MM20B007 Tut 5

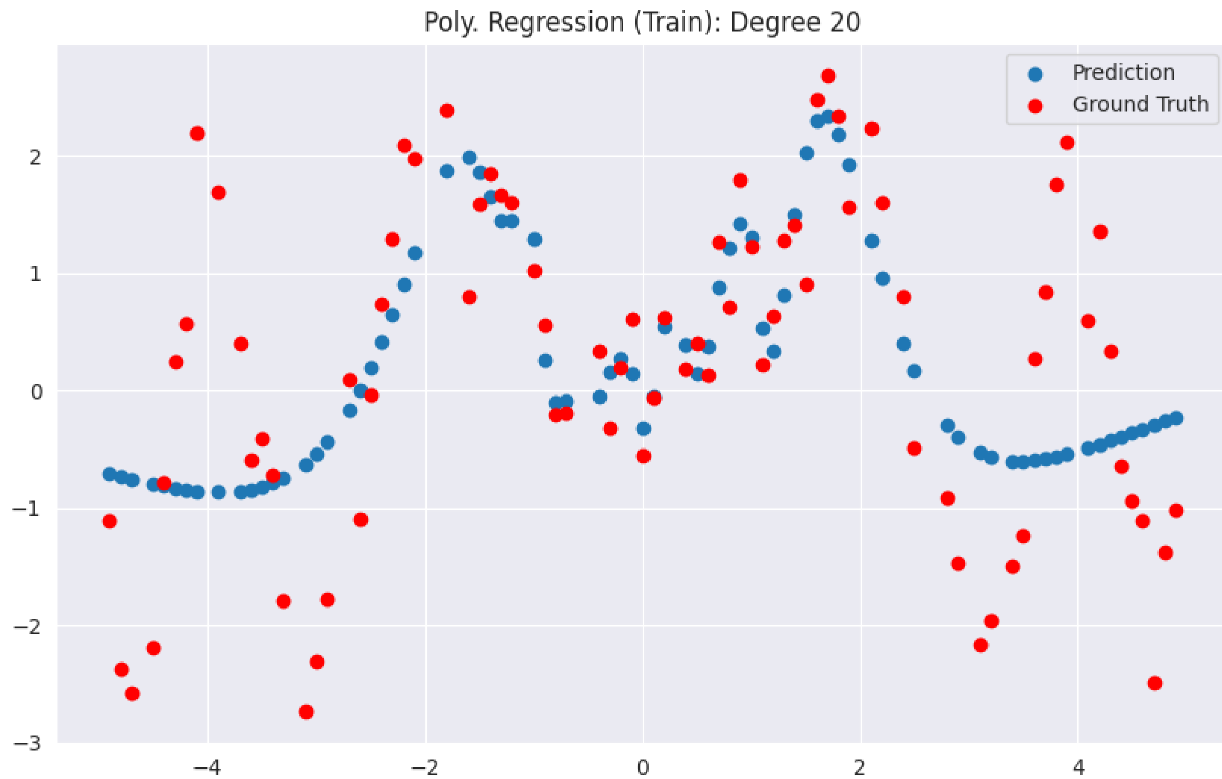
Problem 1

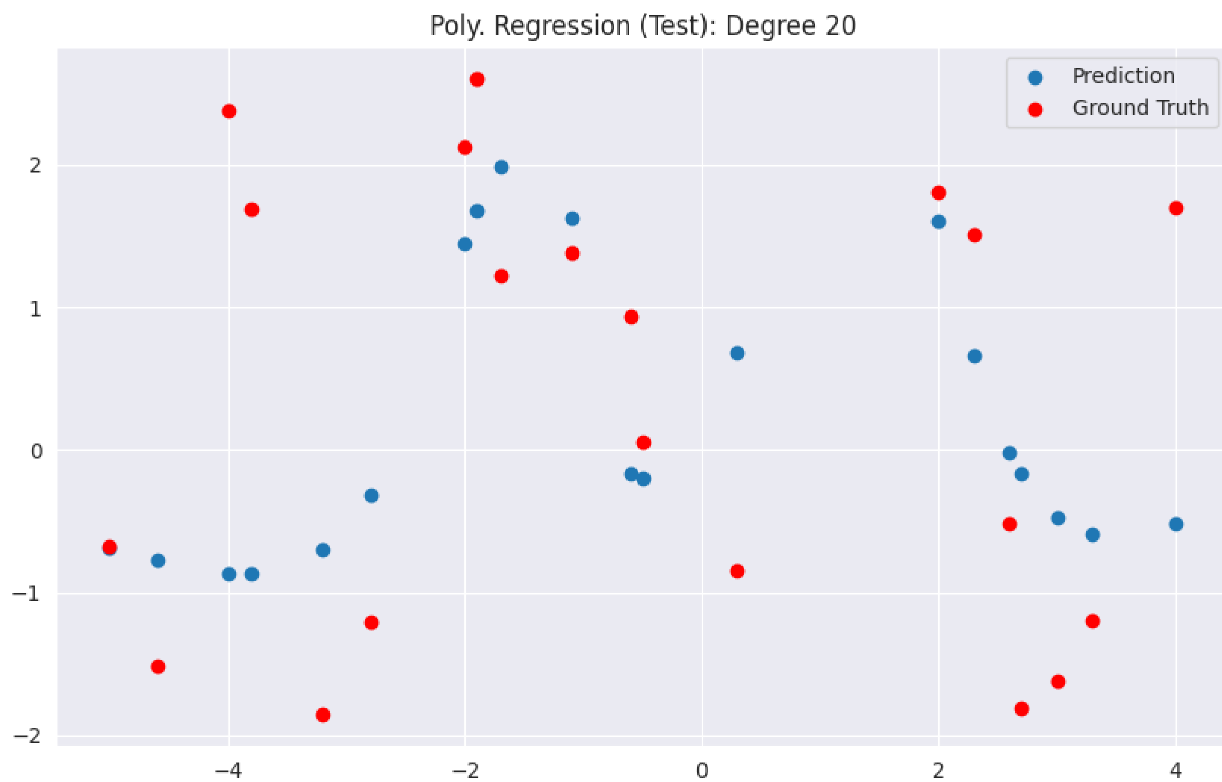
[Bias-Variance Tradeoff] In this problem, the goal is to use Dataset 2 described in the tutorial notebook and plot training and testing error curves against model complexity. Use polynomial regression, discussed in class to fit polynomial of degree k to the data. Search space for the degree of the polynomial can be taken to be k ∈ [1, 30]. Plot following 2 curves: Train/Test MSE vs Degree of Polynomial Regression Report optimal choice of k based on the testing loss.

Importing necessary packages

```
import math
import random
import sklearn
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

num= 100
random.seed = 42
np.random.seed = 42
sns.set_style("darkgrid")

dataset_path = '/content/drive/MyDrive/sem 7/ID5055/Tutorial
5/poly_reg2.csv'

# Function to load data and get train and test data
def load_data(path):
  data = pd.read_csv(path)
  arr = data.to_numpy().T

  return train_test_split(arr[0], arr[1], test_size = 0.2,
random_state = 42, shuffle = True)

# Funtion for poly regression
def poly_regression(path, k = 2, plot = True):
  x_train, x_test, y_train, y_test = load_data(path)
```

```python
  poly_train = PolynomialFeatures(degree = k, include_bias = False)
  poly_x_train = poly_train.fit_transform(x_train.reshape(-1, 1))
  poly_x_train = sklearn.preprocessing.normalize(poly_x_train)

