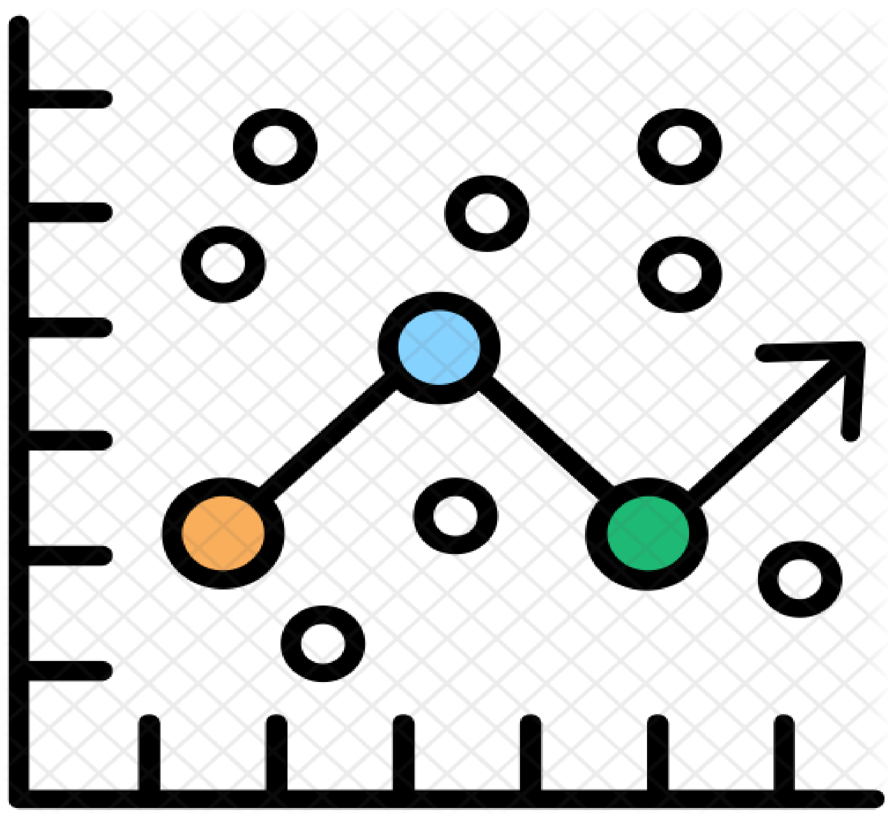
Pattern Recognition

Regression Module

Multiple Linear Regression & Logistic Regression



Kartik Aneja (SBU ID:112817127)

Nitin Asthana (SBU ID:112995559)

Introduction

The goal of the Regression Module is to understand and implement some of the machine learning tasks using Regression. We explored Simple Linear Regression, Multiple Linear Regression and implemented them on chosen datasets from kaggle. We chose the Boston Housing dataset for understanding how to build the regression model and implemented tasks such as prediction, feature reduction and evaluation of residual plots. We implemented the learned techniques like residual plots evaluation, model fitting, p-test on the NYC Airbnb dataset. For the creative component, we carried out classification using logistic regression on the Breast Cancer Dataset.

Datasets

- Boston Housing Data

  Link: https://www.kaggle.com/altavish/boston-housing-dataset

  13 Features

- New York City Airbnb Open Dataset

  Link: https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

  16 Features

- Breast Cancer Dataset

  Link: https://www.kaggle.com/merishnasuwal/breast-cancer-prediction-dataset

  5 Features

Implementation and Results

Boston Housing Dataset

Data Loading and Pre-Processing

There are 506 samples and 13 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features. We load the dataset and print some of it to check if we have the right type of data.

```
[2] # load datset and check if the dataset has right type of data
    df = pd.read_csv('BostonHousing.csv')
    df.head()
```

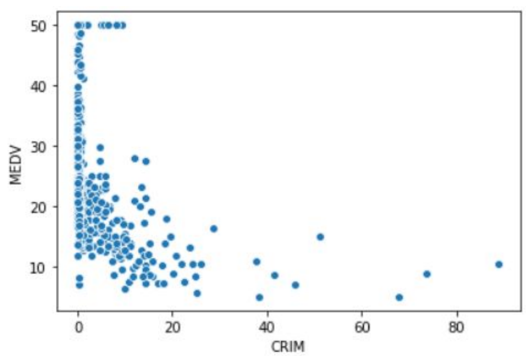
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

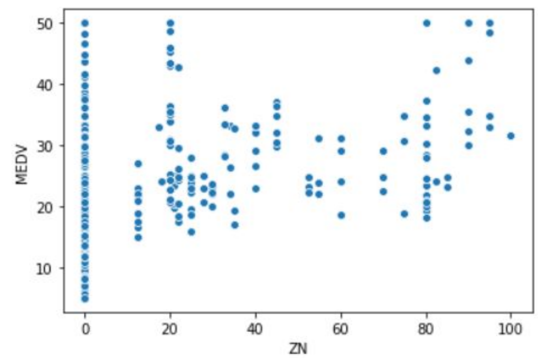
Next, we check for any null values in our dataset.

