Lab 1
PANDAS LIBRARY FUNCTIONS:

In [67]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
```

In [68]:
```
def createdata():
  data = {
      'Age': np.random.randint(18, 70, size=20),
      'Salary': np.random.randint(30000, 120000, size=20),
      'Purchased': np.random.choice([0, 1], size=20),
      'Gender': np.random.choice(['Male', 'Female'], size=20),
      'City': np.random.choice(['New York', 'San Francisco', 'Los Angeles'], size=20)
  }

  df = pd.DataFrame(data)
  return df
```

In [69]:
```
df = createdata()
df.head(10)
```

Out[69]:

	Age	Salary	Purchased	Gender	City
0	66	60300	0	Female	San Francisco
1	48	94247	1	Female	New York
2	55	64292	1	Male	Los Angeles
3	31	112452	1	Male	San Francisco
4	61	58914	1	Male	New York
5	51	87062	0	Female	San Francisco
6	24	90119	0	Male	New York
7	20	113827	1	Female	New York
8	65	77804	0	Female	New York
9	38	93591	0	Male	San Francisco

In [70]:
```
# Introduce some missing values for demonstration
df.loc[5, 'Age'] = np.nan
df.loc[10, 'Salary'] = np.nan
df.head(10)
```

Out[70]:

	Age	Salary	Purchased	Gender	City
0	66.0	60300.0	0	Female	San Francisco
1	48.0	94247.0	1	Female	New York
2	55.0	64292.0	1	Male	Los Angeles

3	31.0	112452.0	1	Male	San Francisco
4	61.0	58914.0	1	Male	New York
5	NaN	87062.0	0	Female	San Francisco
6	24.0	90119.0	0	Male	New York
7	20.0	113827.0	1	Female	New York
8	65.0	77804.0	0	Female	New York
9	38.0	93591.0	0	Male	San Francisco

In [71]:

```
# Basic information about the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Age        19 non-null     float64
 1   Salary     19 non-null     float64
 2   Purchased  20 non-null     int64  
 3   Gender     20 non-null     object 
 4   City       20 non-null     object 
dtypes: float64(2), int64(1), object(2)
memory usage: 932.0+ bytes
None
```

In [72]:

```
# Summary statistics
print(df.describe())
```

```
             Age         Salary  Purchased
count  19.000000      19.000000  20.000000
mean   43.157895   80782.052632   0.450000
std    18.548025   25203.371299   0.510418
min    18.000000   34557.000000   0.000000
25%    26.000000   61824.000000   0.000000
50%    38.000000   87062.000000   0.000000
75%    61.500000  101498.500000   1.000000
max    69.000000  115007.000000   1.000000
```

In [73]:

```
#Code to Find Missing Values
# Check for missing values in each column
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
Age       1
Salary    1
dtype: int64
```

In [74]:

```
#Set the values to some value (zero, the mean, the median, etc.).
# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["Age"]])
imputer2.fit(df_copy[["Salary"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column
df_copy["Age"] = imputer1.transform(df[["Age"]])
df_copy["Salary"] = imputer2.transform(df[["Salary"]])

# Verify that there are no missing values left
print(df_copy["Age"].isnull().sum())
print(df_copy["Salary"].isnull().sum())
```

```
0
0
```

In [75]:

```
#Handling Categorical Attributes
#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column

# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
# Fit and transform the data
df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column
encoded_data = onehot_encoder.fit_transform(df[["City"]])

# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()

# Convert to DataFrame for better visualization
encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["City"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)

df_encoded.drop("Gender", axis=1, inplace=True)
df_encoded.drop("City", axis=1, inplace=True)

print(df_encoded. head())
```

```
    Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  City_New York  \
0  66.0   60300.0          0             1.0               0.0            0.0
1  48.0   94247.0          1             1.0               0.0            1.0
2  55.0   64292.0          1             0.0               1.0            0.0
3  31.0  112452.0          1             0.0               0.0            0.0
4  61.0   58914.0          1             0.0               0.0            1.0
```

```
   City_San Francisco
0                 1.0
1                 0.0
2                 0.0
3                 1.0
4                 0.0
```

In [76]:

```
#Data Transformation
# Min-Max Scaler/Normalization (range 0-1)
#Pros: Keeps all data between 0 and 1; ideal for distance-based models.
#Cons: Can distort data distribution, especially with extreme outliers.
normalizer = MinMaxScaler()
df_encoded[['Salary']] = normalizer.fit_transform(df_encoded[['Salary']])
df_encoded.head()
```

Out[76]:

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	66.0	0.319988	0	1.0	0.0	0.0	1.0
1	48.0	0.741952	1	1.0	0.0	1.0	0.0
2	55.0	0.369608	1	0.0	1.0	0.0	0.0
3	31.0	0.968241	1	0.0	0.0	0.0	1.0
4	61.0	0.302759	1	0.0	0.0	1.0	0.0

In [77]:

```
# Standardization (mean=0, variance=1)
#Pros: Works well for normally distributed data; suitable for many models.
#Cons: Sensitive to outliers.
scaler = StandardScaler()
df_encoded[['Age']] = scaler.fit_transform(df_encoded[['Age']])
df_encoded.head()
```

Out[77]:

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	1.310113	0.319988	0	1.0	0.0	0.0	1.0
1	0.289246	0.741952	1	1.0	0.0	1.0	0.0
2	0.686249	0.369608	1	0.0	1.0	0.0	0.0
3	-0.674906	0.968241	1	0.0	0.0	0.0	1.0
4	1.026538	0.302759	1	0.0	0.0	1.0	0.0

In [78]:

```
#Removing Outliers
# Outlier Detection and Treatment using IQR
#Pros: Simple and effective for mild outliers.
#Cons: May overly reduce variation if there are many extreme outliers.
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['Salary'].quantile(0.25)
Q3 = df_encoded_copy1['Salary'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['Salary'] = np.where(df_encoded_copy1['Salary'] > upper_bound, upper_bound,
                        np.where(df_encoded_copy1['Salary'] < lower_bound, lower_bound, df_encoded_copy1['Salary']))

print(df_encoded_copy1.head())
```

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0  1.310113  0.319988          0             1.0               0.0
1  0.289246  0.741952          1             1.0               0.0
2  0.686249  0.369608          1             0.0               1.0
3 -0.674906  0.968241          1             0.0               0.0
4  1.026538  0.302759          1             0.0               0.0
```

In [80]:

```python
#Removing Outliers
# Median replacement for outliers
#Pros: Keeps distribution shape intact, useful when capping isn't feasible.
#Cons: May distort data if outliers represent real phenomena.
df_encoded_copy3['Salary_zscore'] = stats.zscore(df_encoded_copy3['Salary'])
median_salary = df_encoded_copy3['Salary'].median()
df_encoded_copy3['Salary'] = np.where(df_encoded_copy3['Salary_zscore'].abs() > 3, median_salary, df_encoded_copy3['Salary'
print(df_encoded_copy3.head())
```

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0  1.310113  0.319988          0             1.0               0.0
1  0.289246  0.741952          1             1.0               0.0
2  0.686249  0.369608          1             0.0               1.0
3 -0.674906  0.968241          1             0.0               0.0
4  1.026538  0.302759          1             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            0.0                 1.0      -0.856631
1            1.0                 0.0       0.563151
2            0.0                 0.0      -0.689671
3            0.0                 1.0       1.324547
4            1.0                 0.0      -0.914598
```

```
   City_New York  City_San Francisco
0            0.0                 1.0
1            1.0                 0.0
2            0.0                 0.0
3            0.0                 1.0
4            1.0                 0.0
```

In [79]:

```
#Removing Outliers
# Z-score method
#Pros: Good for normally distributed data.
#Cons: Not suitable for non-normal data; may miss outliers in skewed distributions.

df_encoded_copy2['Salary_zscore'] = stats.zscore(df_encoded_copy2['Salary'])
df_encoded_copy2['Salary'] = np.where(df_encoded_copy2['Salary_zscore'].abs() > 3, np.nan, df_encoded_copy2['Salary'])  # R
print(df_encoded_copy2.head())
```

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0  1.310113  0.319988          0             1.0               0.0
1  0.289246  0.741952          1             1.0               0.0
2  0.686249  0.369608          1             0.0               1.0
3 -0.674906  0.968241          1             0.0               0.0
4  1.026538  0.302759          1             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            0.0                 1.0      -0.856631
1            1.0                 0.0       0.563151
2            0.0                 0.0      -0.689671
3            0.0                 1.0       1.324547
4            1.0                 0.0      -0.914598
```

3/3/25
monday

①

Date
Page

Lab - 1

```
data = pd.read_csv("Housing.csv")
df = pd.DataFrame(data)
print(df.info())
print(df.describe())
print(df["ocean proximity"].value_counts())
missing = df.isnull().sum()
missing_data = missing[missing > 0]
print(missing_data)
```

Diabetes Dataset:

1) No missing values found
2) Gender and CLASS are categorical columns
3) Label encoding is used.
Gender : {'F':0, 'M':1}
CLASS : {'N':0, 'P':1}

Adult Income Dataset:

1) No missing values found.
2) workclass, Education, marital status, occupation, relationship race, gender and native country are categorical columns
3) Label encoding is used.
private → 0
Self_emp → 1
Goverment → 2
Never_worked → 3

4) Difference between min-max scaling & standardization.

min-max Scaling:

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

Scales values for custom range like -1 to 1.
Shrinks range, preserves shape.
used when dataset is not normally distributed.

Standardization:

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

Transforms data to have mean = 0 and $\sigma = 1$ changes scale, preserves shape
used when dataset ~~is not~~ follows normal distribution.