1. **Data Exploration & Visualization (Do it from PDF-Ass1)**

1. Read any csv file.

2. Identify the variables in the file and determine whether any variable has missing values.

3. Impute some of the variables that have missing values using their corresponding mean/median values. Verify whether your task has been correctly done.

4. Compute the Kurtosis and Skewness of the variables and interpret the results obtained.

5. Determine the "summary" information for the numerical variables.

6. Identify the "distributions" of the numerical variables and plot the distributions.

7. Transform the numeric variables into their natural log values.

8. Check whether the numeric variables follow normality conditions.

9. Find the correlation matrix for all the variables in the dataset and plot the graph of the correlation matrix.

10. From the given dataset partition the data into 70-15-15 divisions so to construct the training, validation and test datasets.

*1)read any csv file*

*import pandas as pd*

*# Load the CSV file into a pandas DataFrame*

*file\_path = '/Users/devansh/Downloads/Most Runs All Seasons Combine.csv'*

*data = pd.read\_csv(file\_path)*

*2)identify the variables in the file and determine whether an variable has missing values*

*columns = data.columns.tolist()*

*print("Variables in the file:", columns)*

*# Check for missing values in each column*

*missing\_values = data.isnull().sum()*

*print("\nMissing values in each variable:")*

*print(missing\_values)*

*3)impute sum of the variables that have missing values using their corresponding mean or*

*median values verify whether your task has been correctly done*

*imputed\_sums = data[missing\_data\_cols].select\_dtypes(include=['float64', 'int64']).sum()*

*# Output the results*

*print("\nSum of Variables with Imputed Values:")*

*print(imputed\_sums)*

*4)compute kurtosis and skewness of the variables and interpret the results obtained*

*# Step 4: Compute kurtosis and skewness for all numeric columns*

*kurtosis\_values = data.select\_dtypes(include=['float64', 'int64']).kurt()*

*skewness\_values = data.select\_dtypes(include=['float64', 'int64']).skew()*

*# Combine results into a DataFrame for better interpretation*

*stats\_summary = pd.DataFrame({*

*'Kurtosis': kurtosis\_values,*

*'Skewness': skewness\_values*

*})*

*# Display the results*

*stats\_summary*

*5)determine the summary information for the numerical variables*

*numerical\_summary = data.select\_dtypes(include=['float64', 'int64']).describe()*

*# Display the summary statistics*

*Numerical\_summary*

*6)identify the distribution of the numerical variables and plot the distribution*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*import warnings*

*warnings.filterwarnings("ignore", category=FutureWarning)*

*# Select numerical columns for plotting*

*numerical\_data = data.select\_dtypes(include=['float64', 'int64'])*

*# Create the figure and axes for the plots*

*plt.figure(figsize=(15, 20)) # Adjust figure size for better clarity*

*for i, column in enumerate(numerical\_data.columns, 1):*

*plt.subplot(6, 2, i) # Create subplots (6 rows, 2 columns for this dataset)*

*sns.histplot(numerical\_data[column], kde=True, bins=30, color='blue')*

*plt.title(f'Distribution of {column}', fontsize=12)*

*plt.xlabel(column, fontsize=10)*

*plt.ylabel('Frequency', fontsize=10)*

*plt.tight\_layout()*

*plt.show()*

*7)transform the numerical variables into their natural log values*

*import numpy as np*

*# Apply natural log transformation to numerical variables*

*# Replace zero or negative values with NaN to avoid math domain errors*

*transformed\_data = numerical\_data.copy()*

*transformed\_data = transformed\_data.applymap(lambda x: np.log(x) if x > 0 else np.nan)*

*# Display the first few rows of the transformed data*

*transformed\_data.head()*

*8)check whether the numeric variables follow normalilty distribution .*

*import scipy.stats as stats*

*# Create the figure and axes for the Q-Q plots*

*plt.figure(figsize=(15, 20))*

*for i, column in enumerate(numerical\_data.columns, 1):*

*plt.subplot(6, 2, i)*

*stats.probplot(numerical\_data[column].dropna(), dist="norm", plot=plt)*

*plt.title(f'Q-Q Plot of {column}', fontsize=12)*

*plt.tight\_layout()*

*plt.show()*

*9)find the correlation matrix for the all the variables in the dataset and plot the graph of*