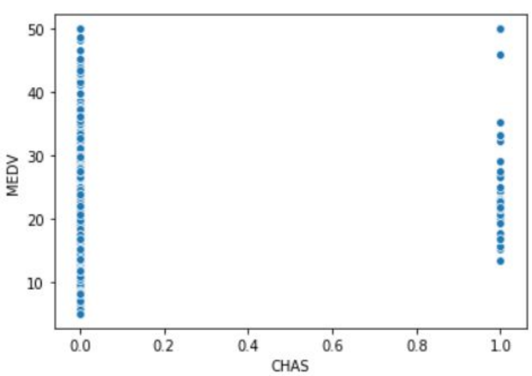
```
[6] print(df.isna().sum())

CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

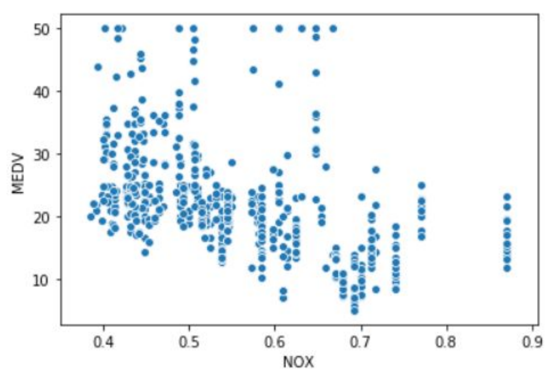
We plotted the scatter plots of all the features with our target variable which is the Median Price of the Houses.

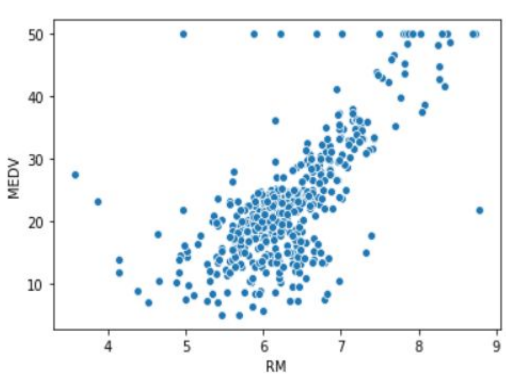
Scatter Plots of each feature with Target Variable:

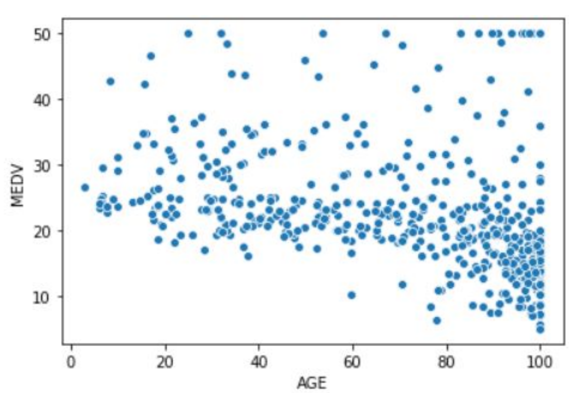


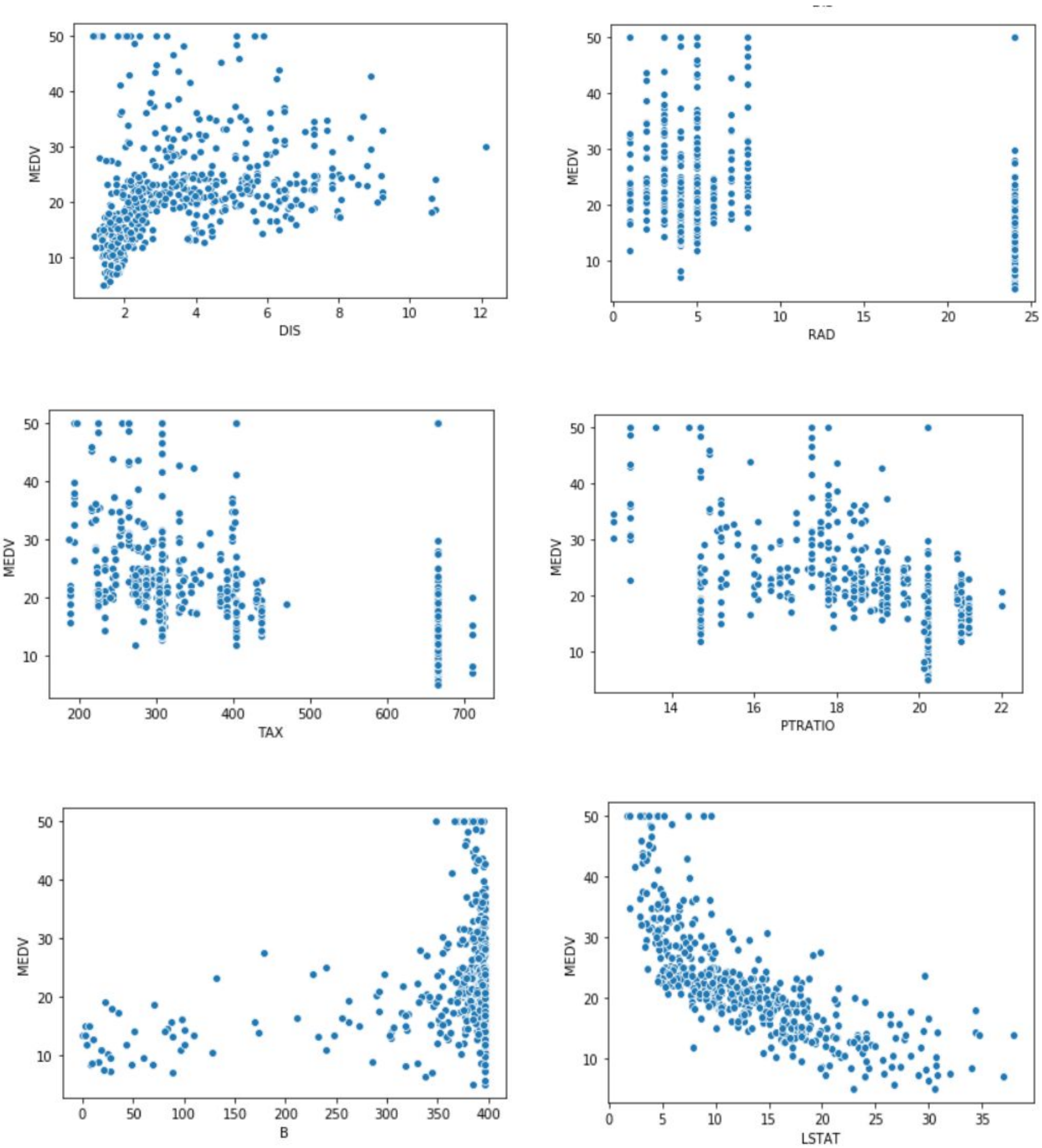












From the above plots, we observed some of the features are discrete like CHAS, RAD, ZN. RM shows a positive linear relationship with the target variable. We can see a few outliers also in RM. Feature LSTAT shows a negative linear (not exactly) relationship with the target variable.

Linear Regression

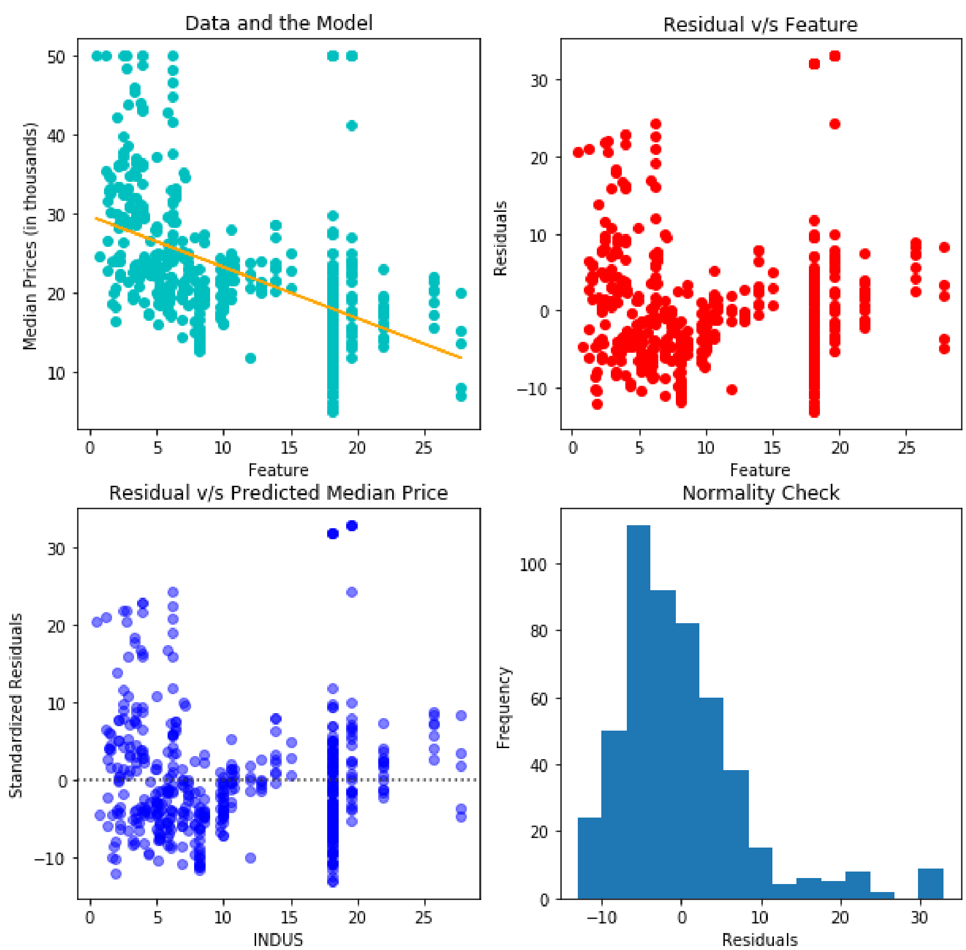
We fitted a regression line using the regression model on every feature and plotted the residuals. We simply looped over every feature and fitted the model on every feature and found the residuals. Below is the code snippet for that.

