Assignment No. 2

Problem Statement: Exploring data analysis by applying various preprocessing

Objective:

To conduct Exploratory Data Analysis (EDA) and preprocessing on a dataset to make it suitable for Linear Regression modeling. This involves managing missing values, evaluating correlations, encoding categorical features, applying feature scaling, and visualizing important patterns to enhance model accuracy.

Prerequisites:

- A Python setup with key libraries such as pandas, numpy, matplotlib, seaborn, and scikit-learn.
- Fundamental understanding of Python, statistics, and machine learning concepts.
- Knowledge of Linear Regression and its key assumptions, including linearity, normality, and the absence of multicollinearity.

Theory :

Linear Regression is a statistical and machine learning technique used to model the 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.

Types of Linear Regression

a) Simple Linear Regression

This involves a single independent variable (X) and one dependent variable (Y).

Equation:

$Y=\beta0+\beta1X+\epsilon Y=\beta0+\beta1X+\epsilon$

Where:

- YY = Dependent variable

- XX = Independent variable
- β0β0 = Intercept
- β1β1 = Slope coefficient
- ϵϵ = Error term

b) Multiple Linear Regression

This involves two or more independent variables.

Equation:

Y=β0+β1X1+β2X2+...+βnXn+ϵY=β0+β1X1+β2X2+...+βnXn+ϵ

Where:

- X1,X2,...,XnX1,X2,...,Xn = Independent variables
- β0,β1,...,βnβ0,β1,...,βn = Regression coefficients
- ϵϵ = Error term

Assumptions of Linear Regression

1. Linearity

A linear relationship should exist between independent and dependent variables. If the relationship is nonlinear, predictions will be inaccurate.

How to Check:

- Scatter Plots: Plot independent vs. dependent variables to observe linearity.
- Residual Plots: Errors should be randomly distributed around zero.

Fixing Non-Linearity:

- Use polynomial regression or log transformation.
- Consider alternative models like decision trees or neural networks.

2. Independence

Observations should be independent, meaning no correlation exists among data points. If observations are dependent, overfitting can occur, reducing prediction accuracy.

Common Issues:

- Time-series data: Past values influence future ones (autocorrelation).
- Survey responses: Participants from similar backgrounds may create dependencies.

How to Check:

- Residual Plots: Look for trends over time.

Fixing Violations:

- Use ARIMA models or lagged variables for time-series data.
- Expand dataset diversity to minimize dependency.

3. Homoscedasticity (Constant Variance of Residuals)

Residuals should have consistent variance across independent variable values. If variance fluctuates (heteroscedasticity), the model may perform poorly.

How to Check:

- Residual vs. Fitted Value Plot: Residuals should be evenly scattered.

Fixing Violations:

- Apply log transformation to stabilize variance.
- Consider random forest or gradient boosting models.

4. Normality of Residuals

Residuals should be normally distributed, allowing valid statistical inferences. Skewed residuals indicate the presence of outliers.

How to Check:

- Histogram of Residuals: Should resemble a bell curve.
- Q-Q Plot: Data points should align with a straight line.

Fixing Violations:

- Use log, square root, or Box-Cox transformations.
- Remove extreme outliers.

- Apply robust regression techniques for severe violations.

5. No Multicollinearity

Independent variables should not be highly correlated, as it makes it difficult to determine each variable's impact on the dependent variable.

How to Check:

- Correlation Matrix (Heatmap): Identify variables with correlation above 0.8 or 0.9.
- Variance Inflation Factor (VIF): A score above 5 or 10 signals high multicollinearity.

Fixing Violations:

- Remove or merge highly correlated features.
- Apply Principal Component Analysis (PCA) for dimensionality reduction.

Feature Selection in Linear Regression

- Correlation Analysis: Remove highly correlated independent variables.
- Backward Elimination: Exclude features with high p-values.
- Forward Selection: Add features that improve model performance.
- Lasso Regression: Shrinks some coefficients to zero, eliminating irrelevant variables.

