1. EDA, FE and Support Vector Classifier Model (Wine Dataset)

2. EDA, FE and Support Vector Regressor Model (Grad Admission Dataset)

Shubham Verma

Follow me on Linkedin: https://lnkd.in/gfPhvjMR

Follow me on GitHub: https://lnkd.in/gky-wyFJ

For the code please check out my Machine Learning repository on GitHub

1. EDA and FE for both dataset

1. Data Profiling
2. Statistical analysis
3. Graphical Analysis
4. Data Scaling

2. SVC on Wine Dataset

1. SVC Model
2. Performance metrics for above model
3. Performance improvement using GridSearchCV

3. SVR on Graduation Admission Dataset

1. SVR Model
2. Performance metrics for above model

Wine Dataset: https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/master/winequality-red.csv

Grad Admission Dataset: https://raw.githubusercontent.com/srinivasav22/Graduate-Admission-Prediction/master/Admission_Predict_Ver1.1.csv

1.0 Importing required libraries

In [ ]:
```
### Pandas and Numpy
import pandas as pd
import numpy as np

### Visualisation libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

### For Q-Q Plot
import scipy.stats as stats

### To ignore warnings
import warnings
warnings.filterwarnings('ignore')

### Machine Learning libraries
import sklearn
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, r2_score

### To be able to see maximum columns on screen
pd.set_option('display.max_columns', 500)

### To save the model
import pickle
```

2.0 Support Vector Classifier for Wine Dataset

In [41]:
```
from IPython import display
display.Image("wine.png")
```

Out[41]:

2.1 Importing Wine Dataset and doing statistical analysis

In [4]:
```
### Importing wine dataset from github
dataset=pd.read_csv("https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/master/winequality-red.csv")
dataset.head()
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [5]:
```
### getting unique values for quality feature
dataset.quality.unique()
```

Out[5]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [42]:
```
### getting count of record for each unique value in quality
dataset.quality.value_counts()
```

Out[42]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [7]:
```
# getting null values and datatypes of all features
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [8]:
```
### getting info about numerical features
dataset.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.0	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
volatile acidity	1599.0	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
citric acid	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

In [43]:
```
### getting null values in each feature
dataset.isnull().sum()
```

Out[43]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

2.2 Visualising Numerical data

In [46]:
```
### getting list of numerical features
numerical_features=[feature for feature in dataset.columns if dataset[feature].dtypes!='O']
print(numerical_features)
```

