Presented by Anniza Mega

Boston House Price Prediction

A journey through the Global Economy

Python Link

Introduction

In this project we are going to use Machine Learning to predict the house prices of city named Boston in US.

The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970.

There are several features given for a house and we have to predicts its value as accurate as possible.

Dataset link

Python link

Step of price prediction

1

Importong libraries and dataset that we need to use it. Overviesw the dataset and define of each column.

2

Detecting Outliners and Handling the outliners. After overview the dataset ,I need to detect outliner first and make a decision what will we do with outliners

3

House Price Prediction. After Handling outliner we can try to predict the price using 3 models of forcasting. In this case I will use Linear Regression , Random Forest and Support vector Mechine

4

Evaluation Models. After do forecasting with three models let’s compare it and find whic one is the best model to predict the house price.

Tools and Libraries

Dataset Overview

Boston House Price dataset has 14 features and their description is given as follows:

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per dollar 10,000.
- PTRATIO pupil-teacher ratio by town
- B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in $1000's

Python link

	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	296.0	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	242.0	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	242.0	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	222.0	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	222.0	18.7	396.90	5.33	36.2

```
# viewing the bottom 5 values of the dataset

df.tail()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88	11.9

```
df.shape
```

```
(506, 14)
```

```
# Checking if they are any null values present in the dataset

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
MEDV       0
dtype: int64
```

Detecting and Handling Outliner

	Feature	Number of Outliers
0	CRIM	66
1	ZN	68
2	INDUS	0
3	CHAS	35
4	NOX	0
5	RM	30
6	AGE	0
7	DIS	5
8	RAD	0
9	TAX	0
10	PTRATIO	15
11	B	77
12	LSTAT	7
13	MEDV	40

Detecting the outliers in the continuous columns

```
[ ] # Function to detect outliers in every feature

    def detect_outliers(df):
      cols = list(df)
      outliers = pd.DataFrame(columns=['Feature', 'Number of Outliers'])
      for column in cols:
        if column in df.select_dtypes(include=np.number).columns:
          q1 = df[column].quantile(0.25)
          q3 = df[column].quantile(0.75)
          iqr = q3 - q1
          fence_low = q1 - (1.5*iqr)
          fence_high = q3 + (1.5*iqr)
          outliers = pd.concat([outliers, pd.DataFrame({'Feature': [column], 'Number of Outliers': [df.loc[(df[column] < fence_low) | (df[column] > fence_high
      return outliers

    detect_outliers(df)
```

```
[ ] detect_outliers(df)
```

	Feature	Number of Outliers
0	CRIM	66
1	ZN	68
2	INDUS	0
3	CHAS	0
4	NOX	0
5	RM	0
6	AGE	0
7	DIS	0
8	RAD	0
9	TAX	0
10	PTRATIO	0
11	B	77
12	LSTAT	0
13	MEDV	0

Treating outliers in the continuous columns

```
from scipy.stats.mstats import winsorize
```

```
# Function to treat outliers

def treat_outliers(dataframe):
    cols = list(dataframe)
    for col in cols:
        if col in dataframe.select_dtypes(include=np.number).columns:
            dataframe[col] = winsorize(dataframe[col], limits=[0.05, 0.1],inclusive=(True, True))

    return dataframe


df = treat_outliers(df)

# Checking for outliers after applying winsorization
# We see this using a fuction called 'detect_outliers', defined above.

