Multiple Linear Regression with Python and Scikit-Learn

This project is about Multiple Linear Regression which is a machine learning algorithm. I build a multiple linear regression model to estimate the relative cpu performance of computer hardware dataset.

Table of contents

The contents of this project are divided into various categories which are given as follows:-

1. Introduction
2. Linear regression intuition
3. Independent and dependent variables
4. Assumptions of linear regression
5. The dataset description
6. The problem statement
7. Import the Python libraries
8. Import the dataset
9. Exploratory Data Analysis
    - Explore types of variables
    - Estimate correlation coefficients
    - Correlation heat map
10. Detect problems within variables
    - Detect missing values
    - Outliers in discrete variables
    - Number of labels – cardinality
11. Linear Regression modeling
    - Divide the dataset into categorical and numerical variables
    - Select the predictor and target variables

- Create separate train and test sets
- Feature Scaling
- Fit the Linear Regression model

12. Predicting the results
    - Predicting the test set results
    - Predicting estimated relative CPU performance values
13. Model slope and intercept terms
14. Evaluate model performance
    - RMSE (Root Mean Square Error)
    - R2 Score
    - Overfitting or Underfitting
    - Cross validation
    - Residual analysis
    - Normality test (Q-Q Plot)
15. Conclusion

1. Introduction

In this project, I build a multiple linear regression model to estimate the relative cpu performance of computer hardware dataset. Relative cpu performance of the computer hardware is described in terms of machine cycle time, main memory, cache memory and minimum and maximum channels as given in the dataset.

I discuss the basics and assumptions of linear regression. I also discuss the advantages and disadvantages and common pitfalls of linear regression. I present the implementation in Python programming language using Scikit-learn. Scikit-learn is the popular machine learning library of Python programming language. I also discuss various tools to evaluate the linear regression model performance.

2. Multiple linear regression intuition

Linear Regression is a machine learning algorithm which is used to establish the linear relationship between dependent and one or more independent variables. This technique is applicable for supervised learning regression problems where we try to predict a continuous variable. Linear Regression can be further classified into two types – Simple and Multiple Linear Regression.

I have discussed the linear regression intuition in detail in the readme document.

In this project, I employ Multiple Linear Regression technique where I have one dependent variable and more than one independent variables.

3. Independent and dependent variables

In this project, I refer Independent variable as Feature variable and Dependent variable as Target variable. These variables are also recognized by different names as follows: -

Independent variable

Independent variable is also called Input variable and is denoted by X. In practical applications, independent variable is also called Feature variable or Predictor variable. We can denote it as: -

Independent or Input variable (X) = Feature variable = Predictor variable

Dependent variable

Dependent variable is also called Output variable and is denoted by y. Dependent variable is also called Target variable or Response variable. It can be denoted it as follows: -

Dependent or Output variable (y) = Target variable = Response variable

4. Assumptions of Linear Regression

The Linear Regression model is based on several assumptions which are as follows:-

1. Linear relationship
2. Multivariate normality
3. No or little multi-collinearity
4. No auto-correlation in error terms
5. Homoscedasticity

I have described these assumptions in more detail in readme document.

5. Dataset description

Now, we should get to know more about the dataset. It is a computer hardware dataset. The dataset consists of information about the computer vendors selling computers, model name of computers and various attributes to estimate the relative performance of CPU. The dataset can be found at the following url –

https://archive.ics.uci.edu/ml/datasets/Computer+Hardware

The dataset description will help us to know more about the data.

Dataset description is given as follows:-

1. vendor name: 30 (adviser, amdahl,apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec, dg, formation, four-phase, gould, honeywell, hp, ibm, ipl, magnuson, microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens, sperry, sratus, wang)
2. Model Name: many unique symbols
3. MYCT: machine cycle time in nanoseconds (integer)
4. MMIN: minimum main memory in kilobytes (integer)
5. MMAX: maximum main memory in kilobytes (integer)
6. CACH: cache memory in kilobytes (integer)
7. CHMIN: minimum channels in units (integer)
8. CHMAX: maximum channels in units (integer)
9. PRP: published relative performance (integer)
10. ERP: estimated relative performance from the original article (integer)

6. The problem statement

A machine learning model is built with the aim of solving a problem. So, first of all I have to define the problem to be solved in this project.

As described earlier, the problem is to estimate the relative CPU performance of computer hardware dataset. Relative CPU performance of the computer hardware is described in terms of machine cycle time, main memory, cache memory and minimum and maximum channels as given in the dataset.

So, let's get started. I will start by importing the required Python libraries.

7. Import the Python libraries

```
# import required Python libraries

# to handle datasets
import numpy as np
import pandas as pd

# for plotting
import matplotlib.pyplot as plt
% matplotlib inline
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')
```

8. Import the dataset

```
# import the dataset

filename = "c:/datasets/machine.data.csv"

df = pd.read_csv(filename, header = None)
```

9. Exploratory Data Analysis

Now, I will perform Exploratory Data Analysis. It provides useful insights into the dataset which is important for further analysis.

First of all, we should check the dimensions of the dataframe as follows:-

```
# view the dimensions of dataframe df

print("Shape of dataframe df: {}".format(df.shape))
```

```
Shape of dataframe df: (209, 10)
```

We can see that there are 209 rows and 10 columns in the dataset. Next, we should get an insight about the dataset.

The **df.head()** function helps us to visualize the first 5 rows of the dataset.

```
# view the top five rows of dataframe df

df.head()
```

	0	1	2	3	4	5	6	7	8	9
0	adviser	32/60	125	256	6000	256	16	128	198	199
1	amdahl	470v/7	29	8000	32000	32	8	32	269	253
2	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
3	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
4	amdahl	470v/7c	29	8000	16000	32	8	16	132	132

We can see that the column names are from 0 to 9. They should be descriptive. So, we should rename them as follows:-

```
# rename columns of dataframe df

col_names = ['Vendor Name','Model Name', 'MYCT', 'MMIN', 'MMAX',
'CACH','CHMIN', 'CHMAX', 'PRP', 'ERP' ]

df.columns = col_names
```

We should now check that the columns have appropriate names.

```
# view the top five rows of dataframe with column names renamed

df.head()
```

```
  Vendor Name Model Name  MYCT  MMIN   MMAX  CACH  CHMIN  CHMAX  PRP
ERP
0     adviser      32/60   125   256   6000   256     16    128  198
199
1      amdahl     470v/7    29  8000  32000    32      8     32  269
253
2      amdahl    470v/7a    29  8000  32000    32      8     32  220
253
3      amdahl    470v/7b    29  8000  32000    32      8     32  172
253
4      amdahl    470v/7c    29  8000  16000    32      8     16  132
132
```

Explore types of variables

In this section, I will explore the types of variables in the dataset.

First, let's view a concise summary of the dataframe with **df.info()** method.

```
# view dataframe summary

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209 entries, 0 to 208
Data columns (total 10 columns):
Vendor Name    209 non-null object
Model Name     209 non-null object
MYCT           209 non-null int64
MMIN           209 non-null int64
MMAX           209 non-null int64
CACH           209 non-null int64
CHMIN          209 non-null int64
CHMAX          209 non-null int64
PRP            209 non-null int64
ERP            209 non-null int64
dtypes: int64(8), object(2)
memory usage: 16.4+ KB
```

We can see that there are categorical and numerical variables in the dataset. Numerical variables have data types int64 and categorical variables are those of type object.

First, let's explore the categorical variables.

```
# find categorical variables
```

```
categorical = [col for col in df.columns if df[col].dtype=='O']
print('There are {} categorical variables'.format(len(categorical)))
```

```
There are 2 categorical variables
```

```
# view the categorical variables

print(categorical)
```

```
['Vendor Name', 'Model Name']
```

So, there are two categorical variables - **Vendor Name** and **Model Name** in the dataset.

Let's explore more about them.

```
# view the top five rows of categorical variables

df[categorical].head()
```

```
  Vendor Name Model Name
0     adviser      32/60
1      amdahl     470v/7
2      amdahl    470v/7a
3      amdahl    470v/7b
4      amdahl    470v/7c
```

```
# exploring the categories in Vendor Name column

df['Vendor Name'].value_counts()
```

```
ibm              32
nas              19
ncr              13
honeywell        13
sperry           13
siemens          12
amdahl            9
cdc               9
burroughs         8
harris            7
dg                7
hp                7
ipl               6
c.r.d             6
dec               6
magnuson          6
formation         5
cambex            5
prime             5
nixdorf           3
gould             3
```

```
perkin-elmer     3
apollo           2
wang             2
bti              2
basf             2
microdata        1
adviser          1
four-phase       1
sratus           1
Name: Vendor Name, dtype: int64
```

ibm is the most frequent category in the **Vendor Name** column.

Next, let's explore the **Model Name** column.

```
print('Number of unique Model Names: ', len(df['Model
Name'].unique()))
print('Number  of instances of models: ', len(df))
```

```
Number of unique Model Names:  209
Number  of instances of models:  209
```

We can see that **Model Name** is a unique identifier for each of the computer models. Thus this is not a variable that we can use to predict the estimated relative performance of computer models. So, we should not use this column for model building.