```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxid
e', 'density', 'pH', 'sulphates', 'alcohol', 'quality']
```

In [47]:
```
### getting count of unique value in each feature
for feature in numerical_features:
    print("'{}' has '{}' No. of unique values".format(feature, dataset[feature].nunique()))
```

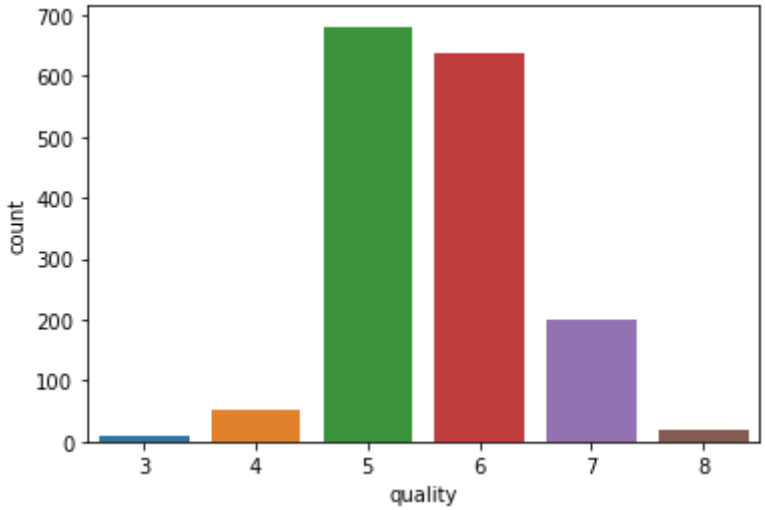
```
'fixed acidity' has '96' No. of unique values
'volatile acidity' has '143' No. of unique values
'citric acid' has '80' No. of unique values
'residual sugar' has '91' No. of unique values
'chlorides' has '153' No. of unique values
'free sulfur dioxide' has '60' No. of unique values
'total sulfur dioxide' has '144' No. of unique values
'density' has '436' No. of unique values
'pH' has '89' No. of unique values
'sulphates' has '96' No. of unique values
