```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
import numpy as np
import pandas as pd

df =
pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master
/BostonHousing.csv')

df.head()
```

```
      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

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

```python
# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],
df.iloc[:,-1], test_size=0.2, random_state=2)

X_train
```

```
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax
\
321  0.18159   0.0   7.38     0  0.493  6.376  54.3  4.5404    5  287

37   0.08014   0.0   5.96     0  0.499  5.850  41.5  3.9342    5  279

286  0.01965  80.0   1.76     0  0.385  6.230  31.5  9.0892    1  241

2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242

25   0.84054   0.0   8.14     0  0.538  5.599  85.7  4.4546    4  307

..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
```

```
22   1.23247   0.0   8.14    0  0.538  6.142  91.7  3.9769   4  307
72   0.09164   0.0  10.81    0  0.413  6.065   7.8  5.2873   4  305
493  0.17331   0.0   9.69    0  0.585  5.707  54.0  2.3817   6  391
15   0.62739   0.0   8.14    0  0.538  5.834  56.5  4.4986   4  307
168  2.30040   0.0  19.58    0  0.605  6.319  96.1  2.1000   5  403

     ptratio       b  lstat
321     19.6  396.90   6.87
37      19.2  396.90   8.77
286     18.2  341.60  12.93
2       17.8  392.83   4.03
25      21.0  303.42  16.51
..       ...     ...    ...
22      21.0  396.90  18.72
72      19.2  390.91   5.52
493     19.2  396.90  12.01
15      21.0  395.62   8.47
168     14.7  297.09  11.10

[404 rows x 13 columns]
```

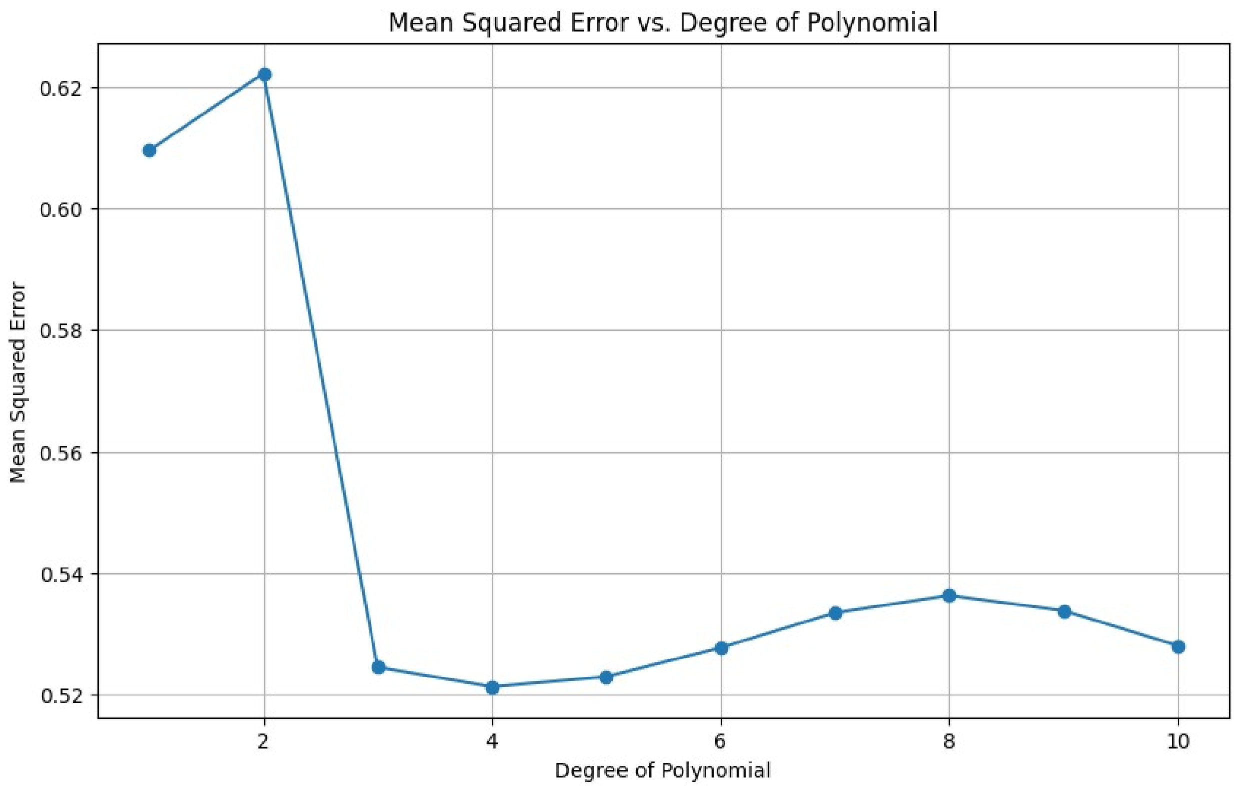
```python
# Create a linear regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Use the trained model to predict the target values in the test set
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error of the model on the test set
mse = mean_squared_error(y_test, y_pred)

