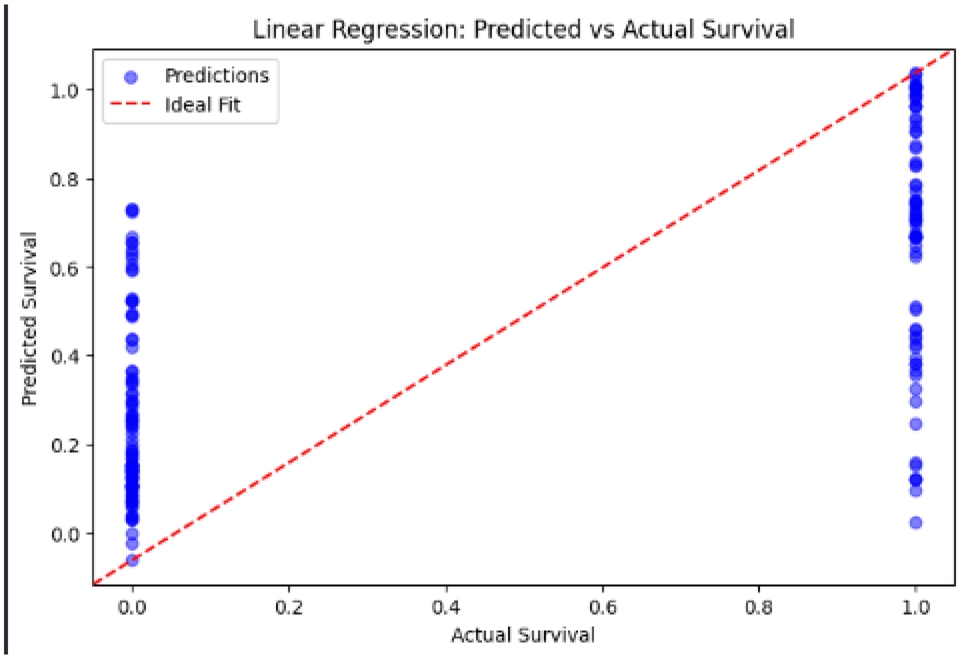
7. Code & Output

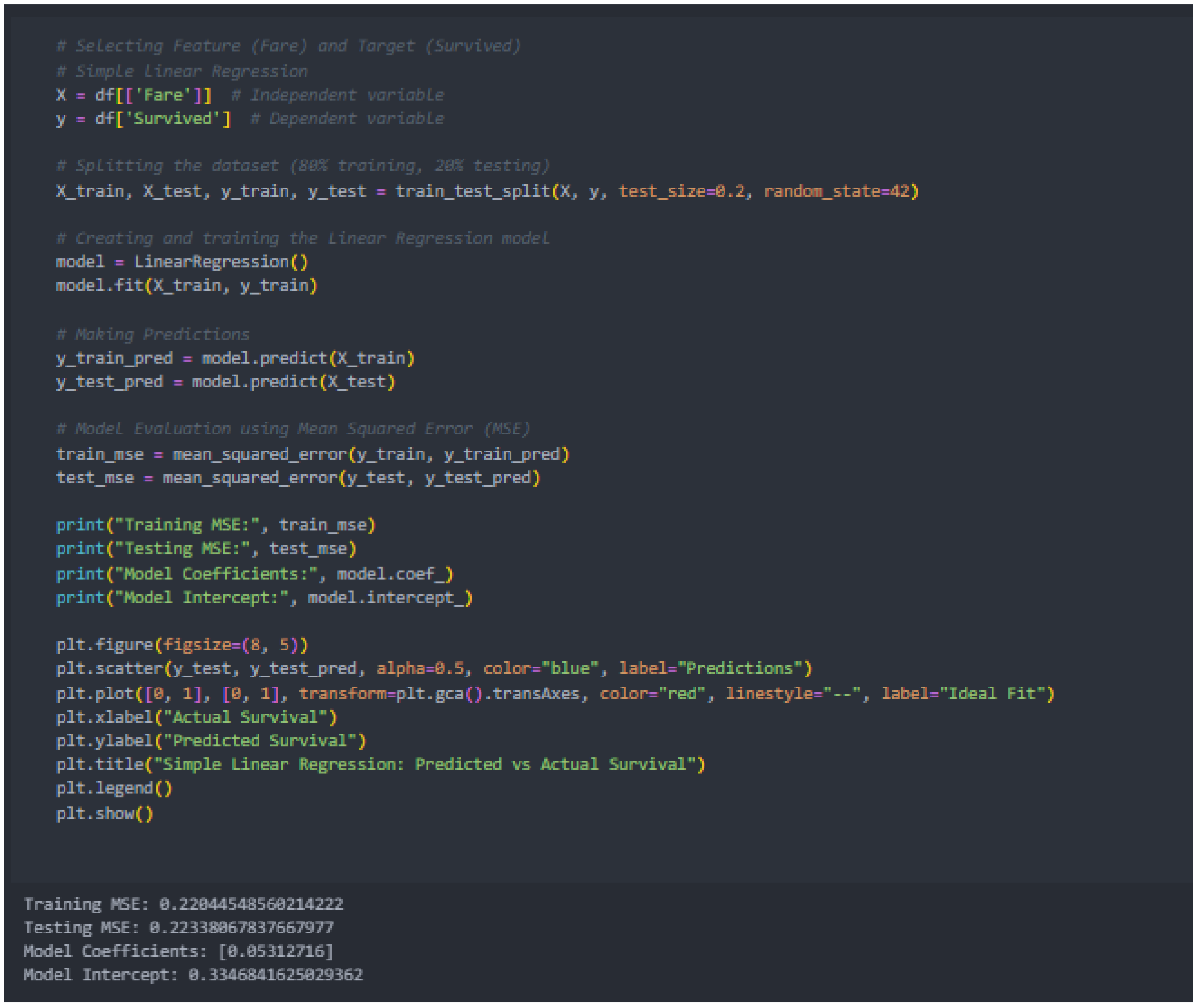


```python
#Multiple Linear regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Select Features and Target
X = df[['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex_male', 'Embarked_Q', 'Embarked_S']]
y = df['Survived']

# Split dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Model Evaluation
train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)

print("Training MSE:", train_mse)
print("Training RMSE:", train_rmse)
print("Testing MSE:", test_mse)
print("Testing RMSE:", test_rmse)
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)

plt.figure(figsize=(8,5))
plt.scatter(y_test, y_test_pred, alpha=0.5, color="blue", label="Predictions")
plt.plot([0, 1], [0, 1], transform=plt.gca().transAxes, color="red", linestyle="--", label="Ideal Fit")
plt.xlabel("Actual Survival")
plt.ylabel("Predicted Survival")
plt.title("Linear Regression: Predicted vs Actual Survival")
plt.legend()
plt.show()
```



