id5055-tutorial-04

September 20, 2023

- [ ] Linear Regression
- [ ] Multivariate Linear Regression
- [ ] Linear Regression wiht non-linear features.
- [ ] Ridge Regression
- [ ] Lasso Regression

[ ]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from sklearn import preprocessing
import statsmodels.api as sm
import scipy.stats as stats
%matplotlib inline

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

[ ]:
```

```

[ ]:
```
# Seed for reproducabiltiy
seed = 0
np.random.seed(seed)
```

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

```
     x
0    1
1  1.1
2  1.2
3  1.3
4  1.3
```

Modified Data with Non-Linear Features

$x, x^2, x^3, \ldots, x^{15}$

```
[ ]: for i in range(2,16):  # power of 1 is already there
         colname = 'x_%d'%i
         data[colname] = data['x']**i
     print(data.head())
```

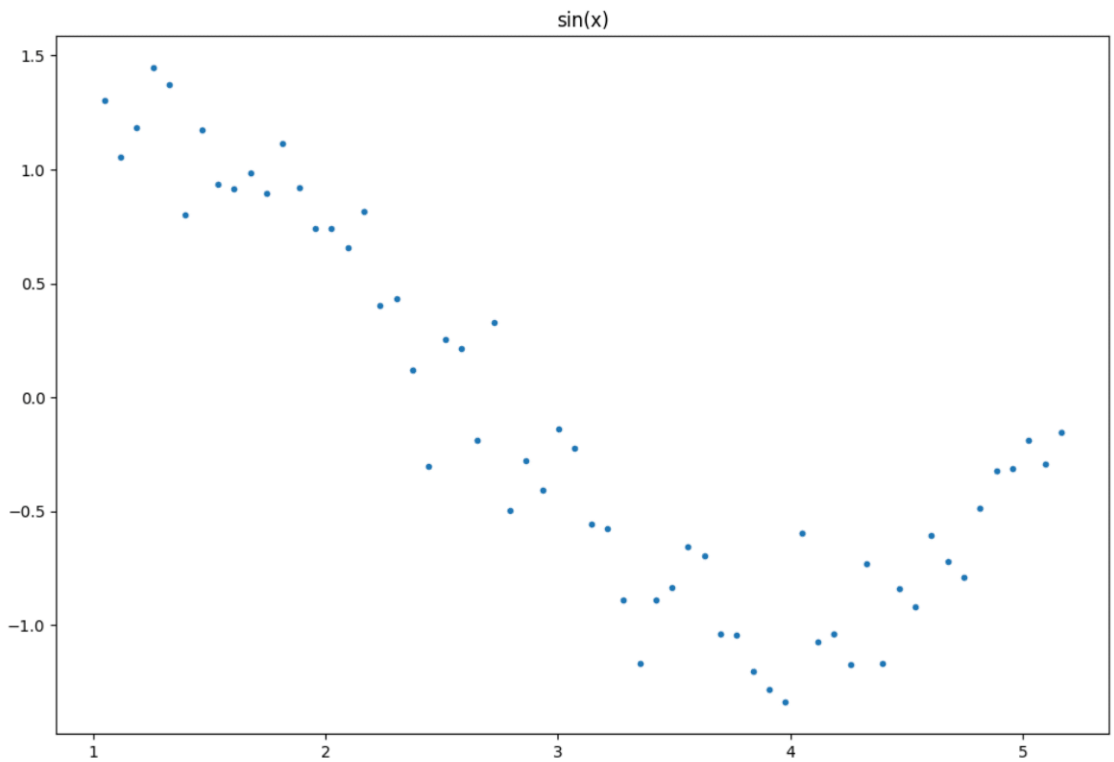
```
     x  x_2  x_3  x_4  x_5  x_6  x_7  x_8  x_9  x_10  x_11  x_12  x_13  x_14  \
0    1  1.1  1.1  1.2  1.3  1.3  1.4  1.4  1.5   1.6   1.7   1.7   1.8   1.9
1  1.1  1.2  1.4  1.6  1.7  1.9  2.2  2.4  2.7     3   3.4   3.8   4.2   4.7
2  1.2  1.4  1.7    2  2.4  2.8  3.3  3.9  4.7   5.5   6.6   7.8   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    19    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_1 = np.sin(1.2*x) + np.random.normal(0, 0.2, len(x))
     data['y_1'] = y_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    19    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_1
0     2  1.3
1   5.3  1.1
2    13  1.2
3    31  1.4
4    69  1.4
```

```
[ ]: plt.title("sin(x)")
     plt.plot(data['x'],data['y_1'],'.')
```

```
[ ]: [<matplotlib.lines.Line2D at 0x7f887129dcf0>]
```



This resembles a sine curve but not exactly because of the noise.

[ ]:

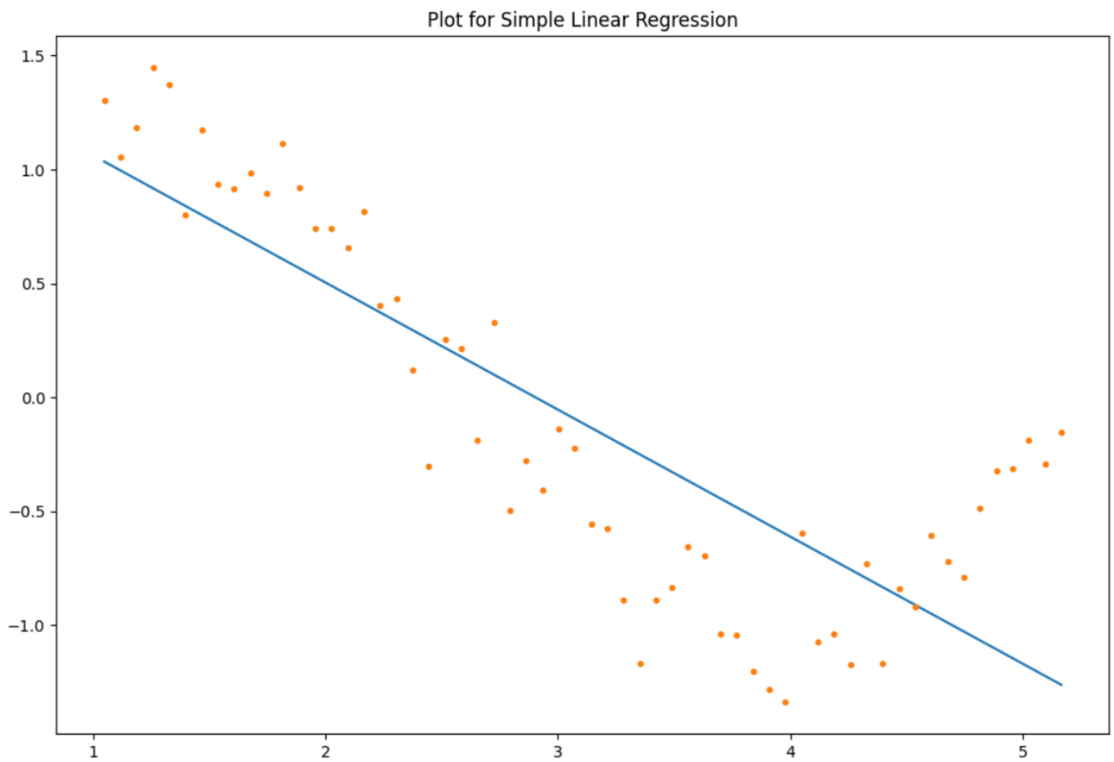
1 Linear Regression

Model: $Y_{pred} = W\mathbf{X} + b$

[ ]:
```
from sklearn.linear_model import LinearRegression
```

[ ]:
```
linReg = LinearRegression()

linReg.fit(data[['x']],data['y_1'])
y_pred = linReg.predict(data[['x']])

plt.plot(data['x'],y_pred)
plt.plot(data['x'],data['y_1'],'.')
plt.title('Plot for Simple Linear Regression')
```

[ ]: Text(0.5, 1.0, 'Plot for Simple Linear Regression')



1.1 Linear Regression with non-linear features

Model: $Y_{pred} = W_1\mathbf{X} + W_2\mathbf{X}^2 + W_3\mathbf{X}^3 ... + B$

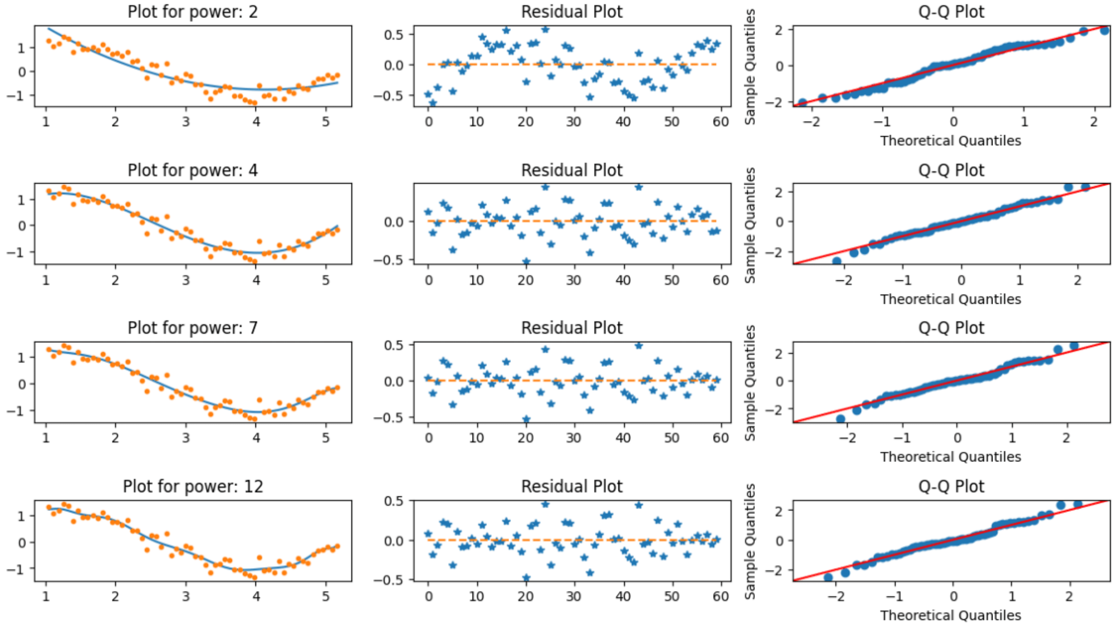
[ ]:
```python
def linear_regression(data, power, models_to_plot):
    predictors=['x']
    if power>=2:
        predictors.extend(['x_%d'%i for i in range(2,power+1)])

    # Fit the model
    linreg = LinearRegression()

    linreg.fit(data[predictors],data['y_1'])
    y_pred = linreg.predict(data[predictors])

    # Check if a plot is to be made for the given power of features
    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_1'], '.')
```

```
        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_1'] - 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_1'] - y_pred, line='45', fit=True, dist=stats.norm,␣
↪ax=ax)
        plt.title('Q-Q Plot')

    # Return the result in pre-defined format
    rss = sum((y_pred-data['y_1'])**2)
    rss = [rss]
    rss.extend([linreg.intercept_])
    rss.extend(linreg.coef_)
    return rss
```

Here RSS refers to the 'Residual Sum of Squares', which is nothing but the sum of squares of errors between the predicted and actual values in the training data set and is known as the cost function or the loss function.