```
features = [1,2,3,4,5,6,7,8,9,10,11,12,13]
for i in features :
  X_feature = df.iloc[:,[i]]
  x_feature = df.iloc[:,[i]]
  np.reshape(X_feature,(-1,1))
  regress = LinearRegression()
  regress.fit(X_feature, Y)
  y_predicted = regress.predict(X_feature)
  res = Y - y_predicted
  fig,ax =  plt.subplots(2,2,figsize=(10,10))
  # print("Feature ",i)
  fig.suptitle(["Feature",i])
  ax[0][0].scatter(x_feature,Y,marker='o',c='c')
  ax[0][0].plot(X_feature,y_predicted,color='orange')
  ax[0][0].set(xlabel='Feature')
  ax[0][0].set(ylabel='Median Prices (in thousands)')
  ax[0][0].set_title('Data and the Model')
  # plt.show()
  ax[0][1].scatter(x_feature,res,marker='o',c='r')
  ax[0][1].set(xlabel='Feature')
  ax[0][1].set(ylabel='Residuals')
  ax[0][1].set_title('Residual v/s Feature')
  ax[1][0].scatter(y_predicted,res,marker='o',c='b')
  ax[1][0].set(xlabel='Predicted Prices')
  ax[1][0].set(ylabel='Residuals')
  ax[1][0].set_title('Residual v/s Predicted Median Price')
  # plt.show()
  ax[1][1].hist(res, bins=15)
  ax[1][1].set(xlabel='Residuals')
  ax[1][1].set(ylabel='Frequency')
  ax[1][1].set_title('Normality Check')
  plt.show()
```

Plots for some of the features:

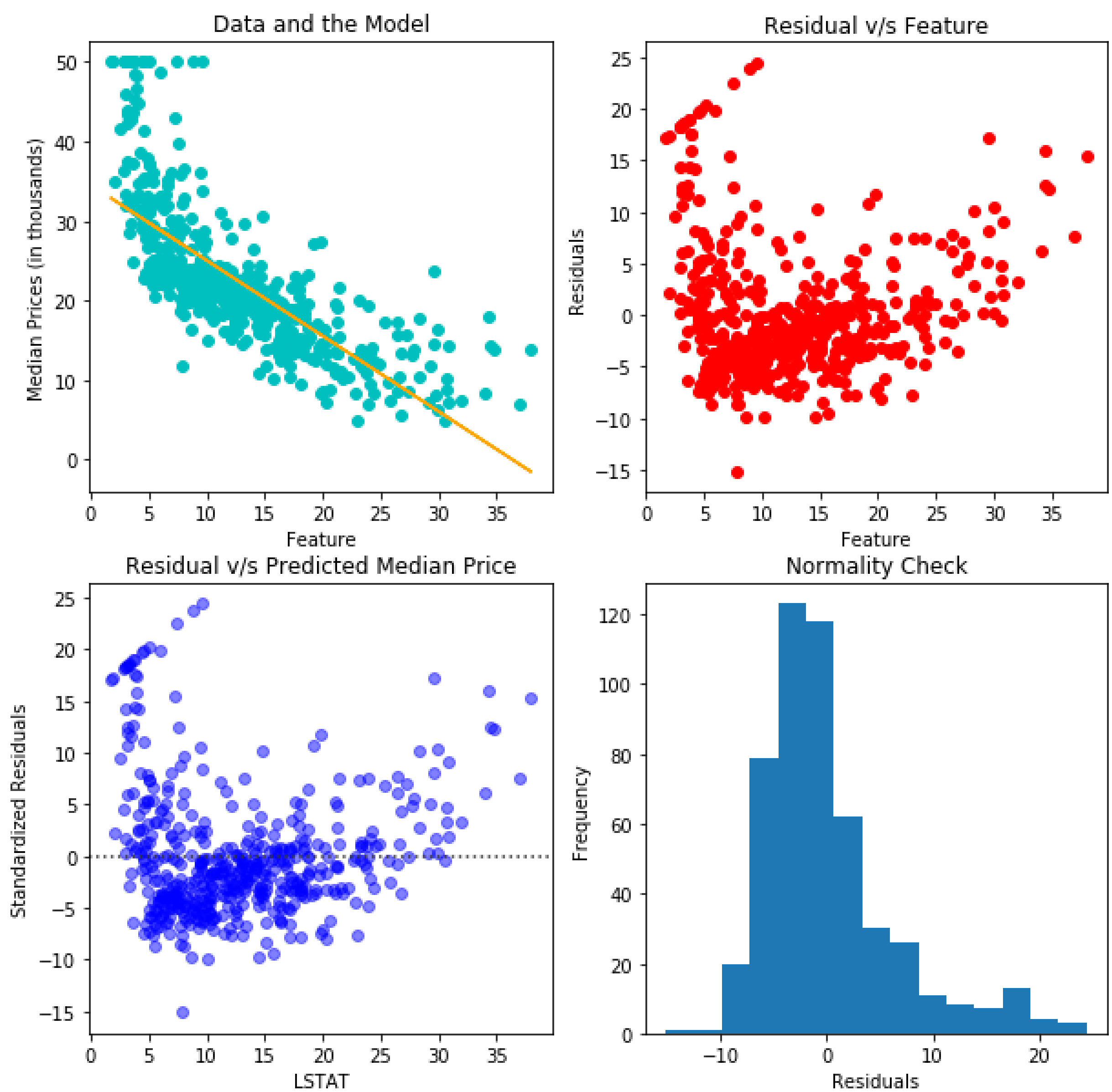
Feature ZN: There is not much spreading of the residuals. The residual plot seems to have a nearly constant variance. The histogram of the residuals shows the residuals are normally distributed with some outliers. There is one cluster present which shows the data is nearly independent. The residuals are Zero Averaged as zero balances the data.

['Feature', 2]

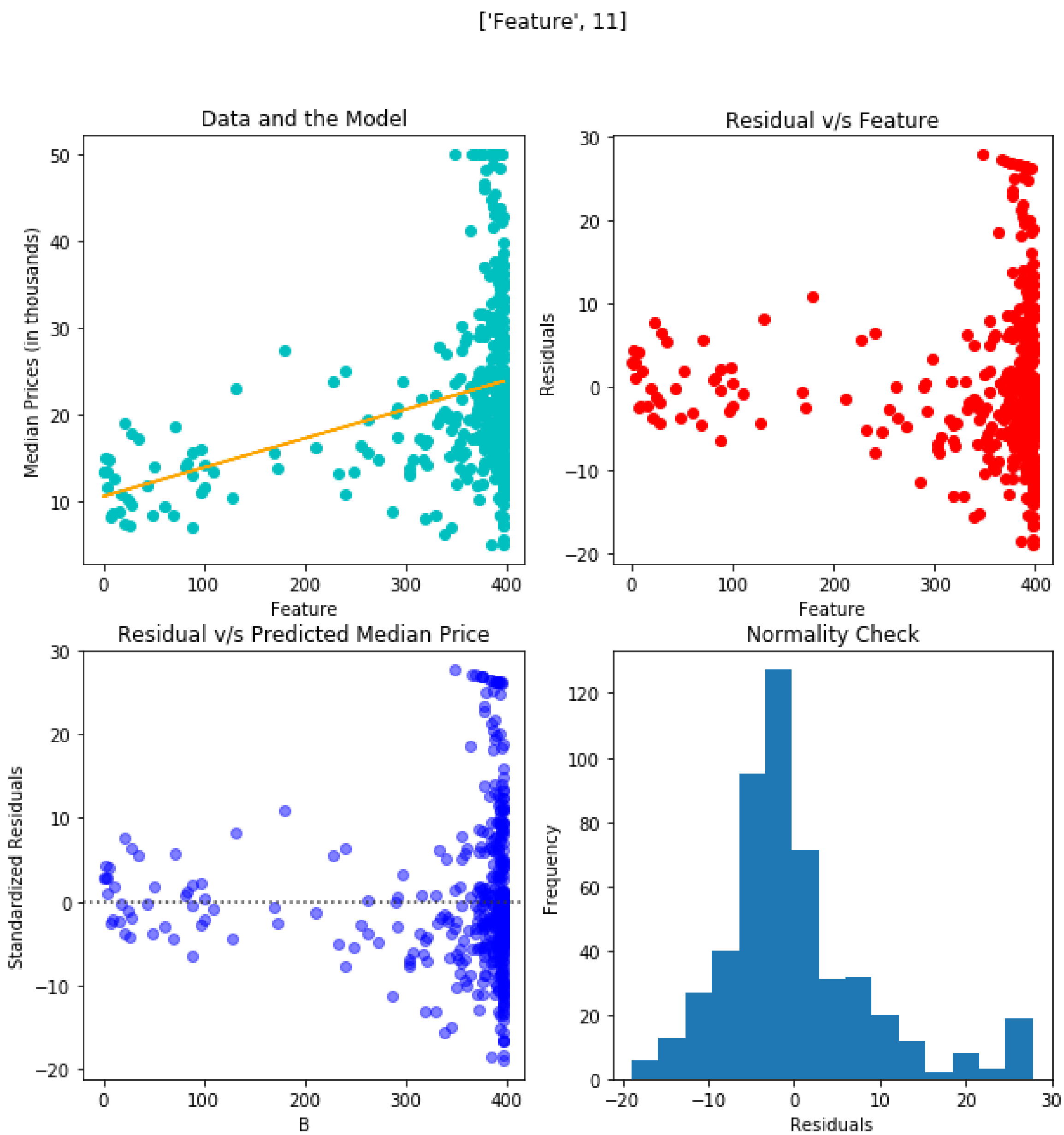


Feature LSTAT: The model looks pretty accurate. If we look closely there is a bit of pattern that the points are on the curve in model. This shows non-linearity (also if you look at residual plot) .The residuals show a bow-shaped pattern that violates the linearity condition.To fix the linearity violation we can add an non-linear term in our regression model. The histogram shows they are normally distributed. Also, the residuals are zero averaged.

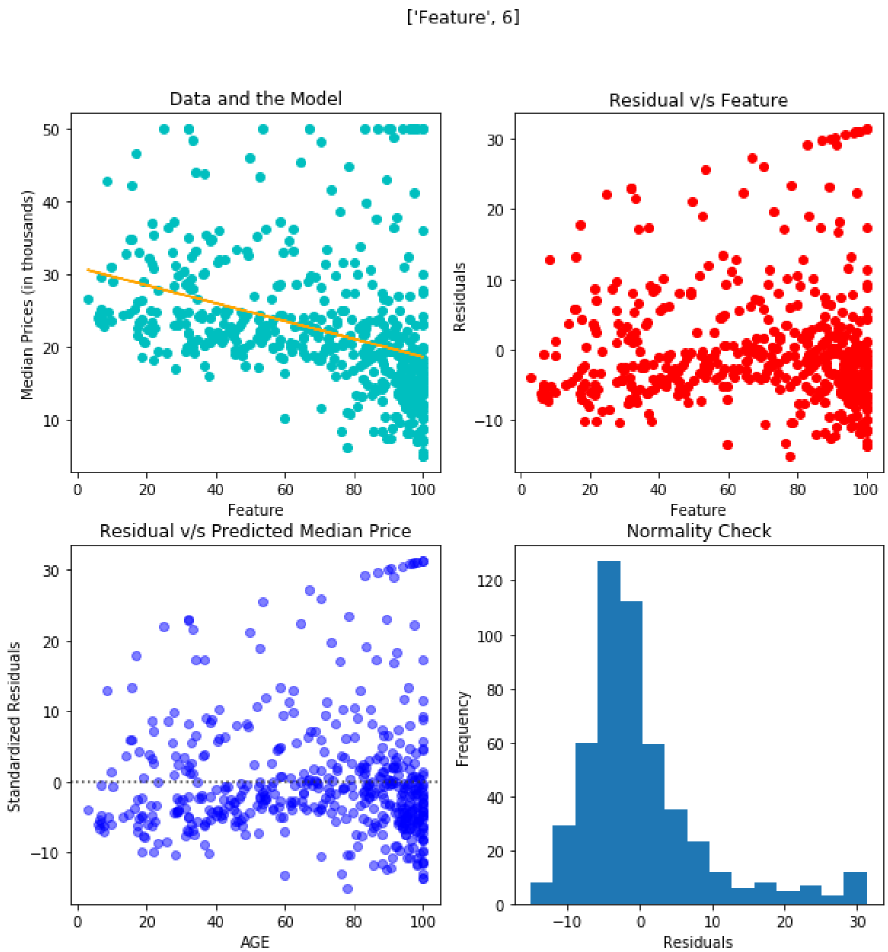
['Feature', 12]



Feature B: The residuals are wedge-shaped which implies there is a violation of constant variance.The residuals get larger as the prediction moves from small to large. This will not affect the model as such but will account for error in p-values of the feature. This can be solved by either transforming the variable (like log transformation). The residuals are zero-averaged as zero balances them. The Histogram shows they are normally distributed with some outliers.



Feature RM: The residuals show a constant variance. The histogram shows they are normally distributed but have some outlier. They're pretty symmetrically distributed, and there isn't any clear pattern.



Multiple Linear Regression

Implemented the multiple Linear regression and plotted the 3-D plots with two dependent and one independent variables. Split the data into training and testing.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
                                                    random_state=5)
```

Import the Linear Regression Model and fit it onto the training data.

```
Regression_Model = LinearRegression()
Regression_Model.fit(X_train, Y_train)
```

Created a mesh grid from the given features and fitted the model on 4 features. First on RM and CRIM, second on the RM and LSTAT.

```
x_surf,y_surf=np.meshgrid(np.linspace(X.RM.min(),X.RM.max(),506),np.linspace(X.LSTAT.min(),X.LSTAT.max(),506))
modX = pd.DataFrame({'RM': x_surf.ravel(),'LSTAT': y_surf.ravel()})
Y_Predict = Regression_Model.predict(modX)
Y_Predict= np.array(Y_Predict)
```

