Dragon Real Estate - Price Predictor

```
import pandas as pd
```

```
housing = pd.read_csv("data.csv")
```

```
housing.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
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

```
housing.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    int64
 4   NOX      506 non-null    float64
 5   RM       501 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

```
housing['CHAS'].value_counts()
```

```
CHAS
0    471
1     35
Name: count, dtype: int64
```

```
housing.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	501.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.289537	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.65
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702907	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.14
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.73
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.888000	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.95
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.209000	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.36
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.629000	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.95
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.97

```
%matplotlib inline
```

```
# # For plotting histogram
# import matplotlib.pyplot as plt
# housing.hist(bins=50, figsize=(20, 15))
```

Train-Test Splitting

```
# For learning purpose
import numpy as np
def split_train_test(data, test_ratio):
    np.random.seed(42)
    shuffled = np.random.permutation(len(data))
    print(shuffled)
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled[:test_set_size]
    train_indices = shuffled[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

```
# train_set, test_set = split_train_test(housing, 0.2)
```

```
# print(f"Rows in train set: {len(train_set)}\nRows in test set: {len(test_set)}\n")
```

```
from sklearn.model_selection import train_test_split
train_set, test_set  = train_test_split(housing, test_size=0.2, random_state=42)
print(f"Rows in train set: {len(train_set)}\nRows in test set: {len(test_set)}\n")
```

```
Rows in train set: 404
Rows in test set: 102
```

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing['CHAS']):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

```
strat_test_set['CHAS'].value_counts()
```

```
CHAS
0    95
1     7
Name: count, dtype: int64
```

```
strat_train_set['CHAS'].value_counts()
```

```
CHAS
0    376
1     28
Name: count, dtype: int64
```

```
# 95/7
```

```
# 376/28
```

```
housing = strat_train_set.copy()
```

Looking for Correlations

```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

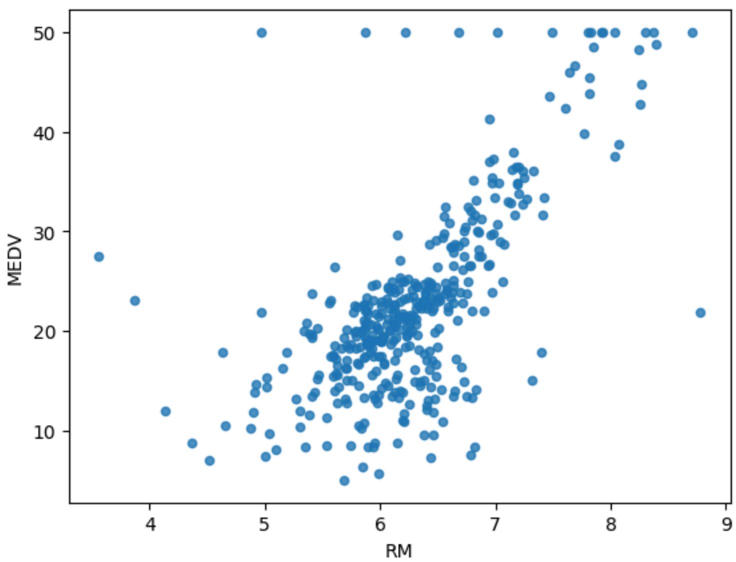
```
MEDV       1.000000
RM         0.680118
B          0.361761
ZN         0.339741
DIS        0.240451
CHAS       0.205066
AGE       -0.364596
RAD       -0.374693
CRIM      -0.393715
NOX       -0.422873
TAX       -0.456657
INDUS     -0.473516
PTRATIO   -0.493534
```

```
LSTAT      -0.740494
Name: MEDV, dtype: float64
```

```
# from pandas.plotting import scatter_matrix
# attributes = ["MEDV", "RM", "ZN", "LSTAT"]
# scatter_matrix(housing[attributes], figsize = (12,8))
```