print('Mean Squared Error:', mse)

r2 = r2_score(y_test, y_pred)

print('r2 score:', r2)
```

```
Mean Squared Error: 18.49542012244846
r2 score: 0.7789207451814409
```

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

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Load the Auto MPG dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-
mpg/auto-mpg.data"
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower',
'Weight', 'Acceleration', 'Model Year', 'Origin']
auto_mpg = pd.read_csv(url, names=column_names, delim_whitespace=True,
na_values='?')

# Drop rows with missing values
auto_mpg = auto_mpg.dropna()

# Define features and target
features = auto_mpg[['Horsepower']]  # Only use 'Horsepower' as a
feature
target = auto_mpg['MPG']

# List of polynomial degrees
degrees = list(range(1, 11))  # Extend degrees to 10

# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=4)

# List to store Mean Squared Errors
mse_list = []

for degree in degrees:
    # Add polynomial features
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    # Create a linear regression model
    model = LinearRegression()

    # Train the model on the polynomial features training data
    model.fit(X_train_poly, y_train)

    # Use the trained model to predict the target values in the test
set
    y_pred = model.predict(X_test_poly)

    # Calculate the Mean Squared Error of the model on the test set
    mse = mean_squared_error(y_test, y_pred)
```

```python
    # Store the Mean Squared Error in the list
    mse_list.append(mse)

# Plot the Mean Squared Error as a function of the degree of the
polynomial
plt.figure(figsize=(10, 6))
plt.plot(degrees, mse_list, marker='o')
plt.title('Mean Squared Error vs. Degree of Polynomial')
plt.xlabel('Degree of Polynomial')
plt.ylabel('Mean Squared Error')
plt.grid(True)
plt.show()
```



