Name: Gore Aniket Machhindra

Email: aniket.m.gore.1901@gmail.com

Role: Machine Learning Intern

Task 1: Loan Prediction Using Machine Learning

Content -

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

Importing Data

```
boston = load_boston()
print(boston.data)
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Convert Data into DataFrame

```
1 boston_df=pd.DataFrame(boston.data)
```

```
1 boston_df.columns=boston.feature_names
```

```
boston_df['PRICE']=boston.target
print(boston_df.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX  ...    TAX  PTRATIO       B  LSTAT  PRICE
0  0.00632  18.0   2.31   0.0  0.538  ...  296.0     15.3  396.90   4.98   24.0
1  0.02731   0.0   7.07   0.0  0.469  ...  242.0     17.8  396.90   9.14   21.6
2  0.02729   0.0   7.07   0.0  0.469  ...  242.0     17.8  392.83   4.03   34.7
3  0.03237   0.0   2.18   0.0  0.458  ...  222.0     18.7  394.63   2.94   33.4
4  0.06905   0.0   2.18   0.0  0.458  ...  222.0     18.7  396.90   5.33   36.2

[5 rows x 14 columns]
```

Shape of the Dataset

```
print(boston.target.shape)
```

```
(506,)
```

Describe the Dataset

```
boston_df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

```
boston_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
```

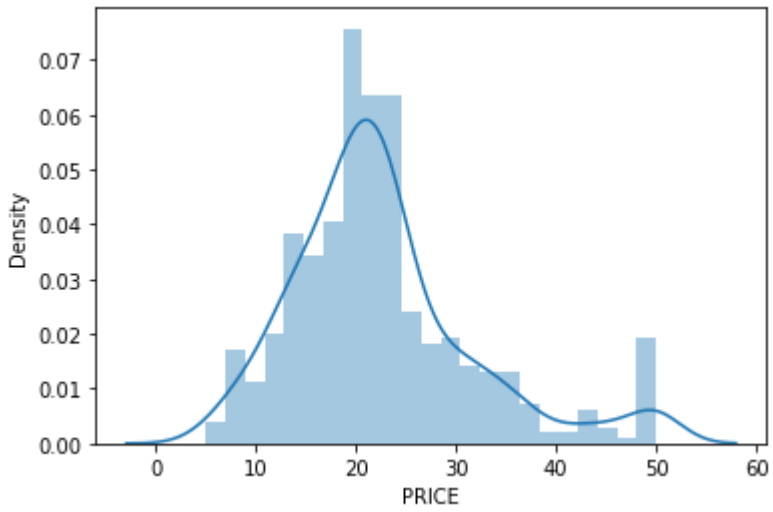
```
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

```
1 boston_df.columns
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'PRICE'],
      dtype='object')
```

