NOTE:

I attached a Rmd file and it's pdf , it contains proof and test that justifies my choice of model.

My baseline model is linear regression model from sklearn, so most of my assumptions are OLS assumptions and i proved them on R programming.

In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
df=pd.read_csv(r"D:\Self Study\Datasets\Mission_ML\Cac_df.csv")
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 305 entries, x001 to y
dtypes: float64(41), int64(264)
memory usage: 232.7 MB
```

In [4]:

```
df.shape
```

Out[4]:

```
(100000, 305)
```

1 EDA

In [5]:

```
pd.options.display.max_columns = None
```

In [6]:

```
df.describe(include='all' )
```

Out[6]:

	x001	x002	x003	x004	x005	x006	
count	1.000000e+05	78568.000000	78568.000000	78576.000000	93890.000000	100000.000000	1
mean	1.218244e+06	125.711727	25.541238	65.393212	178.238545	0.314040	
std	2.728977e+05	115.785117	49.028751	63.592317	124.520628	0.464135	
min	5.170000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	9.743635e+05	32.000000	3.000000	19.000000	87.000000	0.000000	
50%	1.235926e+06	100.000000	8.000000	48.000000	150.000000	0.000000	
75%	1.445326e+06	180.000000	24.000000	92.000000	246.000000	1.000000	
max	1.677197e+06	718.000000	704.000000	704.000000	827.000000	1.000000	

Majority of columns has mean in once and tens place.

1.1 Handelling Missing Values

In [7]:

```
def miss(df):
    nulli = df.isnull().sum()
    nulli_per = 100 * df.isnull().sum() / len(df)
    table = pd.concat([nulli, nulli_per], axis=1)
    table = table.rename(columns={
        0: "Number of Missing Values",
        1: "percentage of Null Values"
    })
    table = table[table.iloc[:, 1] != 0].sort_values(
        "percentage of Null Values", ascending=False).round(0)
    return table
```

In [8]:

```
pop=miss(df)
```

In [9]:

```
pop
```

Out[9]:

	Number of Missing Values	percentage of Null Values
x242	93339	93.0
x295	86533	87.0
x304	81875	82.0
x098	80681	81.0
x155	79051	79.0
x259	77432	77.0
x255	76913	77.0
x257	76913	77.0
x256	76913	77.0
x302	73069	73.0
x268	67253	67.0
x162	66481	66.0
x267	66461	66.0
x266	66461	66.0
x265	66461	66.0
x253	66333	66.0
x297	58112	58.0
x275	56131	56.0
x293	51133	51.0
x289	49756	50.0
x290	49756	50.0
x288	49756	50.0
x148	41785	42.0
x223	37069	37.0
x222	36987	37.0
x041	36872	37.0
x058	36872	37.0
x057	36872	37.0
x238	36744	37.0
x239	36744	37.0
x237	36744	37.0
x287	24821	25.0
x002	21432	21.0
x003	21432	21.0

	Number of Missing Values	percentage of Null Values
x004	21424	21.0
x235	20083	20.0
x045	19674	20.0
x044	19674	20.0
x234	19110	19.0
x272	7189	7.0
x005	6110	6.0

In [10]:

```
print("Number of columns with null values: ",len(pop))
```

```
Number of columns with null values:  41
```

1.1.1 Dropping columns based on percentage of missing values depends upon it's importance in determining y

Here iam planning to impute mean,mode or median for columns with percentage of missing values less than 25%. Before dropping all other columns i decided to check how important is each of my columns or do a correlation study

AIM:

```
1) Impute columns with less than 25% missing values
2) Drop other columns if it has less correlation with y
```

Correlation Test

In [11]:

```
#collecting index of pop

index = pop.index[pop["percentage of Null Values"] >
                  25]  # columns that go through corr test.
corr_test_index = list(index)
corr_test_index.append("y")
impute_index = pop.index[pop["percentage of Null Values"] <=
                         25]  # columns for imputing.
impute_index = list(impute_index)
# Dataframe

df_corr = df[corr_test_index]