Now, let's explore the numerical variables.

```
# find numerical variables

numerical = [col for col in df.columns if df[col].dtype!='O']
print('There are {} numerical variables'.format(len(numerical)))
```

```
There are 8 numerical variables
```

```
# view numerical variables

print(numerical)
```

```
['MYCT', 'MMIN', 'MMAX', 'CACH', 'CHMIN', 'CHMAX', 'PRP', 'ERP']
```

So, there are eight numerical variables in the dataset. Let's explore more about them.

```
# view the top 5 rows of numerical variables

df[numerical].head()
```

```
   MYCT  MMIN   MMAX  CACH  CHMIN  CHMAX  PRP  ERP
0   125   256   6000   256     16    128  198  199
1    29  8000  32000    32      8     32  269  253
```

```
2    29  8000  32000    32      8     32  220  253
3    29  8000  32000    32      8     32  172  253
4    29  8000  16000    32      8     16  132  132
```

We can see that we have eight numerical variables in the dataset. All the eight numerical variables are of discrete type.

On closer inspection, we find that **PRP** is a redundant column in the dataframe. It denotes **published relative performance**. Our target is to predict **estimated relative performance**. So, we should delete **PRP** from the dataframe.

Summary : types of variables

- There are 2 categorical variables and 8 numerical variables.
- The 2 categorical variables, **Vendor Name** and **Model Name** are 2 non-predictive attributes as given in the dataset description. So, I do not use them for model building.
- All of the 8 numerical variables are of discrete type.
- Out of the 8 numerical variables, **PRP** is the linear regression's guess. It is redundant column. I do not use it for model building.
- **ERP** (estimated relative performance is the goal field). It is the target variable.

Estimate correlation coefficients

Our dataset is very small. So, we can compute the standard correlation coefficient (also called Pearson's r) between every pair of attributes.

We can compute it using the `df.corr()` method as follows:-

```
# estimate correlation coefficients

pd.options.display.float_format = '{:,.4f}'.format
corr_matrix = df.corr()
corr_matrix
```

```
         MYCT    MMIN    MMAX    CACH   CHMIN   CHMAX     PRP     ERP
MYCT   1.0000 -0.3356 -0.3786 -0.3210 -0.3011 -0.2505 -0.3071 -0.2884
MMIN  -0.3356  1.0000  0.7582  0.5347  0.5172  0.2669  0.7949  0.8193
MMAX  -0.3786  0.7582  1.0000  0.5380  0.5605  0.5272  0.8630  0.9012
CACH  -0.3210  0.5347  0.5380  1.0000  0.5822  0.4878  0.6626  0.6486
CHMIN -0.3011  0.5172  0.5605  0.5822  1.0000  0.5483  0.6089  0.6106
CHMAX -0.2505  0.2669  0.5272  0.4878  0.5483  1.0000  0.6052  0.5922
PRP   -0.3071  0.7949  0.8630  0.6626  0.6089  0.6052  1.0000  0.9665
ERP   -0.2884  0.8193  0.9012  0.6486  0.6106  0.5922  0.9665  1.0000
```

```
corr_matrix['ERP'].sort_values(ascending=False)
```

```
ERP      1.0000
PRP      0.9665
MMAX     0.9012
MMIN     0.8193
CACH     0.6486
CHMIN    0.6106
CHMAX    0.5922
MYCT    -0.2884
Name: ERP, dtype: float64
```

Interpretation of correlation coefficient

The correlation coefficient ranges from -1 to +1.

When it is close to +1, this signifies that there is a strong positive correlation. So, we can see that there is a strong positive correlation between `ERP` and `MMAX`.

When it is clsoe to -1, it means that there is a strong negative correlation. So, there is a small negative correlation between `ERP` and `MYCT`.

Correlation heat map

```
plt.figure(figsize=(16,10))
plt.title('Correlation of Attributes with ERP')
a = sns.heatmap(corr_matrix, square=True, annot=True, fmt='.2f',
linecolor='white')
a.set_xticklabels(a.get_xticklabels(), rotation=90)
a.set_yticklabels(a.get_yticklabels(), rotation=30)
plt.show()
```

We can see that **ERP** is positively correlated with **MMIN**, **MMAX**, **CACH**, **CHMIN** and **CHMAX**.

Also, there is a strong positive correlation between **ERP** and **MMIN** and also between **ERP** and **MMAX**.

10. Detect problems within variables

Detect missing values

```
# let's visualise the number of missing values
df.isnull().sum()
```

```
Vendor Name    0
Model Name     0
MYCT           0
MMIN           0
MMAX           0
CACH           0
```

```
CHMIN          0
CHMAX          0
PRP            0
ERP            0
dtype: int64
```

We can confirm that there are no missing values in the dataset.

Outliers in discrete variables

```
# let's view the summary statistics of the dataset
df.describe()
```

```
             MYCT         MMIN         MMAX      CACH     CHMIN     CHMAX  \
count    209.0000     209.0000     209.0000 209.0000 209.0000 209.0000
mean     203.8230  2,867.9809 11,796.1531  25.2057   4.6986  18.2679
std      260.2629  3,878.7428 11,726.5644  40.6287   6.8163  25.9973
min       17.0000      64.0000     64.0000   0.0000   0.0000   0.0000
25%       50.0000     768.0000  4,000.0000   0.0000   1.0000   5.0000
50%      110.0000  2,000.0000  8,000.0000   8.0000   2.0000   8.0000
75%      225.0000  4,000.0000 16,000.0000  32.0000   6.0000  24.0000
max    1,500.0000 32,000.0000 64,000.0000 256.0000  52.0000 176.0000

              PRP         ERP
count    209.0000    209.0000
mean     105.6220     99.3301
std      160.8307    154.7571
min        6.0000     15.0000
25%       27.0000     28.0000
50%       50.0000     45.0000
75%      113.0000    101.0000
max    1,150.0000 1,238.0000
```

```
# outlies in discrete variables

for var in ['MYCT', 'MMIN', 'MMAX', 'CACH', 'CHMIN', 'CHMAX']:
    print(df[var].value_counts() / np.float(len(df)))
    print()
```

```
50     0.1196
140    0.0431
300    0.0383
26     0.0383
38     0.0335
320    0.0335
56     0.0335
180    0.0335
800    0.0287
75     0.0287
105    0.0287
```

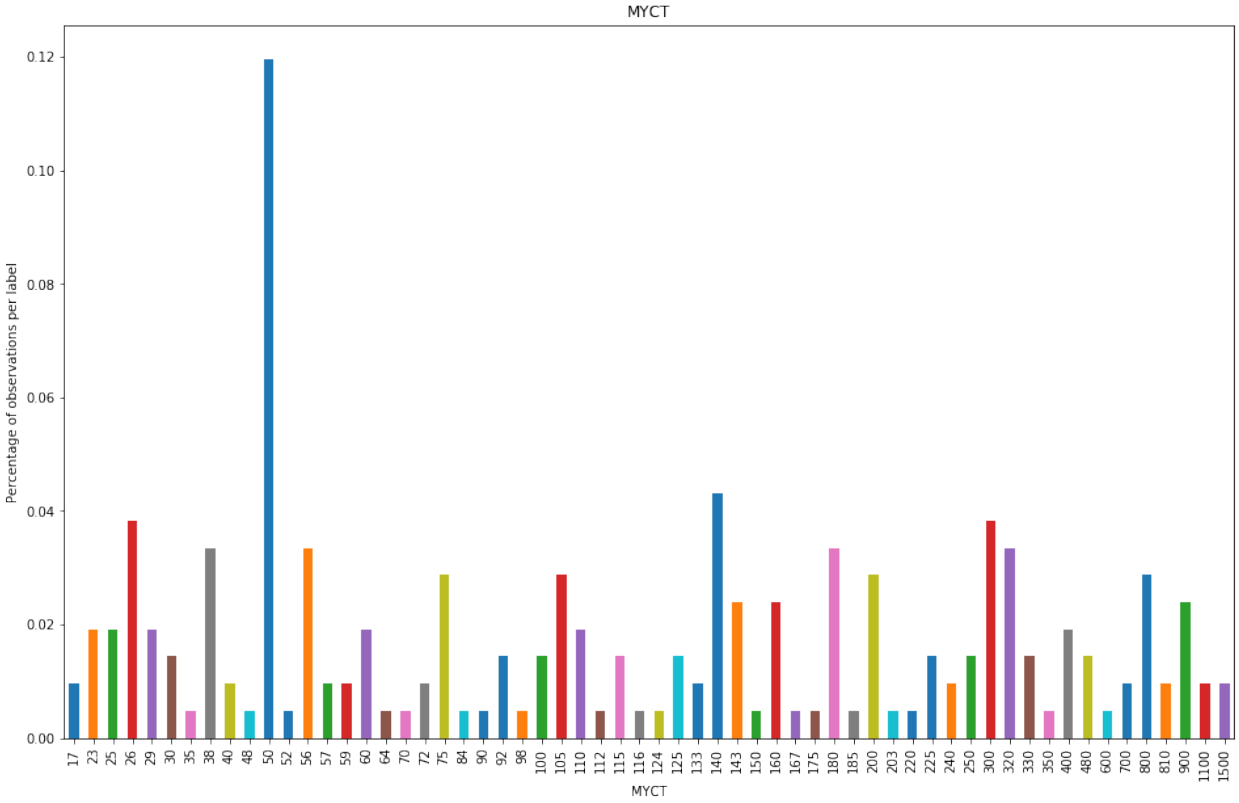
```
200    0.0287
143    0.0239
900    0.0239
160    0.0239
400    0.0191
60     0.0191
29     0.0191
25     0.0191
23     0.0191
110    0.0191
92     0.0144
100    0.0144
250    0.0144
115    0.0144
125    0.0144
30     0.0144
480    0.0144
225    0.0144
330    0.0144
810    0.0096
1500   0.0096
72     0.0096
40     0.0096
57     0.0096
59     0.0096
17     0.0096
133    0.0096
1100   0.0096
240    0.0096
700    0.0096
64     0.0048
220    0.0048
203    0.0048
185    0.0048
175    0.0048
167    0.0048
35     0.0048
150    0.0048
116    0.0048
124    0.0048
70     0.0048
48     0.0048
112    0.0048
52     0.0048
98     0.0048
350    0.0048
600    0.0048
84     0.0048
90     0.0048
```