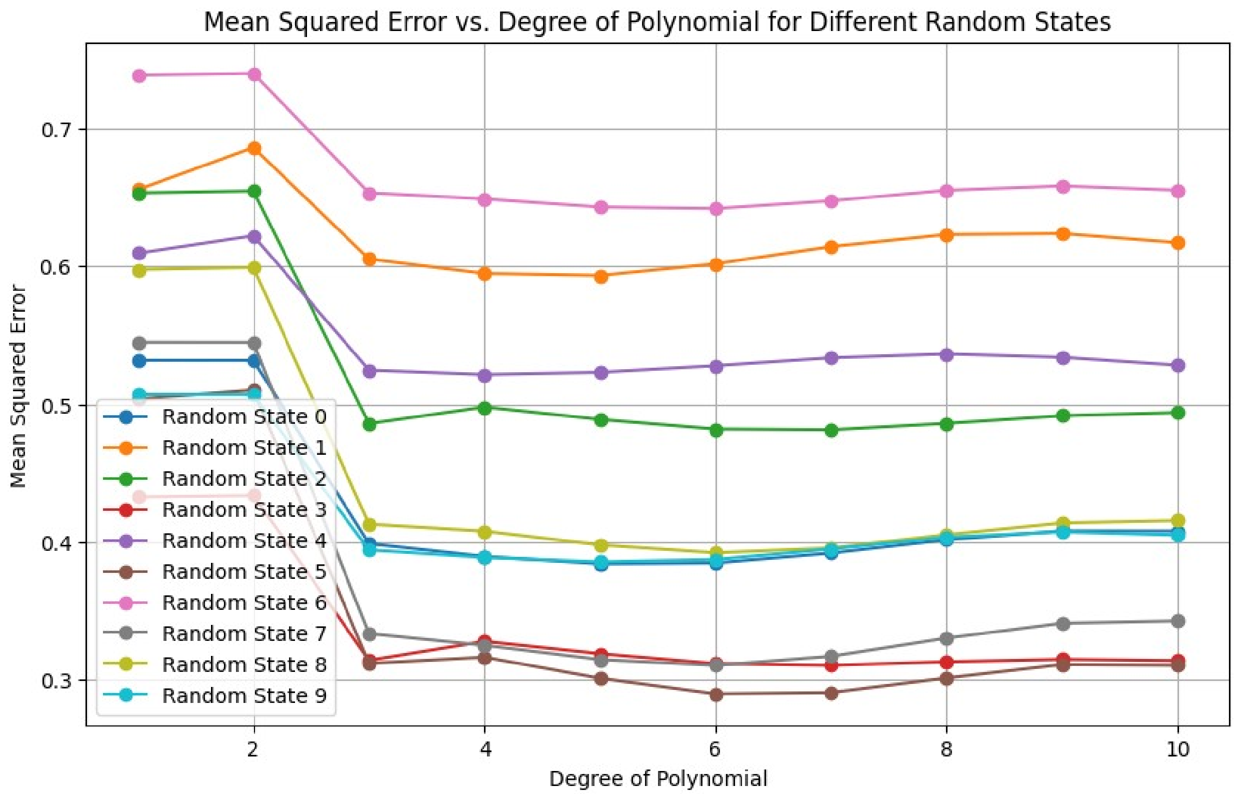

```
auto_mpg

      MPG  Cylinders  Displacement  Horsepower  Weight
Acceleration  \
18.0    8       307.0          130.0       3504.0    12.0              70

15.0    8       350.0          165.0       3693.0    11.5              70

18.0    8       318.0          150.0       3436.0    11.0              70

16.0    8       304.0          150.0       3433.0    12.0              70
```

```
17.0    8      302.0          140.0      3449.0    10.5             70

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

27.0    4      140.0           86.0      2790.0    15.6             82

44.0    4       97.0           52.0      2130.0    24.6             82

32.0    4      135.0           84.0      2295.0    11.6             82

28.0    4      120.0           79.0      2625.0    18.6             82

31.0    4      119.0           82.0      2720.0    19.4             82


      Model Year                     Origin
18.0           1  chevrolet chevelle malibu
15.0           1          buick skylark 320
18.0           1         plymouth satellite
16.0           1              amc rebel sst
17.0           1                ford torino
...          ...                        ...
27.0           1            ford mustang gl
44.0           2                  vw pickup
32.0           1              dodge rampage
28.0           1                ford ranger
31.0           1                 chevy s-10

[392 rows x 8 columns]
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Load the Auto MPG dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-
mpg/auto-mpg.data"
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower',
'Weight', 'Acceleration', 'Model Year', 'Origin']
auto_mpg = pd.read_csv(url, names=column_names, delim_whitespace=True,
na_values='?')

# Drop rows with missing values
auto_mpg = auto_mpg.dropna()

# Define features and target
features = auto_mpg[['Horsepower']]  # Only use 'Horsepower' as a
```

```
feature
target = auto_mpg['MPG']

# List of polynomial degrees
degrees = list(range(1, 11))  # Extend degrees to 10

# List of random states
random_states = list(range(10))

# Create a plot
plt.figure(figsize=(10, 6))

for i, random_state in enumerate(random_states):
    # Split the dataset into a training set and a test set
    X_train, X_test, y_train, y_test = train_test_split(features,
target, test_size=0.2, random_state=random_state)

    # List to store Mean Squared Errors
    mse_list = []

    for degree in degrees:
        # Add polynomial features
        poly = PolynomialFeatures(degree=degree)
        X_train_poly = poly.fit_transform(X_train)
        X_test_poly = poly.transform(X_test)

        # Create a linear regression model
        model = LinearRegression()

        # Train the model on the polynomial features training data
        model.fit(X_train_poly, y_train)

        # Use the trained model to predict the target values in the
test set
        y_pred = model.predict(X_test_poly)

        # Calculate the Mean Squared Error of the model on the test
set
        mse = mean_squared_error(y_test, y_pred)

        # Store the Mean Squared Error in the list
        mse_list.append(mse)

    # Plot the Mean Squared Error as a function of the degree of the
polynomial for the current random state
    plt.plot(degrees, mse_list, marker='o', label=f'Random State
{random_state}')

plt.title('Mean Squared Error vs. Degree of Polynomial for Different
Random States')
```

```
plt.xlabel('Degree of Polynomial')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.grid(True)
plt.show()
```




```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target

# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)

model = DecisionTreeClassifier()

# Train the model on the training data
model.fit(X_train, y_train)

# Use the trained model to predict the target values in the test set
```

```
y_pred_test = model.predict(X_test)

# Calculate the accuracy of the model on the training and test sets
accuracy_test = accuracy_score(y_test, y_pred_test)

print('Test Accuracy:', accuracy_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/
_openml.py:968: FutureWarning: The default value of `parser` will
change from `'liac-arff'` to `'auto'` in 1.4. You can set
`parser='auto'` to silence this warning. Therefore, an `ImportError`
will be raised from 1.4 if the dataset is dense and pandas is not
installed. Note that the pandas parser may return different data
types. See the Notes Section in fetch_openml's API doc for details.
  warn(
```

```
Test Accuracy: 0.8729285714285714
```

