MACHINE LEARNING TECHNIQUES

Case Study: Problem of Regression on Real Estate

GROUP - 4

Aanchal Srivastava

Anmol Sethi

Pragya Sajwan

Shivani Jogi

Swarna Dubey

Yash Bansal

Introduction to the dataset:

	A	B	C	D	E	F	G	H	I	J
1	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE
2	P03210	Karapakkam	1004	131	1	1	3	AbNormal	1	Commercial
3	P09411	Anna Nagar	1986	26	2	1	5	AbNormal	0	Commercial
4	P01812	Adyar	909	70	1	1	3	AbNormal	1	Commercial
5	P05346	Velachery	1855	14	3	2	5	Family	0	Others
6	P06210	Karapakkam	1226	84	1	1	3	AbNormal	1	Others

	K	L	M	N	O	P	Q	R	S
1	UTILITY_AVAIL	STREET	MZZONE	QS_ROOMS	QS_BATHROOM	QS_BEDROOM	QS_OVERALL	COMMIS	SALES_PRICE
2	AllPub	Paved	A	4	3.9	4.9	4.33	144400	7600000
3	AllPub	Gravel	RH	4.9	4.2	2.5	3.765	304049	21717770
4	ELO	Gravel	RL	4.1	3.8	2.2	3.09	92114	13159200
5	NoSewr	Paved	I	4.7	3.9	3.6	4.01	77042	9630290
6	AllPub	Gravel	C	3	2.5	4.1	3.29	74063	7406250

(7109 rows * 19 Columns)

Multiple Linear Regression

- Accessing the data
- Importing Libraries - Pandas, Numpy, Matplotlib and Seaborn
- Reading the data file
- Info of the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
pred = pd.read_csv("chennai_house_price_prediction.csv")
```

```
pred.info()
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   PRT_ID          7109 non-null   object
 1   AREA            7109 non-null   object
 2   INT_SQFT        7109 non-null   int64
 3   DIST_MAINROAD   7109 non-null   int64
 4   N_BEDROOM       7108 non-null   float64
 5   N_BATHROOM      7104 non-null   float64
 6   N_ROOM          7109 non-null   int64
 7   SALE_COND       7109 non-null   object
 8   PARK_FACIL      7109 non-null   int64
 9   BUILDTYPE       7109 non-null   object
 10  UTILITY_AVAIL   7109 non-null   object
 11  STREET          7109 non-null   object
 12  MZZONE          7109 non-null   object
 13  QS_ROOMS        7109 non-null   float64
 14  QS_BATHROOM     7109 non-null   float64
 15  QS_BEDROOM      7109 non-null   float64
 16  QS_OVERALL      7061 non-null   float64
 17  COMMIS          7109 non-null   int64
 18  SALES_PRICE     7109 non-null   int64
```

Data Preprocessing

- Setting the Target (y) as SALES PRICE and other variables as features (X)

```
# target
y=pred['SALES_PRICE']
y
```

```
## features

X= pred.drop(['SALES_PRICE'], axis=1)
X
```

- Dropping irrelevant features

```
# Drop irrelevant features

X_1 =X.drop(['PRT_ID','SALE_COND','UTILITY_AVAIL'],axis=1)
```

- Using unique() to look at unique values and creating dummies for few variables

```
# Convert categorical to numeric using one hot encoding

X_1 = pd.get_dummies(X_1,columns=['AREA','BUILDTYPE','STREET','MZZONE'])
```

Data Preprocessing

- Importing statsmodels and adding constant 1 to each row

```
import statsmodels.api as sm
X_1 = sm.add_constant(X_1)
X_1
```

```
X_1.shape
```

```
(7109, 45)
```

- Treating missing values; filling null values using fillna with mean of those variables

```
X_1.isnull().sum()
```

```
X_1['QS_OVERALL'].fillna(value=X_1['QS_OVERALL'].mean(), inplace=True)
```

```
X_1['N_BEDROOM'].fillna(value=X_1['N_BEDROOM'].mean(), inplace=True)
```

```
X_1['N_BATHROOM'].fillna(value=X_1['N_BATHROOM'].mean(), inplace=True)
```

Splitting data into Train and Test

```
from sklearn.model_selection import train_test_split
```

```
X_train_1, X_test_1, y_train_1, y_test_1 =train_test_split(X_1,y ,test_size =0.2, random_state=10)
```

```
X_train_1.shape, X_test_1.shape, y_train_1.shape, y_test_1.shape
```

```
((5687, 45), (1422, 45), (5687,), (1422,))
```

- Taking a look at X_train and X_test values

```
X_train_1
```

5687 rows × 45 columns

```
X_test_1
```

1422 rows × 45 columns

Building the model

```
mlr_1 = sm.OLS(y_train_1, X_train_1)
```

- Training the model

```
mlr_1 = mlr_1.fit()
```

- Using params to look at parameters and their relation to dependent variable

```
mlr_1.params
```

```
const                2.700263e+06
INT_SQFT             3.782721e+03
DIST_MAINROAD       -6.888493e+01
N_BEDROOM            2.823435e+05
N_BATHROOM          -4.031941e+05
N_ROOM               1.206859e+05
PARK_FACIL           9.914752e+05
QS_ROOMS            -1.695507e+04
QS_BATHROOM         -1.560937e+04
QS_BEDROOM           2.789208e+03
QS_OVERALL           8.513923e+04
COMMIS               2.840802e+00
AREA_Adyar           4.579575e+05
AREA_Adyr            1.168757e+06
AREA_Ana Nagar       4.596758e+06
AREA_Ann Nagar       1.213446e+06
AREA_Anna Nagar      1.912341e+06
AREA_Chormpet       -9.981541e+03
AREA_Chrmpet         5.050556e+05
AREA_Chrompet       -3.400180e+03
AREA_Chrompt         1.940305e+05
AREA_KK Nagar       -1.692252e+06
AREA_KKNagar        -2.066746e+06
AREA_Karapakam      -2.502585e+06
AREA_Karapakkam     -2.033962e+06
AREA_T Nagar         2.051998e+06
AREA_TNagar          2.221258e+06
AREA_Velachery      -1.474939e+06
AREA_Velchery       -1.837474e+06
BUILDTYPE_Comercial  2.026595e+06
BUILDTYPE_Commercial 3.164816e+06
BUILDTYPE_House     -1.349369e+06
BUILDTYPE_Other     -4.606319e+05
BUILDTYPE_Others    -6.811467e+05
STREET_Gravel        1.116480e+06
STREET_No Access     4.600433e+03
STREET_NoAccess      4.874171e+05
STREET_Pavd          4.796557e+05
STREET_Paved         6.121090e+05
MZZONE_A            -9.312163e+05
MZZONE_C            -4.019670e+05
MZZONE_I             1.401905e+05
MZZONE_RH            7.031580e+05
MZZONE_RL            1.286094e+06
MZZONE_RM            1.904003e+06
dtype: float64
```

Diagnosing the model

```
mlr_1.summary2()
```

Model:	OLS	Adj. R-squared:	0.952
Dependent Variable:	SALES_PRICE	AIC:	171092.7797
Date:	2022-11-10 13:57	BIC:	171365.2632
No. Observations:	5687	Log-Likelihood:	-85505.
Df Model:	40	F-statistic:	2827.
Df Residuals:	5646	Prob (F-statistic):	0.00
R-squared:	0.952	Scale:	6.7625e+11

Omnibus:	132.790	Durbin-Watson:	2.043
Prob(Omnibus):	0.000	Jarque-Bera (JB):	192.697
Skew:	0.255	Prob(JB):	0.000
Kurtosis:	3.743	Condition No.:	11448429555476168

- Already receiving a high R-squared value in the first iteration because of a good fit. However, we need to eliminate unnecessary variables and to look at RMSE Value

Multicollinearity

- Importing variance_inflation_factor

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

- Checking for VIF values for features i.e. independent variables:

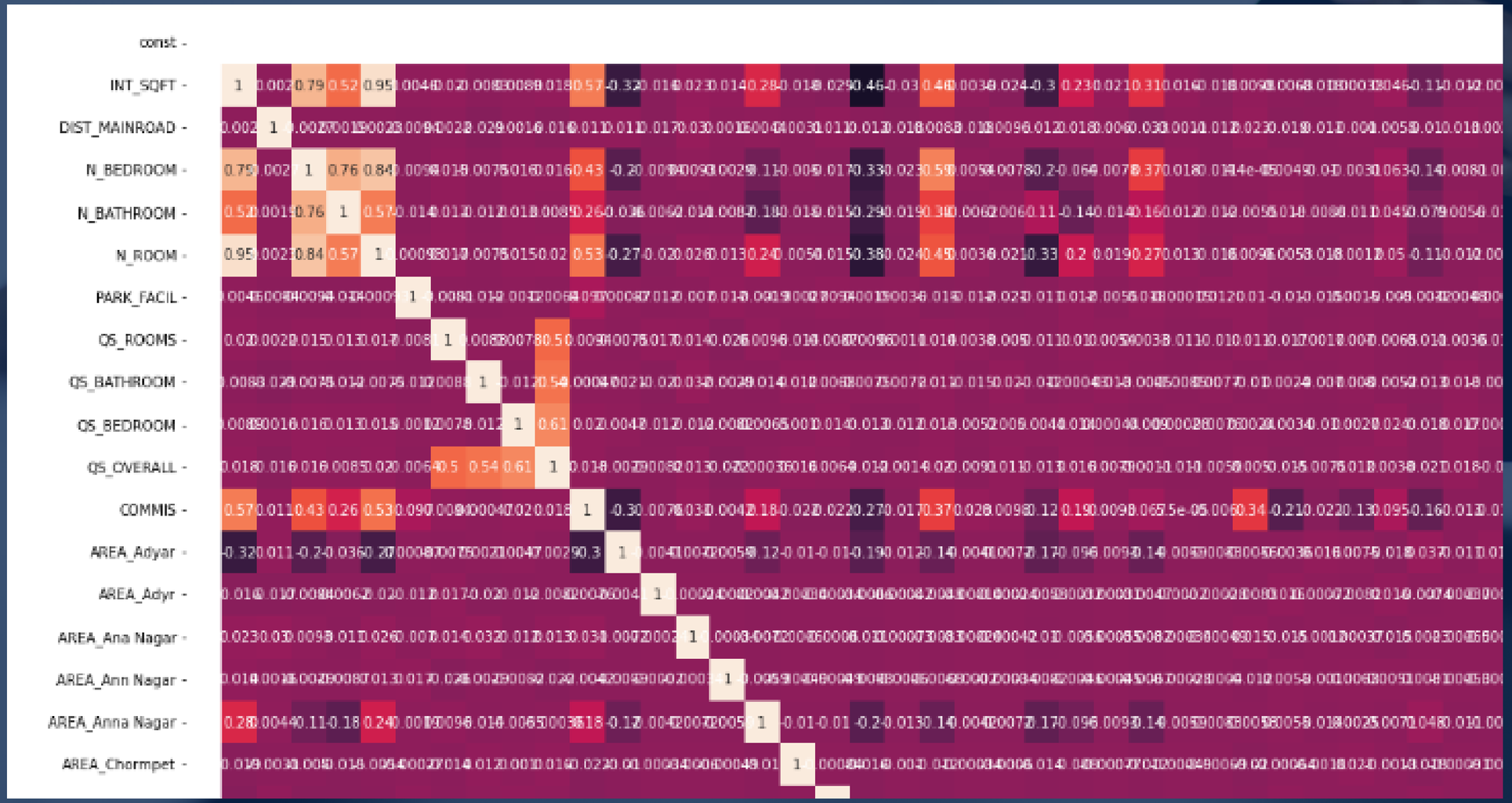
```
#vif > 4
def var_inf_factor(data):
    vif=pd.DataFrame()
    vif['Feature']=data.columns
    vif['VIF_Value']= [variance_inflation_factor(data.values,i)
                       for i in range(data.shape[1])]
    print(vif)
```