'alcohol' has '65' No. of unique values
'quality' has '6' No. of unique values
```

In [48]:
```
continuous_features=[feature for feature in numerical_features if dataset[feature].nunique()>6]
print(continuous_features)
```

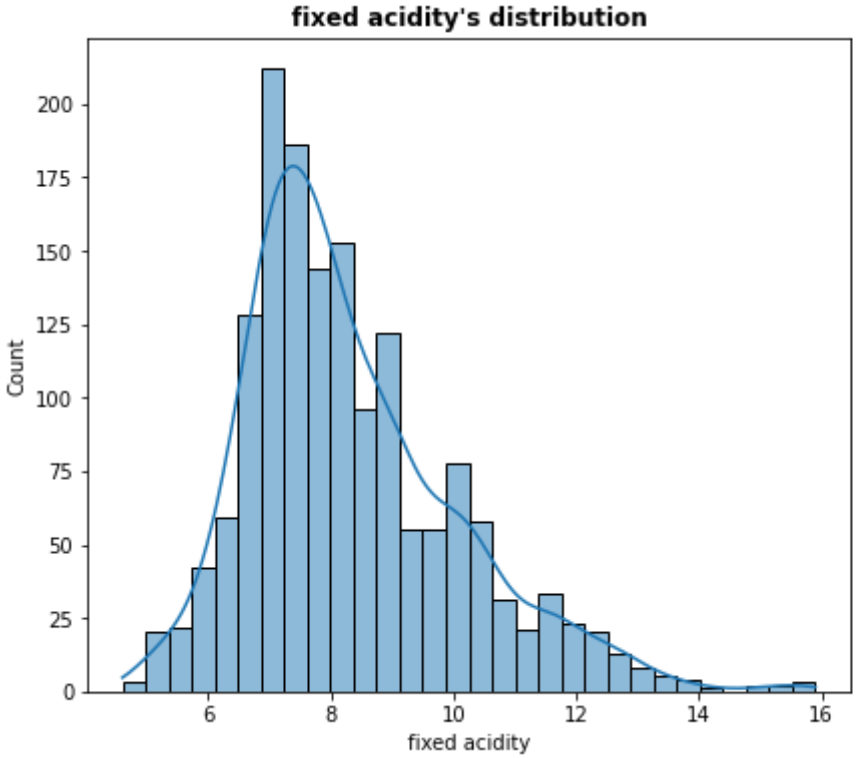
```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxid
e', 'density', 'pH', 'sulphates', 'alcohol']
```

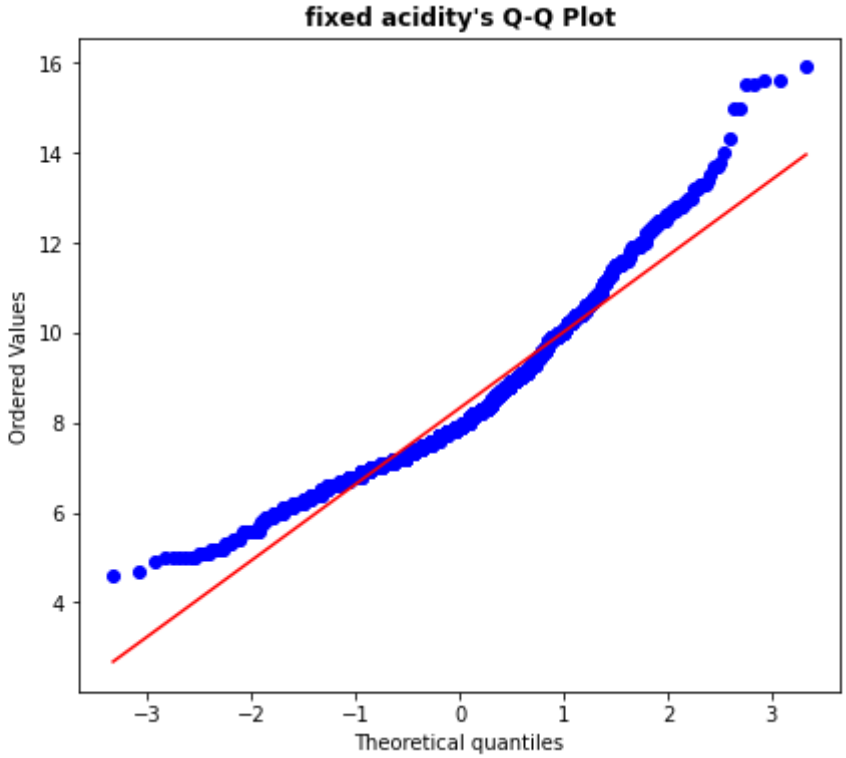
In [14]:
```
### visualising count of quality feature
sns.countplot(data=dataset, x='quality')
```

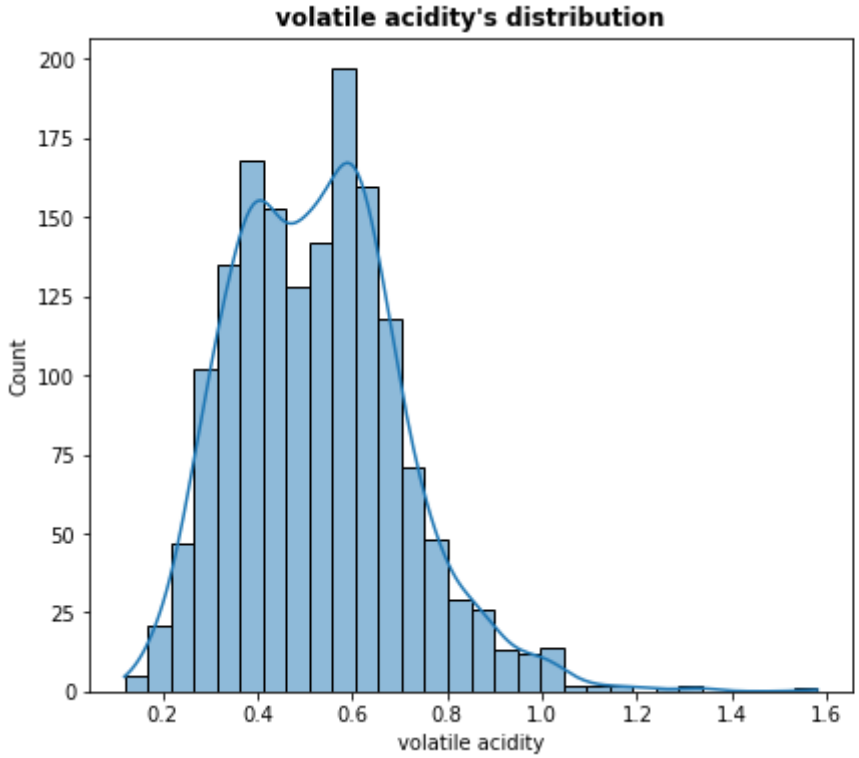
Out[14]:
```
<AxesSubplot:xlabel='quality', ylabel='count'>
```

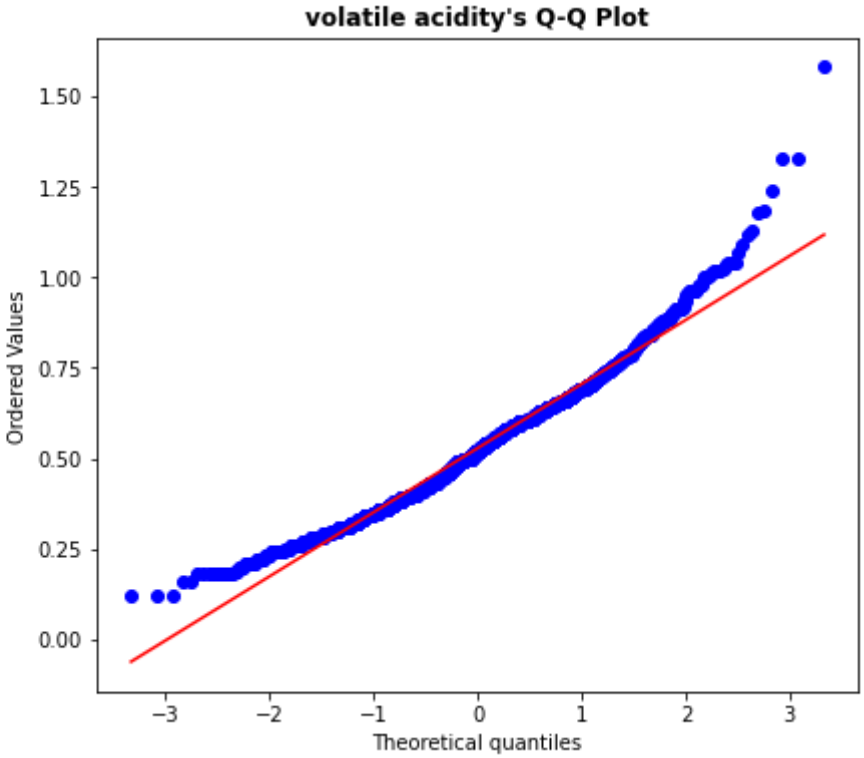


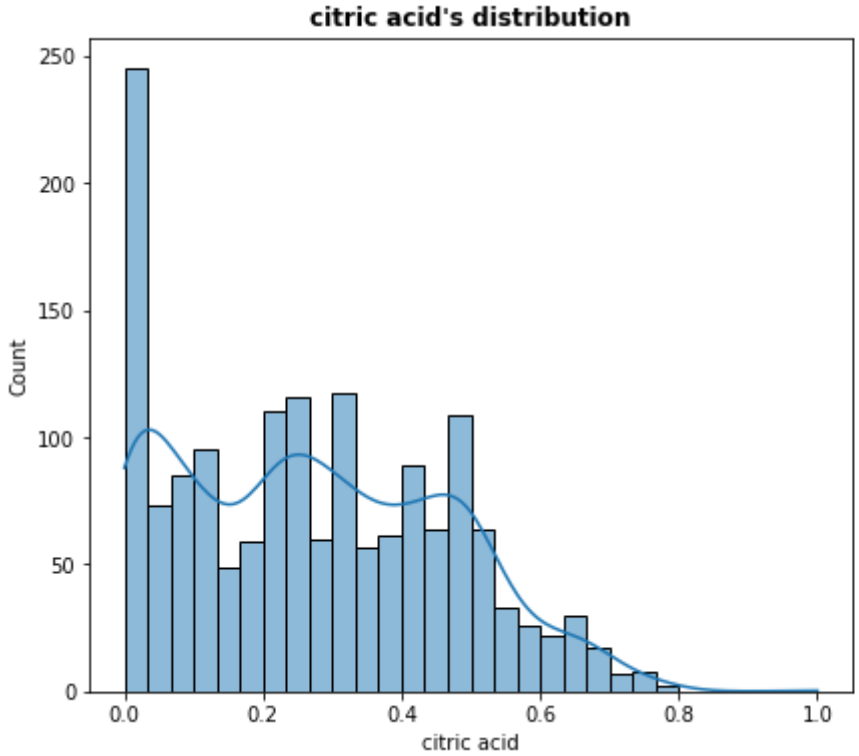
In [15]:
```
### Checking distribution of Continuous numerical features