  poly_test = PolynomialFeatures(degree = k, include_bias = False)
  poly_x_test = poly_test.fit_transform(x_test.reshape(-1, 1))
  poly_x_test = sklearn.preprocessing.normalize(poly_x_test)

  poly_model = LinearRegression()
  poly_model.fit(poly_x_train, y_train)
  y_pred_train = poly_model.predict(poly_x_train)
  y_pred_test = poly_model.predict(poly_x_test)

  mse_train = mean_squared_error(y_train, y_pred_train)
  mse_test = mean_squared_error(y_test, y_pred_test)

  # print('Degree', k)
  # print('Mean Squared Error (TRAIN):', mse_train)
  # print('Mean Squared Error (TEST):', mse_test)

  if plot:
    # Train data visualization
    plt.figure(figsize = (10, 6))
    plt.scatter(x_train, y_pred_train)
    plt.scatter(x_train, y_train, c = 'r')
    plt.legend(['Prediction', 'Ground Truth'])
    plt.title(r'Poly. Regression (Train): Degree {}'.format(k))
    plt.show()

    # Test data visualization
    plt.figure(figsize = (10, 6))
    plt.scatter(x_test, y_pred_test)
    plt.scatter(x_test, y_test, c = 'r')
    plt.legend(['Prediction', 'Ground Truth'])
    plt.title(r'Poly. Regression (Test): Degree {}'.format(k))
    plt.show()

  return mse_train, mse_test

# For dataset 2

a, b = poly_regression(dataset_path, k = 20, plot = True)
print('Degree', 20)
print('Mean Squared Error (TRAIN):', a)
print('Mean Squared Error (TEST):', b)
```


Poly. Regression (Train): Degree 20
Prediction
Ground Truth
2
1
0
−1
−2
−3
−4
−2
0
2
4