```
var_inf_factor(X_1)
```

	Feature	VIF_Value
0	const	0.000000
1	INT_SQFT	18.539409
2	DIST_MAINROAD	1.006672
3	N_BEDROOM	13.567914
4	N_BATHROOM	3.577235
5	N_ROOM	21.475609
6	PARK_FACIL	1.023800
7	QS_ROOMS	4.012710
8	QS_BATHROOM	4.620034
9	QS_BEDROOM	5.649446
10	QS_OVERALL	12.293831
11	COMMIS	2.110629
12	AREA_Adyar	inf
13	AREA_Adyr	inf
14	AREA_Ana Nagar	inf
15	AREA_Ann Nagar	inf

Plotting the heatmap

```
plt.figure(figsize=(25,25))
sns.heatmap(X_1.corr(),annot=True);
```



Eliminating variables with high correlation

```
# eliminating variables with high correlation
features_to_drop_1=['N_BEDROOM','N_BATHROOM','N_ROOM','QS_ROOMS','QS_BATHROOM','QS_BEDROOM']
```

```
features_2=list(set(X_1.columns)-set(features_to_drop_1))
```

- We are left with following features

```
features_2
```

```
['QS_OVERALL',
 'STREET_NoAccess',
 'AREA_Ann Nagar',
 'MZZONE_RH',
 'STREET_Pavd',
 'AREA_KK Nagar',
 'AREA_Adyr',
 'PARK_FACIL',
 'BUILDTYPE_Others',
 'COMMIS',
 'AREA_TNagar',
 'INT_SQFT',
 'AREA_Karapakam',
 'MZZONE_RL',
 'AREA_Velchery',
 'BUILDTYPE_Other',
 'AREA_Ana Nagar',
 'MZZONE_I',
 'AREA_Velachery',
 'BUILDTYPE_Comercial',
 'AREA_T Nagar',
 'STREET_Paved',
 'AREA_Chormpet',
 'AREA_Karapakkam',
 'AREA_Chrmpet',
 'DIST_MAINROAD',
 'AREA_Adyar',
 'AREA_Chrompet',
 'MZZONE_C',
 'BUILDTYPE_Commercial',
 'AREA_Anna Nagar',
 'STREET_Gravel',
 'AREA_Chrompt',
 'const',
 'MZZONE_RM',
 'BUILDTYPE_House',
 'STREET_No Access',
 'MZZONE_A',
 'AREA_KKNagar']
```

```
len(features_2)
```

```
39
```

Continuing with the new feature set

```
X_2=X_1[features_2]
```

- Splitting into train and test

```
# Splitting X_2 to train and test

X_train_2,X_test_2,y_train_2,y_test_2=train_test_split(
                    X_2,y,test_size=0.2, random_state=10)
```

```
X_train_2.shape,X_test_2.shape,y_train_2.shape,y_test_2.shape
```

```
((5687, 39), (1422, 39), (5687,), (1422,))
```

```
# Building the model

mlr_2=sm.OLS(y_train_2,X_train_2)
```

```
# Fit

mlr_2=mlr_2.fit()
```

```
# Diagnosis

mlr_2.summary2()
```

Model:	OLS	Adj. R-squared:	0.951
Dependent Variable:	SALES_PRICE	AIC:	171186.9640
Date:	2022-11-10 13:58	BIC:	171419.5719
No. Observations:	5687	Log-Likelihood:	-85558.
Df Model:	34	F-statistic:	3264.
Df Residuals:	5652	Prob (F-statistic):	0.00
R-squared:	0.952	Scale:	6.8827e+11

Continuing with the new feature set

- Picking variables with VIF Values < 4

```
# VIF

var_inf_factor(X_2)
```

	Feature	VIF_Value
0	QS_OVERALL	1.004536
7	PARK_FACIL	1.022527
8	BUILDTYPE_Others	inf
9	COMMIS	2.106964
10	AREA_TNagar	inf
11	INT_SQFT	3.832909
25	DIST_MAINROAD	1.005898

- Now, there is no multicolinearity

- Picking variables with p < 0.05 and creating third feature set X_3

```
# Features with p < 0.05
features_3=['INT_SQFT','PARK_FACIL',
            'DIST_MAINROAD','COMMIS',
            'QS_OVERALL']
```

```
X_3=X_2[features_3]
```

- Followed by Splitting into train and test , building the model and training the model

Summary of the third feature set

```
mlr_3.summary2()
```

Model:	OLS	Adj. R-squared (uncentered):	0.945
Dependent Variable:	SALES_PRICE	AIC:	184581.8116
Date:	2022-11-10 13:59	BIC:	184615.0413
No. Observations:	5687	Log-Likelihood:	-92286.
Df Model:	5	F-statistic:	1.961e+04
Df Residuals:	5682	Prob (F-statistic):	0.00
R-squared (uncentered):	0.945	Scale:	7.2938e+12

```
var_inf_factor(X_3)
```

	Feature	VIF_Value
0	INT_SQFT	13.491140
1	PARK_FACIL	2.005173
2	DIST_MAINROAD	3.771129
3	COMMIS	6.324505
4	QS_OVERALL	10.910225

- Picking the variables with least VIF Values for the fourth feature set

Continuing with the new feature set

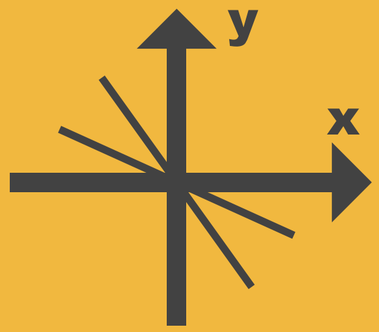
```
features_4=['DIST_MAINROAD','PARK_FACIL']
X_4=X_3[features_4]
```

- Again splitting into train and test , building the model and training the model and looking at the summary

```
mlr_4.summary2()
```

Model:	OLS	Adj. R-squared (uncentered):	0.740
Dependent Variable:	SALES_PRICE	AIC:	193432.4468
Date:	2022-11-10 14:00	BIC:	193445.7386
No. Observations:	5687	Log-Likelihood:	-96714.
Df Model:	2	F-statistic:	8092.
Df Residuals:	5685	Prob (F-statistic):	0.00
R-squared (uncentered):	0.740	Scale:	3.4600e+13

Our first model!