for i in continuous_features:
    plt.figure(figsize=(15,6))
    plt.subplot(121)
    sns.histplot(data=dataset, x=i, kde=True, bins=30)
    plt.title("{}'s distribution".format(i),fontweight="bold")

    plt.subplot(122)
    stats.probplot(dataset[i], dist='norm', plot=plt)
    plt.title("{}'s Q-Q Plot".format(i),fontweight="bold")
    plt.show();
```

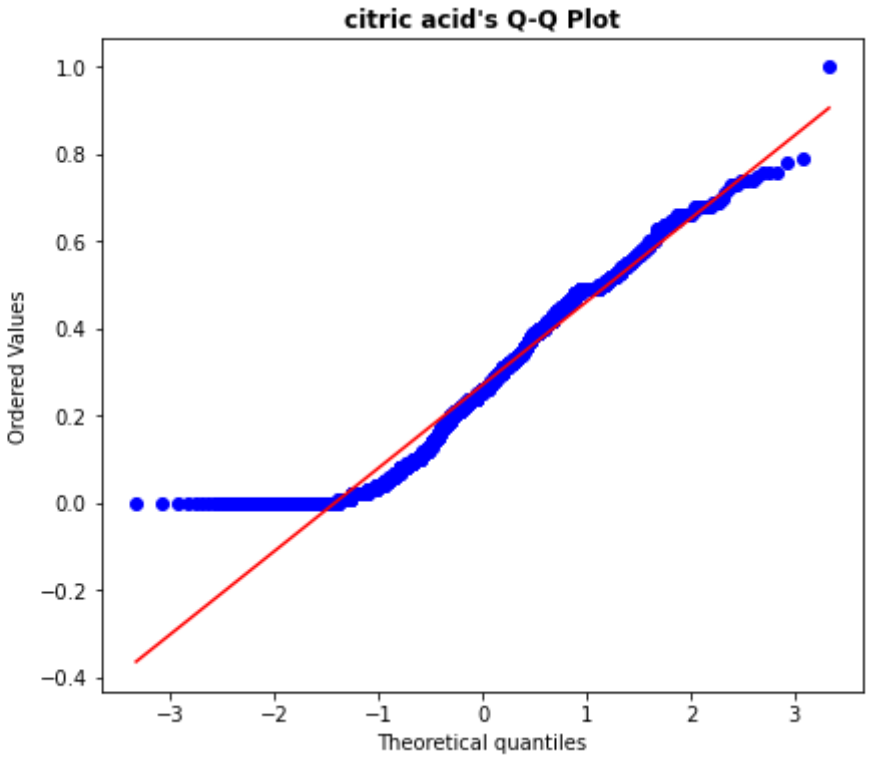

fixed acidity's distribution
Count
fixed acidity
0
25
50
75
100
125
150