```
housing.plot(kind="scatter", x="RM", y="MEDV", alpha=0.8)
```

```
<Axes: xlabel='RM', ylabel='MEDV'>
```



Trying out Attribute combinations

```
housing["TAXRM"] = housing['TAX']/housing['RM']
```

```
housing.head()
```

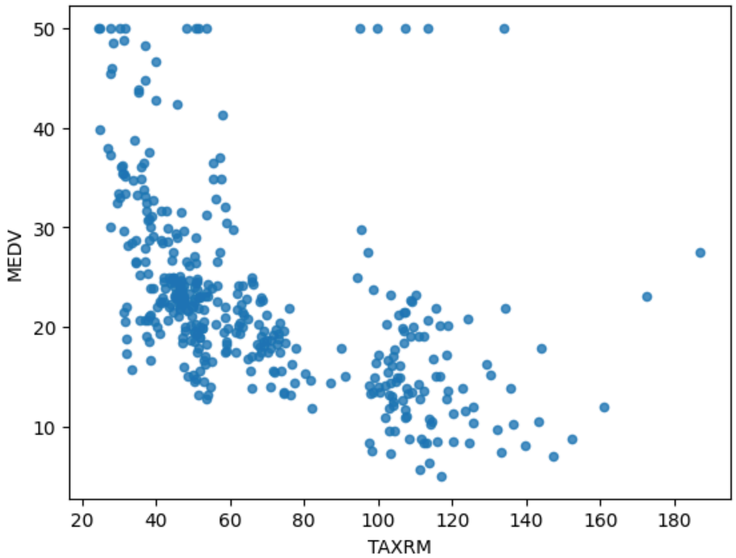
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	TAXRM
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6.57	21.9	51.571709
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5.99	24.5	42.200452
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68	16.7	102.714374
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	19.6	396.90	6.87	23.1	45.012547
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	19.6	396.90	6.15	23.0	45.468948

```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

```
MEDV        1.000000
RM          0.680118
B           0.361761
ZN          0.339741
DIS         0.240451
CHAS        0.205066
AGE        -0.364596
RAD        -0.374693
CRIM       -0.393715
NOX        -0.422873
TAX        -0.456657
INDUS      -0.473516
PTRATIO    -0.493534
TAXRM      -0.527283
LSTAT      -0.740494
Name: MEDV, dtype: float64
```

```
housing.plot(kind="scatter", x="TAXRM", y="MEDV", alpha=0.8)
```

```
<Axes: xlabel='TAXRM', ylabel='MEDV'>
```




```
housing = strat_train_set.drop("MEDV", axis=1)
housing_labels = strat_train_set["MEDV"].copy()
```

Missing Attributes

```
# To take care of missing attributes, you have three options:
#     1. Get rid of the missing data points
#     2. Get rid of the whole attribute
#     3. Set the value to some value(0, mean or median)
```

```
a = housing.dropna(subset=["RM"]) #Option 1
a.shape
# Note that the original housing dataframe will remain unchanged
```

```
(399, 13)
```

```
housing.drop("RM", axis=1).shape # Option 2
# Note that there is no RM column and also note that the original housing dataframe will remain unchanged
```

```
(404, 12)
```

```
median = housing["RM"].median() # Compute median for Option 3
```

```
housing["RM"].fillna(median) # Option 3
# Note that the original housing dataframe will remain unchanged
```

```
254    6.108
348    6.635
476    6.484
321    6.376
326    6.312
       ...
155    6.152
423    6.103
98     7.820
455    6.525
216    5.888
Name: RM, Length: 404, dtype: float64
```

```
housing.shape
```

```
(404, 13)
```

```
housing.describe() # before we started filling missing attributes
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	404.000000	404.000000	404.000000	404.000000	404.000000	399.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.00
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.286005	69.039851	3.746210	9.735149	412.341584	18.473267	353.392822	12.79
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.713507	28.258248	2.099057	8.731259	168.672623	2.129243	96.069235	7.23
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.000000	13.000000	0.320000	1.73
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.882000	44.850000	2.035975	4.000000	284.000000	17.400000	374.617500	6.84
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.216000	78.200000	3.122200	5.000000	337.000000	19.000000	390.955000	11.57
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.633000	94.100000	5.100400	24.000000	666.000000	20.200000	395.630000	17.10
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	36.98

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
imputer.fit(housing)
```

```
SimpleImputer
SimpleImputer(strategy='median')
```

```
imputer.statistics_
```

```
array([2.86735e-01, 0.00000e+00, 9.90000e+00, 0.00000e+00, 5.38000e-01,
       6.21600e+00, 7.82000e+01, 3.12220e+00, 5.00000e+00, 3.37000e+02,
       1.90000e+01, 3.90955e+02, 1.15700e+01])
```

```
X = imputer.transform(housing)
```

```
housing_tr = pd.DataFrame(X, columns=housing.columns)
```

```
housing_tr.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.00
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.285139	69.039851	3.746210	9.735149	412.341584	18.473267	353.392822	12.79
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.709110	28.258248	2.099057	8.731259	168.672623	2.129243	96.069235	7.23
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.000000	13.000000	0.320000	1.73
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.884750	44.850000	2.035975	4.000000	284.000000	17.400000	374.617500	6.84
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.216000	78.200000	3.122200	5.000000	337.000000	19.000000	390.955000	11.57
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630250	94.100000	5.100400	24.000000	666.000000	20.200000	395.630000	17.10
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	36.98

Scikit-learn Design

Primarily, three types of objects

1. Estimators - It estimates some parameter based on a dataset. Eg. imputer. It has a fit method and transform method. Fit method - Fits the dataset and calculates internal parameters
2. Transformers - transform method takes input and returns output based on the learnings from fit(). It also has a convenience function called fit_transform() which fits and then transforms.
3. Predictors - LinearRegression model is an example of predictor. fit() and predict() are two common functions. It also gives score() function which will evaluate the predictions.

Feature Scaling

Primarily, two types of feature scaling methods:

1. Min-max scaling (Normalization) (value - min)/(max - min) Sklearn provides a class called MinMaxScaler for this

2. Standardization (value - mean)/std Sklearn provides a class called StandardScaler for this

Creating a Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    #     ..... add as many as you want in your pipeline
    ('std_scaler', StandardScaler()),
])
```

```
housing_num_tr = my_pipeline.fit_transform(housing)
```

```
housing_num_tr.shape
```

```
(404, 13)
```