```
mlr_4.params
```

```
DIST_MAINROAD     6.202485e+04
PARK_FACIL        5.273678e+06
dtype: float64
```

The model is:

SOLD PRICE = DIST_MAINROAD * 6.202485e+04 + PARK_FACIL * 5.273678e+06

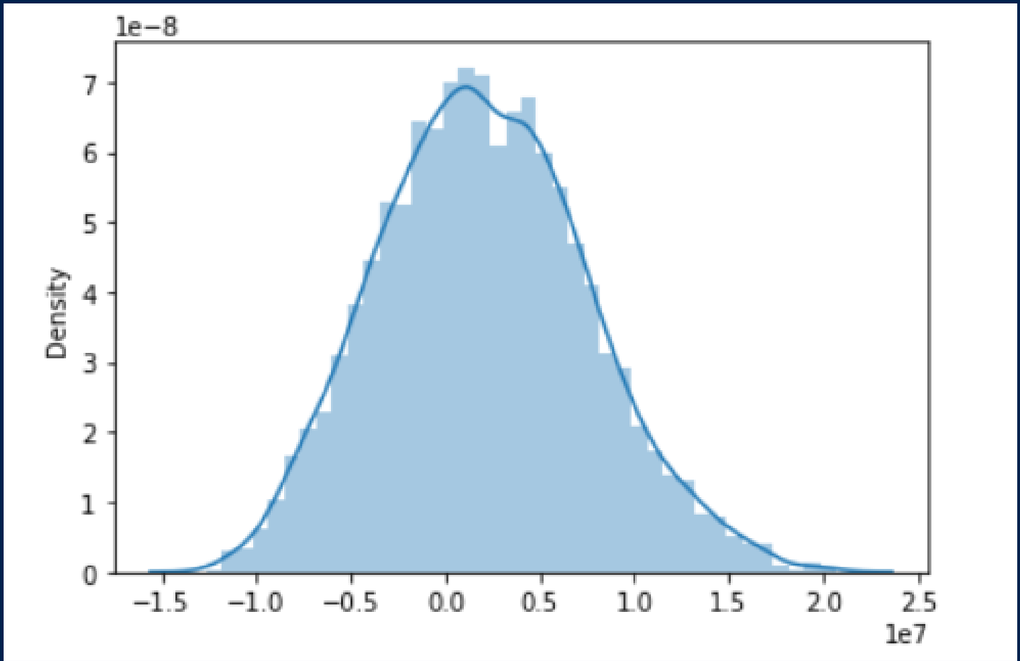
- The above linear equation shows the relationship between chosen independent variables (Distance from the mainroad & Parking Facility) and dependent variable (Sold Price)

Residual Analysis

- Checking the Normality and plotting dist plot and prob plot

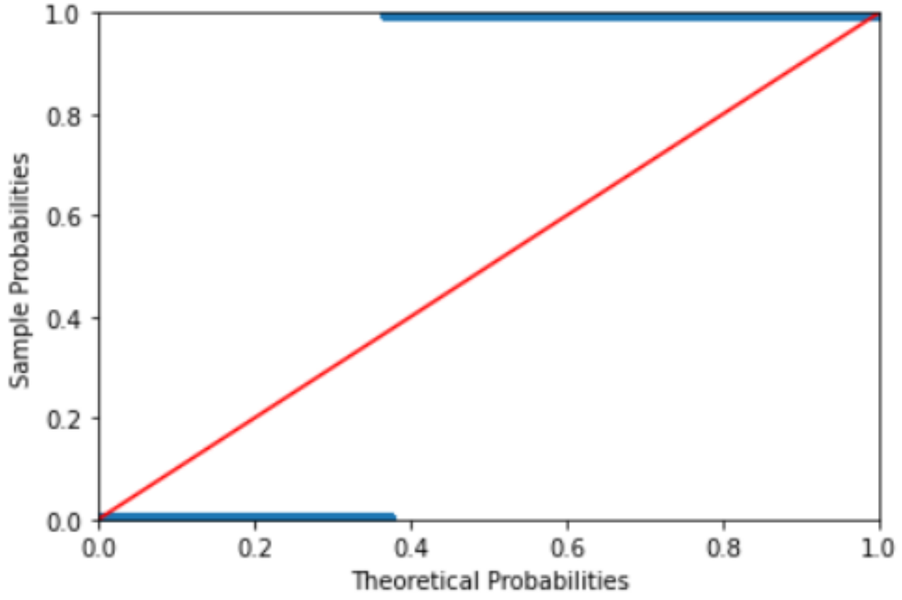
```
mlr_4.resid
```

```
sns.distplot(mlr_4.resid);
```




```
## Prob prob plot

def prob_prob_plot(model):
    probplot=sm.ProbPlot(model.resid)
    probplot.ppplot(line='45')
    plt.show();
```

```
prob_prob_plot(mlr_4)
```



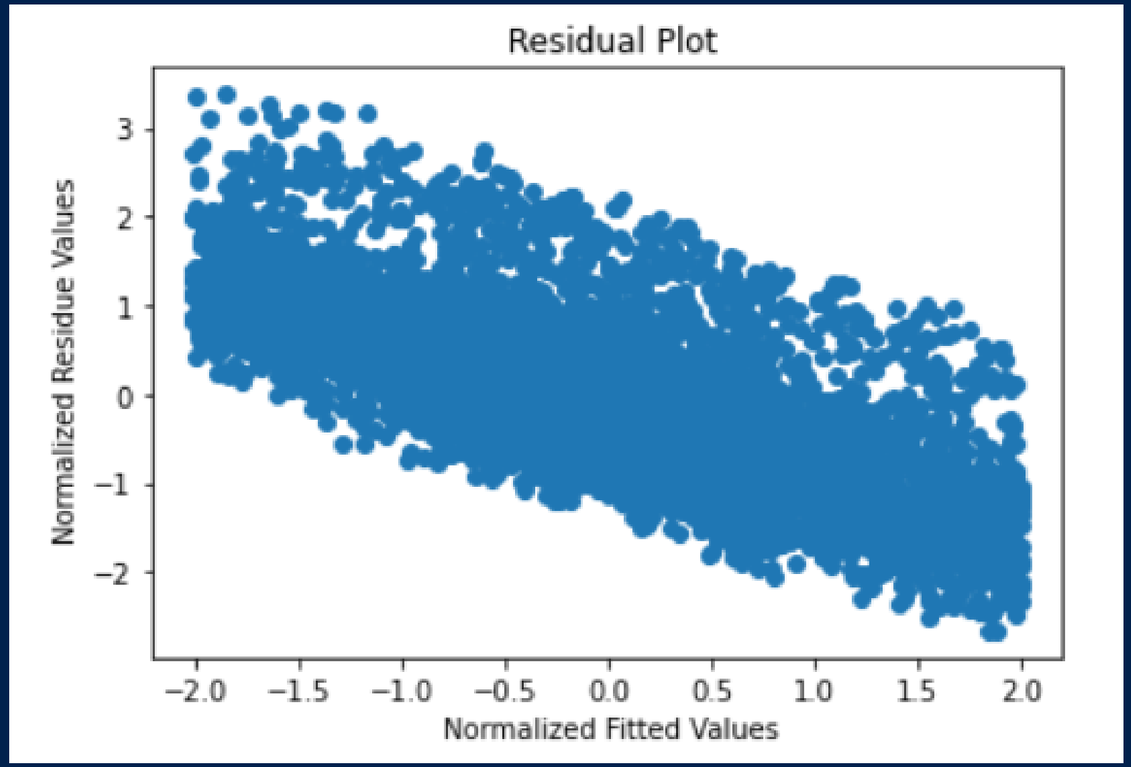
- Plot of residual values is not perfectly following normal distribution

Checking Homoscedasticity

```
def standard(data):
    return (data-data.mean())/data.std()
```

```
# Plotting residual plot

def residual_plot(model):
    plt.scatter(standard(model.fittedvalues),
                standard(model.resid))
    plt.xlabel(' Normalized Fitted Values')
    plt.ylabel(' Normalized Residue Values')
    plt.title(' Residual Plot')
    plt.show();
```

```
# Calling the function

residual_plot(mlr_4)
```



- Parallel lines can be observed. Hence, the assumption of homoscedasticity is fulfilled

Checking for Outliers : Z Score, Cooks Distance and Leverage Distance

```
from scipy.stats import zscore

z_score=zscore(X_4)
```

```
z_score[z_score>3].count()
```

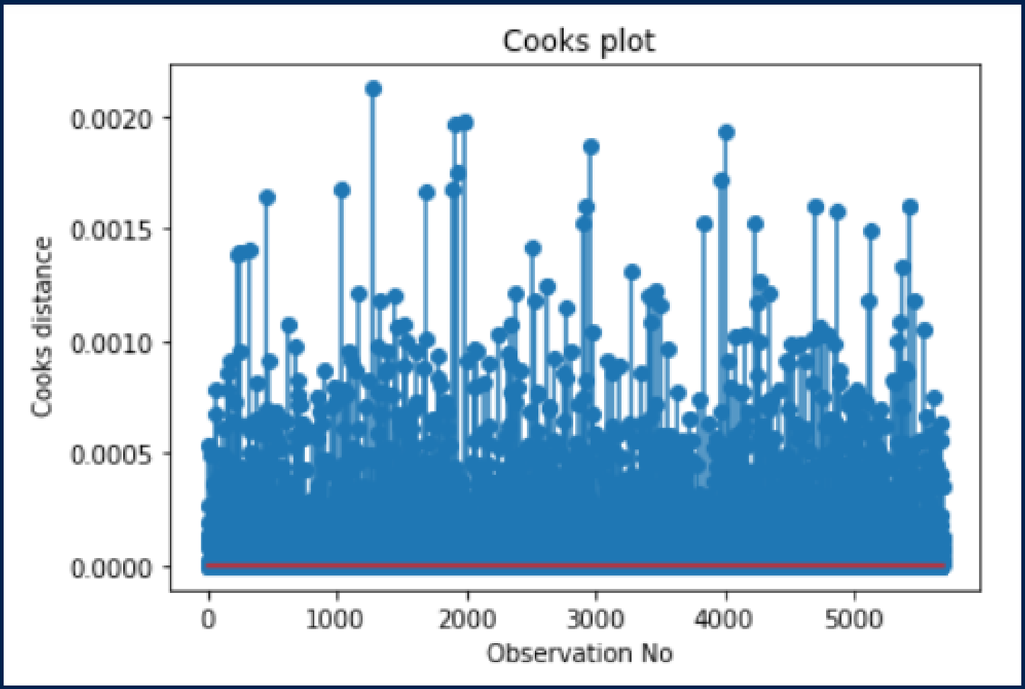
```
DIST_MAINROAD    0
PARK_FACIL       0
dtype: int64
```

```
z_score[z_score <-3].count()
```

```
DIST_MAINROAD    0
PARK_FACIL       0
dtype: int64
```

- No outliers found from both Z Score and Cooks Distance