detect_outliers(df)
```

The outliers displayed above in the CRIM, ZN, and B columns are not actually outliers; instead, they represent the predominant values within our dataset

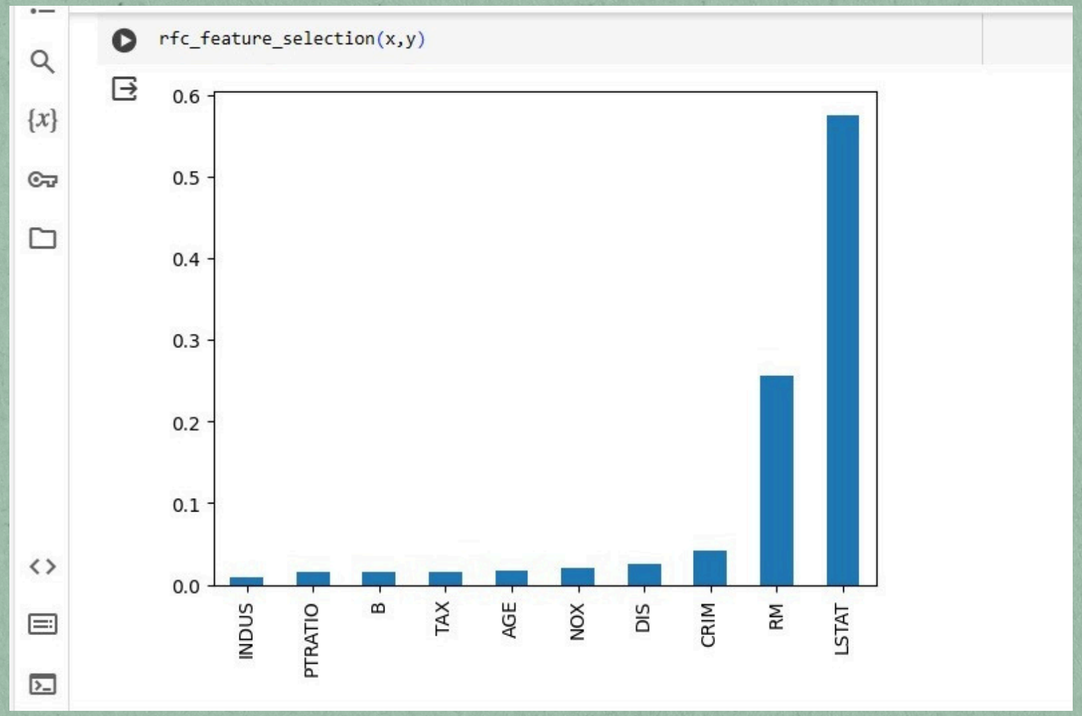
Python link

House Price Prediction

```
[ ]  # Predictors

     x = df.iloc[:,:-1]

     # This means that we are using all the columns, except 'MEDV', to predict the house price

     # Target

     y = df.iloc[:,-1]

     # This is because MEDV is the 'Median value of owner-occupied homes in $1000s'.
     # This shows that this is what we need to predict. So we call it the target variable.
```

```
[ ]  def rfc_feature_selection(dataset,target):
         X_train, X_test, y_train, y_test = train_test_split(dataset, target, test_size=0.2, random_state=42)
         rfc = RandomForestRegressor(random_state=42)
         rfc.fit(X_train, y_train)
         y_pred = rfc.predict(X_test)
         rfc_importances = pd.Series(rfc.feature_importances_, index=dataset.columns).sort_values().tail(10)
         rfc_importances.plot(kind='bar')
         plt.show()
```

```
rfc_feature_selection(x,y)
```



House Price Prediction

"It's evident from the feature importance analysis that certain features significantly influence house price prediction. LSTAT, RM, DIS, and CRIM emerge as the most crucial predictors based on their respective importance rankings. This suggests that other columns in the dataset may not contribute significantly to accurate house price predictions."

Pythonlink

1. Linear Regression (Training Data)

```
[ ] y_pred=lr.predict(xtrain)
```

Model Evaluation

- Training Data

```
[ ] print("R^2: ",metrics.r2_score(ytrain, y_pred))
    print("Adusted R^2: ", 1-(1-metrics.r2_score(ytrain, y_pred))*(len(ytrain)-1)/(len(ytrain)-xtrain.shape[1]-1))
    print("MAE: ", metrics.mean_absolute_error(ytrain, y_pred))
    print("MSE: ", metrics.mean_squared_error(ytrain, y_pred))
    print("RMSE: ",np.sqrt(metrics.mean_squared_error(ytrain, y_pred)))
```