175
200
6
8
10
12
14
16

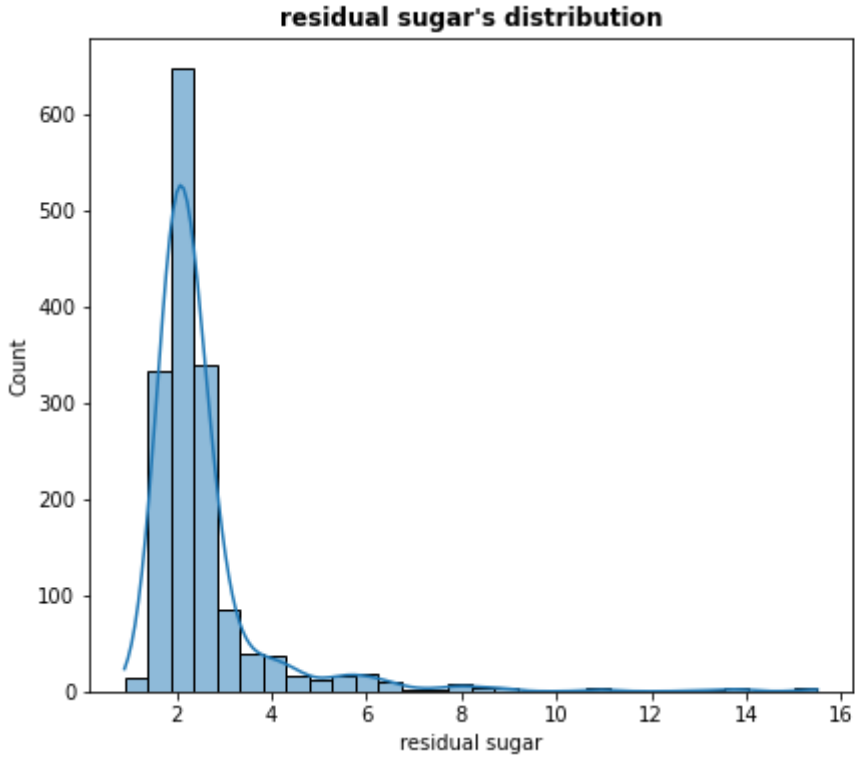

fixed acidity's Q-Q Plot
Ordered Values
Theoretical quantiles
4
6
8
10
12
14
16
−3
−2
−1
0
1
2
3

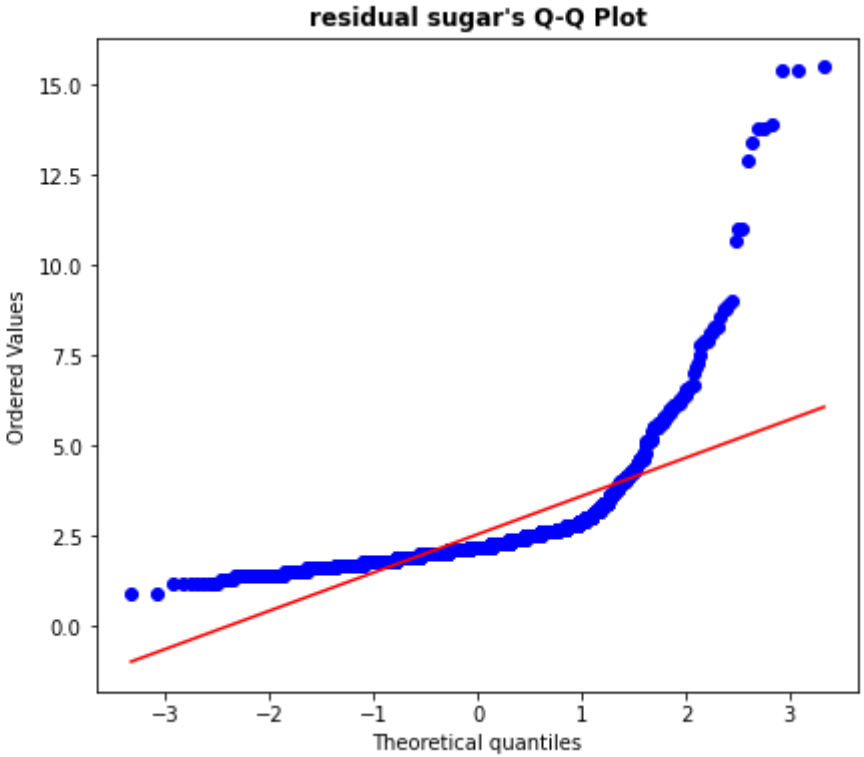

volatile acidity's distribution
Count
volatile acidity
0
25
50
75
100
125
150
175
200
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

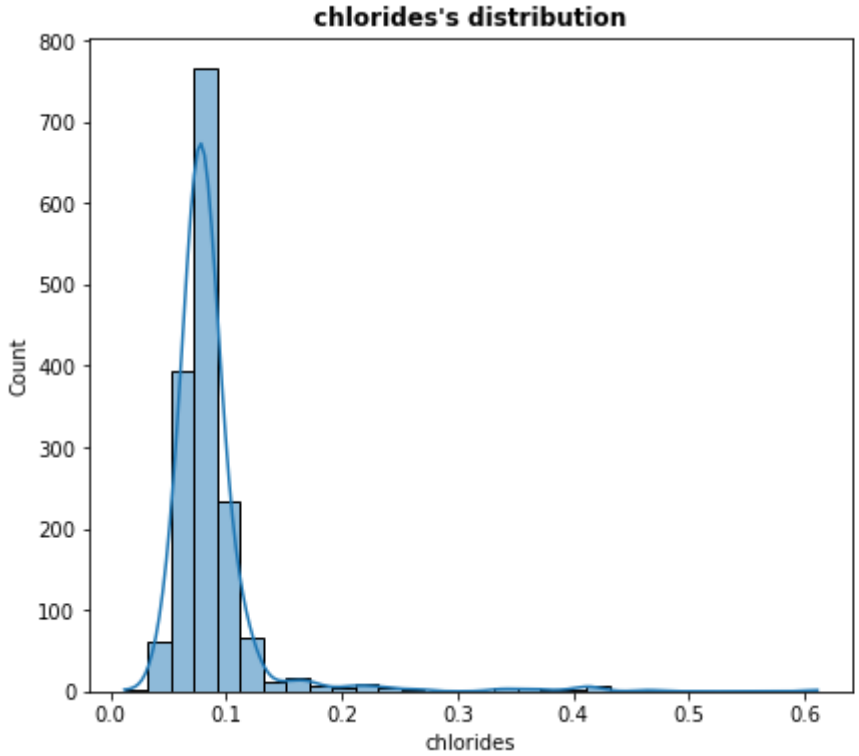

volatile acidity's Q-Q Plot
Ordered Values
Theoretical quantiles