# Correlation with y
for i in df_corr:
    print(i, df_corr['y'].corr(df_corr[i]))
```

```
x242 -0.5478435216960045
x295 -0.24235402991262694
x304 -0.36813797062247333
x098 0.20585673348728786
x155 -0.5420901564826546
x259 -0.2081109037561265
x255 -0.14040871344585965
x257 -0.17782568172472343
x256 -0.20192363535825414
x302 -0.4838519345711038
x268 -0.28296990792976245
x162 -0.40355610759133703
x267 0.06006773669931324
x266 0.03574416366491076
x265 0.07456223464607242
x253 -0.4738232347394058
x297 -0.37477200541628714
x275 -0.020511220811565242
x293 -0.370417113443797
x289 0.00653090667333702
x290 -0.009339253406241999
x288 -0.018679951269441525
x148 -0.4837540352296139
x223 -0.051106590440712724
x222 -0.05178852330355621
x041 -0.6908397412590069
x058 -0.5868111610915547
x057 -0.6365099217971081
x238 0.2826543813293275
x239 0.48730517576056753
x237 0.49930234502061555
y 1.0
```

am stating my threshold correlation coefficient as +-0.60 , as a result none of the above columns are important for my model.

In [12]:

```
k = list(impute_index)

k.append("y")
dfi = df[k]
for i in dfi:
    print(i, dfi['y'].corr(dfi[i]))
```

```
x287 -0.5621909182417315
x002 0.48574426200475757
x003 0.12120685688911217
x004 0.4196825152123885
x235 0.620393873555563
x045 0.14543239605978614
x044 0.2124581779194169
x234 0.11824018706936343
x272 -0.06676143497881762
x005 0.5759698679383911
y 1.0
```

Eventhough the columns selected for imputing contains columns with less than threshold corr value, we are not droping it since it may have a significant impact on finding y

Droping

In [13]:

```
#Droping
corr_test_index.remove("y")
data = df.drop(corr_test_index, axis=1)
```

In [14]:

```
corr_test_index
```

Out[14]:

```
['x242',
 'x295',
 'x304',
 'x098',
 'x155',
 'x259',
 'x255',
 'x257',
 'x256',
 'x302',
 'x268',
 'x162',
 'x267',
 'x266',
 'x265',
 'x253',
 'x297',
 'x275',
 'x293',
 'x289',
 'x290',
 'x288',
 'x148',
 'x223',
 'x222',
 'x041',
 'x058',
 'x057',
 'x238',
 'x239',
 'x237']
```

Imputing

In [15]:

```
data[impute_index]
```

Out[15]:

	x287	x002	x003	x004	x235	x045	x044	x234	x272	x005
0	NaN	NaN	NaN	NaN	300.0	300.0	300.0	0.0	0.0000	8.0
1	1.0	4.0	3.0	3.0	NaN	NaN	NaN	NaN	0.9339	4.0
2	NaN	NaN	NaN	NaN	1800.0	200.0	1800.0	1026.0	0.2281	96.0
3	2.0	63.0	14.0	38.0	4000.0	100.0	4000.0	4340.0	0.8204	258.0
4	NaN	34.0	25.0	29.0	1000.0	300.0	1000.0	186.0	0.1000	34.0
...	...	...	...	...	...	...	...	...	...	...
99995	5.0	200.0	3.0	157.0	500.0	250.0	500.0	0.0	0.3308	200.0
99996	1.0	292.0	80.0	159.0	47500.0	100.0	47500.0	5126.0	0.0872	292.0
99997	5.0	35.0	4.0	26.0	5000.0	300.0	5000.0	5663.0	0.4824	57.0
99998	NaN	4.0	3.0	3.0	300.0	300.0	300.0	378.0	1.1650	4.0
99999	5.0	134.0	19.0	75.0	14000.0	300.0	14000.0	775.0	0.0707	678.0

100000 rows × 10 columns

In [16]:

```
data[impute_index].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   x287    75179 non-null  float64
 1   x002    78568 non-null  float64
 2   x003    78568 non-null  float64
 3   x004    78576 non-null  float64
 4   x235    79917 non-null  float64
 5   x045    80326 non-null  float64
 6   x044    80326 non-null  float64
 7   x234    80890 non-null  float64
 8   x272    92811 non-null  float64
 9   x005    93890 non-null  float64
dtypes: float64(10)
memory usage: 7.6 MB
```

Every column for imputing has dtype as float so we can fill Null values with mean of each columns.

In [17]:

```
for i in data:
    if i in impute_index:
        data[i].fillna(data[i].mean(), inplace=True)
```

In [18]:

```
data
```

Out[18]:

	x001	x002	x003	x004	x005	x006	x007	x008	x009	x010	x011	x012	x013
0	1540332	125.711727	25.541238	65.393212	8.0	1	0	1	0	0	0	0	0
1	823066	4.000000	3.000000	3.000000	4.0	0	2	2	0	0	0	0	0
2	1089795	125.711727	25.541238	65.393212	96.0	1	0	0	0	1	3	4	1
3	1147758	63.000000	14.000000	38.000000	258.0	0	0	0	1	2	1	1	1
4	1229670	34.000000	25.000000	29.000000	34.0	1	0	0	0	3	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	1573467	200.000000	3.000000	157.000000	200.0	1	3	3	0	0	0	1	0
99996	1653422	292.000000	80.000000	159.000000	292.0	1	1	1	1	2	0	4	3
99997	1284669	35.000000	4.000000	26.000000	57.0	0	1	1	5	10	4	0	0

In [19]:

```
data.isnull().sum().sum()
```

Out[19]:

0

Our data contains zero null values

In [ ]:

2 Base line Model : Linear Regression

In [20]:

```
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = data.drop("y", axis=1)
y = data["y"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

regr = LinearRegression()

regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)
```

In [21]:

```
y_pred
```

Out[21]:

```
array([729.26738997, 756.50888232, 647.92271594, ..., 657.08521699,
       533.02752296, 565.66365713])
```

In [22]:

```
import sklearn.metrics as sm
print("Mean absolute error =",round(sm.mean_absolute_error(y_test, y_pred),4))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_pred),4))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_pred),4))
print("Explain variance score =", round(sm.explained_variance_score(y_test, y_pred),4))

print("R2 score =", 100*round(sm.r2_score(y_test, y_pred),4))
```

```
Mean absolute error = 38.9266
Mean squared error = 2558.3118
Median absolute error = 31.2718
Explain variance score = 0.819
R2 score = 81.89
```

2.1 Handelling Outlier

In [23]:

```
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
```

In [24]:

```
outlier=((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).sum()
```

In [25]:

```
outlier_df=outlier.to_frame()

outlier_df["Percentage"]=100*(outlier_df[0]/len(data))
```

In [26]:

```
outlier_df
```

Out[26]:

	0	Percentage
x001	389	0.389
x002	6796	6.796
x003	9266	9.266
x004	6767	6.767
x005	3614	3.614
...	...	...
x299	0	0.000
x300	0	0.000
x301	10624	10.624
x303	18062	18.062
y	0	0.000

274 rows × 2 columns

In [27]:

```
outlier_df.describe()
```

Out[27]:

	0	Percentage
count	274.000000	274.000000
mean	9464.076642	9.464077
std	6568.430284	6.568430
min	0.000000	0.000000
25%	4420.750000	4.420750
50%	8516.500000	8.516500
75%	14123.000000	14.123000
max	24939.000000	24.939000

In [28]:

```
outlier_df['Percentage'].idxmax()# MAX OUTLIER index
```

Out[28]:

```
'x247'
```

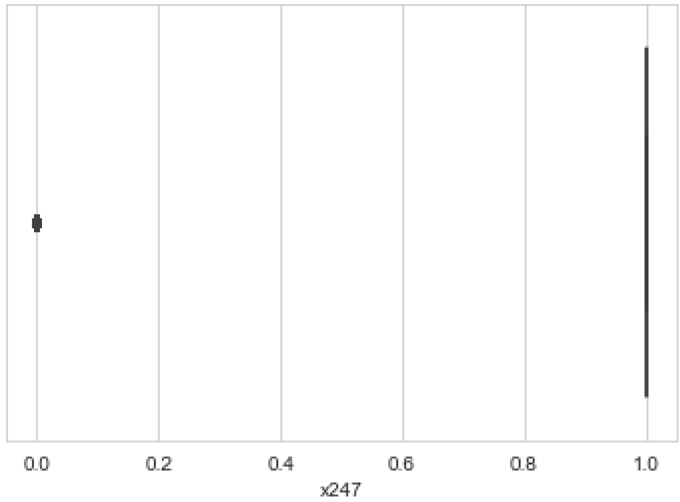
In [29]:

```
import seaborn as sns
sns.set_style("whitegrid")

sns.boxplot(data["x247"])
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x28980df1400>
```



In [30]:

```
(data["x247"]==1).sum()/(data["x247"]==0).sum()
```

Out[30]:

```
3.009783872649264
```

That is in the above column, value 1 is 3 times repeated than 0, so 0 is treated as an outlier.

Here 0 's freguency is too low as a result it is been considered as outliers.

In [31]:

```
index_out=outlier_df.index[outlier_df[0]==0]
```

In [32]:

```
outlier_df.drop(index_out)
```

Out[32]:

	0	Percentage
x001	389	0.389
x002	6796	6.796
x003	9266	9.266
x004	6767	6.767
x005	3614	3.614
...	...	...
x292	15840	15.840
x294	13295	13.295
x296	10619	10.619
x301	10624	10.624
x303	18062	18.062

252 rows × 2 columns

In [33]:

```
for k in data:
    value = np.percentile(data[k], 0.75)
    for i in range(len(k)):
        if (data[k][i]) > value:
            data[k][i] = value
```

```
<ipython-input-33-602ebde6268d>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  data[k][i] = value
```

In [34]:

```
data
```

Out[34]:

	x001	x002	x003	x004	x005	x006	x007	x008	x009	x010	x011	x012	x013	x014	x015	x0
0	511289	2.0	0.0	1.0	4.0	0	0	0	0	0	0	0	0	0	0	
1	511289	2.0	0.0	1.0	4.0	0	0	0	0	0	0	0	0	0	0	
2	511289	2.0	0.0	1.0	4.0	0	0	0	0	0	0	0	0	0	0	
3	511289	2.0	0.0	1.0	4.0	0	0	0	0	0	0	0	0	0	0	
4	1229670	34.0	25.0	29.0	34.0	1	0	0	0	3	0	0	0	0	3	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
99995	1573467	200.0	3.0	157.0	200.0	1	3	3	0	0	0	1	0	12	16	
99996	1653422	292.0	80.0	159.0	292.0	1	1	1	1	2	0	4	3	6	17	
99997	1284669	35.0	4.0	26.0	57.0	0	1	1	5	10	4	0	0	0	20	

3 Model 1

(Outlier handelled + Null_Values_Handelled)

In [35]:

```
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = data.drop("y",axis=1)
y = data["y"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

regr = LinearRegression()

regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)
```

In [36]:

```
import sklearn.metrics as sm
print("Mean absolute error =",round(sm.mean_absolute_error(y_test, y_pred),4))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_pred),4))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_pred),4))
print("Explain variance score =", round(sm.explained_variance_score(y_test, y_pred),4))

print("R2 score =", 100*round(sm.r2_score(y_test, y_pred),4))
```

```
Mean absolute error = 38.7445
Mean squared error = 2561.6097
Median absolute error = 30.8662
Explain variance score = 0.8184
R2 score = 81.84
```

3.1 Feature Engineering

The provided data lacks semantic implecation of each column , as a result i can't judge the unit or scale used to measure each column observations.

3.1.0.1 AIM: Standardization and Scaling of columns with contineous value.

3.1.0.2 Reasons:

Here iam normalising my data becuase columns range differs drastically , for an instance column2 has 127 while col1 contains values such as 1229670, if i dint run a normalisation naturally col1 have more impact on y(which may or mayn't true).

3.1.1 Standardization

In [ ]:

```

```

In [37]:

```
import pandas as pd
from sklearn import preprocessing

x = data.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
dat = pd.DataFrame(x_scaled)
```

In [38]:

```
# saving standardized data for R programming
# dat.to_csv("D:dwld/dat_df.csv", header=True, index=True)
```

In [39]:

```
dat.shape
```

Out[39]:

```
(100000, 274)
```

In [40]:

```
data.shape
```

Out[40]:

```
(100000, 274)
```

In [41]:

```
dat.describe()
```

Out[41]:

	0	1	2	3	4	
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000(
mean	0.726257	0.175082	0.036279	0.092886	0.215520	0.314(
std	0.162780	0.142942	0.061731	0.080073	0.145899	0.464
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.580804	0.064067	0.005682	0.036932	0.113664	0.000(
50%	0.736817	0.175086	0.019886	0.092888	0.192261	0.000(
75%	0.861708	0.211699	0.036280	0.109375	0.286578	1.000(
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000(

In [ ]:

In [42]:

```
dat
```

Out[42]:

	0	1	2	3	4	5	6	7	8	
0	0.304633	0.002786	0.000000	0.001420	0.004837	0.0	0.000000	0.000000	0.000000	0.(
1	0.304633	0.002786	0.000000	0.001420	0.004837	0.0	0.000000	0.000000	0.000000	0.(
2	0.304633	0.002786	0.000000	0.001420	0.004837	0.0	0.000000	0.000000	0.000000	0.(
3	0.304633	0.002786	0.000000	0.001420	0.004837	0.0	0.000000	0.000000	0.000000	0.(
4	0.733087	0.047354	0.035511	0.041193	0.041112	1.0	0.000000	0.000000	0.000000	0.(
...	...	...	...	...	...	...	...	...	...	
99995	0.938134	0.278552	0.004261	0.223011	0.241838	1.0	0.068182	0.027778	0.000000	0.(
99996	0.985820	0.406685	0.113636	0.225852	0.353083	1.0	0.022727	0.009259	0.012346	0.(
99997	0.765890	0.048747	0.005682	0.036932	0.068924	0.0	0.022727	0.009259	0.061728	0.:
99998	0.855476	0.005571	0.004261	0.004261	0.004837	0.0	0.045455	0.018519	0.000000	0.(
99999	0.952136	0.186630	0.026989	0.106534	0.819831	0.0	0.000000	0.000000	0.012346	0.(

100000 rows × 274 columns

4 Model 2

Standardized data + Outlier Handelling + Null_Values_Handelled)

In [43]:

```
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = dat.drop(273, axis=1)
y = dat[273]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

regr = LinearRegression()

regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)
```

In [44]:

```
import sklearn.metrics as sm
print("Mean absolute error =",round(sm.mean_absolute_error(y_test, y_pred),4))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_pred),4))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_pred),4))
print("Explain variance score =", round(sm.explained_variance_score(y_test, y_pred),4))

print("R2 score =", 100*round(sm.r2_score(y_test, y_pred),4))
```

```
Mean absolute error = 0.0723
Mean squared error = 0.0093
Median absolute error = 0.0574
Explain variance score = 0.8071
R2 score = 80.71000000000001
```

Final Model using linear regression obtained an accuracy of 81% with mean abs error = 0.0718

1.Removed columns with high amount of null values and imputed with mean for the columns with low null values.

2.Imputed Outliers with 0.75 percentile.

3. Standardized the data

5 REPORT

5.1 List of any assumptions that you made

A1. The linear regression model is “linear in parameters.” (parameters are alpha and beta values)