Poly. Regression (Test): Degree 20
Prediction
Ground Truth
2
1
0
−1
−2
−4
−2
0
2
4

```
Degree 20
Mean Squared Error (TRAIN): 1.0783998135488606
Mean Squared Error (TEST): 1.7840628969537604

# Relation between degree of polynomial regression and MSE

def k_vs_mse(k, degree_of_polyfit):
    mse_corresponding_to_k = {}

    for i in range(1, k):
        train_mse, test_mse = poly_regression(dataset_path, k=i,
plot=False)
        mse_corresponding_to_k[i] = [train_mse, test_mse]

    [print(f'for k: {k} goodness values are
{mse_corresponding_to_k[k]}') for k in mse_corresponding_to_k.keys()]

    train_mse_values = [mse[0] for mse in
mse_corresponding_to_k.values()]
    test_mse_values = [mse[1] for mse in
mse_corresponding_to_k.values()]
    degrees = list(mse_corresponding_to_k.keys())

    # Fit polynomial curves to the scatter points
    train_coefficients = np.polyfit(degrees, train_mse_values,
degree_of_polyfit)
    test_coefficients = np.polyfit(degrees, test_mse_values,
degree_of_polyfit)

    # Create polynomial functions
    train_poly = np.poly1d(train_coefficients)
    test_poly = np.poly1d(test_coefficients)

    x_values = np.linspace(min(degrees), max(degrees), 100)

    fig, ax = plt.subplots(1, 2, figsize=(12, 6))

    # Plot training error (MSE) vs Degree of Polynomial Regression (k)
with the fitted curve
    ax[0].scatter(degrees, train_mse_values, label='Training Data')
    ax[0].plot(x_values, train_poly(x_values), label='Fitted
Polynomial', color='red')
    ax[0].set_ylabel('MSE of train')
    ax[0].set_xlabel('Degree of Polynomial Regression')
    ax[0].set_title('Training Error (MSE) vs Degree of Polynomial
Regression (k)')
    ax[0].legend()

    # Plot test error (MSE) vs Degree of Polynomial Regression (k)
with the fitted curve
```

```
    ax[1].scatter(degrees, test_mse_values, label='Test Data')
    ax[1].plot(x_values, test_poly(x_values), label='Fitted 
Polynomial', color='red')
    ax[1].set_ylabel('MSE of test')
    ax[1].set_xlabel('Degree of Polynomial Regression')
    ax[1].set_title('Test Error (MSE) vs Degree of Polynomial 
Regression (k)')
    ax[1].legend()

    plt.tight_layout()
    plt.show()

k_vs_mse(31, 2)
```

```
for k: 1 goodness values are [1.922344044282697, 2.472440789929736]
for k: 2 goodness values are [1.8803850247470137, 2.528384419991515]
for k: 3 goodness values are [1.4620617871706039, 2.0479225453996497]
for k: 4 goodness values are [1.5416281443086395, 2.1460324189503117]
for k: 5 goodness values are [1.4564690063263845, 1.9959867579135373]
for k: 6 goodness values are [1.3270551293251713, 2.050579893096593]
for k: 7 goodness values are [1.4488152523761275, 1.9911302504294752]
for k: 8 goodness values are [1.2300870260618988, 2.1086299148178536]
for k: 9 goodness values are [1.2727103940988558, 2.0602288832805344]
for k: 10 goodness values are [1.2179282875955129, 2.0569774301257566]
for k: 11 goodness values are [1.2380848616381255, 1.8568492208285445]
for k: 12 goodness values are [1.2023398227894166, 2.0567793378852266]
for k: 13 goodness values are [1.1442877929155122, 2.0473147698417256]
for k: 14 goodness values are [1.1470533793992557, 2.153307808606411]
for k: 15 goodness values are [1.1308749336937194, 1.9933067318772575]
for k: 16 goodness values are [1.11598138099663, 1.8632171134452615]
for k: 17 goodness values are [1.1227812447074352, 1.8930783652379621]
for k: 18 goodness values are [1.0809907774378966, 1.7177410773096344]
for k: 19 goodness values are [1.0910379812114213, 1.736333537200751]
for k: 20 goodness values are [1.0783998135488606, 1.7840628969537604]
for k: 21 goodness values are [1.069520386386658, 1.961110413343603]
for k: 22 goodness values are [1.07254265361323, 1.8015447556556317]
for k: 23 goodness values are [1.014742946851216, 1.772323051486029]
for k: 24 goodness values are [1.0303277408733742, 1.9015641003735617]
for k: 25 goodness values are [1.01258502651729, 1.7349299296303637]
for k: 26 goodness values are [0.9877751739949623, 1.8014077799179087]
for k: 27 goodness values are [1.0089568667296025, 1.792258707208886]
for k: 28 goodness values are [0.7633244981137262, 7.169016071958339]
for k: 29 goodness values are [0.9128770444295684, 7.558650513480917]
for k: 30 goodness values are [0.6980473212217384, 7.98531878837407]
```

Observation

1. From the test loss it is clear that the most optimum degree of polynomial regression (k) value is 18 because for k: 18 the train-test goodness values are [1.0809907774378966, 1.7177410773096344], which the least we can find.

