

LinearRegression

November 26, 2018

1 Linear Regression

Linear Regression is one of the simplest yet fundamental statistical learning techniques. It is a great initial step towards more advanced and computationally demanding methods.

This article aims to cover a statistically sound approach to Linear Regression and its inferences while tying these to popular statistical packages and reproducing the results.

We first begin with a brief description of Linear Regression and move on to investigate it in light of a dataset.

1.1 1 - Description

Linear regression on p variables focusses on fitting a straight line in p -dimensions that passes as close as possible to the data points in order to reduce error.

General Characteristics: - A supervised learning technique - Useful for predicting a quantitative response - Linear Regression attempts to fit a function to predict a response variable - The problem is reduced to a parametric problem of finding a set of parameters - The function shape is limited (as a function of the parameters)

1.2 2- Advertising and Housing Datasets

The Advertising dataset is obtained from <http://www-bcf.usc.edu/~gareth/ISL/data.html> and contains 200 datapoints of sales of a particular product, and TV, newspaper and radio advertising budgets (all figures are in units of \$1,000s).

The HousePrice dataset is obtained from <https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data> and contains 1460 houses along many properties (only quantitative properties) including their sales price.

First we import the required libraries

```
In [2]: # Import modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
from numpy.random import RandomState
import math
%matplotlib inline
```

Then we import the datasets

```
In [3]: # Import Advertising dataset (http://www-bcf.usc.edu/~gareth/ISL/data.html)
        advert = pd.read_csv("Advertising.csv").iloc[:,1:]

        # Import House Prices dataset - Only quantitative fields and cleaned (https://www.kaggle.com/lucianopereira/house-prices-advanced-regression-techniques)
        housePrice = pd.read_csv("HousePrice.csv").iloc[:,1:]

In [4]: print("Number of observations (n) =",advert.shape[0])
        print("Number of predictor variables (p) =",advert.shape[1]-1)
        print()
        print("Advertising.csv")
        display(advert.head())
```

Number of observations (n) = 200

Number of predictor variables (p) = 3

Advertising.csv

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

```
In [5]: print("Number of observations (n) =",housePrice.shape[0])
        print("Number of predictor variables (p) =",housePrice.shape[1]-1)
        print()
        print("HousePrice.csv")
        display(housePrice.head())
```

Number of observations (n) = 1460

Number of predictor variables (p) = 34

HousePrice.csv

	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	\
0	8450	7	5	2003	2003	196.0	
1	9600	6	8	1976	1976	0.0	
2	11250	7	5	2001	2002	162.0	
3	9550	7	5	1915	1970	0.0	
4	14260	8	5	2000	2000	350.0	

	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	...	WoodDeckSF	\
0	706	0	150	856	...	0	

1	978	0	284	1262	...	298
2	486	0	434	920	...	0
3	216	0	540	756	...	0
4	655	0	490	1145	...	192

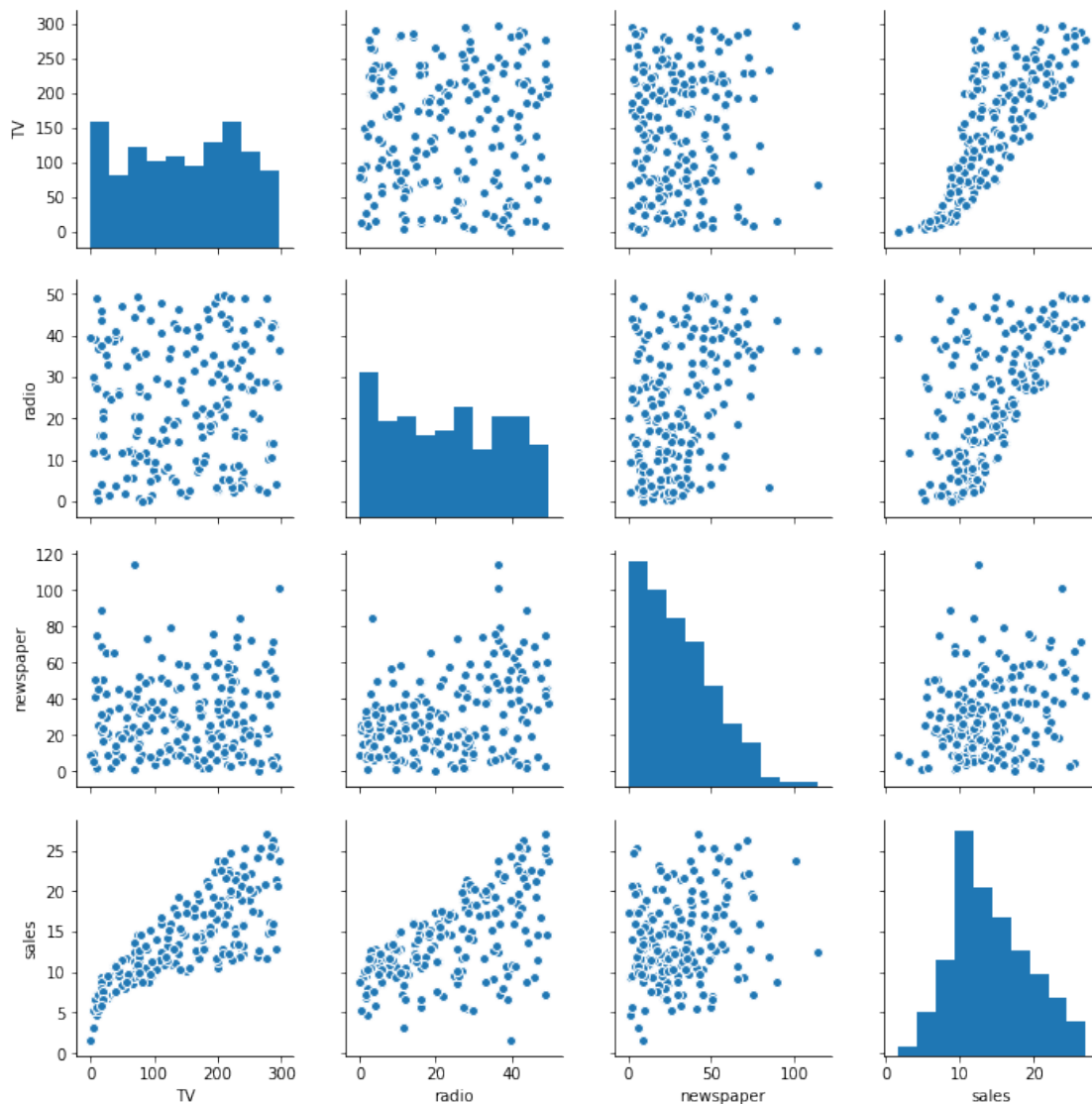
	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	\
0	61	0	0	0	0	0	
1	0	0	0	0	0	0	
2	42	0	0	0	0	0	
3	35	272	0	0	0	0	
4	84	0	0	0	0	0	

	MoSold	YrSold	SalePrice
0	2	2008	208500
1	5	2007	181500
2	9	2008	223500
3	2	2006	140000
4	12	2008	250000

[5 rows x 35 columns]

For the Advertising dataset the response variable is "sales". The predictor variables are "TV", "radio" and "newspaper". We can produce a pairplot of the data below.

```
In [6]: ax = sns.pairplot(data=advert)
```



By looking at a pairplot to see the simple relationships between the variables, we see a strong positive correlation between sales and TV. A similar relationship between sales and radio is also observed. Newspaper and radio seem to have a slight positive correlation also. We can see this in the correlation matrix below.

```
In [7]: advert.corr()
```

```
Out [7]:
```

	TV	radio	newspaper	sales
TV	1.000000	0.054809	0.056648	0.782224
radio	0.054809	1.000000	0.354104	0.576223
newspaper	0.056648	0.354104	1.000000	0.228299
sales	0.782224	0.576223	0.228299	1.000000

We may want to fit a line to this data which is as close as possible. We describe the Linear Regression model next and then apply it to this data.

1.3 3- Linear Regression

The idea behind *Linear Regression* is that we reduce the problem of estimating the response variable, $Y = \text{sales}$, by assuming there is a linear function of the predictor variables, $X_1 = \text{TV}$, $X_2 = \text{radio}$ and $X_3 = \text{newspaper}$ which describes Y . This reduces the problem to that of solving for the parameters $\beta_0, \beta_1, \beta_2$ and β_3 in the equation:

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon$$

where ϵ is an error term. After approximating the coefficients β_i as $\hat{\beta}_i$, we obtain an approximation, \hat{Y} of Y . The coefficients $\hat{\beta}_i$ are obtained using the observed realisations of the random variables X_i . Namely, $X_i = (x_{1i}, x_{2i}, x_{3i}, \dots, x_{ni})$ are n observations of X_i where $i = 1, 2, \dots, p$.

We first limit the problem to $p = 1$. For example, we are looking to estimate the coefficients in the equation

$$Y \approx \beta_0 + \beta_1 X_1 + \epsilon$$

using the n data points $(x_{11}, y_{11}), (x_{21}, y_{21}), \dots, (x_{n1}, y_{n1})$. We can define the prediction discrepancy of a particular prediction as the difference between the observed value and the predicted value. This is represented in mathematical notation for observation i as $y_i - \hat{y}_i$. Letting $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1$ we have $y_i - \hat{y}_i = \epsilon_i$. i.e. the error in the prediction of point observation i (also called the *ith residual*).

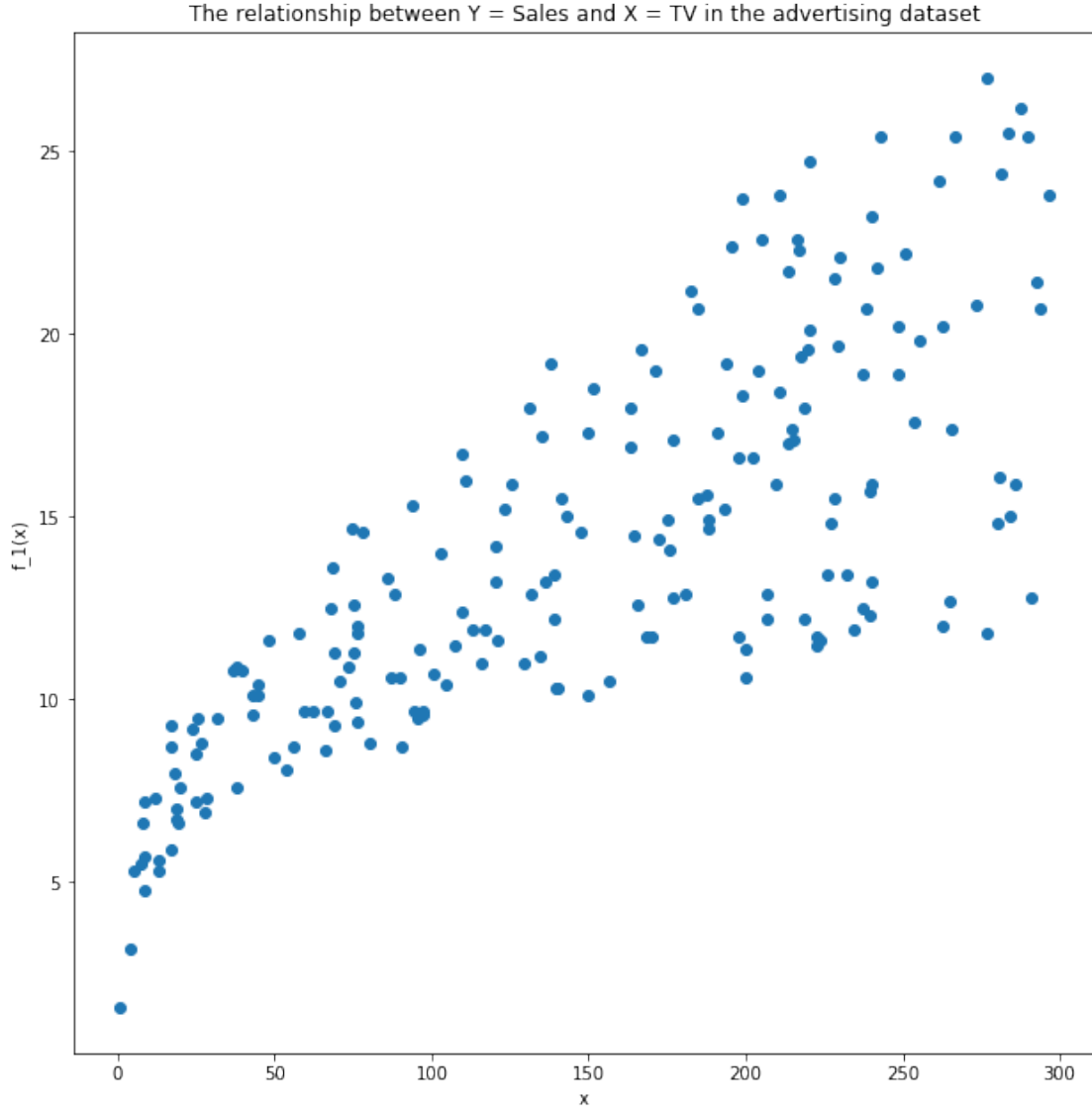
In summary, we are looking for a straight line to fit to the following data points as well as possible:

```
In [8]: # Get the figure handle and set figure size
fig = plt.figure(figsize=(8,8))

# Get the axis
axes = fig.add_axes([0.1,0.1,1,1])

# Plot onto the axis
axes.scatter(data=advert, x='TV', y='sales')

# Set the labels and title
axes.set_xlabel('x')
axes.set_ylabel('f_1(x)')
axes.set_title('The relationship between Y = Sales and X = TV in \
the advertising dataset')
plt.show()
```



In order to calculate appropriate values for parameters β_i , we would need a method of defining what it means for a line to be a good fit. A popular method is "Ordinary Least Squares". This method relies on minimising the Residual Sum of Squared errors (RSS). i.e. we are looking to minimise $RSS = \sum_{i=1}^n \epsilon_i^2$. While this intuitively makes sense, this can also be arrived at using a *Maximum Likelihood Estimation* (MLE) approach (see Appendix).

For the 1-parameter case we have that (the semi-colon below means 'the value of the parameters' given 'the data we have observed')

$$RSS(\hat{\beta}_0, \hat{\beta}_1; X) = \sum_{i=1}^n \epsilon_i^2 = \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

We would like to find the parameters (β_0, β_1) which minimise RSS. We first find the partial derivatives:

$$\frac{\partial RSS}{\partial \hat{\beta}_0} = -2[\sum_{i=1}^n y_i - \sum_{i=1}^n \hat{\beta}_0 - \sum_{i=1}^n \hat{\beta}_1 x_i]$$

$$\frac{\partial RSS}{\partial \hat{\beta}_1} = -2[\sum_{i=1}^n y_i x_i - \sum_{i=1}^n \hat{\beta}_0 x_i - \sum_{i=1}^n \hat{\beta}_1 x_i^2]$$

Then setting these to zero and solving

$$\frac{\partial RSS}{\partial \hat{\beta}_0} = 0 \implies \hat{\beta}_0 = \frac{\sum_{i=1}^n y_i - \hat{\beta}_1 \sum_{i=1}^n x_i}{n} = \frac{n\bar{y} - \hat{\beta}_1 n\bar{x}}{n} = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$\frac{\partial RSS}{\partial \hat{\beta}_1} = 0 \implies \sum_{i=1}^n y_i x_i - \hat{\beta}_0 \sum_{i=1}^n x_i - \hat{\beta}_1 \sum_{i=1}^n x_i^2 = 0$$

$$\implies \hat{\beta}_1 = \frac{n\bar{y}\bar{x} - \sum_{i=1}^n y_i x_i}{n\bar{x}^2 - \sum_{i=1}^n x_i^2} = \frac{\sum_{i=1}^n y_i x_i - n\bar{y}\bar{x}}{\sum_{i=1}^n x_i^2 - n\bar{x}^2} = \frac{\sum_{i=1}^n y_i x_i - n\bar{y}\bar{x} - n\bar{y}\bar{x} + n\bar{y}\bar{x}}{\sum_{i=1}^n x_i^2 - n\bar{x}^2 - n\bar{x}^2 + n\bar{x}^2}$$

$$= \frac{\sum_{i=1}^n x_i y_i - \sum_{i=1}^n y_i \bar{x} - \sum_{i=1}^n x_i \bar{y} + \sum_{i=1}^n \bar{y} \bar{x}}{\sum_{i=1}^n x_i^2 - \sum_{i=1}^n x_i \bar{x} - \sum_{i=1}^n x_i \bar{x} + \sum_{i=1}^n \bar{x}^2}$$

Where, in the penultimate line we completed the square and in the last equality we used $n\bar{y}\bar{x} = \sum_{i=1}^n y_i \bar{x} = \sum_{i=1}^n x_i \bar{y}$ and $n\bar{x}^2 = n\bar{x}\bar{x} = \sum_{i=1}^n x_i \bar{x}$. Factorising

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Additionally, the point (\bar{x}, \bar{y}) lies on this line where $\bar{x} = \sum_{i=1}^n x_i$ and $\bar{y} = \sum_{i=1}^n y_i$ (see Appendix).

We have now found the values of $(\hat{\beta}_0, \hat{\beta}_1)$ which corresponds to the extrema of RSS. We will still need to show that this is indeed a minima.

From Calculus, we know that if $\frac{\partial^2 RSS}{\partial \hat{\beta}_0^2} \frac{\partial^2 RSS}{\partial \hat{\beta}_1^2} - (\frac{\partial^2 RSS}{\partial \hat{\beta}_0 \partial \hat{\beta}_1})^2 > 0$, this is an extrema and not an inflexion point. Additionally, if $\frac{\partial^2 RSS}{\partial \hat{\beta}_0^2} > 0$ and $\frac{\partial^2 RSS}{\partial \hat{\beta}_1^2} > 0$ this is a minima.

We have that

$$\frac{\partial^2 RSS}{\partial \hat{\beta}_0^2} = 2n > 0$$

$$\frac{\partial^2 RSS}{\partial \hat{\beta}_1^2} = 2 \sum_{i=1}^n x_i^2 > 0$$

$$\frac{\partial^2 RSS}{\partial \hat{\beta}_0 \partial \hat{\beta}_1} = 2 \sum_{i=1}^n x_i$$

So,

$$\frac{\partial^2 RSS}{\partial \hat{\beta}_0^2} \frac{\partial^2 RSS}{\partial \hat{\beta}_1^2} - (\frac{\partial^2 RSS}{\partial \hat{\beta}_0 \partial \hat{\beta}_1})^2 = (2n)(2 \sum_{i=1}^n x_i^2) - (2 \sum_{i=1}^n x_i)^2 > 0 \forall n > 1 \text{ (see Appendix).}$$

This means that this is indeed a minima (since we have satisfied the conditions stated above).

The equation

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1$$

then defines a straight line of best fit which minimises the expected value of the errors (residuals). From the form of this line, we can see that $\hat{\beta}_0$ corresponds to the value of \hat{Y} if the independent variable X_1 is zero. $\hat{\beta}_1$ is then the gradient.