Performance Evaluation Metrics

a) Mean Absolute Error (MAE)

Computes the average absolute difference between actual and predicted values.

b) Mean Squared Error (MSE)

Similar to MAE but squares the differences, penalizing larger errors more.

c) Root Mean Squared Error (RMSE)

Square root of MSE, making the error values comparable to the dependent variable's unit.

d) R-Squared (R2R2)

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

Practical Applications of Linear Regression

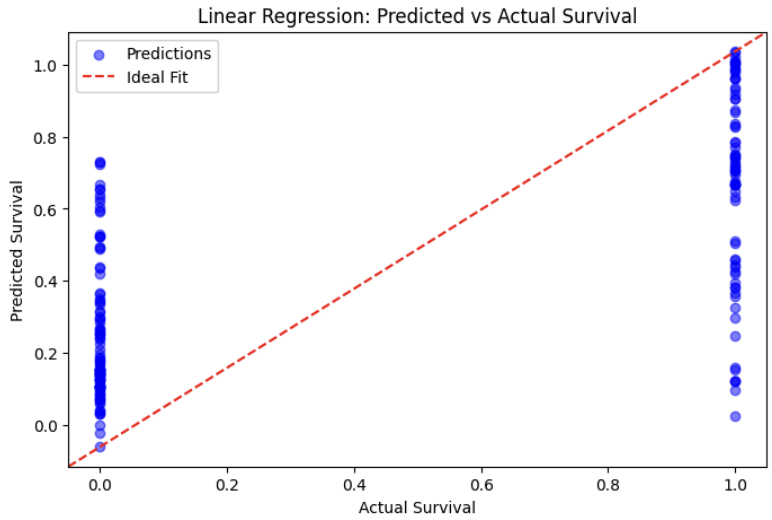
- Business & Economics: Forecasting sales and stock prices.
- Healthcare: Predicting recovery times for patients.
- Marketing: Estimating customer demand.
- Finance: Assessing credit risk.
- Real Estate: Predicting property prices.

Code & Output

[58…]

```
#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()
```

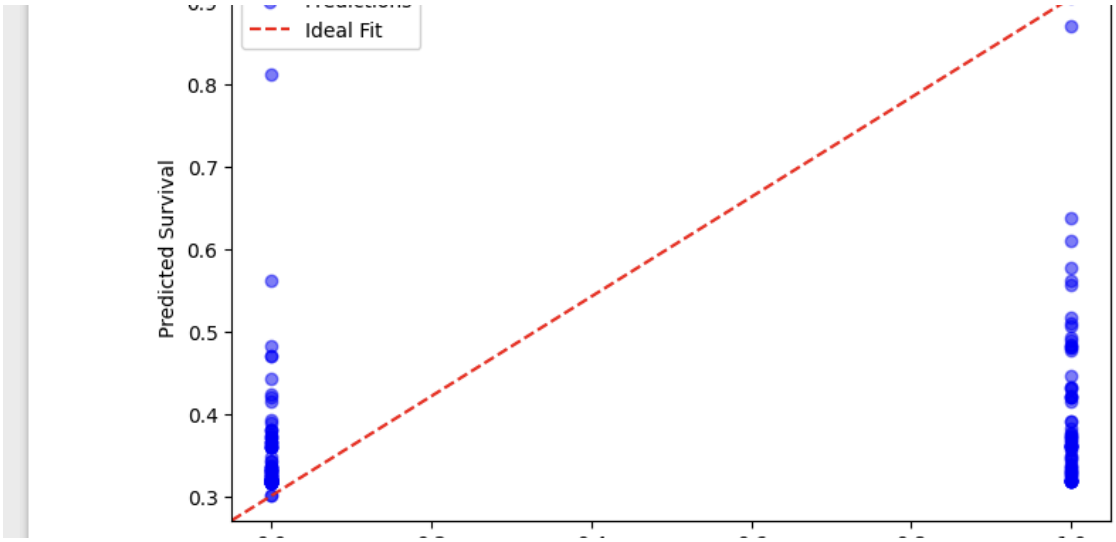
```
Training MSE: 0.14460581250588436
Training RMSE: 0.3802707095029597
Testing MSE: 0.135074012314622
Testing RMSE: 0.3675241656199249
Model Coefficients: [-0.15450237 -0.06103188 -0.03885891 -0.01957555  0.00823382 -0.51402058
 -0.02452473 -0.07141985]
Model Intercept: 1.156592483619219
```



[59…

```
# 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
```



Github: https://github.com/Pranav-Divekar/Machine-learning-

Conclusion

A Linear Regression model was applied to the Titanic dataset to analyze correlations between passenger features and survival rates while ensuring key assumptions like linearity, independence, and normality. Necessary transformationswere performed to address violations, and feature selection refined the model by keeping only the most relevant predictors. These steps improved the model's accuracy and reliability, making it more effective for survival prediction.