```
R^2:  0.756033827767427
Adusted R^2:  0.7535880516046944
MAE:  2.790765406558843
MSE:  12.525910446232047
RMSE:  3.5391962994770503
```

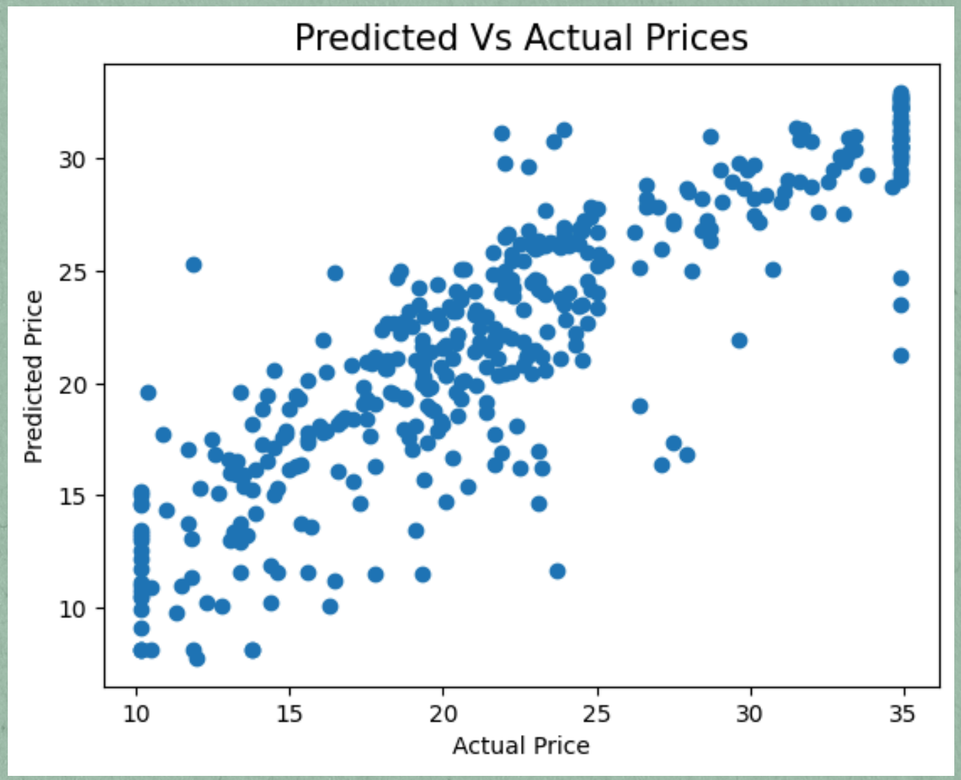
```
[ ] print(metrics.max_error(ytrain, y_pred))
```

```
13.629839810655223
```


```
[ ] print(metrics.max_error(ytrain, y_pred))
```

```
13.629839810655223
```


```
[ ] # visualizing the difference between the actual and predicted price

    plt.scatter(ytrain, y_pred)
    plt.xlabel("Actual Price")
    plt.ylabel("Predicted Price")
    plt.title("Predicted Vs Actual Prices", fontsize=15)
    plt.show()
```



1. Linear Regression (Test Data)

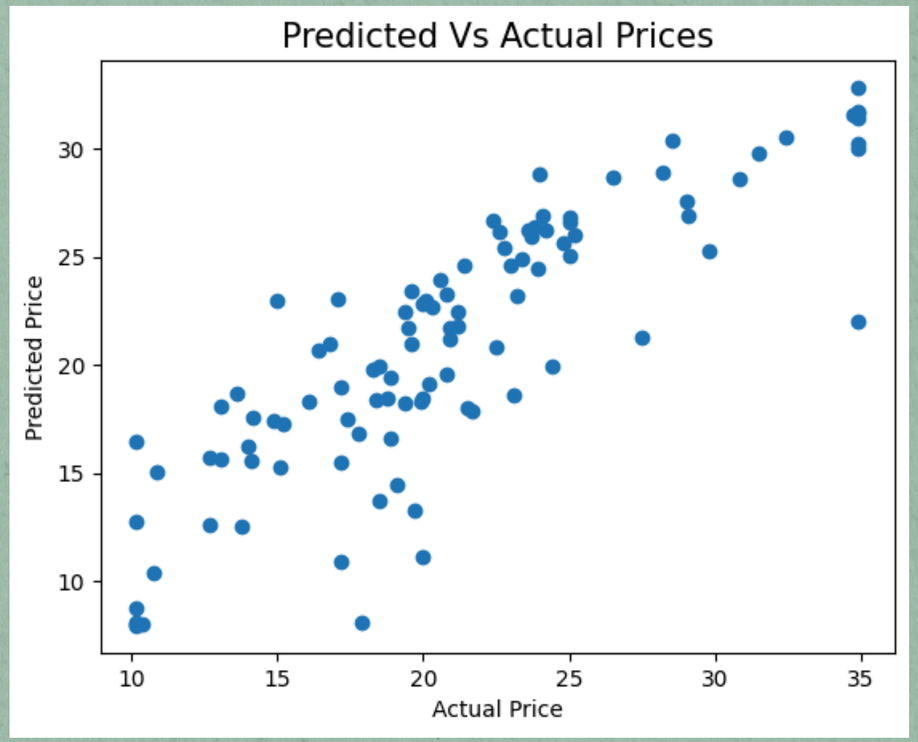
Test data

