MM20B007 Tutorial 4

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
import statsmodels.api as sm
import scipy.stats as stats
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 8

# Seed for the reproucibility
np.random.seed(0)

# Generating Data
x = np.array([i*np.pi/180 for i in range(60, 300, 4)])
data = pd.DataFrame(x, columns = ['x'])

for i in range(2, 16):
  colname = f'x_{i}'
  data[colname] = round(data['x']**i, 1)
data['x'] = round(data['x'], 1)
print(data.head())
```

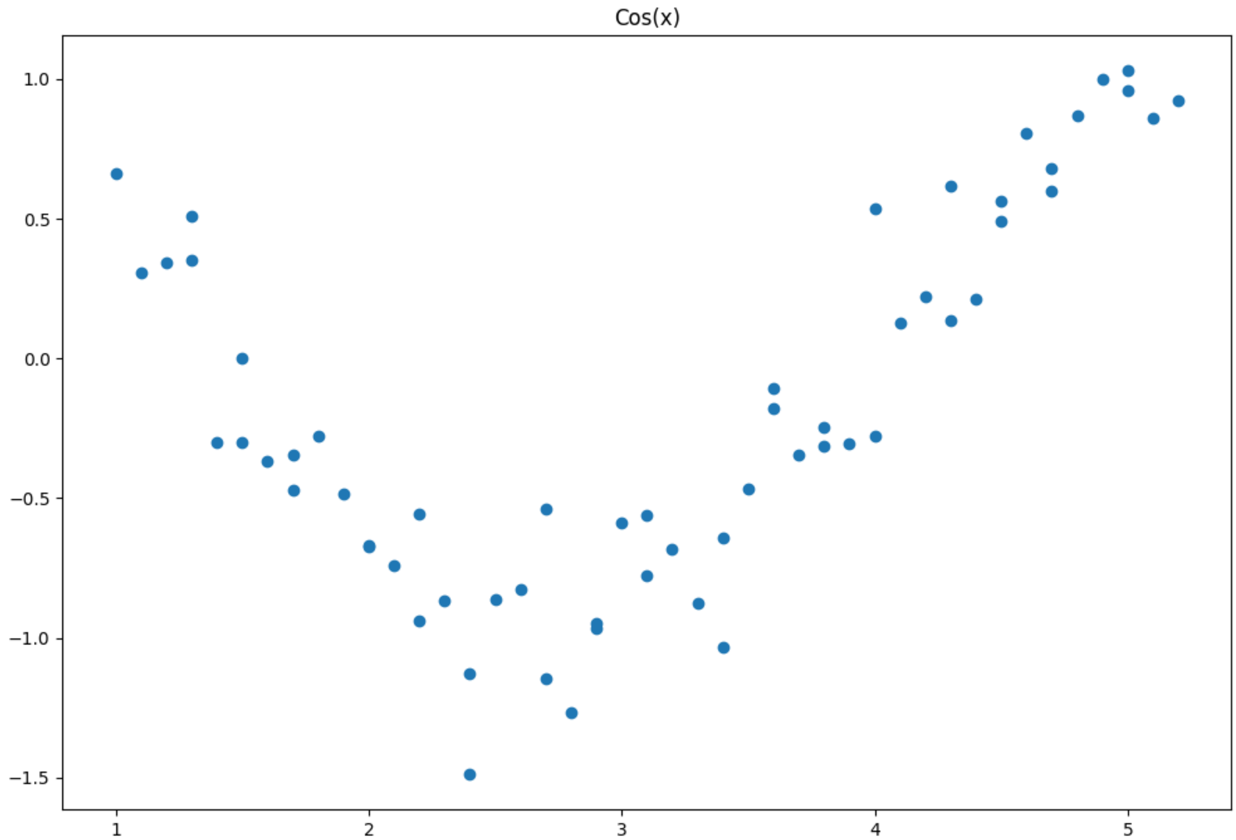
```
    x  x_2  x_3  x_4  x_5  x_6  x_7  x_8  x_9  x_10  x_11  x_12  x_13  
x_14  \
0   1  1.1  1.1  1.2  1.3  1.3  1.4  1.4  1.5   1.6   1.7   1.7   1.8   
1.9   
1 1.1  1.2  1.4  1.6  1.7  1.9  2.2  2.4  2.7     3   3.4   3.8   4.2   
4.7   
2 1.2  1.4  1.7    2  2.4  2.8  3.3  3.9  4.7   5.5   6.6   7.8   9.3    
11   
3 1.3  1.6    2  2.5  3.1  3.9  4.9  6.2  7.8   9.8    12    16    20    
24   
4 1.3  1.8  2.3  3.1  4.1  5.4  7.2  9.6   13    17    22    30    39    
52   

   x_15  
0     2  
1   5.3  
2    13  
3    31  
4    69  
```

```
y = np.cos(1.2*x) + np.random.normal(0, 0.2, len(x))
data['y'] = y
data.head()
```

```
     x  x_2  x_3  x_4  x_5  x_6  x_7  x_8  x_9  x_10  x_11  x_12  x_13
x_14  \
0   1  1.1  1.1  1.2  1.3  1.3  1.4  1.4  1.5   1.6   1.7   1.7   1.8
1.9
1 1.1  1.2  1.4  1.6  1.7  1.9  2.2  2.4  2.7     3   3.4   3.8   4.2
4.7
2 1.2  1.4  1.7    2  2.4  2.8  3.3  3.9  4.7   5.5   6.6   7.8   9.3
11
3 1.3  1.6    2  2.5  3.1  3.9  4.9  6.2  7.8   9.8    12    16    20
24
4 1.3  1.8  2.3  3.1  4.1  5.4  7.2  9.6   13    17    22    30    39
52