```
def cooks_dist(model):
    model_influence=model.get_influence()
    (c,_)=model_influence.cooks_distance
    plt.stem(np.arange(len(X_train_4)),c)
    plt.xlabel(' Observation No')
    plt.ylabel(' Cooks distance')
    plt.title(' Cooks plot')
    plt.show();
```

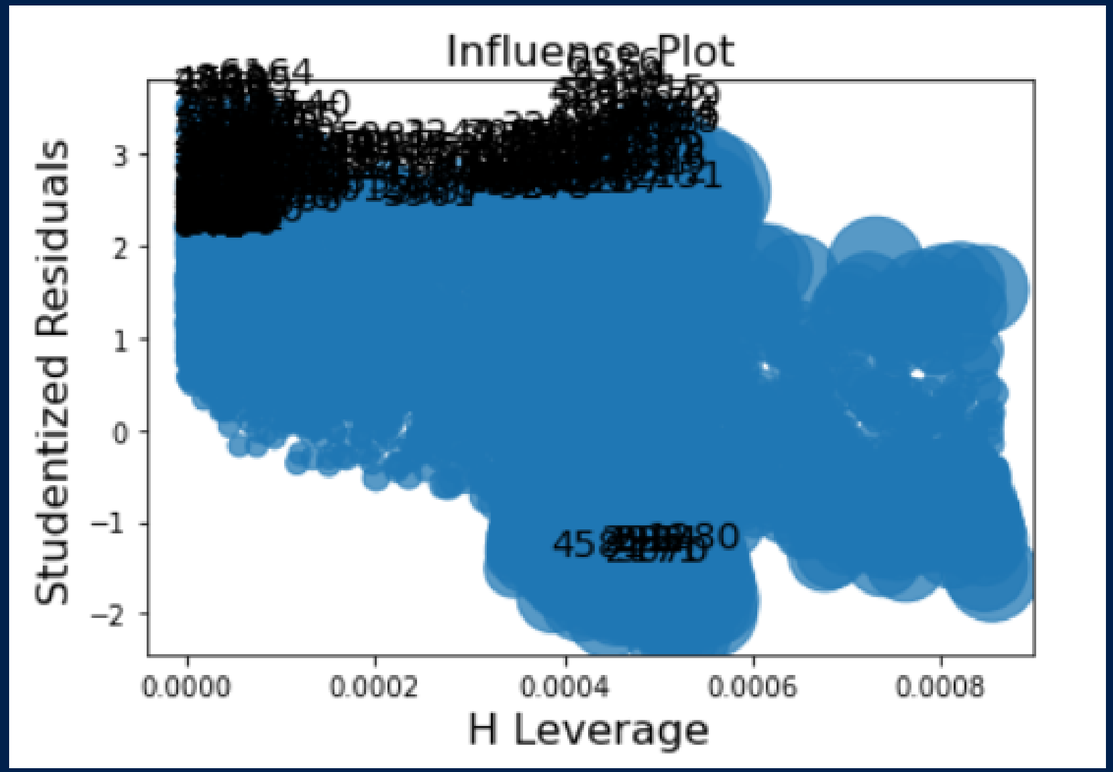


Checking for Outliers : Z Score, Cooks Distance and Leverage Distance

```
n=5687 # No of training data

k= 2 # No of features in the model

lev_cutoff= (3*(k+1))/n

print(' The Leverage cut off:',lev_cutoff)
```

```
 The Leverage cut off: 0.0015825567082820467
```

```
from statsmodels.graphics.regressionplots import influence_plot
influence_plot(mlr_4)
```



- No outliers found from Leverage Distance as well

Predicting the values and Performance of the model

```
y_pred= mlr_4.predict(X_test_5)
y_pred
```

```
461     8.786864e+06
1798    7.912134e+06
3655    1.615958e+07
5207    1.341043e+07
1125    1.874419e+06
            ...
5334    1.740919e+07
4302    9.849035e+06
3831    1.303555e+07
364     1.005938e+07
591     2.249303e+06
Length: 1422, dtype: float64
```

```
from sklearn.metrics import r2_score,mean_squared_error
```

```
r2= r2_score(y_test_5, y_pred)
print('R2:', r2)

mse = mean_squared_error(y_test_5, y_pred)

rmse= np.sqrt(mse)

print(' RMSE:',rmse)
```

```
R2: -1.325636213158024
 RMSE: 5833326.364981253
```

- RMSE Value is very high which is not acceptable.
  So, we need to transform the target variable

Transforming the target variable - Log and Square Root

- We tried applying the log function. However, it further increased the RMSE value. Hence, we move forward with applying square root
- Applying square root:

```
y_sq= np.sqrt(y)
```

```
y_sq.min(),y_sq.max()
```

```
(1468.6303142724516, 4864.909043342948)
```

- We again split the data into train and test , build the model, train the model and diagnose the model and carry out the whole process again till our model is ready

```
mlr_4.params
```

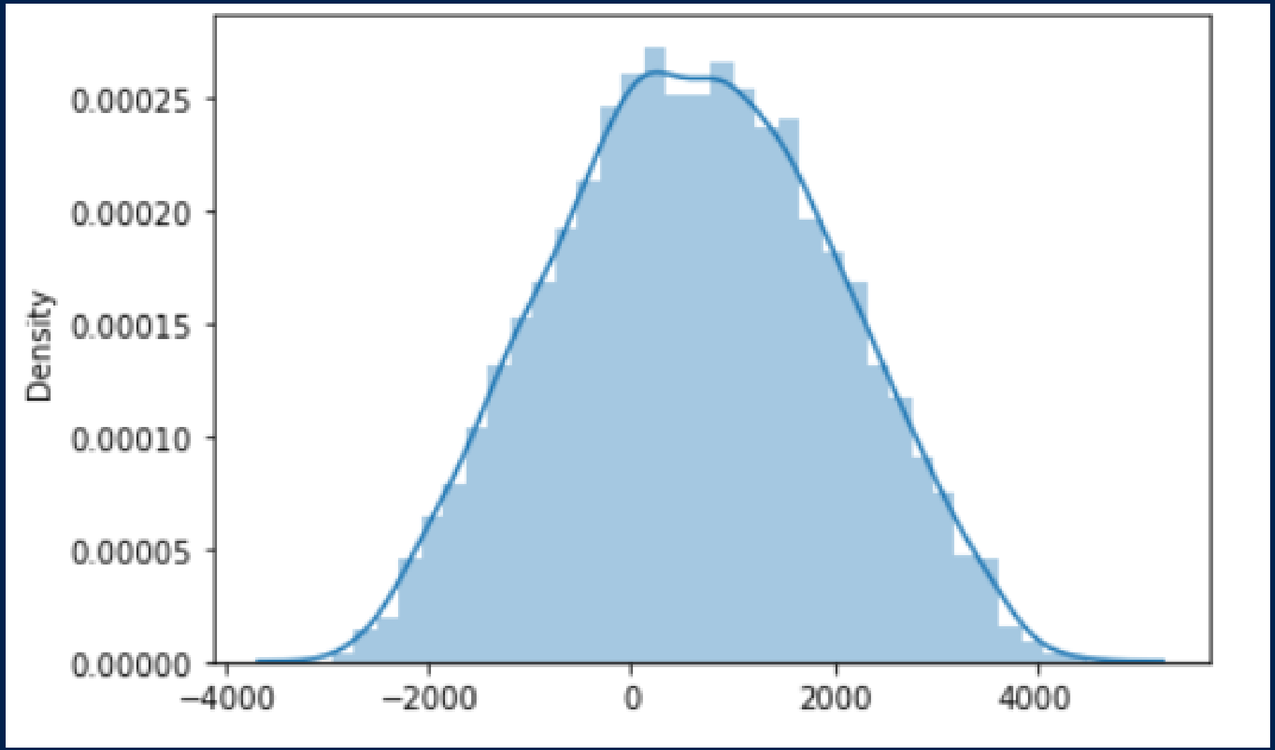
```
DIST_MAINROAD      18.938151
PARK_FACIL       1454.635790
dtype: float64
```

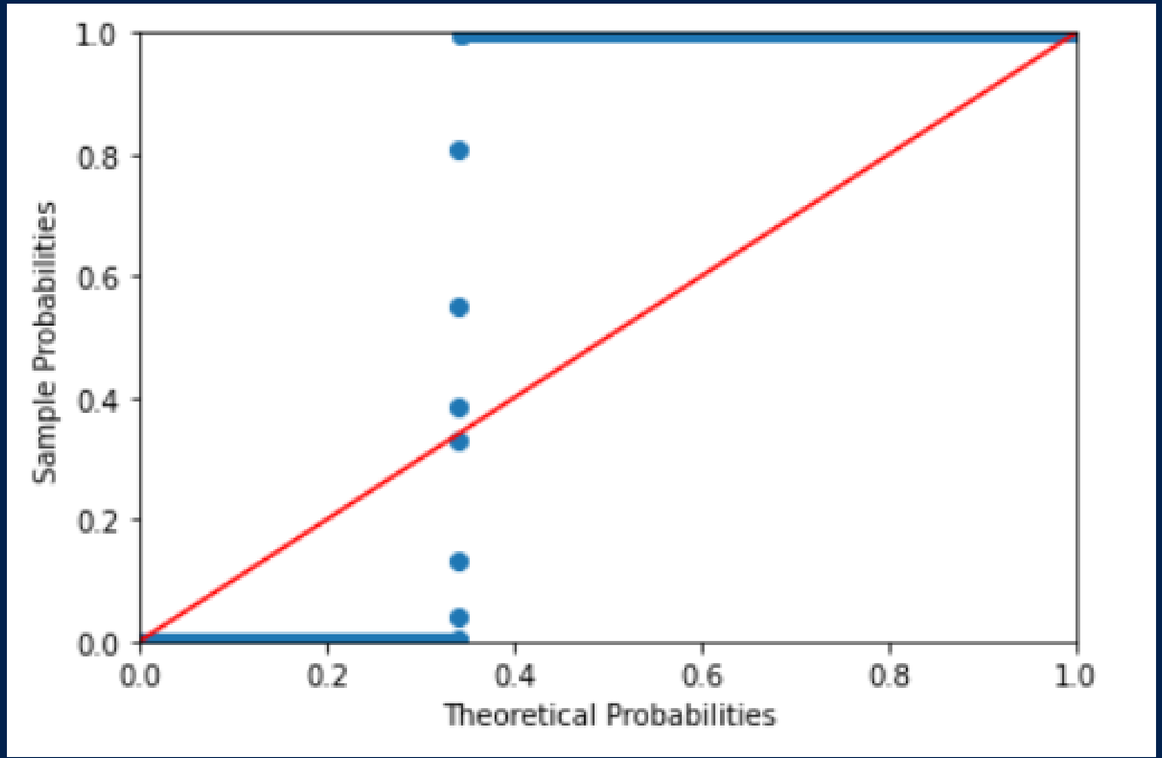
The model is:

SOLD PRICE = DIST_MAINROAD * 18.938151 +PARK_FACIL * 1454.635790

Residual Analysis after applying square root

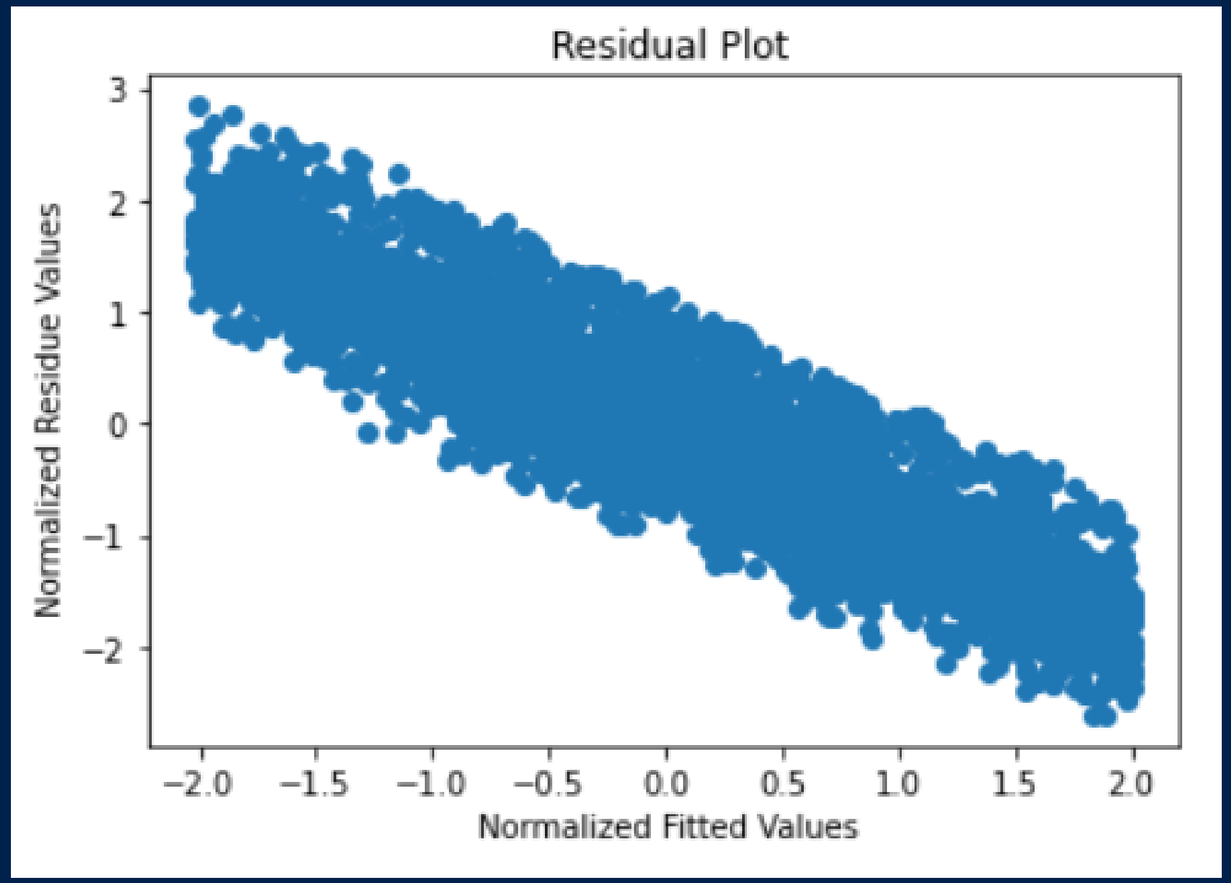
- Checking the Normality and plotting dist plot and prob plot





- Better dist plot, but still not perfectly following normal distribution.
- Values coming closer to the line in prob plot, but still not acceptable.

Checking Homoscedasticity and Outliers - Square root



- Clearer parallel lines can be observed. Hence, the assumption of homoscedasticity is fulfilled.
- No outliers found using Z Score, Cooks Distance and Leverage distance.

Predicting the values and Performance of the model after using square root