```
[ ] # Predicting the Test data with model

    ytest_pred=lr.predict(xtest)

    lin_acc=metrics.r2_score(ytest, ytest_pred)
    print("R^2: ",lin_acc)
    print("Adusted R^2: ", 1-(1-metrics.r2_score(ytest, ytest_pred))*(len(ytest)-1)/(len(ytest)-xtest.shape[1]-1))
    print("MAE: ", metrics.mean_absolute_error(ytest, ytest_pred))
    print("MSE: ", metrics.mean_squared_error(ytest, ytest_pred))
    print("RMSE: ",np.sqrt(metrics.mean_squared_error(ytest, ytest_pred)))
```

```
R^2:  0.7115739765489515
Adusted R^2:  0.6996801199117948
MAE:  2.770324519556731
MSE:  12.254732709613286
RMSE:  3.500676036084071
```

```
[ ] print(metrics.max_error(ytest, ytest_pred))
```

```
12.913450562946032
```



Interpretation:

- R^2 and Adjusted R^2 values show that a significant portion of the variance in house prices is captured by your model, although it performs slightly less well on the test data than on the training data.
- MAE, MSE, and RMSE values are very close between training and test datasets, indicating that the average prediction error remains consistent and low.
- Max Error is slightly lower on the test data, suggesting that the model might be slightly better at handling the worst-case predictions on new data.

Overall, the model seems to generalize well from training to test data, maintaining similar accuracy and error metrics. This consistency suggests that your linear regression model is robust and performs well on both seen and unseen data. If you want to further improve the model, you could look into more advanced regression techniques, feature engineering, or regularization methods.

2. Random Forest

- Training Data

```
print("R^2: ",metrics.r2_score(ytrain, y_pred))
print("Adusted R^2: ", 1-(1-metrics.r2_score(ytrain, y_pred))*(len(ytrain)-1)/(len(ytrain)-xtrain.shape[1]-1))
print("MAE: ", metrics.mean_absolute_error(ytrain, y_pred))
print("MSE: ", metrics.mean_squared_error(ytrain, y_pred))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(ytrain, y_pred)))

print("\nMaximum Error: ",metrics.max_error(ytrain, y_pred))
```

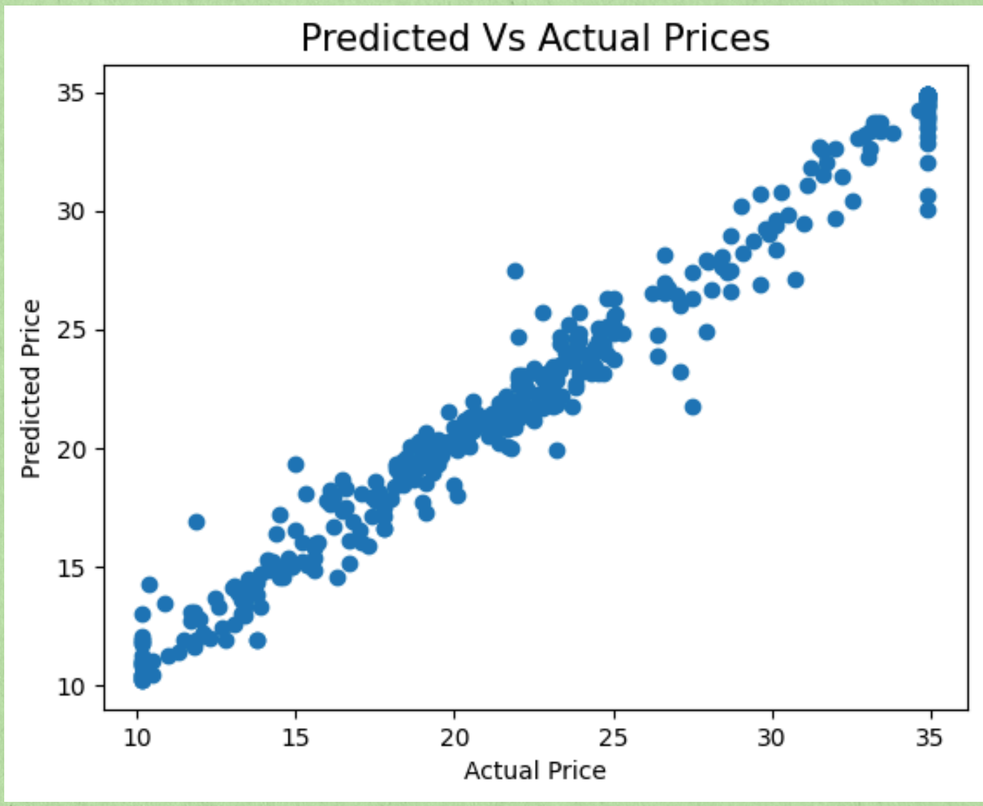
```
R^2:  0.9735482838765903
Adusted R^2:  0.9732831037650774
MAE:  0.7940439194812317
MSE:  1.3581056106218035
RMSE:  1.1653778831871675