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−3
−2
−1
0
1
2
3

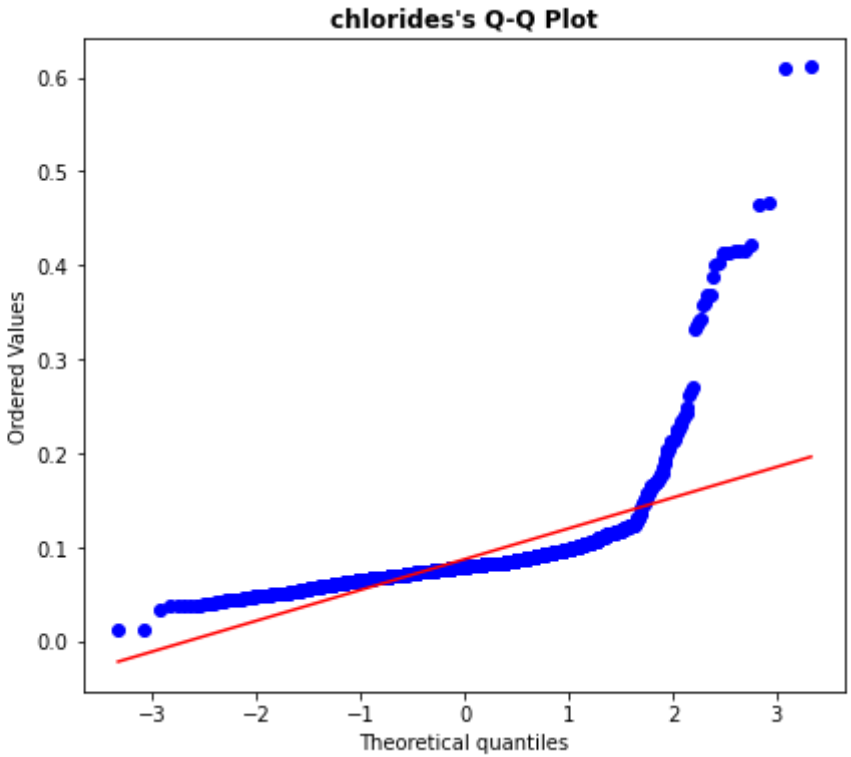

citric acid's distribution
Count
citric acid
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0

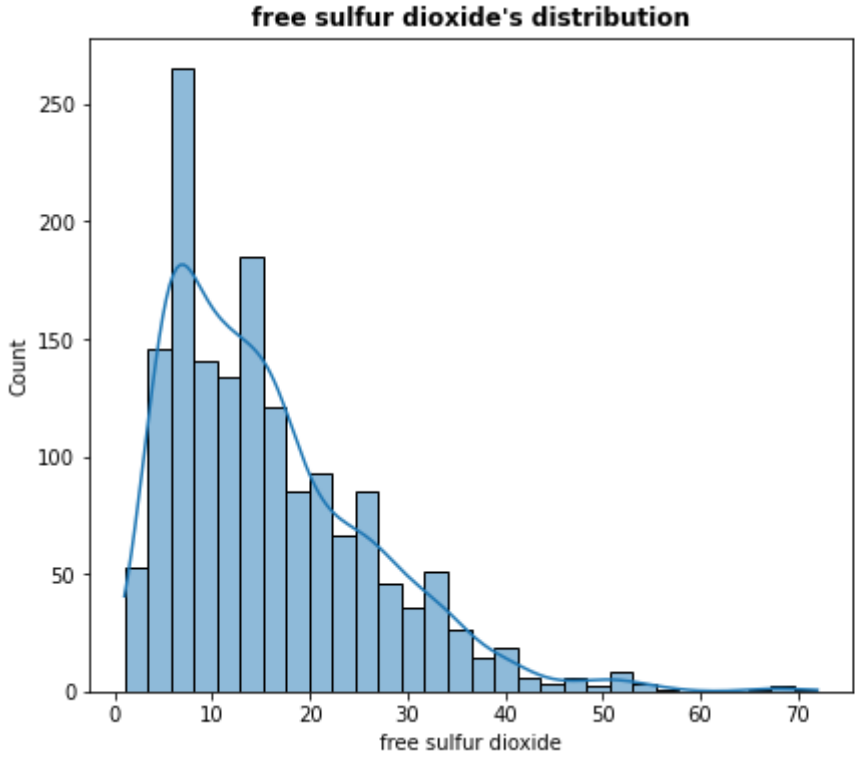

citric acid's Q-Q Plot
Ordered Values
Theoretical quantiles
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
−3
−2
−1
0
1
2
3

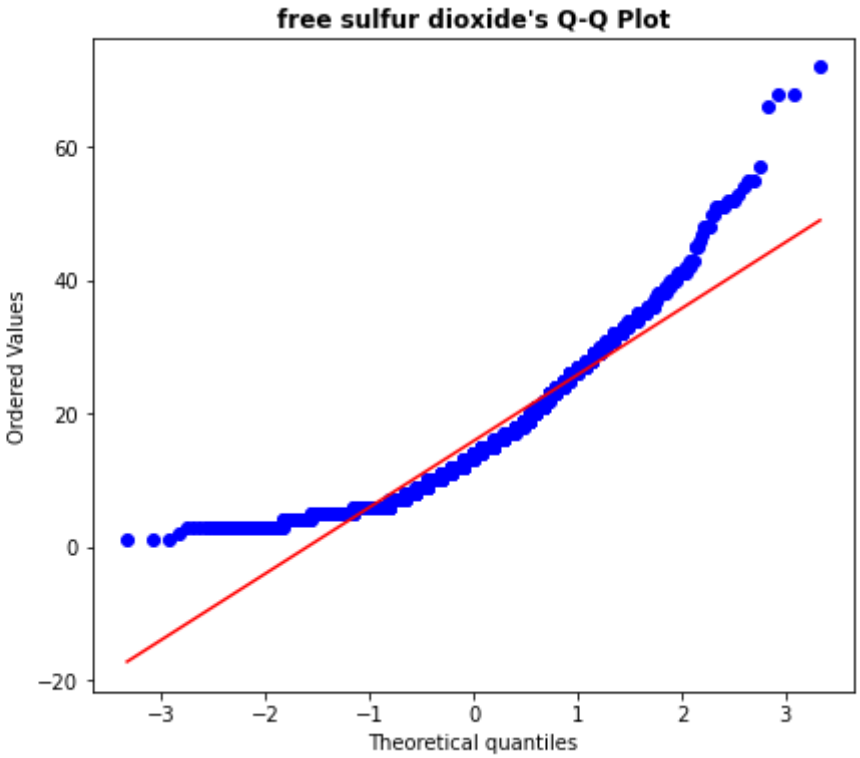

residual sugar's distribution
Count
residual sugar
0