   x_15    y
0     2 0.66
1   5.3 0.31
2    13 0.34
3    31 0.51
4    69 0.35
```

```
plt.title('Cos(x)')
plt.scatter(data['x'], data['y'])
```

```
<matplotlib.collections.PathCollection at 0x7807d363c2e0>
```

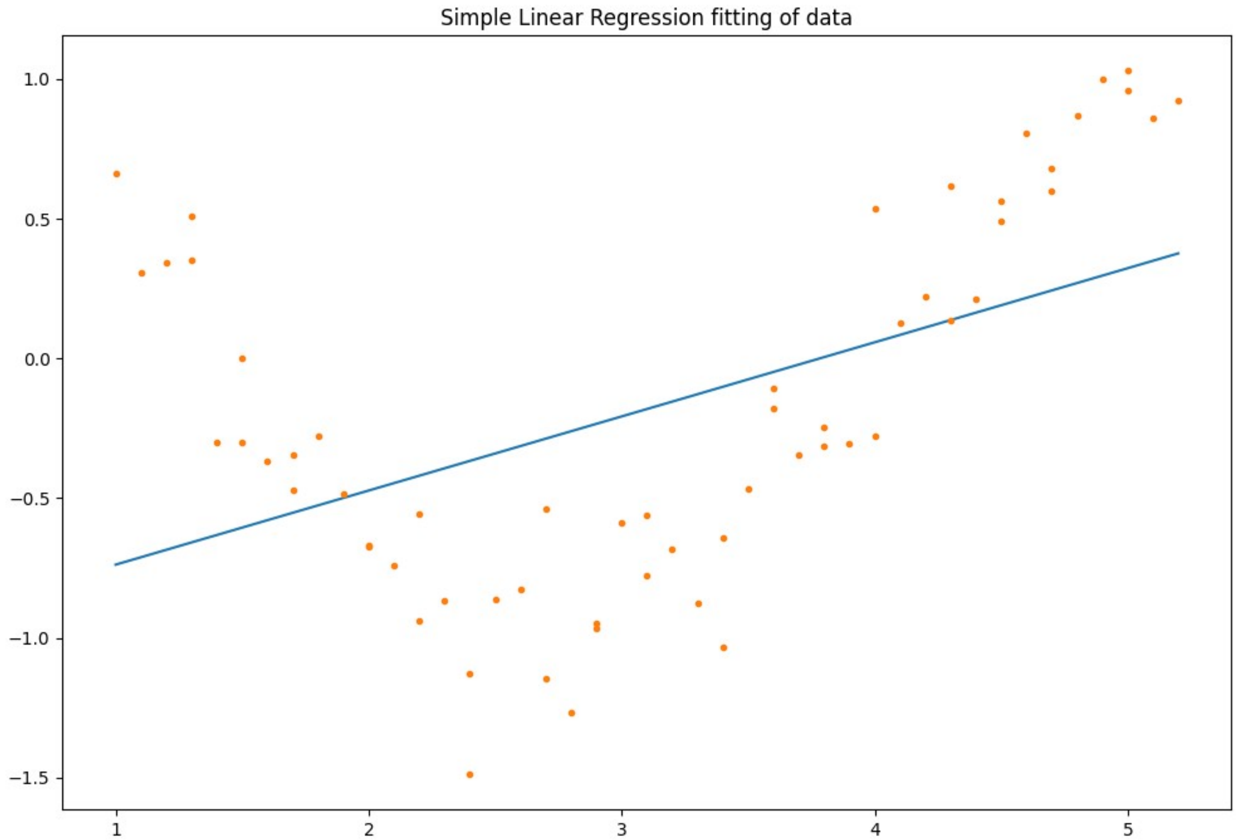


1. Linear Regression

Model: $Y_p = \beta X + \beta_0$

```
linreg = LinearRegression()
linreg.fit(data[['x']], data['y'])
y_prediction = linreg.predict(data[['x']])

plt.plot(data['x'], y_prediction)
plt.plot(data['x'], data['y'], '.')
plt.title('Simple Linear Regression fitting of data')
```

```
Text(0.5, 1.0, 'Simple Linear Regression fitting of data')
```

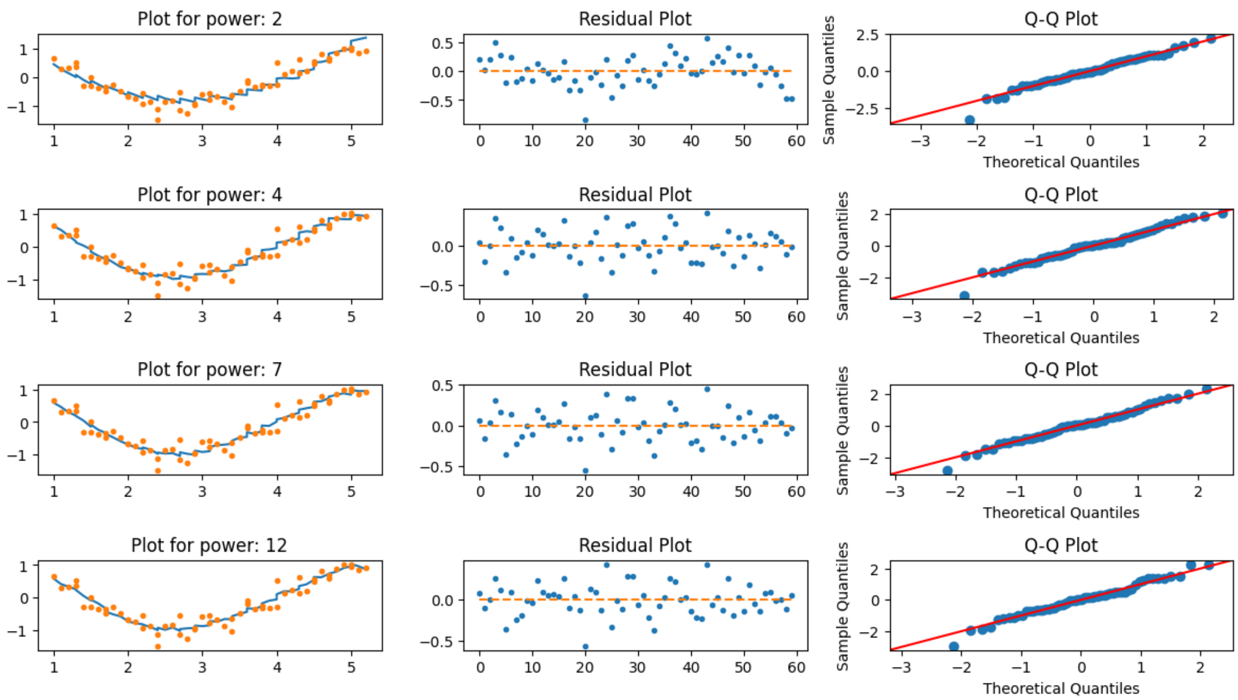


1.1 Linear Regression with non-linear features

Model: $Y_p = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^2 \ldots \beta_n X^n$

```
def linear_regression(data, power, models_to_plot):
  predictors = ['x']
  if power >= 2:
    predictors.extend([f'x_{i}' for i in range(2, power + 1)])