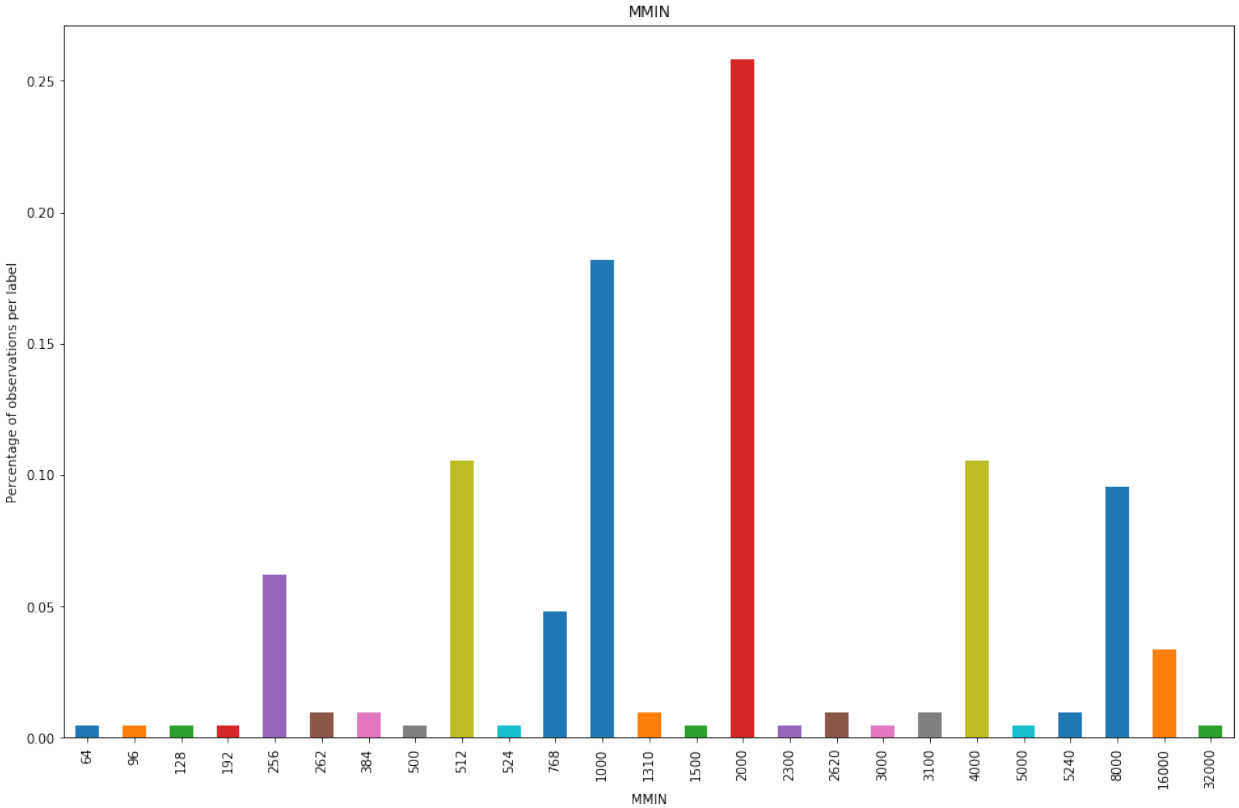
```
Name: MYCT, dtype: float64

2000    0.2584
1000    0.1818
4000    0.1053
512     0.1053
8000    0.0957
256     0.0622
768     0.0478
16000   0.0335
262     0.0096
3100    0.0096
5240    0.0096
1310    0.0096
2620    0.0096
384     0.0096
32000   0.0048
1500    0.0048
524     0.0048
64      0.0048
192     0.0048
96      0.0048
3000    0.0048
500     0.0048
5000    0.0048
128     0.0048
2300    0.0048
Name: MMIN, dtype: float64

8000    0.2057
16000   0.1675
4000    0.1579
32000   0.1100
2000    0.0813
12000   0.0478
1000    0.0335
6000    0.0287
3000    0.0239
5000    0.0239
24000   0.0191
64000   0.0191
6200    0.0144
2620    0.0096
10480   0.0096
512     0.0096
20970   0.0096
768     0.0048
64      0.0048
6300    0.0048
```

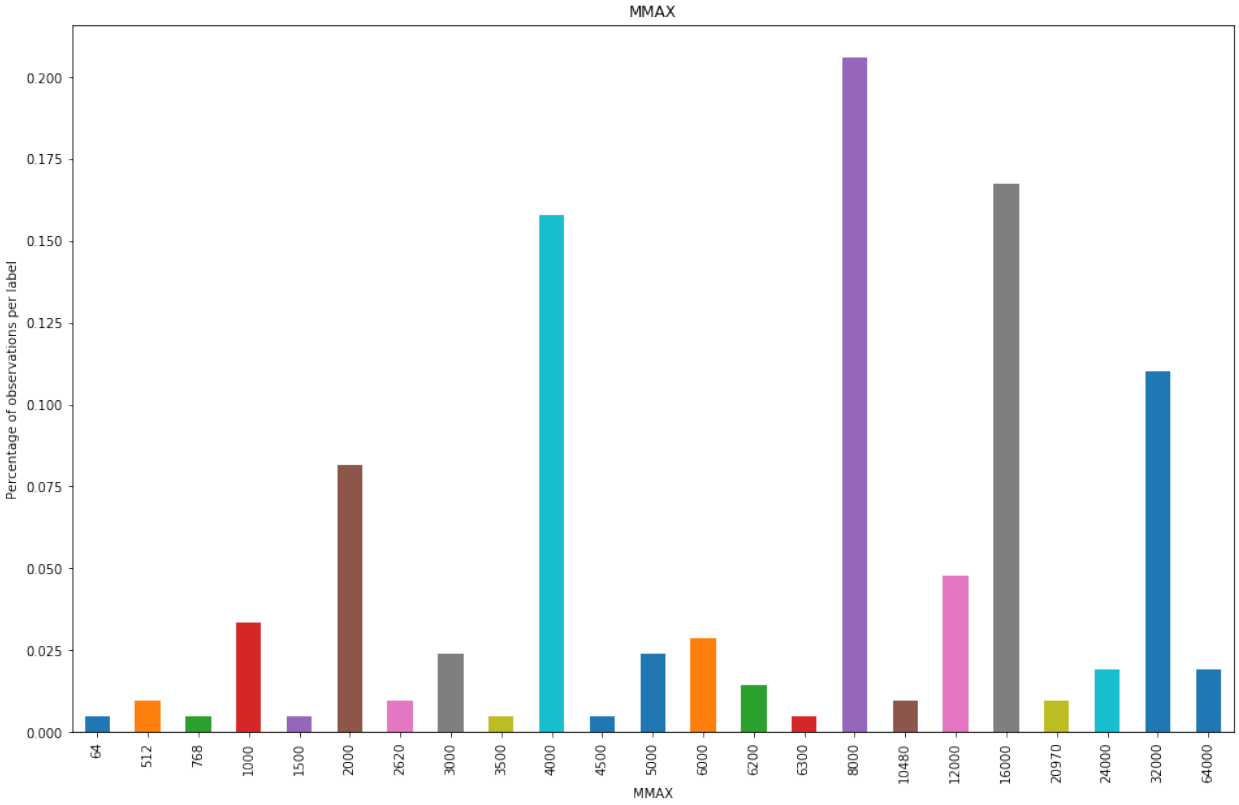
```
3500    0.0048
1500    0.0048
4500    0.0048
Name: MMAX, dtype: float64

0     0.3301
8     0.1483
32    0.1100
64    0.0957
16    0.0670
4     0.0383
24    0.0335
128   0.0287
6     0.0239
2     0.0191
30    0.0191
1     0.0096
9     0.0096
256   0.0096
48    0.0096
65    0.0096
112   0.0096
131   0.0096
12    0.0048
142   0.0048
96    0.0048
160   0.0048
Name: CACH, dtype: float64

1    0.4498
3    0.1340
8    0.0861
6    0.0766
12   0.0526
16   0.0478
4    0.0383
5    0.0335
2    0.0287
0    0.0239
52   0.0096
32   0.0048
26   0.0048
24   0.0048
7    0.0048
Name: CHMIN, dtype: float64

6     0.1435
24    0.1148
8     0.0957
32    0.0718
```