```
y_pred= mlr_5.predict(X_test_5)
y_pred
```

```
461     2529.101200
1798    2262.593284
3655    4775.382211
5207    3937.785902
1125     571.088393
           ...
5334    5156.107806
4302    2852.717956
3831    3823.568223
364     3064.841040
591      685.306071
Length: 1422, dtype: float64
```

```
from sklearn.metrics import r2_score,mean_squared_error
```

```
r2= r2_score(y_test_5, y_pred)
print('R2:', r2)

mse = mean_squared_error(y_test_5, y_pred)

rmse= np.sqrt(mse)

print(' RMSE:',rmse)
```

```
R2: -5.922108046253983
 RMSE: 1497.1597574649973
```

- RMSE Value reduced significantly after square rooting the target variable. So, we need to transform the target variable further in the same direction to get ideal results

Transforming the target variable - Cube Root

- Applying cube root:

```
y_cb= np.cbrt(y)
```

```
y_cb.min(),y_cb.max()
```

```
(129.2037123478417, 287.11098994433064)
```

- Moreover, we are changing one of the variables in the feature from Parking Facility to Overall Sq ft.

```
features_4=['DIST_MAINROAD','QS_OVERALL']
X_4=X_3[features_4]
```

- We again split the data into train and test , build the model, train the model and diagnose the model and carry out the whole process again till our model is ready