  # fitting the model
  linreg = LinearRegression()
  linreg.fit(data[predictors], data['y'])
  y_pred = linreg.predict(data[predictors])

  if power in models_to_plot:
    x, y, z = models_to_plot[power]
    plt.subplot(x, y, z)
    plt.tight_layout()
    plt.plot(data['x'], y_pred)
    plt.plot(data['x'], data['y'], '.')

    plt.title('Plot for power: %d'%power)

    plt.subplot(x, y, z+1)
```

```
    plt.tight_layout()
    xlen = np.arange(y_pred.shape[0])
    plt.plot(xlen, data['y'] - y_pred, ".")
    plt.plot(xlen, 0*xlen, "--")
    plt.title('Residual Plot')

    ax = plt.subplot(x, y, z+2)
    plt.tight_layout()
    sm.qqplot(data['y'] - y_pred, line = '45', fit=True, dist =
stats.norm, ax = ax)
    plt.title('Q-Q Plot')

  rss = sum ((y_pred-data['y'])**2)
  rss = [rss]
  rss.extend([linreg.intercept_])
  rss.extend(linreg.coef_)
  return rss

# Dataframe to store the results
col = ['rss', 'intercept'] + [f'coef_x_{i}' for i in range(1, 16)]
ind = [f'model_pow_{i}' for i in range(1, 16)]
coef_matrix_simple = pd.DataFrame(index = ind, columns = col)

models_to_plot = {2:(5,3,1), 4:(5,3,4), 7:(5,3,7), 12:(5,3,10), 25:
(5,3,13)}

for i in range(1, 16):
  coef_matrix_simple.iloc[i-1, 0:i+2] = linear_regression(data, power
= i, models_to_plot = models_to_plot)
```




```
pd.options.display.float_format = '{:,.2g}'.format
print(coef_matrix_simple)
```

```
              rss intercept coef_x_1 coef_x_2 coef_x_3 coef_x_4
coef_x_5  \
model_pow_1    20        -1     0.27      NaN      NaN      NaN
NaN
model_pow_2   3.9       2.2     -2.2     0.39      NaN      NaN
NaN
model_pow_3   3.2       3.6     -3.9        1   -0.068      NaN
NaN
model_pow_4   2.4       2.4     -1.5    -0.56     0.34   -0.036
NaN
model_pow_5   2.4       2.3     -1.5    -0.36      0.2  -0.0031   -
0.0026
model_pow_6   2.3       1.7     -0.9     0.12    -0.58     0.34    -
0.067
model_pow_7   2.3       1.6    -0.74    0.079    -0.78     0.54     -
0.14
model_pow_8   2.2       1.3    -0.65     0.38     -0.6   -0.082
0.28
model_pow_9   2.2       1.3    -0.64      0.4    -0.64   -0.093
0.32
model_pow_10  2.2       1.3    -0.62     0.46    -0.59    -0.11
0.18
model_pow_11  2.2       1.3    -0.54     0.49    -0.65    -0.18
0.18
model_pow_12  2.2      0.93    -0.43     0.77    -0.63    -0.15    -
```

```
0.032
model_pow_13 2.2       0.9    -0.36     0.83     -0.6    -0.26    -
0.18
model_pow_14 2.1       1.1     -0.4     0.67    -0.59    -0.32    -
0.17
model_pow_15 2.1       1.1     -0.4     0.67    -0.58    -0.32    -
0.19

             coef_x_6 coef_x_7 coef_x_8 coef_x_9 coef_x_10
coef_x_11  \
model_pow_1       NaN      NaN      NaN      NaN       NaN       NaN

model_pow_2       NaN      NaN      NaN      NaN       NaN       NaN

model_pow_3       NaN      NaN      NaN      NaN       NaN       NaN

model_pow_4       NaN      NaN      NaN      NaN       NaN       NaN

model_pow_5       NaN      NaN      NaN      NaN       NaN       NaN

model_pow_6    0.0044      NaN      NaN      NaN       NaN       NaN

model_pow_7     0.016 -0.00073      NaN      NaN       NaN       NaN

model_pow_8     -0.11    0.017 -0.00097      NaN       NaN       NaN

model_pow_9     -0.13    0.023  -0.0018  4.4e-05       NaN       NaN

model_pow_10    0.017   -0.042    0.013  -0.0017     8e-05       NaN

model_pow_11     0.15    -0.16    0.058   -0.011    0.0011  -4.4e-05

model_pow_12    0.044     0.19    -0.18    0.071    -0.014    0.0015

model_pow_13     0.19     0.31     -0.4      0.2    -0.052    0.0078

model_pow_14     0.38     0.29    -0.63     0.43     -0.16     0.036

model_pow_15     0.37     0.31    -0.62      0.4     -0.14     0.027


             coef_x_12 coef_x_13 coef_x_14 coef_x_15
model_pow_1        NaN       NaN       NaN       NaN
model_pow_2        NaN       NaN       NaN       NaN
model_pow_3        NaN       NaN       NaN       NaN
model_pow_4        NaN       NaN       NaN       NaN
model_pow_5        NaN       NaN       NaN       NaN
model_pow_6        NaN       NaN       NaN       NaN
model_pow_7        NaN       NaN       NaN       NaN
model_pow_8        NaN       NaN       NaN       NaN
model_pow_9        NaN       NaN       NaN       NaN
```