Leave One Out Cross Validation (LOOCV)

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import LeaveOneOut, cross_val_score

# Load the Boston Housing dataset
df =
pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master
/BostonHousing.csv')
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]

# Create a linear regression model
model = LinearRegression()

# Create a LeaveOneOut cross-validator
loo = LeaveOneOut()

# Use cross_val_score for the dataset with the model and LOOCV
# This will return the scores for each iteration of LOOCV
scores = cross_val_score(model, X, y, cv=loo,
scoring='neg_mean_squared_error')

mse_scores = -scores  # Invert the sign of the scores

# Print the mean MSE over all LOOCV iterations
print("Mean MSE:", mse_scores.mean())
```

```
Mean MSE: 23.72574551947613
```

```
scores
```

```
array([-3.72979719e+01, -1.19996083e+01, -1.74878280e+01, -
2.37110133e+01,
       -7.04199800e+01, -1.22185247e+01, -1.06607826e-02, -
6.08323283e+01,
       -2.76061960e+01, -4.35492529e-04, -1.71891129e+01, -
7.57242810e+00,
       -6.52257805e-01, -7.37772576e-01, -1.20890968e+00, -
3.73396196e-01,
       -6.92394876e+00, -3.54752744e-01, -1.69701375e+01, -
4.35547121e-02,
       -1.20164044e+00, -3.82170339e+00, -4.13536519e-01, -
4.99399777e-01,
       -6.33146773e-03, -2.72147161e-01, -1.33111630e+00, -
8.65299886e-03,
       -1.36231825e+00, -1.57472599e-02, -1.61151928e+00, -
1.31345333e+01,
       -2.06757013e+01, -1.44397569e+00, -4.47571447e-02, -
2.46360177e+01,
       -5.59248522e+00, -4.55268647e+00, -3.28660467e+00, -
3.31240848e-01,
       -5.01414236e-01, -2.13097952e+00, -9.67926125e-03, -
8.52754852e-03,
       -3.10007107e+00, -8.03207324e+00, -1.84421470e-01, -
2.15724847e+00,
       -3.10857366e+01, -4.97324456e+00, -2.56665919e+00, -
1.24320814e+01,
       -7.23742751e+00, -4.33023528e-01, -1.38938898e+01, -
1.94058162e+01,
       -2.61868556e-02, -2.47283378e+00, -2.41117390e+00, -
2.26666815e+00,
       -7.12690074e-01, -6.70735171e+00, -3.33202049e+00, -
6.23714796e+00,
       -1.00186283e+02, -4.95784462e+01, -3.98629341e+01, -
8.08527151e-01,
       -4.76265413e-04, -1.35769793e-02, -1.05610644e+00, -
1.89273016e-03,
       -3.24426332e+00, -4.35204956e-01, -2.10591702e+00, -
6.77973936e+00,
       -8.93921534e+00, -6.72924955e+00, -3.95389157e-03, -
4.68203230e+00,
       -1.67027556e-01, -9.77016174e+00, -1.54940300e+00, -
4.72129700e+00,
       -7.94795914e-01, -1.43988785e+00, -1.12186591e-01, -
1.39147249e+01,
       -5.17230815e+01, -4.65764932e+00, -2.09274511e+01, -
3.00759957e+01,
       -3.80683199e+01, -1.76112021e+01, -4.34841219e+01, -
5.18896413e-02,
```