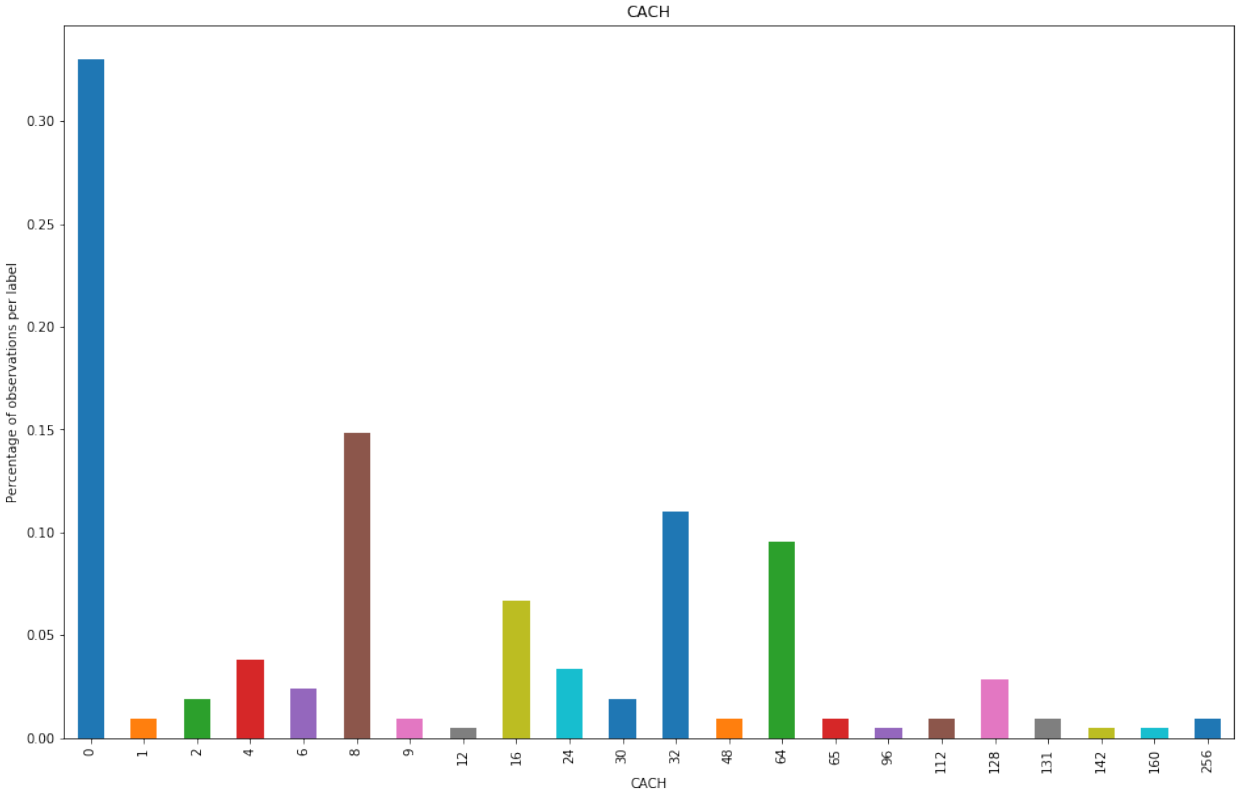
```
5     0.0622
16    0.0574
2     0.0526
4     0.0526
1     0.0478
3     0.0431
12    0.0383
20    0.0239
0     0.0239
64    0.0239
10    0.0191
14    0.0191
38    0.0144
54    0.0144
7     0.0096
176   0.0096
128   0.0096
104   0.0096
13    0.0048
15    0.0048
26    0.0048
28    0.0048
31    0.0048
48    0.0048
52    0.0048
112   0.0048
19    0.0048
Name: CHMAX, dtype: float64
```

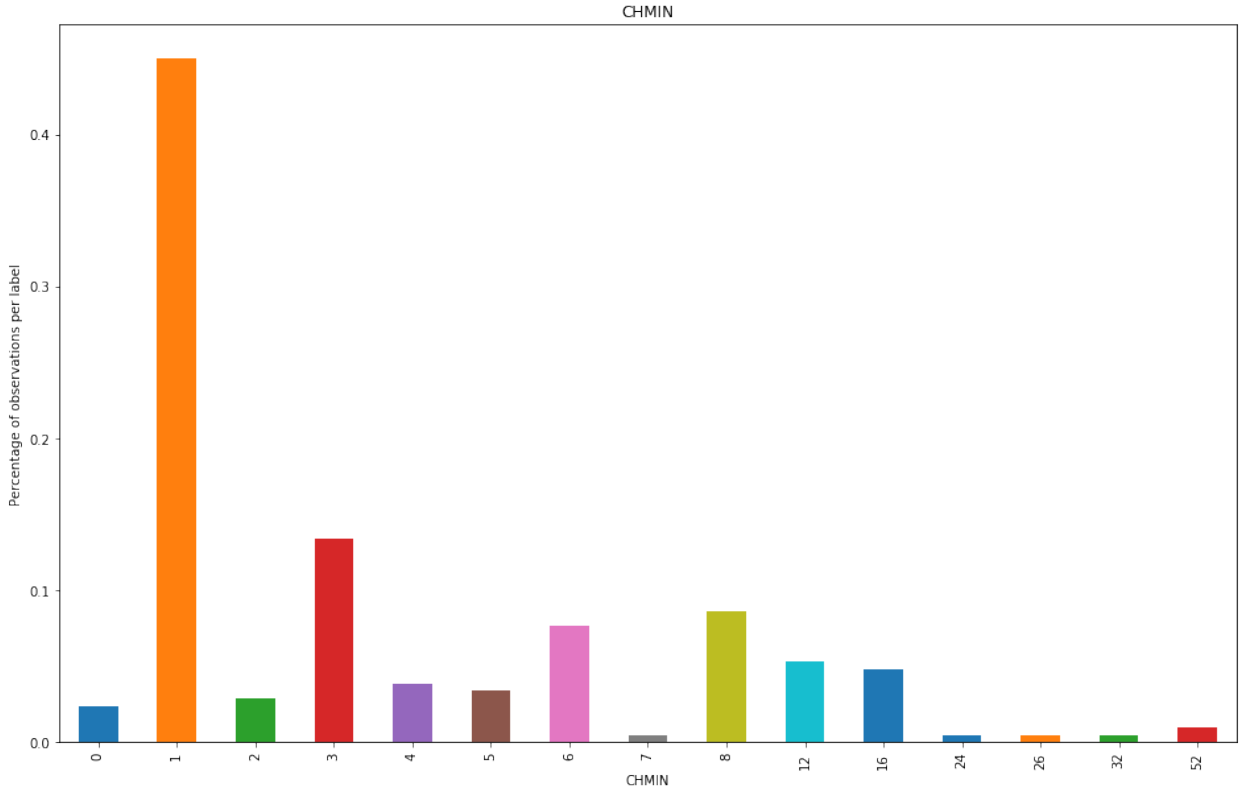
```
# detect outliers in discrete variables

for var in ['MYCT', 'MMIN', 'MMAX', 'CACH', 'CHMIN', 'CHMAX']:
    plt.figure(figsize=(16,10))
    (df.groupby(var)[var].count() / np.float(len(df))).plot.bar()
    plt.ylabel('Percentage of observations per label')
    plt.title(var)
    plt.show()
```