```
model_pow_10        NaN        NaN        NaN        NaN
model_pow_11        NaN        NaN        NaN        NaN
model_pow_12  -6.2e-05        NaN        NaN        NaN
model_pow_13  -0.00064    2.2e-05        NaN        NaN
model_pow_14    -0.005    0.00039  -1.3e-05        NaN
model_pow_15   -0.0029     0.0001      9e-06   -7.3e-07
```

Observations

1. As we move from left to right in the DataFrame (from lower to higher polynomial degrees), more coefficients become non-null. This is because higher-degree polynomial regressions can capture more complex relationships between variables but may also be more prone to overfitting.

2. Also the size of coefficents increases with the increase in model complexity. This is a problem because in the starting that feature is considered a good predictor and given lot of emphasis but when it becomes too large the algorithm began to model intricate relations to estimate the output and end up overfitting.

3. To avoid this a penalty should be introduced that will act as a trade off between magnitude of coefficents and efficeny of model.

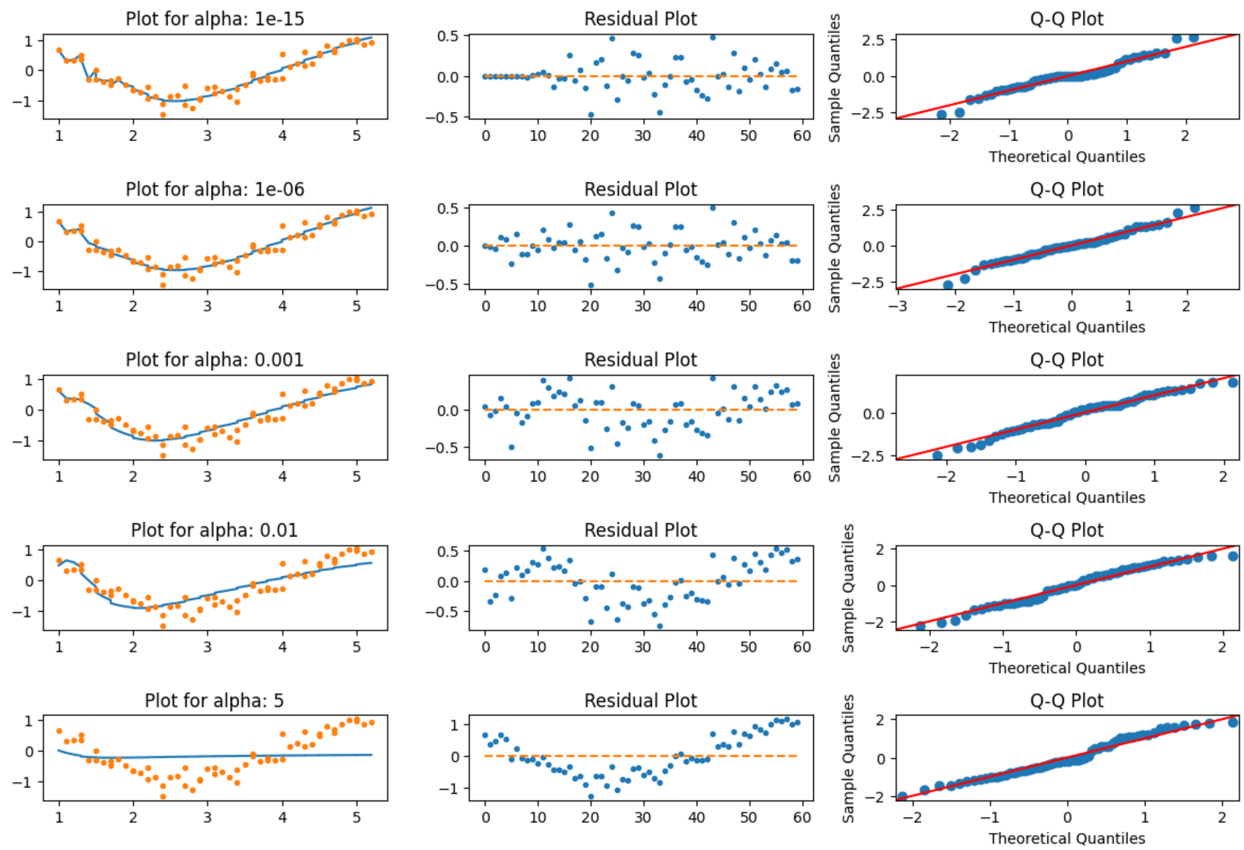
2. Ridge Regression

Ridge loss formula: $L = \Sigma\left(\hat{y}_i - y_i\right)^2 + \lambda \Sigma \beta^2$

The larger the value of λ, the more likely the coeffcients get closer and closer to zero but never become 0.

```
def ridge_regression(data, predictors, alpha, models_to_plot = {}):
  dataX = preprocessing.normalize(data[predictors])

  ridgereg = Ridge(alpha = alpha)
  ridgereg.fit(dataX, data['y'])
  y_pred = ridgereg.predict(dataX)

  if alpha in models_to_plot:

    x, y, z = models_to_plot[alpha]

    plt.subplot(x, y, z)
    plt.tight_layout()
    plt.plot(data['x'], y_pred)
    plt.plot(data['x'], data['y'], '.')

    plt.title('Plot for alpha: %.3g'%alpha)

    plt.subplot(x, y, z+1)
```

```
    plt.tight_layout()
    xlen = np.arange(y_pred.shape[0])
    plt.plot(xlen, data['y'] - y_pred, ".")
    plt.plot(xlen, 0*xlen, "--")
    plt.title('Residual Plot')

    ax = plt.subplot(x, y, z+2)
    plt.tight_layout()
    sm.qqplot(data['y'] - y_pred, line = '45', fit=True, dist =
stats.norm, ax = ax)
    plt.title('Q-Q Plot')

  rss = sum ((y_pred-data['y'])**2)
  ret = [rss]
  ret.extend([ridgereg.intercept_])
  ret.extend(ridgereg.coef_)
  return ret

predictors = ['x']
predictors.extend([f'x_{i}' for i in range(2, 16)])

alpha_ridge = [1e-15, 1e-6, 1e-3, 1e-2, 5]

# Dataframe to store the results
col = ['rss', 'intercept'] + [f'coef_x_{i}' for i in range(1, 16)]
ind = ['alpha_%.2g'%alpha_ridge[i] for i in range(len(alpha_ridge))]
coef_matrix_ridge = pd.DataFrame(index = ind, columns = col)

models_to_plot = {1e-15:(5,3,1), 1e-6:(5,3,4), 1e-3:(5,3,7), 1e-2:
(5,3,10), 5:(5,3,13)}

for i in range(len(alpha_ridge)):
  coef_matrix_ridge.iloc[i, ] = ridge_regression(data, predictors,
alpha_ridge[i], models_to_plot = models_to_plot)
```




```
pd.options.display.float_format = '{:,.2g}'.format
print(coef_matrix_ridge)

              rss  intercept  coef_x_1  coef_x_2  coef_x_3  coef_x_4  coef_x_5
\
alpha_1e-15  1.9   -3.4e+03  -8.1e+03   1.1e+04   4.3e+03  -2.2e+03  -1.2e+03

alpha_1e-06  2.1        -25       -65   1.2e+02       -61       -42        34

alpha_0.001  3.6         10       1.8       1.9      0.16     -0.96      -1.8

alpha_0.01   6.6          4     -0.91     -0.96      -1.7        -2        -2

alpha_5       24    -0.0084      0.04     0.045     0.049     0.055     0.062


              coef_x_6  coef_x_7  coef_x_8  coef_x_9  coef_x_10  coef_x_11
coef_x_12  \
alpha_1e-15  -3.5e+03   1.1e+04  -1.5e+04   7.4e+03    5.7e+03   -6.6e+03
3.8e+03
alpha_1e-06        29        42   1.2e+02  -1.4e+02         24        -76
52
alpha_0.001      -4.2      -6.4      -8.3      -8.8       -5.8        1.7
12
```

