bostan Housing

```
https://www.kaggle.com/datasets/schirmerchad/bostonhoustingmlnd?
resource=download

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import sklearn
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

boston = pd.read_csv("C:/Users/Acer/Downloads/housingdata.csv")

boston.shape

(506, 14)

boston

        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX
\
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296

1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242

2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242

3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222

4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222

..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...

501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273

502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273

503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273

504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273

505  0.04741   0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273


     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
```

```
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90    NaN  36.2
..       ...     ...    ...   ...
501     21.0  391.99    NaN  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

```
boston .info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 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    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

```
boston = boston.dropna()
```

```
x = boston[boston.columns[ :-1]] # input 1st
y = boston[boston.columns[-1]]   #last columns as o/p
```

```
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=42)
```

```
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

```
(315, 13) (79, 13) (315,) (79,)
```

```
model=LinearRegression()
model.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_train_pred = model.predict(x_train)
y_test_pred = model.predict(x_test)

print(y_train_pred.shape,y_test_pred.shape)
```

```
(315,) (79,)
```

```
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)

print("Simple Linear Regression Model")
print(f"Training MSE : {train_mse:2f}")
print(f"Testing MSE : {test_mse:2f}")
```

```
Simple Linear Regression Model
Training MSE : 16.692213
Testing MSE : 31.454048
```

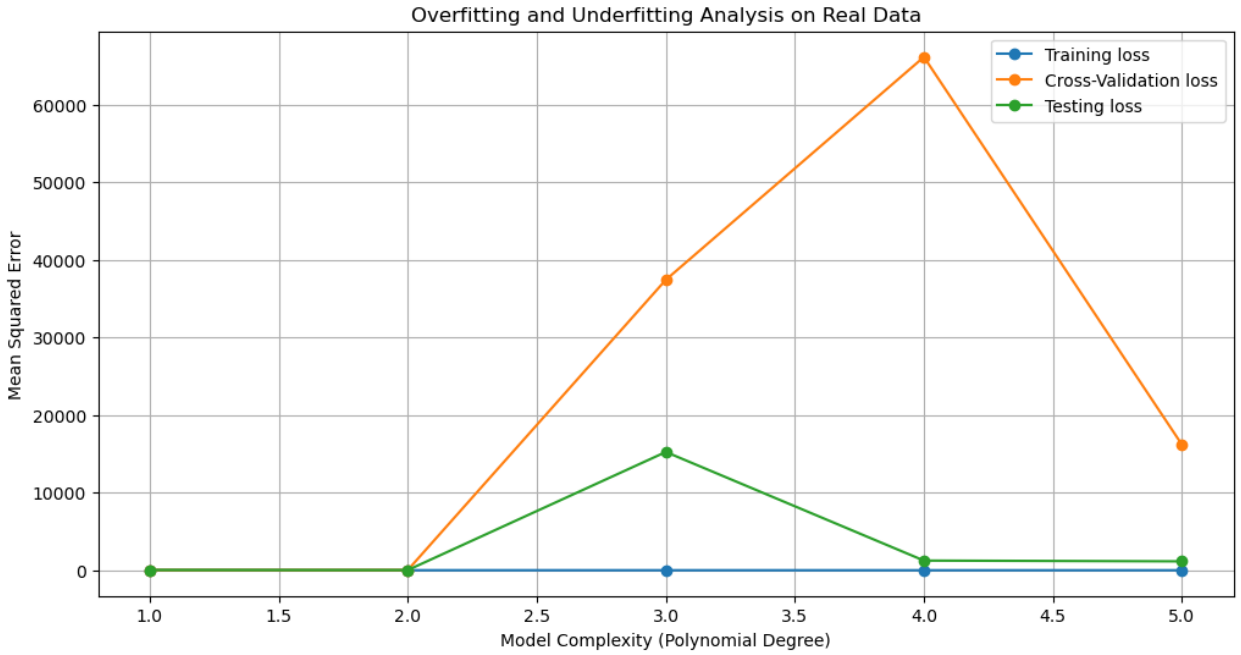
```
if train_mse < test_mse and test_mse - train_mse > 10 :
    print("The model might be overfitting i.e(low training error ,
high testing error)")
elif train_mse > test_mse :
        print("The model might be underfitting i.e(high training error
,low testing error)")
else:
        print("The model has a good balance between training error and
testing error")
```

```
The model might be overfitting i.e(low training error , high testing
error)
```

Cross Validation

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

#define a function to evaluate models with different complexities
def evaluate_model(degree):
#Create a pipeline with polynomial features and Linear regression
    model=Pipeline([
        ("poly_features", PolynomialFeatures(degree=degree,
include_bias=False)),
        ("lin_reg", LinearRegression()),
   ])
    #Cross-validation scores
    cross_val_scores = cross_val_score(model, x_train, y_train, cv=5,
scoring="neg_mean_squared_error")
    mean_cv_score= -np.mean(cross_val_scores)
```

```
    #Train the model on the full training set
    model.fit(x_train, y_train)

    #Predict on training and testing sets
    y_train_pred=model.predict(x_train)
    y_test_pred=model.predict(x_test)

    #Calculate errors
    train_error=mean_squared_error(y_train, y_train_pred)
    test_error=mean_squared_error(y_test, y_test_pred)

    return mean_cv_score, train_error, test_error

degrees=range(1, 6) #Polynomial degrees from 1 to 9
cv_errors, train_errors, test_errors=[],[],[]

for degree in degrees:
    cv_error, train_error, test_error = evaluate_model(degree)
    cv_errors.append(cv_error)
    train_errors.append(train_error)
    test_errors.append(test_error)

#Plot the errors
plt.figure(figsize=(12, 6))
plt.plot(degrees, train_errors, label="Training loss", marker="o")
plt.plot(degrees, cv_errors, label="Cross-Validation loss",
marker="o")
plt.plot(degrees, test_errors, label="Testing loss", marker="o")
plt.xlabel("Model Complexity (Polynomial Degree)")
plt.ylabel("Mean Squared Error")
plt.title("Overfitting and Underfitting Analysis on Real Data")
plt.legend()
plt.grid(True)
plt.show()
```


Overfitting and Underfitting Analysis on Real Data
Training loss
Cross-Validation loss
Testing loss
Mean Squared Error
Model Complexity (Polynomial Degree)
0
10000
20000
30000
40000
50000
60000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0