```
mlr_4.params
```

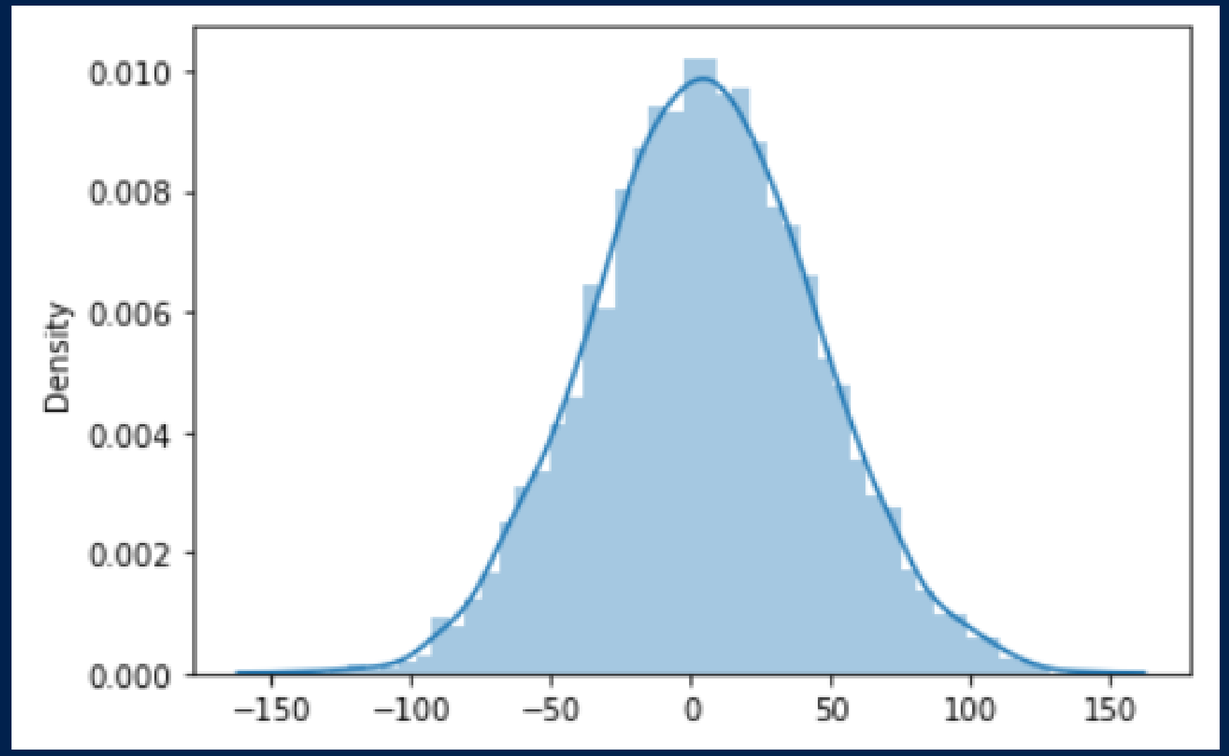
```
DIST_MAINROAD      0.153955
QS_OVERALL        56.702292
dtype: float64
```

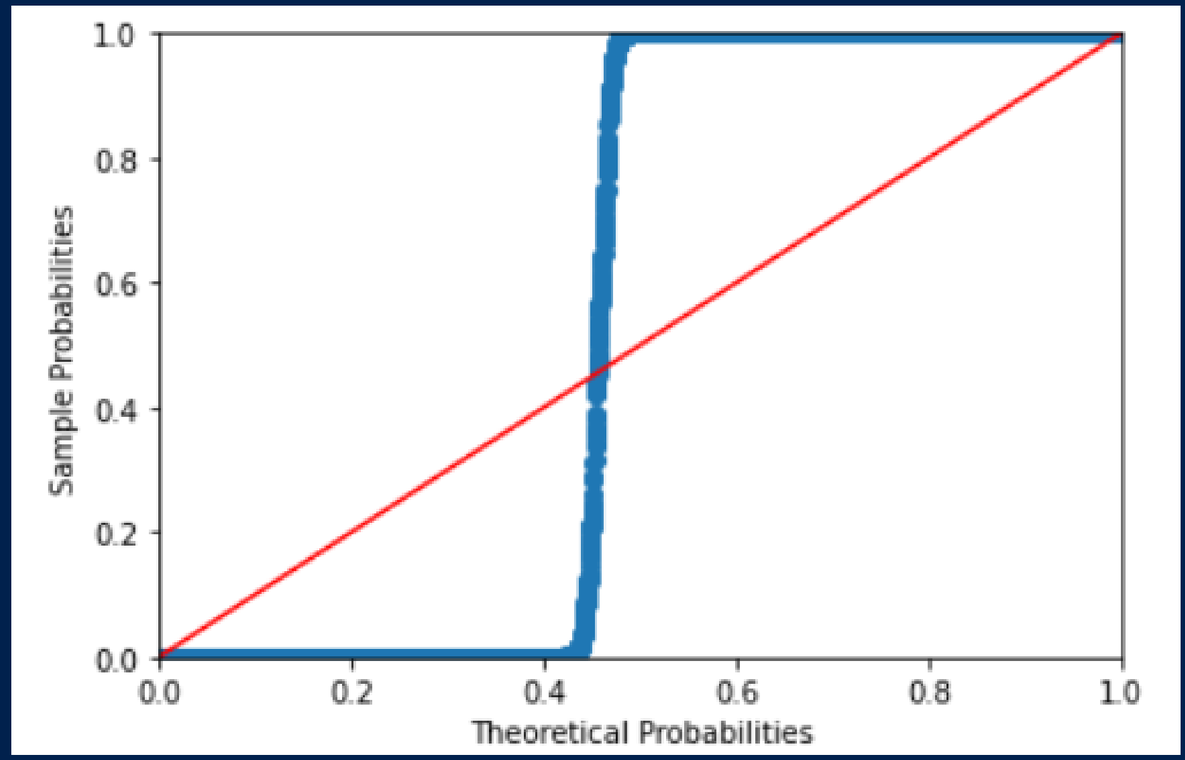
The model is:

SOLD PRICE = DIST_MAINROAD * 0.153955 + QS_OVERALL * 56.702292

Residual Analysis after applying cube root

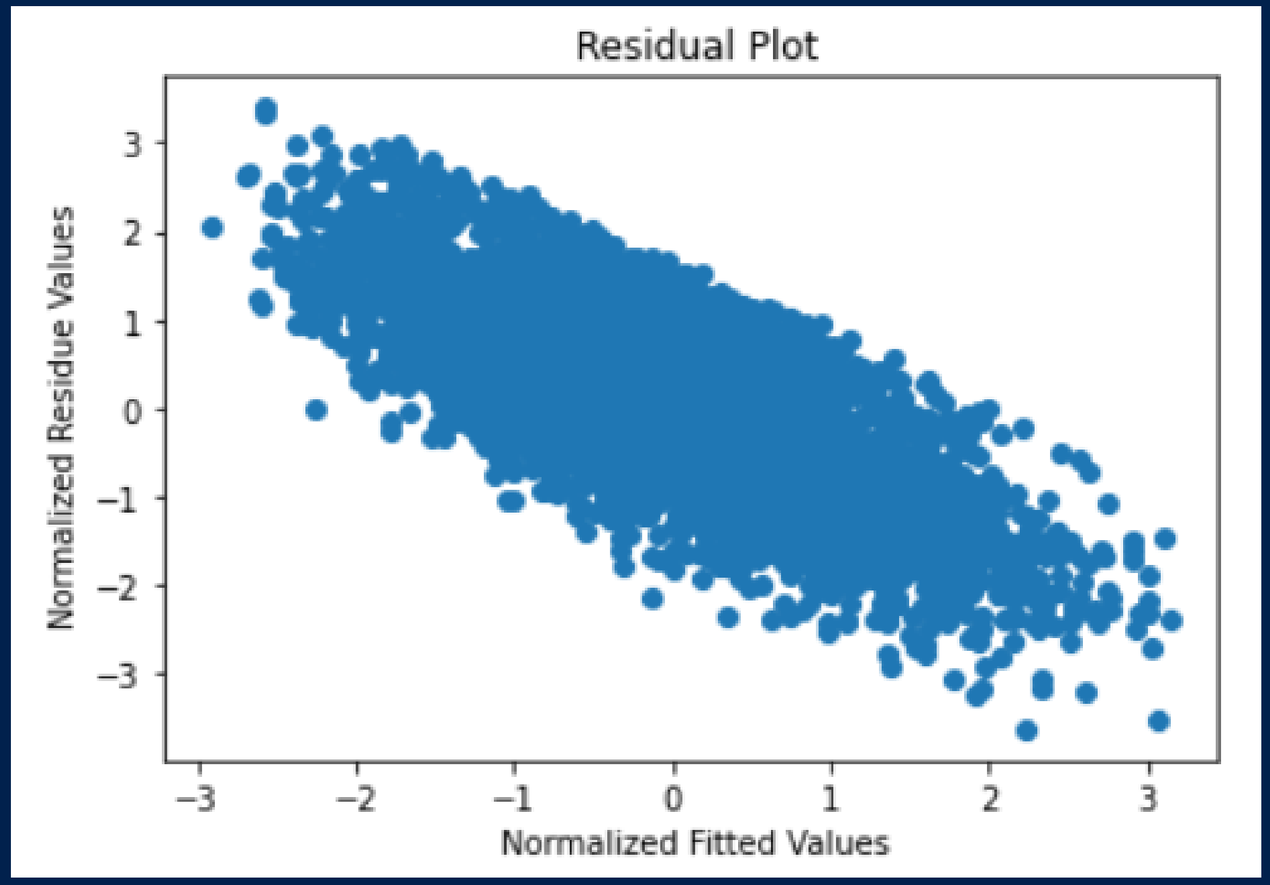
- Checking the Normality and plotting dist plot and prob plot





- Nearly perfect dist plot, but still not perfectly following normal distribution.
- Values coming closer to the line in prob plot, but still not acceptable.

Checking Homoscedasticity and Outliers - Cube root



- Parallel lines broadened, but still clearly parallel. Hence, the assumption of homoscedasticity is fulfilled.
- No outliers found using Z Score, Cooks Distance and Leverage distance.

Predicting the values and Performance of the model after using cube root

```
y_pred= mlr_5.predict(X_test_5)
y_pred
```

```
461      232.668615
1798     152.298837
3655     196.565696
5207     189.176651
1125     202.415812
            ...
5334     231.399716
4302     189.405113
3831     200.712836
364      239.161042
591      252.078859
Length: 1422, dtype: float64
```

```
from sklearn.metrics import r2_score,mean_squared_error
```

```
r2= r2_score(y_test_5, y_pred)
print('R2:', r2)

mse = mean_squared_error(y_test_5, y_pred)

rmse= np.sqrt(mse)

print(' RMSE:',rmse)
```

```
R2: -1.3973749871150498
 RMSE: 39.59720163414942
```

- RMSE Value further reduced significantly after cube rooting the target variable. So, we need to transform the target variable further in the same direction to get ideal results

Transforming the target variable - Quadruple Root

- Applying Quadruple root:

```
y_qr= np.sqrt(np.sqrt(y))
```

- We again split the data into train and test , build the model, train the model and diagnose the model and carry out the whole process again till our model is ready