2. However, when we fitted the test losses in a polynomial curve the dip comes between k values 10 and 15, so we can consider them as optimal values.

3. As we increase the degree of polynomial regression we see that train mse reduces drastically but the test mse increases implying that the model overfitted.

Problem 2

[Akaike and Bayesian Information Criteria] In this problem, the goal is to use Dataset 2 described in the tutorial notebook and plot AIC, BIC and AICc curves against model complexity. Use polynomial regression, discussed in class to fit polynomial of degree k to the data. Calculate AIC, BIC and AICc (described in the notebook). Search space for the degree of the polynomial can be taken to be $k \in [1, 30]$. Plot following 3 curves: AIC/BIC/AICc vs Degree of Polynomial Regression. Report optimal choice of k for each information criterion as well as corresponding test MSE.

Importing necessary packages

```
import math
import random
import sklearn
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

num= 100
random.seed = 42
np.random.seed = 42
sns.set_style("darkgrid")

dataset_path = '/content/drive/MyDrive/sem 7/ID5055/Tutorial 
5/poly_reg2.csv'

# Function to load data and get train and test data
def load_data(path):
  data = pd.read_csv(path)
  arr = data.to_numpy().T

  return train_test_split(arr[0], arr[1], test_size = 0.2, 
random_state = 42, shuffle = True)

# Funtion for poly regression
def poly_regression(path, k = 2, print_values = True):
  x_train, x_test, y_train, y_test = load_data(path)

  poly_train = PolynomialFeatures(degree = k, include_bias = False)
```

```python
  poly_x_train = poly_train.fit_transform(x_train.reshape(-1, 1))
  poly_x_train = sklearn.preprocessing.normalize(poly_x_train)

  poly_test = PolynomialFeatures(degree = k, include_bias = False)
  poly_x_test = poly_test.fit_transform(x_test.reshape(-1, 1))
  poly_x_test = sklearn.preprocessing.normalize(poly_x_test)

  poly_model = LinearRegression()
  poly_model.fit(poly_x_train, y_train)
  y_pred_train = poly_model.predict(poly_x_train)
  y_pred_test = poly_model.predict(poly_x_test)

  mse_train = mean_squared_error(y_train, y_pred_train)
  mse_test = mean_squared_error(y_test, y_pred_test)

  # find AIC/BIC/AICc for the given model
  num = len(poly_model.coef_) + 1
  n = len(poly_x_train)
  AIC = n*math.log(mse_train) + 2*num
  BIC = n*math.log(mse_train) + math.log(n)*num
  AICc = n*math.log(mse_train) + 2*num + (2*num*(num + 1))/(n-num-1)

  if print_values:
    print('Degree:', k)
    print('AIC (TRAIN):', AIC)
    print('BIC (TRAIN):', BIC)
    print('AICc (TRAIN):', AICc)

  return AIC, BIC, AICc

# For dataset 2
aic, bic, aicc = poly_regression(dataset_path, k = 20, print_values =
False)
print('Degree:', 20)
print('AIC (TRAIN):', aic)
print('BIC (TRAIN):', bic)
print('AICc (TRAIN):', aicc)
```

```
Degree: 20
AIC (TRAIN): 48.038263062384345
BIC (TRAIN): 98.06082239053585
AICc (TRAIN): 63.969297545142965
```

