Q5) Write a program to perform Data Preprocessing Functions on the data set publically available.

AIM: Perform data preprocessing functions on the Iris dataset.

SOFTWARE USED: Python, Jupyter Notebook (or any Python IDE)

THEORY: Data preprocessing is a crucial step in machine learning pipelines. It involves transforming raw data into a format that is suitable for machine learning algorithms. In this experiment, we'll focus on three common preprocessing techniques: standardization, min-max scaling, and one-hot encoding.

- Standardization (or Z-score normalization) transforms the data to have a mean of 0 and a standard deviation of 1. It helps in making different features comparable.
- Min-max scaling scales the data to a fixed range, usually between 0 and 1, preserving the relative relationships between data points.
- One-hot encoding is used for categorical variables, converting them into a binary representation suitable for machine learning algorithms.

DATASET USED: Iris dataset, a classic dataset in machine learning containing measurements of iris flowers.

CODE:
```python
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] +
['target'])

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(iris_df.head())

# Split the dataset into features and target
X = iris.data
y = iris.target

# Split data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardization using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Min-Max scaling using MinMaxScaler
minmax_scaler = MinMaxScaler()
X_train_minmax = minmax_scaler.fit_transform(X_train)
X_test_minmax = minmax_scaler.transform(X_test)

# One-hot encoding for target variable
encoder = OneHotEncoder(categories='auto', sparse=False)
y_train_encoded = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_encoded = encoder.transform(y_test.reshape(-1, 1))

# Display preprocessed data
print("\nAfter Standardization (Scaled features):")
print("X_train_scaled:")
print(X_train_scaled[:5])
print("X_test_scaled:")
print(X_test_scaled[:5])

print("\nAfter Min-Max Scaling:")
print("X_train_minmax:")
print(X_train_minmax[:5])
print("X_test_minmax:")
print(X_test_minmax[:5])

print("\nAfter One-Hot Encoding (Target variable):")
print("y_train_encoded:")
print(y_train_encoded[:5])
print("y_test_encoded:")
print(y_test_encoded[:5])
```

This code performs standardization, min-max scaling, and one-hot encoding on the Iris dataset, displaying the preprocessed data.

OUTPUT:

```
First few rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0

 After Standardization (Scaled features):
X_train_scaled:
[[-0.4134164  -1.46200287 -0.09951105 -0.32339776]
 [ 0.55122187 -0.50256349  0.71770262  0.35303182]
 [ 0.67180165  0.21701605  0.95119225  0.75888956]
 [ 0.91296121 -0.02284379  0.30909579  0.2177459 ]
 [ 1.63643991  1.41631528  1.30142668  1.70589097]]
X_test_scaled:
[[ 0.3100623  -0.50256349  0.484213   -0.05282593]
 [-0.17225683  1.89603497 -1.26695916 -1.27039917]
 [ 2.23933883 -0.98228318  1.76840592  1.43531914]
 [ 0.18948252 -0.26270364  0.36746819  0.35303182]
 [ 1.15412078 -0.50256349  0.54258541  0.2177459 ]]
```