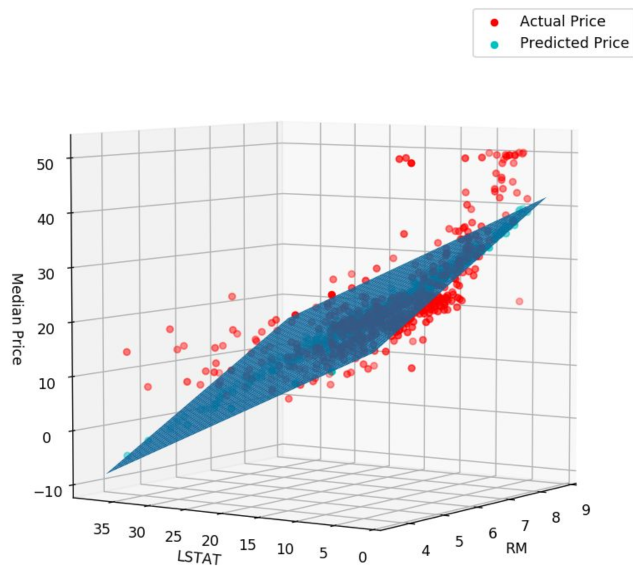
```
Y_predict = Regression_Model.predict(X)
```

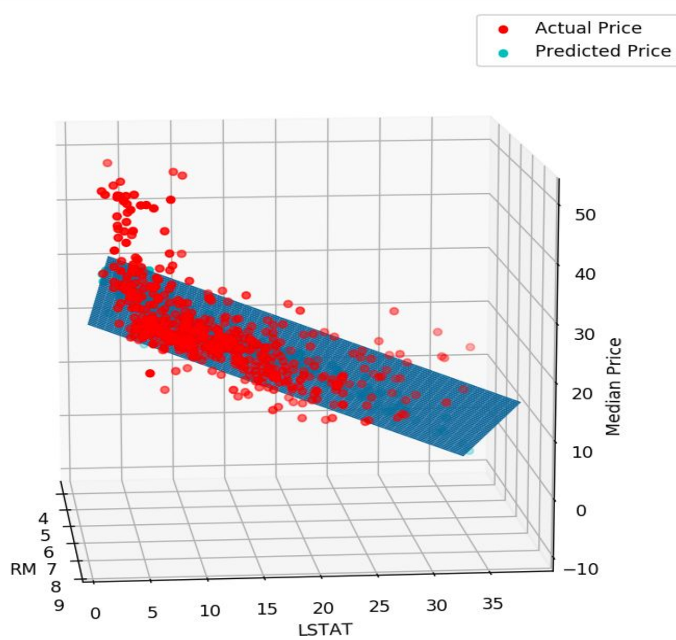
```
fig = plt.figure(figsize=(8,8))
ax=fig.add_subplot(111,projection='3d')
ax.scatter(X.RM,X.LSTAT,Y,c='r',label='Actual Price')
ax.scatter(X.RM,X.LSTAT,Y_predict,c='c',label='Predicted Price')
Axes3D.plot_surface(ax,x_surf,y_surf,Y_Predict.reshape(x_surf.shape))
ax.set_zlabel('Median Price')
ax.set_xlabel('RM')
ax.set_ylabel('LSTAT')
ax.legend()
plt.show()
```

Used the 3D plots to visualize the regression plane. The predicted price lies on the plane.

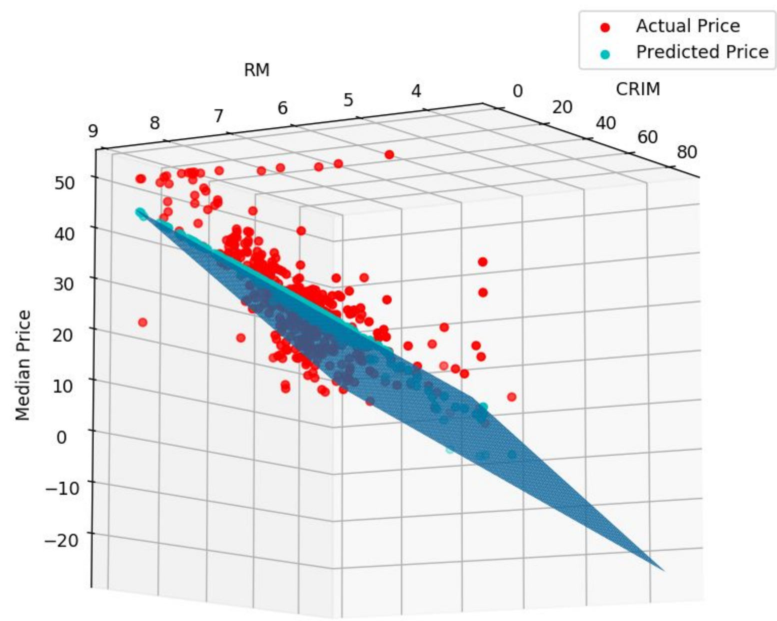
RM and LSTAT: The coefficient of the plane are found as below,

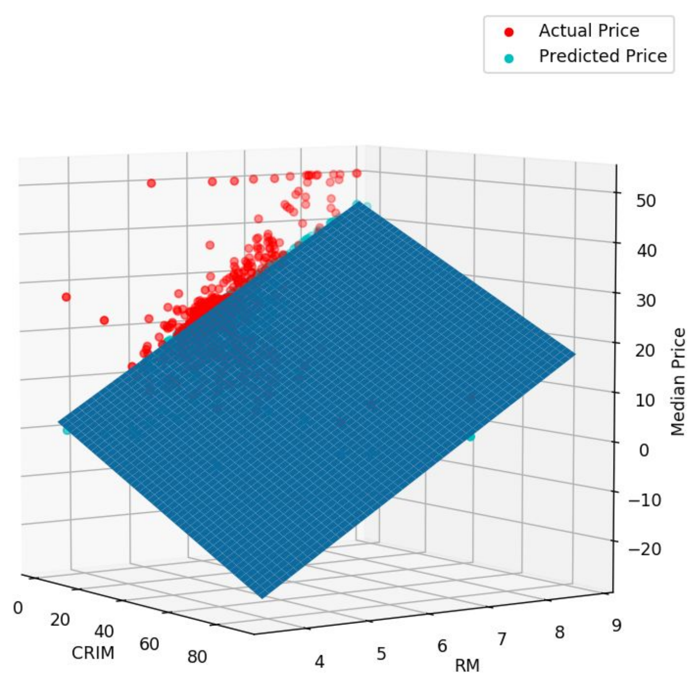
```
Regression_Model.coef_

array([ 4.58938833, -0.71722954])
```



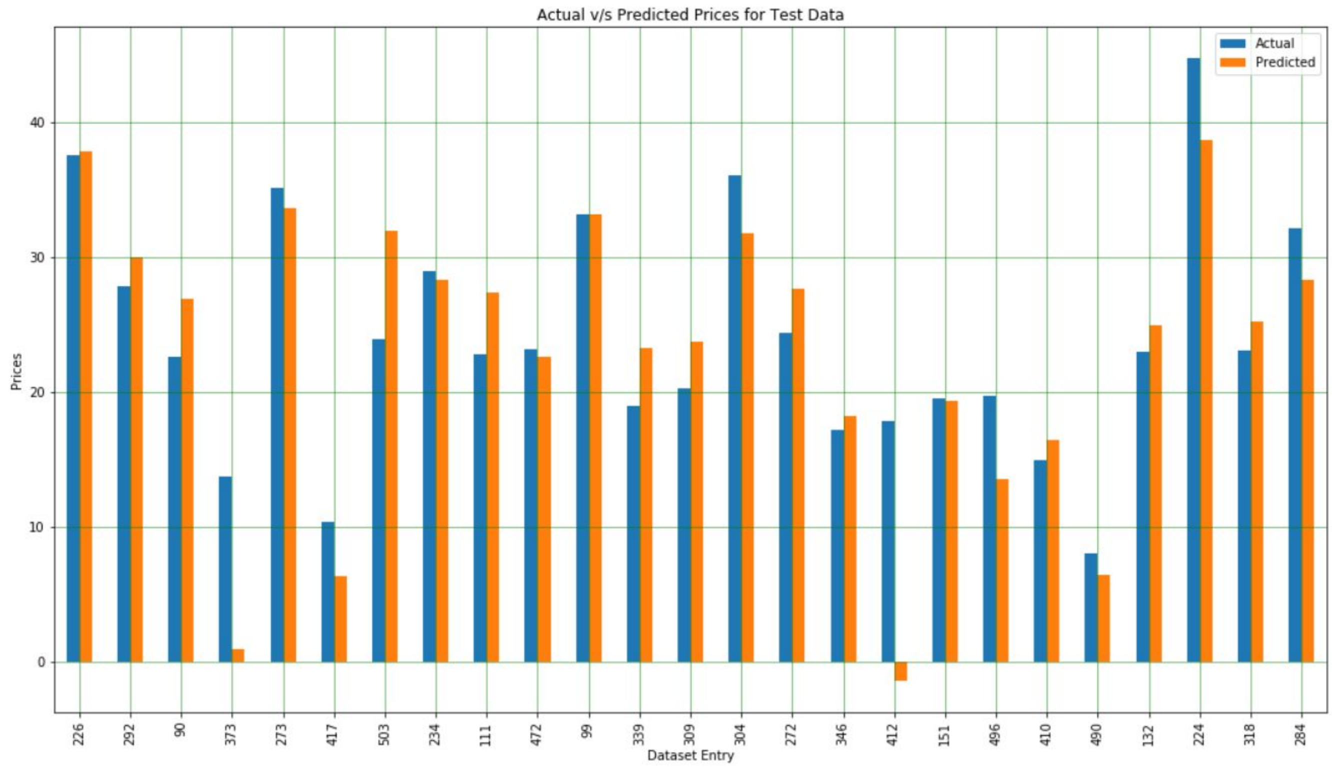


RM and CRIM v/s Median Price





Visualized the prediction of the model using barplots for some test data.