Maximum Error:  5.7529999999999895
```

```
# visualizing the difference between the actual and predicted price

plt.scatter(ytrain, y_pred)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Predicted Vs Actual Prices", fontsize=15)
plt.show()
```

```
rfr= RandomForestRegressor()

rfr.fit(xtrain, ytrain)
```

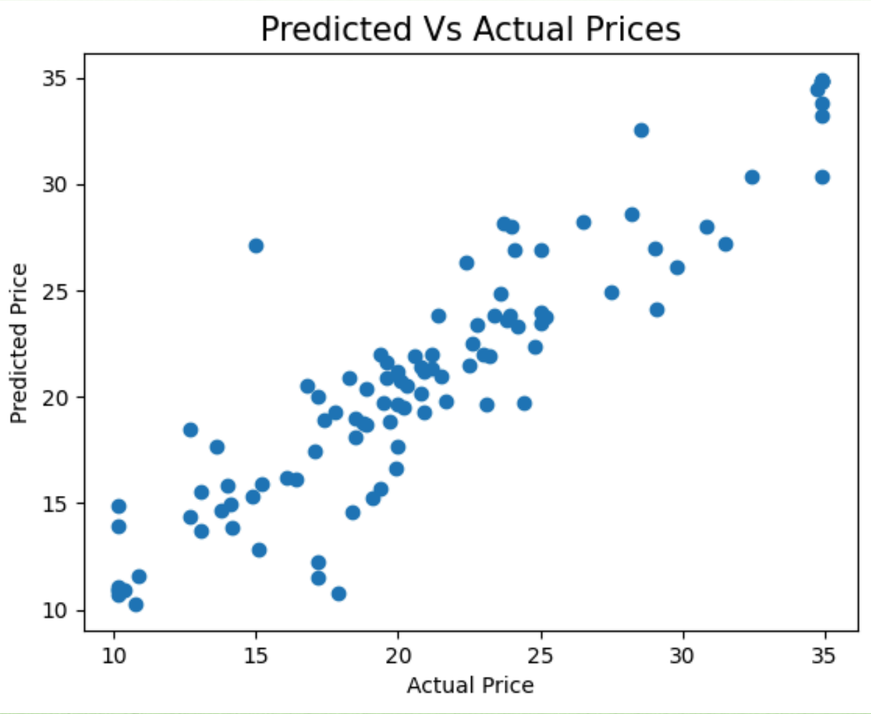
RandomForestRegressor
RandomForestRegressor()

```
y_pred=rfr.predict(xtrain)
```



2. Random Forest

```
[ ]  # Predicting the Test data with model

     ytest_pred=rfr.predict(xtest)

     rfr_acc=metrics.r2_score(ytest, ytest_pred)
     print("R^2: ",rfr_acc)
     print("Adusted R^2: ", 1-(1-metrics.r2_score(ytest, ytest_pred))*(len(ytest)-1)/(len(ytest)-xtest.shape[1]-1))
     print("MAE: ", metrics.mean_absolute_error(ytest, ytest_pred))
     print("MSE: ", metrics.mean_squared_error(ytest, ytest_pred))
     print("RMSE: ",np.sqrt(metrics.mean_squared_error(ytest, ytest_pred)))

     print("\nMaximum Error: ",metrics.max_error(ytest, ytest_pred))

     R^2:  0.8316798685798867
     Adusted R^2:  0.8247388322326655
     MAE:  1.8862285074130654
     MSE:  7.1516369969665785
     RMSE:  2.674254474982996