```
mlr_4.params
```

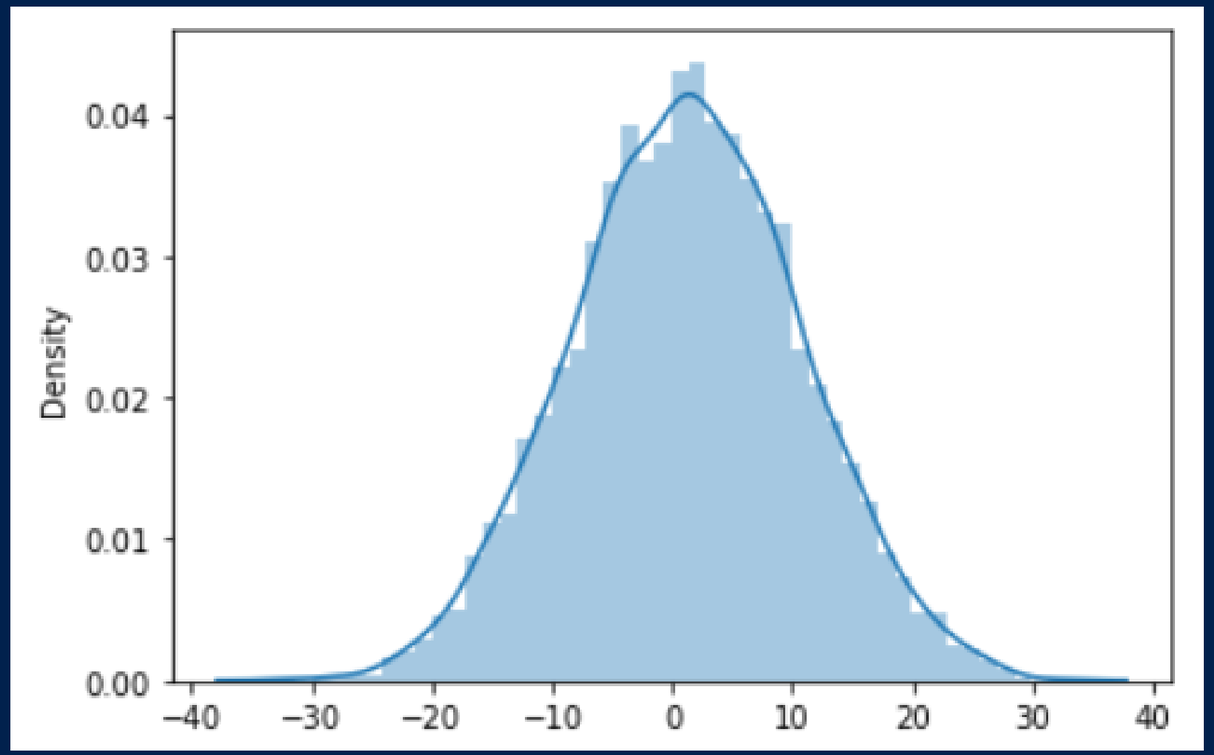
```
DIST_MAINROAD     0.039914
QS_OVERALL       14.723315
dtype: float64
```

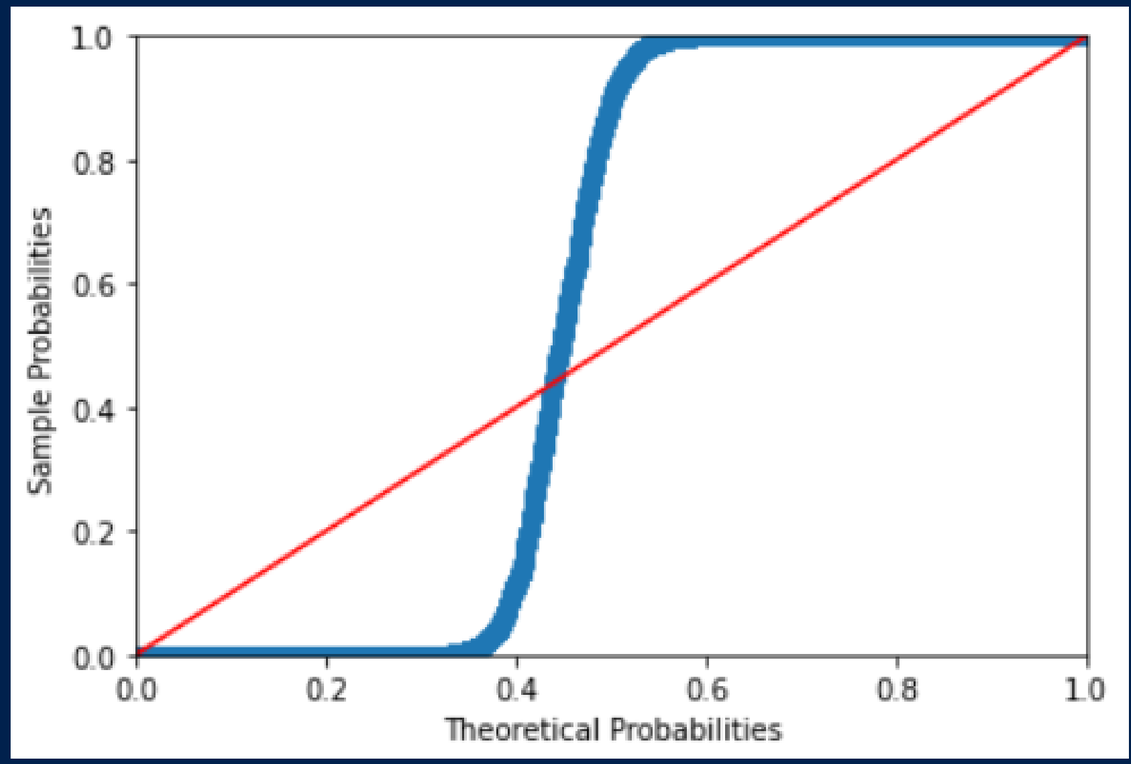
The model is:

SOLD PRICE = DIST_MAINROAD * 0.039914 + QS_OVERALL * 14.723315

Residual Analysis after applying Quadruple root

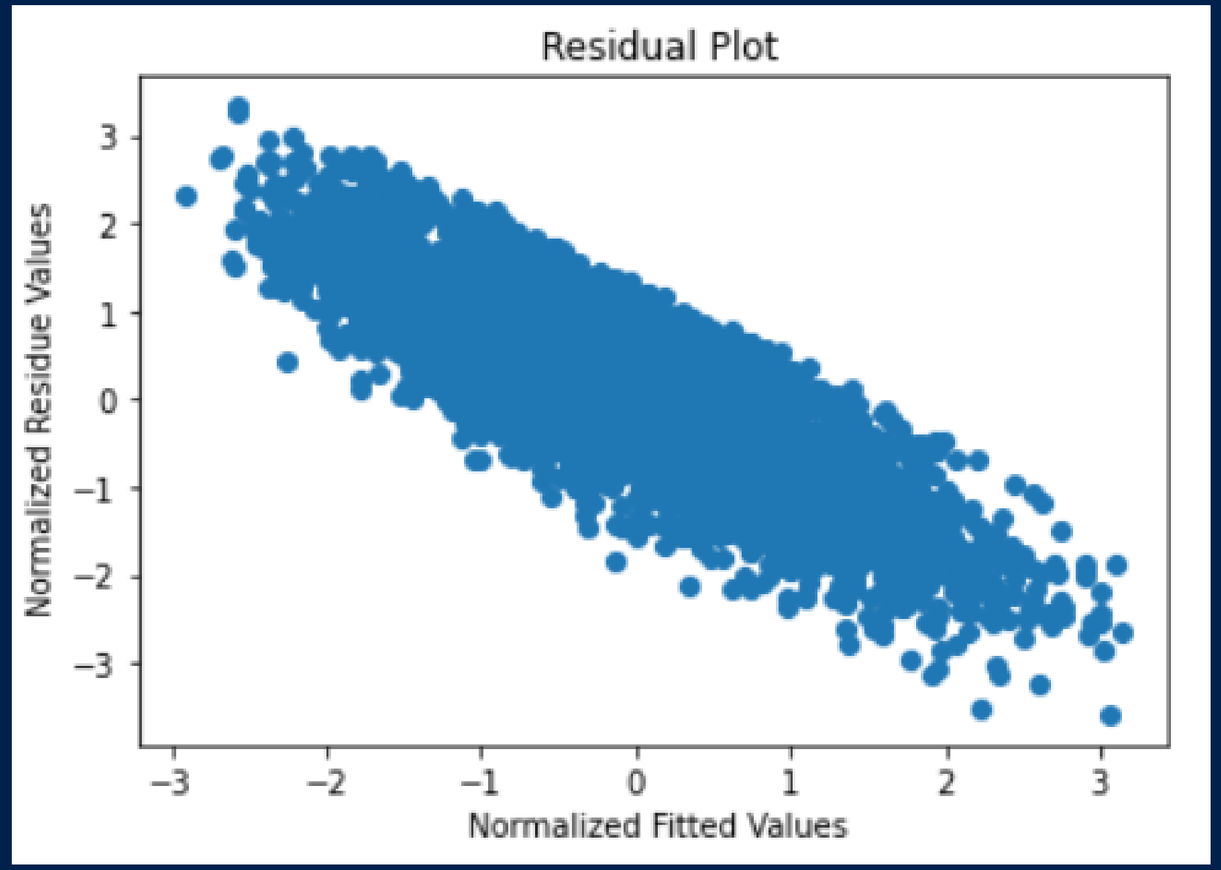
- Checking the Normality and plotting dist plot and prob plot





- Nearly perfect dist plot, but still not perfectly following normal distribution.
- Values coming closer to the line in prob plot, but still not acceptable.

Checking Homoscedasticity and Outliers - Quadruple root



- Parallel lines narrowing down. Hence, the assumption of homoscedasticity is fulfilled.
- No outliers found using Z Score, Cooks Distance and Leverage distance.

Predicting the values and Performance of the model after using Quadruple root

```
y_pred= mlr_5.predict(X_test_5)
y_pred
```

```
461      60.432166
3358     61.688949
3751     57.568239
2386     50.026631
1125     52.581776
           ...
6010     48.609131
4903     53.886501
6806     45.495044
3832     69.878265
364      62.080209
Length: 1422, dtype: float64
```

```
from sklearn.metrics import r2_score,mean_squared_error
```

```
r2= r2_score(y_test_5, y_pred)
print('R2:', r2)

mse = mean_squared_error(y_test_5, y_pred)

rmse= np.sqrt(mse)

print(' RMSE:',rmse)
```

```
R2: -2.567719236959250 3
 RMSE: 9.414739520812054
```

- RMSE Value further reduced significantly after quadruple rooting the target variable. So, we need to transform the target variable further in the same direction to get ideal results

Transforming the target variable - 1/8th Root

- Applying 1/8th root:

```
y_8r= np.sqrt(np.sqrt(np.sqrt(y)))
```

- We again split the data into train and test , build the model, train the model and diagnose the model and carry out the whole process again till our model is ready