```
1 sns.distplot(boston_df['PRICE'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7facc54ec8d0>
```

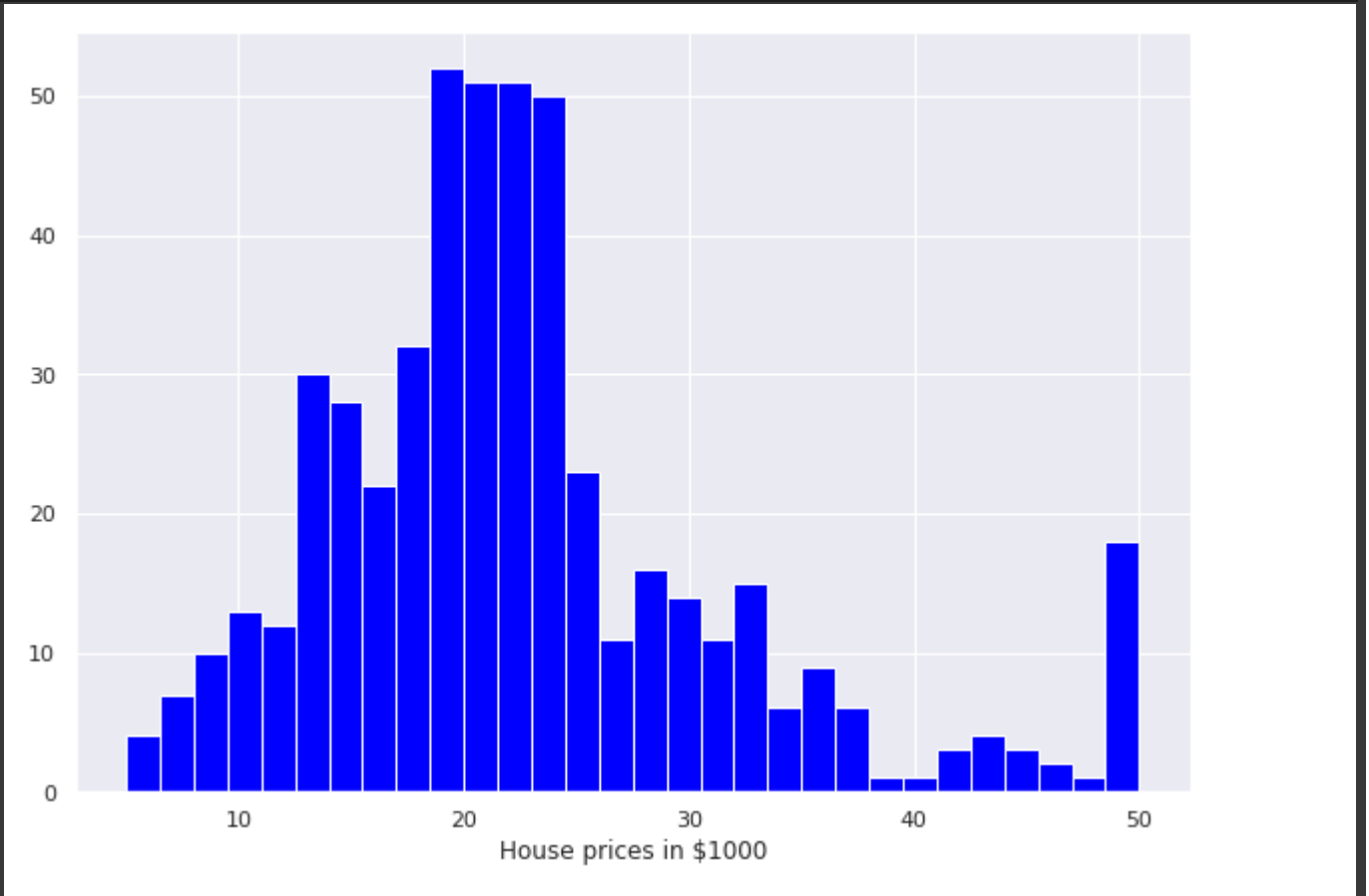


Checking Null Values

```
1 boston_df.isnull().sum()
```

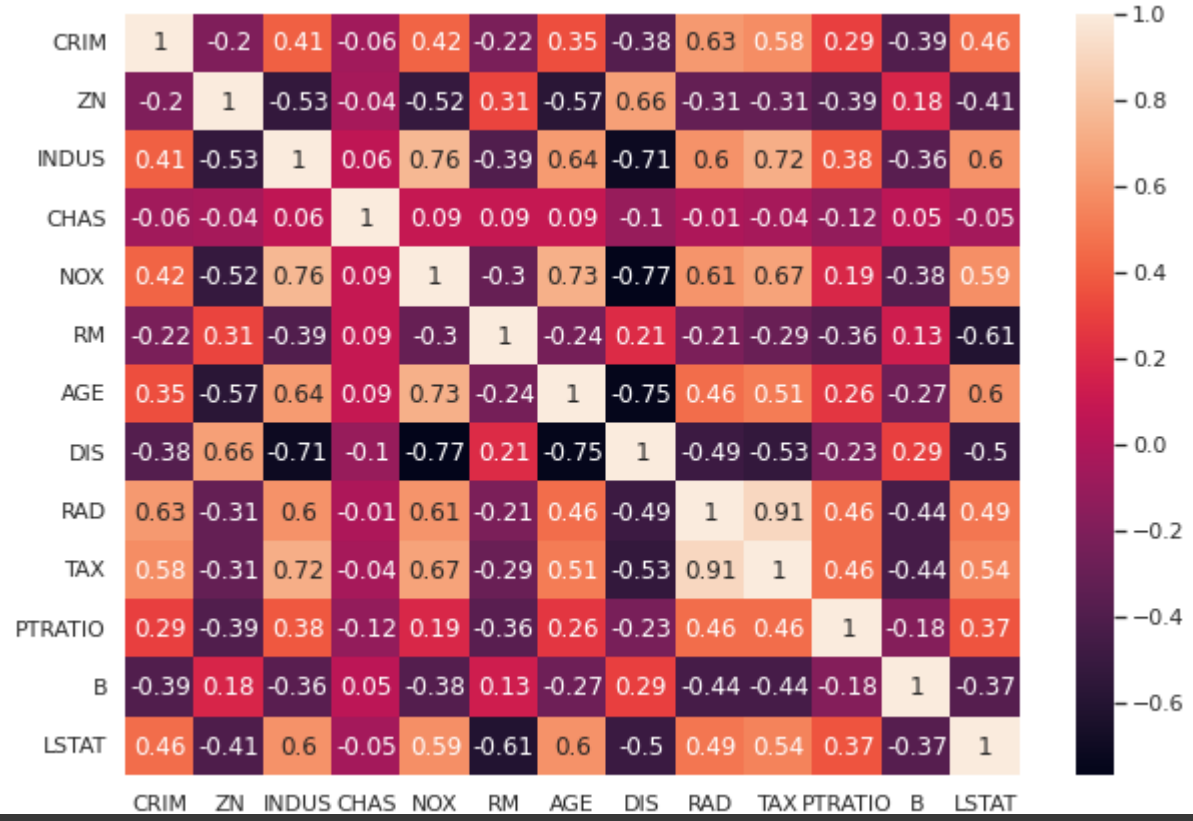
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