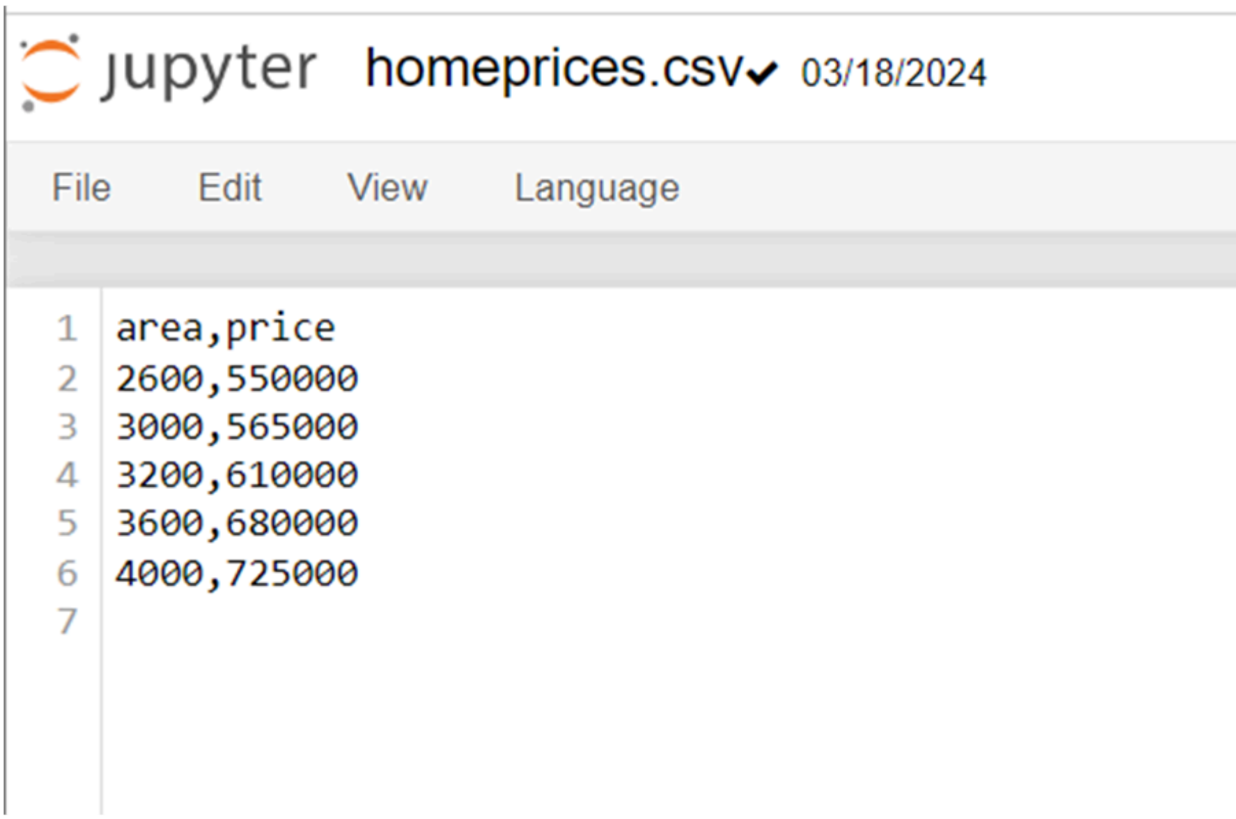
Q6) Write a program to demonstrate Single Linear Regression using Numpy, Matplot and Panda Library.

AIM: Demonstrate Single Linear Regression using Numpy, Matplotlib, and Pandas Library.

SOFTWARE USED: Python, Jupyter Notebook (or any Python IDE), Numpy, Matplotlib, Pandas

THEORY: Single Linear Regression is a simple linear regression model that predicts the relationship between one independent variable and one dependent variable. In this experiment, we aim to fit a line to the data points to best represent the relationship between the independent variable (feature) and dependent variable (target). We use the least squares method to find the line that minimizes the sum of squared differences between the observed and predicted values.

DATASET USED: homeprices.csv dataset containing home prices and corresponding areas.




```
area,price
2600,550000
3000,565000
3200,610000
3600,680000
4000,725000
```

CODE and OUTPUT:

In [1]:
```python
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
```

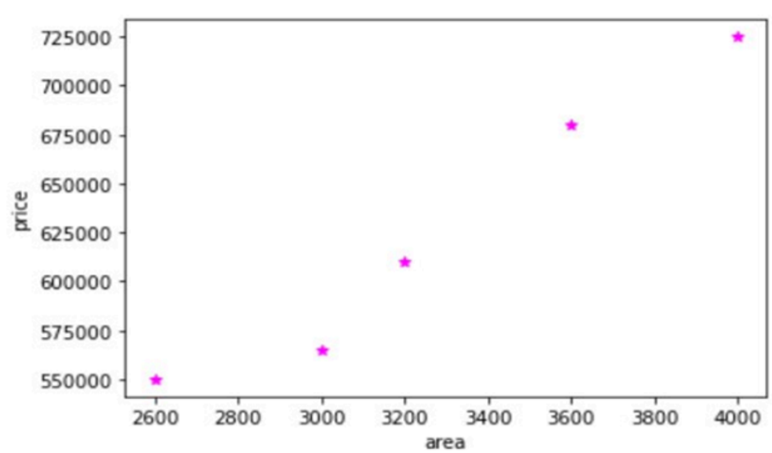
In [2]:
```python
df = pd.read_csv('homeprices.csv')
df
```

Out[2]:

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

In [4]:
```python
%matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='magenta',marker='*')
```

Out[4]: <matplotlib.collections.PathCollection at 0x1770f412ee0>



In [5]:
```
new_df = df.drop('price',axis='columns')
new_df
```
Out[5]:

	area
0	2600
1	3000
2	3200
3	3600
4	4000

In [6]:
```
price = df.price
price
```
Out[6]:
```
0    550000
1    565000
2    610000
3    680000
4    725000
Name: price, dtype: int64
```

In [7]:
```
# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,price)
```
Out[7]: `LinearRegression()`

(1) Predict price of a home with area = 3300 sqr ft

In [8]:
```
reg.predict([[3300]])
```
```
C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with f
eature names
  warnings.warn(
```
Out[8]: `array([628715.75342466])`

In [9]:
```
reg.coef_
```
Out[9]: `array([135.78767123])`

In [10]:
```
reg.intercept_
```
Out[10]: `180616.43835616432`

Y = m * X + b (m is coefficient and b is intercept)

In [11]:
```
3300*135.78767123 + 180616.43835616432
```
Out[11]: `628715.7534151643`

(1) Predict price of a home with area = 5000 sqr ft

In [12]:
```
reg.predict([[5000]])
```

```
C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
X does not have valid feature names, but LinearRegression was fitted with f
eature names
  warnings.warn(
```

Out[12]:
```
array([859554.79452055])
```

Generate CSV file with list of home price predictions

In [13]:
```
area_df = pd.read_csv("areas.csv")
area_df.head(4)
```

Out[13]:

	area
0	1000
1	1500
2	2300
3	3540

In [15]:
```
p = reg.predict(area_df)
p
```

Out[15]:
```
array([ 316404.10958904,  384297.94520548,  492928.08219178,
        661304.79452055,  740061.64383562,  799808.21917808,
        926090.75342466,  650441.78082192,  825607.87671233,
        492928.08219178, 1402705.47945205, 1348390.4109589 ,
       1144708.90410959])
```

In [17]:
```
area_df['prices']=p
area_df
```

Out[17]:

	area	prices
0	1000	3.164041e+05
1	1500	3.842979e+05
2	2300	4.929281e+05
3	3540	6.613048e+05
4	4120	7.400616e+05
5	4560	7.998082e+05
6	5490	9.260908e+05
7	3460	6.504418e+05
8	4750	8.256079e+05
9	2300	4.929281e+05
10	9000	1.402705e+06
11	8600	1.348390e+06
12	7100	1.144709e+06

In [18]:
```
area_df.to_csv("prediction.csv")

df_pred_res = pd.read_csv("prediction.csv")
df_pred_res
```

Out[18]:

	Unnamed: 0	area
0	0	1000
1	1	1500
2	2	2300
3	3	3540
4	4	4120
5	5	4560
6	6	5490
7	7	3460
8	8	4750
9	9	2300
10	10	9000
11	11	8600
12	12	7100

This code performs Single Linear Regression using the dataset from `homeprices.csv`. It calculates the coefficients of the regression line and plots the dataset along with the regression line.