```
mlr_4.params
```

```
DIST_MAINROAD    0.005282
QS_OVERALL       1.951001
dtype: float64
```

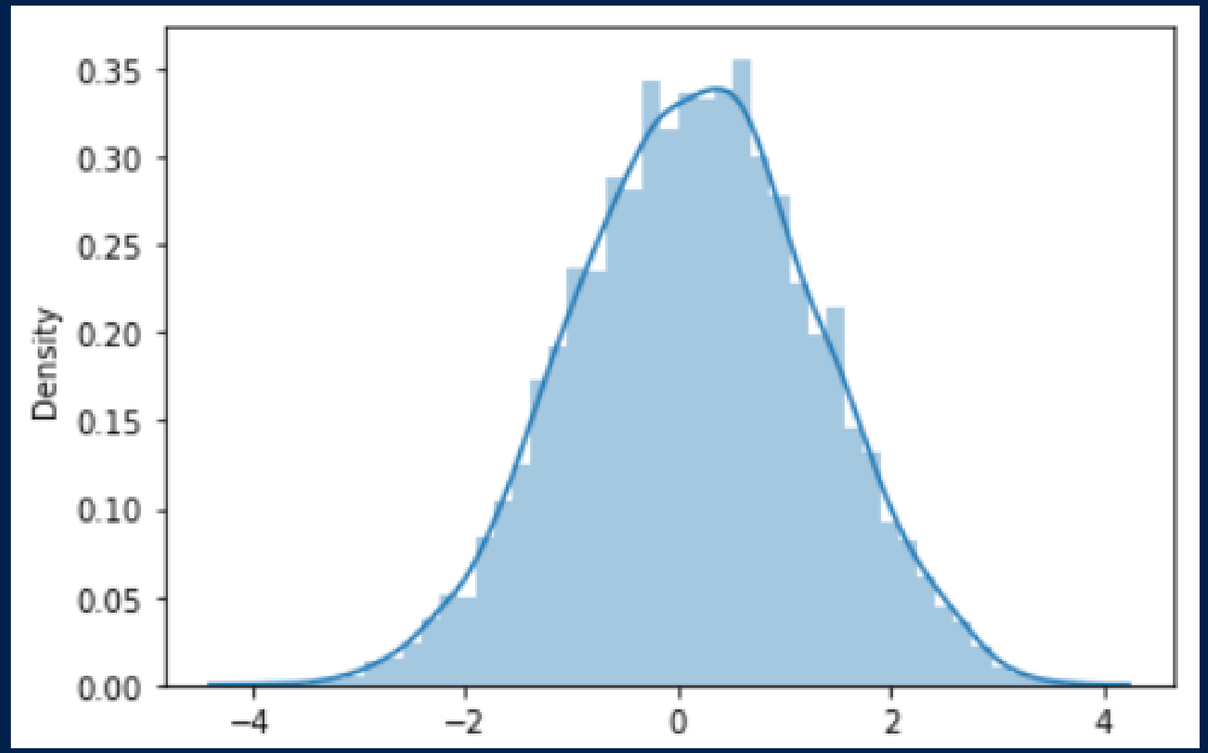
The model is:

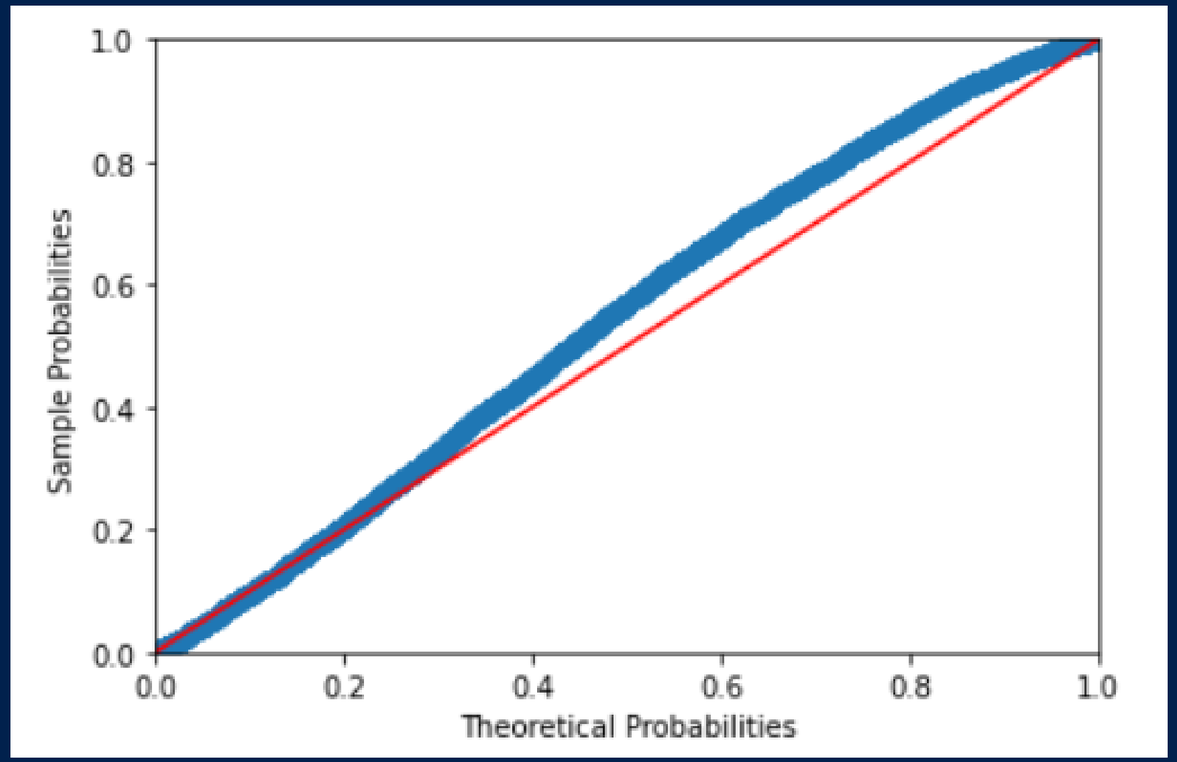
SOLD PRICE = DIST_MAINROAD * 0.005282 +QS_OVERALL * 1.951001

Final Model!!

Residual Analysis after applying 1/8th root

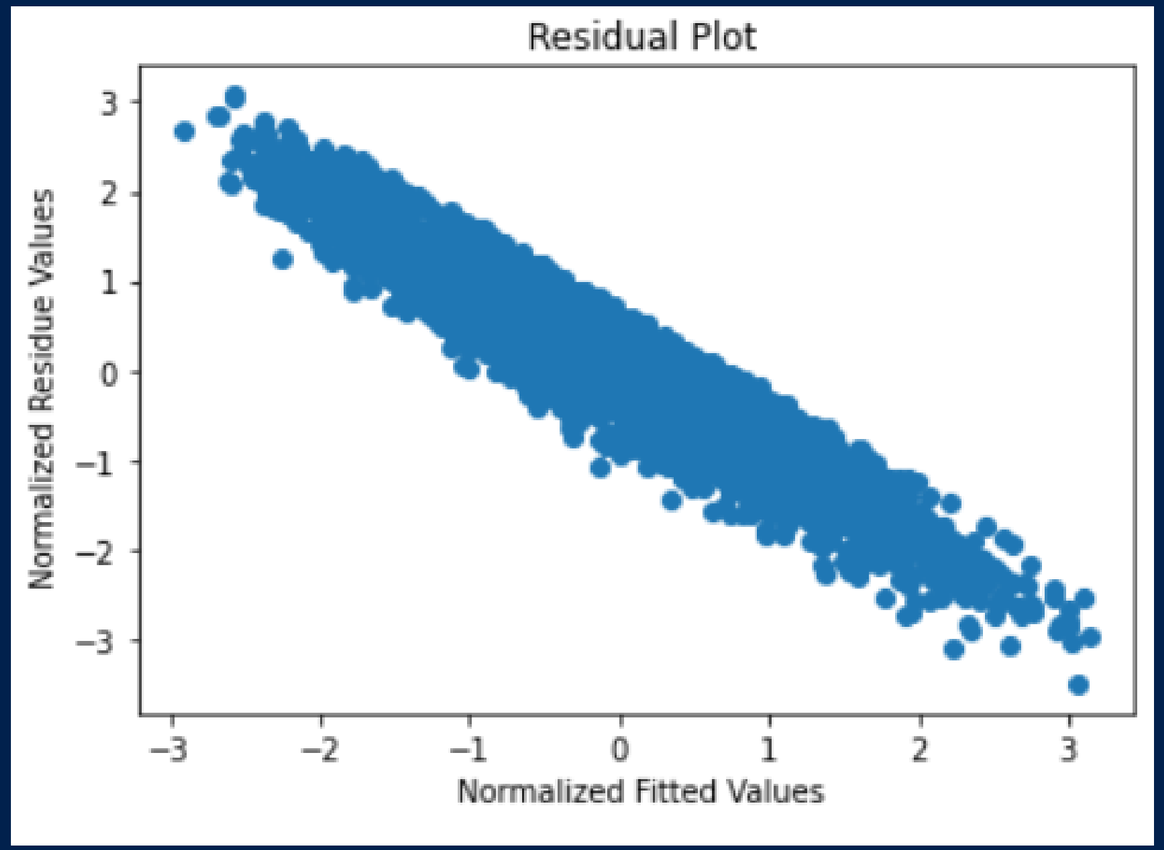
- Checking the Normality and plotting dist plot and prob plot





- Following almost normal distribution.
- Values are almost plotted on the line in prob plot, seems ideal.

Checking Homoscedasticity and Outliers - 1/8th root



- Clear and narrowed parallel lines can be observed. Hence, the assumption of homoscedasticity is fulfilled.
- No outliers found using Z Score, Cooks Distance and Leverage distance.

Predicting the values and Performance of the model after using 1/8th root

```
y_pred= mlr_5.predict(X_test_5)
y_pred
```

```
461      8.004560
1798     5.239330
3655     6.756989
5207     6.504615
1125     6.964616
           ...
5334     7.954894
4302     6.514955
3831     6.901881
364      8.223470
591      8.673455
Length: 1422, dtype: float64
```

```
from sklearn.metrics import r2_score,mean_squared_error
```

```
r2= r2_score(y_test_5, y_pred)
print('R2:', r2)

mse = mean_squared_error(y_test_5, y_pred)

rmse= np.sqrt(mse)

print(' RMSE:',rmse)
```

```
R2: -38.97654142206644
 RMSE: 0.383052616354967 5
```

- Finally got the least value of RMSE after 1/8th rooting of the target variable. Thus, we have the ideal results

Group 4
thanks you!!