*the correlation matrix*

*# Step 1: Compute the correlation matrix for numerical columns*

*correlation\_matrix = numerical\_data.corr()*

*# Step 2: Plot the heatmap of the correlation matrix*

*plt.figure(figsize=(12, 8)) # Adjust figure size for clarity*

*sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5,*

*vmin=-1, vmax=1.2)*

*plt.title('Correlation Matrix of Variables', fontsize=16)*

*plt.show()*

*10)from the given dataset partition the data into 70-15-15 divisions to construct the*

*training , validation and test dataset respectively*

*from sklearn.model\_selection import train\_test\_split*

*# Step 1: Split the data into 70% training and 30% temporary (for validation and test)*

*train\_data, temp\_data = train\_test\_split(data, test\_size=0.30, random\_state=42)*

*# Step 2: Split the temporary data into 50% validation and 50% test, which is 15% each of the*

*total data*

*validation\_data, test\_data = train\_test\_split(temp\_data, test\_size=0.50, random\_state=42)*

*# Display the shape of the resulting datasets to confirm the split*

*print(f"Data Partition");*

*print(f"Training dataset size: {train\_data.shape}")*

*print(f"Validation dataset size: {validation\_data.shape}")*

*print(f"Test dataset size: {test\_data.shape}")*

1. **Linear Regression Model for a dataset of your choice (Do it from PDF-Ass3)**

* Read the dataset of your choice.
* Describe the data.
* Build a linear regression model.
* Analyze the predicted values of the response variable.
* Compute the residuals and plot the residual values.
* Determine performance of the linear regression model

*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.linear\_model import LinearRegression*

*from sklearn.metrics import mean\_squared\_error, r2\_score*

*df = pd.read\_csv("C:\\Users\\sanke\\OneDrive\\Documents\\6th*

*Sem\\DMPML\\Student\_Performance.csv")*

*df = pd.get\_dummies(df, columns=['Extracurricular Activities'], drop\_first=True)*

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

*X = df.drop("Performance Index", axis=1)*

*y = df["Performance Index"]*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*model = LinearRegression()*

*model.fit(X\_train, y\_train)*

*y\_pred = model.predict(X\_test)*

*print("Predicted Values: ", y\_pred)*

*print("Actual Values: ", y\_test.values)*

*residuals = y\_test.values - y\_pred*

*plt.figure(figsize=(8, 6))*

*sns.residplot(x=y\_pred, y=residuals, lowess=True, line\_kws={'color': 'red'})*

*plt.title("Residuals Plot")*

*plt.xlabel("Predicted Performance Index")*

*plt.ylabel("Residuals")*

*Software Engineering Laboratory*

*plt.show()*

*mse = mean\_squared\_error(y\_test, y\_pred)*

*rmse = np.sqrt(mse)*

*r2 = r2\_score(y\_test, y\_pred)*

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

*print(f"Root Mean Squared Error: {rmse}")*

*print(f"R-Squared: {r2}")*

**04- Logistic Regression Model (Do it from PDF-Ass4)**

* Read the dataset.
* Build a logistic regression model using Python and predict in your test data set.
* Develop some metrics to determine the performance of your classification model.

*import pandas as pd*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.metrics import accuracy\_score, classification\_report*

*# Load the dataset*

*file\_path = "Data for repository(ass4).csv"*

*df = pd.read\_csv(file\_path)*

*# Convert Revenue into binary classification (High = 1 if above median, Low = 0 if below median)*

*df['Revenue\_High'] = (df['Revenue(INR)'] > df['Revenue(INR)'].median()).astype(int)*

*# Drop irrelevant columns*

*df.drop(columns=['Movie\_Name', 'Lead\_Star', 'Director', 'Music\_Director', 'Revenue(INR)'],*

*inplace=True)*

*# Encode categorical variables*

*categorical\_cols = ['Release\_Period', 'Whether\_Remake', 'Whether\_Franchise', 'Genre', 'New\_Actor',*

*'New\_Director', 'New\_Music\_Director']*

*df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)*

*# Define features and target*

*X = df.drop(columns=['Revenue\_High'])*

*y = df['Revenue\_High']*

*# Split data into train and test sets*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)*

*# Scale numerical features*

*scaler = StandardScaler()*

*X\_train[['Number\_of\_Screens', 'Budget(INR)']] = scaler.fit\_transform(X\_train[['Number\_of\_Screens',*