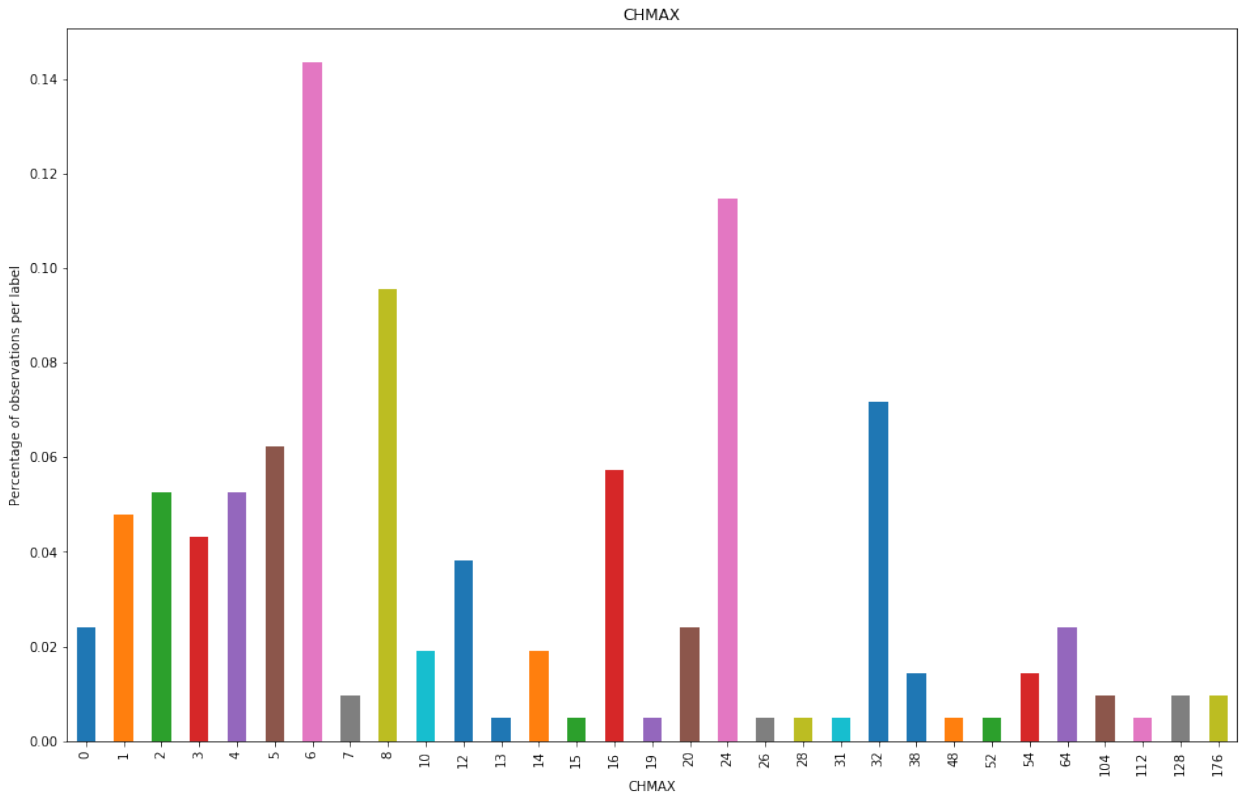

MYCT
Percentage of observations per label
0.12
0.10
0.08
0.06
0.04
0.02
0.00
17 23 25 26 29 30 35 38 40 48 50 52 56 57 59 60 64 70 72 75 84 90 92 98 100 105 110 112 115 116 124 125 133 140 143 150 160 167 175 180 185 200 203 220 225 240 250 300 320 330 350 400 480 600 700 800 810 900 1100 1500
MYCT


MMIN
Percentage of observations per label
0.25
0.20
0.15
0.10
0.05
0.00
64 96 128 192 256 262 384 500 512 524 768 1000 1310 1500 2000 2300 2620 3000 3100 4000 5000 5240 8000 16000 32000
MMIN


MMAX
Percentage of observations per label
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
64
512
768
1000
1500
2000
2620
3000
3500
4000
4500
5000
6000
6200
6300
8000
10480
12000
16000
20970
24000
32000
64000
MMAX


CACH
Percentage of observations per label
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
4
6
8
9
12
16
24
30
32
48
64
65
96
112
128
131
142
160
256
CACH


CHMIN
Percentage of observations per label
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
6
7
8
12
16
24
26
32
52
CHMIN

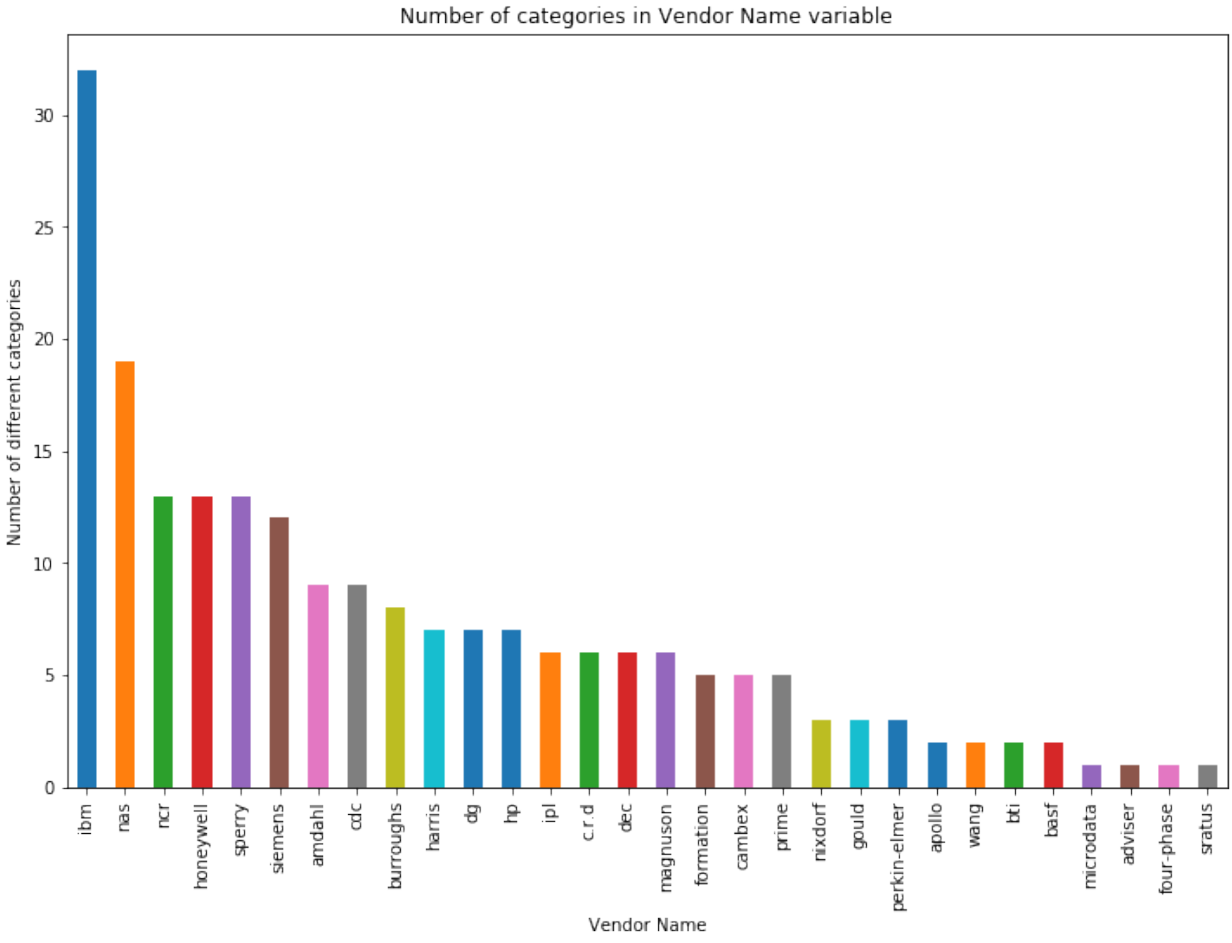

CHMAX
Percentage of observations per label
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
1
2
3
4
5
6
7
8
10
12
13
14
15
16
19
20
24
26
28
31
32
38
48
52
54
64
104
112
128
176
CHMAX

From the above plot, we can see that the discrete variables show values that are shared by a tiny proportion of variable values in the dataset. For linear regression modeling, this does not cause any problem.

Number of labels: cardinality

Now, I will examine the categorical variable **Vendor Name**. First I will determine whether it show high cardinality. This is a high number of labels.

```
# plot the categorical variable

plt.figure(figsize=(12,8))
(df['Vendor Name'].value_counts()).plot.bar()
plt.title('Number of categories in Vendor Name variable')
plt.xlabel('Vendor Name')
plt.ylabel('Number of different categories')
plt.show()
```



We can see that the **Vendor Name** variable, contain only a few labels. So, we do not have to deal with high cardinality.

11. Linear Regression Modeling

Now, I discuss the most important part of this project which is the Linear Regression model building.

First of all, I will divide the dataset into categorical and numerical variables as follows:-

Divide the dataset into categorical and numerical variables

```
df_cat = df.iloc[:,:2]

df_num = df.iloc[:, 2:]

df_num.head()
```

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
0	125	256	6000	256	16	128	198	199
1	29	8000	32000	32	8	32	269	253
2	29	8000	32000	32	8	32	220	253
3	29	8000	32000	32	8	32	172	253
4	29	8000	16000	32	8	16	132	132

Select the predictor and target variables

```
X = df_num.iloc[:,0:6]

y = df_num.iloc[:,-1]
```

Create separate train and test sets

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =
0.3,random_state = 0)
```

View the dimensions of X_train, X_test, y_train, y_test

```
X_train.shape, y_train.shape
```

```
((146, 6), (146,))
```

```
X_test.shape, y_test.shape
```

```
((63, 6), (63,))
```

```
# let's inspect the training dataframe

X_train.head()
```

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX
61	800	256	8000	0	1	4
24	320	128	6000	0	1	12

```
30    25  1310  2620   131     12     24
60   800   256  8000     0      1      4
56   220  1000  8000    16      1      2
```

```
X_train.describe()
```

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX
count	146.0000	146.0000	146.0000	146.0000	146.0000	146.0000
mean	205.8082	2,799.9726	11,741.2055	25.5685	4.5479	19.2397
std	249.6152	3,865.5077	11,879.6456	41.6903	6.5770	28.8810
min	17.0000	64.0000	64.0000	0.0000	0.0000	0.0000
25%	50.0000	512.0000	4,000.0000	0.0000	1.0000	5.0000
50%	115.5000	2,000.0000	8,000.0000	8.0000	1.5000	8.0000
75%	240.0000	4,000.0000	16,000.0000	32.0000	6.0000	24.0000
max	1,500.0000	32,000.0000	64,000.0000	256.0000	52.0000	176.0000

Feature Scaling

