Krushikesh Shelar D15-C Roll No:51

EXPERIMENT -1

Aim:

- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren’t useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- standardization and normalization of columns

Data preprocessing

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Why is Data Preprocessing important?

Preprocessing of data is mainly to check the data quality. The quality can be checked by the following:

- Accuracy: To check whether the data entered is correct or not.
- Completeness: To check whether the data is available or not recorded.
- Consistency: To check whether the same data is kept in all the places that do or do not match.
- Timeliness: The data should be updated correctly.
- Believability: The data should be trustable.

Interpretability: The understandability of the data.

Dataset: Aircraft Fleet Dataset:

Krushikesh Shelar D15-C Roll No:51

1) Loading Data in Pandas

Step 1: Load Data in Pandas

```python
import pandas as pd
df = pd.read_csv("Fleet Data.csv")
df.head()
```

	Parent Airline	Airline	Aircraft Type	Current	Future	Historic	Total	Orders	Unit Cost	Total Cost (Current)	Average Age
0	Aegean Airlines	Aegean Airlines	Airbus A319	1.0	NaN	3.0	4.0	NaN	$90	$90	11.6
1	Aegean Airlines	Olympic Air	Airbus A319	NaN	NaN	8.0	8.0	NaN	$90	$0	NaN
2	Aegean Airlines	Aegean Airlines	Airbus A320	38.0	NaN	3.0	41.0	NaN	$98	$3,724	7.5
3	Aegean Airlines	Olympic Air	Airbus A320	NaN	NaN	9.0	9.0	NaN	$98	$0	NaN
4	Aegean Airlines	Aegean Airlines	Airbus A321	8.0	NaN	NaN	8.0	NaN	$115	$919	10.3

2)Description of the dataset.

```
Dataset Shape: (1583, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1583 entries, 0 to 1582
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Parent Airline        1583 non-null   object
 1   Airline               1583 non-null   object
 2   Aircraft Type         1583 non-null   object
 3   Current               859 non-null    float64
 4   Future                188 non-null    float64
 5   Historic              1113 non-null   float64
 6   Total                 1484 non-null   float64
 7   Orders                348 non-null    float64
 8   Unit Cost             1548 non-null   object
 9   Total Cost (Current)  1556 non-null   object
 10  Average Age           820 non-null    float64
dtypes: float64(6), object(5)
memory usage: 136.2+ KB
Dataset Description:            Current      Future     Historic        Total      Orders  \
count  859.000000  188.000000  1113.000000  1484.000000  348.000000
mean    24.033760    3.382979    14.513028    24.955526   26.419540
std     41.091234    4.656331    23.763373    46.651526   43.024179
min      1.000000    1.000000     1.000000     1.000000    1.000000
25%      5.000000    1.000000     3.000000     4.000000    5.000000
50%     12.000000    2.000000     7.000000    11.000000   13.500000
75%     26.500000    4.000000    16.000000    27.000000   28.250000
max    718.000000   38.000000   325.000000   952.000000  400.000000

       Average Age
count   820.000000
mean     10.115000
std       6.859362
min       0.100000
25%       5.000000
50%       8.900000
75%      14.500000
max      39.000000
```

`df.info()`: Provides an overview of the dataset, including:

- Number of rows and columns.

- Data types of each column (e.g., int, float, object).
- Number of non-null (non-missing) values in each column.

`df.describe()`: Generates summary statistics for numeric columns, such as:

- `count`: Number of non-missing values.
- `mean`: Average value.
- `std`: Standard deviation.
- `min`, `25%`, `50%` (median), `75%`, and `max`: Percentile values.

3) Drop columns that aren't useful: Columns like Orders and Average Age which had very little data may not contribute to analysis and we can't replace them with mean as it would make a lot of data irrelavant. Removing irrelevant columns reduces complexity.

```
# Drop columns that are not useful for analysis
df = df.drop(columns=['Orders', 'Future','Average Age'])

# Display the first few rows after dropping columns
df.head()
```

	Parent Airline	Airline	Aircraft Type	Current	Historic	Total	Unit Cost	Total Cost (Current)
0	Aegean Airlines	Aegean Airlines	Airbus A319	1.0	3.0	4.0	$90	$90
1	Aegean Airlines	Olympic Air	Airbus A319	NaN	8.0	8.0	$90	$0
2	Aegean Airlines	Aegean Airlines	Airbus A320	38.0	3.0	41.0	$98	$3,724
3	Aegean Airlines	Olympic Air	Airbus A320	NaN	9.0	9.0	$98	$0
4	Aegean Airlines	Aegean Airlines	Airbus A321	8.0	NaN	8.0	$115	$919