In the following we construct 3 functions dependent on a single independent variable and attach an error term and calculate the best fit. The three functions are chosen as:

- 1- $f_1(x) = 4.67 + 5.07 * x$
- 2- $f_2(x) = 4.67 + 5.07 * x^2$
- 3- $f_3(x) = 4.67 + 5.07 * \sin(x)$

```
In [8]: #f_1(x)=4.67+5.07x
def f_1(x):
    return 4.67 + 5.07*x

#f_2(x)=4.67+5.07x^2
def f_2(x):
    return 4.67 + 5.07*x**2

#f_3(x)=4.67+5.07sin(x/20)
def f_3(x):
    return 4.67 + 5.07*math.sin(x/20)

In [9]: # Set the seed
r = np.random.RandomState(101)

# Choose 1000 random observations for x between 0 and 100
X = 100*r.rand(1000)

#Error term with sigma = 10, mu = 0
E_1 = 10*r.randn(1000)

#Error term with sigma = 500, mu = 0
E_2 = 500*r.randn(1000)

#Error term with sigma = 19, mu = 0
E_3 = 1*r.randn(1000)

#Response variables
Y_1 = list(map(f_1,X))+E_1
Y_2 = list(map(f_2,X))+E_2
Y_3 = list(map(f_3,X))+E_3
```

First we look at what f_1 looks like

```
In [10]: # Plot
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.plot(X,Y_1,'. ')

# Set labels and title
```

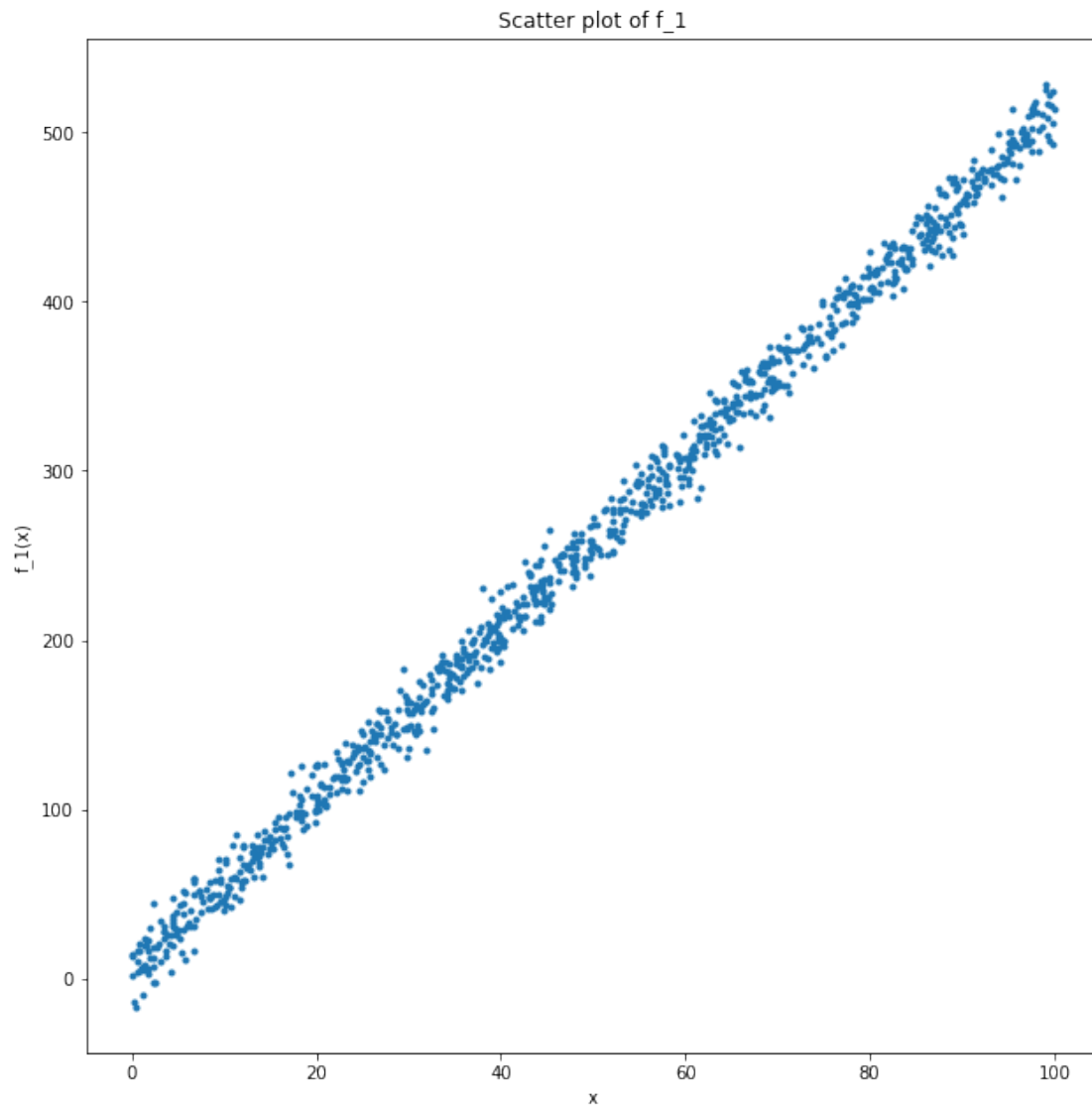


```

axes.set_xlabel('x')
axes.set_ylabel('f_1(x)')
axes.set_title('Scatter plot of f_1')

plt.show()

```



The task is to fit the model $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1$ to the data. We know that

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

and

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

```

In [11]: #Find the mean of the data for f_1
x_bar1 = np.mean(X)
y_bar1 = np.mean(Y_1)

numerator = 0
denominator = 0

for i in range(len(Y_1)):
    # Add to the numerator for beta_1
    numerator += (X[i] - x_bar1)*(Y_1[i] - y_bar1)

    # Add to the denominator for beta_1
    denominator += (X[i] - x_bar1)**2

beta1_1 = numerator/denominator
beta1_0 = y_bar1 - beta1_1*x_bar1

print('Y = {beta_0} + {beta_1} * X'.\
      format(beta_0 = beta1_0, beta_1 = beta1_1))

Y = 5.50124312485292 + 5.064254524922961 * X

```

Below, we see how the line defined by the equation above fits the data for f_1

```

In [12]: # 1000 linearly spaced numbers
x1 = np.linspace(0,99,1000)

# The equation using the betas above
y1 = beta1_0 + beta1_1 * x1

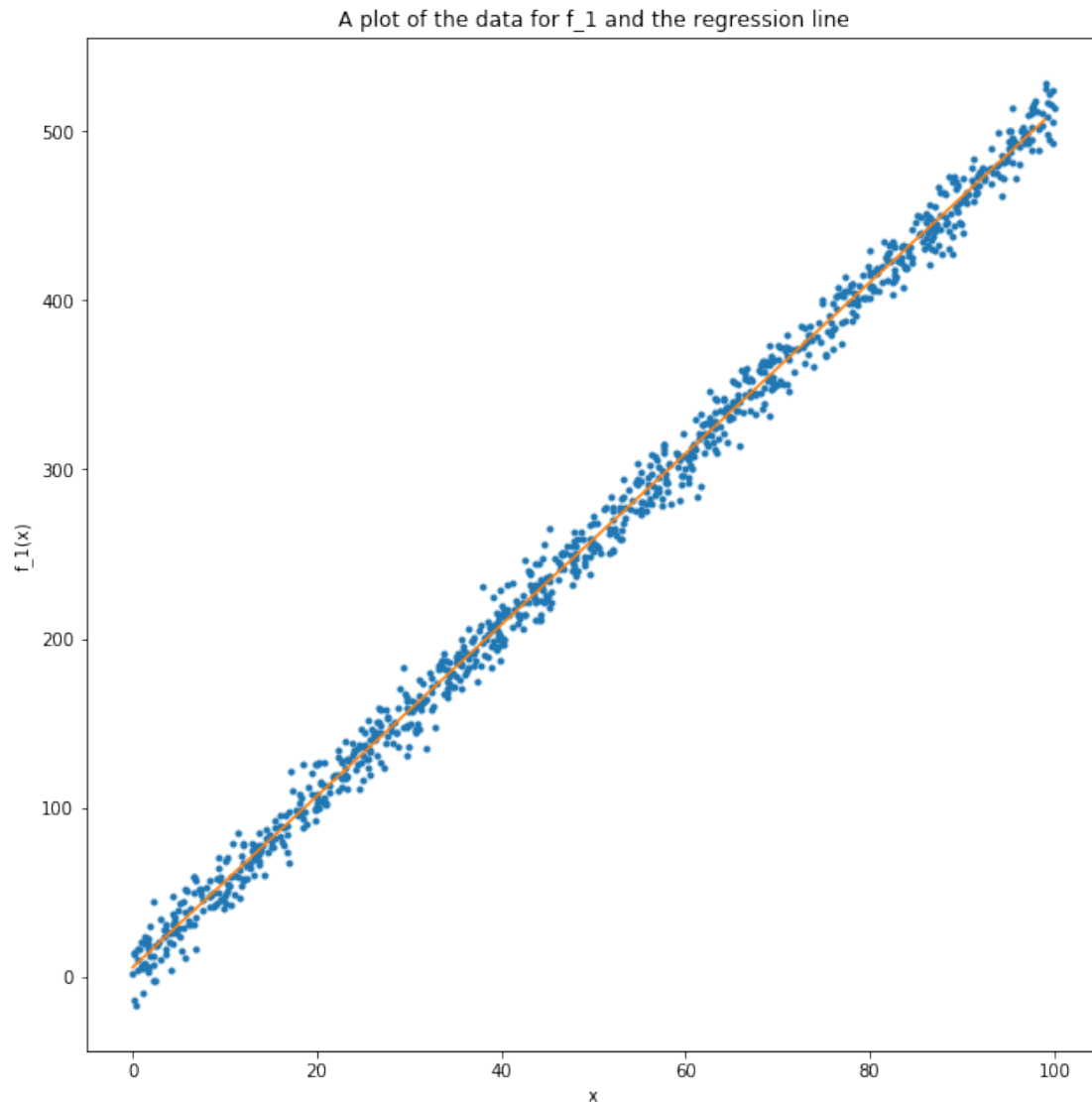
# Plot the observed data
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.plot(X,Y_1, '. ')

# Plot the regression line
axes.plot(x1,y1)

# Set labels and title
axes.set_xlabel('x')
axes.set_ylabel('f_1(x)')
axes.set_title('A plot of the data for f_1 and the regression line')

plt.show()

```



Let's see what the residuals look like by plotting them. The residual require the knowledge of the actual response variables. So we use the regression line above to predict the response variable using the observed predictor variables. Then we plot them using a histogram to gain some insight into their distribution

```
In [13]: # The fitted values are the predicted values given the observed values
         y1_fitted = beta1_0 + beta1_1 * X

         # The residuals are the differences between our predicted values and
         # the observed responses
         Res_1 = y1_fitted - Y_1

         # Plot the residuals
         fig = plt.figure(figsize=(8,8))
```

```

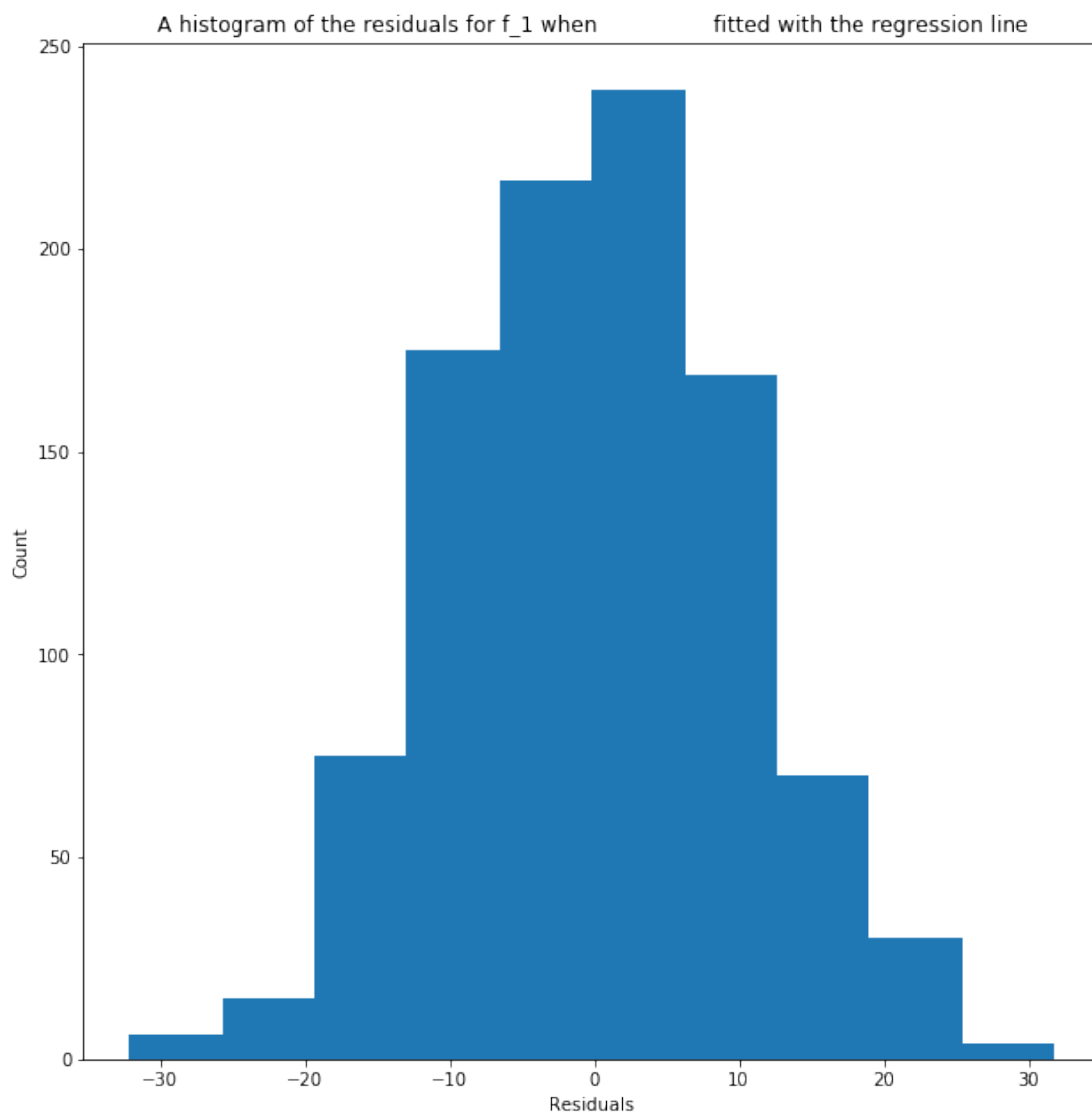
axes = fig.add_axes([0.1,0.1,1,1])
axes.hist(Res_1)

# Set labels and title
axes.set_xlabel('Residuals')
axes.set_ylabel('Count')
axes.set_title('A histogram of the residuals for f_1 when \
               fitted with the regression line')

plt.show()

print('This is roughly a normal distribution with mean {mean} \n\
and standard deviation {std}'.format(mean=np.mean(Res_1),std=np.std(Res_1)))

```



This is roughly a normal distribution with mean $-1.2157386208855315e-14$ and standard deviation 10.08588495757817

Since the residuals are roughly normally distributed, our model may be a good choice. In fact, the standard deviation for the residuals was roughly equal to the standard deviation for the error term when we constructed the function f_1 . A model may suffer from two types of error: error due to a discrepancy between the chosen function shape (here a linear model) and the true function shape (reducible); error due to random noise (irreducible). We can see here that the residuals are from irreducible error. Now let's do the same for f_2 .

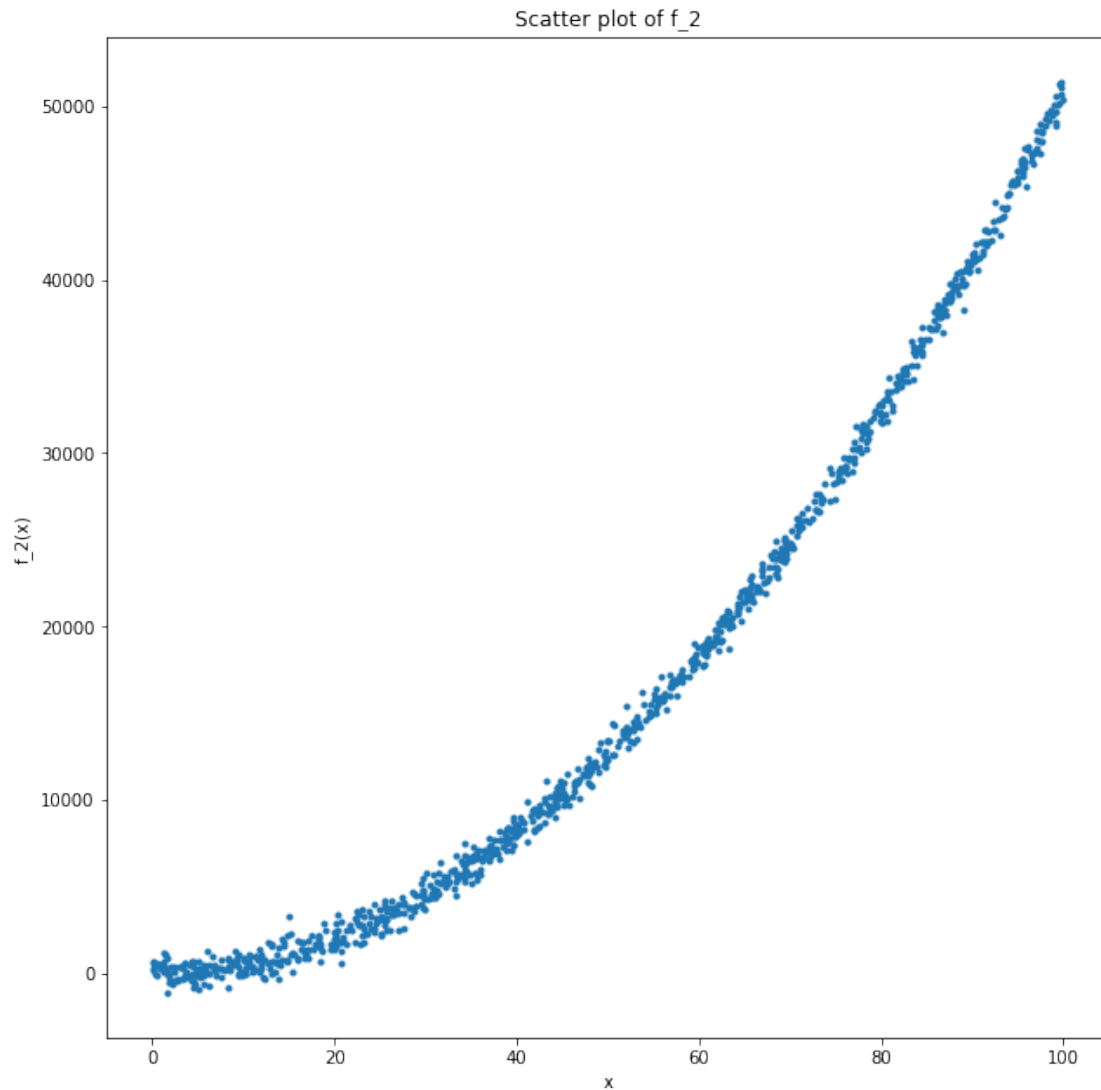
```
In [14]: # Get figure handle
fig = plt.figure(figsize=(8,8))

# Get axis handle and specify size
axes = fig.add_axes([0.1,0.1,1,1])

# Plot onto this axis
axes.plot(X,Y_2, '.')

# Set the axis labels
axes.set_xlabel('x')
axes.set_ylabel('f_2(x)')
axes.set_title('Scatter plot of f_2')

Out[14]: Text(0.5,1,'Scatter plot of f_2')
```



```
In [15]: #Find the mean of the data for  $f_2$ 
x_bar2 = np.mean(X)
y_bar2 = np.mean(Y_2)

numerator = 0
denominator = 0

for i in range(len(Y_2)):
    # Add to the numerator for  $\beta_1$ 
    numerator += (X[i] - x_bar2)*(Y_2[i] - y_bar2)

    # Add to the denominator for  $\beta_1$ 
    denominator += (X[i] - x_bar2)**2
```

```

beta2_1 = numerator/denominator
beta2_0 = y_bar2 - beta2_1*x_bar2

print('Y = {beta_0} + {beta_1} * X'.format(beta_0 = beta2_0, beta_1 = beta2_1))

```

Y = -8445.98030682202 + 506.16066894401735 * X

Below, we see how the line defined by the equation above fits the data for f_2

```

In [16]: # 1000 linearly spaced numbers
x2 = np.linspace(0,99,1000)

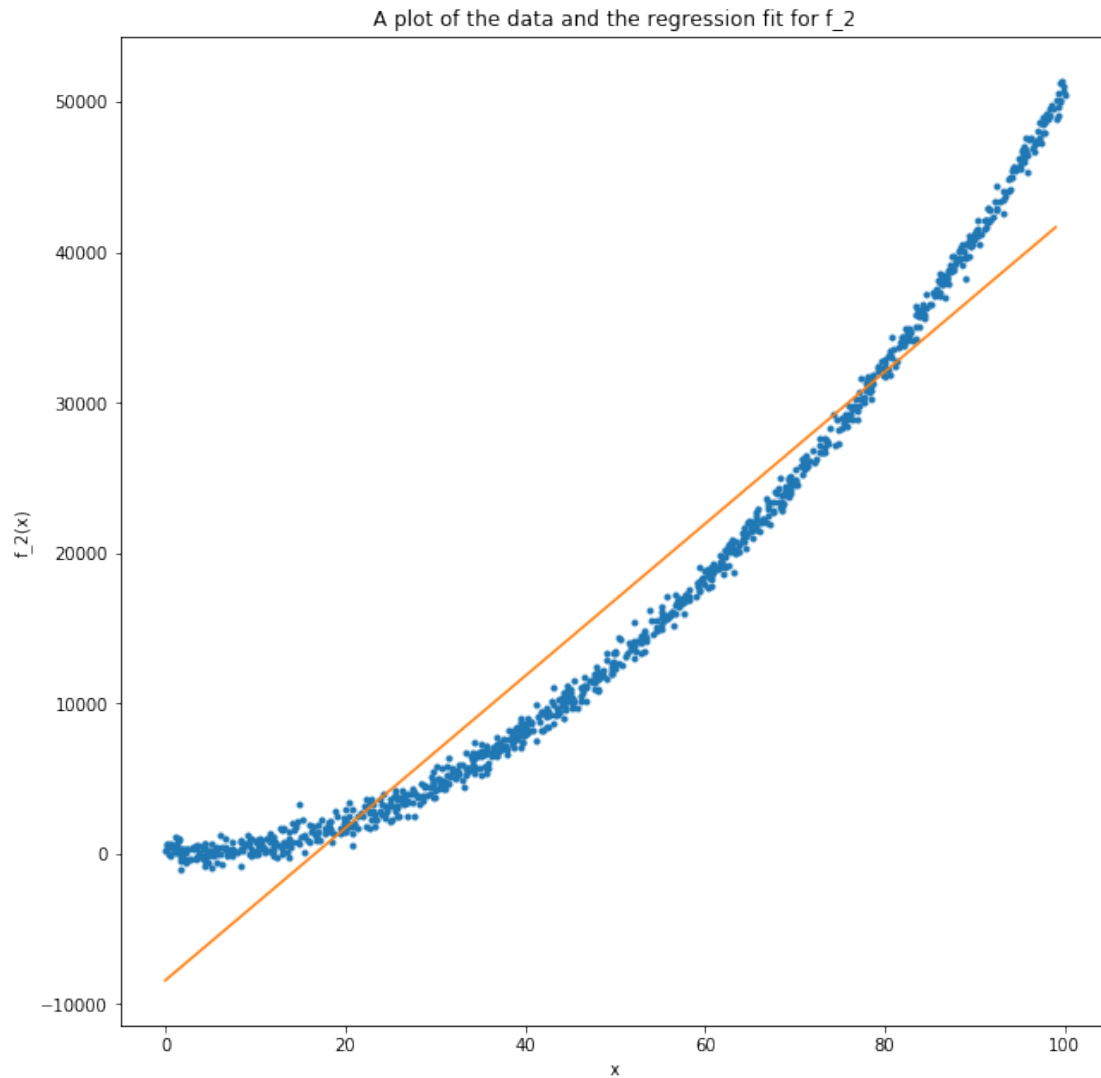
# The predicted responses of these 1000 numbers
y2 = beta2_0 + beta2_1 * x2

# Plot
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.plot(X,Y_2, '.')
axes.plot(x2,y2)

# Set labels and title
axes.set_xlabel('x')
axes.set_ylabel('f_2(x)')
axes.set_title('A plot of the data and the regression fit for f_2')

plt.show()