Evaluating the model and printing the summary of the statistics.

```
df['intercept']=1
lm=sm.OLS(df['MEDV'],df[['intercept','CRIM', 'ZN', 'INDUS','NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT']])
results = lm.fit()
results.summary()
```

Summary of the model:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.741
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	108.1
Date:	Fri, 06 Mar 2020	**Prob (F-statistic):**	6.72e-135
Time:	11:53:51	**Log-Likelihood:**	-1498.8
No. Observations:	506	**AIC:**	3026.
Df Residuals:	492	**BIC:**	3085.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
intercept	36.4595	5.103	7.144	0.000	26.432	46.487
CRIM	-0.1080	0.033	-3.287	0.001	-0.173	-0.043
ZN	0.0464	0.014	3.382	0.001	0.019	0.073
INDUS	0.0206	0.061	0.334	0.738	-0.100	0.141
CHAS	2.6867	0.862	3.118	0.002	0.994	4.380
NOX	-17.7666	3.820	-4.651	0.000	-25.272	-10.262
RM	3.8099	0.418	9.116	0.000	2.989	4.631
AGE	0.0007	0.013	0.052	0.958	-0.025	0.027
DIS	-1.4756	0.199	-7.398	0.000	-1.867	-1.084
RAD	0.3060	0.066	4.613	0.000	0.176	0.436
TAX	-0.0123	0.004	-3.280	0.001	-0.020	-0.005
PTRATIO	-0.9527	0.131	-7.283	0.000	-1.210	-0.696
B	0.0093	0.003	3.467	0.001	0.004	0.015
LSTAT	-0.5248	0.051	-10.347	0.000	-0.624	-0.425

Omnibus:	178.041	**Durbin-Watson:**	1.078
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	783.126
Skew:	1.521	**Prob(JB):**	8.84e-171
Kurtosis:	8.281	**Cond. No.**	1.51e+04

Summary Table Interpretation:

P>|t| (p-test): It is a two-tailed hypothesis test where the null hypothesis is that the feature has no effect on MEDV (Target variable). If the p-value of a feature is so low it is approximately zero, then there is strong statistical evidence to reject the claim that feature has no effect on MEDV.

We calculated the RMSE value of our regression model as follows:

```
print("RMSE of the model: ",np.sqrt(((Y_predict-Y_test) ** 2).mean()))
```

```
RMSE of the model:  4.568292042303191
```

Feature Reduction using p-values and Co-linearity

If we look at the features which have p-values different than zero like INDUS which has p-value 0.738, we can remove this feature as p-value suggests it has no effect on our target variable.

Summary of the model after removing INDUS:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.741
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	117.3
Date:	Fri, 06 Mar 2020	**Prob (F-statistic):**	6.42e-136
Time:	12:41:35	**Log-Likelihood:**	-1498.9
No. Observations:	506	**AIC:**	3024.
Df Residuals:	493	**BIC:**	3079.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
intercept	36.3639	5.091	7.143	0.000	26.361	46.366
CRIM	-0.1084	0.033	-3.304	0.001	-0.173	-0.044
ZN	0.0459	0.014	3.368	0.001	0.019	0.073
CHAS	2.7164	0.856	3.173	0.002	1.034	4.399
NOX	-17.4295	3.681	-4.735	0.000	-24.662	-10.197
RM	3.7970	0.416	9.132	0.000	2.980	4.614
AGE	0.0007	0.013	0.053	0.958	-0.025	0.027
DIS	-1.4896	0.195	-7.648	0.000	-1.872	-1.107
RAD	0.2999	0.064	4.710	0.000	0.175	0.425
TAX	-0.0118	0.003	-3.489	0.001	-0.018	-0.005
PTRATIO	-0.9471	0.130	-7.308	0.000	-1.202	-0.692
B	0.0093	0.003	3.461	0.001	0.004	0.015
LSTAT	-0.5235	0.051	-10.361	0.000	-0.623	-0.424

Omnibus:	178.124	**Durbin-Watson:**	1.079
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	784.481
Skew:	1.521	**Prob(JB):**	4.49e-171

As you can see after removing the feature INDUS from the data, the R-squared values remained constant. Also, we calculated the RMSE value to check the fit of our model. RMSE value didn't change so the feature reduction of INDUS has no effect on our model and prediction.

```
newY_predict = regression_model.predict(newX_test)
print("New RMSE after removing INDUS: ",np.sqrt(((newY_predict-newY_test) ** 2).mean()))
```

```
New RMSE after removing INDUS:  4.568585265156598
```

Now same as above we can remove the AGE feature as its p-value is near to 1. After removing the feature AGE we looked at the table again and the value of R-squared remained the same that is 0.741.

Summary table after removing AGE :

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.741
Model:	OLS	**Adj. R-squared:**	0.735
Method:	Least Squares	**F-statistic:**	128.2
Date:	Fri, 06 Mar 2020	**Prob (F-statistic):**	5.54e-137
Time:	12:55:06	**Log-Likelihood:**	-1498.9
No. Observations:	506	**AIC:**	3022.
Df Residuals:	494	**BIC:**	3072.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
intercept	36.3411	5.067	7.171	0.000	26.385	46.298
CRIM	-0.1084	0.033	-3.307	0.001	-0.173	-0.044
ZN	0.0458	0.014	3.390	0.001	0.019	0.072
CHAS	2.7187	0.854	3.183	0.002	1.040	4.397
NOX	-17.3760	3.535	-4.915	0.000	-24.322	-10.430
RM	3.8016	0.406	9.356	0.000	3.003	4.600
DIS	-1.4927	0.186	-8.037	0.000	-1.858	-1.128
RAD	0.2996	0.063	4.726	0.000	0.175	0.424
TAX	-0.0118	0.003	-3.493	0.001	-0.018	-0.005
PTRATIO	-0.9465	0.129	-7.334	0.000	-1.200	-0.693
B	0.0093	0.003	3.475	0.001	0.004	0.015
LSTAT	-0.5226	0.047	-11.019	0.000	-0.616	-0.429

Omnibus:	178.430	**Durbin-Watson:**	1.078
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	787.785
Skew:	1.523	**Prob(JB):**	8.60e-172
Kurtosis:	8.300	**Cond. No.**	1.47e+04

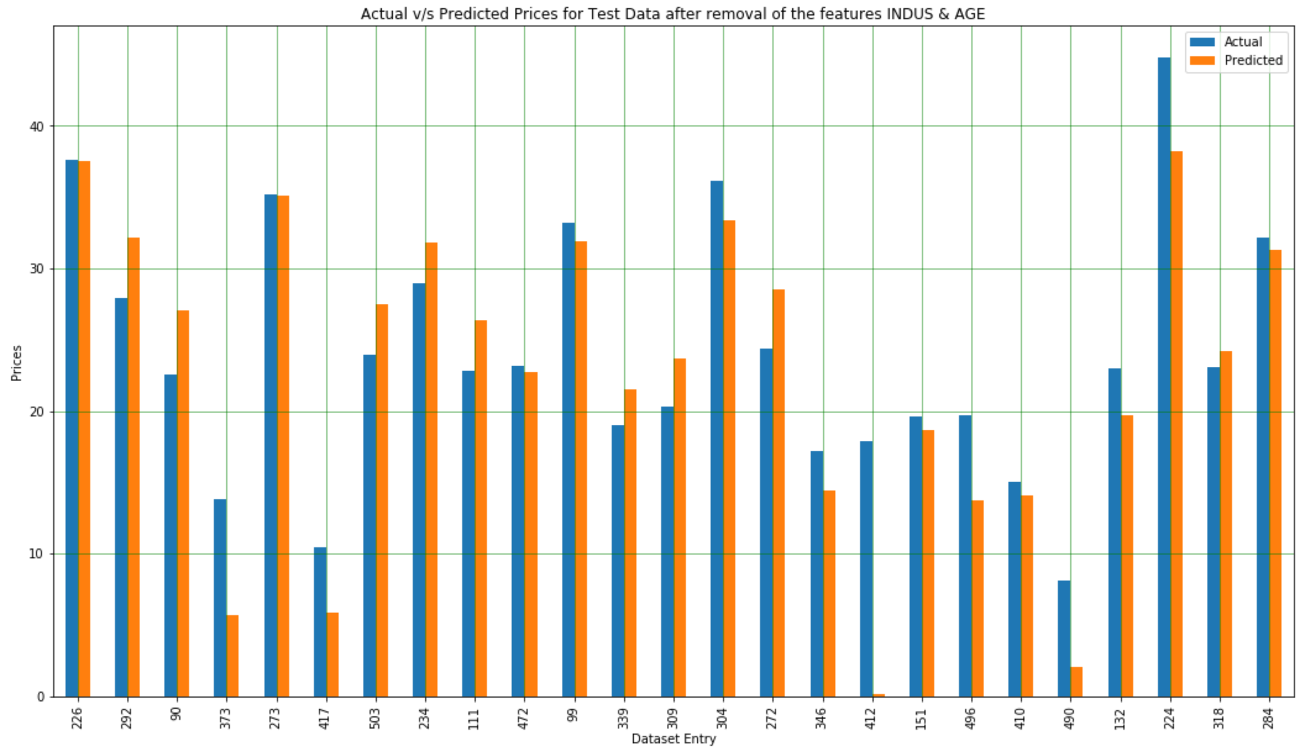
We calculated the RMSE of the model and it was found to be the same which implies the removal of the feature AGE didn’t alter the fitness of the model.

```
new_Y_predict = regression_model.predict(new_X_test)

print("New RMSE after removing AGES: ",np.sqrt(((new_Y_predict-new_Y_test) ** 2).mean()))
```

```
New RMSE after removing AGES:  4.568585265156598
```

Bar Plot of the actual v/s predicted prices after removal of INDUS and AGE:

The prediction looks almost the same after removing the two features.