4)Drop rows with maximum missing values.

```
df = df.dropna(thresh=len(df.columns) - 2)
```

- Drops rows where more than 2 columns have missing (NaN) values.

```
print(f"Rows before dropping: {df.shape[0]}")

# Drop rows with maximum missing values
df = df.dropna(thresh=len(df.columns) - 2)

# Number of rows after dropping
print(f"Rows after dropping: {df.shape[0]}")

# Display the first few rows after dropping rows
df.head()
```

```
Rows before dropping: 1583
Rows after dropping: 1492
```

	Parent Airline	Airline	Aircraft Type	Current	Historic	Total	Unit Cost	Total Cost (Current)
0	Aegean Airlines	Aegean Airlines	Airbus A319	1.0	3.0	4.0	$90	$90
1	Aegean Airlines	Olympic Air	Airbus A319	NaN	8.0	8.0	$90	$0
2	Aegean Airlines	Aegean Airlines	Airbus A320	38.0	3.0	41.0	$98	$3,724
3	Aegean Airlines	Olympic Air	Airbus A320	NaN	9.0	9.0	$98	$0
4	Aegean Airlines	Aegean Airlines	Airbus A321	8.0	NaN	8.0	$115	$919

5)Take care of missing data.

`df.fillna(df.mean())`: Replaces missing values (NaN) in numeric columns with the mean of that column.

```
# Count missing values before filling
missing_before = df.isna().sum().sum()
print(f"Missing values before: {missing_before}")

# Fill missing values with the mean for numerical columns
# df = df.fillna(df.mean())
df = df.fillna(df.select_dtypes(include=['number']).mean())

# Count missing values after filling
missing_after = df.isna().sum().sum()
print(f"Missing values after: {missing_after}")
```

```
Missing values before: 1081
Missing values after: 17
```

6)create dummy variables.

`pd.get_dummies()`: Converts categorical variables into dummy variables (binary indicators: 0 or 1).

- Example: The `Gender` column becomes `Gender_Male` (1 if Male, 0 otherwise).

`columns=['...']`: Specifies which columns to convert.
`drop_first=True`: Avoids the "dummy variable trap" by dropping one dummy variable to prevent multicollinearity.

	Parent Airline	Airline	Current	Historic	Total	Unit Cost	Total Cost (Current)	Aircraft Type_ATR 42-300F/-320F	Aircraft Type_ATR 42-600	Aircraft Type_ATR 42/72	...	Aircraft Type_McDonnell Douglas MD-90	Aircraft Type_Saab 2000	Aircraft Type_Saab 340	Aircr Type_Suk Super
0	Aegean Airlines	Aegean Airlines	1.000000	3.000000	4.0	$90	$90	False	False	False	...	False	False	False	F
1	Aegean Airlines	Olympic Air	24.270907	8.000000	8.0	$90	$0	False	False	False	...	False	False	False	F
2	Aegean Airlines	Aegean Airlines	38.000000	3.000000	41.0	$98	$3,724	False	False	False	...	False	False	False	F
3	Aegean Airlines	Olympic Air	24.270907	9.000000	9.0	$98	$0	False	False	False	...	False	False	False	F
4	Aegean Airlines	Aegean Airlines	8.000000	14.629091	8.0	$115	$919	False	False	False	...	False	False	False	F

7)Find out outliers (manually)

We first identified the outliers using Python by calculating the interquartile range (IQR) and marking values outside the IQR as outliers. Then, we imported the data into Excel, where we marked the outliers with a 'Yes' label. Finally, we used Excel's filtering and deletion tools to manually remove the outliers from the dataset.

```python
import numpy as np

# Select only numeric columns
numeric_cols = df.select_dtypes(include=[np.number])

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

# Define outlier boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find outliers
outliers = (numeric_cols < lower_bound) | (numeric_cols > upper_bound)

# Print number of outliers per column
print(outliers.sum())

print(lower_bound,upper_bound)
```