```



```
In [17]: # The fitted values are the predicted values given the observed values
y2_fitted = beta2_0 + beta2_1 * X

# The residuals are the differences between our predicted values and
# the observed responses
Res_2 = y2_fitted - Y_2

# Plot the residuals
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.hist(Res_2)

# Set labels and title
```



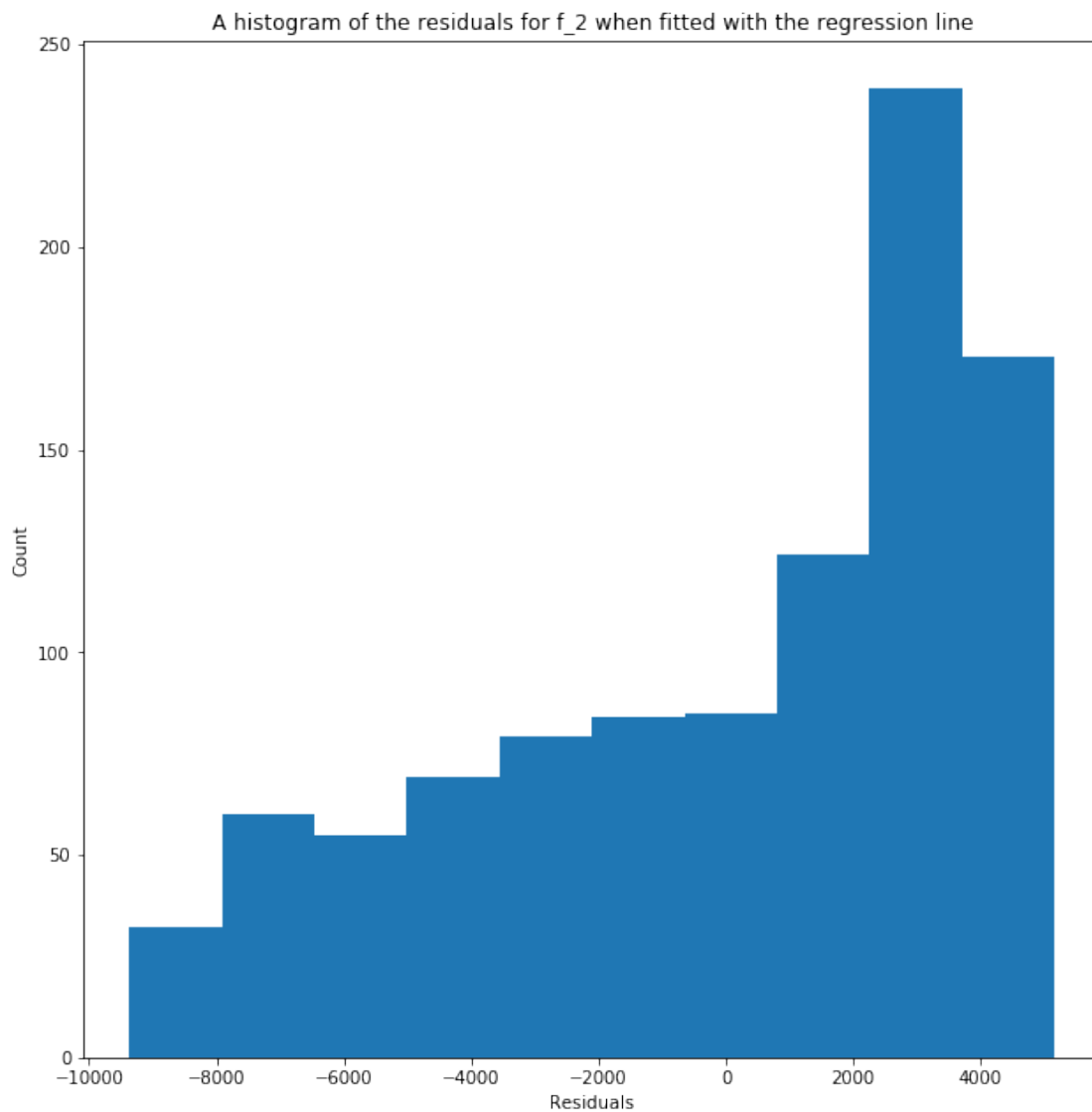
```

axes.set_xlabel('Residuals')
axes.set_ylabel('Count')
axes.set_title('A histogram of the residuals for f_2 when fitted with the regression line')

plt.show()

print('The residuals are certainly not from a normal distribution')

```



The residuals are certainly not from a normal distribution

So let's try X^2 as a parameter instead of X in our linear model

```

In [18]: # Create  $X^2$  parameter
X_2 = X**2

#Find the mean of the data for  $f_2$ 
x_bar22 = np.mean(X_2)
y_bar22 = np.mean(Y_2)

numerator = 0
denominator = 0

for i in range(len(Y_2)):
    # Calculate the numerator for  $\beta_1$ 
    numerator += (X_2[i] - x_bar22)*(Y_2[i] - y_bar22)

    # Calculate the denominator for  $\beta_1$ 
    denominator += (X_2[i] - x_bar22)**2

beta22_1 = numerator/denominator
beta22_0 = y_bar22 - beta22_1*x_bar22

print('Y = {beta_0} + {beta_1} * X^2'.format(beta_0 = beta22_0, beta_1 = beta22_1))

Y = 14.470063153316005 + 5.075020979320466 * X^2

```

Below, we see how the new line defined by the equation above fits the data for f_2

```

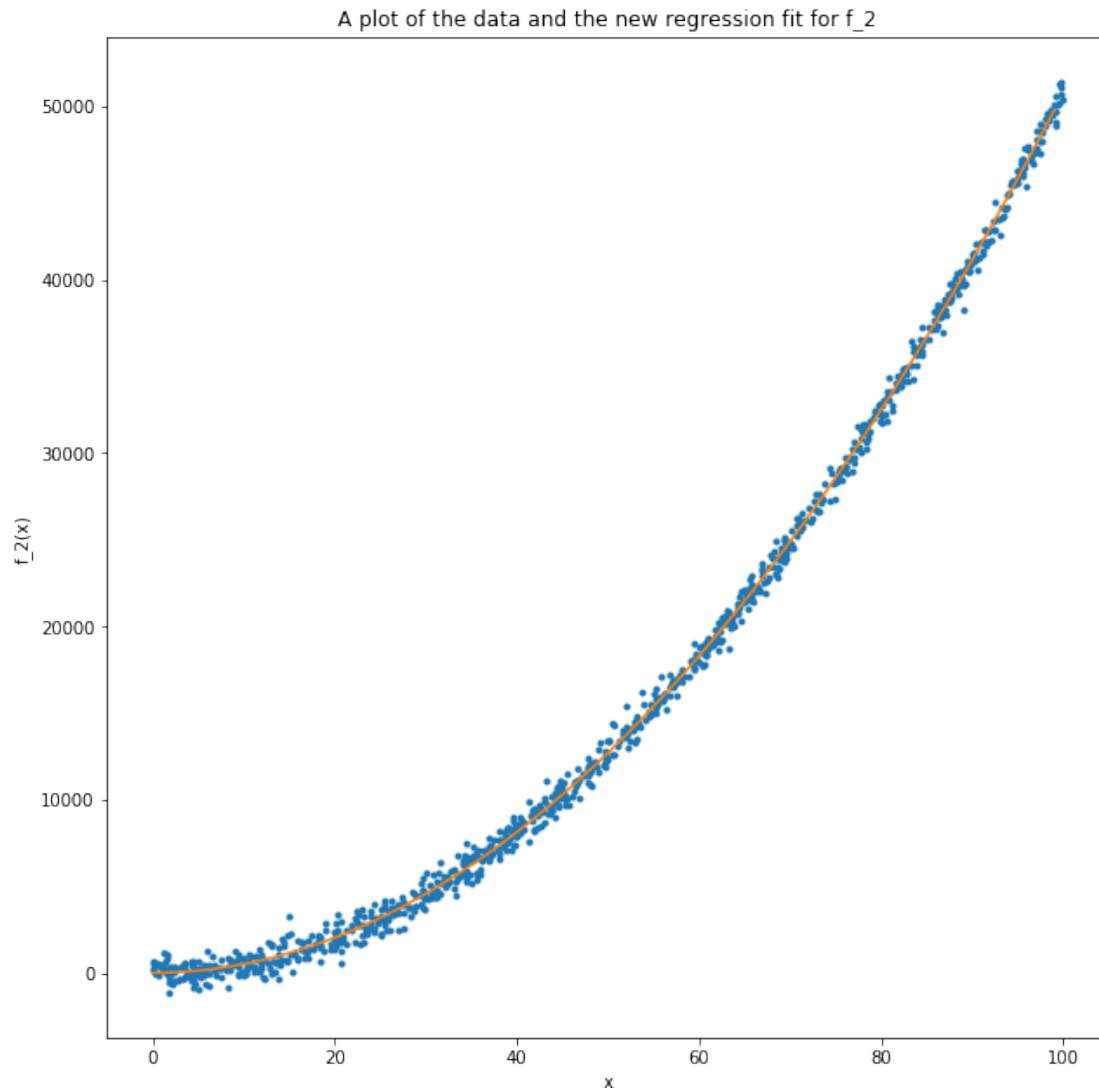
In [19]: # 1000 linearly spaced numbers
x22 = np.linspace(0,99,1000)

# Predicted responses to the 1000 numbers
y22 = beta22_0 + beta22_1 * ((x22)**2)

# Plot this regression line and the data
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.plot(X,Y_2, '.')
axes.plot(x22,y22)

# Set labels and title
axes.set_xlabel('x')
axes.set_ylabel('f_2(x)')
axes.set_title('A plot of the data and the new regression fit for  $f_2$ ')
plt.show()

```



Now we investigate the residuals to see if the new regression fit using X^2 as a parameter yields residuals that look more normally distributed

```
In [20]: # The fitted values are the predicted values given the observed values
y22_fitted = beta22_0 + beta22_1 * X**2

# The residuals are the differences between our predicted values and
# the observed responses
Res_22 = y22_fitted - Y_2

# Plot the residuals
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.hist(Res_22)
```

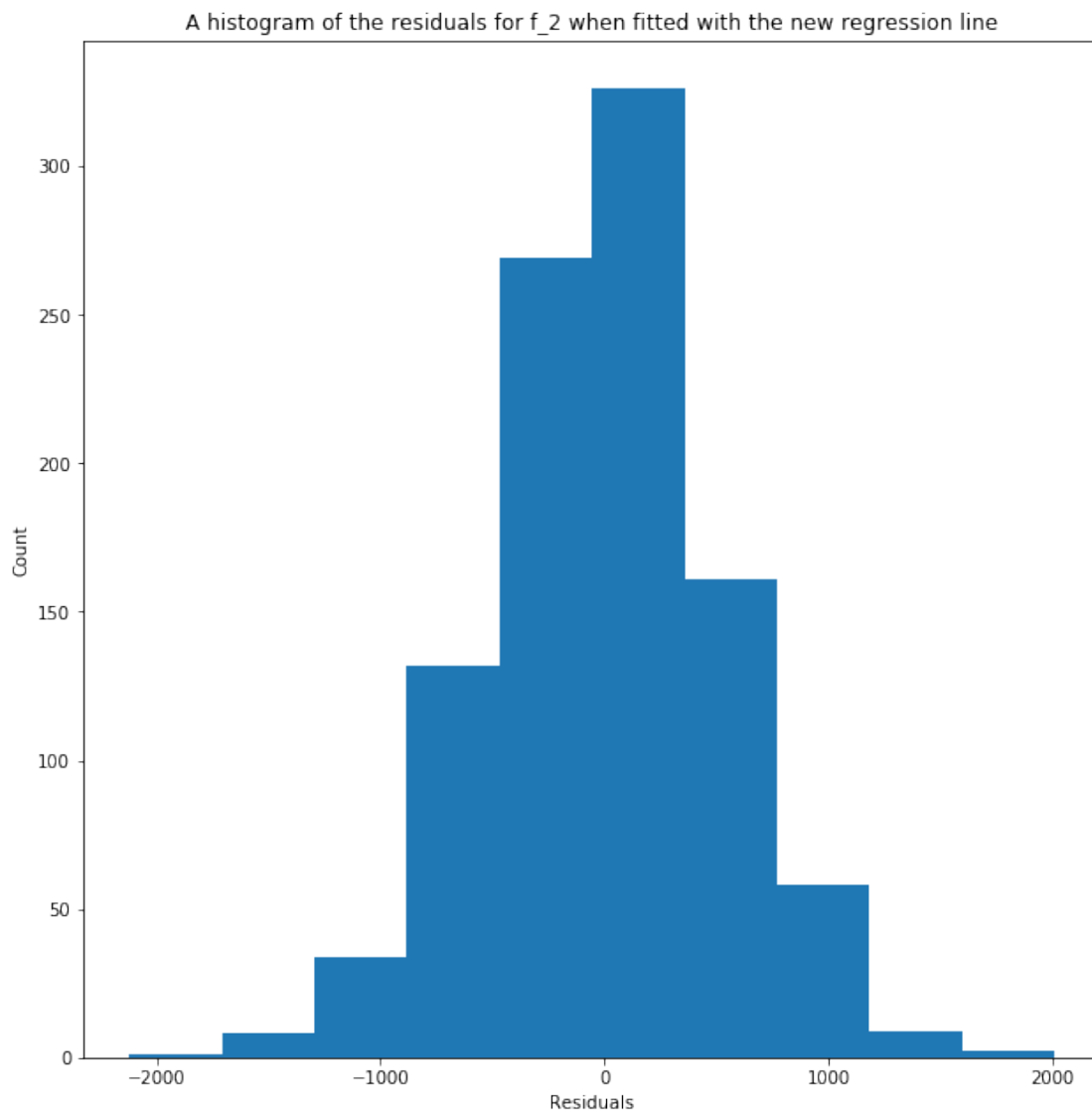
```

# Set labels and title
axes.set_xlabel('Residuals')
axes.set_ylabel('Count')
axes.set_title('A histogram of the residuals for f_2 when fitted with the new regress

plt.show()

print('This is roughly a normal distribution with mean {mean} and standard deviation {std}.'.format(mean=np.mean(Res_22),std=np.std(Res_22)))

```



This is roughly a normal distribution with mean $-1.1250449460931123 \times 10^{-12}$ and standard deviation

This shows that we can transform an independent variable and apply linear regression in order to regress the response variable onto the transformed Explanatory variable. This increases the power of linear regression techniques. Note also that the standard deviation from the residual distribution is close to the 500 for the errors when the function was created.

Now let's apply linear regression to f_3 in a similar manner

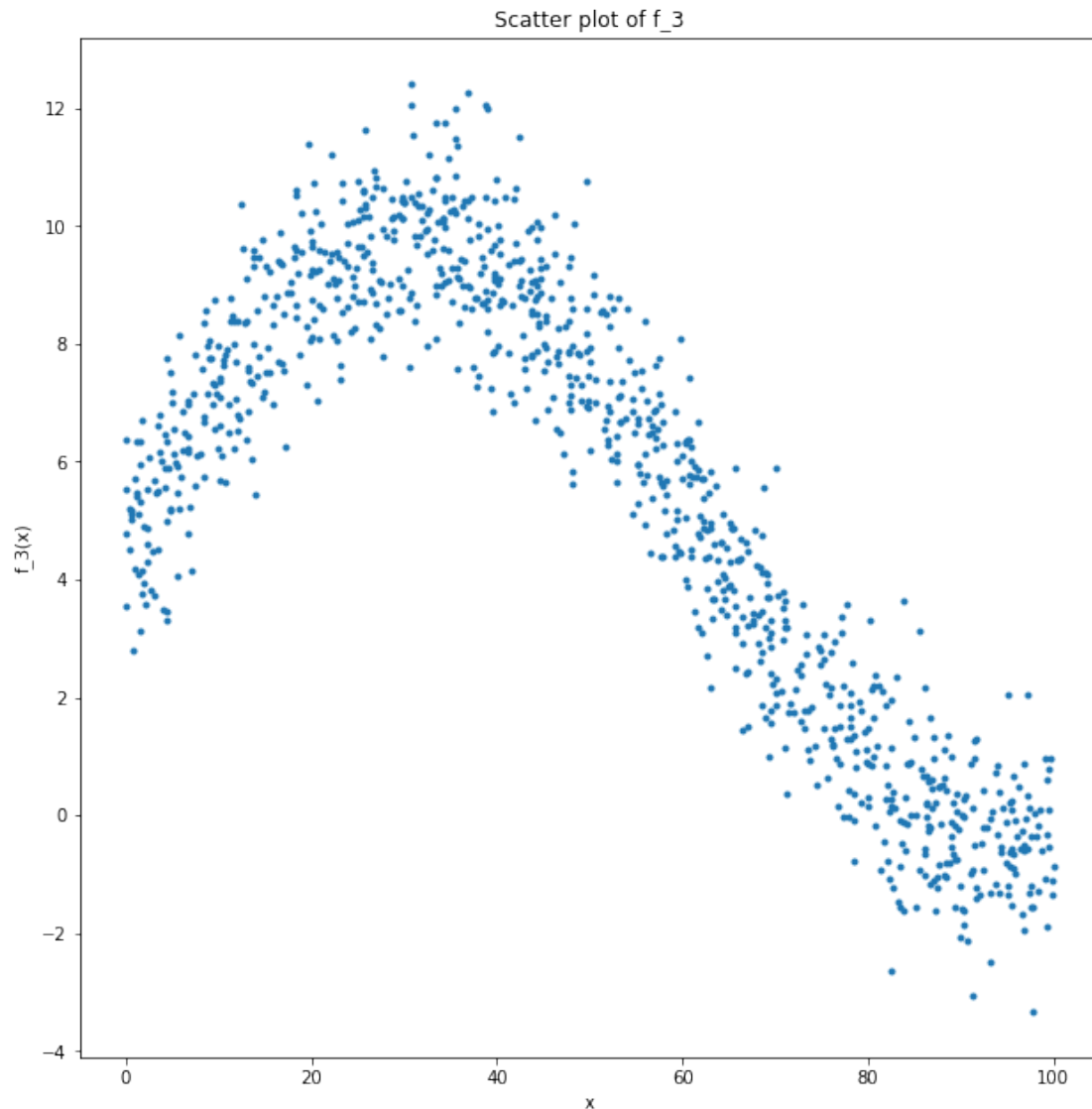
```
In [21]: # Get figure handle
fig = plt.figure(figsize=(8,8))

# Get axis handle and specify size
axes = fig.add_axes([0.1,0.1,1,1])

# Plot onto this axis
axes.plot(X,Y_3, '.')

# Set the axis labels
axes.set_xlabel('x')
axes.set_ylabel('f_3(x)')
axes.set_title('Scatter plot of f_3')

plt.show()
```



It is very clear from the above scatter plot that we will not be able to get away with fitting a linear line to the data. This is a hint that we should use transformed variables. But let's carry out a linear fit to show that the results can be misleading when we only consider the residuals plot to assess the quality of fit

```
In [22]: #Find the mean of the data for f_3
x_bar3 = np.mean(X)
y_bar3 = np.mean(Y_3)

numerator = 0
denominator = 0

for i in range(len(Y_3)):
    numerator += (X[i] - x_bar3)*(Y_3[i] - y_bar3)
```

```

        denominator += (X[i] - x_bar3)**2

    beta3_1 = numerator/denominator
    beta3_0 = y_bar3 - beta3_1*x_bar3

    print('Y = {beta_0} + {beta_1} * X'.format(beta_0 = beta3_0, beta_1 = beta3_1))

Y = 10.511143457700811 + -0.1011987818100197 * X