```
alpha_0.01       -2.3      -2.4        -2     -0.92        1.3        4.1
6.1
alpha_5         0.068     0.077     0.083      0.09       0.09      0.073
0.013

            coef_x_13 coef_x_14 coef_x_15
alpha_1e-15      8e+02        74   3.4e+03
alpha_1e-06         79       -54        33
alpha_0.001         14       -24      -5.4
alpha_0.01           3       -12      -1.3
alpha_5          -0.12     -0.33    -0.071
```

Observations

1. The rss value increased when alpha value is increased, implying that high alpha value result in underfitting.

2. As mentioned earlier the coefficients will be small but never be zero.

3. Lasso Regression

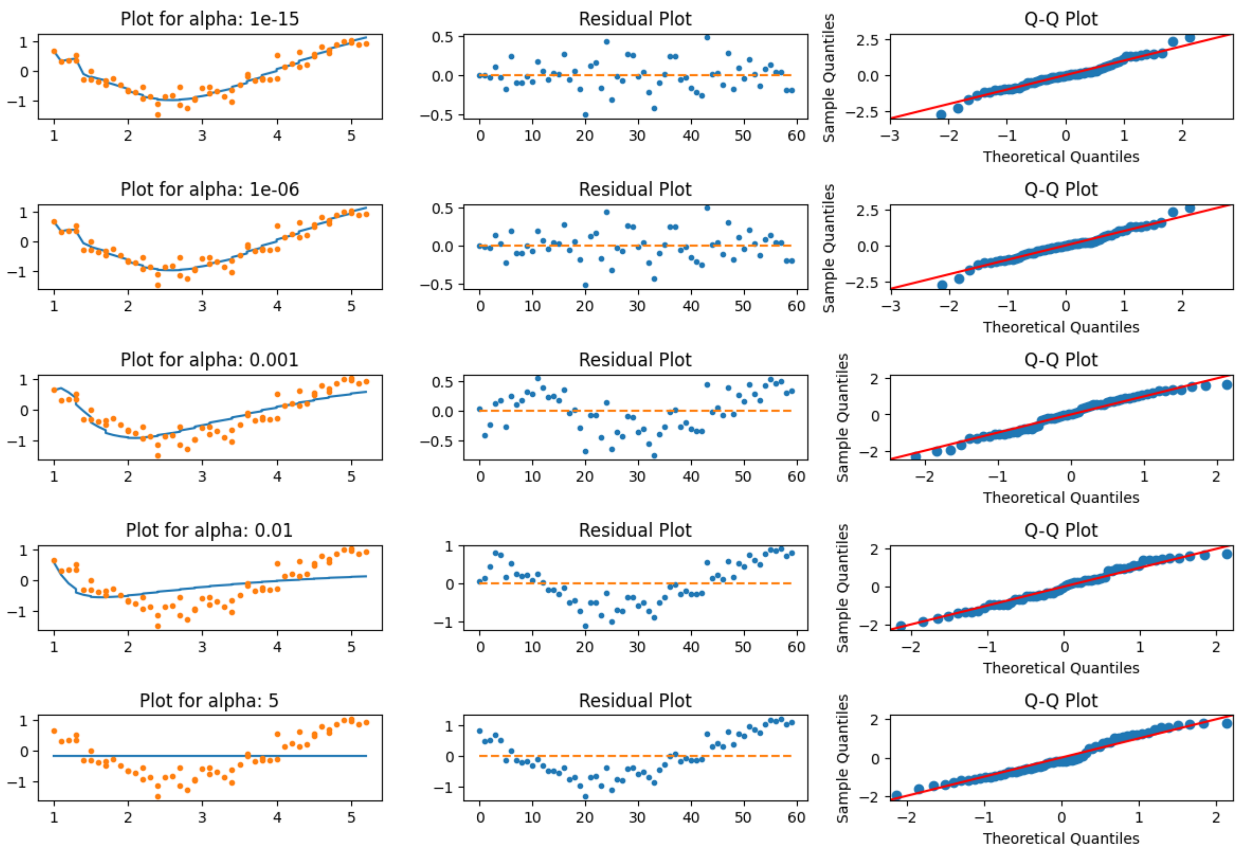
Lasso loss formula: $L = \Sigma\left(\hat{y}_i - y_i\right)^2 + \lambda\Sigma\left|\beta\right|$

1. As λ increases bias increases and variance decrease.
2. Theoretically when λ = ∞, all coefficients are eliminated.

```
def lasso_regression(data, predictors, alpha, models_to_plot = {}):
  dataX = preprocessing.normalize(data[predictors])

  lassoreg = Lasso(alpha = alpha, max_iter = int(1e5))
  lassoreg.fit(dataX, data['y'])
  y_pred = lassoreg.predict(dataX)

  if alpha in models_to_plot:

    x, y, z = models_to_plot[alpha]

    plt.subplot(x, y, z)
    plt.tight_layout()
    plt.plot(data['x'], y_pred)
    plt.plot(data['x'], data['y'], '.')

    plt.title('Plot for alpha: %.3g'%alpha)

    plt.subplot(x, y, z+1)
    plt.tight_layout()
    xlen = np.arange(y_pred.shape[0])
    plt.plot(xlen, data['y'] - y_pred, ".")
```

```
    plt.plot(xlen, 0*xlen, "--")
    plt.title('Residual Plot')

    ax = plt.subplot(x, y, z+2)
    plt.tight_layout()
    sm.qqplot(data['y'] - y_pred, line = '45', fit=True, dist =
stats.norm, ax = ax)
    plt.title('Q-Q Plot')

  rss = sum ((y_pred-data['y'])**2)
  ret = [rss]
  ret.extend([lassoreg.intercept_])
  ret.extend(lassoreg.coef_)
  return ret

predictors = ['x']
predictors.extend([f'x_{i}' for i in range(2, 16)])

alpha_lasso = [1e-15, 1e-6, 1e-3, 1e-2, 5]