*'Budget(INR)']])*

*X\_test[['Number\_of\_Screens', 'Budget(INR)']] = scaler.transform(X\_test[['Number\_of\_Screens',*

*'Budget(INR)']])*

*# Train logistic regression model*

*model = LogisticRegression()*

*model.fit(X\_train, y\_train)*

*# Predictions*

*y\_pred = model.predict(X\_test)*

*# Model performance metrics*

*accuracy = accuracy\_score(y\_test, y\_pred)*

*classification\_rep = classification\_report(y\_test, y\_pred)*

*# Print results*

*print(f"Accuracy: {accuracy:.2f}")*

*print("Classification Report:\n", classification\_rep)*

**5. Build a decision Tree model on dataset of your choice and check for the performance of the model.**

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.tree import DecisionTreeClassifier, plot\_tree*

*from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix*

*# Load dataset (Iris)*

*from sklearn.datasets import load\_iris*

*data = load\_iris()*

*X, y = data.data, data.target*

*# Split dataset into training & testing sets*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Build Decision Tree model*

*model = DecisionTreeClassifier(random\_state=42)*

*model.fit(X\_train, y\_train)*

*# Make predictions*

*y\_pred = model.predict(X\_test)*

*# Evaluate model performance*

*accuracy = accuracy\_score(y\_test, y\_pred)*

*print(f"Model Accuracy: {accuracy:.2f}")*

*print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))*

*print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))*

*# Visualize the Decision Tree*

*plt.figure(figsize=(10, 8))*

*plot\_tree(model, feature\_names=data.feature\_names, class\_names=data.target\_names, filled=True)*

*plt.title("Decision Tree Visualization")*

*plt.show()*

**6. Build a k-means clustering model on dataset of your choice and check for the performance of the model.**

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*from sklearn.datasets import load\_iris*

*from sklearn.cluster import KMeans*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.decomposition import PCA*

*# Load dataset (Iris)*

*data = load\_iris()*

*X = data.data*

*# Normalize the data*

*scaler = StandardScaler()*

*X\_scaled = scaler.fit\_transform(X)*

*# Apply K-Means for multiple k values*

*k\_values = range(1, 10)*

*inertia = []*

*for k in k\_values:*

*kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)*

*kmeans.fit(X\_scaled)*

*inertia.append(kmeans.inertia\_)*

*# Elbow Method Plot*

*plt.figure(figsize=(8, 5))*

*plt.plot(k\_values, inertia, marker='o', linestyle='--')*

*plt.xlabel("Number of Clusters (k)")*

*plt.ylabel("Inertia")*

*plt.title("Elbow Method for Optimal k")*

*plt.show()*

*# Choosing optimal k (e.g., k=3 for Iris dataset)*

*k\_optimal = 3*

*kmeans = KMeans(n\_clusters=k\_optimal, random\_state=42, n\_init=10)*

*clusters = kmeans.fit\_predict(X\_scaled)*

*(Principal Component Analysis (PCA) is a dimensionality reduction technique used in machine learning to transform high-dimensional data into a lower-dimensional space while retaining most of the original information)*

*# Visualizing clusters using PCA*

*pca = PCA(n\_components=2)*

*X\_pca = pca.fit\_transform(X\_scaled)*

*plt.figure(figsize=(8, 5))*

*sns.scatterplot(x=X\_pca[:, 0], y=X\_pca[:, 1], hue=clusters, palette="viridis", s=100)*

*plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], color='red', marker='X', s=200, label='Centroids')*

*plt.xlabel("Principal Component 1")*

*plt.ylabel("Principal Component 2")*

*plt.title(f"K-Means Clustering Visualization (k={k\_optimal})")*

*plt.legend()*

*plt.show()*

**7. Association Rule Mining**

Choose data-set of your choice having more than 500 purchase transactions of minimum 10 different items

Using Apriori algorithm mine association rules with

(a) Minimum support = 1% and confidence =30%

(b) Minimum support = 2% and confidence =40%

(c) minimum support = 3% and confidence =50%

Display first 5 rules in each of the above cases.

Sort all rules based on "lift" and display first 5 rules.

Interpret the confidence value of any 2 rules obtained

Plot the rules

Plot the rules using group method

Display first 5 rules in each of the above cases but with minimum length 5.