```

Below, we see how the line defined by the equation above fits the data for f_3

```

In [23]: # 1000 linearly spaced numbers
x3 = np.linspace(0,99,1000)

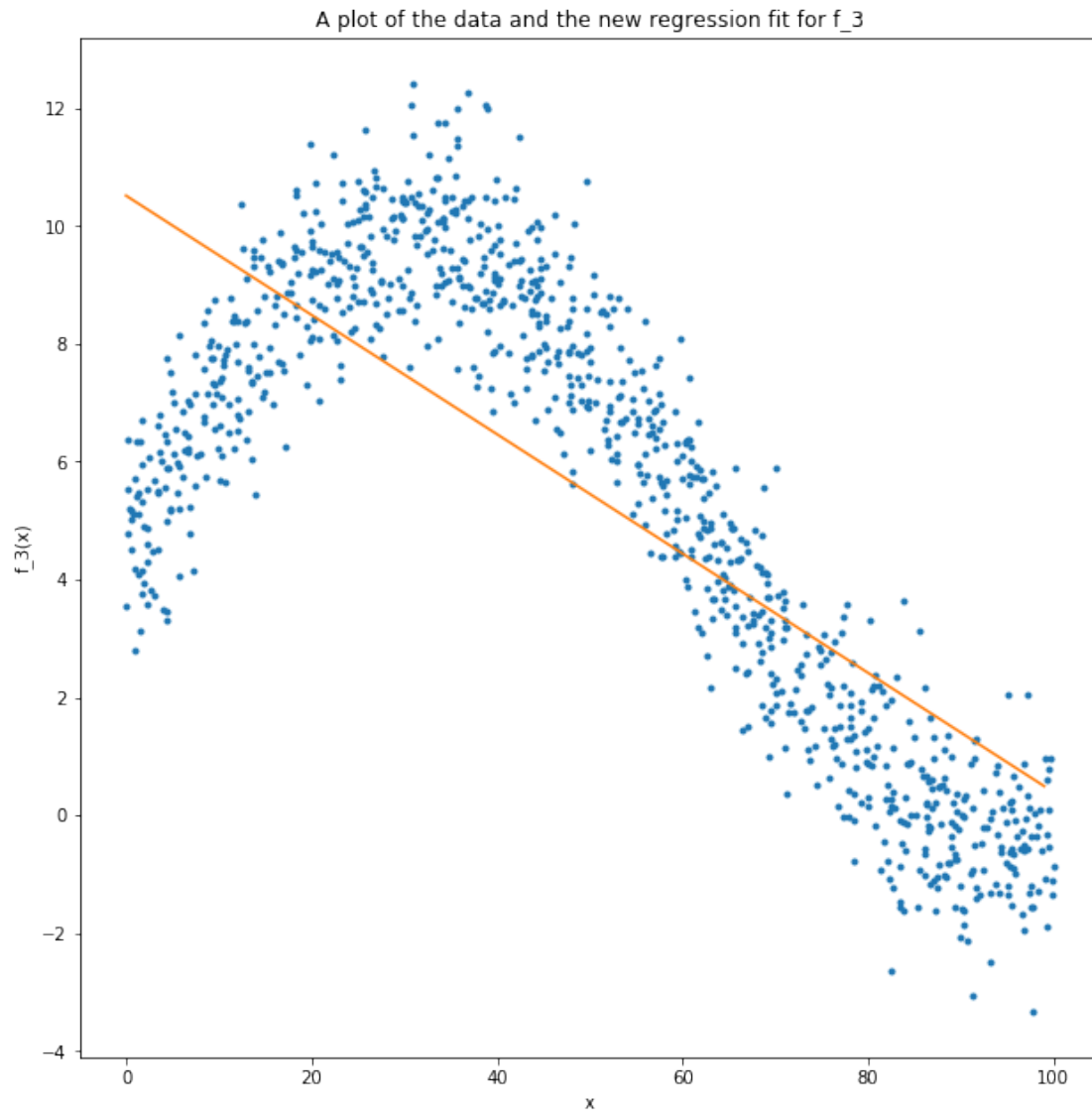
# Predict the response for those numbers
y3 = beta3_0 + beta3_1 * x3

# Plot both the data and the fit
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.plot(X,Y_3, '.')
axes.plot(x3,y3)

# Set the labels and title
axes.set_xlabel('x')
axes.set_ylabel('f_3(x)')
axes.set_title('A plot of the data and the new regression fit for f_3')

plt.show()

```



We now assess the residuals

```
In [24]: # The fitted values are the predicted values given the observed values
y3_fitted = beta3_0 + beta3_1 * X

# The residuals are the differences between our predicted values and
# the observed responses
Res_3 = y3_fitted - Y_3

# Plot the residuals
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.hist(Res_3)
```



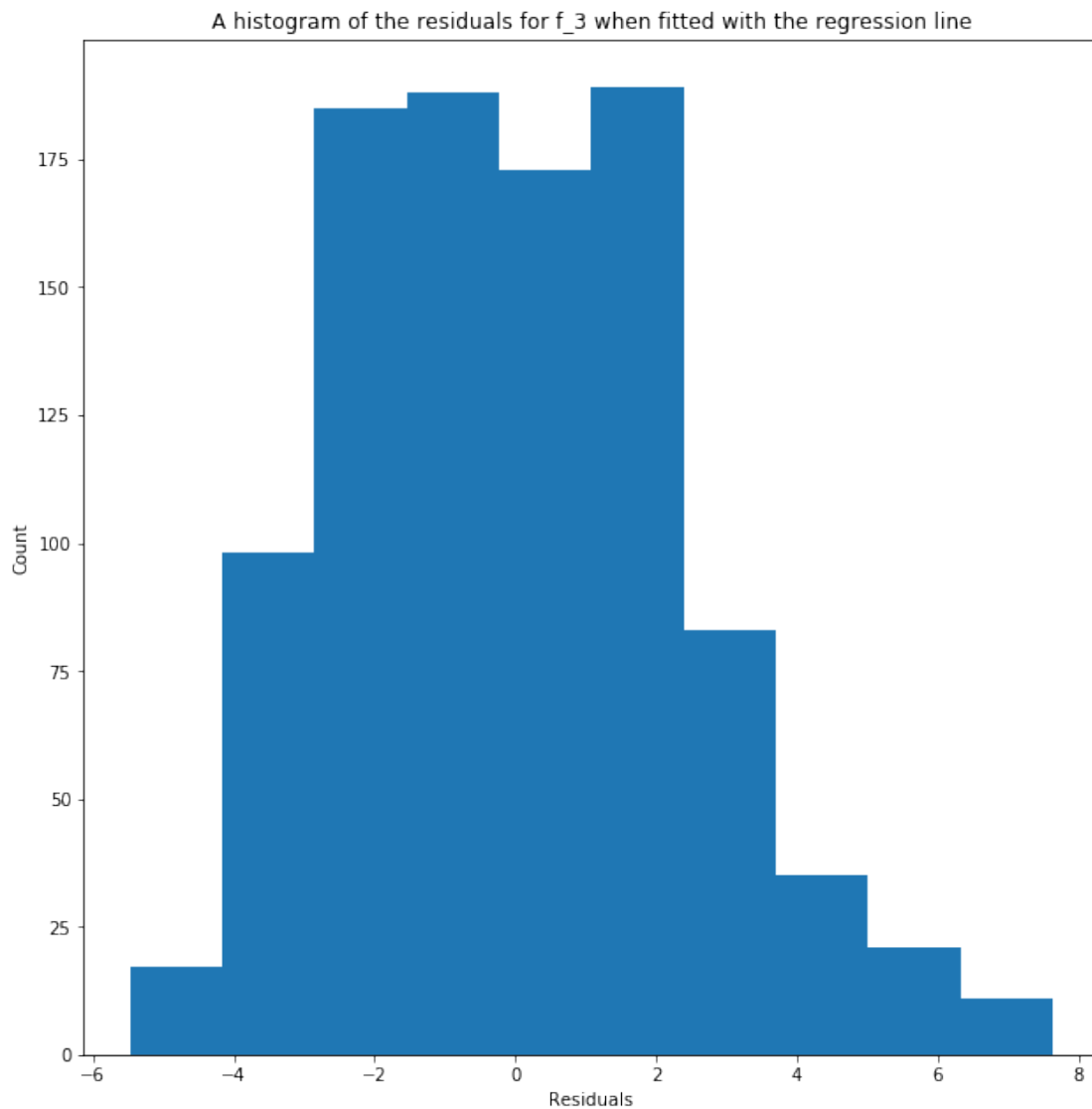
```

# Set labels and title
axes.set_xlabel('Residuals')
axes.set_ylabel('Count')
axes.set_title('A histogram of the residuals for f_3 when fitted with the regression line')

plt.show()

print('This not a normal distribution but it is not that far off.')

```



This not a normal distribution but it is not that far off.

Even though a plot of the residuals does not show a clear divergence from a normal distribution, it is clear from the predicted-observed plot that this is not a good model and does not fit the

data in a satisfactory manner. We therefore need additional tools in order to assess the level of fit.

A metric we can use in order to assess the accuracy of the fit is the *R-Squared* (R^2) statistic. The R^2 statistic measures the percentage of variability of the response variable that is explained by the explanatory variable. This is mathematically expressed as:

$$R^2 = \frac{TSS - RSS}{TSS}$$

where $TSS = \sum_{i=1}^n (y_i - \bar{y})^2$ is the *total sum of squares* and $RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$ is the *residual sum of squares*.

STATEMENT: The *Residual Squared Error* $RSE = \sqrt{\frac{RSS}{n-2}}$ is a measure of lack of fit. R^2 , as the form above suggests, is the proportion of variance that is explained. For a simple linear regression with 1 parameter (see Appendix):

$$R^2 = \text{Cor}(X, Y)^2 = \left(\frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}} \right)^2$$

However, for multiple linear regression this does not hold. It is not clear how to adapt the Correlation in order to explain the fit of a multiple regression model. R^2 however, is a clearly defined metric which is easily extended to multiple regression.

Below, we calculate this metric for f_3

```
In [25]: # TSS
TSS_3 = 0

# RSS
RSS_3 = 0

for i in range(len(X)):
    TSS_3 += (Y_3[i] - y_bar3)**2
    RSS_3 += (Y_3[i] - y3_fitted[i])**2

# R^2 for f_3
R_sq_3 = (TSS_3 - RSS_3)/TSS_3
print('R^2 = {}'.format(R_sq_3))

R^2 = 0.5940625125965683
```

This means that roughly 59% of the variability in Y_3 is explained by X . Let's calculate the R^2 statistic for all the models above. To do this, we create a function that accepts observed and fitted values and returns the TSS and RSS of the fit

```
In [26]: def TSS_RSS(y_observed, y_fitted):
    """
    A function that calculates the TSS and RSS of a fit given observed
    and fitted values
    y_observed := Observed data as a series object
    y_fitted := Fitted data as a series object
```

```

output := A (TSS,RSS) tuple of floats
'''

# TSS
TSS = 0

# RSS
RSS = 0

# Get the mean of the observed values
y_bar = np.mean(y_observed)

for i in range(len(y_observed)):
    TSS += (y_observed[i] - y_bar)**2
    RSS += (y_observed[i] - y_fitted[i])**2

return TSS,RSS

```

```

In [27]: # Calculate the TSS and RSS for the fitted regression line to f_1
TSS_1, RSS_1 = TSS_RSS(Y_1,y1_fitted)

# Calculate the R^2 for the fit to f_1
R_sq_1 = (TSS_1 - RSS_1)/TSS_1
print('Model for Y_1: Explanatory variable X for Y_1 - R^2 = {}'.format(R_sq_1))

# Calculate the TSS and RSS for the fitted regression line to f_2
TSS_2,RSS_2 = TSS_RSS(Y_2,y2_fitted)

# Calculate the R^2 for the fit to f_2
R_sq_2 = (TSS_2 - RSS_2)/TSS_2
print('Model for Y_2: Explanatory variable X for Y_2 - R^2 = {}'.format(R_sq_2))

# Calculate the TSS and RSS for the new fitted regression line to f_2
TSS_22,RSS_22 = TSS_RSS(Y_2,y22_fitted)

# Calculate the R^2 for the new fit to f_2
R_sq_22 = (TSS_22 - RSS_22)/TSS_22
print('Model for Y_2: Explanatory variable X^2 for Y_2 - R^2 = {}'.format(R_sq_22))

# Calculate the TSS and RSS for the fitted regression line to f_3
TSS_3,RSS_3 = TSS_RSS(Y_3,y3_fitted)

```

```

# Calculate the  $R^2$  for the fit to  $f_3$ 
R_sq_3 = (TSS_3 - RSS_3)/TSS_3
print('Model for Y_3: Explanatory variable X for Y_3 -  $R^2$  = {}'.format(R_sq_3))

Model for Y_1: Explanatory variable X for Y_1 -  $R^2$  = 0.9951845734408926
Model for Y_2: Explanatory variable X for Y_2 -  $R^2$  = 0.9336613222418227
Model for Y_2: Explanatory variable  $X^2$  for Y_2 -  $R^2$  = 0.99880452106502
Model for Y_3: Explanatory variable X for Y_3 -  $R^2$  = 0.5940625125965683

```

From the above we can see that the model for Y_1 that is linear in X is satisfactory; The model for Y_2 that is non-linear explains more variability of the response variable than the linear model (note that in this case, the R^2 metric alone wouldn't tell us whether the fit linear in X was terrible. But along with the residual plot we would arrive at the correct conclusion); The model for Y_3 shows that we are probably not fitting the correct form of the function, i.e. we have introduced bias in that the real function is not of the form $a + bX$ for constants a and b and that applying a model non-linear in X may provide a boost to the explained variance. We can try combinations of X , X^2 , X^3 as well. We do this after we have introduced a much simpler way of obtaining the above fits using Scikit-Learn packages.

Below, we use `sklearn.linear_model.LinearRegression()` in order to fit and `sklearn.metrics.r2_score()` in order to calculate the R^2 statistic. We will see that the results match the manual results above

```

In [28]: # Import the linear model and the metric we'll be using
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Create the model object
lm1 = LinearRegression()

# Fit this model to the data for  $f_1$ 
lm1.fit(X.reshape(-1,1),Y_1.reshape(-1,1))

print('Model for Y_1: Explanatory variable X for Y_1')
print('beta_0 = {}'.format(lm1.intercept_[0]))
print('beta_1 = {}'.format(lm1.coef_[0][0]))

# Get the fitted values and print it
y1_fitted_sklearn = lm1.intercept_[0] + lm1.coef_[0][0]*X
print('R^2 = {}'.format(r2_score(Y_1,y1_fitted_sklearn)))

print()
print()

lm2 = LinearRegression()
lm2.fit(X.reshape(-1,1),Y_2.reshape(-1,1))
print('Model for Y_2: Explanatory variable X for Y_2')

```

```

print('beta_0 = {}'.format(lm2.intercept_[0]))
print('beta_1 = {}'.format(lm2.coef_[0][0]))
y2_fitted_sklearn = lm2.intercept_[0] + lm2.coef_[0][0]*X
print('R^2 = {}'.format(r2_score(Y_2,y2_fitted_sklearn)))

print()
print()

lm22 = LinearRegression()
lm22.fit((X**2).reshape(-1,1),Y_2.reshape(-1,1))
print('Model for Y_2: Explanatory variable X^2 for Y_2')
print('beta_0 = {}'.format(lm22.intercept_[0]))
print('beta_1 = {}'.format(lm22.coef_[0][0]))
y22_fitted_sklearn = lm22.intercept_[0] + lm22.coef_[0][0]*X**2
print('R^2 = {}'.format(r2_score(Y_2,y22_fitted_sklearn)))

print()
print()

lm3 = LinearRegression()
lm3.fit(X.reshape(-1,1),Y_3.reshape(-1,1))
print('Model for Y_3: Explanatory variable X for Y_3')
print('beta_0 = {}'.format(lm3.intercept_[0]))
print('beta_1 = {}'.format(lm3.coef_[0][0]))
y3_fitted_sklearn = lm3.intercept_[0] + lm3.coef_[0][0]*X
print('R^2 = {}'.format(r2_score(Y_3,y3_fitted_sklearn)))

print()
print()

# Now we try adding the variables X,X^2 and X^3

#Create transformed variables
X2 = X**2
X3 = X**3

lm32 = LinearRegression()
X3_collection = pd.concat([pd.DataFrame(X,columns=['X']),\
                           pd.DataFrame(X**2,columns=['X2']),\
                           pd.DataFrame(X**3,columns=['X3'])],axis=1)
lm32.fit(X3_collection,Y_3.reshape(-1,1))
print('Model for Y_3: Explanatory variables X,X^2,X^3 for Y_3')
print('beta_0 = {}'.format(lm32.intercept_[0]))
print('beta_1 = {}'.format(lm32.coef_[0][0]))
print('beta_2 = {}'.format(lm32.coef_[0][1]))
print('beta_3 = {}'.format(lm32.coef_[0][2]))
y32_fitted_sklearn = lm32.intercept_[0] + lm32.coef_[0][0]*X + \
                    lm32.coef_[0][1]*X**2 + lm32.coef_[0][2]*X**3

```

```
print('R^2 = {}'.format(r2_score(Y_3,y32_fitted_sklearn)))
```

Model for Y_1: Explanatory variable X for Y_1

```
beta_0 = 5.501243124853005
```

```
beta_1 = 5.064254524922959
```

```
R^2 = 0.9951845734408926
```

Model for Y_2: Explanatory variable X for Y_2

```
beta_0 = -8445.980306821977
```

```
beta_1 = 506.16066894401644
```

```
R^2 = 0.9336613222418227
```

Model for Y_2: Explanatory variable X^2 for Y_2

```
beta_0 = 14.470063153316005
```

```
beta_1 = 5.075020979320466
```

```
R^2 = 0.99880452106502
```

Model for Y_3: Explanatory variable X for Y_3

```
beta_0 = 10.511143457700808
```

```
beta_1 = -0.10119878181001966
```

```
R^2 = 0.5940625125965684
```

Model for Y_3: Explanatory variables X,X^2,X^3 for Y_3

```
beta_0 = 3.664431201636692
```

```
beta_1 = 0.48709842203796394
```

```
beta_2 = -0.011179330358454434
```

```
beta_3 = 5.867605764948042e-05
```

```
R^2 = 0.9229011520420615
```

In the above, we fit a model using 3 explanatory variables, namely X , X^2 , X^3 with coefficients β_1 , β_2 , β_3 respectively. We can see that we have a much improved R^2 statistic for the fitted model to f_3 meaning we have managed to explain much more of the data using the transformed variables we have created. We can plot the model to see how well it follows the response variable.

```
In [29]: # 1000 linearly spaced numbers
```

```
x32 = np.linspace(0,99,1000)
```

```
y32 = lm32.intercept_[0] + lm32.coef_[0][0]*x32 + lm32.coef_[0][1]*x32**2\
      + lm32.coef_[0][2]*x32**3
```

```
# Plot the data and the fit
```

```
fig = plt.figure(figsize=(8,8))
```

```
axes = fig.add_axes([0.1,0.1,1,1])
```

```
axes.plot(X,Y_3,'.')
```

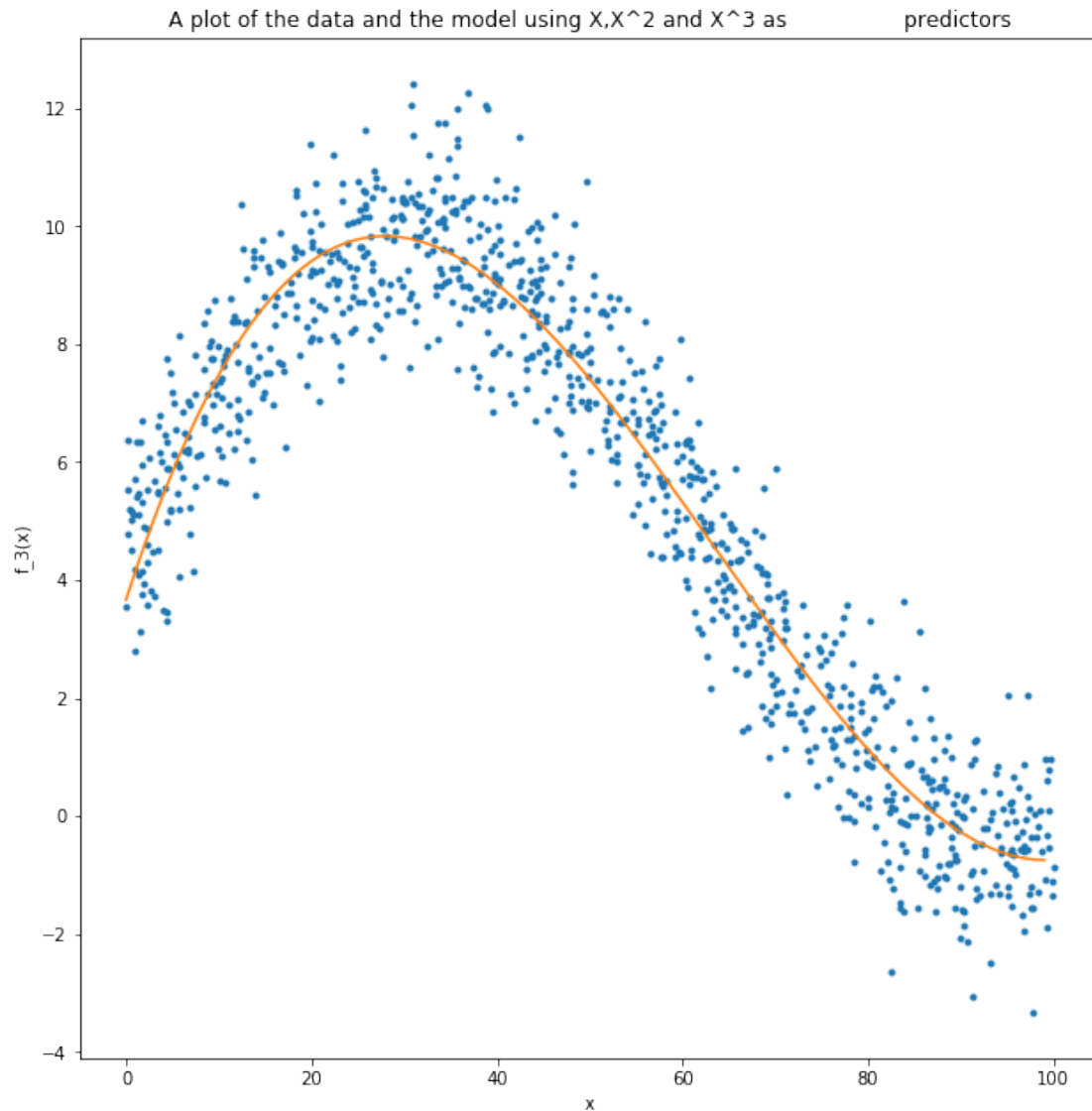
```

axes.plot(x32,y32)

# Set the labels and title
axes.set_xlabel('x')
axes.set_ylabel('f_3(x)')
axes.set_title('A plot of the data and the model using X,X^2 and X^3 as \
predictors')

plt.show()

```



We can also check the residuals plot

```

In [30]: # Calculate the fitted values using the observed values
y32_fitted_sklearn = lm32.intercept_[0] + lm32.coef_[0][0]*X + \