# Dataframe to store the results
col = ['rss', 'intercept'] + [f'coef_x_{i}' for i in range(1, 16)]
ind = ['alpha_%.2g'%alpha_lasso[i] for i in range(len(alpha_lasso))]
coef_matrix_lasso = pd.DataFrame(index = ind, columns = col)

models_to_plot = {1e-15:(5,3,1), 1e-6:(5,3,4), 1e-3:(5,3,7), 1e-2:
(5,3,10), 5:(5,3,13)}

for i in range(len(alpha_lasso)):
  coef_matrix_lasso.iloc[i, ] = lasso_regression(data, predictors,
alpha_lasso[i], models_to_plot = models_to_plot)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_coordinate_descent.py:631: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 1.045e+00, tolerance: 2.627e-03
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordina
te_descent.py:631: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations, check the scale of
the features or consider increasing regularisation. Duality gap:
3.591e-01, tolerance: 2.627e-03
  model = cd_fast.enet_coordinate_descent(
```




```python
pd.options.display.float_format = '{:,.2g}'.format
print(coef_matrix_lasso)
```

```
              rss intercept coef_x_1 coef_x_2 coef_x_3 coef_x_4 coef_x_5
\
alpha_1e-15 2.1         -25 -2.6e+02  2.6e+02 -1.1e+02       62       -8

alpha_1e-06 2.1        -2.7 -1.8e+02  1.7e+02      -86     -4.5        0

alpha_0.001 6.4         2.7       -0       -0       -0       -0       -0

alpha_0.01   17         2.2        0        0        0        0        0

alpha_5      26       -0.18        0        0        0        0        0


              coef_x_6 coef_x_7 coef_x_8 coef_x_9 coef_x_10 coef_x_11
coef_x_12  \
alpha_1e-15  1.4e+02       35       61 -2.1e+02        46       -64
89
alpha_1e-06  1.3e+02       59       67    -2e+02        21       -58
69
alpha_0.001       -0     -9.8       -0       -0        -0         0
14
```

```
alpha_0.01          0        0        0        0         0         0
0
alpha_5             0        0        0        0         0         0
0

            coef_x_13 coef_x_14 coef_x_15
alpha_1e-15        43       -43        33
alpha_1e-06        46       -47        11
alpha_0.001         0       -12         0
alpha_0.01          0      -3.3      -1.4
alpha_5            -0        -0        -0
```

Observations

1. As we mentioned Lasso has the ability to elimate coefficients, which can be seen clearly.
2. For the same value of alpha, the value of coefficients is much smaller than what we got from ridge regression.
3. An important observation is that with increasing value of alpha the rss value increased drastically implying that model is underfitted.

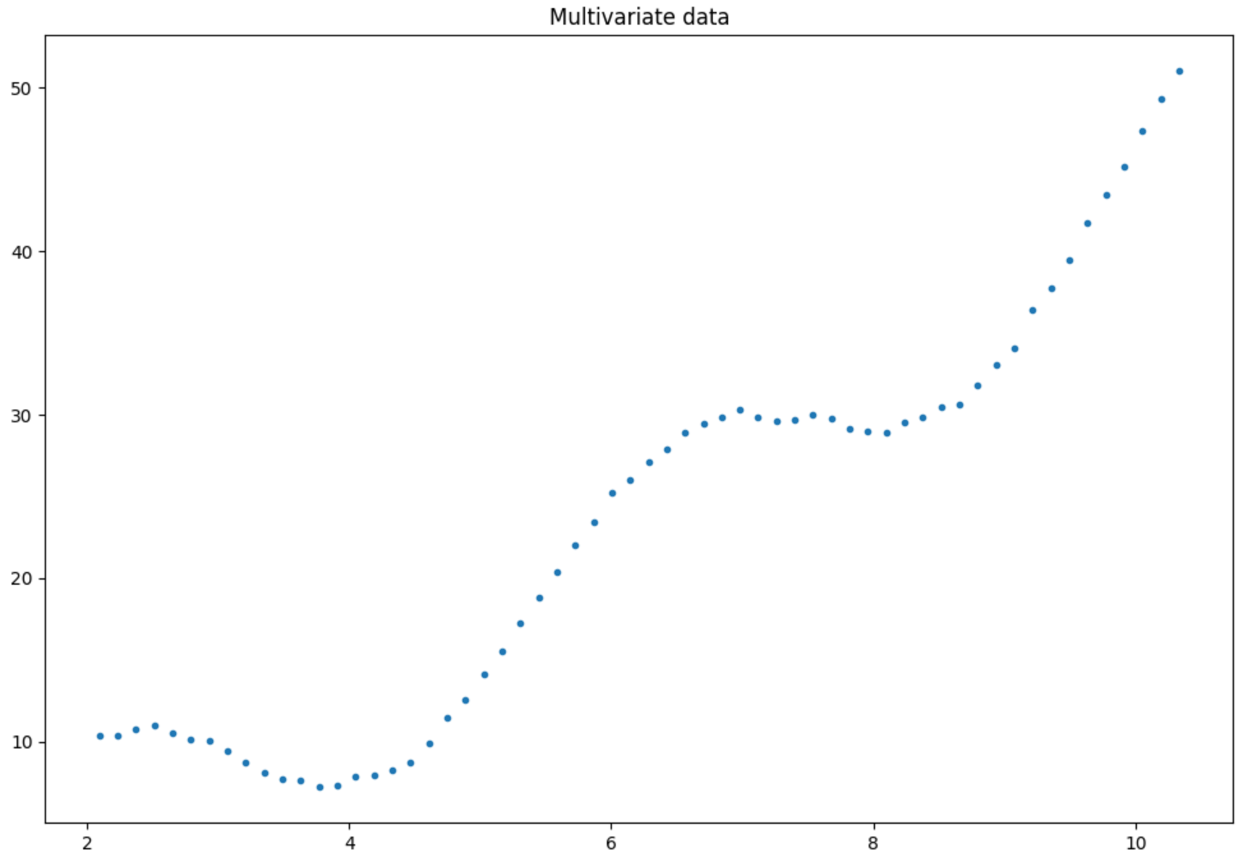
4. Multivariate Regression

Formula: $2x + x^2 + 5Sin(x) + \epsilon$