```
        -1.14056526e+01, -8.99315837e+00, -7.96409775e+01, -
9.32347582e-01,
        -8.79658073e+00, -8.47154999e-01, -1.56179398e+00, -
1.05458947e+00,
        -1.84301340e+00, -9.60618351e-01, -5.56763550e+00, -
1.27234319e-01,
        -8.42278110e+00, -1.43119138e-01, -1.14150296e+00, -
1.42119152e+01,
        -4.01230954e+00, -4.18567992e+00, -4.57030533e+01, -
4.66318080e+00,
        -4.84127460e+00, -2.06842310e+01, -4.23215802e-03, -
2.28282551e+00,
        -7.94133663e-03, -5.33604801e+00, -3.73181696e-03, -
1.00286599e+00,
        -3.51652360e+00, -1.32546920e+00, -1.35592260e+00, -
1.08989264e+00,
        -9.27611127e-01, -6.22952422e-02, -7.35595030e-01, -
3.79653617e-02,
        -9.11057740e+00, -7.29742916e+00, -5.77148513e+00, -
7.36695352e-01,
        -2.41840651e+00, -5.38560581e+00, -2.78241452e-01, -
1.91694952e+00,
        -1.95824678e-01, -1.18717950e+02, -1.64849076e+00, -
1.31249807e+01,
        -1.05190809e+01, -3.49233265e+00, -5.40314474e-02, -
4.13557806e+01,
        -7.24343808e+01, -3.86523010e-01, -4.77423451e-01, -
1.86129863e+00,
        -2.72514403e+01, -4.88715741e+00, -3.26775345e+01, -
2.42421318e+01,
        -3.11394847e-01, -6.89729602e+01, -2.40386649e+01, -
5.71237974e+00,
        -3.62551814e+01, -1.88331029e+02, -1.00351073e+02, -
7.59080669e+01,
        -4.60196620e+00, -1.53145660e-01, -1.77642847e+02, -
5.38550273e-01,
        -7.14135919e+00, -1.99202499e+01, -2.80076599e+01, -
2.84722293e+01,
        -1.59718902e-02, -3.08941170e+01, -1.59342444e+01, -
1.81470989e+00,
        -6.00460836e+00, -2.11722843e+01, -2.41778128e+00, -
1.88926450e+01,
        -2.70094290e+01, -7.38078464e+01, -1.68376895e+01, -
2.38895128e+00,
        -1.42614594e+01, -2.40732316e+01, -2.08289504e+02, -
2.13640770e+00,
        -7.14383988e+00, -1.54449979e-01, -4.03849335e+01, -
4.58824932e-02,
        -1.25685521e+01, -1.06127968e+00, -6.23019623e+00, -
8.89608857e+01,
```

```
        -8.51668862e+00, -6.00617110e+00, -1.17088066e-02, -
2.49622137e+01,
        -5.50881699e+00, -2.87543981e+01, -2.91960200e+01, -
4.61521337e+01,
        -5.12955269e+01, -8.47509427e-03, -5.21510650e-01, -
2.20152014e+01,
        -8.82557414e-01, -9.96934272e+00, -5.20630999e-01, -
5.55587207e+00,
        -1.24445311e-01, -8.51709357e+00, -1.84347981e+02, -
2.44018562e-01,
        -1.18009190e+01, -1.22442331e-01, -1.26490148e+01, -
4.82145420e+01,
        -4.53630851e+01, -4.76207853e+00, -2.31144684e+01, -
1.28229561e-01,
        -4.34167374e+01, -1.11444259e+02, -2.03410475e-04, -
6.54535546e-01,
        -1.34898765e+02, -7.38220020e-02, -3.47160511e-02, -
2.58868698e+00,
        -1.40373665e+01, -1.29964545e+02, -7.88886228e+00, -
1.63453599e+00,
        -2.68274503e+01, -1.52510081e+00, -2.28155823e+01, -
2.66834161e+01,
        -2.86503056e+01, -1.36358924e+01, -3.77632410e+00, -
1.41293028e+01,
        -1.71567435e+00, -2.74347410e+01, -1.89162160e+01, -
4.32374416e-01,
        -1.06259433e+01, -4.65598898e+00, -3.86785481e-02, -
6.07092407e-02,
        -2.33375528e+01, -1.85751392e+02, -4.55265907e+00, -
6.70212898e-01,
        -4.50434824e+01, -4.92960202e+01, -2.49837830e-01, -
2.56253945e+01,
        -1.08523998e+00, -3.74282037e+01, -6.61849817e+01, -
1.26503523e+01,
        -4.71473248e-01, -3.23663303e+01, -2.95647628e-01, -
9.01555225e+01,
        -1.83754238e+01, -2.71993566e+01, -1.47986072e+00, -
4.16432443e+00,
        -1.72123191e+01, -8.35280270e-02, -1.49704590e+01, -
3.31492872e+00,
        -6.26842867e+00, -3.28104114e+00, -1.61504270e+00, -
4.56649462e-02,
        -4.54050434e+01, -1.17602153e+00, -3.47545840e+01, -
3.23102832e+01,
        -3.85649622e-01, -2.95648798e+01, -3.22171217e-06, -
1.51769690e+01,
        -2.48907876e+01, -4.59385309e+00, -2.65527478e+01, -
9.14411477e+00,
        -1.68929868e+01, -3.92826934e+00, -7.86297102e+00, -
2.15863841e-02,
```