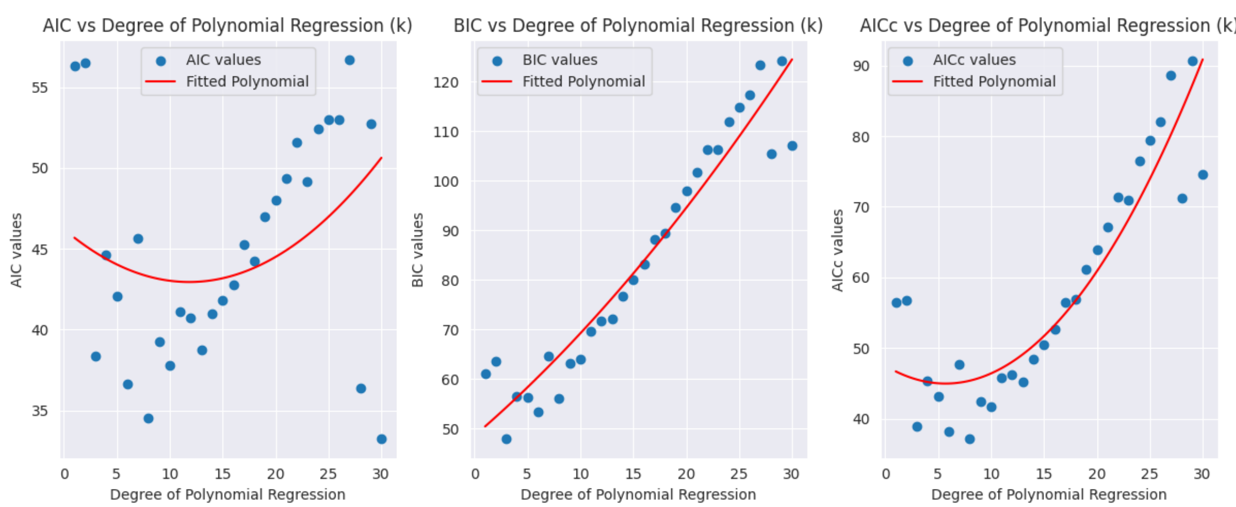
```python
# Relation between degree of polynomial regression and MSE

def k_vs_mse(k, degree_of_polyfit):
    creterion_value_vs_k = {}

    for i in range(1, k):
        aic, bic, aicc = poly_regression(dataset_path, k=i,
print_values = False)
```

```
        creterion_value_vs_k[i] = [aic, bic, aicc]

    AIC_values = [goodness_val[0] for goodness_val in
creterion_value_vs_k.values()]
    BIC_values = [goodness_val[1] for goodness_val in
creterion_value_vs_k.values()]
    AICc_values = [goodness_val[2] for goodness_val in
creterion_value_vs_k.values()]
    degrees = list(creterion_value_vs_k.keys())

    [print(f'for k: {k} goodness values are
{creterion_value_vs_k[k]}') for k in creterion_value_vs_k.keys()]

    # Fit polynomial curves to the scatter points
    aic_coefficients = np.polyfit(degrees, AIC_values,
degree_of_polyfit)
    bic_coefficients = np.polyfit(degrees, BIC_values,
degree_of_polyfit)
    aicc_coefficients = np.polyfit(degrees, AICc_values,
degree_of_polyfit)

    # Create polynomial functions
    AIC_poly = np.poly1d(aic_coefficients)
    BIC_poly = np.poly1d(bic_coefficients)
    AICc_poly = np.poly1d(aicc_coefficients)

    x_values = np.linspace(min(degrees), max(degrees), 100)

    fig, ax = plt.subplots(1, 3, figsize=(12, 5))

    # Plot AIC vs Degree of Polynomial Regression (k) with the fitted
curve
    ax[0].scatter(degrees, AIC_values, label='AIC values')
    ax[0].plot(x_values, AIC_poly(x_values), label='Fitted
Polynomial', color='red')
    ax[0].set_ylabel('AIC values')
    ax[0].set_xlabel('Degree of Polynomial Regression')
    ax[0].set_title('AIC vs Degree of Polynomial Regression (k)')
    ax[0].legend()

    # Plot BIC vs Degree of Polynomial Regression (k) with the fitted
curve
    ax[1].scatter(degrees, BIC_values, label='BIC values')
    ax[1].plot(x_values, BIC_poly(x_values), label='Fitted
Polynomial', color='red')
    ax[1].set_ylabel('BIC values')
    ax[1].set_xlabel('Degree of Polynomial Regression')
    ax[1].set_title('BIC vs Degree of Polynomial Regression (k)')
    ax[1].legend()
```

```
    # Plot AICc vs Degree of Polynomial Regression (k) with the fitted
curve
    ax[2].scatter(degrees, AICc_values, label='AICc values')
    ax[2].plot(x_values, AICc_poly(x_values), label='Fitted
Polynomial', color='red')
    ax[2].set_ylabel('AICc values')
    ax[2].set_xlabel('Degree of Polynomial Regression')
    ax[2].set_title('AICc vs Degree of Polynomial Regression (k)')
    ax[2].legend()

    plt.tight_layout()
    plt.show()

k_vs_mse(31, 2)
```

```
for k: 1 goodness values are [56.28362382382081, 61.04767709316857,