```
sns.set(rc={'figure.figsize':(10,7)})
plt.hist(boston_df['PRICE'],color ="blue", bins=30)
plt.xlabel("House prices in $1000")
plt.show()
```




```
bos_1=pd.DataFrame(boston.data, columns=boston.feature_names)

correlation_matrix=bos_1.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faca9f7b510>
```



Split the Data

```
1 X = boston_df.iloc[:,:-1].values
2 y = boston_df.iloc[:,-1].values
```

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

```
1 lm = LinearRegression()
2 lm.fit(X_train,y_train)
3
4 print('Coefficients:  \n',lm.coef_)
```

```
Coefficients:
 [-8.85049752e-02  5.02928536e-02  2.03483110e-02  3.75427054e+00
 -1.77483714e+01  3.24776492e+00  1.20008182e-02 -1.40916141e+00
  2.63880691e-01 -1.03440009e-02 -9.51780874e-01  6.11600491e-03
 -5.97133217e-01]
```

```
1 lm.fit(X_train,y_train)
```

```
LinearRegression()
```

```
1 predictions = lm.predict(X_test)
```

```
predictions
```

```
array([40.11113508, 27.38971873, 16.64700435, 16.98475572, 31.12920137,
       32.17489772, 38.5534506 ,  8.16734819, 33.48547457,  7.21877263,
       30.45404514, 13.44085219, 16.25354375, 17.34359227, 25.1543491 ,
       20.44171457,  7.30340549, 33.13892161, 28.41293108, 24.58522513,
       12.44673568, 20.25489284, 22.48601345, 24.42119495, 33.92740928,
       18.63104614, 32.32820984, 18.67352155, 27.36115374, 34.46174375,
       19.84089751, 18.40373436, 37.15821555, 44.94610923, 30.27513579,
       22.00760066, 16.0127978 , 18.16328402,  4.33298095, 30.93867591,
       24.15262229, 17.17277775, 34.10334259, 13.89433899, 17.46893797,
       25.30893285, 30.35309561, 16.10339452, 26.91513852, 22.98227547,
       32.14815603, 37.34454946, 22.90074019, 17.56894548, 30.18430234,
        0.10360753, 20.22573888, 16.82248142, 23.15487984, 21.16760077,
       30.5734497 ,  3.15502223, 15.92340596, 20.06361892, 10.43608925,
       24.28745773, 24.00445196, 19.86245393, 17.63614975, 19.44871423,
       23.81075322, 21.16261396, 23.47439589, 19.98453898, 27.05134381,
       21.84066905, 36.80150664,  8.16676015, 28.70036278, 17.12188494,
       15.50979745, 19.294246  , 30.15336215, 17.37264235, 10.73425764,
       21.52916918, 21.69200241, 33.12540671, 22.30189309, 21.94929448,
       12.85610293, 11.57605846, 22.66292798, 33.65426492,  6.08353957,
       34.76875886,  7.95929671, 31.90690271,  8.7752099 , 20.72989525,
       32.72047022, 21.34319049, 27.16332024, 24.1623896 , 22.68986244,
       25.17800744, 24.52779596, 30.66139018, 37.3362994 , 33.2147882 ,
       23.21825086, 36.13381973, 23.89682341, 22.07728572, 30.06489707,
       27.24105283, 29.31188864, 31.45057926, 26.74107102, 29.58735987,
       16.90021886, 20.60029568, 21.96586941, 36.77042006, 25.24859328,
       23.08568697, 15.32758657,  5.918702  , 14.80932341, 23.67342037,
       26.74592331, 34.09978093, 23.93815977, 19.9868425 , 24.73687974,
       26.10434151, 30.71721237, 26.62262586, 34.1263333 , 22.67915823,
       13.13096496, 36.60828941, 32.25783559, 15.89281425, 24.785974  ,
       19.32107821, 19.5968184 , 24.52106619, 26.34621695, 29.83423805,
       16.69898193, 16.61243821])
```