```
        -7.31845169e-02, -6.19975407e-01, -4.57161739e+01, -
8.84860490e+00,
        -3.74061691e+01, -4.90879273e+01, -6.33118434e+00, -
9.31305942e-02,
        -8.70789441e+00, -5.86512735e+00, -4.92575516e+00, -
2.13451915e+01,
        -3.49044391e+01, -1.09724640e+01, -6.29575307e+00, -
2.33948307e+01,
        -1.53267427e+01, -1.58254057e+01, -2.87031714e+00, -
1.91225639e+01,
        -3.45063114e-02, -2.05239818e+00, -1.43069022e+00, -
1.06937712e-01,
        -1.19837605e+00, -3.17683701e+00, -6.19879545e+00, -
9.22738099e-01,
        -1.41603295e-02, -4.80410232e-03, -4.76517559e-01, -
8.33683385e+00,
        -3.78108829e+00, -2.95446912e+00, -3.44163149e+00, -
8.52880419e+00,
        -1.58704465e+01, -3.69818560e-03, -7.65403694e-01, -
2.41533579e-01,
        -4.51295193e-01, -6.36556722e-01, -2.54273062e+00, -
5.21979388e+00,
        -7.66807993e+00, -5.80646977e+00, -3.40868376e+01, -
1.48944839e+01,
        -7.24390426e+00, -9.68732655e-01, -6.21378927e+00, -
4.99569154e+00,
        -9.68952435e+00, -2.10983445e+01, -6.32452573e+00, -
1.38528290e+01,
        -3.21831154e+00, -2.58215726e+01, -1.68538358e+01, -
1.80108267e+01,
        -3.68301598e+00, -1.14144604e+00, -2.74418592e-01, -
1.20887385e+01,
        -5.77277531e+00, -9.75228358e-01, -7.09182880e+00, -
1.29149310e+01,
        -2.85283734e+02, -2.14991110e+02, -4.27516466e+01, -
1.77993768e+02,
        -7.87394677e+02, -3.35165668e+02, -2.64080081e+02, -
6.54393639e+02,
        -6.33948965e+02, -6.33920222e+01, -1.86951633e+02, -
1.11627843e+02,
        -1.53278613e+01, -5.00286144e+01, -7.83456278e+00, -
4.54074784e+01,
        -3.27190023e+01, -5.93660627e+01, -4.66605614e+00, -
5.99865366e-01,
        -3.21396004e+01, -7.77477043e-01, -2.02465585e+01, -
3.37825818e+00,
        -1.47046382e+01, -7.55625886e+00, -4.58832206e+00, -
3.57928740e+01,
        -3.79010646e-02, -4.22828265e+01, -2.81819559e+01, -
5.32818884e+01,
```