56.43946797966497]
for k: 2 goodness values are [56.51812450163737, 63.66420440565901,
56.83391397532158]
for k: 3 goodness values are [38.3878098016933, 47.915916340388826,
38.92114313502663]
for k: 4 goodness values are [44.62712756068876, 56.537260734058165,
45.43793837149958]
for k: 5 goodness values are [42.08120141077858, 56.37336121882187,
43.231886342285435]
for k: 6 goodness values are [36.63698390532576, 53.311170348042936,
38.19253946088132]
for k: 7 goodness values are [45.659692409938415, 64.71590548732947,
47.68786142402292]
for k: 8 goodness values are [34.56679358218486, 56.005033294249785,
37.13822215361343]
for k: 9 goodness values are [39.29190359595836, 63.112169942697165,
42.48030939305981]
for k: 10 goodness values are [37.77210322999954, 63.974396211412234,
41.65445617117601]
for k: 11 goodness values are [41.0852575422678, 69.66957715835437,
45.74197396017824]
for k: 12 goodness values are [40.74156084991042, 71.70790710067088,
46.25671236506194]
for k: 13 goodness values are [38.78259428429718, 72.13096716973152,
45.24413274583564]
for k: 14 goodness values are [40.97571002699764, 76.70610954710587,
48.47571002699764]
for k: 15 goodness values are [41.83932885848351, 79.95175501326561,
50.474249493404145]
for k: 16 goodness values are [42.7787344100135, 83.27318719946948,
52.649702151948986]
for k: 17 goodness values are [45.26470890633885, 88.14118833046871,
56.47782366043721]
for k: 18 goodness values are [44.230240568869505, 89.48874662767325,
```

```
56.89690723553617]
for k: 19 goodness values are [46.970361556346944, 94.61089424982457,
61.20764969194016]
for k: 20 goodness values are [48.038263062384345, 98.06082239053585,
63.969297545142965]
for k: 21 goodness values are [49.376824877100525, 101.78141083992591,
67.1312108420128]
for k: 22 goodness values are [51.60257130383719, 106.38918390133645,
71.3168570181229]
for k: 23 goodness values are [49.17082608685432, 106.33946531902747,
70.98900790503613]
for k: 24 goodness values are [52.39015573207503, 111.94082159892206,
76.4642298061491]
for k: 25 goodness values are [53.0005194623922, 114.9332119639131,
79.49108550012805]
for k: 26 goodness values are [53.01598689484152, 117.33070603103631,
82.0929099717646]
for k: 27 goodness values are [56.71335935393057, 123.41010512479924,
88.55649660883253]
for k: 28 goodness values are [36.39423634460403, 105.47300875014659,
71.19423634460404]
for k: 29 goodness values are [52.70767364048206, 124.1684726806985,
90.66685731395145]
for k: 30 goodness values are [33.24252935430757, 107.0853550291979,
74.5758626876409]
```