NOTE: A residual is a measure of how far away a point is vertically from the regression line. Simply, it is the error between a predicted value and the observed actual value. Residual $= y - \hat{y}$

[ ]:
```
# Initialize a dataframe to store the results:
col = ['rss','intercept'] + ['coef_x_%d'%i for i in range(1,16)]
ind = ['model_pow_%d'%i for i in range(1,16)]
coef_matrix_simple = pd.DataFrame(index=ind, columns=col)

# Define the powers for which a plot is required:
models_to_plot = {2:(5,3,1), 4:(5,3,4), 7:(5,3,7), 12:(5,3,10), 25:(5,3,13)}

# Iterate through all powers and assimilate results
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
coef_matrix_simple
```

[ ]:

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	\
model_pow_1	13	1.6	-0.56	NaN	NaN	NaN	NaN	
model_pow_2	5.4	3.8	-2.2	0.27	NaN	NaN	NaN	
model_pow_3	2.3	0.28	2.1	-1.3	0.16	NaN	NaN	
model_pow_4	2.3	-0.79	3.8	-2.2	0.39	-0.018	NaN	
model_pow_5	2.2	2	-2	2.3	-1.2	0.26	-0.018	
model_pow_6	2.2	-0.96	5.5	-5.1	2.4	-0.71	0.11	
model_pow_7	2.2	10	-28	36	-23	8.6	-1.8	
model_pow_8	2.2	10	-28	36	-23	8.7	-1.8	
model_pow_9	2.2	-0.9	15	-36	42	-28	12	
model_pow_10	2.2	-3.3e+02	1.4e+03	-2.7e+03	2.8e+03	-1.9e+03	8.4e+02	
model_pow_11	2.2	-7.2e+02	3.2e+03	-6.4e+03	7.3e+03	-5.3e+03	2.6e+03	
model_pow_12	2.1	1.4e+03	-7.6e+03	1.8e+04	-2.6e+04	2.4e+04	-1.5e+04	
model_pow_13	2.1	9e+03	-5e+04	1.2e+05	-1.8e+05	1.8e+05	-1.2e+05	
model_pow_14	2.1	1.7e+03	-7.6e+03	1.4e+04	-1e+04	-2.4e+03	1.2e+04	
model_pow_15	2.1	44	-2.5e+02	4.2e+02	-1e+02	-3.9e+02	3.1e+02	

	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	-0.007	NaN	NaN	NaN	NaN	NaN
model_pow_7	0.21	-0.0099	NaN	NaN	NaN	NaN
model_pow_8	0.21	-0.01	1.2e-05	NaN	NaN	NaN
model_pow_9	-2.9	0.45	-0.038	0.0014	NaN	NaN
model_pow_10	-2.5e+02	49	-6.2	0.46	-0.015	NaN
model_pow_11	-9.1e+02	2.2e+02	-35	3.7	-0.23	0.0062
model_pow_12	6.7e+03	-2.1e+03	4.8e+02	-75	7.7	-0.46
model_pow_13	5.9e+04	-2.1e+04	5.6e+03	-1.1e+03	1.4e+02	-13
model_pow_14	-1.3e+04	7.7e+03	-3.2e+03	9.1e+02	-1.8e+02	26
model_pow_15	2.5e+02	-5.7e+02	4.6e+02	-2.3e+02	73	-16

	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	0.013	NaN	NaN	NaN
model_pow_13	0.69	-0.017	NaN	NaN
model_pow_14	-2.4	0.13	-0.0032	NaN
model_pow_15	2.4	-0.23	0.013	-0.00034

It is clearly evident that the size of coefficients increases exponentially with an increase in model complexity.

Hopefully, this gives some intuition into why putting a constraint on the magnitude of coefficients can be a good idea to reduce model *complexity*.

⊙ But what is the problem with the increasing size of coefficients?

→ It means that a lot of emphasis has been given on that feature, i.e., the particular feature is a good predictor for the outcome. However, when it becomes too large, the algorithm starts modeling intricate relations to estimate the output and ends up overfitting the particular training data.

[ ]:

```

```

2 Ridge Regression

- Higher the alpha value, more restriction on the coefficients;
- low alpha → more generalization,