```
# Feature Scaling - I use the StandardScaler from sklearn

# import the StandardScaler class from preprocessing library
from sklearn.preprocessing import StandardScaler

# instantiate an object scaler
scaler = StandardScaler()

# fit the scaler to the training set and then transform it
X_train = scaler.fit_transform(X_train)

# transform the test set
X_test = scaler.transform(X_test)
```

The scaler is now ready, we can use it in a machine learning algorithm when required.

Fit the Linear Regression model

```
# fit the linear regression model

# import the LinearRegression class from linear_model library
from sklearn.linear_model import LinearRegression

# instantiate an object lr
lr = LinearRegression()


# Train the model using the training sets
lr.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
normalize=False)
```

12. Predicting the results

I have built the linear regression model. Now it is time to predict the results.

Predicting the test set results

```
# Predict on the test data set
y_pred = lr.predict(X_test)
```

Predicting estimated relative CPU performance values

```
#print("Predicted ERP - estimated relative performance for the first
five values")

lr.predict(X_test)[0:5]

array([ 53.25899879,  -7.30914167,  85.61134478, 333.46353054,
        88.17105392])
```

13. Model slope and intercept terms

The slope parameters(w) are also called weights or coefficients. They are stored in the **coef_** attribute.

The offset or intercept(b) is stored in the **intercept_** attribute.

So, the model slope is given by **lr.coef_** and model intercept term is given by **lr.intercept_**.

```
print("Number of coefficients:", len(lr.coef_))

print("Estimated coefficients: {}".format(lr.coef_))

print("Estimated intercept: {}".format(lr.intercept_))

Number of coefficients: 6
Estimated coefficients: [17.70202595 59.11241774 78.35042681
16.53981449 -0.35410978 38.97256261]
Estimated intercept: 100.0
```

I constructed a dataframe that contains features and estimated coefficients.

```
dataset = list(zip(pd.DataFrame(X_train).columns, lr.coef_))

pd.DataFrame(data = dataset, columns = ['Features', 'Estimated
Coefficients']).set_index('Features')
```

```
          Estimated Coefficients
Features
0                         17.7020
1                         59.1124
2                         78.3504
3                         16.5398
4                         -0.3541
5                         38.9726
```

14. Evaluate model performance

I have built the linear regression model and use it to predict the results. Now, it is the time to evaluate the model performance. We want to understand the outcome of our model and we want to know whether the performance is acceptable or not. For regression problems, there are several ways to evaluate the model performance. These are listed below:-

• RMSE (Root Mean Square Error)

• R2 Score

• Overfitting Vs Underfitting

• Cross validation

• Residual analysis

• Normality test

I have described these measures in following sections:-

i. RMSE

RMSE stands for **Root Mean Square Error**. RMSE is the standard deviation of the residuals. RMSE gives us the standard deviation of the unexplained variance by the model. It can be calculated by taking square root of Mean Squared Error.

RMSE is an absolute measure of fit. It gives us how spread the residuals are, given by the standard deviation of the residuals. The more concentrated the data is around the regression line, the lower the residuals and hence lower the standard deviation of residuals. It results in lower values of RMSE. So, lower values of RMSE indicate better fit of data.

RMSE can be calculated as follows:-

```
# RMSE(Root Mean Square Error)

from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("RMSE value : {:.2f}".format(rmse))

RMSE value : 37.99
```

Interpretation

The RMSE value has been found to be 37.99. It means the standard deviation for our prediction is 37.99. So, sometimes we expect the predictions to be off by more than 37.99 and other times we expect less than 37.99.

ii. R2 Score

R2 Score is another metric to evaluate performance of a regression model. It is also called **Coefficient of Determination**. It gives us an idea of goodness of fit for the linear regression models. It indicates the percentage of variance that is explained by the model.

R2 Score = Explained Variation/Total Variation

Mathematically, we have

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

The total sum of squares, $SS_{tot} = \sum_i (y_i - \acute{y})^2$

The regression sum of squares (explained sum of squares), $SS_{reg} = \sum_i (f_i - \acute{y})^2$

The sum of squares of residuals (residual sum of squares), $SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$

In general, the higher the R2 Score value, the better the model fits the data. Usually, its value ranges from 0 to 1. So, we want its value to be as close to 1. Its value can become negative if our model is wrong.

R2 score value can be found as follows:-

```
# R2 Score

from sklearn.metrics import r2_score
print("R2 Score value: {:.2f}".format(r2_score(y_test, y_pred)))

R2 Score value: 0.92
```

Interpretation

In business decisions, the benchmark for the R2 score value is 0.7. It means if R2 score value >= 0.7, then the model is good enough to deploy on unseen data whereas if R2 score value < 0.7, then the model is not good enough to deploy.

Our R2 score value has been found to be 0.92. It means that this model explains 92% of the variance in our dependent variable. So, the R2 score value confirms that the model is good enough to deploy because it provides good fit to the data.

iii. Overfitting Vs Underfitting

```
# Evaluating training set performance

print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
```

```
Training set score: 0.91
```

```
# Evaluating test set performance

print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

```
Test set score: 0.92
```

Interpretation

Training set and test set performances are comparable. An R Square value of 0.92 is very good.

iv. Cross validation

Cross-validation is a vital step in evaluating a model. It maximizes the amount of data that is used to train the model.

In cross-validation, we split the training data into several subgroups. Then we use each of them in turn to evaluate the model fitted on the remaining portion of the data.

It helps us to obtain reliable estimates of the model's generalization performance. So, it helps us to understand how well the model performs on unseen data.

We can perform cross validation as follows:-

```
# import the library
from sklearn.model_selection import cross_val_score

# Compute 5-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(lr, X, y, cv=5)

# print the 5-fold cross-validation scores
print(cv_scores.round(4))
```

```
[ 0.8484 -0.864   0.7149  0.8755  0.7707]
```

```
# print the avarage 5-fold cross-validation scores
print("Average 5-Fold CV Score:
{}".format(np.mean(cv_scores).round(4)))
```

```
Average 5-Fold CV Score: 0.4691
```

Interpretation

There is a large fluctuation in the cross validation scores of the model.

The average 5-fold cross validation score is very poor and hence the linear regression model is not a great fit to the data.