```

```

lm32.coef_[0][1]*X**2 + lm32.coef_[0][2]*X**3

# Calculate the residuals
Res_32 = y32_fitted_sklearn - Y_3

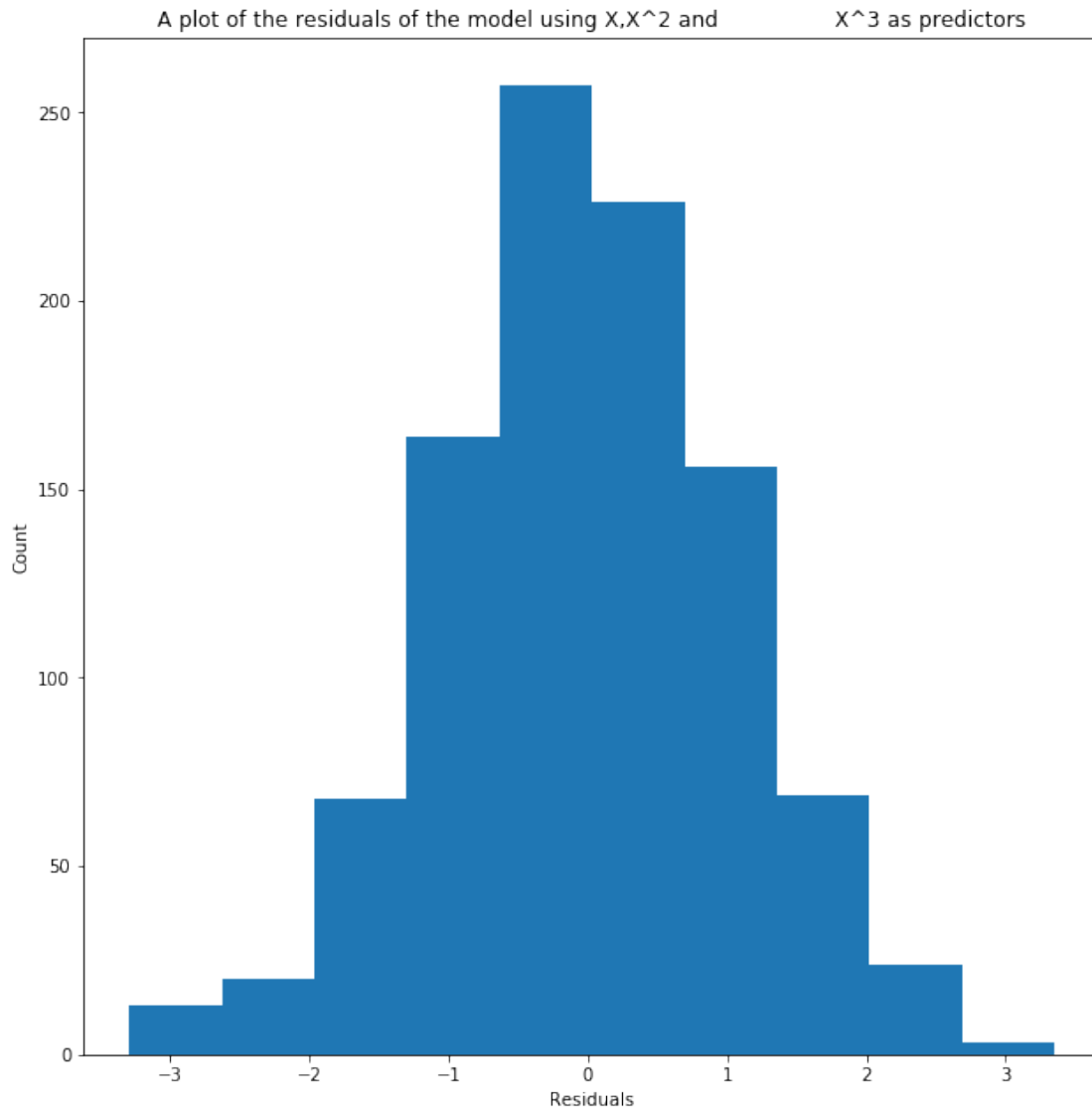
# Plot the residuals
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.hist(Res_32)

# Set the lables and title
axes.set_xlabel('Residuals')
axes.set_ylabel('Count')
axes.set_title('A plot of the residuals of the model using X,X^2 and \
               X^3 as predictors')

plt.show()

print('This is roughly a normal distribution with mean {mean} and\n \
      standard deviation {std}'.format(mean=np.mean(Res_32),std=np.std(Res_32)))

```

This is roughly a normal distribution with mean $-1.7408297026122454e-15$ and standard deviation 1.043797076853439

It is not a surprise that we were able to fit a function of the form $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$. Using Taylor expansion, $f(x) = \sin(x)$ estimated around the point $x = 0$ as

$$\begin{aligned}
 f(x) &= f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 + O(x^4) \\
 &= \sin(0) + \cos(0)x - \frac{\sin(0)}{2!}x^2 - \frac{\cos(0)}{3!}x^3 + O(x^4) \\
 &= x - \frac{x^3}{6} + O(x^4)
 \end{aligned}$$

If we apply Taylor series expansion to $f(x) = 4.67 + 5.07\sin(x/20)$ instead:

$$f(x=0) = 4.67 + \frac{5.07}{20} \cos(0)x - \frac{5.07}{20^3} \cos(0)x^3/(3!) = 4.67 + 0.25x - 1 \times 10^{-4}x^3$$

Let's plot this along with the above for smaller values of X for which this approximation of sin(x) is acceptable.

```
In [31]: # 1000 linearly spaced numbers
x32 = np.linspace(0,50,1000)

# Predictions
y32 = lm32.intercept_[0] + lm32.coef_[0][0]*x32 + lm32.coef_[0][1]*x32**2\
      + lm32.coef_[0][2]*x32**3

# Prediction using Taylor expansion
y_taylor_32 = 4.67 + (5.07/20)*x32 + 0*x32**2 - (5.07/(20**3 * 6))*x32**3

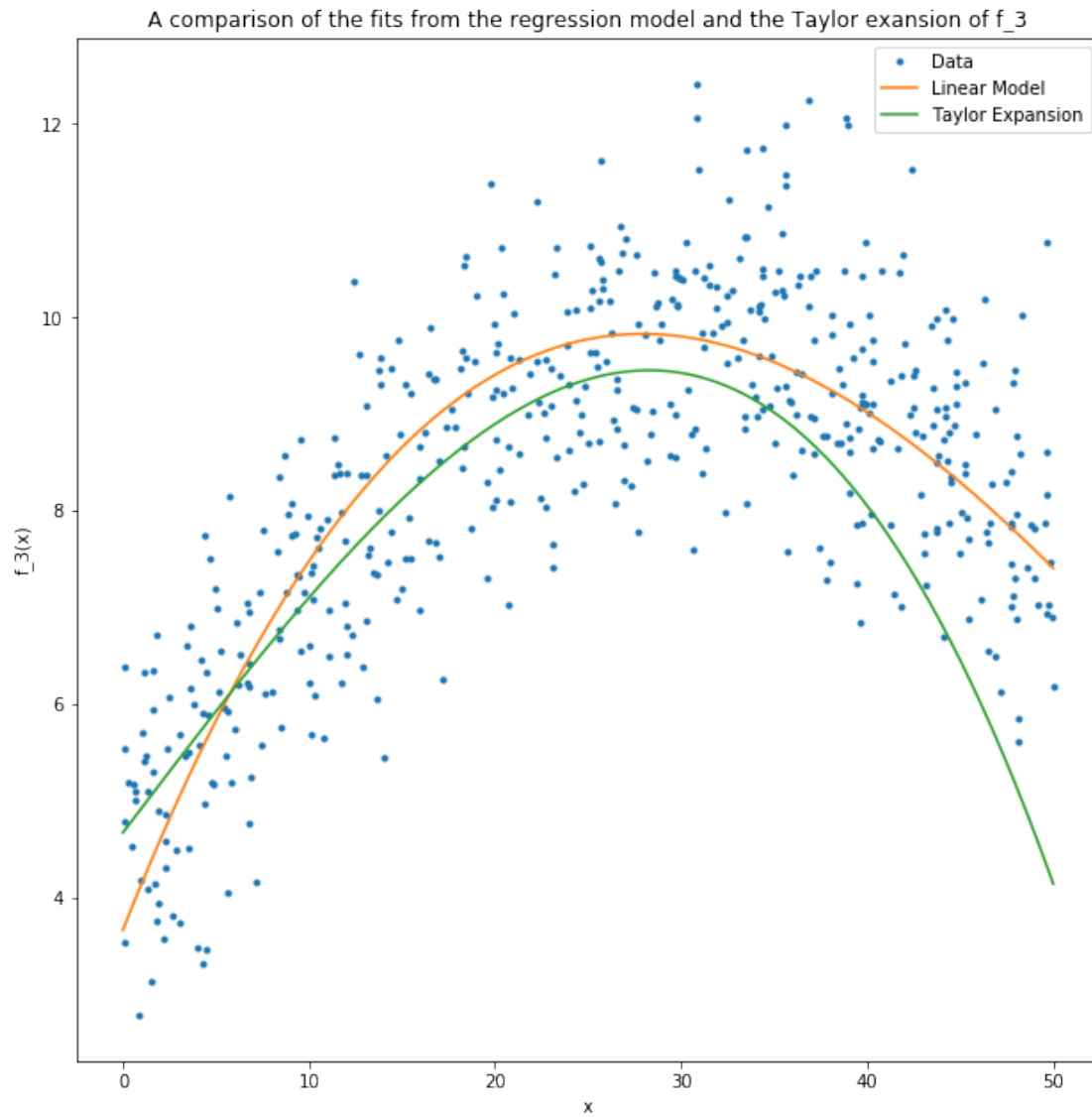
# Only get the observed predictors and response where the predictors are less
# than 50
X_small = list(filter(lambda x: x < 50,X))
Y_small = Y_3[pd.concat([pd.DataFrame(X,columns=['X']),\
                          pd.DataFrame(Y_3,columns=['Y'])],axis=1).\
               apply(lambda x: x[0]<50,axis=1)]

# Plot the data, the fitted model and the taylor expansion
fig = plt.figure(figsize=(8,8))
axes = fig.add_axes([0.1,0.1,1,1])
axes.plot(X_small,Y_small,'.',label='Data')
axes.plot(x32,y32,label='Linear Model')
axes.plot(x32,y_taylor_32,label='Taylor Expansion')

# Set the labels and title
axes.set_xlabel('x')
axes.set_ylabel('f_3(x)')
axes.set_title('A comparison of the fits from the regression model and the \
Taylor exansion of f_3')

# Add the legend
axes.legend()

plt.show()
```



In addition to the R^2 statistic, it is useful to assess whether a variable is statistically significant. To do this for a variable X with coefficient β_1 , we test the null hypothesis

$$H_O : \beta_1 = 0$$

against

$$H_A : \beta_1 \neq 0$$

For the first model we have the fitted model

```
In [32]: print('f(x) = {} + {} X'.format(lm1.intercept_[0], lm1.coef_[0][0]))
```

```
f(x) = 5.501243124853005 + 5.064254524922959 X
```

The standard errors of the estimators $\hat{\beta}_0$ and $\hat{\beta}_1$ for the coefficients have the form

$$SE(\beta_0) = \sqrt{\sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right]} \approx RSE \sqrt{\left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right]}$$

Where RSE is the *residual standard error* estimating the population $\sigma = \sqrt{\text{Var}(\epsilon)}$ and has the form $RSE = \sqrt{\frac{\sum_{i=1}^n \epsilon_i^2}{n-2}} = \sqrt{\frac{RSS}{n-2}}$.

$$SE(\beta_1) = \sqrt{\frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}} \approx RSE \sqrt{\frac{1}{\sum_{i=1}^n (x_i - \bar{x})^2}}$$

(PROOF of these equations?)

Using the standard errors, we can then conduct the hypothesis test above as a t-test. We have that

$$\frac{\hat{\beta}_0 - \beta_0^{(0)}}{SE(\beta_0)} \sim t_{n-2}$$

$$\frac{\hat{\beta}_1 - \beta_1^{(0)}}{SE(\beta_1)} \sim t_{n-2}$$

where $^{(0)}$ denotes the null value.

(PROOF that this is distributed as student t?)

```
In [33]: # we need the scipy.stats package for the t-distribution
from scipy import stats

# number of observations n
n = len(X)

# residual standard error
RSE_1 = np.sqrt(RSS_1/(n-2))

# variance of x = sum (x_i - x_bar)^2. Note that this is the
# population variance calculation
# so we would need to multiply by n
varx_1 = np.var(X)

# mean of x
meanx_1 = np.mean(X)

SE_beta_0 = RSE_1 * np.sqrt(1.0/n + meanx_1**2/(n*varx_1))
SE_beta_1 = RSE_1 * np.sqrt(1.0/(n*varx_1))

print('SE(beta_0) = {}, SE(beta_1) = {}'.format(SE_beta_0, SE_beta_1))

# null hypothesis
betanull_0 = 0
```

```

betanull_1 = 0

tstatistic1_0 = (beta1_0 - betanull_0)/SE_beta_0
tstatistic1_1 = (beta1_1 - betanull_1)/SE_beta_1

print('beta_0 t-statistic = {}'.format(tstatistic1_0))
print('beta_1 t-statistic = {}'.format(tstatistic1_1))

# p-value
# the following function calculates the area under the student t pdf with
# 2 degrees of freedom that is less than -4.303
stats.t.cdf(-4.303,2)

# calculate the p-value using the tstatistic and degrees of freedom n-2
pval1_0 = stats.t.cdf(-tstatistic1_0,n-2)
pval1_1 = stats.t.cdf(-tstatistic1_1,n-2)

print('p-value for beta_0 = {}'.format(pval1_0))
print('p-value for beta_1 = {}'.format(pval1_1))
print('These are both statistically significant!')

SE(beta_0) = 0.6406034056188337, SE(beta_1) = 0.011151051418375258
beta_0 t-statistic = 8.587595814509644
beta_1 t-statistic = 454.150405635995
p-value for beta_0 = 1.685985282508196e-17
p-value for beta_1 = 0.0
These are both statistically significant!

In [34]: def calcpvalue(X,y_observed,y_fitted,beta_0,beta_1,betanull_0,betanull_1):
    '''
    A function to calculate whether the coefficients in a model with 1
        variable is statistically significant.
    X = a list for the data for the variable
    y_observed = the observed values for the response variable
    y_fitted = the predicted values of the model
    beta_0 = the intercept of the model
    beta_1 = the coefficient of the explanatory variable in the model
    betanull_0 = null hypothesis value for the intercept (usually 0)
    betanull_1 = null hypothesis value for the coefficient of the response
        variable (usually 0)
    '''
    # number of observations n
    n = len(X)

    # calculate RSS
    temp,RSS = TSS_RSS(y_observed,y_fitted)

```

```

# residual standard error
RSE = np.sqrt(RSS/(n-2))

# variance of x = sum (x_i - x_bar)^2. Note that this is the population
# variance calculation
# so we would need to multiply by n
varx = np.var(X)

# mean of x
meanx = np.mean(X)

SE_beta_0 = RSE * np.sqrt(1.0/n + meanx**2/(n*varx))
SE_beta_1 = RSE * np.sqrt(1.0/(n*varx))

print('SE(beta_0) = {}, SE(beta_1) = {}'.format(SE_beta_0, SE_beta_1))

# null hypothesis
betanull_0 = 0
betanull_1 = 0

tstatistic1_0 = (beta_0 - betanull_0)/SE_beta_0
tstatistic1_1 = (beta_1 - betanull_1)/SE_beta_1

print('beta_0 t-statistic = {}'.format(tstatistic1_0))
print('beta_1 t-statistic = {}'.format(tstatistic1_1))

# p-value

# calculate the p-value using the tstatistic and degrees of freedom n-2
# Multiply by 2 since it's a 2 tailed test
if(tstatistic1_0 > 0):
    pval_0 = stats.t.cdf(-tstatistic1_0, n-2)*2
else:
    pval_0 = stats.t.cdf(tstatistic1_0, n-2)*2

if(tstatistic1_1 > 0):
    pval_1 = stats.t.cdf(-tstatistic1_1, n-2)*2
else:
    pval_1 = stats.t.cdf(tstatistic1_1, n-2)*2

print('p-value for beta_0 = {}'.format(pval_0))
print('p-value for beta_1 = {}'.format(pval_1))
if((pval_0 <= 0.05) and (pval_1 <= 0.05)):
    print('These are both statistically significant!')
elif(pval_0 <= 0.05):
    print('Only beta_0 is statistically significant!')
elif(pval_1 <= 0.05):
    print('Only beta_1 is statistically significant!')

```

```

else:
    print('The parameters of this model are not statistically significant!')

```

We can do the same calculations for significance for all the models using this function

```

In [35]: print('Model for Y_1: Explanatory variable X for Y_1')
         calcpvalue(X,Y_1,y1_fitted,beta1_0,beta1_1,0,0)

         print()
         print()

         print('Model for Y_2: Explanatory variable X for Y_2')
         calcpvalue(X,Y_2,y2_fitted,beta2_0,beta2_1,0,0)

         print()
         print()

         print('Model for Y_2: Explanatory variable X^2 for Y_2')
         calcpvalue(X**2,Y_2,y22_fitted,beta22_0,beta22_1,0,0)

         print()
         print()

         print('Model for Y_3: Explanatory variable X for Y_3')
         calcpvalue(X,Y_3,y3_fitted,beta3_0,beta3_1,0,0)

```

```

Model for Y_1: Explanatory variable X for Y_1
SE(beta_0) = 0.6406034056188337, SE(beta_1) = 0.011151051418375258
beta_0 t-statistic = 8.587595814509644
beta_1 t-statistic = 454.150405635995
p-value for beta_0 = 3.371970565016392e-17
p-value for beta_1 = 0.0
These are both statistically significant!

```

```

Model for Y_2: Explanatory variable X for Y_2
SE(beta_0) = 245.34955295438897, SE(beta_1) = 4.2708256878947495
beta_0 t-statistic = -34.424274285888536
beta_1 t-statistic = 118.51588098729522
p-value for beta_0 = 8.125468707425302e-172
p-value for beta_1 = 0.0
These are both statistically significant!

```

```

Model for Y_2: Explanatory variable X^2 for Y_2
SE(beta_0) = 24.614546607361707, SE(beta_1) = 0.005557804748590844
beta_0 t-statistic = 0.5878663289694033
beta_1 t-statistic = 913.1340896074505

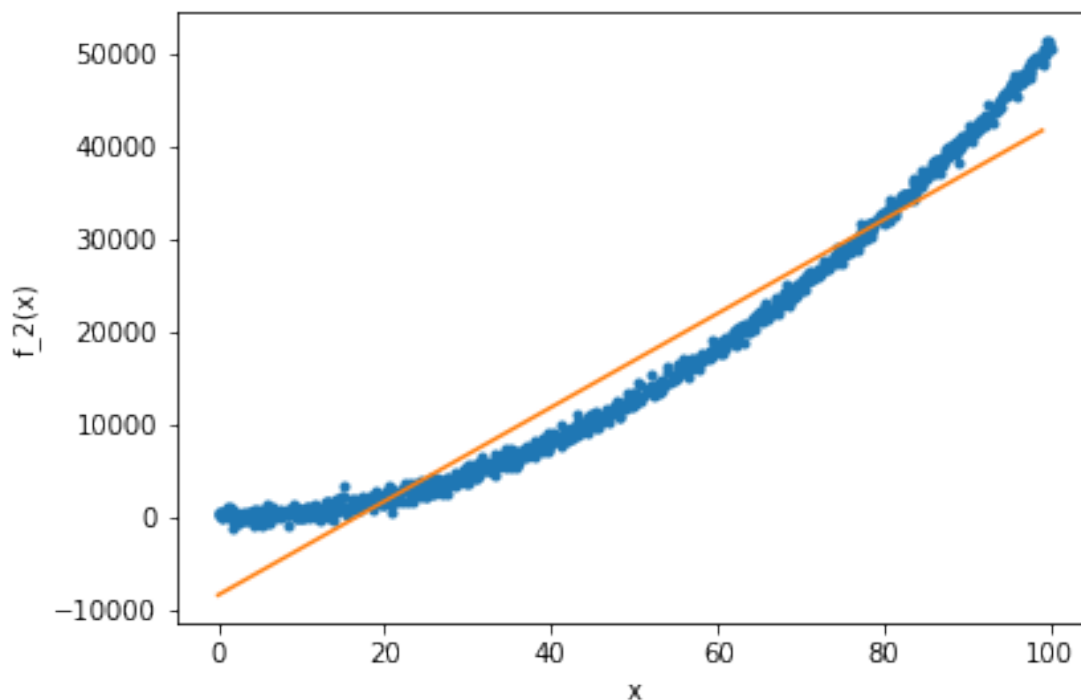
```

```
p-value for beta_0 = 0.5567550098751695
p-value for beta_1 = 0.0
Only beta_1 is statistically significant!
```

```
Model for Y_3: Explanatory variable X for Y_3
SE(beta_0) = 0.15212372264589394, SE(beta_1) = 0.0026480337730023893
beta_0 t-statistic = 69.09601786545896
beta_1 t-statistic = -38.21657519695403
p-value for beta_0 = 0.0
p-value for beta_1 = 1.3682773718716098e-197
These are both statistically significant!
```

```
In [36]: fig = plt.figure()
        axes = fig.add_axes([0.1,0.1,0.8,0.8])
        axes.plot(X,Y_2, '. ')
        axes.set_xlabel('x')
        axes.set_ylabel('f_2(x)')
        axes.plot(x2,y2)
```

```
Out[36]: [<matplotlib.lines.Line2D at 0x23a2f59b1d0>]
```



We can use the statsmodels.api to verify our results


```
In [37]: import statsmodels.api as sm
        from scipy import stats
```