     Maximum Error:  12.143000000000008
```



Interpretation:

- Higher R^2 and Adjusted R^2 values indicate that the Random Forest model explains more variability in the house prices.
- Lower MAE, MSE, and RMSE values indicate that the Random Forest model has more accurate and reliable predictions, with smaller average errors.
- Lower Maximum Error indicates that the worst-case prediction is slightly better with the Random Forest model.

Overall, the Random Forest model offers a substantial improvement over the Linear Regression model in terms of predictive accuracy and error metrics. This suggests that the Random Forest model is better suited for predicting house prices in this dataset, likely due to its ability to capture complex relationships and interactions between the features.

3. Support Vector Mechine

3.Support Vector Machine (SVM)

```
[ ] svm_reg=svm.SVR()
    svm_reg.fit(xtrain, ytrain)
```

```
SVR()
```

```
[ ] y_pred=svm_reg.predict(xtrain)
```

Model Evaluation

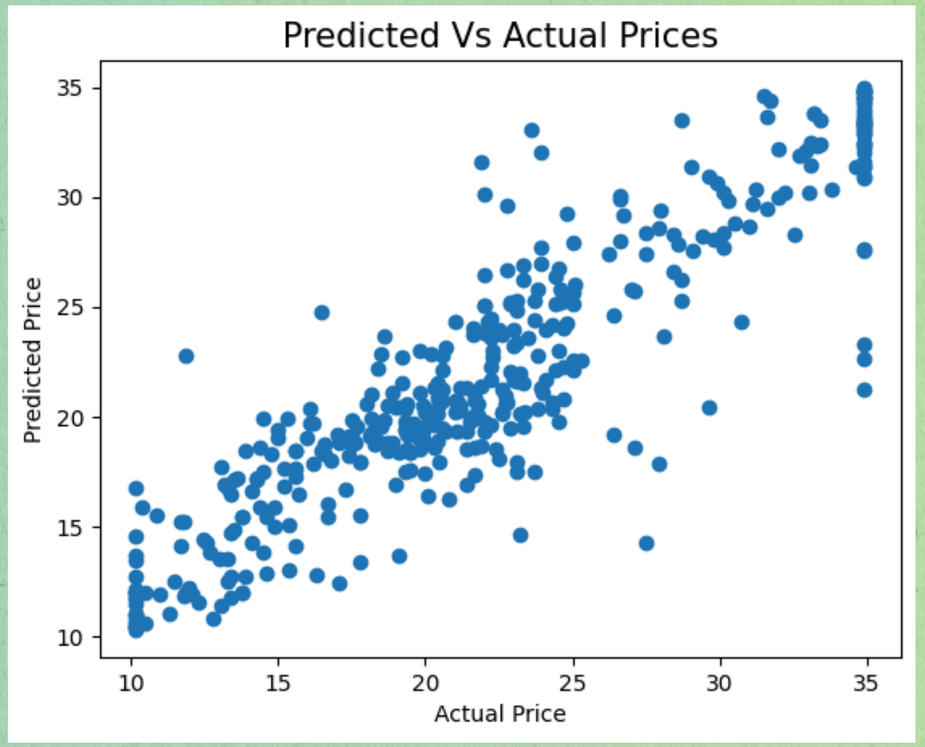
- Training Data

+ Code + Text

```
[ ] print("R^2: ",metrics.r2_score(ytrain, y_pred))
    print("Adusted R^2: ", 1-(1-metrics.r2_score(ytrain, y_pred))*(len(ytrain)-1)/(len(ytrain)-xtrain.shape[1]-1))
    print("MAE: ", metrics.mean_absolute_error(ytrain, y_pred))
    print("MSE: ", metrics.mean_squared_error(ytrain, y_pred))
    print("RMSE: ",np.sqrt(metrics.mean_squared_error(ytrain, y_pred)))

    print("\nMaximum Error: ",metrics.max_error(ytrain, y_pred))
```

```
R^2:  0.8212766305533041
Adusted R^2:  0.8194849175763949
MAE:  2.1734427566760885
MSE:  9.176161186002581
RMSE:  3.029217916559088