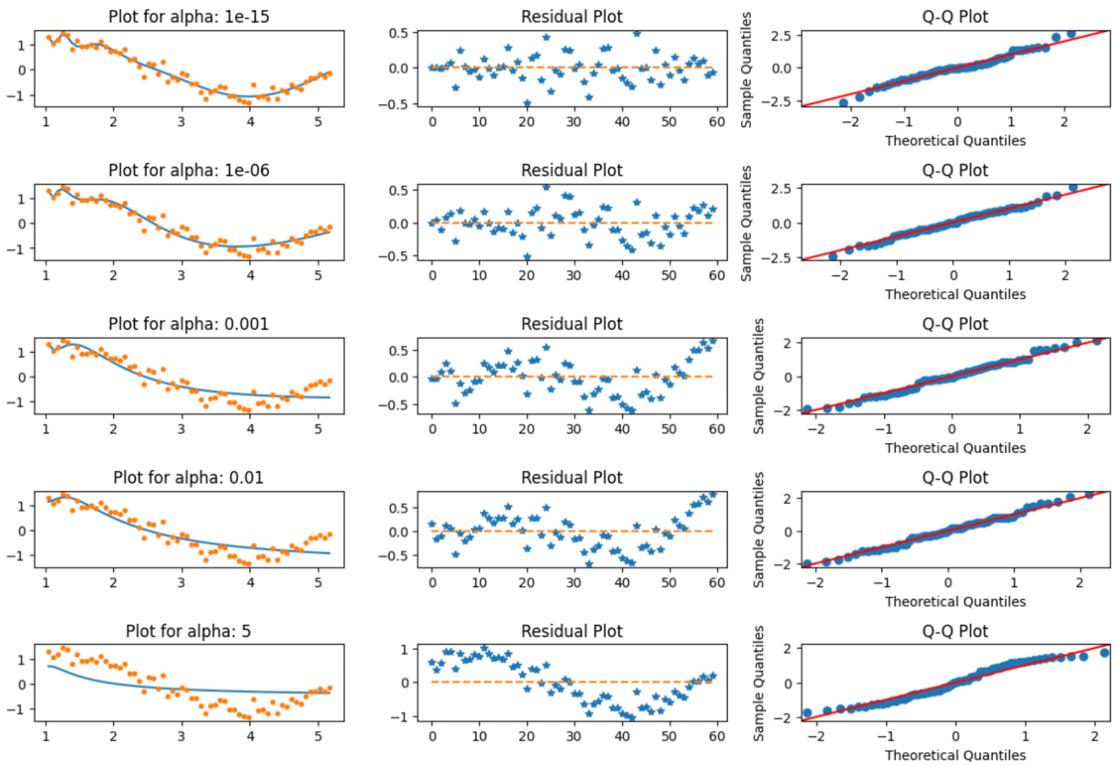
- **Ridge Loss Formula:**
$$L = \sum(\hat{y}_i - y_i)^2 + \lambda \sum \beta^2$$
Sum of Errors + Sum of the squares of coefficients
- Ridge assigns a penalty that is the squared magnitude of the coefficients to the loss function multiplied by lambda.
- As Lasso does, ridge also adds a penalty to coefficients the model overemphasizes.
- The value of lambda also plays a key role in how much weight you assign to the penalty for the coefficients.
- The larger your value of lambda, the more likely your coefficients get closer and closer to zero.
- Unlike lasso, the ridge model will not shrink these coefficients to zero.

[ ]:
```
from sklearn.linear_model import Ridge
def ridge_regression(data, predictors, alpha, models_to_plot={}):
    # Normalize
    dataX = preprocessing.normalize(data[predictors])

    # Fit the model

    ## Ridge = LinearModel + \alpha * ||W||_2^2
    ridgereg = Ridge(alpha=alpha)
    ridgereg.fit(dataX,data['y_1'])
    y_pred = ridgereg.predict(dataX)

    # Check if a plot is to be made for the entered alpha
    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_1'], '.')
        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_1'] - 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_1'] - y_pred, line='45', fit=True, dist=stats.norm,␣
 ↪ax=ax)
        plt.title('Q-Q Plot')
```

```
    #Return the result in pre-defined format
    rss = sum((y_pred-data['y_1'])**2)
    ret = [rss]
    ret.extend([ridgereg.intercept_])
    ret.extend(ridgereg.coef_)
    return ret
```

