

# assignment4

March 4, 2024

## Importing Necessary Libraries

```
[1]: import pandas as pd
import numpy as np
```

C:\Users\aaakas\AppData\Local\Temp\ipykernel\_2824\2162656668.py:1:

DeprecationWarning:

Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),

(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)

but was not found to be installed on your system.

If this would cause problems for you,

please provide us feedback at <https://github.com/pandas-dev/pandas/issues/54466>

```
import pandas as pd
```

```
[2]: boston = pd.read_csv("assignment4-dataset.csv")
```

## Removing Missing Values

```
[3]: boston.replace("?", np.nan, inplace=True)
boston.isnull().sum()
```

```
[3]: CRIM      20
     ZN        20
     INDUS    20
     CHAS     20
     NOX       0
     RM        0
     AGE      20
     DIS       0
     RAD       0
     TAX       0
     PTRATIO   0
     B         0
     LSTAT    20
     MEDV      0
dtype: int64
```

```
[4]: count_CRIM = boston["CRIM"].value_counts()
count_ZN = boston["ZN"].value_counts()
count_INDUS = boston["INDUS"].value_counts()
count_CHAS = boston["CHAS"].value_counts()
count_AGE = boston["AGE"].value_counts()
count_LSTAT = boston["LSTAT"].value_counts()
```

```
[5]: boston["CRIM"].replace(np.NaN, count_CRIM.index[0], inplace=True)
boston["ZN"].replace(np.NaN, count_ZN.index[0], inplace=True)
boston["INDUS"].replace(np.NaN, count_INDUS.index[0], inplace=True)
boston["CHAS"].replace(np.NaN, count_CHAS.index[0], inplace=True)
boston["AGE"].replace(np.NaN, count_AGE.index[0], inplace=True)
boston["LSTAT"].replace(np.NaN, count_LSTAT.index[0], inplace=True)
```

C:\Users\aaakas\AppData\Local\Temp\ipykernel\_2824\973233367.py:1: FutureWarning:  
A value is trying to be set on a copy of a DataFrame or Series through chained  
assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work  
because the intermediate object on which we are setting values always behaves as  
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using  
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)  
instead, to perform the operation inplace on the original object.

```
boston["CRIM"].replace(np.NaN, count_CRIM.index[0], inplace=True)
```

C:\Users\aaakas\AppData\Local\Temp\ipykernel\_2824\973233367.py:2: FutureWarning:  
A value is trying to be set on a copy of a DataFrame or Series through chained  
assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work  
because the intermediate object on which we are setting values always behaves as  
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using  
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)  
instead, to perform the operation inplace on the original object.

```
boston["ZN"].replace(np.NaN, count_ZN.index[0], inplace=True)
```

C:\Users\aaakas\AppData\Local\Temp\ipykernel\_2824\973233367.py:3: FutureWarning:  
A value is trying to be set on a copy of a DataFrame or Series through chained  
assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work  
because the intermediate object on which we are setting values always behaves as  
a copy.

For example, when doing `'df[col].method(value, inplace=True)'`, try using `'df.method({col: value}, inplace=True)'` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

```
boston["INDUS"].replace(np.NaN, count_INDUS.index[0], inplace=True)
```

C:\Users\aaakas\AppData\Local\Temp\ipykernel\_2824\973233367.py:4: FutureWarning:  
A value is trying to be set on a copy of a DataFrame or Series through chained  
assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work  
because the intermediate object on which we are setting values always behaves as  
a copy.

For example, when doing `'df[col].method(value, inplace=True)'`, try using `'df.method({col: value}, inplace=True)'` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

```
boston["CHAS"].replace(np.NaN, count_CHAS.index[0], inplace=True)
```

C:\Users\aaakas\AppData\Local\Temp\ipykernel\_2824\973233367.py:5: FutureWarning:  
A value is trying to be set on a copy of a DataFrame or Series through chained  
assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work  
because the intermediate object on which we are setting values always behaves as  
a copy.

For example, when doing `'df[col].method(value, inplace=True)'`, try using `'df.method({col: value}, inplace=True)'` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

```
boston["AGE"].replace(np.NaN, count_AGE.index[0], inplace=True)
```

C:\Users\aaakas\AppData\Local\Temp\ipykernel\_2824\973233367.py:6: FutureWarning:  
A value is trying to be set on a copy of a DataFrame or Series through chained  
assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work  
because the intermediate object on which we are setting values always behaves as  
a copy.

For example, when doing `'df[col].method(value, inplace=True)'`, try using `'df.method({col: value}, inplace=True)'` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

```
boston["LSTAT"].replace(np.NaN, count_LSTAT.index[0], inplace=True)
```

```
[6]: boston.isnull().sum()
```

```
[6]: 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
      MEDV    0
      dtype: int64
```

### Linear Regression using Sklearn

```
[7]: from sklearn.linear_model import LinearRegression
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import mean_squared_error, r2_score
```

```
[8]: x = boston.loc[:, boston.columns != "MEDV"]
      y = boston["MEDV"]
```

```
[9]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
      ↪ random_state=1)
```

```
[10]: model = LinearRegression()
```

```
[11]: model.fit(x_train, y_train)
```

```
[11]: LinearRegression()
```

```
[12]: y_pred = model.predict(x_test)
```

### Find MSE and R2 Score

```
[13]: mse = mean_squared_error(y_test, y_pred)
      mse
```

```
[13]: 22.29910362810643
```

```
[14]: r2_score(y_test, y_pred)
```

```
[14]: 0.7748894887292299
```

## Plotting Regression Line using Seaborn

```
[15]: import seaborn as sns
```

```
[16]: sns.regplot(data = boston, x=y_pred, y=y_test, fit_reg=True,
↳scatter_kws={"alpha":0.4}, line_kws={"color":"red"})

plt.title("Scatter Plot with Regression Line")
plt.show()
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[16], line 3
      1 sns.regplot(data = boston, x=y_pred, y=y_test, fit_reg=True,
↳scatter_kws={"alpha":0.4}, line_kws={"color":"red"})
----> 3 plt.title("Scatter Plot with Regression Line")
      4 plt.show()

NameError: name 'plt' is not defined
```