```
num_features = 3
x = np.array([
    [i*np.pi/90 for i in range(60, 300, 4)],
    [i*np.pi/180 for i in range(60, 300, 4)],
    [i/5 for i in range(60)]
]).T

y = 2*x[:, 0] + x[:, 1]**2 + 5*np.cos(x[:, 2]) + np.random.normal(0,
0.2, len(x))
data = pd.DataFrame(x, columns = [f'f{i}' for i in
range(num_features)])
data['y'] = y

data.head()

   f0  f1  f2  y
0 2.1   1   0 10
1 2.2 1.1 0.2 10
2 2.4 1.2 0.4 11
3 2.5 1.3 0.6 11
4 2.7 1.3 0.8 11
```

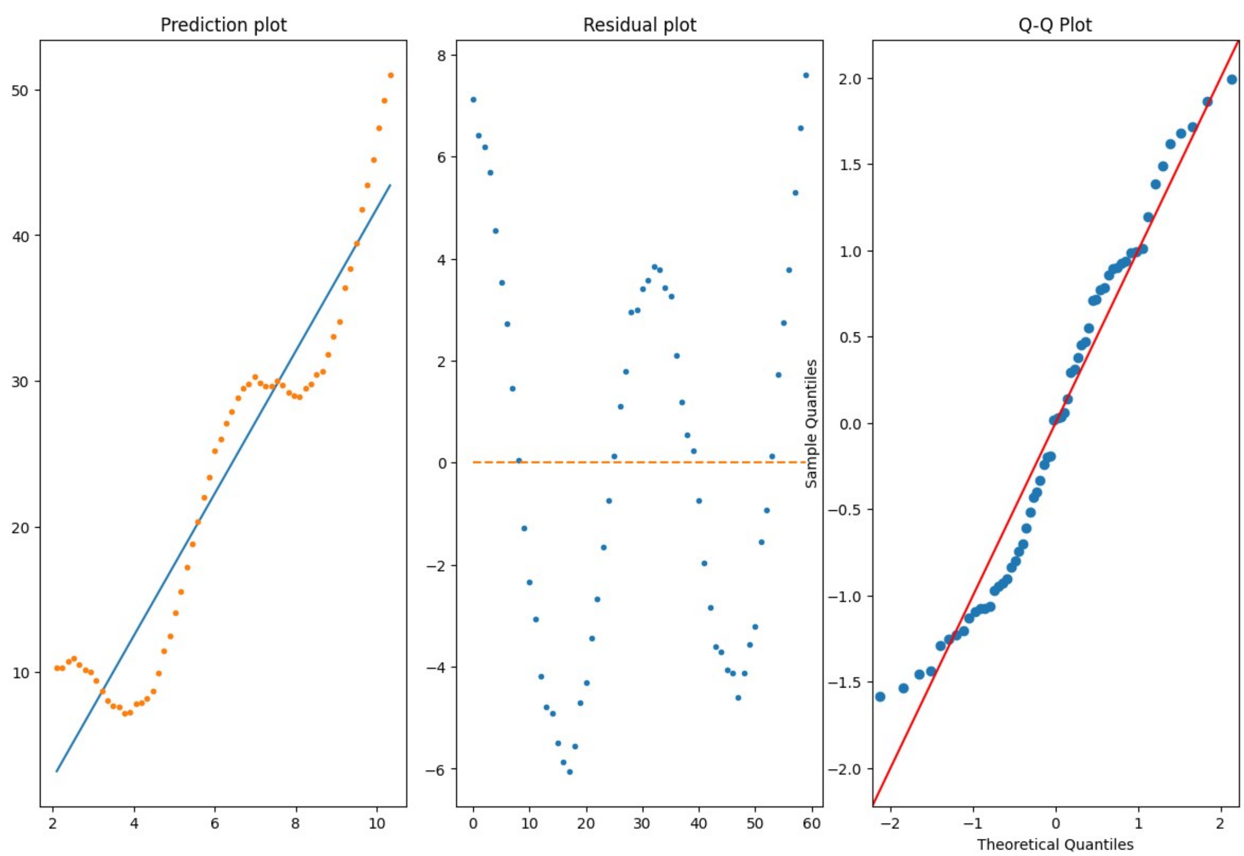
```python
plt.title('Multivariate data')
plt.plot(data['f0'], data['y'], '.')
```

```
[<matplotlib.lines.Line2D at 0x7807d3a18070>]
```




```python
def linear_reg(data, predictors):
  linreg = LinearRegression()

  linreg.fit(data[predictors], data['y'])
  y_pred = linreg.predict(data[predictors])

  plt.subplot(1, 3, 1)
  plt.tight_layout()
  plt.plot(data['f0'], y_pred)
  plt.plot(data['f0'], data['y'], '.')
  plt.title('Prediction plot')

  plt.subplot(1, 3, 2)
  plt.tight_layout()
  xlen = np.arange(y_pred.shape[0])
  plt.plot(xlen, data['y'] - y_pred, '.')
  plt.plot(xlen, 0*xlen, '--')
  plt.title('Residual plot')
```

```
  ax = plt.subplot(1, 3, 3)
  plt.tight_layout()
  sm.qqplot(data['y'] - y_pred, line = '45', fit = True, dist =
stats.norm, ax = ax)
  plt.title('Q-Q Plot')

  rss = sum((y_pred - data['y'])**2)
  rss = [rss]
  rss.extend([linreg.intercept_])
  rss.extend([linreg.coef_])
  return rss

predictors = [f'f{i}' for i in range(num_features)]
_ = linear_reg(data, predictors)
print(_)

[877.6512449517711, -0.6541105641963405, array([1.47795133,
0.73897567, 2.11700934])]
```



Find the minimum value of num poly features such that the model fits properly.

```
def poly_reg(data, predictors):
  linreg = LinearRegression()

  linreg.fit(data[predictors], data['y'])
  y_pred = linreg.predict(data[predictors])

  rss = sum((y_pred - data['y'])**2)
  rss = [rss]
  rss.extend([linreg.intercept_])
  rss.extend([linreg.coef_])
  return rss

predictors = [f'f{i}' for i in range(num_features)]
for i in range(3, 40):
  num_poly_features = i

  poly = PolynomialFeatures(num_poly_features)
  x_poly = poly.fit_transform(data[predictors])

  new_predictors = ['bias'] + [f'f{i - 1}' for i in range(1,
x_poly.shape[1])]
  new_data = pd.DataFrame(x_poly, columns = new_predictors)
  new_data['y'] = data['y']