Observations

For degree of polynomial regression = 8 we have the lowest [AIC, BIC, AICc] combinaton, which is [34.56679358218486, 56.005033294249785, 37.13822215361343].

Hence the optimal model has degree of polynomial regression = 8.

If we see individually

1. **AIC**: the optimal model has degree of polynomial regression is 30 with score = 33.24252935430757, and MSE value for test data for k = 30 is 7.985318788837407.
2. **BIC**: the optimal model has degree of polynomial regression is 3 with score = 47.915916340388826 and MSE value for test data for k = 3 is 2.0479225453996497.
3. **AICc**: the optimal model has degree of polynomial regression is 8 with score = 37.13822215361343 and MSE value for test data for k = 8 is 2.1086299148178536.

Problem 3

[K-Fold Cross Validation] In this problem, the goal is to use diabetes dataset from sklearn library and plot k-fold cross- validation scores against model complexity. Use polynomial regression, discussed in class to fit polynomial of degree k to the data. Search space for the degree of the polynomial can be taken to be k ∈ [1, 10]. Plot following curve: Cross Validation Score vs Degree of Polynomial Regression (Note: The plots may blow up for some model complexities. The goal is to infer this.) Report optimal choice of k based on cross val score() function in the sklearn library.

Importing necessary packages

```
import math
import random
import sklearn
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
from sklearn.datasets import load_diabetes
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

num= 100
random.seed = 42
np.random.seed = 42
sns.set_style("darkgrid")

X, Y = load_diabetes(return_X_y = True, as_frame = True)
X = X[['age', 'sex', 'bmi', 'bp']]
print(X.head())
print(Y.head())
```

```
        age       sex       bmi        bp
0  0.038076  0.050680  0.061696  0.021872
1 -0.001882 -0.044642 -0.051474 -0.026328
2  0.085299  0.050680  0.044451 -0.005670
3 -0.089063 -0.044642 -0.011595 -0.036656
4  0.005383 -0.044642 -0.036385  0.021872
0    151.0
1     75.0
2    141.0
3    206.0
```

```
4    135.0
Name: target, dtype: float64
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size =
0.8, random_state = 0)

# For degree 1
n = 1
pipel = Pipeline([('poly', PolynomialFeatures(n)),
                  ('scaler', StandardScaler()),
                  ('linea', LinearRegression())])
pipel.fit(X_train, Y_train)

# For degree 10
n = 10
pipel_10 = Pipeline([('poly', PolynomialFeatures(n)),
                  ('scaler', StandardScaler()),
                  ('linea', LinearRegression())])
pipel_10.fit(X_train, Y_train)
```

```
Pipeline(steps=[('poly', PolynomialFeatures(degree=10)),
                ('scaler', StandardScaler()), ('linea',
LinearRegression())])
```

```
print('In train set:')
print('The model trained with polynomial features of degree 1',
r2_score (Y_train, pipel.predict(X_train)))
print('The model trained with polynomial features of degree 10',
r2_score (Y_train, pipel_10.predict(X_train)))

print('In test set')
print('The model trained with polynomial features of degree 1',
r2_score (Y_test, pipel.predict(X_test)))
print('The model trained with polynomial features of degree 10',
r2_score(Y_test, pipel_10.predict(X_test)))
```

```
In train set:
The model trained with polynomial features of degree 1
0.4253556823737591
The model trained with polynomial features of degree 10 1.0
In test set
The model trained with polynomial features of degree 1
0.2740192519276704
The model trained with polynomial features of degree 10 -
270198.49137645063
```

Using validation data (standard method)

```
# Train validation split
X_train_new, X_val, Y_train_new, Y_val = train_test_split(X_train,
```

```
Y_train, train_size = 0.8, random_state = 0)

val_score = []
for i in range(1, 11):
  n = i
  pipelN = Pipeline([('poly', PolynomialFeatures(n)),
                     ('scaler', StandardScaler()),
                     ('linea', LinearRegression())])
  pipelN.fit(X_train_new, Y_train_new)

  Y_val_pred = pipelN.predict(X_val)
  r2 = r2_score(Y_val, Y_val_pred)
  val_score.append(r2)

for i in range(len(val_score)):
  print(f'for n = {i + 1} the r2 score is {val_score[i]}')
```

```
for n = 1 the r2 score is 0.3906511557827157
for n = 2 the r2 score is 0.29494011303425083
for n = 3 the r2 score is 0.08842108965674966
for n = 4 the r2 score is 0.0716492733682087
for n = 5 the r2 score is -1.9720872056870138
for n = 6 the r2 score is -384.32360841460945
for n = 7 the r2 score is -76791.71266746201
for n = 8 the r2 score is -654104351.0393577
for n = 9 the r2 score is -3353450.73752144
for n = 10 the r2 score is -287213.194140729
```

The most optimal value of degree of polynomial regression from R squared score is 1.

