```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
from sklearn.datasets import fetch_openml



data_url = "http://lib.stat.cmu.edu/datasets/boston"

# Read the dataset from the URL
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)


boston = fetch_openml(name='boston', version=3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1027: UserWarning: Version 3 of dataset boston is inactive, mear
  warn(
```

```
print(boston)
```

```
502  0.04527   0.0  11.93    0  0.573  6.120  76.7  2.2875   1  273     21.0
503  0.06076   0.0  11.93    0  0.573  6.976  91.0  2.1675   1  273     21.0
504  0.10959   0.0  11.93    0  0.573  6.794  89.3  2.3889   1  273     21.0
505  0.04741   0.0  11.93    0  0.573  6.030  80.8  2.5050   1  273     21.0

          B  LSTAT
0    396.90   4.98
1    396.90   9.14
2    392.83   4.03
3    394.63   2.94
4    396.90   5.33
..      ...    ...
501  391.99   9.67
502  396.90   9.08
503  396.90   5.64
504  393.45   6.48
505  396.90   7.88

[506 rows x 13 columns], 'target': 0      N
1      P
2      N
3      N
4      N
      ..
501    P
502    P
503    N
504    P
505    P
Name: binaryClass, Length: 506, dtype: category
Categories (2, object): ['N', 'P'], 'frame':         CRIM    ZN  INDUS CHAS    NOX     RM   AGE     DIS RAD  TAX  PTRATIO  \
0    0.00632  18.0   2.31    0  0.538  6.575  65.2  4.0900   1  296     15.3
1    0.02731   0.0   7.07    0  0.469  6.421  78.9  4.9671   2  242     17.8
2    0.02729   0.0   7.07    0  0.469  7.185  61.1  4.9671   2  242     17.8
3    0.03237   0.0   2.18    0  0.458  6.998  45.8  6.0622   3  222     18.7
4    0.06905   0.0   2.18    0  0.458  7.147  54.2  6.0622   3  222     18.7
..       ...   ...    ...  ...    ...    ...   ...     ...  ..  ...      ...
501  0.06263   0.0  11.93    0  0.573  6.593  69.1  2.4786   1  273     21.0
502  0.04527   0.0  11.93    0  0.573  6.120  76.7  2.2875   1  273     21.0
503  0.06076   0.0  11.93    0  0.573  6.976  91.0  2.1675   1  273     21.0
504  0.10959   0.0  11.93    0  0.573  6.794  89.3  2.3889   1  273     21.0
505  0.04741   0.0  11.93    0  0.573  6.030  80.8  2.5050   1  273     21.0

          B  LSTAT binaryClass
0    396.90   4.98           N
1    396.90   9.14           P
2    392.83   4.03           N
3    394.63   2.94           N
4    396.90   5.33           N
..      ...    ...         ...
501  391.99   9.67           P
502  396.90   9.08           P
503  396.90   5.64           N
504  393.45   6.48           P
505  396.90   7.88           P

[506 rows x 14 columns], 'categories': None, 'feature_names': ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', '
```

```
house_price_dataframe = pd.DataFrame(boston.data)
```

```
house_price_dataframe.head(5)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

Next steps: Generate code with house_price_dataframe | View recommended plots | New interactive sheet

```
target = raw_df.values[1::2, 2]
```

```
house_price_dataframe["Price"] = target
```

```
house_price_dataframe.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price	Price
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2	36.2

Next steps: Generate code with house_price_dataframe | View recommended plots | New interactive sheet

```
house_price_dataframe = house_price_dataframe.drop('price', axis=1)
```

```
house_price_dataframe.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
house_price_dataframe = house_price_dataframe.astype(float)
```

```
house_price_dataframe.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Next steps: Generate code with house_price_dataframe | View recommended plots | New interactive sheet