```
print('Mean Absolute Error: ', metrics.mean_absolute_error(y_test, predictions))
print('Mean Squared Error: ',metrics.mean_squared_error(y_test,predictions))
print('Root of Mean Squared Error: ',np.sqrt(metrics.mean_squared_error(y_test,predictions
```

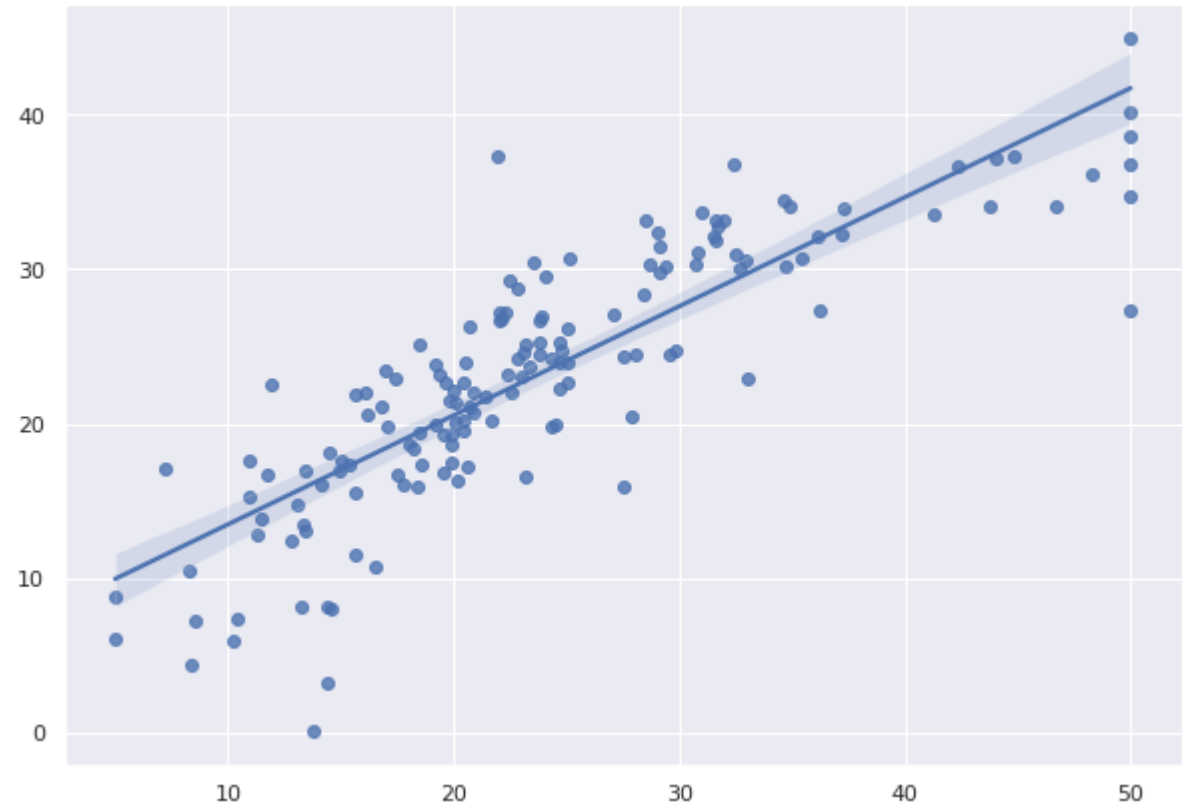
```
Mean Absolute Error:  3.8356963614189263
Mean Squared Error:  28.547585271468208
Root of Mean Squared Error:  5.342994036256096
```

Plotting the Model

```
from sklearn.metrics import mean_squared_error, r2_score

lin_model = LinearRegression()
model = lin_model
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

```

```
 8 rmse_score = np.sqrt(mean_squared_error(y_test, y_pred))
 9 rsquared_score = r2_score(y_test, y_pred)
10 print('RMSE score:', rmse_score)
11 print('R2 score:', rsquared_score)
12 print('\nScatter plot of y_test against y_pred:')
13 sns.regplot(y_test, y_pred);
```

```
RMSE score: 5.342994036256096
R2 score: 0.7123963332666865

Scatter plot of y_test against y_pred:
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
1 sns.distplot((y_test-predictions),bins=50)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fac9fc5d490>
```