[ ]:
```
#Initialize predictors to be set of 15 powers of x
predictors=['x']
predictors.extend(['x_%d'%i for i in range(2,16)])

#Set the different values of alpha to be tested
alpha_ridge = [1e-15, 1e-6, 1e-3, 1e-2, 5]

#Initialize the dataframe for storing coefficients.
col = ['rss','intercept'] + ['coef_x_%d'%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)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216:
LinAlgWarning: Ill-conditioned matrix (rcond=9.86554e-17): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```



[ ]:
```
pd.options.display.float_format = '{:,.2g}'.format
coef_matrix_ridge
```

[ ]:

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	coef_x_12	coef_x_13	coef_x_14	coef_x_15
alpha_1e-15	2	2.1e+03	4.6e+04	-3.1e+05	4.8e+05	1e+05	-6.4e+05	-5.5e+04	8.7e+05	-2.6e+05	-8.7e+05	1.1e+06	-6.2e+05	2e+05	-3.7e+04	3.4e+03	-2.3e+03
alpha_1e-06	2.7	1.5e+02	-34	-65	-87	-92	-73	-27	41	1.1e+02	1.1e+02	-5.1	-2.2e+02	-2.4e+02	2.7e+02	-1.2e+02	-1.4e+02
alpha_0.001	6.1	2.6	0.7	1.3	1.8	2.1	1.9	1.2	-0.36	-2.8	-5.7	-7.6	-5.4	3.8	14	-5.2	-3
alpha_0.01	7	0.93	1	0.89	0.68	0.41	0.048	-0.4	-0.92	-1.5	-1.8	-1.8	-0.73	1.6	4.1	1.5	-2.3
alpha_5	21	0.056	0.043	0.048	0.055	0.064	0.075	0.09	0.11	0.14	0.18	0.24	0.33	0.45	0.59	0.56	-0.57

- High alpha values can lead to significant underfitting.
- The RSS increases with an increase in alpha.
- Though the coefficients are really small, they are NOT zero.

3 Lasso Regression

The acronym ''LASSO" stands for Least Absolute Shrinkage and Selection Operator.

- LASSO uses shrinkage.
  - Shrinkage is where data values are shrunk towards a central point as the mean.
- The lasso procedure encourages simple, sparse models.
- **Lasso Formula:**

$$L = \sum(\hat{y_i} - y_i)^2 + \lambda \sum |\beta|$$

- $\beta \rightarrow$ magnitude of coefficients
- The value of lambda also plays a key role in how much weight you assign to the penalty for the coefficients.
- This penalty reduces the value of many coefficients to zero, all of which are eliminated.
- Depending upon λ:
  - When $\lambda = 0$, no parameters are eliminated. The estimate is equal to the one found with linear regression.
  - As λ increases, more and more coefficients are set to zero and eliminated (theoretically, when $\lambda = \infty$, all coefficients are eliminated).
  - As λ increases, bias increases.
  - As λ decreases, variance increases.

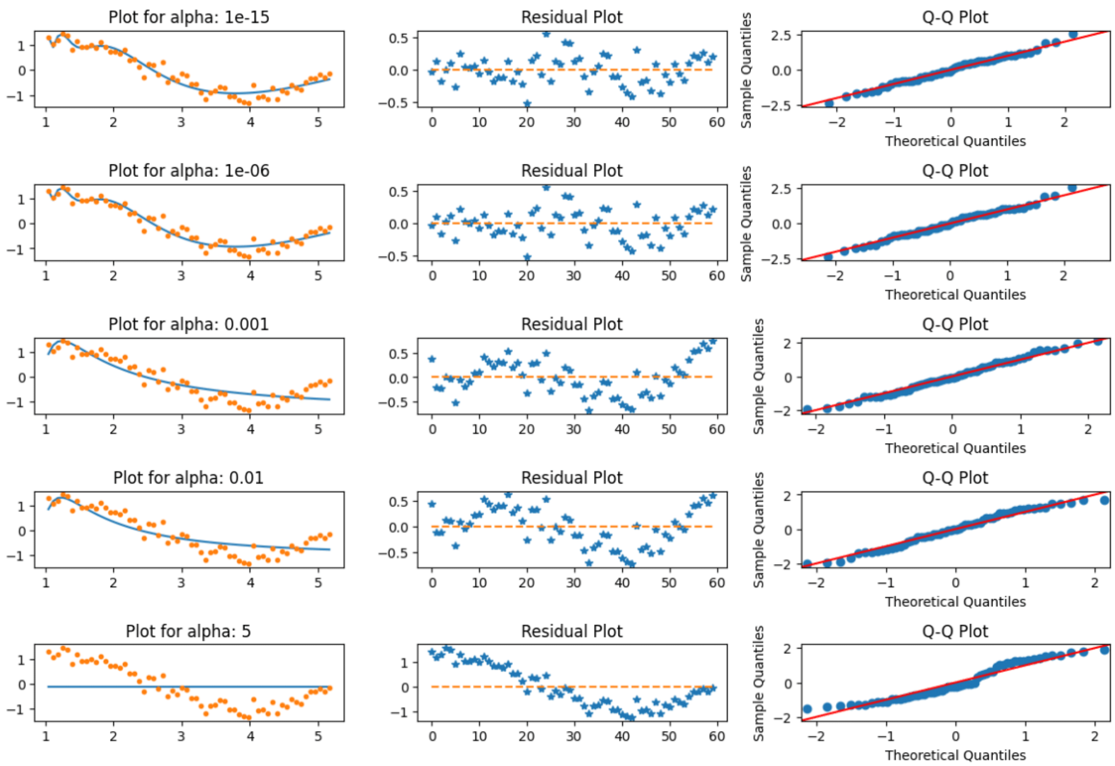
[ ]:
```python
from sklearn.linear_model import Lasso
def lasso_regression(data, predictors, alpha, models_to_plot={}):
    # Normalize
    dataX = preprocessing.normalize(data[predictors])

    #Fit the model

    ## Lasso = LinearModel + \alpha ||W||_1
    lassoreg = Lasso(alpha=alpha, max_iter=int(1e5))
    lassoreg.fit(dataX,data['y_1'])
    y_pred = lassoreg.predict(dataX)

    #Check if a plot is to be made for the entered alpha
    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_1'], '.')
        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_1'] - 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_1'] - y_pred, line='45', fit=True, dist=stats.norm,␣
↪ax=ax)
        plt.title('Q-Q Plot')


    #Return the result in pre-defined format
    rss = sum((y_pred-data['y_1'])**2)
    ret = [rss]
    ret.extend([lassoreg.intercept_])
    ret.extend(lassoreg.coef_)
    return ret
```

[ ]:
```
#Initialize predictors to all 15 powers of x
predictors=['x']
predictors.extend(['x_%d'%i for i in range(2,16)])

#Define the alpha values to test
alpha_lasso = [1e-15, 1e-6, 1e-3, 1e-2, 5]

#Initialize the dataframe to store coefficients
col = ['rss','intercept'] + ['coef_x_%d'%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)

#Define the models to plot
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)}