Selecting a desired model for Dragon Real Estates

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
# model = LinearRegression()
# model = DecisionTreeRegressor()
model = RandomForestRegressor()
model.fit(housing_num_tr, housing_labels)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
some_data = housing.iloc[:5]
```

```
some_labels = housing_labels.iloc[:5]
```

```
prepared_data = my_pipeline.transform(some_data)
```

```
model.predict(prepared_data)
```

```
array([22.28 , 25.636, 16.6  , 23.39 , 23.427])
```

```
list(some_labels)
```

```
[21.9, 24.5, 16.7, 23.1, 23.0]
```

Evaluating the model

```
from sklearn.metrics import mean_squared_error
housing_predictions = model.predict(housing_num_tr)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)
```

```
rmse
```

```
1.248558644244959
```

Using better evaluation technique - Cross Validation

```
# 1 2 3 4 5 6 7 8 9 10
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, housing_num_tr, housing_labels, scoring="neg_mean_squared_error", cv=10)
rmse_scores = np.sqrt(-scores)
```

```
rmse_scores
```

```
array([2.86723166, 2.81231295, 4.3748573 , 2.74891253, 3.32402344,
       2.64400338, 4.89236806, 3.3339339 , 3.45804938, 3.19782011])
```

```
def print_scores(scores):
    print("Scores:", scores)
    print("Mean: ", scores.mean())
    print("Standard deviation: ", scores.std())
```

```
print_scores(rmse_scores)
```

```
Scores: [2.86723166 2.81231295 4.3748573  2.74891253 3.32402344 2.64400338
 4.89236806 3.3339339  3.45804938 3.19782011]
Mean:  3.365351272268311
Standard deviation:  0.6960281165722647
```

Quiz: Convert this notebook into a python file and run the pipeline using Visual Studio Code

Saving the model

```
from joblib import dump, load
dump(model, 'Dragon.joblib')
```

```
['Dragon.joblib']
```

Testing the model on test data

```
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
# print(final_predictions, list(Y_test))
```

```
final_rmse
```

```
2.9379472659077632
```

```
prepared_data[0]
```

```
array([-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
       -0.25011401, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
       -0.97491834,  0.41164221, -0.86091034])
```

Using the model

```
from joblib import dump, load
import numpy as np
model = load('Dragon.joblib')
features = np.array([[-5.43942006, 4.12628155, -1.6165014, -0.67288841, -1.42262747,
       -11.44443979304, -49.31238772,  7.61111401, -26.0016879 , -0.5778192 ,
       -0.97491834,  0.41164221, -66.86091034]])
model.predict(features)
```

```
array([24.783])
```

```
Start coding or generate with AI.
```