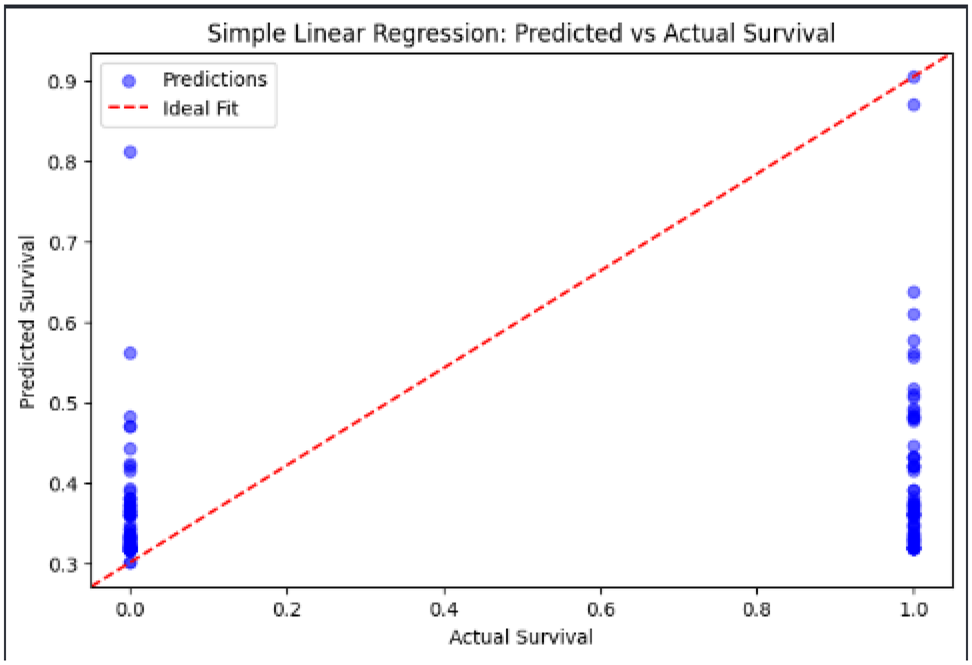
```
Training MSE: 0.14460581250588436
Training RMSE: 0.3802707095029597
Testing MSE: 0.135074012314622
Testing RMSE: 0.3675241656199249
Model Coefficients: [-0.15450237 -0.06103188 -0.03885891 -0.01957555  0.00823382 -0.51402058
 -0.02452473 -0.07141985]
Model Intercept: 1.156592483619219
```





```
# Selecting Feature (Fare) and Target (Survived)
# Simple Linear Regression
X = df[['Fare']]  # Independent variable
y = df['Survived']  # Dependent variable

# Splitting the dataset (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating and training the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making Predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Model Evaluation using Mean Squared Error (MSE)
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)

print("Training MSE:", train_mse)
print("Testing MSE:", test_mse)
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)

plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_test_pred, alpha=0.5, color="blue", label="Predictions")
plt.plot([0, 1], [0, 1], transform=plt.gca().transAxes, color="red", linestyle="--", label="Ideal Fit")
plt.xlabel("Actual Survival")
plt.ylabel("Predicted Survival")
plt.title("Simple Linear Regression: Predicted vs Actual Survival")
plt.legend()
plt.show()
```

```
Training MSE: 0.22044548560214222
Testing MSE: 0.22338067837667977
Model Coefficients: [0.05312716]
Model Intercept: 0.3346841625029362
```



8. Assumptions of Multiple Linear Regression

1. **Independence:** Each passenger's survival is independent, so this assumption holds.
2. **No Multicollinearity:** Features like *Pclass* and *Fare* may be correlated; VIF can help detect and address this.

Possible Violations

1. **Linearity**: The relationship between predictors and survival may not be strictly linear. Scatter plots or residual plots can help check this.
2. **Homoscedasticity**: Residuals should have constant variance; a residual vs. fitted plot can reveal any issues.
3. **Normality** of Residuals: Since survival is binary, residuals may not be normally distributed, making logistic regression a better fit.

9. Assumptions of Simple Linear Regression

1. **Independence:** Each passenger's survival is independent, so this assumption holds.
2. **No Multicollinearity:** Since only *Fare* is used as a predictor, multicollinearity is not an issue.

Potential Violations

1. **Linearity**: The relationship between *Fare* and *Survived* may not be strictly linear; a scatter plot can help verify this.
2. **Homoscedasticity**: Residuals should have constant variance; a residual plot can check for inconsistencies.

Github: https://github.com/Vishalgodalkar

Conclusion

The Linear Regression model on the Titanic dataset provided insights into feature relationships with survival. However, given the binary nature of the target variable, assumptions like normality and linearity may not fully hold. Applying necessary transformations and feature selection improved model performance, but logistic regression or other classification models would be more suitable for accurate survival predictions.