```
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
```

```
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
```

```
df = createdata()
df.head(10)
```

	Age	Salary	Purchased	Gender	City
0	42	99908	1	Female	New York
1	27	89661	1	Female	Los Angeles
2	29	95665	0	Female	New York
3	53	59711	1	Female	Los Angeles
4	31	98323	0	Male	New York
5	52	78059	1	Male	San Francisco
6	65	77807	1	Male	Los Angeles
7	69	104990	0	Male	San Francisco
8	55	81753	0	Male	New York
9	56	62566	0	Female	San Francisco

```
df.shape
```

```
(20, 5)
```

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

Show hidden output

```
# 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
```

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

Show hidden output

```
#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
```

```
#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
```

```
#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  42.0  99908.0          1             1.0               0.0            1.0
1  27.0  89661.0          1             1.0               1.0            0.0
2  29.0  95665.0          0             1.0               0.0            1.0
3  53.0  59711.0          1             1.0               1.0            0.0
4  31.0  98323.0          0             0.0               0.0            1.0

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

```
#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()
```

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	42.0	0.921414	1	1.0	0.0	1.0	0.0
1	27.0	0.762958	1	1.0	1.0	0.0	0.0
2	29.0	0.855802	0	1.0	0.0	1.0	0.0
3	53.0	0.299824	1	1.0	1.0	0.0	0.0
4	31.0	0.896904	0	0.0	0.0	1.0	0.0

```
# 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()
```

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	-0.204903	0.921414	1	1.0	0.0	1.0	0.0
1	-1.196369	0.762958	1	1.0	1.0	0.0	0.0
2	-1.064174	0.855802	0	1.0	0.0	1.0	0.0
3	0.522172	0.299824	1	1.0	1.0	0.0	0.0
4	-0.931978	0.896904	0	0.0	0.0	1.0	0.0

```
#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 -0.204903  0.921414          1             1.0               0.0
1 -1.196369  0.762958          1             1.0               1.0
2 -1.064174  0.855802          0             1.0               0.0
3  0.522172  0.299824          1             1.0               1.0
4 -0.931978  0.896904          0             0.0               0.0

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

```
#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'])  # Replace outlie
print(df_encoded_copy2.head())
```

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0 -0.204903  0.921414          1             1.0               0.0
1 -1.196369  0.762958          1             1.0               1.0
2 -1.064174  0.855802          0             1.0               0.0
3  0.522172  0.299824          1             1.0               1.0
4 -0.931978  0.896904          0             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            1.0                 0.0       1.213641
```

```
1           0.0                 0.0      0.575541
2           1.0                 0.0      0.949421
3           0.0                 0.0     -1.289500
4           1.0                 0.0      1.114940
```

```
#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 -0.204903  0.921414          1             1.0               0.0
1 -1.196369  0.762958          1             1.0               1.0
2 -1.064174  0.855802          0             1.0               0.0
3  0.522172  0.299824          1             1.0               1.0
4 -0.931978  0.896904          0             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            1.0                 0.0       1.213641
1            0.0                 0.0       0.575541
2            1.0                 0.0       0.949421
3            0.0                 0.0      -1.289500
4            1.0                 0.0       1.114940
```