100
200
300
400
500
600
2
4
6
8
10
12
14
16

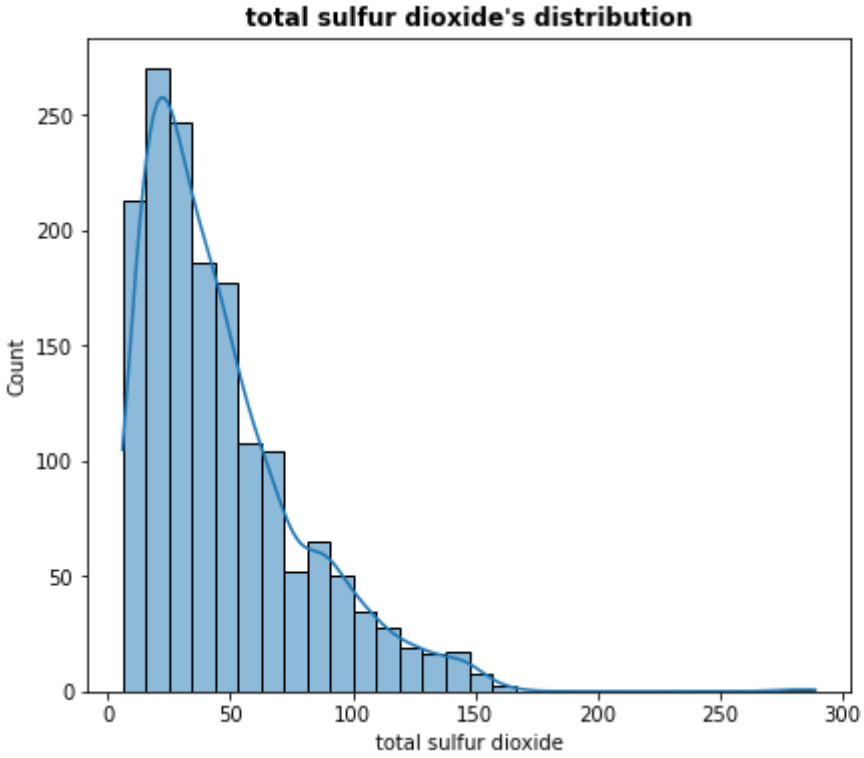

residual sugar's Q-Q Plot
Ordered Values
Theoretical quantiles
0.0
2.5
5.0
7.5
10.0
12.5
15.0
−3
−2
−1
0
1
2
3

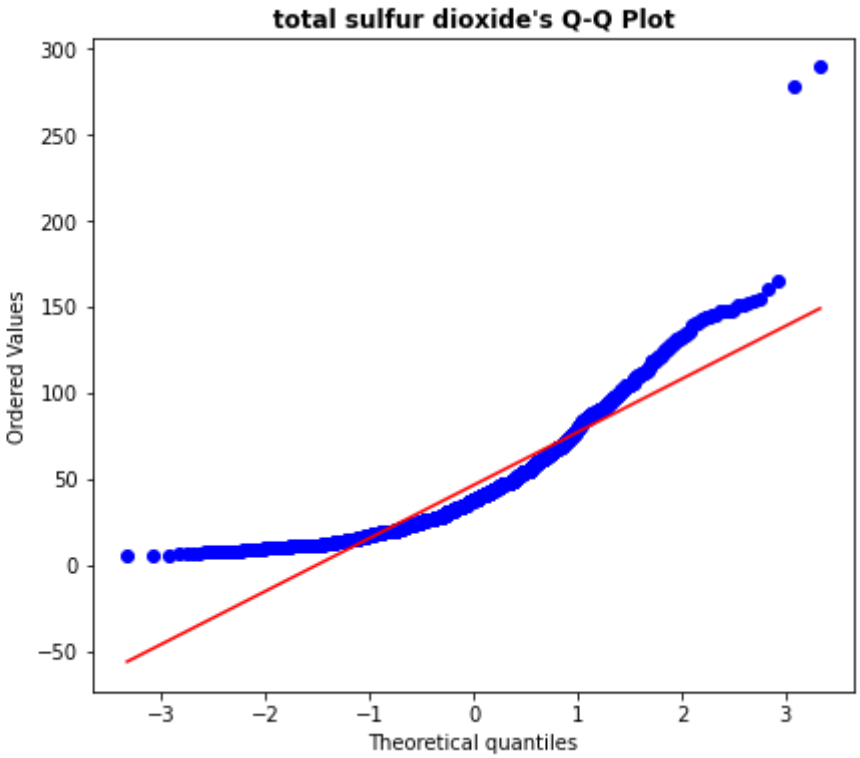

chlorides's distribution
Count
chlorides
0
100
200
300
400
500
600
700
800
0.0
0.1
0.2
0.3
0.4
0.5
0.6

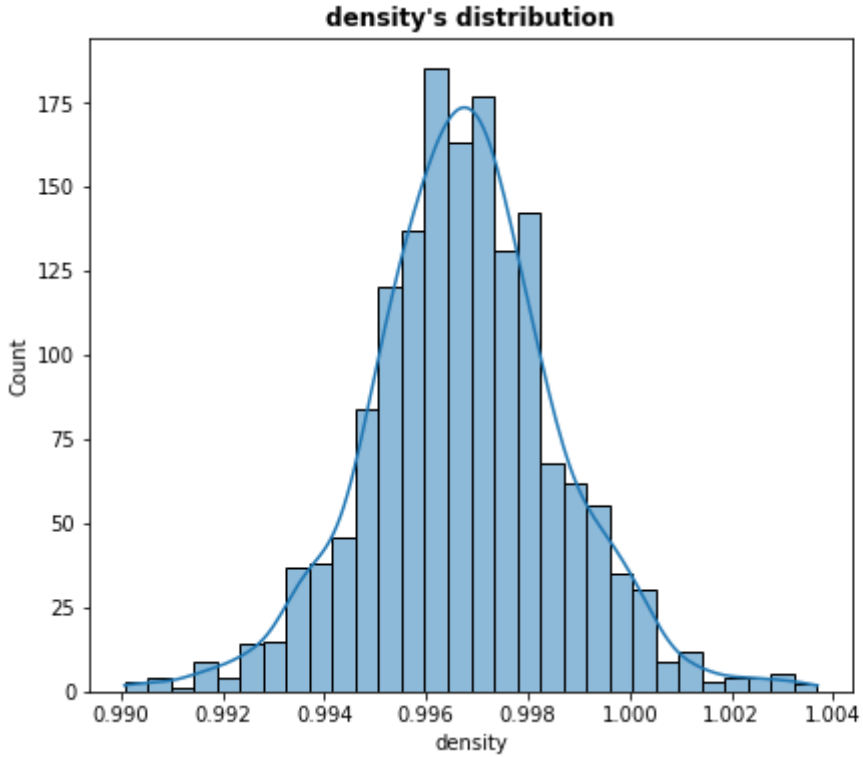

chlorides's Q-Q Plot
Ordered Values
Theoretical quantiles