#Iterate over alpha values:
for i in range(len(alpha_lasso)):
    coef_matrix_lasso.iloc[i,] = lasso_regression(data, predictors,␣
↪alpha_lasso[i], 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: 1.433e+00, tolerance: 4.025e-03
  model = cd_fast.enet_coordinate_descent(
/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: 1.007e+00, tolerance: 4.025e-03
  model = cd_fast.enet_coordinate_descent(
```



[ ]:
```
coef_matrix_lasso
```

[ ]:
```
                rss intercept coef_x_1 coef_x_2 coef_x_3 coef_x_4 coef_x_5  \
alpha_1e-15 2.9      2e+02  4.1e+02 -5.6e+02 -3.4e+02 -1.3e+02       41
alpha_1e-06 2.9    1.9e+02  2.6e+02 -3.5e+02 -3.3e+02 -1.3e+02     -7.8
alpha_0.001 7.4       -1.1        0        0        0        0        0
alpha_0.01  8.1       -1.1        0        0        0        0        0
alpha_5      40      -0.11        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  1.5e+02  1.8e+02  1.1e+02      -29  -1.7e+02    -2e+02       -60
```

alpha_1e-06	1.1e+02	1.9e+02	1.2e+02	4.7	-1.8e+02	-2.1e+02	-52
alpha_0.001	0	-0	-0	-0	-0	-0	0
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	1.2e+02	-92	-1.9e+02
alpha_1e-06	1.2e+02	-90	-1.8e+02
alpha_0.001	4.8	1.8	-0.38
alpha_0.01	5.7	0.41	-0
alpha_5	0	0	-0

Apart from the expected inference of higher RSS for higher alphas, we can see the following:

- For the same values of alpha, the coefficients of lasso regression are much smaller than that of ridge regression (compare row 1 of the 2 tables).
- For the same alpha, lasso has higher RSS (poorer fit) as compared to ridge regression.
- Many of the coefficients are zero, even for very small values of alpha.

- The ridge coefficients are a reduced factor of the simple linear regression coefficients and thus never attain zero values but very small values.
- The lasso coefficients become zero in a certain range and are reduced by a constant factor, which explains their low magnitude in comparison to the ridge.

[ ]:

```

```

4 Multivariate Regression

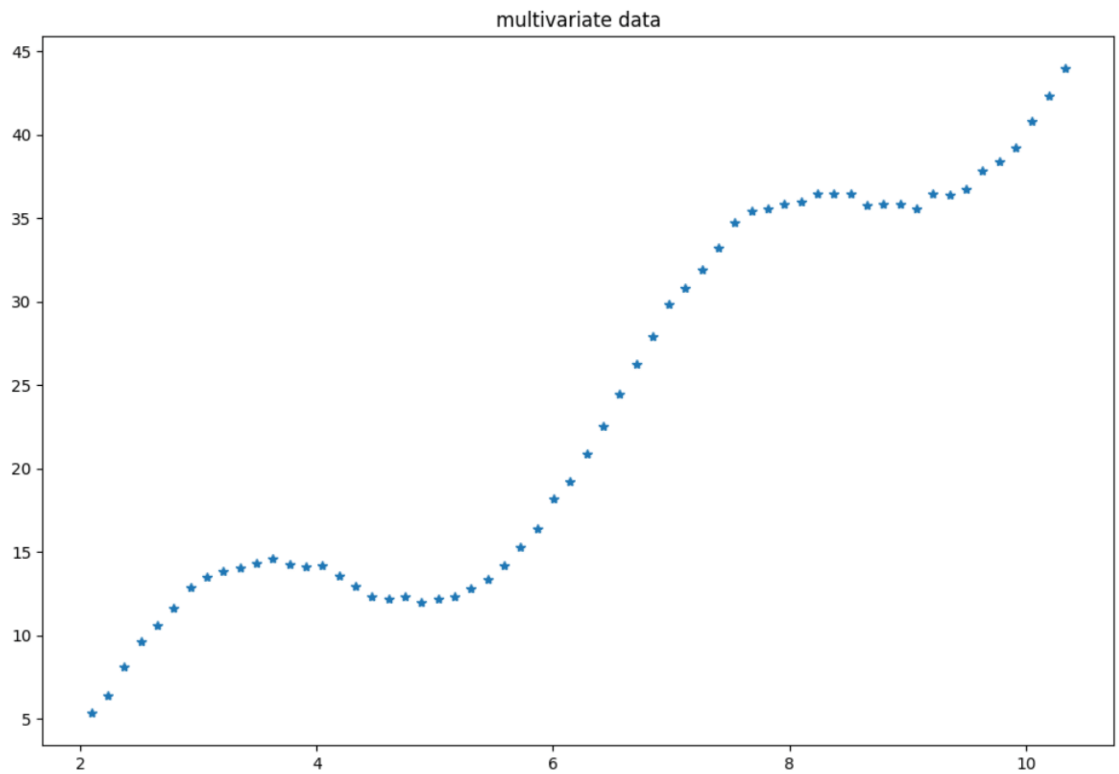
[ ]:

```
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.sin(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  5.4
1  2.2  1.1  0.2  6.4
2  2.4  1.2  0.4  8.1
3  2.5  1.3  0.6  9.7
4  2.7  1.3  0.8   11
```

[ ]:
```
plt.title("multivariate data")
plt.plot(data['f0'], data['y'], '*')
```

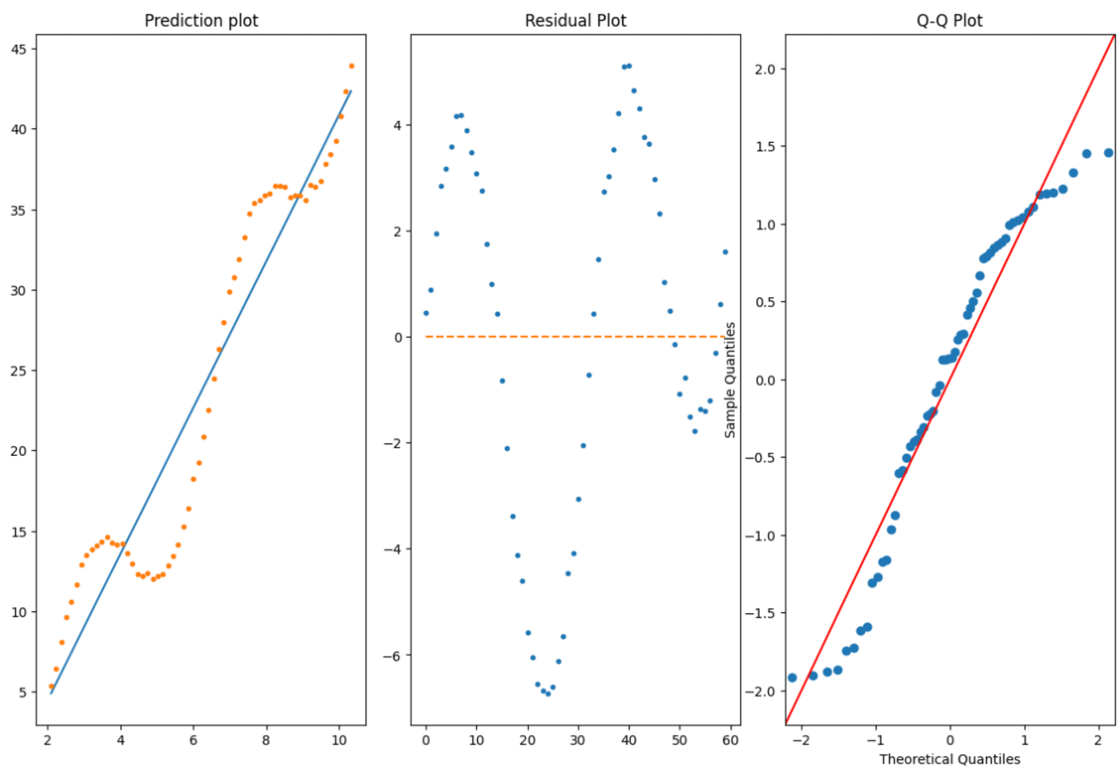
[ ]:
```
[<matplotlib.lines.Line2D at 0x7f88708c5990>]
```



[ ]:
```
def linear_regression(data, predictors):
    # Fit the model
    linreg = LinearRegression()

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

    # Check if a plot is to be made for the given power of features
    plt.subplot(1, 3, 1)
    plt.tight_layout()
```

```python
    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')

    # Return the result in pre-defined format
    rss = sum((y_pred-data['y'])**2)
    rss = [rss]
    rss.extend([linreg.intercept_])
    rss.extend(linreg.coef_)
    return rss
```

[ ]:
```python
predictors = [f"f{i}" for i in range(num_features)]
_ = linear_regression(data, predictors)
```



[ ]:
```
from sklearn.preprocessing import PolynomialFeatures
```

[ ]:
```
predictors = [f"f{i}" for i in range(num_features)]

num_poly_features = 20

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

# example for 2 features: f0, f1, f2, f0^2, f1^2, f2^2, f0f1, f1f2, f2f0, bias
x_poly.shape
```

[ ]:
```
(60, 1771)
```

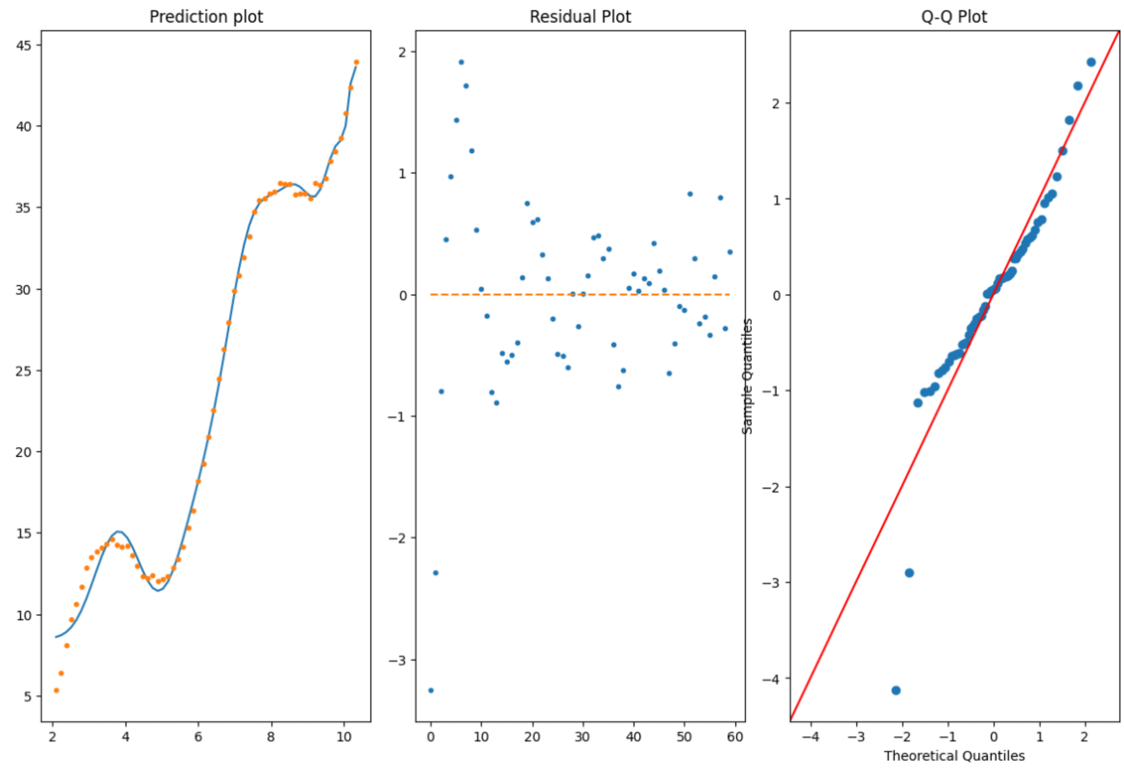
[ ]:
```
new_predictors = ["bias"] + [f"f{i-1}" for i in range(1, x_poly.shape[1])]
new_data = pd.DataFrame(x_poly, columns=new_predictors)
new_data['y'] = data['y']
```

[ ]:
```
new_data[new_predictors].head()
```

[ ]:
```
   bias  f0  f1  f2  f3  f4   f5  f6   f7   f8  …   f1760   f1761   f1762  \
0     1 2.1   1   0 4.4 2.2    0 1.1    0    0  …       0       0       0
1     1 2.2 1.1 0.2   5 2.5 0.45 1.2 0.22 0.04  … 5.5e-08 9.9e-09 1.8e-09
2     1 2.4 1.2 0.4 5.6 2.8 0.95 1.4 0.47 0.16  …  0.0002 6.6e-05 2.2e-05
3     1 2.5 1.3 0.6 6.3 3.2  1.5 1.6 0.75 0.36  …   0.028   0.014  0.0065
4     1 2.7 1.3 0.8   7 3.5  2.1 1.8  1.1 0.64  …     1.1    0.66     0.4