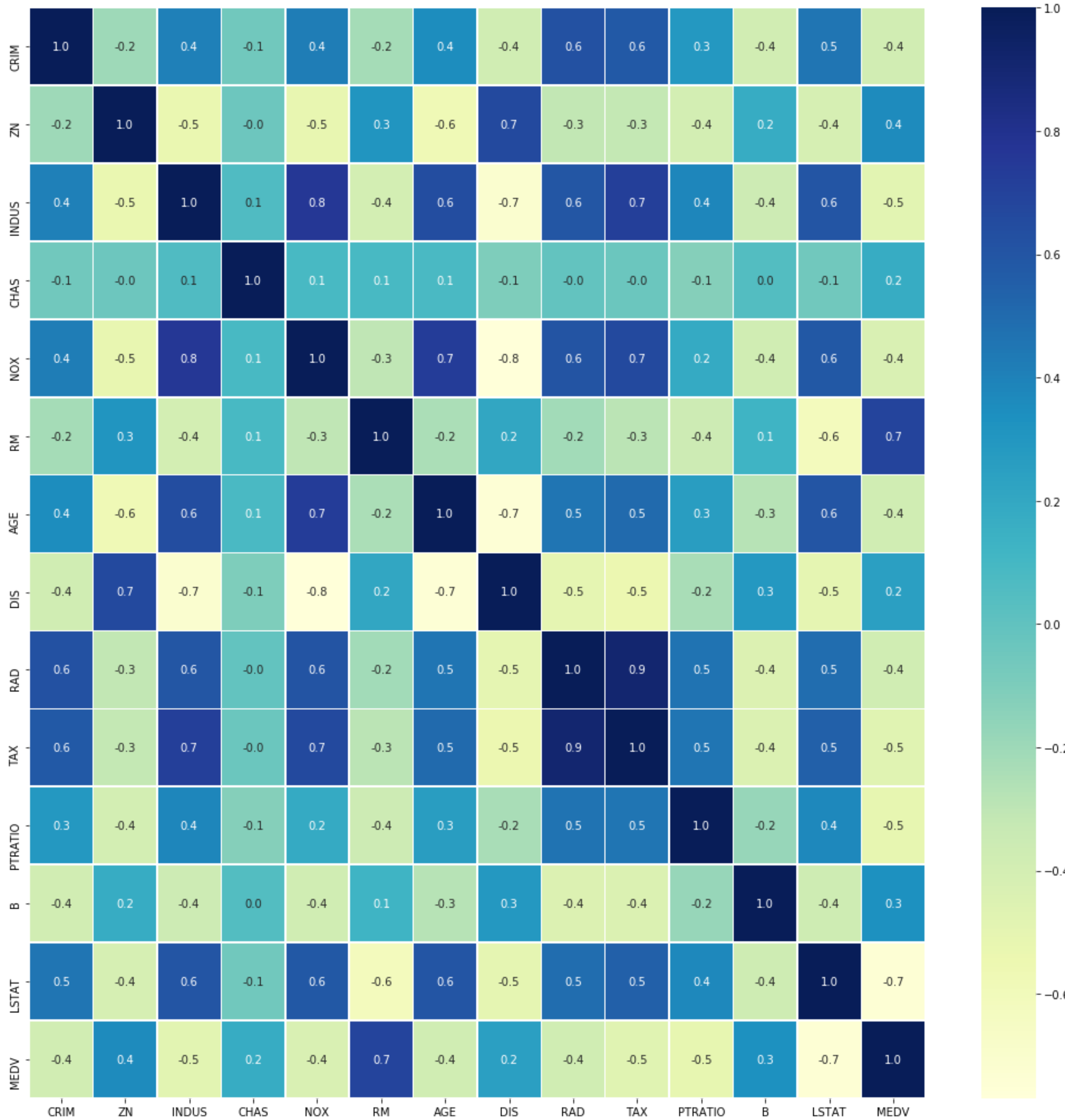


Apart from the Hypothesis testing of p-values, we can look at the co-linearity of the features and perform feature reduction through it. The correlation between the features can be best viewed through the heatmap. Below is the code snippet to plot the heatmap.

```
# Check for co-linearity among features and with preidiction feature
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(df.corr(),annot=True, linewidths=.5, fmt= '.1f',ax=ax,cmap="YlGnBu")
```

In HeatMap each square shows the correlation between the features on each axis. Correlation ranges from -1 to +1. Values closer to zero means there is no linear trend between the two variables. The closer to 1 the correlation is the more positively correlated they are; that is as one increases so does the other and the closer to 1 the stronger this relationship is. A correlation closer to -1 is similar, but instead of both increasing one variable will decrease as the other increases.

HeatMap for the Boston Housing Dataset



By looking at the heatmap plot we can infer that the TAX and RAD both are strongly correlated. So out of these two, we can drop one feature as it will not majorly affect our prediction task.

We calculated the RMSE for our model after removing the RAD feature. RMSE reduced by 0.04 which implies our model got improved a little. The reason RAD was dropped is due to its correlation with our dependent variable is low compared to the TAX feature.

```
nY_predict = regression_model.predict(nX_test)

print("New RMSE after removing RAD: ",np.sqrt(((nY_predict-nY_test) ** 2).mean()))
```

```
New RMSE after removing RAD:  4.502033819232363
```

After feature reduction, we have reduced our features from 13 to 10. Further feature reduction can be done using Principal Component Analysis.

New York City Airbnb DataSet

We predicted the price of the Airbnb in NYC using Multiple Linear Regression.

Data Loading and Pre-Processing

The dataset contains a total of 16 features and 48895 entries. The following are the features of the data. Price is our dependent variable.

```
df.columns
```

```
Index(['id', 'name', 'host_id', 'host_name', 'neighbourhood_group',
       'neighbourhood', 'latitude', 'longitude', 'room_type', 'price',
       'minimum_nights', 'number_of_reviews', 'last_review',
       'reviews_per_month', 'calculated_host_listings_count',
       'availability_365', 'intercept'],
      dtype='object')
```

Checked for null values in data. Some columns contain null values. We have dealt with them as follows.

```
df.isna().sum()
```

```
id                                    0
name                                 16
host_id                               0
host_name                            21
neighbourhood_group                   0
neighbourhood                         0
latitude                              0
longitude                             0
room_type                             0
price                                 0
minimum_nights                        0
number_of_reviews                     0
last_review                       10052
reviews_per_month                 10052
calculated_host_listings_count        0
availability_365                      0
dtype: int64
```

Features like id, name, host_id, host_name, last_review don't give any information about price, so we decided to drop them.

Reviews_per_month contains 10052 null values. Since reviews seem like an important factor that plays a role in price in this era so we decided to fill these null values with the mean value of the reviews_per_month column.

Handling Categorical Co-Variates

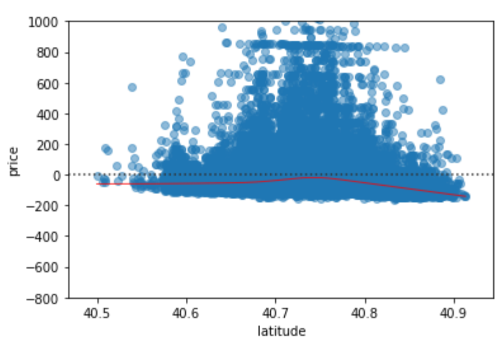
We found three categorical features: neighbourhood_group and room_type.

We handled them using one-hot encoding. It is a way of representing categorical values as binary vectors. It makes the categorical feature more expressive. After one hot encoding, our features increased to 16.

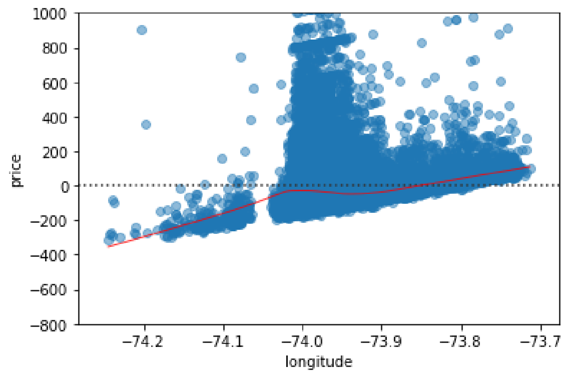
```
encoded_df.info()
#categoriaacal features are handled using one hot encoding

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 17 columns):
latitude                             48895 non-null float64
longitude                            48895 non-null float64
price                                48895 non-null int64
minimum_nights                       48895 non-null int64
number_of_reviews                    48895 non-null int64
reviews_per_month                    38843 non-null float64
calculated_host_listings_count       48895 non-null int64
availability_365                     48895 non-null int64
price_log                            48895 non-null float64
neighbourhood_group_Bronx            48895 non-null uint8
neighbourhood_group_Brooklyn         48895 non-null uint8
neighbourhood_group_Manhattan        48895 non-null uint8
neighbourhood_group_Queens           48895 non-null uint8
neighbourhood_group_Staten Island    48895 non-null uint8
room_type_Entire home/apt            48895 non-null uint8
room_type_Private room               48895 non-null uint8
room_type_Shared room                48895 non-null uint8
dtypes: float64(4), int64(5), uint8(8)
memory usage: 3.7 MB
```

Residuals Plots

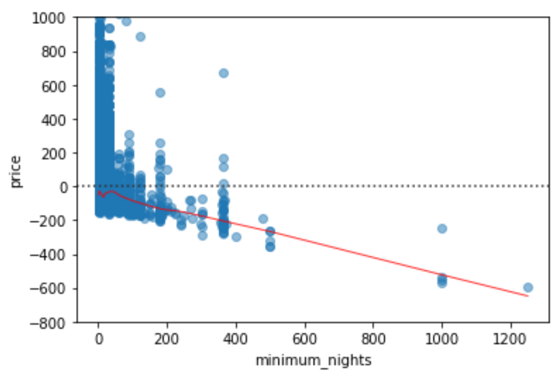
Latitude: Covariance of this feature's residuals is first increasing then decreasing, which implies the problem of Heteroscedasticity. Residuals get larger as the prediction moves from small to large (or from large to small). This will affect the p-value of the feature.