```
 From fig1(below) the red line near to the dense cluster is flat indicating linear
ity in parameters
```

A2. There is a random sampling of observations.

A3. The residual mean should be zero

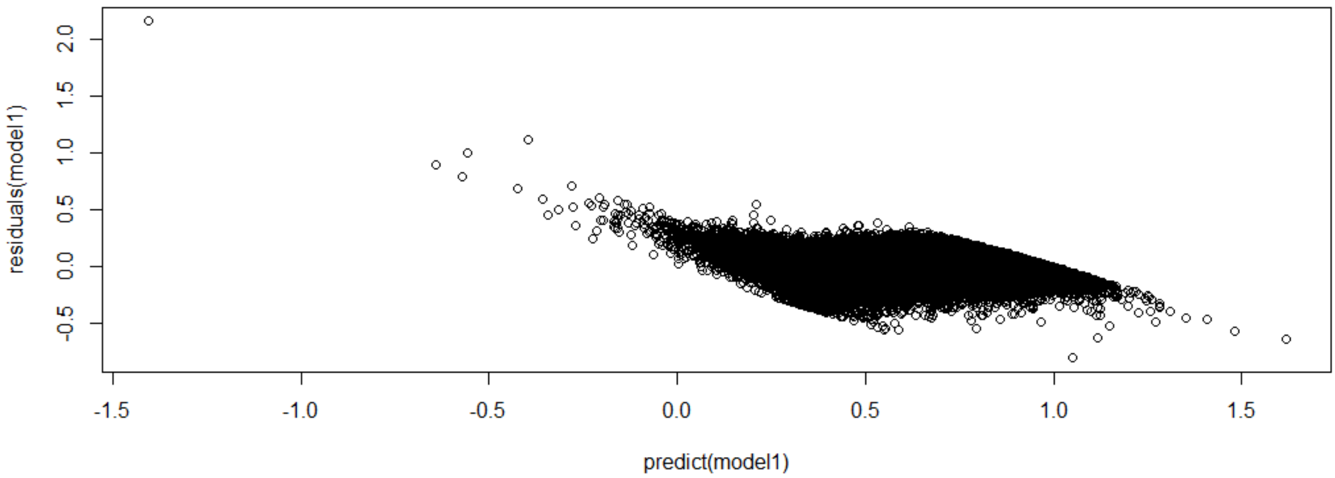
```
 In R file i showed mean of residuals is zero that is assumption A3 is also prove
d.
```

A4. There is no multi-collinearity (or perfect collinearity).

A5. Spherical errors: There is homoscedasticity and no autocorrelation

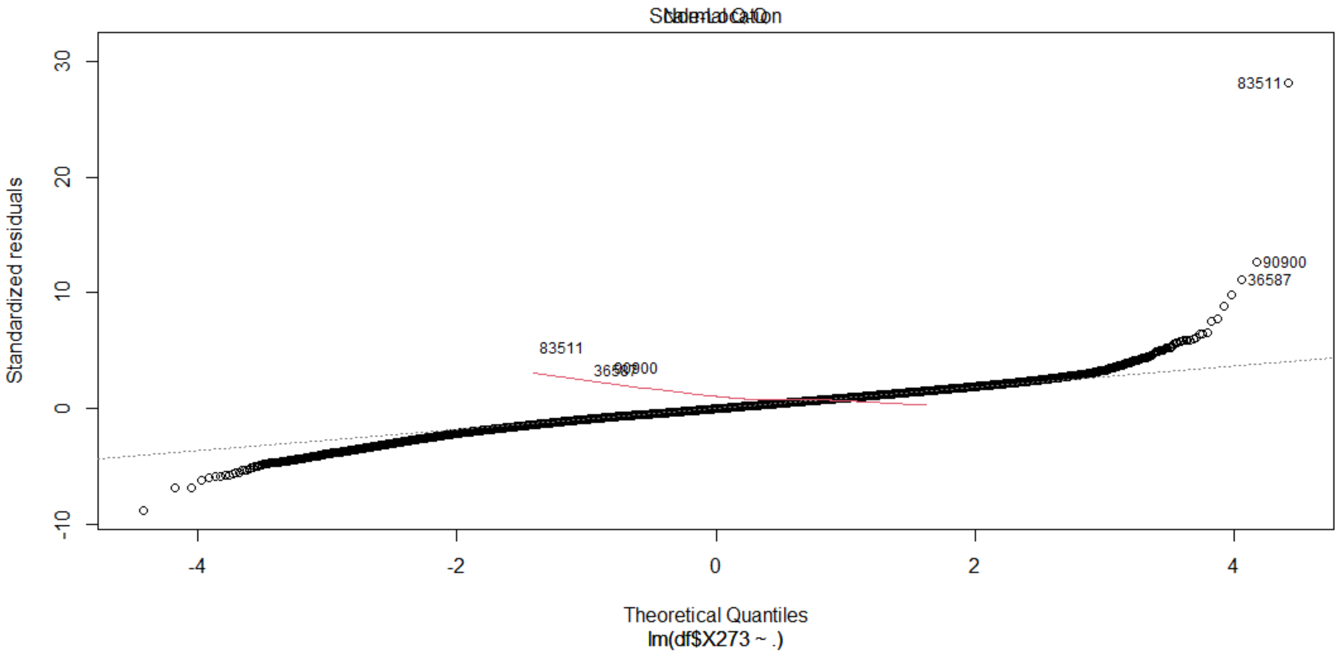
```
The below figure  proves my A5 assumptions since error varies with constant varian
ce with respect to variablesnjkkhyug
```

fig:1



A6: Optional Assumption: Error terms should be normally distributed

fig2:



The above plot is a Q-Q- plot or Quantile - Quantile chart, X axis as theorectical X and Y axis as standardized residuals.

if standardized error is linear w.r.t my Theorectical value then we can say my residuals are normally distributed.

Therefore from the above figure i can prove my A6 assumption(linearity is shown by dotted black line).

I also assumed that y or error are independent, that totally depends upon the method of data collection.

Note: A reasonable number of assumption been satisfied is enough to go with OLS(real life scenario)

5.2 Description of your methodology and solution path.

My aim was to predict a continous variable so i selected regression algorithm class, among this class linear regression is the simplest model that one can find.

Sklearn linear Regression algorithm is purely based on OLS, so before i fit my data into it i have to validate my data against the OLS assumption.

For OLS assumption validation i used R programming

I created a standardized data and loaded it into my R notebook, fitted my data into a linear model using the lm() function and obtained the summary data, most of the assumption validation is done through plots as explained above, i created residual plots and did a thorough study of all plots and proved my assumptions to be satisfied.

After i made surity about OLS assumptions i decided to go ahead with my model.

Python

I created a baseline model using linear regression algorithm, Before fitting my data to baseline model i removed columns with high level of missing data and low collinearity with my target. And imputed the columns with low level of missing values and resonable level of collinearity with mean of respective columns. To improve model performance i checked for outliers in each of my columns and imputed outliers with 0.75 percentile, becuase the data points are showing exponential increase so imputed the maximum value in my data's bound of IQR. For the sake of statisfying most of assumption and improve the understanding level of algorithm, i standardized my data so that model won't consider the column with high magnitude as judging factor. After continous iteration and improving my base model i created my final model with 81% accuracy and 0.07(<3) mean absolute error.

5.3 List of algorithms and techniques you used

1. Created a function that shows the percentage of missing values in each of my column.
2. Used pandas to drop and impute missing values
3. Linear Regression algorithm from sklearn.
4. Train_Test_Split from sklearn.
5. Evaluation Metrics from sklearn.