Using K-fold Validation data

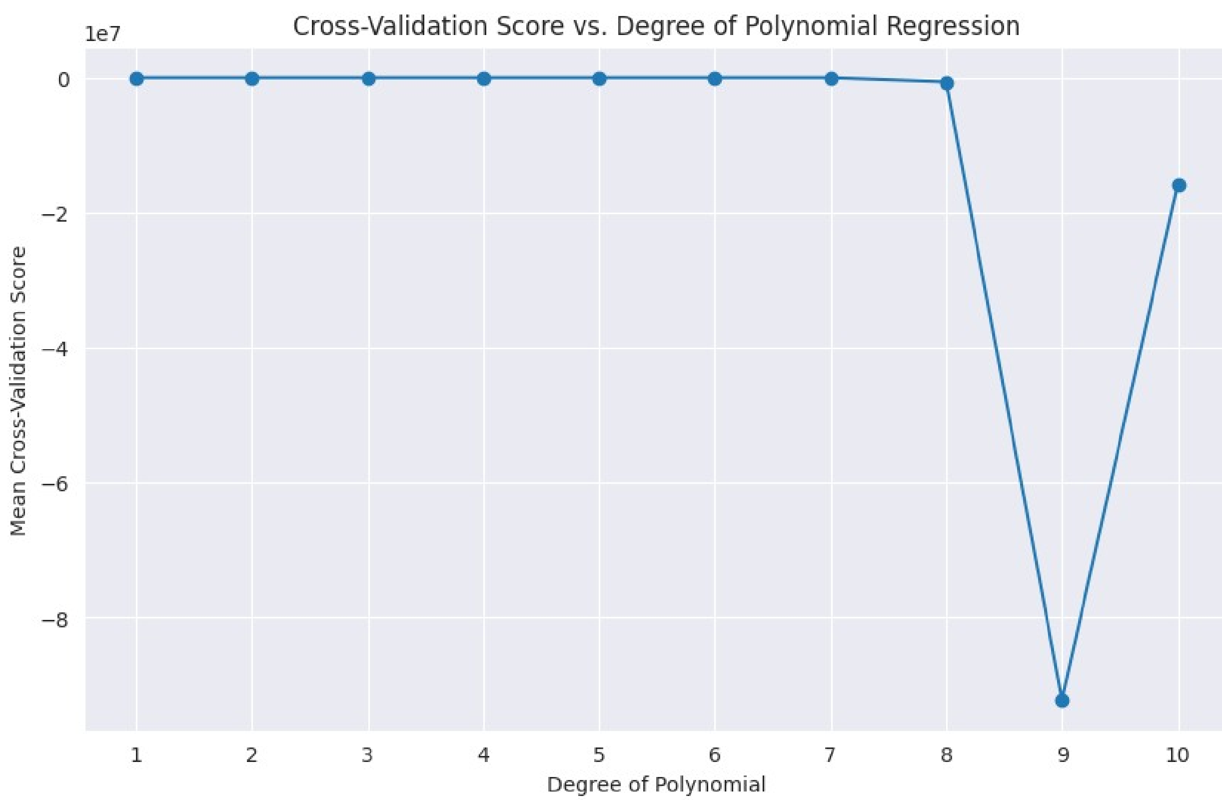
```
# K fold

val_score = []
for i in range(1, 11):
  n = i
  pipelN = Pipeline([('poly', PolynomialFeatures(n)),
                     ('scaler', StandardScaler()),
                     ('linea', LinearRegression())])

  c_val_score = cross_val_score(pipelN, X, Y, cv = 5)
  val_score.append(c_val_score)

for i in range(len(val_score)):
  print(f'for n = {i+1} the 5 fold cross validation scores are:
{val_score[i]}. \n')
```

```
for n = 1 the 5 fold cross validation scores are: [0.26674525
0.39888947 0.4044412  0.36174561 0.44085106].
```

```
for n = 2 the 5 fold cross validation scores are: [0.29185137
0.4148694  0.4115444  0.36153382 0.44716499].

for n = 3 the 5 fold cross validation scores are: [0.2536539
0.42396216 0.29426441 0.30035585 0.41031307].

for n = 4 the 5 fold cross validation scores are: [ 0.127432
0.33041057 -0.2376402   0.05233274 -0.52874947].

for n = 5 the 5 fold cross validation scores are: [-0.66584588
0.06143416 -0.68972109 -3.68048818 -1.74863488].

for n = 6 the 5 fold cross validation scores are: [  -4.65953005  -
11.36916347 -119.05432564 -396.81990538  -32.84833486].

for n = 7 the 5 fold cross validation scores are: [ -167.99117562  -
123.37035313  -585.17712374  -119.62323418
 -1809.26356993].

for n = 8 the 5 fold cross validation scores are: [  -11800.95458353
-70360.49535961 -1350682.27746277  -483454.11048919
 -1136808.29176504].

for n = 9 the 5 fold cross validation scores are: [-5.89376151e+06 -
1.50703663e+07 -1.08180865e+08 -5.83676863e+07
 -2.73054552e+08].

for n = 10 the 5 fold cross validation scores are: [  -761436.79129272
-959762.02534456 -66714408.2307476
  -3035130.07800055  -7336386.03230894].


# Since  val_score is a list of lists, where each inner list contains
cross-validation scores for a specific degree
# Convert it to a numpy array for easier manipulation
val_scores = np.array(val_score)

# Calculate the mean of the cross-validation scores for each degree
mean_scores = val_scores.mean(axis=1)

[print(f'for n = {i + 1} the mean score is {mean_scores[i]}') for i in
range(len(mean_scores))]

# Create an array of degrees for the x-axis
degrees = np.arange(1, 11)

# Plot the cross-validation scores vs. degree
plt.figure(figsize=(10, 6))
plt.plot(degrees, mean_scores, marker='o', linestyle='-')
plt.title('Cross-Validation Score vs. Degree of Polynomial
```

```
Regression')
plt.xlabel('Degree of Polynomial')
plt.ylabel('Mean Cross-Validation Score')
plt.grid(True)
plt.xticks(degrees)
plt.show()
```

```
for n = 1 the mean score is 0.3745345179227967
for n = 2 the mean score is 0.38539279680196864
for n = 3 the mean score is 0.3365098777114664
for n = 4 the mean score is -0.05124287343688407
for n = 5 the mean score is -1.34465117311117
for n = 6 the mean score is -112.95025188138845
for n = 7 the mean score is -561.0850913177359
for n = 8 the mean score is -610621.2259320281
for n = 9 the mean score is -92113446.39559889
for n = 10 the mean score is -15761424.631538872
```



Observation

Since we are using Linear Regression the default performance metrics used by cross_val_score is R^2, and we can see that it is highest for degree of polynomial regression = 2. Hence the most optimal value of degree of polynomial regression from k-fold cross validation is 2.