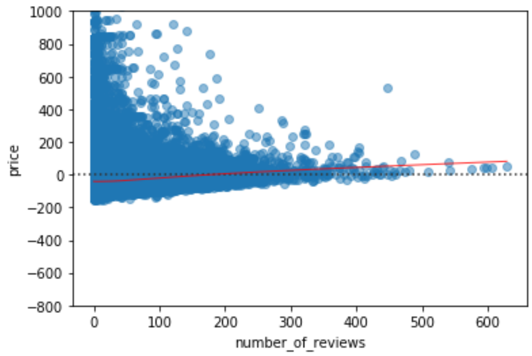
Longitude: The residuals have a bow-shaped pattern which accounts for non-linearity.The predictions would be way off, meaning the model doesn't accurately represent the relationship between longitude and Prices. We have to create a non-linear model to fix the linearity violation.

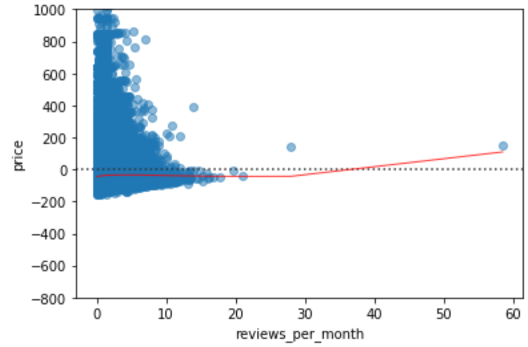


Minimum_nights: Few outliers in this feature. In the worst case, the model can pivot to try to get closer to that point at the expense of being close to all the others, and end up being just entirely wrong.

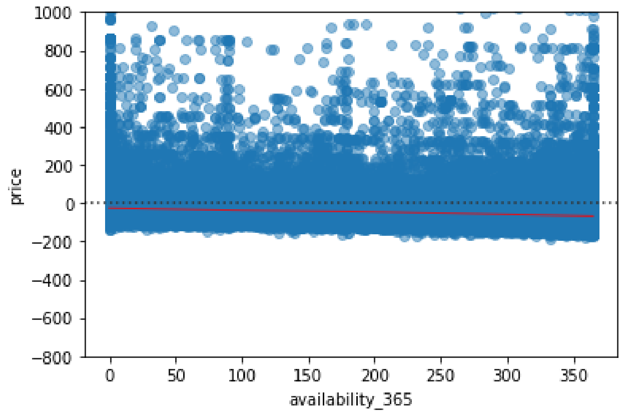


Number_of_reviews and Reviews_per_month: The wedge-shaped pattern implies variance is not constant. Residuals get larger as the prediction moves from small to large. It will affect the p-values of the feature.





Availablity_365: Variance of this feature has many ups and downs on very large values, which will affect p-values. Y-axis seems to be unbalanced.



Multiple Linear Regression

We started with 16 features and fitted the model. The parameters of the model came to be as follows. The R-squared of the model came to be very low that is 0.098, which indicates a poor model fit. Also, we calculated the RMSE value of the model, which came around 199.97, which is not at all good.

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.098
Model:	OLS	**Adj. R-squared:**	0.098
Method:	Least Squares	**F-statistic:**	410.5
Date:	Sat, 07 Mar 2020	**Prob (F-statistic):**	0.00
Time:	09:34:03	**Log-Likelihood:**	-3.3485e+05
No. Observations:	48895	**AIC:**	6.697e+05
Df Residuals:	48881	**BIC:**	6.699e+05
Df Model:	13		
Covariance Type:	nonrobust		

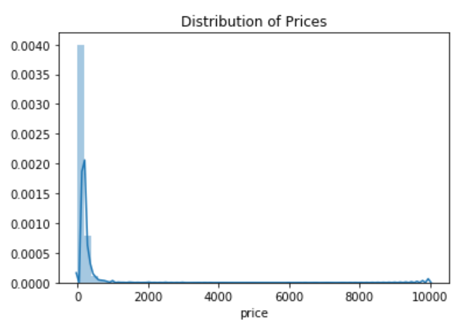
	coef	std err	t	P>|t|	[0.025	0.975]
latitude	-198.0261	31.400	-6.307	0.000	-259.571	-136.481
longitude	-516.6685	36.123	-14.303	0.000	-587.471	-445.866
minimum_nights	-0.0229	0.052	-0.444	0.657	-0.124	0.078
number_of_reviews	-0.3546	0.028	-12.789	0.000	-0.409	-0.300
reviews_per_month	2.8874	0.826	3.494	0.000	1.268	4.507
calculated_host_listings_count	-0.1771	0.033	-5.331	0.000	-0.242	-0.112
availability_365	0.1922	0.008	22.757	0.000	0.176	0.209
neighbourhood_group_Bronx	-1.124e+04	1206.842	-9.317	0.000	-1.36e+04	-8878.251
neighbourhood_group_Brooklyn	-1.128e+04	1205.912	-9.350	0.000	-1.36e+04	-8911.525
neighbourhood_group_Manhattan	-1.121e+04	1208.158	-9.282	0.000	-1.36e+04	-8846.510
neighbourhood_group_Queens	-1.125e+04	1204.249	-9.339	0.000	-1.36e+04	-8886.691
neighbourhood_group_Staten Island	-1.14e+04	1209.850	-9.420	0.000	-1.38e+04	-9025.735
room_type_Entire home/apt	-1.871e+04	2011.719	-9.300	0.000	-2.27e+04	-1.48e+04
room_type_Private room	-1.882e+04	2011.616	-9.354	0.000	-2.28e+04	-1.49e+04
room_type_Shared room	-1.885e+04	2011.599	-9.372	0.000	-2.28e+04	-1.49e+04

Omnibus:	110400.350	**Durbin-Watson:**	1.848

```
print("RMSE of the model: ",np.sqrt(((Y_predict-Y_test) ** 2).mean()))
```

```
RMSE of the model:  199.9774642639664
```

After a lot of tuning and feature reductions, the model didn't fit well. Then we decided to look at the distribution of the price (dependent variable). The distribution seems skewed towards the right, which means it is not normally distributed. As the data size is not so big this skewed dependent variable might be the reason for the bad fitting of the model.



We tried to make it normally distributed by using log transformation, and added a new column “modified_price”.

Fitted model after the price modification:

OLS Regression Results

Dep. Variable:	modified_price	**R-squared (uncentered):**	0.980
Model:	OLS	**Adj. R-squared (uncentered):**	0.980
Method:	Least Squares	**F-statistic:**	3.348e+05
Date:	Sat, 07 Mar 2020	**Prob (F-statistic):**	0.00
Time:	10:11:52	**Log-Likelihood:**	-50833.
No. Observations:	48895	**AIC:**	1.017e+05
Df Residuals:	48888	**BIC:**	1.017e+05
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
latitude	-0.6576	0.050	-13.164	0.000	-0.756	-0.560
longitude	-0.4257	0.028	-15.475	0.000	-0.480	-0.372
minimum_nights	4.531e-05	0.000	0.293	0.769	-0.000	0.000
number_of_reviews	-0.0006	8.3e-05	-6.752	0.000	-0.001	-0.000
reviews_per_month	-0.0114	0.002	-4.635	0.000	-0.016	-0.007
calculated_host_listings_count	0.0023	9.75e-05	23.299	0.000	0.002	0.002
availability_365	0.0005	2.49e-05	18.183	0.000	0.000	0.001

R-squared improved drastically. Also, the RMSE error approached zero which means the model is a good fit.

```
print("RMSE of the model after price modification: ",np.sqrt(((nY_predict-nY_test) ** 2).mean()))
```

```
RMSE of the model after price modification:  4.398927809859766e-15
```

In the end, using the p-values, we removed the minimum_nights feature. Also, the heatmap of the data didn't show any correlation between features. Reduced 6 features in total from the dataset.

Creative Component

Two-Class Classification Using Logistic Regression on Breast Cancer Dataset

The dataset have 5 features and one target variable. The task is to classify the type of cancer into Malignant and Benign. We loaded the dataset and performed EDA.