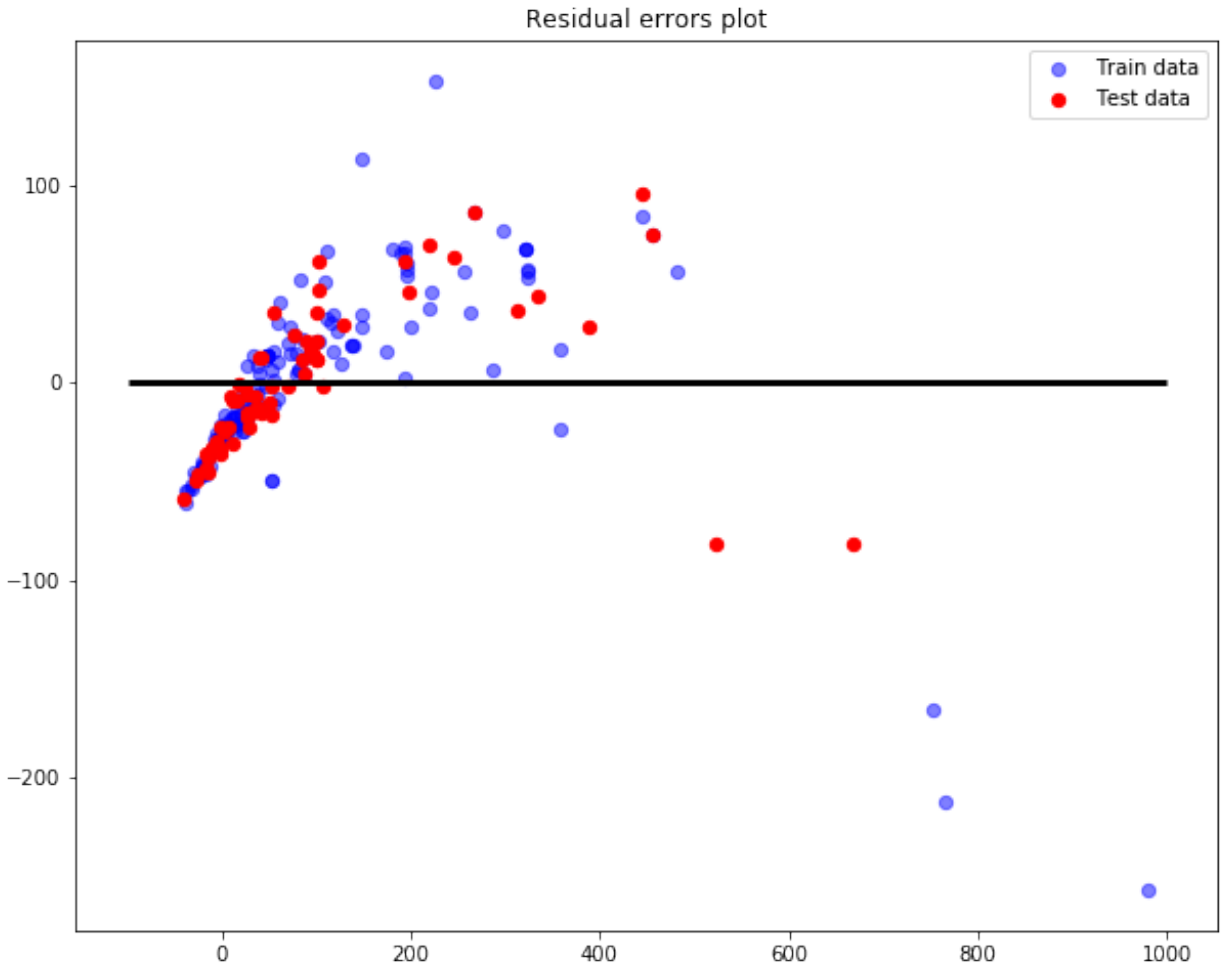
v. Residual analysis

A linear regression model may not represent the data appropriately. The model may be a poor fit to the data. So, we should validate our model by defining and examining residual plots. The difference between the observed value of the dependent variable (y) and the predicted value (ŷi) is called the **residual** and is denoted by e. The scatter-plot of these residuals is called **residual plot**.

If the data points in a residual plot are randomly dispersed around horizontal axis and an approximate zero residual mean, a linear regression model may be appropriate for the data. Otherwise a non-linear model may be more appropriate.

Now, I will plot the residual errors.

```
# Plot for residual error

# adjust the figure size
plt.figure(figsize=(10,8))

# plotting residual errors in training data
plt.scatter(lr.predict(X_train), lr.predict(X_train) - y_train, c =
'b', s = 40, label = 'Train data', alpha = 0.5)

# plotting residual errors in test data
plt.scatter(lr.predict(X_test), lr.predict(X_test) - y_test, c = 'r',
s = 40, label = 'Test data')

# plotting line for zero residual error
plt.hlines(y = 0, xmin = -100, xmax = 1000, linewidth = 3)

# plotting legend
plt.legend(loc = 'upper right')

# plot title
plt.title("Residual errors plot")

# function to show plot
plt.show()
```



Interpretation of residual plots

A regression model that has nicely fit the data will have its residuals display randomness (i.e., lack of any pattern). This comes from the **homoscedasticity** assumption of regression modeling. Typically scatter plots between residuals and predictors are used to confirm the assumption. Any pattern in the scatter-plot, results in a violation of this property and points towards a poor fitting model.

Residual errors plot show that the data is randomly scattered around line zero. The plot does not display any pattern in the residuals. Hence, we can conclude that the Linear Regression model is a good fit to the data.

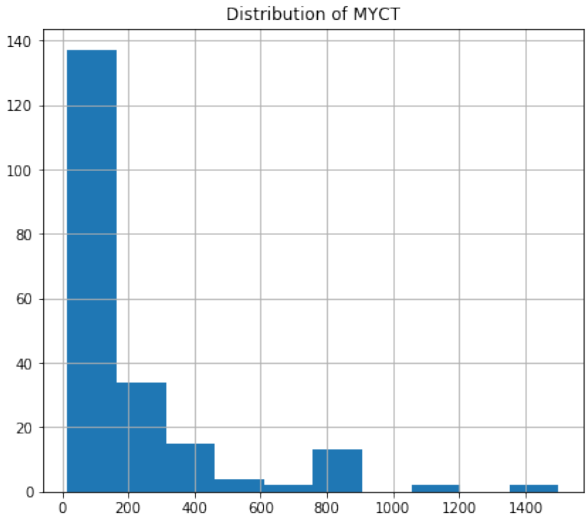
vi. Normality test (Q-Q Plot)

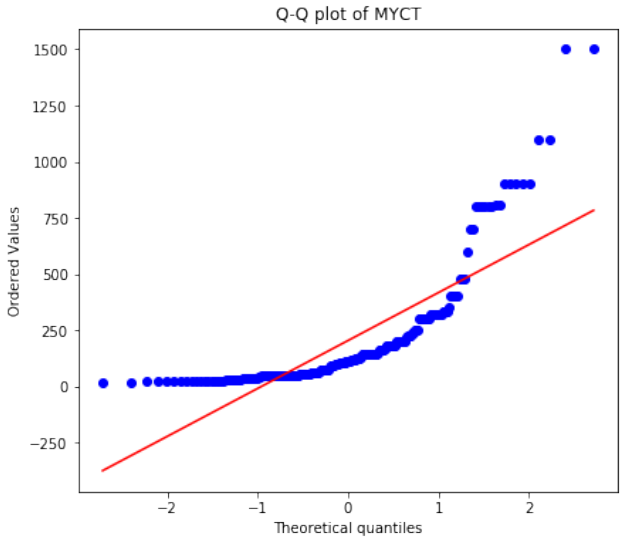
This is a visual or graphical test to check for normality of the data. This test helps us identify outliers and skewness. The test is performed by plotting the data verses theoretical quartiles. The same data is also plotted on a histogram to confirm normality.

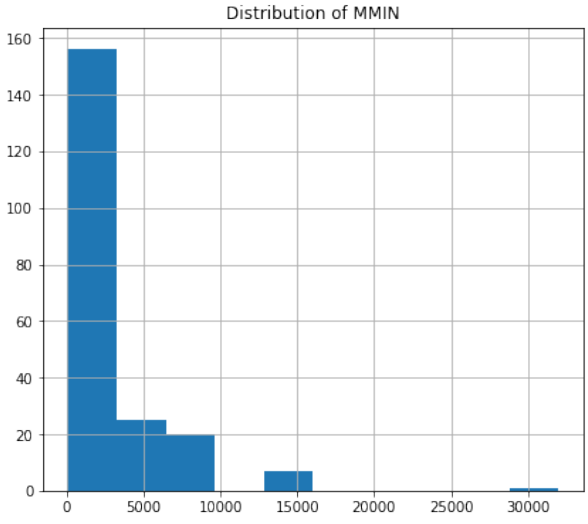
Any deviation from the straight line in normal plot or skewness/multi-modality in histogram shows that the data does not pass the normality test.

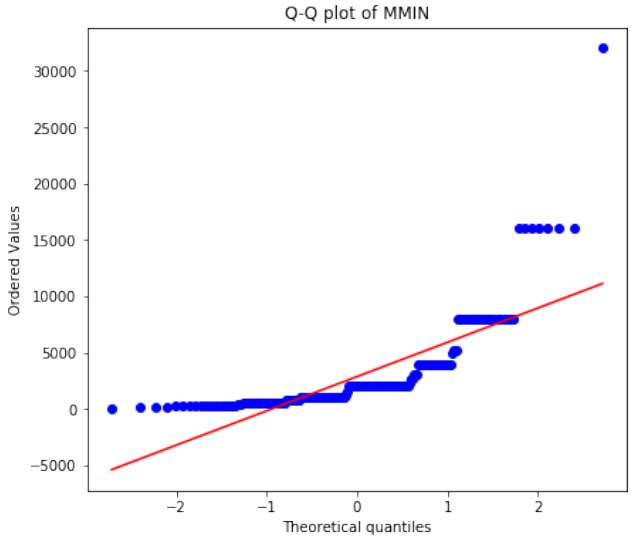
Now, I will plot the Q-Q Plot as follows:-

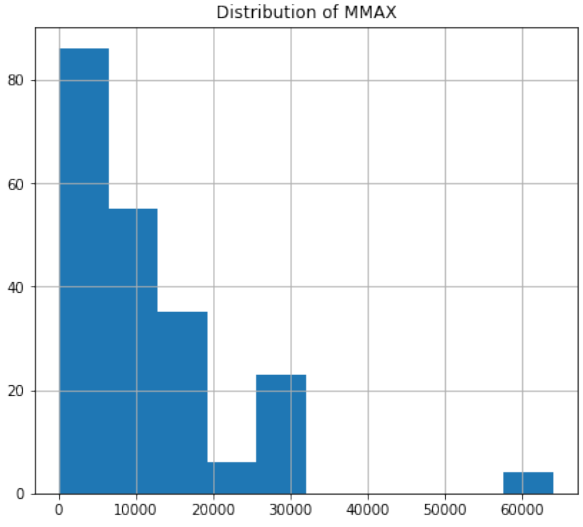
```
# plotting the Q-Q plot

import pylab
import scipy.stats as stats


for var in ['MYCT', 'MMIN', 'MMAX', 'CACH', 'CHMIN', 'CHMAX']:

    plt.figure(figsize=(15,6))

    plt.subplot(1, 2, 1)
    df[var].hist()
    plt.title('Distribution of '+ var)

    plt.subplot(1, 2, 2)
    stats.probplot(df[var], dist="norm", plot=pylab)
    plt.title('Q-Q plot of '+ var)

    plt.show()
```