C:\Users\HVAD\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The pandas.core.datetools module is deprecated. Use pandas.tseries.tools instead.

```
from pandas.core import datetools
```

```
In [38]: print('Model for Y_1: Explanatory variable X for Y_1')
```

```
# add a column of ones to X
X_new = sm.add_constant(X)
```

```
# ordinary least squares approach to optimisation
est = sm.OLS(Y_1, X_new)
```

```
# fit the data to the model using OLS
est2 = est.fit()
```

```
# print a summary of the model
print(est2.summary())
```

```
print()
print()
```

```
#re-run the above for all the models
```

```
print('Model for Y_2: Explanatory variable X for Y_2')
X_new = sm.add_constant(X)
est = sm.OLS(Y_2, X_new)
est2 = est.fit()
print(est2.summary())
```

```
print()
print()
```

```
print('Model for Y_2: Explanatory variable X^2 for Y_2')
X_new = sm.add_constant(X**2)
est = sm.OLS(Y_2, X_new)
est2 = est.fit()
print(est2.summary())
```

```
print()
print()
```

```
print('Model for Y_3: Explanatory variable X for Y_3')
X_new = sm.add_constant(X)
est = sm.OLS(Y_3, X_new)
est2 = est.fit()
```

```

print(est2.summary())

print()
print()

print('Model for Y_3: Explanatory variables X,X^2,X^3 for Y_3')
# concatenate multiple variables
X_new = sm.add_constant(pd.concat([pd.DataFrame(X,columns=['X']),\
                                   pd.DataFrame(X**2,columns=['X2']),\
                                   pd.DataFrame(X**3,columns=['X3'])],axis=1))

est = sm.OLS(Y_3, X_new)
est2 = est.fit()
print(est2.summary())

```

Model for Y_1: Explanatory variable X for Y_1
 OLS Regression Results

```

=====
Dep. Variable:          y      R-squared:          0.995
Model:                OLS      Adj. R-squared:       0.995
Method:             Least Squares      F-statistic:      2.063e+05
Date:                Tue, 13 Nov 2018      Prob (F-statistic):      0.00
Time:                09:31:02      Log-Likelihood:      -3730.1
No. Observations:      1000      AIC:              7464.
Df Residuals:          998      BIC:              7474.
Df Model:              1
Covariance Type:      nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	5.5012	0.641	8.588	0.000	4.244	6.758
x1	5.0643	0.011	454.150	0.000	5.042	5.086

```

=====
Omnibus:              0.350      Durbin-Watson:      1.952
Prob(Omnibus):        0.839      Jarque-Bera (JB):    0.376
Skew:                 -0.045      Prob(JB):            0.828
Kurtosis:             2.970      Cond. No.            115.
=====

```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Model for Y_2: Explanatory variable X for Y_2
 OLS Regression Results

```

=====
Dep. Variable:          y      R-squared:          0.934
Model:                OLS      Adj. R-squared:       0.934
Method:             Least Squares      F-statistic:      1.405e+04

```

Date: Tue, 13 Nov 2018 Prob (F-statistic): 0.00
Time: 09:31:02 Log-Likelihood: -9678.1
No. Observations: 1000 AIC: 1.936e+04
Df Residuals: 998 BIC: 1.937e+04
Df Model: 1
Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	-8445.9803	245.350	-34.424	0.000	-8927.440	-7964.520
x1	506.1607	4.271	118.516	0.000	497.780	514.541
Omnibus:	136.837		Durbin-Watson:	1.872		
Prob(Omnibus):	0.000		Jarque-Bera (JB):	102.303		
Skew:	0.681		Prob(JB):	6.10e-23		
Kurtosis:	2.227		Cond. No.	115.		

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Model for Y_2: Explanatory variable X^2 for Y_2
OLS Regression Results

Dep. Variable: y R-squared: 0.999
Model: OLS Adj. R-squared: 0.999
Method: Least Squares F-statistic: 8.338e+05
Date: Tue, 13 Nov 2018 Prob (F-statistic): 0.00
Time: 09:31:02 Log-Likelihood: -7670.0
No. Observations: 1000 AIC: 1.534e+04
Df Residuals: 998 BIC: 1.535e+04
Df Model: 1
Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	14.4701	24.615	0.588	0.557	-33.832	62.772
x1	5.0750	0.006	913.134	0.000	5.064	5.086
Omnibus:	5.725		Durbin-Watson:	2.021		
Prob(Omnibus):	0.057		Jarque-Bera (JB):	7.275		
Skew:	0.018		Prob(JB):	0.0263		
Kurtosis:	3.416		Cond. No.	6.64e+03		

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 6.64e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Model for Y_3: Explanatory variable X for Y_3

OLS Regression Results

```

=====
Dep. Variable:          y      R-squared:          0.594
Model:                  OLS    Adj. R-squared:      0.594
Method:                 Least Squares  F-statistic:      1461.
Date:                   Tue, 13 Nov 2018  Prob (F-statistic):  1.37e-197
Time:                   09:31:02  Log-Likelihood:    -2292.4
No. Observations:      1000    AIC:              4589.
Df Residuals:          998     BIC:              4599.
Df Model:               1
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	10.5111	0.152	69.096	0.000	10.213	10.810
x1	-0.1012	0.003	-38.217	0.000	-0.106	-0.096

```

=====
Omnibus:                26.494  Durbin-Watson:          1.871
Prob(Omnibus):          0.000  Jarque-Bera (JB):        28.130
Skew:                   -0.405  Prob(JB):                7.79e-07
Kurtosis:               2.860  Cond. No.:               115.
=====

```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Model for Y_3: Explanatory variables X,X^2,X^3 for Y_3

OLS Regression Results

```

=====
Dep. Variable:          y      R-squared:          0.923
Model:                  OLS    Adj. R-squared:      0.923
Method:                 Least Squares  F-statistic:      3974.
Date:                   Tue, 13 Nov 2018  Prob (F-statistic):  0.00
Time:                   09:31:02  Log-Likelihood:    -1461.8
No. Observations:      1000    AIC:              2932.
Df Residuals:          996     BIC:              2951.
Df Model:               3
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	3.6644	0.128	28.526	0.000	3.412	3.917

X	0.4871	0.011	43.605	0.000	0.465	0.509
X2	-0.0112	0.000	-42.571	0.000	-0.012	-0.011
X3	5.868e-05	1.74e-06	33.743	0.000	5.53e-05	6.21e-05

```
=====
Omnibus:                0.415    Durbin-Watson:                1.980
Prob(Omnibus):           0.813    Jarque-Bera (JB):         0.368
Skew:                    0.046    Prob(JB):                 0.832
Kurtosis:                3.019    Cond. No.                 1.46e+06
=====
```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.46e+06. This might indicate that there are strong multicollinearity or other numerical problems.

It looks like the intercept for *Model for Y_2: Explanatory variable X^2 for Y_2* is not statistically significant. The intercept can then be omitted from the model and fitted again.

```
In [39]: print('Model for Y_2: Explanatory variable X^2 for Y_2')
         est = sm.OLS(Y_2, X**2)
         est2 = est.fit()
         print(est2.summary())
```

Model for Y_2: Explanatory variable X^2 for Y_2
OLS Regression Results

```
=====
Dep. Variable:          y    R-squared:                0.999
Model:                  OLS    Adj. R-squared:            0.999
Method:                 Least Squares    F-statistic:          1.878e+06
Date:                   Tue, 13 Nov 2018    Prob (F-statistic):      0.00
Time:                   09:31:03    Log-Likelihood:         -7670.2
No. Observations:       1000    AIC:                   1.534e+04
Df Residuals:           999    BIC:                   1.535e+04
Df Model:                1
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
x1	5.0775	0.004	1370.392	0.000	5.070	5.085

```
=====
Omnibus:                6.001    Durbin-Watson:                2.020
Prob(Omnibus):           0.050    Jarque-Bera (JB):             7.710
Skew:                    0.019    Prob(JB):                     0.0212
Kurtosis:                3.428    Cond. No.                     1.00
=====
```

Warnings:

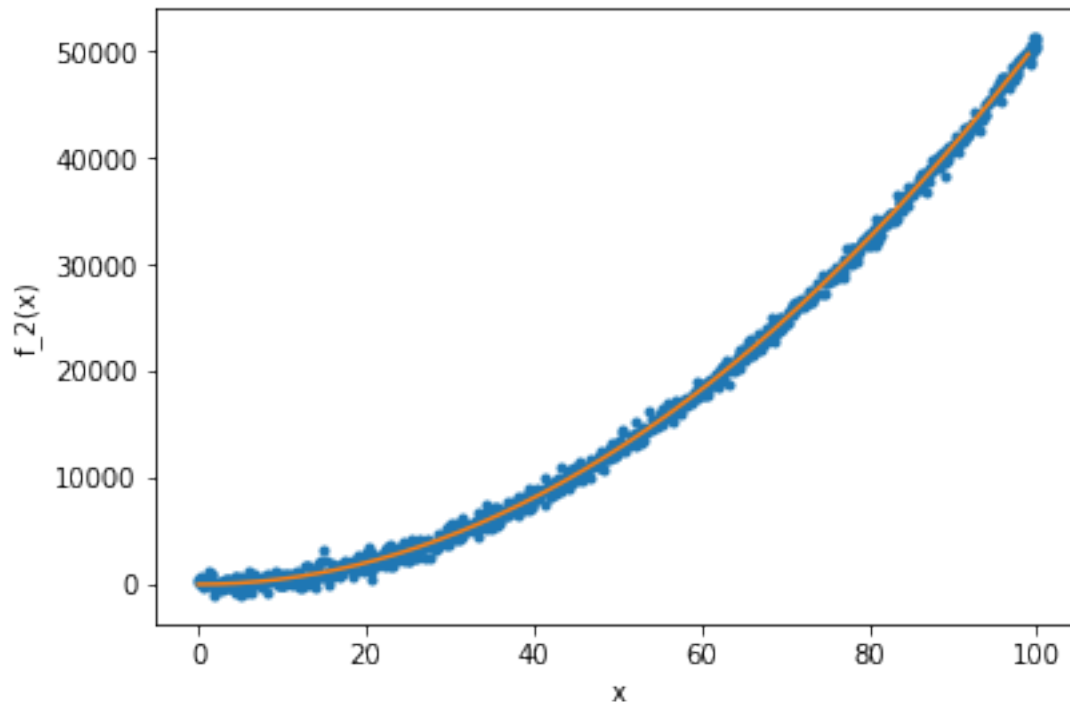
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

This is a good fit also

```
In [40]: x23 = np.linspace(0,99,1000) # 1000 linearly spaced numbers
         y23 = est2.params[0] * x23**2

         fig = plt.figure()
         axes = fig.add_axes([0.1,0.1,0.8,0.8])
         axes.plot(X,Y_2, '. ')
         axes.set_xlabel('x')
         axes.set_ylabel('f_2(x)')
         axes.plot(x23,y23)
```

```
Out[40]: [<matplotlib.lines.Line2D at 0x23a30ab4b70>]
```



If we set $\beta_0 = 0$ in the derivation for $\hat{\beta}_0$ and $\hat{\beta}_1$ earlier in the article, we would have obtained the equation

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n y_i x_i}{\sum_{i=1}^n x_i^2}$$

Using this equation, we can reproduce the statsmodels solution above. Note that removing β_0 has changed β_1 slightly:

```
In [41]: # remember that we are fitting the variable X^2
sum1 = np.sum(Y_2*X**2)
sum2 = np.sum(X**4)

beta23_1 = sum1/sum2

print('Y ~ {} X^2'.format(beta23_1))

Y ~ 5.077455649665152 X^2
```

F-Statistic The F-Statistic answers the question 'Is there evidence that at least one of the explanatory variables is related to the response variable?'. This corresponds to a hypothesis test with:

$$H_0 : \beta_0, \beta_1, \dots, \beta_p = 0$$

$$H_A : \text{at least one of } \beta_i \text{ is non-zero}$$

The F-Statistic has the form:

$$F = \frac{(TSS - RSS)/p}{RSS/(n - p - 1)}$$

where p is the number of explanatory variables/parameters.

(DERIVATION of this equation?)

If H_0 is not true, the numerator in the above equation becomes larger, i.e. $F > 1$. If H_0 is true, then the F-Statistic is close to 1.

(PROOF of this - take expectation of numerator and denominator and these are both equal to $\text{Var}(\epsilon)$. If H_A is true then the numerator $> \text{Var}(\epsilon)$)

We can use this to calculate the F-Statistics of the above models:

```
In [42]: def FStat(n,p,TSS,RSS):
          F = ((TSS-RSS)/p)/(RSS/(n-p-1))
          print('The F-Statistic is {}'.format(F))

In [43]: # we didn't calculate the last model ourselves, we used sklearn
          # so we retrieve the coefficients
beta32_0 = lm32.intercept_[0]
beta32_1 = lm32.coef_[0][0]
beta32_2 = lm32.coef_[0][1]
beta32_3 = lm32.coef_[0][2]

In [44]: print('Model for Y_1: Explanatory variable X for Y_1')
          FStat(len(X),1,TSS_1,RSS_1)

          print()
          print()

          #re-run the above for all the models
```

```

print('Model for Y_2: Explanatory variable X for Y_2')
FStat(len(X),1,TSS_2,RSS_2)

print()
print()

print('Model for Y_2: Explanatory variable X^2 for Y_2')
FStat(len(X),1,TSS_22,RSS_22)

print()
print()

print('Model for Y_3: Explanatory variable X for Y_3')
FStat(len(X),1,TSS_3,RSS_3)

print()
print()

TSS_32,RSS_32 = TSS_RSS(Y_3,y32_fitted_sklearn)

print('Model for Y_3: Explanatory variables X,X^2,X^3 for Y_3')
# now we have 3 explanatory variables
FStat(len(X),3,TSS_32,RSS_32)

```

Model for Y_1: Explanatory variable X for Y_1
The F-Statistic is 206252.59093933867

Model for Y_2: Explanatory variable X for Y_2
The F-Statistic is 14046.014046194661

Model for Y_2: Explanatory variable X^2 for Y_2
The F-Statistic is 833813.8656032282

Model for Y_3: Explanatory variable X for Y_3
The F-Statistic is 1460.506619784441

Model for Y_3: Explanatory variables X,X^2,X^3 for Y_3
The F-Statistic is 3974.1603226694533

These match the *statsmodels* outputs. We can also find the p-value of a coefficient/intercept using the F-Statistic. The F-Statistic formula becomes:

$$F = \frac{(RSS_0 - RSS)/q}{RSS/(n - p - 1)}$$

where RSS_0 is the residual sum of squares for the model with q removed parameters. The corresponding hypothesis test is then

$$H_0 : \{\beta_i = 0\} \text{ where } i \text{ takes on the } q \text{ removed parameters}$$

$$H_A : \text{at least one of those } q \text{ parameters is non-zero}$$

Above, we ran a model for Y_2 which had an intercept, coefficient of X^2 and RSS of:

```
In [45]: beta22_0, beta22_1, RSS_22
```

```
Out[45]: (14.470063153316005, 5.075020979320466, 268902718.6114595)
```

Here, we are going to calculate the p-value of the intercept for Y_2 when we try to fit an intercept as well as X^2 . We do this by first fitting the full model including the intercept and getting the RSS value, then we fit the model without the intercept and get the RSS value. The Coefficient of X^2 and RSS for the model without the intercept was calculated to be

```
In [46]: TSS_23, RSS_23 = TSS_RSS(Y_2, beta23_1 * X**2)
print('beta_1 = {}, RSS_0 = {}'.format(beta23_1, RSS_23))
```

```
beta_1 = 5.077455649665152, RSS_0 = 268995834.0780044
```

We now create a function to apply the formula shown above for calculating the F-Statistic for comparing models

```
In [47]: def FStatCompare(n,p,q,RSS0,RSS):
    '''
    A function to calculate the F-Statistic when we are comparing models
    with different number of parameters.
    RSS0 is a sub-model of RSS
    '''
    F = ((RSS0-RSS)/q)/(RSS/(n-p-1))
    print('The F-Statistic is {}'.format(F))
    return F
```

Now we can confirm the p-value for the intercept

```
In [48]: # This is the fitted values for the model with no intercept
Y23_fitted = beta23_1 * X**2

# These are the TSS and RSS for this model with no intercept
TSS_2_test, RSS_2_test = TSS_RSS(Y_2, Y23_fitted)

# RSS_22 is the RSS for the model with the intercept. RSS_23 is the RSS
# for the model without the intercept. We have p = 0 and q = 1 (i.e. we have
# removed 1 parameter but there was only 1 parameter to begin with)
F = FStatCompare(len(X), 0, 1, RSS_23, RSS_22)
```

```

# the following function calculates the area underneath the cdf F-distribution
# with dfn(degrees of freedom in the numerator)=1,
# dfd(degrees of freedom in the denominator)=len(X)-2 less than 0.5
stats.f.cdf(0.5,1,len(X)-2)

print('The p-value of the intercept is {}'.format(1-stats.f.cdf(F,1,len(X)-2)))

```

The F-Statistic is 0.3459331001141355

The p-value of the intercept is 0.5565574505496756

Note that above, we removed the intercept and used the F-Statistic to calculate the p-value for the intercept. We can also remove the coefficient of X^2 and calculate the p-value of this coefficient using the same procedure as above. First fit the model as we have done before

```

In [49]: lmOnlyIntercept = LinearRegression()
lmOnlyIntercept.fit((X*0).reshape(-1,1),Y_2.reshape(-1,1))
print('Model for Y_2: No explanatory variable for Y_2')
print('beta_0 = {}'.format(lmOnlyIntercept.intercept_[0]))
yOnlyIntercept_fitted_sklearn = lmOnlyIntercept.intercept_[0] + X*0
print('R^2 = {}'.format(r2_score(Y_2,yOnlyIntercept_fitted_sklearn)))

```

Model for Y_2: No explanatory variable for Y_2

beta_0 = 16763.308428792458

R^2 = 0.0

Next, calculate the RSS for this model we have just fitted

```

In [50]: TSS_OnlyIntercept,RSS_OnlyIntercept = TSS_RSS(Y_2,yOnlyIntercept_fitted_sklearn)
print('beta_0 = {}, RSS_0 = {}'.format(lmOnlyIntercept.intercept_[0],\
                                        RSS_OnlyIntercept))

```

beta_0 = 16763.308428792458, RSS_0 = 224933046282.3772

And now we calculate the p-value of the coefficient of X^2

```

In [51]: # These are the TSS and RSS for this model with only intercept
TSS_2_test,RSS_2_test = TSS_RSS(Y_2,yOnlyIntercept_fitted_sklearn)

# RSS_22 is the RSS for the model with the intercept. RSS_23 is the RSS
# for the model without the intercept. We have p = 0 and q = 1 (i.e. we have
# removed 1 parameter but there was only 1 parameter to begin with)
F = FStatCompare(len(X),0,1,RSS_2_test,RSS_22)

# the following function calculates the area underneath the cdf F-distribution
# with dfn(degrees of freedom in the numerator)=1,
# dfd(degrees of freedom in the denominator)=len(X)-2 less than 0.5

```

```
stats.f.cdf(0.5,1,len(X)-2)

print('The p-value of the X^2 coefficient is {}'.format(1-stats.f.cdf(F,1,len(X)-2)))
```

The F-Statistic is 834649.3504385022

The p-value of the X^2 coefficient is 1.1102230246251565e-16

1.3.1 Synergy Effect

Suppose we have the following function

$$f(x) = 4.67 + 2 * X_1 + 3 * X_2 + 5.07X_1 * X_2$$

We can see that there is a mixed term ' X_1X_2 '. This is called a synergy effect.