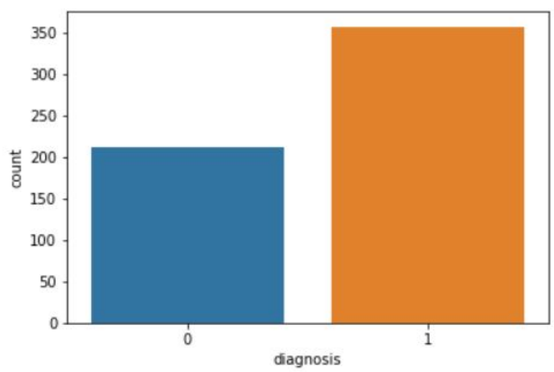
```
df_O= pd.read_csv('Breast_cancer_data.csv')     #load the .csv file into the df_O variable
df_O.columns                                     #checking the columns of the dataset
#df_O.head
```

```
Index(['mean_radius', 'mean_texture', 'mean_perimeter', 'mean_area',
       'mean_smoothness', 'diagnosis'],
      dtype='object')
```

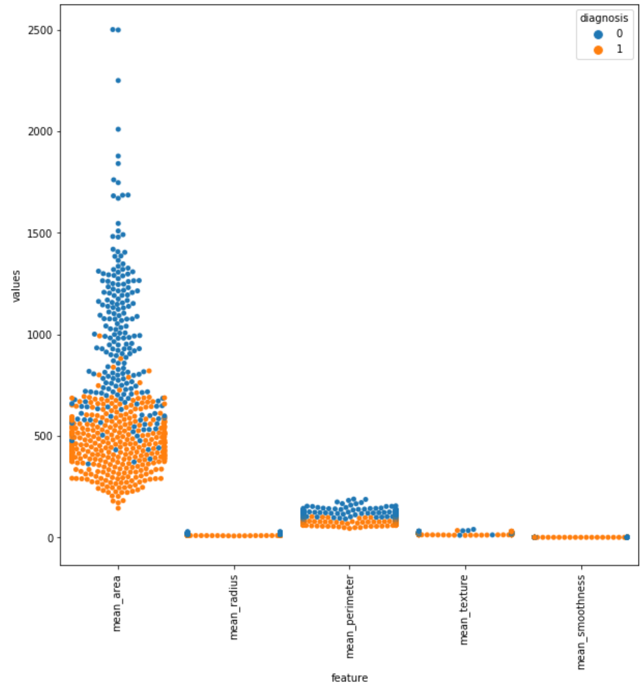
Count of both classes visualized.

```
#countplot for getting the number of entries in each case.
sns.countplot(df_O.diagnosis,label="COUNT")
B,M=df_O.diagnosis.value_counts()
print('Number of Benign: ',B)
print('Number of Malignant : ',M)
```

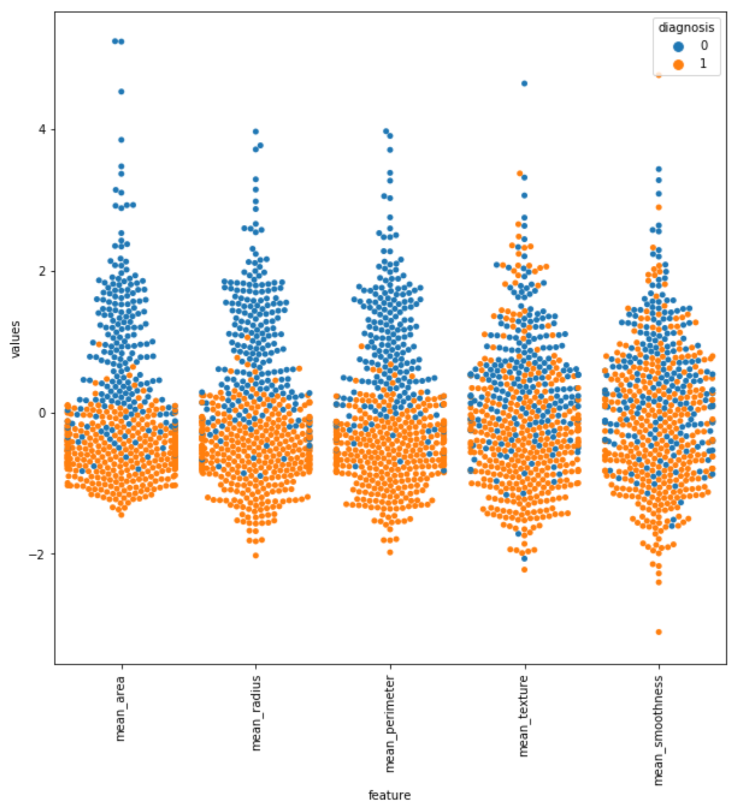
```
Number of Benign:  357
Number of Malignant :  212
```



The swarm plots show the features have different ranges, that means we have to scale them without changing their mean.



After normalizing the features, swarm plot is plotted as follows.



We can see how the two classes are separated into two features. In the mean_smootheness feature two classes didn't seem to be separated, so it won't be good for classification. While features like 'mean_radius' and 'mean_perimeter shows good separation between two classes, which can be useful for classification tasks.

Next, we splitted the data into train & test (60:40) and fitted the logistic function on the dataset including all features in it.

```
y_predict = modelLR.predict(x_test)
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm= confusion_matrix(y_test, y_predict)
print("Confusion Matrix")
print(cm)
```

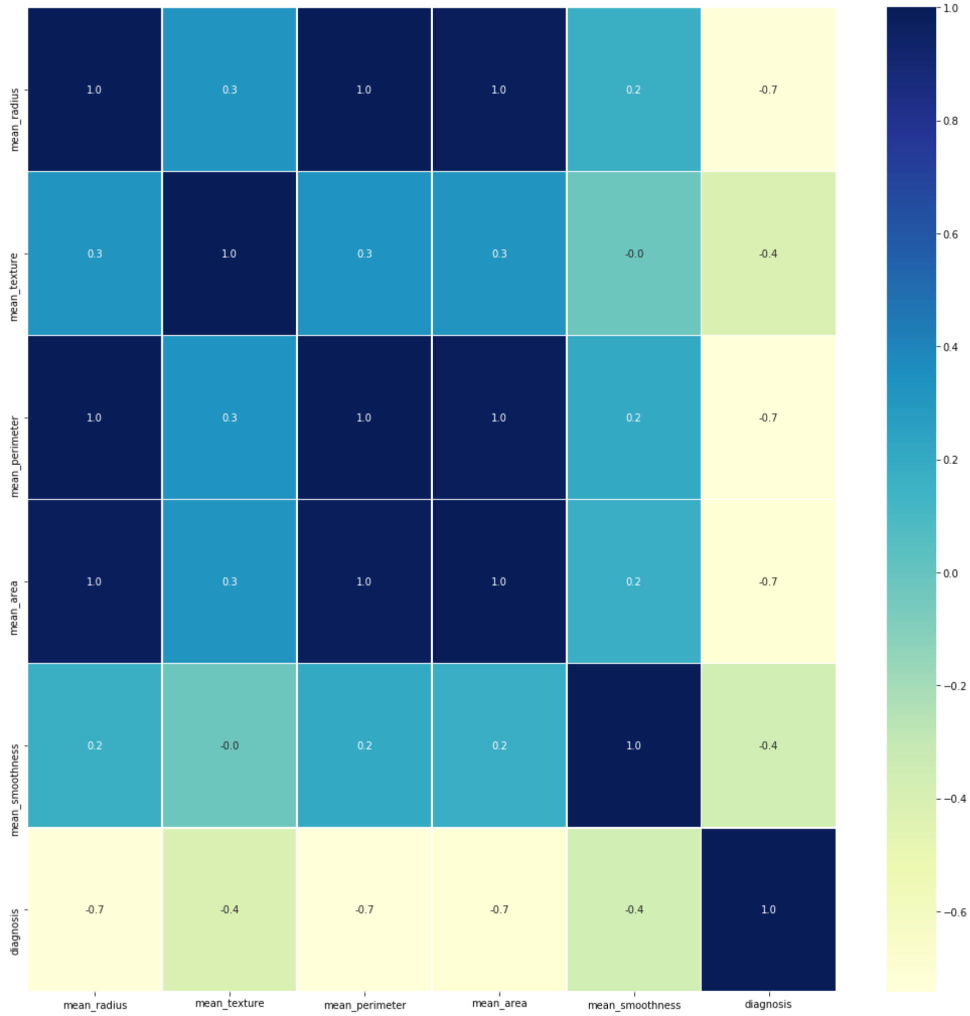
```
Confusion Matrix
[[114   9]
 [ 11 208]]
```

```
from sklearn import metrics
print("Accuracy of Logistic Model:",
      metrics.accuracy_score(y_test, y_predict))
```

```
Accuracy of Logistic Model: 0.9415204678362573
```

Due to the small amount of data, accuracy of the model came to be very high around 94 percent.

Now if we look at the heatmap of the data, the three features 'mean_radius', 'mean_perimeter' and 'mean_area' are highly correlated, in fact they give the same information about the target variable. So we can drop two of them from our model and still perform the classification task.



The accuracy of the model remained the same after feature reduction, which implies we have retained the important features.

```
from sklearn import metrics
print("Accuracy of Logistic Model after feature reduction:",
      metrics.accuracy_score(y_test, y_predict))
```

```
Accuracy of Logistic Model after feature reduction: 0.9415204678362573
```

The dataset had 5 features, we reduced it to 3 using correlation.

Conclusion

We implemented various regression techniques on three different datasets. We performed a prediction task on Boston Housing Dataset and NYC airbnb dataset using Multiple Linear regression. Studied the regression model using residual plots using various parameters like covariance, linearity and normality. Evaluated the performance of the model using r-squared and RMSE values. Carried out the feature reduction using hypothesis testing (p-values) and correlation. Performed logistic regression on Breast Cancer dataset for creative component.

Libraries Used

pandas - for handling the data frames and reading csv files

For visualization:

Seaborn

matplotlib.pyplot

Mpl_toolkits.mplot3d

numpy - for transformation of datasets

Sklearn.model_selection - for splitting datasets

Sklearn.linear_model - for regression models

Statistics - for calculating mean

References

1. https://www.researchgate.net/post/Is_linear_regression_valid_when_the_outcome_dependant_variable_not_normally_distributed
2. https://www.researchgate.net/post/Is_linear_regression_valid_when_the_outcome_dependant_variable_not_normally_distributed
3. https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit
4. https://stats.stackexchange.com/questions/12053/what-should-i-check-for-normality-raw-data-or-residuals
5. https://www.statisticshowto.datasciencecentral.com/multicollinearity/
6. https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86
7. https://www.statisticshowto.datasciencecentral.com/residual-plot/
8. https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics
9. https://towardsdatascience.com/linear-regression-in-python-9a1f5f000606
10. https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/
11. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.htmll
12. http://docs.statwing.com/interpreting-residual-plots-to-improve-your-regression/#outlier-header