     f1763   f1764   f1765   f1766   f1767   f1768   f1769
0        0       0       0       0       0       0       0
1 3.2e-10 5.7e-11   1e-11 1.8e-12 3.3e-13 5.9e-14   1e-14
2 7.5e-06 2.5e-06 8.5e-07 2.9e-07 9.7e-08 3.3e-08 1.1e-08
3  0.0031  0.0015  0.0007 0.00034 0.00016 7.7e-05 3.7e-05
4    0.24    0.14   0.087   0.053   0.032   0.019   0.012

[5 rows x 1771 columns]
```

[ ]:
```

```

[ ]:
```
_ = linear_regression(new_data, new_predictors)
```



[ ]:

[ ]:

5 Questions

[ ]:
```
np.random.seed(0)

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):  # power of 1 is already there
    colname = 'x_%d'%i
    data[colname] = data['x']**i
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	19	24	
4	1.3	1.8	2.3	3.1	4.1	5.4	7.2	9.6	13	17	22	30	39	52	

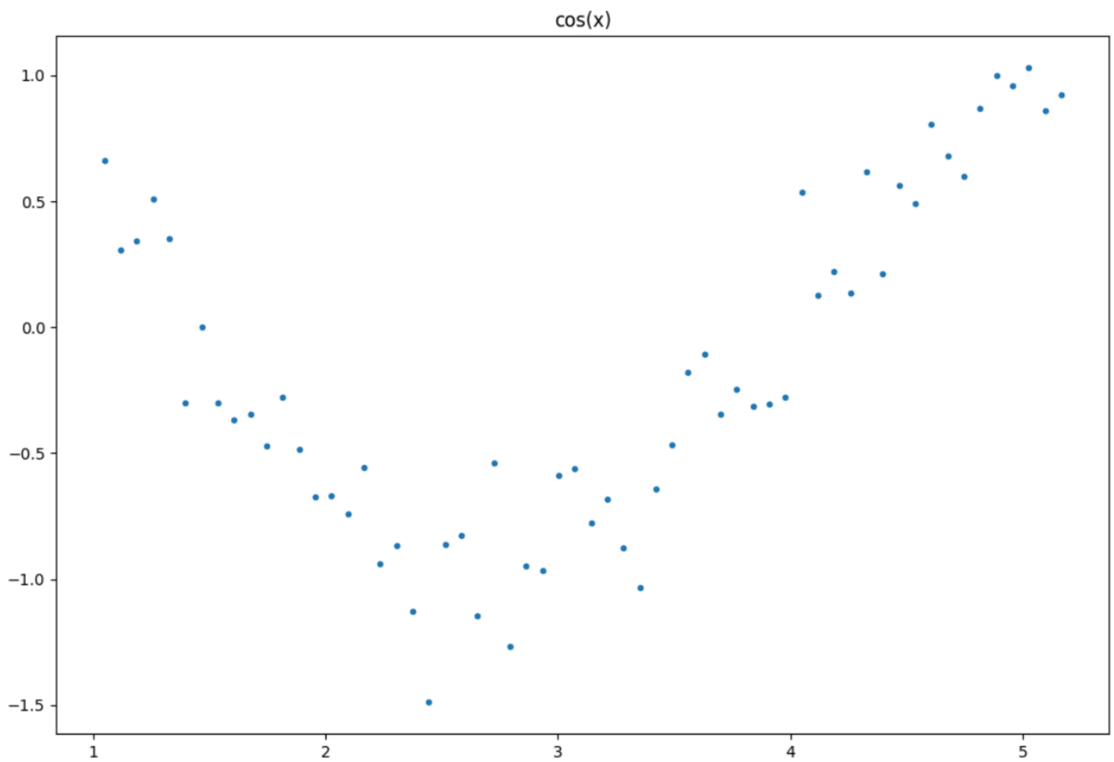
```
   x_15
0     2
1   5.3
2    13
3    31
4    69
```

[ ]:
```python
y_2 = np.cos(1.2*x) + np.random.normal(0, 0.2, len(x))
data['y_2'] = y_2

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    19    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_2
0     2 0.66
1   5.3 0.31
2    13 0.34
3    31 0.51
4    69 0.35
```

[ ]:
```python
plt.title("cos(x)")
plt.plot(data['x'], data['y_2'], '.')
```

[ ]: `[<matplotlib.lines.Line2D at 0x7f88710a99f0>]`




```
[ ]: # Q1. Run linear, non-linear, lasso, ridge regression on the given cosine data.
     ↪Report your observations.
     # Q2. Find the mimumum value of num_poly_features such that the model fits
     ↪properly.
```

```
[ ]:
```