Let's define this function and plot it

```
In [52]: # We will need to plot in 3D
from mpl_toolkits.mplot3d import Axes3D

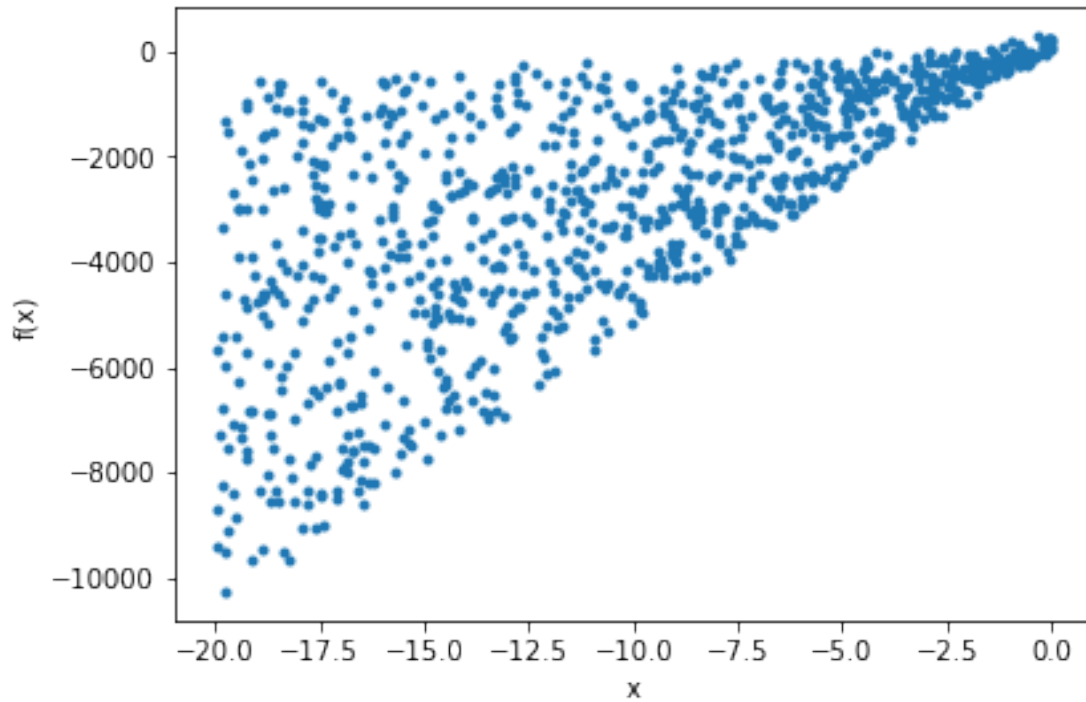
#f(x)=4.67+2*X_1+3*X_2+5.07X_1*X_2
def f(x1,x2):
    return 4.67+2*x1+30*x2+5.07*x1*x2
# Set the seed
r = np.random.RandomState(101)
X_1 = 100*r.rand(1000)
X_2 = -20*r.rand(1000)

#Error term with sigma = 10, mu = 0
E = 100*r.randn(1000)

#Response variables
Y = list(map(f,X_1,X_2))+E

fig = plt.figure()
axes = fig.add_axes([0.1,0.1,0.8,0.8])
axes.plot(X_2,Y,'.')
axes.set_xlabel('x')
axes.set_ylabel('f(x)')
```

```
Out [52]: Text(0,0.5,'f(x)')
```

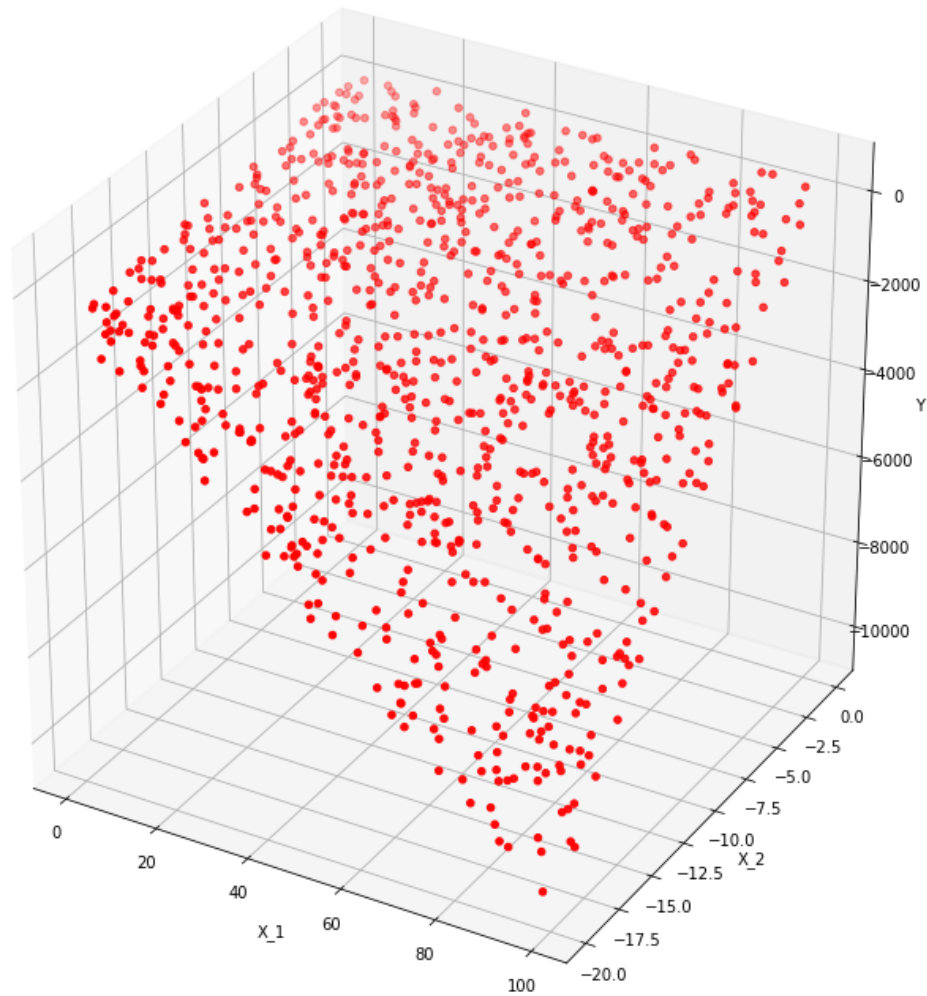


```
In [53]: fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X_1, X_2, Y, c='r', marker='o')

ax.set_xlabel('X_1')
ax.set_ylabel('X_2')
ax.set_zlabel('Y')

plt.tight_layout()
```



Suppose we continued to fit a linear regression model with parameters X_1 and X_2 with the assumption that there is no synergy effect.

```
In [54]: print('Model for Y: Explanatory variable X_1 and X_2 for Y')
X_new = sm.add_constant(pd.concat([pd.DataFrame(X_1,columns=['X_1']),\
                                   pd.DataFrame(X_2,columns=['X_2'])],axis=1))

est = sm.OLS(Y, X_new)
est2 = est.fit()
print(est2.summary())
```

```
Model for Y: Explanatory variable X_1 and X_2 for Y
              OLS Regression Results
```

```
=====
Dep. Variable:              y    R-squared:              0.864
```

```

Model:                                OLS    Adj. R-squared:            0.864
Method:                             Least Squares    F-statistic:            3169.
Date:                               Tue, 13 Nov 2018    Prob (F-statistic):      0.00
Time:                               09:31:06    Log-Likelihood:         -8160.3
No. Observations:                    1000    AIC:                    1.633e+04
Df Residuals:                        997    BIC:                    1.634e+04
Df Model:                            2
Covariance Type:                     nonrobust

```

```

=====
              coef      std err          t      P>|t|      [0.025      0.975]
-----
const          2562.3530      71.652      35.761      0.000      2421.746      2702.960
X_1             -49.6977       0.937     -53.063      0.000      -51.536      -47.860
X_2             279.5368       4.683      59.693      0.000       270.347       288.726
=====
Omnibus:                 3.561    Durbin-Watson:           1.909
Prob(Omnibus):            0.169    Jarque-Bera (JB):         4.035
Skew:                     -0.022    Prob(JB):                 0.133
Kurtosis:                 3.308    Cond. No.                  155.
=====

```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The above output shows that the R^2 is almost 87% with both X_1 and X_2 being statistically significant. Below, we show that including the synergy term X_1X_2 into the model as well greatly improves the R^2 metric.

```

In [55]: print('Model for f: Explanatory variables X_1, X_2 and X_1 * X_2 for Y_2')
          X_new = sm.add_constant(pd.concat([pd.DataFrame(X_1,columns=['X_1']),\
                                              pd.DataFrame(X_2,columns=['X_2']),\
                                              pd.DataFrame(X_1*X_2,columns=['X_12'])],axis=1))

          est = sm.OLS(Y, X_new)
          est2 = est.fit()
          print(est2.summary())

```

Model for f: Explanatory variables X_1, X_2 and X_1 * X_2 for Y_2

OLS Regression Results

```

=====
Dep. Variable:              y    R-squared:            0.998
Model:                    OLS    Adj. R-squared:            0.998
Method:                  Least Squares    F-statistic:            1.651e+05
Date:                    Tue, 13 Nov 2018    Prob (F-statistic):      0.00
Time:                    09:31:06    Log-Likelihood:         -6052.5
No. Observations:        1000    AIC:                    1.211e+04
Df Residuals:            996    BIC:                    1.213e+04
Df Model:                 3

```

```

Covariance Type:          nonrobust
=====
              coef      std err          t      P>|t|      [0.025      0.975]
-----
const          17.9262      13.164        1.362      0.174      -7.907      43.759
X_1              2.0906       0.231        9.054      0.000       1.637       2.544
X_2             30.3293       1.122       27.035      0.000      28.128      32.531
X_12              5.0841       0.020      257.798      0.000       5.045       5.123
=====
Omnibus:                8.045   Durbin-Watson:                2.015
Prob(Omnibus):           0.018   Jarque-Bera (JB):           11.082
Skew:                    0.035   Prob(JB):                   0.00392
Kurtosis:                3.511   Cond. No.                   2.69e+03
=====

```

Warnings:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.69e+03. This might indicate that there are strong multicollinearity or other numerical problems.

We have seen above that adding a term X_1X_2 significantly increased the R^2 statistic. Instead of adding this joint term, what would be the effect on R^2 if we added random noise? We see the effect below.

```

In [56]: # Set the seed
         r = np.random.RandomState(11)

         #Error term with sigma = 10, mu = 0
         noise = 100*r.randn(1000)

         print('Model for Y: Explanatory variable X_1 and X_2 for Y')
         X_new = sm.add_constant(pd.concat([pd.DataFrame(X_1,columns=['X_1']),\
                                             pd.DataFrame(X_2,columns=['X_2']),\
                                             pd.DataFrame(noise,columns=['Noise'])],axis=1))

         est = sm.OLS(Y, X_new)
         est2 = est.fit()
         print(est2.summary())

```

```

Model for Y: Explanatory variable X_1 and X_2 for Y
              OLS Regression Results

```

```

=====
Dep. Variable:          y      R-squared:                0.865
Model:                  OLS      Adj. R-squared:           0.864
Method:                 Least Squares      F-statistic:        2123.
Date:                   Tue, 13 Nov 2018     Prob (F-statistic):      0.00
Time:                   09:31:06     Log-Likelihood:        -8157.8
No. Observations:      1000      AIC:                  1.632e+04

```

Df Residuals: 996 BIC: 1.634e+04
Df Model: 3
Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	2548.0238	71.812	35.482	0.000	2407.103	2688.945
X_1	-49.6148	0.936	-53.033	0.000	-51.451	-47.779
X_2	278.5613	4.695	59.331	0.000	269.348	287.775
Noise	-0.5863	0.267	-2.196	0.028	-1.110	-0.062
Omnibus:	3.763	Durbin-Watson:	1.906			
Prob(Omnibus):	0.152	Jarque-Bera (JB):	4.348			
Skew:	-0.006	Prob(JB):	0.114			
Kurtosis:	3.323	Cond. No.	271.			

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Including an unrelated, random noise term to the model increases the R^2 statistic. This makes sense since when fitting the model to the training data, in the worst case, the model could choose a predictor's coefficient to be zero. This means that the R^2 statistic for the training data should never decrease as a function of the number of predictors. The main reason for introducing such a metric is to gauge how well the model describes the population from which our data originates from. However, if it never decreases then how can it be determined whether the added parameter is useful or not?

In order to cater for this, the Adjusted R^2 metric can be used. This metric applies a penalty to the usual R^2 the more predictors that are used. This way, it is not possible that the Adjusted R^2 can increase indefinitely. At some point, the contribution to the R^2 of adding a new predictor will be overcome by the penalty attributed to adding that new parameter. The Adjusted R^2 is as follows:

$$\text{Adjusted } R^2 = 1 - \frac{RSS/(n - p - 1)}{TSS/(n - 1)}$$

where p is the number of predictor. It can be seen in the above model that the Adjusted R^2 did not increase with the addition of another predictor.

Another approach we can apply to take into account that the test R^2 will always be smaller than the training R^2 , is to divide the data we have into a training set and a testing set. We can then train the model on the training set and test it on the unseen testing set in order to determine how well it has performed.

We tackle this in the next section.

1.3.2 Cross Validation

Cross Validation is a technique to estimate how well a model will perform on unseen data. As mentioned in the previous section, the entire data set available can be divided into two: a training

set and a testing set. The question then becomes, 'what portion of the dataset should be the training set?'. This question can be expressed as follows:

- Let the number of observations be n , then the training set is $n - k$ where $k \in [1, n - 1]$

The reason this question is important is that the choice of k greatly influences the bias in our cross validation. If $k = \lfloor n/2 \rfloor$ then the test error will be greatly overestimated since the final model will be trained on n observations, not $\lfloor n/2 \rfloor$ observations. On the other hand, if $k = 1$, the variance of our test error will be very large since the technique will depend greatly on which observation we chose as the test observation.

Going further, we can divide the entire dataset into roughly n/k subsets. We can then run n/k different cross validations leaving a different subset as the test set at each iteration. The test error (or R^2) can then be approximated as the average of the different subset test errors. This immediately means that if n is large, choosing to assess the model performance using cross validation with $k = 1$ could be computationally intense. Therefore, a value for k somewhere in the range $(1, \lfloor n/2 \rfloor)$ may be wiser.

To start things off, let's fit a linear regression model to the house prices dataset and test it on a portion of the data.

```
In [57]: from sklearn.linear_model import LinearRegression
         from sklearn.cross_validation import train_test_split
         from sklearn.metrics import r2_score, mean_squared_error
```

```
C:\Users\HVAD\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: 
    "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [58]: housePrice.columns
```

```
Out[58]: Index(['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
               'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
               '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
               'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
               'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
               'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
               'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
              dtype='object')
```

```
In [59]: # The predictor and response
         X = housePrice['YearBuilt'].values.reshape(-1,1)
         y = housePrice['SalePrice'].values.reshape(-1,1)

         # Make 33% of this dataset a test set
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)

         # Linear Regression model object
         model = LinearRegression()

         # Fit this model using the training data
```

```

model.fit(X_train,y_train)

# Predict
predictions = model.predict(X_test)

# Get the RSS
tss,rss = TSS_RSS(y_test,predictions)

# The MSE is RSS/n_test
MSE = rss/len(y_test)

print('The MSE is {}'.format(MSE))

# Plot the predictions
plt.scatter(X_test,y_test)
plt.scatter(X_test,predictions)

```

Let's see what the MSE is when we use the 'LotArea' predictor to predict 'SalePrice'.

```

In [69]: # The predictor and response
X = housePrice['LotArea'].values.reshape(-1,1)
y = housePrice['SalePrice'].values.reshape(-1,1)

# Make 33% of this dataset a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Linear Regression model object
model = LinearRegression()

# Fit this model using the training data
model.fit(X_train,y_train)

# Predict
predictions = model.predict(X_test)

# Get the RSS
tss,rss = TSS_RSS(y_test,predictions)

# The MSE is RSS/n_test
MSE = rss/len(y_test)

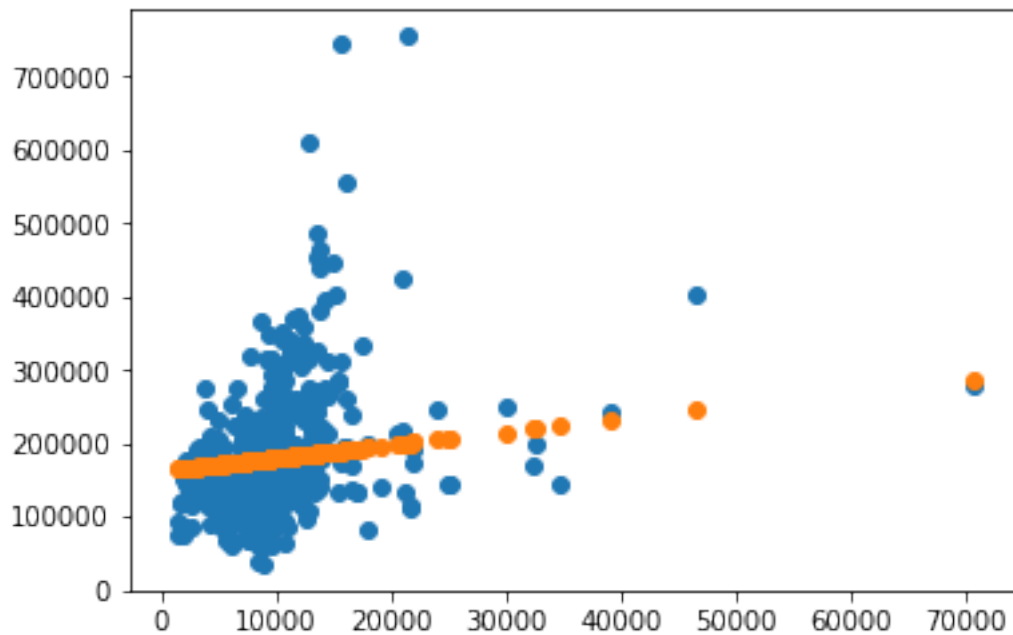
print('The MSE is {}'.format(MSE))

# Plot the predictions
plt.scatter(X_test,y_test)
plt.scatter(X_test,predictions)

```

[2.27491446e+09]

Out[69]: <matplotlib.collections.PathCollection at 0x23a2f682358>



An important point to note in the above MSE calculations is that these MSE results are highly biased. We used a train - test split of 33%. However, in reality, we have the full dataset to train our model on. This means that the above is overestimating the test MSE of the model. In other words, by using only a subset of our dataset to train our model, we are not making use of the full power of the data we have. We can go to the other extreme and select one single observation from our data set of n observations as a test set and the remaining $n-1$ observations as a training set. This is called Leave One Out Cross Validation (LOOCV). We do that below.

```
In [248]: import random

# The predictor and response
X = housePrice['LotArea']
y = housePrice['SalePrice']

# Select a random element to be the test set
r = random.SystemRandom()
testint = r.randint(0, len(X))

# The train set
X_train = X.copy().values.reshape(-1,1)
y_train = y.copy().values.reshape(-1,1)

# The test set is that one observation
X_test = X_train[testint]
X_test = X_test.reshape(1,-1)
```

```

# The train set is all observations except that one observation
np.delete(X_train,testint)

# The test set consist of one response
y_test = y_train[testint]
np.delete(y_train,testint)

# The Linear Regression model object
model = LinearRegression()

# Fit the model
model.fit(X_train,y_train)

# Predict
predictions = model.predict(X_test)

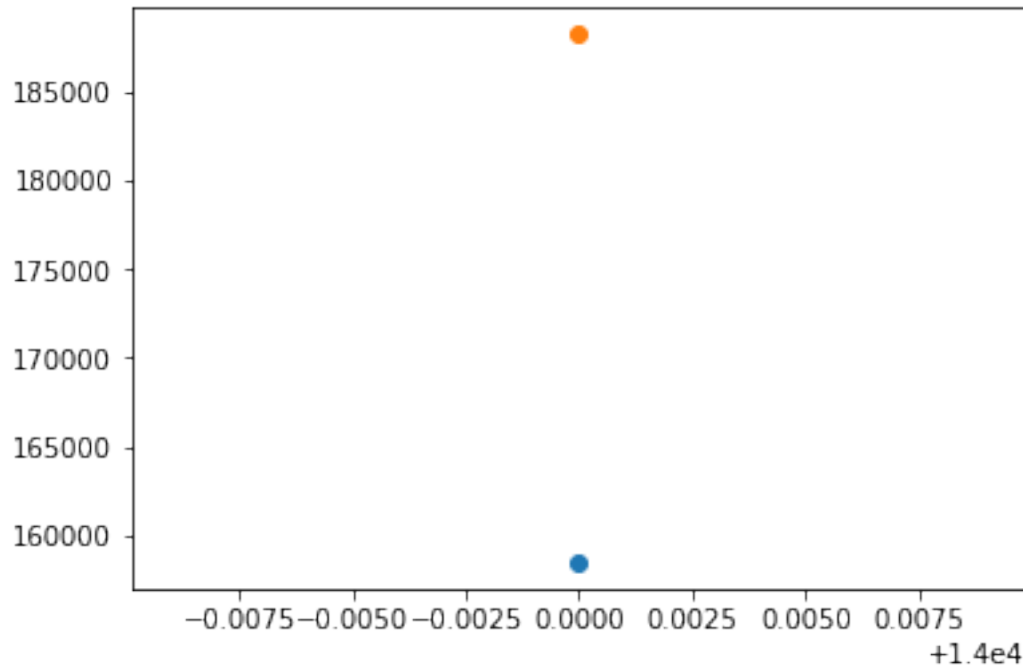
# Get the MSE. MSE = RSS/n
tss,rss = TSS_RSS(y_test,predictions)
MSE = rss/len(y)
print(MSE)

# Plot
plt.scatter(X_test,y_test)
plt.scatter(X_test,predictions)

```

[605626.97016195]

Out[248]: <matplotlib.collections.PathCollection at 0x23a35a981d0>



Each time we run the above code, we get a completely different test MSE. This is because the test MSE depends on which observation we chose to test the model on. So this reduces the bias to a minimum but has a large variance. We can iterate over all the cases where for each iteration we leave a different observation as a test observation. Then we calculate the average MSE over all these Cross Validations.

```
In [249]: # The predictor and response
X = housePrice['YearBuilt']
y = housePrice['SalePrice']

# The MSE array. Each element is the MSE of a particular Cross Validation
MSE = []

# Perform LOOCV on the data using Linear Regression
for i in range(len(X)):
    # The training set
    X_train = X.copy().values.reshape(-1,1)
    y_train = y.copy().values.reshape(-1,1)

    # The test set is a single observation
    X_test = X_train[i]
    X_test = X_test.reshape(1,-1)
    X_train = np.delete(X_train,i)

    # The test set is a single observation
    y_test = y_train[i]
```

```

y_test = y_test.reshape(1,-1)
y_train = np.delete(y_train,i)

# Train the model
model = LinearRegression()

# Fit
model.fit(X_train.reshape(-1,1),y_train.reshape(-1,1))

# Predict
predictions = model.predict(X_test)

# Calculate the MSE. MSE = RSS/n_test
tss,rss = TSS_RSS(y_test,predictions)
MSE.append(rss[0])

# Print the mean MSE value
print(np.mean(MSE))

```

4597328547.297892

```

In [250]: X = housePrice['LotArea']
          y = housePrice['SalePrice']

MSE = []

for i in range(len(X)):
    X_train = X.copy().values.reshape(-1,1)
    y_train = y.copy().values.reshape(-1,1)

    X_test = X_train[i]
    X_test = X_test.reshape(1,-1)
    X_train = np.delete(X_train,i)

    y_test = y_train[i]
    y_test = y_test.reshape(1,-1)
    y_train = np.delete(y_train,i)

    # Train
    model = LinearRegression()

    # Fit
    model.fit(X_train.reshape(-1,1),y_train.reshape(-1,1))

    # Predict
    predictions = model.predict(X_test)

```

```

        # MSE
        tss,rss = TSS_RSS(y_test,predictions)
        MSE.append(rss[0])

    print(np.mean(MSE))

5954196196.345752

In [251]: from sklearn.cross_validation import cross_val_score
          from sklearn.metrics import mean_squared_error

In [253]: X = housePrice['LotArea']
          y = housePrice['SalePrice']

          linregCVScores = cross_val_score(LinearRegression(),X.values.reshape(-1,1),y.values.)
          -linregCVScores.mean()

Out[253]: 5954196196.345752