Maximum Error:  13.695622531656916
```

```
# visualizing the difference between the actual and predicted price

plt.scatter(ytrain, y_pred)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Predicted Vs Actual Prices", fontsize=15)
plt.show()
```

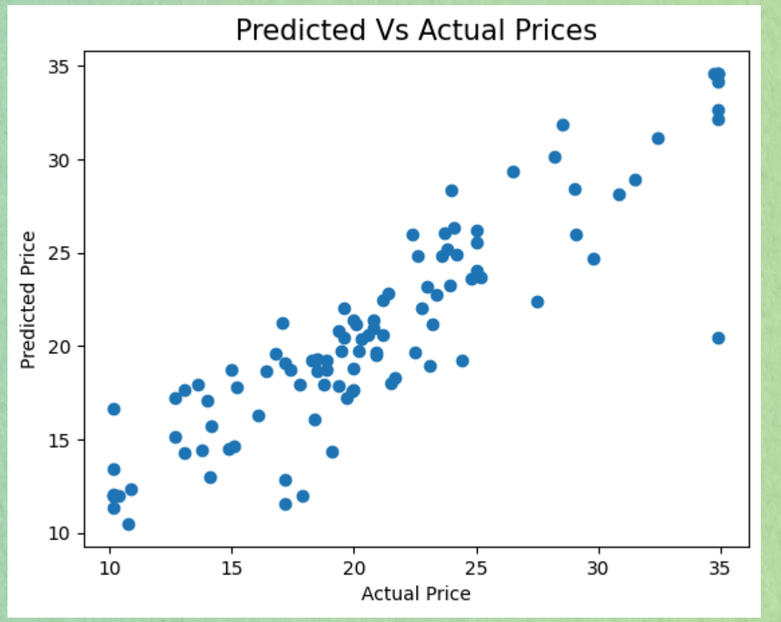


3. Support Vector Mechine (Test Data)

- Test Data

```
# Predicting the Test data with model

ytest_pred=svm_reg.predict(xtest)

svm_acc=metrics.r2_score(ytest, ytest_pred)
print("R^2: ",svm_acc)
print("Adusted R^2: ", 1-(1-metrics.r2_score(ytest, ytest_pred))*(len(ytest)-1)/(len(ytest)-xtest.shape[1]-1))
print("MAE: ", metrics.mean_absolute_error(ytest, ytest_pred))
print("MSE: ", metrics.mean_squared_error(ytest, ytest_pred))
print("RMSE: ",np.sqrt(metrics.mean_squared_error(ytest, ytest_pred)))

print("\nMaximum Error: ",metrics.max_error(ytest, ytest_pred))
```

```
R^2:  0.809820498710117
Adusted R^2:  0.8019780450486784
MAE:  2.062196717432315
MSE:  8.080404560133665
RMSE:  2.84260524169883

Maximum Error:  14.472162274524429
```



Interpretation :

- Higher R^2 and Adjusted R^2 values than Linear Regression indicate that the SVM model explains more variability in house prices, though less than Random Forest.
- Lower MAE, MSE, and RMSE values than Linear Regression indicate that the SVM model has more accurate and reliable predictions, with smaller average errors. However, these values are higher compared to the Random Forest model, suggesting that the Random Forest model is still the most accurate.
- Higher Maximum Error indicates that the worst-case prediction is higher with the SVM model compared to both Linear Regression and Random Forest, suggesting that the SVM model might have some outliers with significant prediction errors.

Summary

Overall, the SVM model offers a clear improvement over the Linear Regression model in terms of predictive accuracy and error metrics. However, the Random Forest model still outperforms both the Linear Regression and SVM models, indicating that it is the best-suited model for predicting house prices in this dataset. The higher maximum error in the SVM model suggests that it may not handle outliers as well as the Random Forest model.

Evaluation Model Comparison

Evaluation Comparison of all the 3 methods

```
[ ]  models=pd.DataFrame({
         'Model':['Linear Regression', 'Random Forest', 'Support Vector Machine'],
         'R_squared Score':[lin_acc*100, rfr_acc*100,svm_acc*100]
     })
     models.sort_values(by='R_squared Score', ascending=False)
```

	Model	R_squared Score
1	Random Forest	83.167987
2	Support Vector Machine	80.982050
0	Linear Regression	71.157398

Evaluation Model Comparison

Based on the (R^2) scores provided:

1. Random Forest: (R^2 = 83.167987)
2. Support Vector Machine: (R^2 = 80.982050)
3. Linear Regression: (R^2 = 71.157398)

The Random Forest model has the highest (R^2) score (83.16%), followed by the Support Vector Machine (80.98%), and then Linear Regression (71.16%).

So, in terms of (R^2) score, the Random Forest model appears to perform the best among the three models in capturing the variance in the dependent variable (house prices) based on the independent variables (features).

Presented by Anniza Mega

Thank you

Python Link