```
I selected most relevant  metrics from  sklearn such as mean absolute error,med
ian absolute error,mean squared error            explained variance score and R
squared foe the evaluation of my models performance.
```

6. linear model algorithm in R packages

```
   Used to fit and obtain residual plots to determine and to prove my assumptio
ns stated above.
```

7. corrgram library

```
   Used to plot correlation matrix to check multicollinearity assumption.
```

8. corrplot library

```
   Used to creat a correlation matrix.
```

5.4 List of tools and frameworks you used

1. Python

   1.Numpy

2.Pandas

3.Seaborn

4.Sklearn_linearRegression

5.Sklearn_Train_Test_Split

6.Sklearn_Evaluation Metrics

2. R Programming
   A. linear model algorithm in R packages
   B. corrgram library
   C. corrplot library

5.5 Results and evaluation of your models

1.Base line Model: Mean absolute error = 38.7395

```
Mean squared error = 2549.7368

Median absolute error = 31.0812

Explain variance score = 0.8183
R2 score = 81.83
```

Accuracy of my baseline model is 81% but it has a mean abs error of 38.7

2. So i removed my ouliers and run my model again:

```
Mean absolute error = 38.777

Mean squared error = 2546.7118

Median absolute error = 31.2474

Explain variance score = 0.819

R2 score = 81.89999999999999
```

Accuracy and mean abs error didn't changed much but mean squared error dropped slightely.

3. After standardization:

```
Mean absolute error = 0.0714

Mean squared error = 0.0087

Median absolute error = 0.0571

Explain variance score = 0.8196

R2 score = 81.96
```

Accuracy is close to 82% that is 82% of variation in Y can be predicted using my model. Most importantly mean absolute error dropped significantly to 0.0714.

In [ ]:

```

```