```
        -4.76129636e+01, -6.30466832e+01, -2.66836661e+00, -
2.23280202e+01,
        -4.15271212e+01, -1.15799963e+02, -3.89670699e+01, -
2.28914837e+01,
        -1.39665335e+00, -1.44905158e+01, -1.58390416e+01, -
6.52503227e+01,
        -1.27588287e+01, -6.10843325e+01, -6.49569980e-02, -
6.17014490e-02,
        -2.92540605e+02, -2.15683092e+01, -1.48380511e+02, -
6.14617653e+00,
        -3.74288794e+01, -1.28971245e+01, -1.07238631e+01, -
4.15539287e+01,
        -8.59945628e+00, -1.57554383e+01, -5.34542583e+00, -
5.49460320e-02,
        -9.39636101e+00, -2.81528502e+00, -4.10145773e+01, -
1.14007595e+01,
        -1.11368252e+01, -1.32928346e+01, -1.41873330e+01, -
2.25356497e+01,
        -3.12691199e+01, -7.83217966e+00, -1.89861962e+01, -
1.56693948e-03,
        -2.58151965e+01, -1.54075139e-02, -1.36367226e+01, -
7.30268710e-02,
        -5.06274047e+00, -3.94932251e-02, -1.19782965e-01, -
7.30079571e+00,
        -5.22904200e-01, -3.23155428e-02, -7.95328810e+00, -
3.14777427e+01,
        -1.20064705e+01, -1.82908719e+01, -1.05763075e+01, -
1.83119917e+01,
        -6.34396660e+00, -2.31843764e+01, -1.55789037e-01, -
3.15951600e+00,
        -1.75398458e-04, -4.20294074e-01, -5.35596178e+00, -
2.28212551e+00,
        -7.23533137e+00, -6.31714029e+00, -7.75886371e-02, -
5.17140962e+00,
        -1.20467897e+00, -4.23009940e+00, -2.30805411e+01, -
4.80334066e+00,
        -4.61073483e+00, -2.37217425e+00, -7.39262581e-02, -
1.20045892e+01,
        -5.73795938e-01, -1.85105817e+01, -6.97065728e+00, -
8.23908088e+00,
        -1.52296222e+01, -2.16058211e-01, -2.17655358e+01, -
2.20269614e-01,
        -2.29686896e-01, -1.21482099e+01, -1.34717369e+01, -
5.44009174e-01,
        -1.37613113e+00, -1.09158712e+00, -3.19520391e-01, -
5.47953117e-01,
        -1.30481386e+01, -1.75327795e+00, -2.33540468e+01, -
2.96720368e-02,
        -2.04045788e+01, -1.41901580e+00, -1.57280877e+01, -
4.11132371e+01,
```

```
       -3.32780322e+01, -6.65070419e-01, -9.81706658e-03, -
9.32985801e-01,
       -1.36926967e+01, -1.33604541e+00, -3.27781656e+00, -
1.46379888e+01,
       -1.78577924e+01, -1.13697230e+02])
```