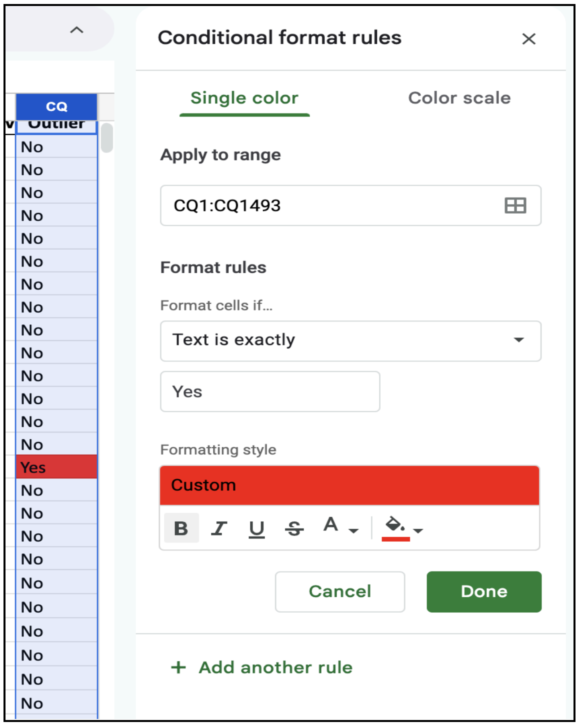
```
Current     123
Historic    131
Total       150
dtype: int64
Current    -11.406360
Historic   -11.943636
Total      -28.000000
dtype: float64 Current     45.677267
Historic    30.572727
Total       60.000000
dtype: float64
```

```python
# Mark outliers: If any of the columns in the 'outliers' DataFrame are True, mark as 'Yes'
df['Outlier'] = outliers.any(axis=1).map({True: 'Yes', False: 'No'})

# Export the DataFrame with the 'Outlier' column to an Excel file
df.to_excel('marked_outliers.xlsx', index=False)

print("Outliers marked and exported to 'marked_outliers.xlsx'.")
```

```
Outliers marked and exported to 'marked_outliers.xlsx'.
```

```python
# Remove rows where 'Outlier' is 'Yes'
df = df[df['Outlier'] == 'No']  # Overwrite the df to remove the outliers

# Now df is cleaned and free from outliers
print("Outliers removed, and df is updated without outliers.")
```

```
Outliers removed, and df is updated without outliers.
```




```
[15] # Remove rows where 'Outlier' is 'Yes'
     df = df[df['Outlier'] == 'No']  # Overwrite the df to remove the outliers

     # Now df is cleaned and free from outliers
     print("Outliers removed, and df is updated without outliers.")

Outliers removed, and df is updated without outliers.

df.shape

(1492, 9)
```

8) standardization and normalization of columns

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Standardization equation

$$X' = \frac{X - \mu}{\sigma}$$

To standardize your data, we need to import the StandardScalar from the sklearn library and apply it to our dataset.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Normalization equation

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

To normalize your data, you need to import the MinMaxScalar from the sklearn library and apply it to our dataset.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization
scaler = StandardScaler()
df_standardized = pd.DataFrame(scaler.fit_transform(df.select_dtypes(include=['float64', 'int64'])), columns=df.select_dtypes(include=['float64', 'int64']).columns)

# Normalization
min_max_scaler = MinMaxScaler()
df_normalized = pd.DataFrame(min_max_scaler.fit_transform(df.select_dtypes(include=['float64', 'int64'])), columns=df.select_dtypes(include=['float64', 'int64']).columns)

# Display the first few rows after standardization and normalization
print("Standardized Data:")
print(df_standardized.head())

print("Normalized Data:")
print(df_normalized.head())
```

```
Standardized Data:
    Current        Future     Total
0 -0.745112 -5.431067e-16 -0.454163
1  0.000000 -5.431067e-16 -0.367472
2  0.451790 -5.431067e-16  0.347727
3  0.000000 -5.431067e-16 -0.345799
4 -0.518671 -5.431067e-16 -0.367472
Normalized Data:
    Current    Future     Total
0  0.000000  0.064405  0.003155
1  0.032125  0.064405  0.007361
2  0.051604  0.064405  0.042061
3  0.032125  0.064405  0.008412
4  0.009763  0.064405  0.007361
```

Conclusion:

Thus we have understood how to perform data preprocessing which can further be taken into exploratory data analysis and further in the Model preparation sequence.