```

Let's observe now which approach (value of k in k-fold cross validation) predicts the test MSE best. We split our train and test data. Then estimate the test MSE using the training data.

```

In [448]: import math

          datasetMSEEstimatek_10 = []
          datasetMSEEstimatek_20 = []
          datasetMSEEstimatek_100 = []
          datasetMSEActual = []

          # The predictor and response
          X = housePrice['LotArea'].values.reshape(-1,1)
          y = housePrice['SalePrice'].values.reshape(-1,1)

          for j in range(500):
              X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_

              linregCVScores = cross_val_score(LinearRegression(),X_train.reshape(-1,1),y_train)
              datasetMSEEstimatek_10.append(-linregCVScores.mean())

              linregCVScores = cross_val_score(LinearRegression(),X_train.reshape(-1,1),y_train)
              datasetMSEEstimatek_20.append(-linregCVScores.mean())

              linregCVScores = cross_val_score(LinearRegression(),X_train.reshape(-1,1),y_train)
              datasetMSEEstimatek_100.append(-linregCVScores.mean())

          model = LinearRegression()

```

```

model.fit(X_train,y_train)
predictions = model.predict(X_test)
datasetMSEActual.append(mean_squared_error(y_test,predictions))

if j%50 == 0:
    print('Step = {}'.format(j))

print('The mean MSE Estimation using K-fold CV with k = 10 is : {}'.format(np.mean(d
print('The mean MSE Estimation using K-fold CV with k = 20 is : {}'.format(np.mean(d
print('The mean MSE Estimation using K-fold CV with k = 100 is : {}'.format(np.mean(
print('The actual MSE on this data set is : {}'.format(np.mean(datasetMSEActual)))

fig,axes = plt.subplots(nrows = 2,ncols = 2,sharex=True)
fig.set_size_inches(20,10)

axes[0][0].hist(list(map(math.log,datasetMSEEstimatek_10)))
axes[0][0].set_title('CV with k = 10')

axes[0][1].hist(list(map(math.log,datasetMSEEstimatek_20)))
axes[0][1].set_title('CV with k = 20')

axes[1][0].hist(list(map(math.log,datasetMSEEstimatek_100)))
axes[1][0].set_title('CV with k = 100')

axes[1][1].hist(list(map(math.log,datasetMSEActual)))
axes[1][1].set_title('Averaged Actual MSE over different train/test splits')

```

```

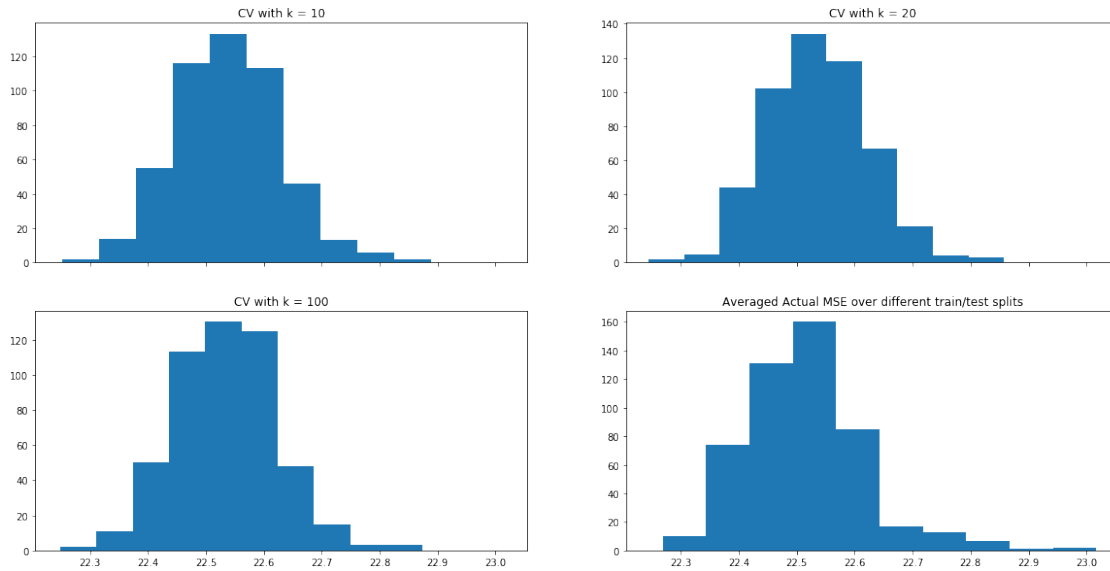
Step = 0
Step = 50
Step = 100
Step = 150
Step = 200
Step = 250
Step = 300
Step = 350
Step = 400
Step = 450
The mean MSE Estimation using K-fold CV with k = 10 is : 6170090398.833329
The mean MSE Estimation using K-fold CV with k = 20 is : 6161875851.874799
The mean MSE Estimation using K-fold CV with k = 100 is : 6146620803.335432
The actual MSE on this data set is : 6046571007.118181

```

```

Out[448]: Text(0.5,1,'Averaged Actual MSE over different train/test splits')

```

```
In [447]: import math
```

```
datasetMSEEstimatek_10 = []
datasetMSEEstimatek_20 = []
datasetMSEEstimatek_100 = []
datasetMSEActual = []
```

```
# The predictor and response
```

```
X = housePrice['YearBuilt'].values.reshape(-1,1)
```

```
y = housePrice['SalePrice'].values.reshape(-1,1)
```

```
for j in range(500):
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_s
```

```
    linregCVScores = cross_val_score(LinearRegression(),X_train.reshape(-1,1),y_train
```

```
    datasetMSEEstimatek_10.append(-linregCVScores.mean())
```

```
    linregCVScores = cross_val_score(LinearRegression(),X_train.reshape(-1,1),y_train
```

```
    datasetMSEEstimatek_20.append(-linregCVScores.mean())
```

```
    linregCVScores = cross_val_score(LinearRegression(),X_train.reshape(-1,1),y_train
```

```
    datasetMSEEstimatek_100.append(-linregCVScores.mean())
```

```
    model = LinearRegression()
```

```
    model.fit(X_train,y_train)
```

```
    predictions = model.predict(X_test)
```

```

datasetMSEActual.append(mean_squared_error(y_test,predictions))

if j%50 == 0:
    print('Step = {}'.format(j))

print('The mean MSE Estimation using K-fold CV with k = 10 is : {}'.format(np.mean(d
print('The mean MSE Estimation using K-fold CV with k = 20 is : {}'.format(np.mean(d
print('The mean MSE Estimation using K-fold CV with k = 100 is : {}'.format(np.mean(
print('The actual MSE on this data set is : {}'.format(np.mean(datasetMSEActual)))

fig,axes = plt.subplots(nrows = 2,ncols = 2,sharex=True)
fig.set_size_inches(20,10)

axes[0][0].hist(list(map(math.log,datasetMSEEstimatek_10)))
axes[0][0].set_title('CV with k = 10')

axes[0][1].hist(list(map(math.log,datasetMSEEstimatek_20)))
axes[0][1].set_title('CV with k = 20')

axes[1][0].hist(list(map(math.log,datasetMSEEstimatek_100)))
axes[1][0].set_title('CV with k = 100')

axes[1][1].hist(list(map(math.log,datasetMSEActual)))
axes[1][1].set_title('Averaged Actual MSE over different train/test splits')

```

```

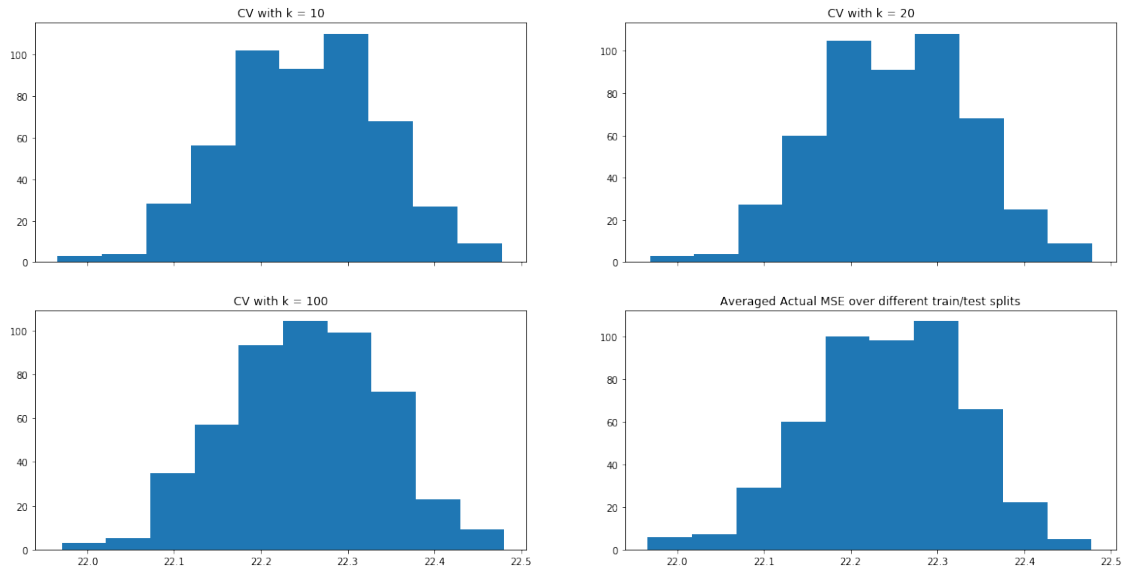
Step = 0
Step = 50
Step = 100
Step = 150
Step = 200
Step = 250
Step = 300
Step = 350
Step = 400
Step = 450
The mean MSE Estimation using K-fold CV with k = 10 is : 4622083078.925203
The mean MSE Estimation using K-fold CV with k = 20 is : 4621267854.562424
The mean MSE Estimation using K-fold CV with k = 100 is : 4620965999.249676
The actual MSE on this data set is : 4587675012.9899845

```

```

Out[447]: Text(0.5,1,'Averaged Actual MSE over different train/test splits')

```



We can see above that the MSE when we choose the 'LotArea' predictor is not as good as using 'YearBuilt'. So in this case we choose 'YearBuilt' over 'LotArea' to include in our linear regression model.

It can be seen that there is a general pattern in the above when comparing the MSE estimates from varying blocks (k) in Cross Validation. Namely, the larger k is, the more blocks we use to split the data up and the less portion of the data there is for the test set and the more iteration that is required per Cross Validation. For example, suppose $n = 1000$, when $k = 10$ we have 10 blocks with 100 observations per block. So we fit the model 10 times, each time leaving out a different k block when training. When comparing this with the case where $k = 100$, we have 100 blocks each with 10 observations. This means that the training set for each of these model fits is a lot closer to reality in that we will be using the entire dataset to fit the model. However, this comes at a computational cost, in this case we would need to run a model fit the predict 100 times instead of 10.

Depending on the computational cost of the model used, we may choose a smaller value of $k = 10$ when comparing the same model but with different parameters.

We can try out all the predictors and choose the one that minimises the mean squared errors. This can be done by using a loop as below.

```
In [468]: a=X.drop('SalePrice',axis=1)
          a.columns
```

```
Out[468]: Index(['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
                'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
                '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
                'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
                'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
                'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold'],
                dtype='object')
```

```

In [482]: # Run through each model in the correct order and run CV on it and save the best CV
bestMeanCV = -1
bestMeanCVModel = []

X = housePrice.drop('SalePrice',axis=1)

# y is the response variable
y = housePrice['SalePrice']

for i in X.columns:
    # First set X to be the full set of remaining parameters
    X = housePrice.loc[:,i]

    linregCVScores = cross_val_score(LinearRegression(),X.values.reshape(-1,1),y,score_func=neg_mean_squared_error)

    if bestMeanCV > -linregCVScores.mean():
        bestMeanCV = -linregCVScores.mean()
        bestMeanCVModel = i
    elif bestMeanCV == -1:
        bestMeanCV = -linregCVScores.mean()
        bestMeanCVModel = i

print('The final best model is {} and its TEST MSE is {}'.format(bestMeanCVModel,bestMeanCV))

```

The final best model is OverallQual and its TEST R^2 is 2371260934.069027

We can then iterate through the predictors adding it to the model each time in order to improve the test MSE of the model. For instance, in the above, we have selected as the first predictor in our model, the predictor 'OverallQual'. Next, we cycle through all the remaining predictors to include in our model along with 'OverallQual' and repeat. The final result will be a list of all predictors in the order they were added. Once we get to a point where adding another predictor to the model does not improve the test MSE, then we stop there.

```

In [517]: # Run through each model in the correct order and run CV on it and save the best CV
bestMeanCV = -1
bestMeanCVModel = []
oldArraySize = 0

X = housePrice.drop('SalePrice',axis=1)

columnsArray = X.columns

# y is the response variable
y = housePrice['SalePrice']

while oldArraySize != len(X):
    bestPredictor = ''

```

```

oldArraySize = len(X.columns)
for i in columnsArray:
    thisModel = bestMeanCVModel.copy()
    thisModel.append(i)
    # First set X to be the full set of remaining parameters
    x = X.loc[:,thisModel]

    if len(x.columns) == 1:
        linregCVScores = cross_val_score(LinearRegression(),x.values.reshape(-1,1),y,scoring='neg_mean_squared_error')
    else:
        linregCVScores = cross_val_score(LinearRegression(),x,y,scoring='neg_mean_squared_error')

    if bestMeanCV > -linregCVScores.mean():
        bestMeanCV = -linregCVScores.mean()
        bestPredictor = i
    elif bestMeanCV == -1:
        bestMeanCV = -linregCVScores.mean()
        bestPredictor = i

if bestPredictor not in columnsArray:
    break

columnsArray.drop(bestPredictor)
bestMeanCVModel.append(bestPredictor)
print('{} was added with test MSE {}'.format(bestMeanCVModel[-1],bestMeanCV))

print('The final best model is {} and its TEST MSE is {}'.format(bestMeanCVModel,bestMeanCV))

```

```

OverallQual was added with test MSE 2371260934.069027
GrLivArea was added with test MSE 1821343747.225343
BsmtFinSF1 was added with test MSE 1653396814.390045
GarageCars was added with test MSE 1522575852.8088708
YearRemodAdd was added with test MSE 1477506784.3227725
LotArea was added with test MSE 1445259871.6905437
MasVnrArea was added with test MSE 1418244120.6007636
KitchenAbvGr was added with test MSE 1399446462.6115096
1stFlrSF was added with test MSE 1376812086.0548677
YearBuilt was added with test MSE 1366762325.3833976
OverallCond was added with test MSE 1352021476.9079423
ScreenPorch was added with test MSE 1346347855.6538568
WoodDeckSF was added with test MSE 1339278365.3061795
TotRmsAbvGrd was added with test MSE 1334799185.148417
BedroomAbvGr was added with test MSE 1315922220.7805562
EnclosedPorch was added with test MSE 1315305925.97893
PoolArea was added with test MSE 1314438332.6508195
LotArea was added with test MSE 1314390836.7395625
OverallCond was added with test MSE 1313986931.202222

```

The final best model is ['OverallQual', 'GrLivArea', 'BsmtFinSF1', 'GarageCars', 'YearRemodAdd']

Our final model is now contained in bestMeanCVModel.

```
In [141]: import time
```

```
now = time.time()
```

```
time.sleep(2)
```

```
print(time.time()-now)
```

```
2.0045924186706543
```

```
In [231]: np.delete(a,2)
```

```
Out[231]: array([5, 4, 4])
```

```
In [232]: a
```

```
Out[232]: array([5, 4, 5, 4])
```

1.4 Appendix

1.4.1 A1 - $(2n)(2 \sum_{i=1}^n x_i^2) - (2 \sum_{i=1}^n x_i)^2 > 0$ for non-trivial X

Statement: $(2n)(2 \sum_{i=1}^n x_i^2) - (2 \sum_{i=1}^n x_i)^2 > 0 \forall n > 1$

Proof: We prove this by induction on n . If $n = 1$, we have $(2n)(2 \sum_{i=1}^n x_i^2) - (2 \sum_{i=1}^n x_i)^2 = 0$, but this is not what we want.

Let $n = 2 > 1$. Then

$$\begin{aligned} (2n)(2 \sum_{i=1}^n x_i^2) - (2 \sum_{i=1}^n x_i)^2 &= 2x_1^2 + 2x_2^2 - (x_1 + x_2)^2 \\ &= 2x_1^2 + 2x_2^2 - x_1^2 - x_2^2 - 2x_1x_2 = x_1^2 + x_2^2 - 2x_1x_2 = (x_1 - x_2)^2 > 0 \end{aligned}$$

So we have proved the assertion for $n = 2$.

Let us prove the statement for $n+1$ assuming it is true for n .

i.e. Assume $n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2 > 0$

Then

$$\begin{aligned} (n+1) \sum_{i=1}^{n+1} x_i^2 - (\sum_{i=1}^{n+1} x_i)^2 &= (n+1) [\sum_{i=1}^n x_i^2 + x_{n+1}^2] - (\sum_{i=1}^n x_i + x_{n+1})^2 \\ &= [n \sum_{i=1}^n x_i^2 + \sum_{i=1}^n x_i^2 + (n+1)x_{n+1}^2] - (\sum_{i=1}^n x_i)^2 - x_{n+1}^2 + 2x_{n+1} \sum_{i=1}^n x_i \\ &= n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2 + \sum_{i=1}^n x_i^2 + (n+1)x_{n+1}^2 - x_{n+1}^2 + 2x_{n+1} \sum_{i=1}^n x_i \end{aligned}$$

by the assumption for n we have

$$> \sum_{i=1}^n x_i^2 + (n+1)x_{n+1}^2 - x_{n+1}^2 + 2x_{n+1} \sum_{i=1}^n x_i$$

by the assumption for n that $\sum_{i=1}^n x_i^2 > \frac{1}{n}(\sum_{i=1}^n x_i)^2$ we have

$$\begin{aligned} &> \frac{1}{n}(\sum_{i=1}^n x_i)^2 + (n+1)x_{n+1}^2 - x_{n+1}^2 + 2x_{n+1} \sum_{i=1}^n x_i = \frac{1}{n}(\sum_{i=1}^n x_i)^2 + nx_{n+1}^2 + 2x_{n+1} \sum_{i=1}^n x_i \\ &= \frac{1}{n}[(\sum_{i=1}^n x_i)^2 + n^2x_{n+1}^2 + 2nx_{n+1} \sum_{i=1}^n x_i] \\ &= \frac{1}{n} \left(\sum_{i=1}^n x_i + nx_{n+1} \right)^2 > 0 \end{aligned}$$

This proves the statement. This assumes that at least one X_i is non-zero.

1.4.2 A2 - Maximum Likelihood Estimation (MLE)

Let's assume that there is a linear relationship between the response and predictor variables and that any discrepancy is due to random noise, this is expressed as

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

where the errors are normally distributed, $\epsilon \sim N(0, \sigma^2)$. Then, the response variable given the data are normally distributed

$$Y_i | X_i \sim N(\beta_0 + \beta_1 X_i, \sigma^2)$$

where the mean or expectation is

$$E[Y_i | X_i] = E[\beta_0 + \beta_1 X_i + \epsilon_i] = E[\beta_0] + E[\beta_1 X_i] + E[\epsilon_i] = \beta_0 + \beta_1 X_i$$

The probability density function for Y_i is then

$$P(Y_i = y_i | X_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}[y_i - (\beta_0 + \beta_1 x_i)]^2\right)$$

Then, if the Y_i observations are independent of each other, we have that the likelihood of $\beta = (\beta_0, \beta_1)$ (the probability of observing this data given these parameters) is

$$L(\beta) = P(Y|\beta, X) = P(Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n | \beta, X) = P(Y_1 = y_1 | \beta, X_1)P(Y_2 = y_2 | \beta, X_2) \dots P(Y_n = y_n | \beta, X_n)$$

where the last equality is due to the independence of each observation and that Y_i is only dependent on β and X_i . Using the probability density function above, this becomes

$$L(\beta) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}[y_i - (\beta_0 + \beta_1 x_i)]^2\right) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^n [y_i - (\beta_0 + \beta_1 x_i)]^2\right)$$

Therefore, maximising this function with respect to β , corresponds to finding values for β which maximises the probability of obtaining this response data given the predictor data. Instead of working with this equation as it stands, we note that the right hand side of the above equation is positive for all values of β and x_i . This means that we can apply a handy trick in that since the log function is a monotonically increasing function, maximising $\log(L(\beta))$ is the same as maximising $L(\beta)$. Due to the existence of \exp in $L(\beta)$, we may choose the natural logarithm so that the exponential disappears (we will still denote this as \log).

$$l(\beta) = \log(L(\beta)) = \log \left(\left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \right) - \frac{1}{2\sigma^2} \sum_{i=1}^n [y_i - (\beta_0 + \beta_1 x_i)]^2$$

Since the first term on the right-hand side is indifferent to the choice of β , maximising $l(\beta)$ corresponds to maximising the last term on the right-hand side

$$\max_{\beta} l(\beta) = \max_{\beta} \left(-\frac{1}{2\sigma^2} \sum_{i=1}^n [y_i - (\beta_0 + \beta_1 x_i)]^2 \right)$$

which is equivalent to

$$\max_{\beta} l(\beta) = \min_{\beta} \left(\sum_{i=1}^n [y_i - (\beta_0 + \beta_1 x_i)]^2 \right) = \min_{\beta} RSS$$

where $RSS = \sum_{i=1}^n \epsilon_i^2$. Note that for multiple predictors (p predictors), the above becomes

$$\max_{\beta} l(\beta) = \min_{\beta} \left(\sum_{i=1}^n \left[y_i - \left(\beta_0 + \sum_{j=1}^p \beta_j x_{ij} \right) \right]^2 \right) = \min_{\beta} RSS$$

where x_{ij} is the j^{th} predictor for observation i .