  ret = poly_reg(new_data, new_predictors)
  print(f'for num_poly_features {i} the root square sum is {ret[0]}')
```

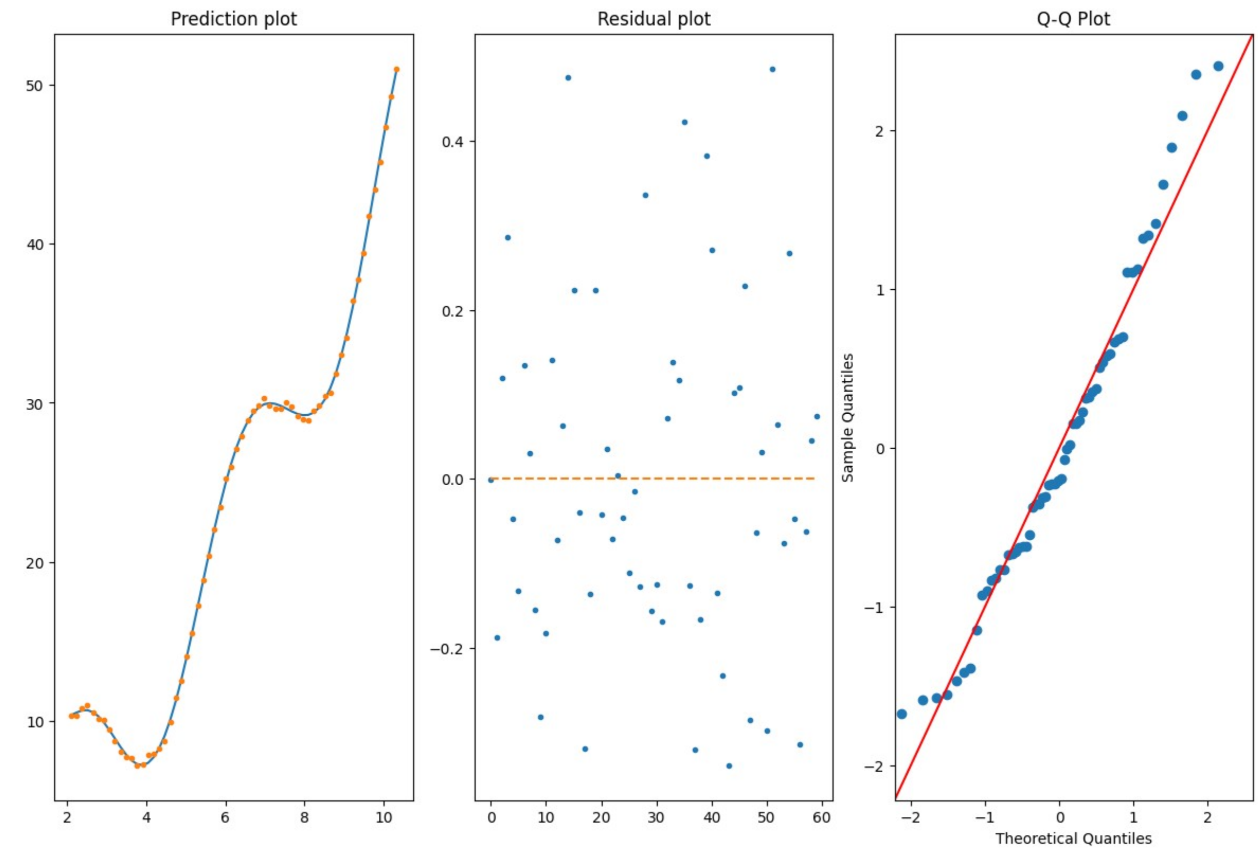
```
for num_poly_features 3 the root square sum is 682.9120358599463
for num_poly_features 4 the root square sum is 219.5130705727887
for num_poly_features 5 the root square sum is 160.47223675720454
for num_poly_features 6 the root square sum is 13.980713606597453
for num_poly_features 7 the root square sum is 8.04111960051769
for num_poly_features 8 the root square sum is 2.4473763040035985
for num_poly_features 9 the root square sum is 2.2749206483583495
for num_poly_features 10 the root square sum is 2.2731453735508254
for num_poly_features 11 the root square sum is 2.2722831375915455
for num_poly_features 12 the root square sum is 2.2539841763787596
for num_poly_features 13 the root square sum is 2.1749335003188395
for num_poly_features 14 the root square sum is 2.216803607336493
for num_poly_features 15 the root square sum is 2.238589285199747
for num_poly_features 16 the root square sum is 2.239455778018568
for num_poly_features 17 the root square sum is 2.3071093450342337
for num_poly_features 18 the root square sum is 2.513887957514842
for num_poly_features 19 the root square sum is 2.8877559180741295
for num_poly_features 20 the root square sum is 2.952106651424152
for num_poly_features 21 the root square sum is 3.9259121007301925
```

```
for num_poly_features 22 the root square sum is 7.246112408562274
for num_poly_features 23 the root square sum is 12.482427911721702
for num_poly_features 24 the root square sum is 20.630742006860164
for num_poly_features 25 the root square sum is 29.17248050472624
for num_poly_features 26 the root square sum is 34.53300762661395
for num_poly_features 27 the root square sum is 39.62415744520716
for num_poly_features 28 the root square sum is 39.90057780835804
for num_poly_features 29 the root square sum is 45.144403510874106
for num_poly_features 30 the root square sum is 58.29350522406081
for num_poly_features 31 the root square sum is 81.72547871486084
for num_poly_features 32 the root square sum is 116.14578647307108
for num_poly_features 33 the root square sum is 163.84992491104092
for num_poly_features 34 the root square sum is 220.78428214963714
for num_poly_features 35 the root square sum is 292.18674672316905
for num_poly_features 36 the root square sum is 373.60083916588275
for num_poly_features 37 the root square sum is 464.4454568410076
for num_poly_features 38 the root square sum is 560.6728079298354
for num_poly_features 39 the root square sum is 665.6622871055324
```

So it is clear from above values that for number of polynomial features = 8 onwards value of RSS decreses very slowly and it is minimum for 13, then it starts increasing again.

Hence we can say that the minimum value of num poly features such that the model fits properly is 8.

```
num_poly_features = 8

poly = PolynomialFeatures(num_poly_features)
x_poly = poly.fit_transform(data[predictors])

new_predictors = ['bias'] + [f'f{i - 1}' for i in range(1,
x_poly.shape[1])]
new_data = pd.DataFrame(x_poly, columns = new_predictors)
new_data['y'] = data['y']

ret = linear_reg(new_data, new_predictors)
```



The above plots are for num_poly_features = 8. It is quite visible that both the prediction plot and Q-Q plot are fitted properly.