```
X.shape
```

```
(506, 13)
```

K Fold Cross Validation

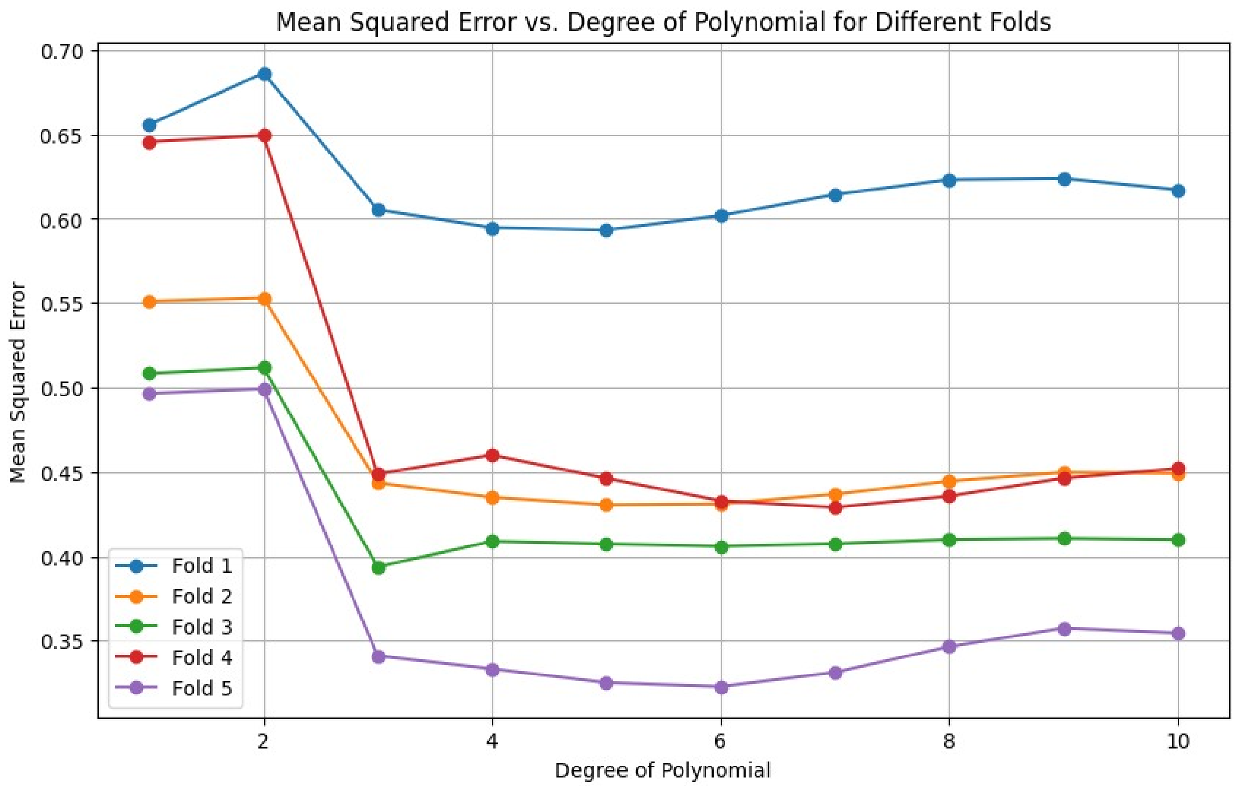
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
import pandas as pd

# Load the Boston Housing dataset
df =
pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master
/BostonHousing.csv')
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]

# Initialize a Linear Regression model
model = LinearRegression()

# Initialize the KFold parameters
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Use cross_val_score on the model and dataset
scores = cross_val_score(model, X, y, cv=kfold, scoring='r2')

print("R2 scores for each fold:", scores)
print("Mean R2 score across all folds:", scores.mean())
```

```
R2 scores for each fold: [0.75981355 0.60908125 0.76975858 0.71639463
0.61663293 0.79789535
 0.76682601 0.79453027 0.74066667 0.59908146]
Mean R2 score across all folds: 0.7170680714871446
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Load the Auto MPG dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-
```

```
mpg/auto-mpg.data"
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower',
'Weight', 'Acceleration', 'Model Year', 'Origin']
auto_mpg = pd.read_csv(url, names=column_names, delim_whitespace=True,
na_values='?')

# Drop rows with missing values
auto_mpg = auto_mpg.dropna()

# Define features and target
features = auto_mpg[['Horsepower']]  # Only use 'Horsepower' as a
feature
target = auto_mpg['MPG']

# Convert to numpy arrays for easier manipulation
X = features.to_numpy()
y = target.to_numpy()

# List of polynomial degrees
degrees = list(range(1, 11))  # Extend degrees to 10

# Create a plot
plt.figure(figsize=(10, 6))

# Create a 10-fold cross validator
kf = KFold(n_splits=5, shuffle=True, random_state=1)

for i, (train_index, test_index) in enumerate(kf.split(X)):
    # Split the data
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # List to store Mean Squared Errors
    mse_list = []

    for degree in degrees:
        # Add polynomial features
        poly = PolynomialFeatures(degree=degree)
        X_train_poly = poly.fit_transform(X_train)
        X_test_poly = poly.transform(X_test)

        # Create a linear regression model
        model = LinearRegression()

        # Train the model on the polynomial features training data
        model.fit(X_train_poly, y_train)

        # Use the trained model to predict the target values in the
test set
        y_pred = model.predict(X_test_poly)
```

```
        # Calculate the Mean Squared Error of the model on the test
set
        mse = mean_squared_error(y_test, y_pred)

        # Store the Mean Squared Error in the list
        mse_list.append(mse)

    # Plot the Mean Squared Error as a function of the degree of the
polynomial for the current fold
    plt.plot(degrees, mse_list, marker='o', label=f'Fold {i + 1}')

plt.title('Mean Squared Error vs. Degree of Polynomial for Different
Folds')
plt.xlabel('Degree of Polynomial')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.grid(True)
plt.show()
```



Stratified K Fold

```
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
```

```python
# Load iris dataset
data = load_iris()
X, y = data.data, data.target

# Create a Logistic Regression model
model = LogisticRegression(max_iter=10000, random_state=42)

# Create StratifiedKFold object
skf = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

# Perform stratified cross validation
scores = cross_val_score(model, X, y, cv=skf, scoring='accuracy')

# Print the accuracy for each fold
print("Accuracies for each fold: ", scores)
print("Mean accuracy across all folds: ", scores.mean())
```

```
Accuracies for each fold:  [1.         0.96666667 0.93333333 1.
0.93333333]
Mean accuracy across all folds:  0.9666666666666668
```