```
house_price_dataframe.shape
```

```
(506, 14)
```

```
house_price_dataframe.isnull().sum()
```

	0
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

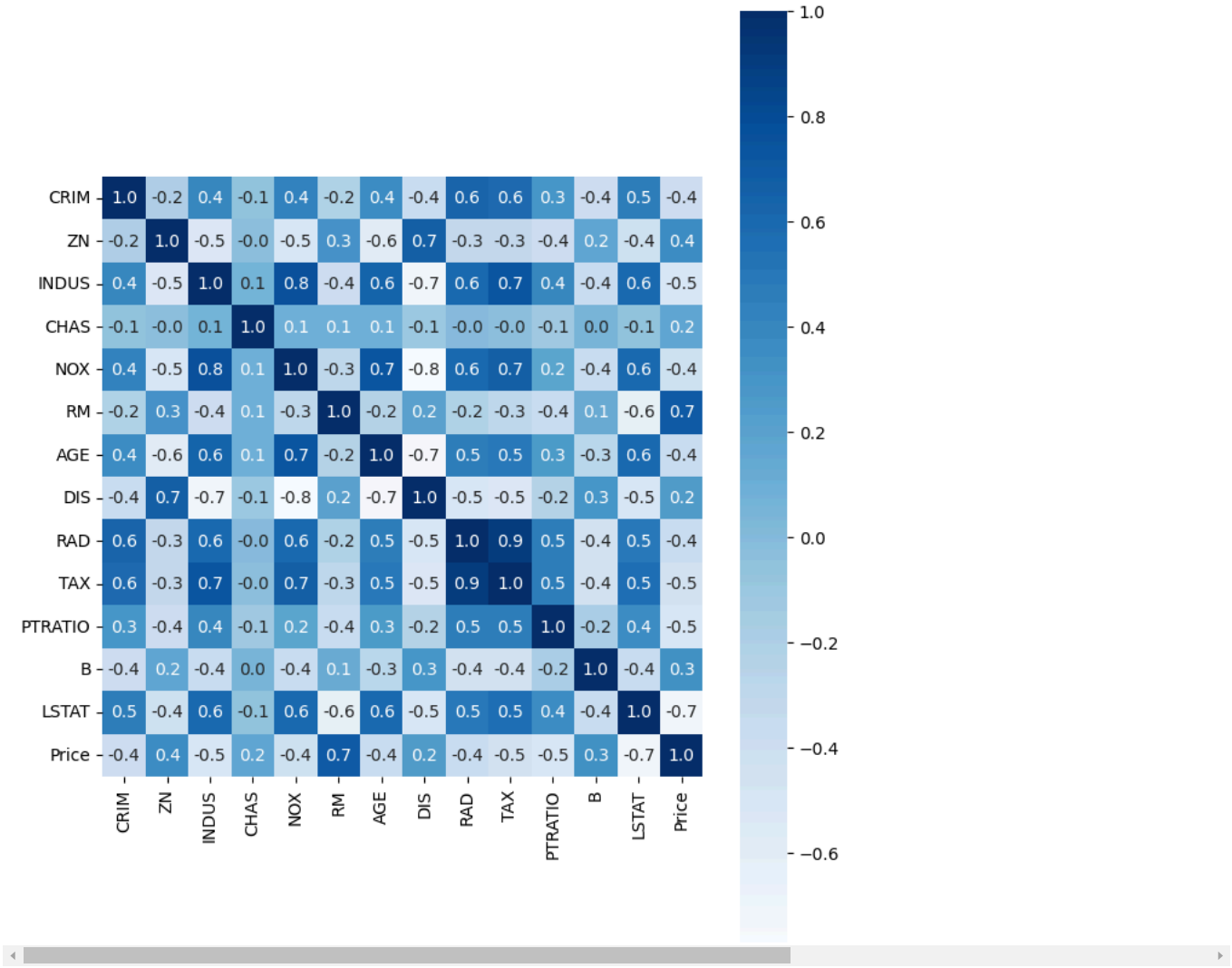
```
house_price_dataframe.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATI(
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00000(
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.45553
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.16494
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.60000(
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.40000(
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.05000(
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.20000(
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.00000(

```
correlation = house_price_dataframe.corr()
```

```
plt.figure(figsize = (8, 10))
sns.heatmap(correlation,
            cbar = True, # side color bar
            square = True, # square the boxes
            fmt='.1f', # decimal numbers after decimal .1f mean 1 digit after point(.2f mean 2 digits)
            annot= True, # feartures and the values will show
            annot_kws= {'size':10}, # size of the numbers in box
            cmap='Blues')
```

```
<Axes: >
```




```
X = house_price_dataframe.drop(['Price'], axis = 1)
Y = house_price_dataframe['Price']


display(X , Y)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

	Price
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
...	...
501	22.4
502	20.6
503	23.9
504	22.0
505	11.9

506 rows × 1 columns

dtype: float64

Next steps: Generate code with X View recommended plots New interactive sheet

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state= 2)
```

```
model=XGBRegressor()
```

```
model.fit(X_train, Y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

Prediction on training data

```
training_data_prediction = model.predict(X_train)
```

```
score_1 = metrics.r2_score(Y_train, training_data_prediction)
```

```
score_2 = metrics.mean_absolute_error(Y_train, training_data_prediction
```

```
print("R Square error : ", score_1) # 0 mean perfect

print("Mean Absolute Error : ", score_2)
```

```
R Square error :  0.9999980039471451
Mean Absolute Error :  0.0091330346494618
```

Prediction on test data

```
test_data_prediction = model.predict(X_test)


score_1 = metrics.r2_score(Y_test, test_data_prediction)


score_2 = metrics.mean_absolute_error(Y_test, test_data_prediction)


print("R Square error : ", score_1) # 0 mean perfect

print("Mean Absolute Error : ", score_2)
```

```
R Square error :  0.9051721149855378
Mean Absolute Error :  2.9748727686264927
```