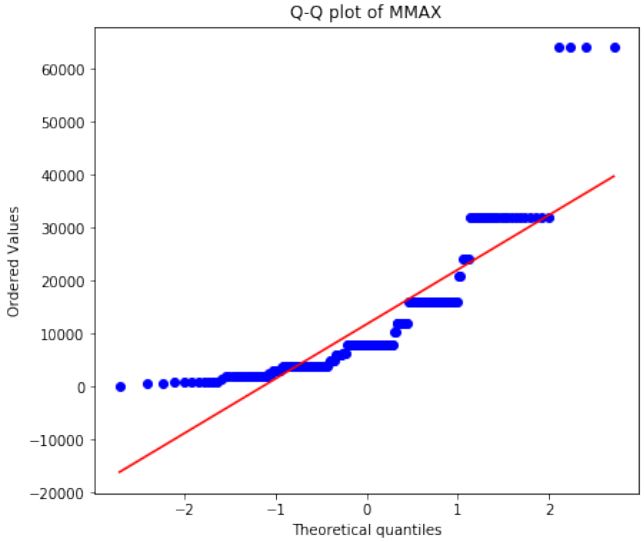


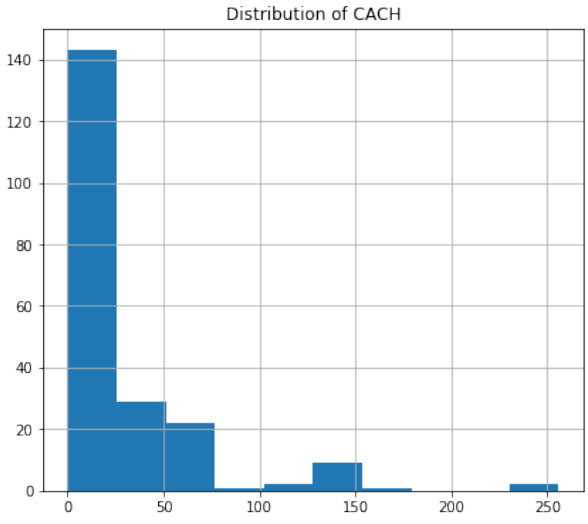


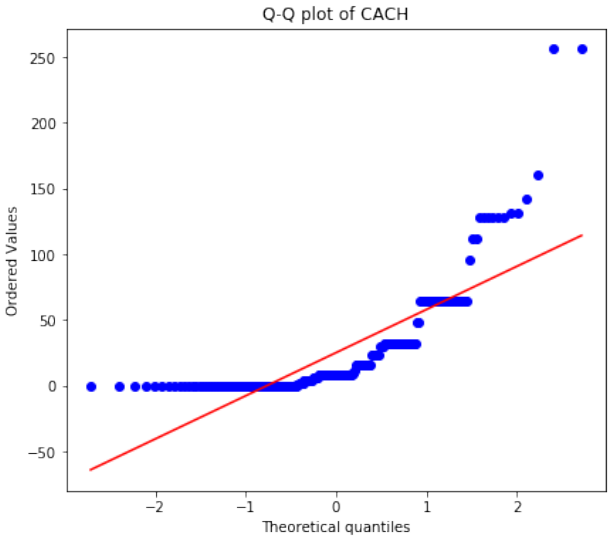

Distribution of MMIN
160
140
120
100
80
60
40
20
0
0
5000
10000
15000
20000
25000
30000

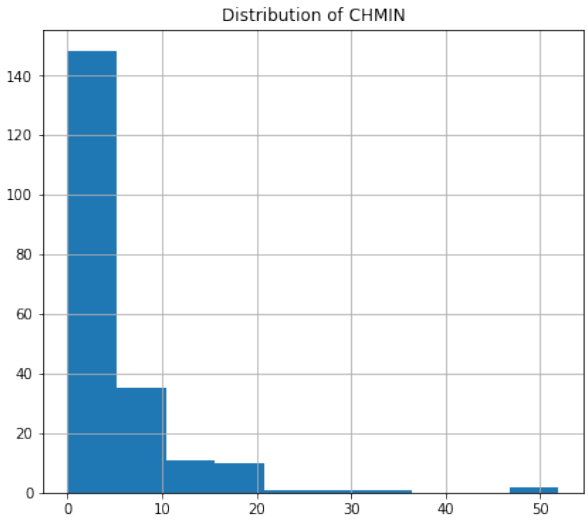

Q-Q plot of MMIN
Ordered Values
30000
25000
20000
15000
10000
5000
0
−5000
−2
−1
0
1
2
Theoretical quantiles

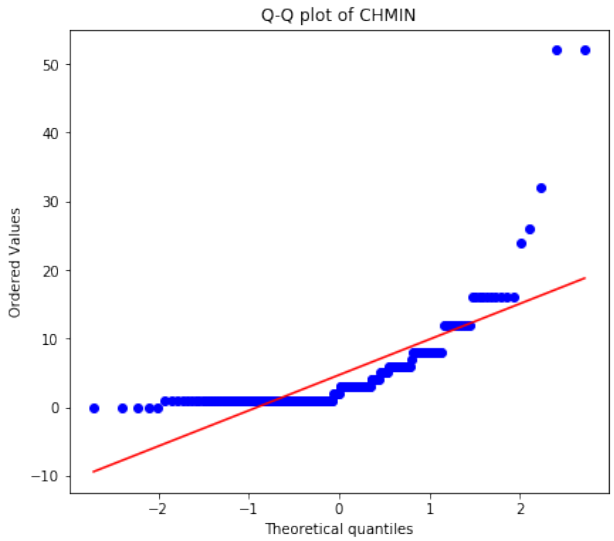

Distribution of MMAX
80
60
40
20
0
0
10000
20000
30000
40000
50000
60000

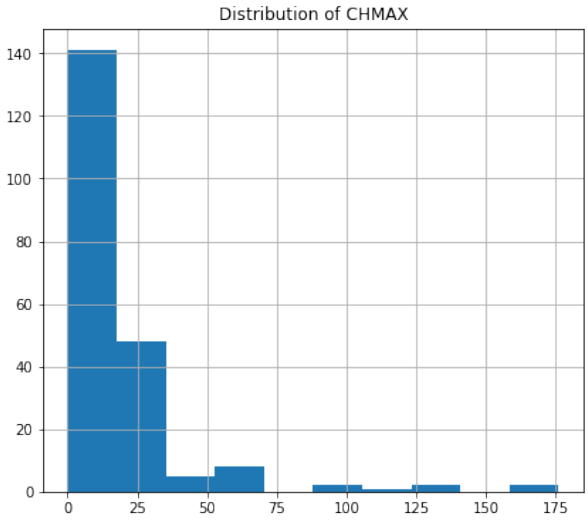

Q-Q plot of MMAX
Ordered Values
60000
50000
40000
30000
20000
10000
0
−10000
−20000
−2
−1
0
1
2
Theoretical quantiles

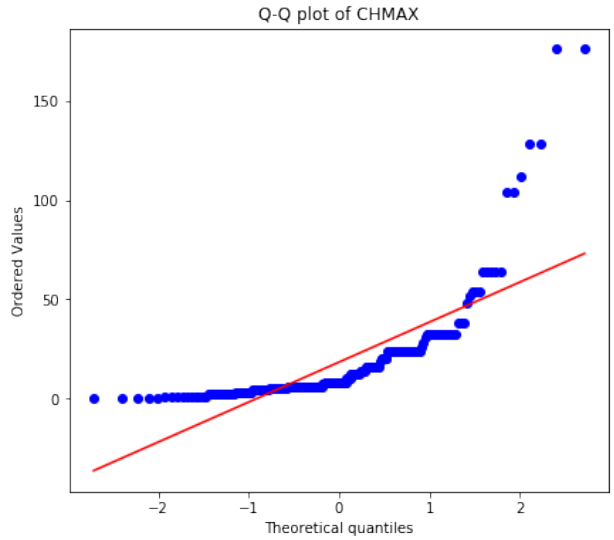

Distribution of CACH
140
120
100
80
60
40
20
0
0
50
100
150
200
250


Q-Q plot of CACH
Ordered Values
250
200
150
100
50
0
−50
−2
−1
0
1
2
Theoretical quantiles









Interpretation

From the distribution plots, we can see that all the above variables are positively skewed. The Q-Q plot of all the variables confirm that the variables are not normally distributed.

Hence, the variables do not pass the normality test.

15. Conclusion

I carry out residual analysis to check for homoscedasticity assumption. Residual errors plot show that the data is randomly scattered around line zero. The plot does not display any pattern in the residuals. Hence, we can conclude that the Linear Regression model is a good fit to the data.

The r-squared or the coefficient of determination is 0.4691 on an average for 5-fold cross validation. It means that the predictor is only able to explain 46.91% of the variance in the target variable. This indicates that the model is not a good fit to the data.

I carry out normality test to check for distribution of the variables. We can see that the variables do not follow the normal distribution. The Q-Q plots confirm the same.

So, we can conclude that the linear regression model is unable to model the data to generate decent results. It should be noted that the model is performing equally on both training and testing datasets. It seems like a case where we would need to model this data using methods that can model non-linear relationships. Also variables need to be transformed to satisfy the normality assumption.