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−3
−2
−1
0
1
2
3

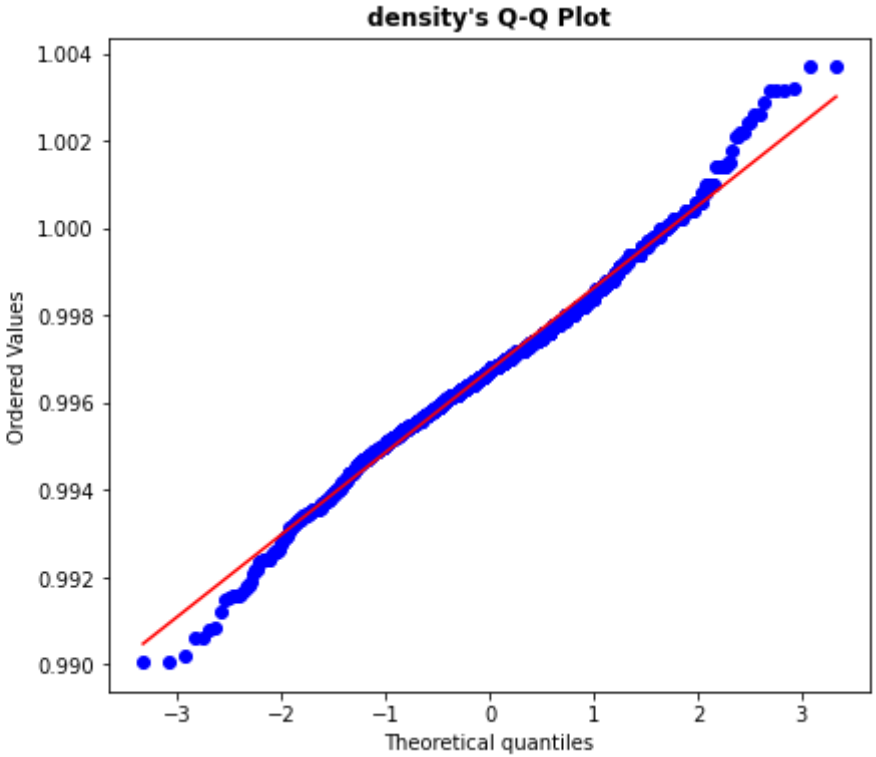

free sulfur dioxide's distribution
Count
free sulfur dioxide
0
50
100
150
200
250
0
10
20
30
40
50
60
70

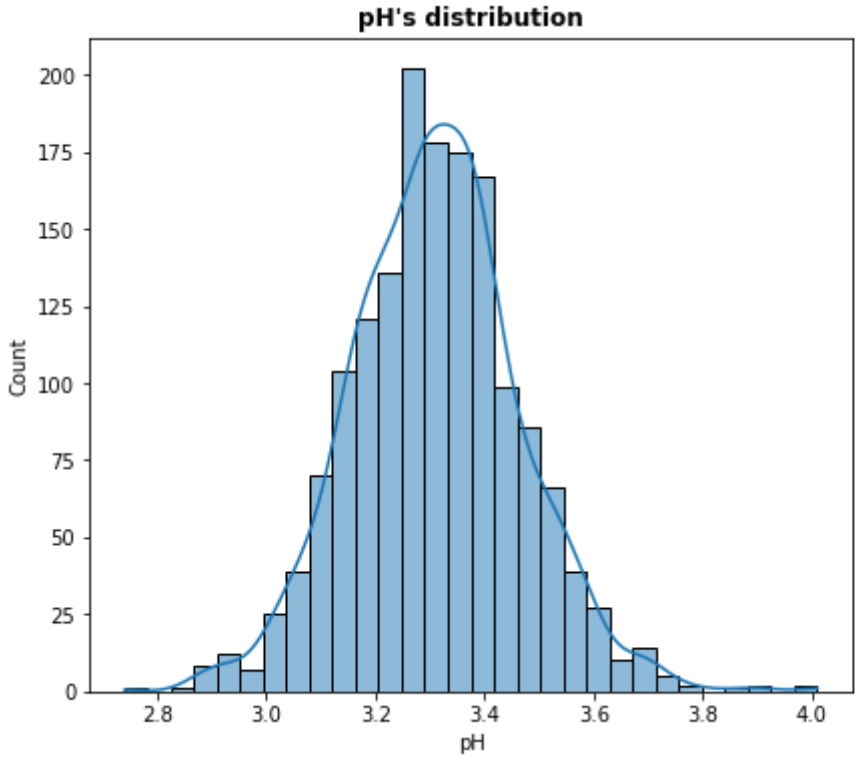

free sulfur dioxide's Q-Q Plot
Ordered Values
Theoretical quantiles
−20
0
20
40
60
−3
−2
−1
0
1
2
3

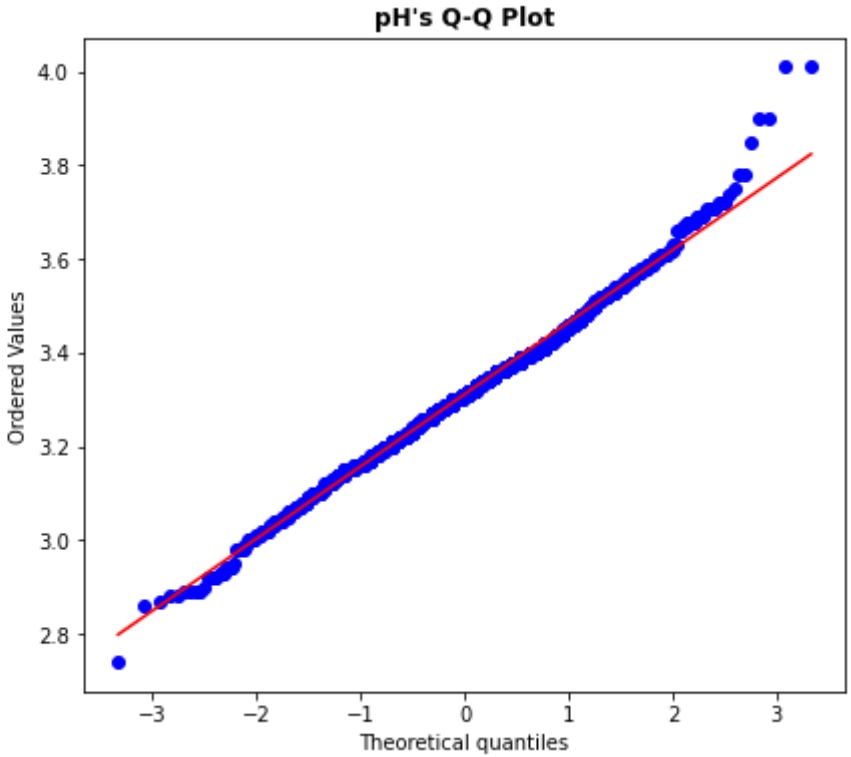

total sulfur dioxide's distribution
Count
total sulfur dioxide
0
50
100
150
200
250
300

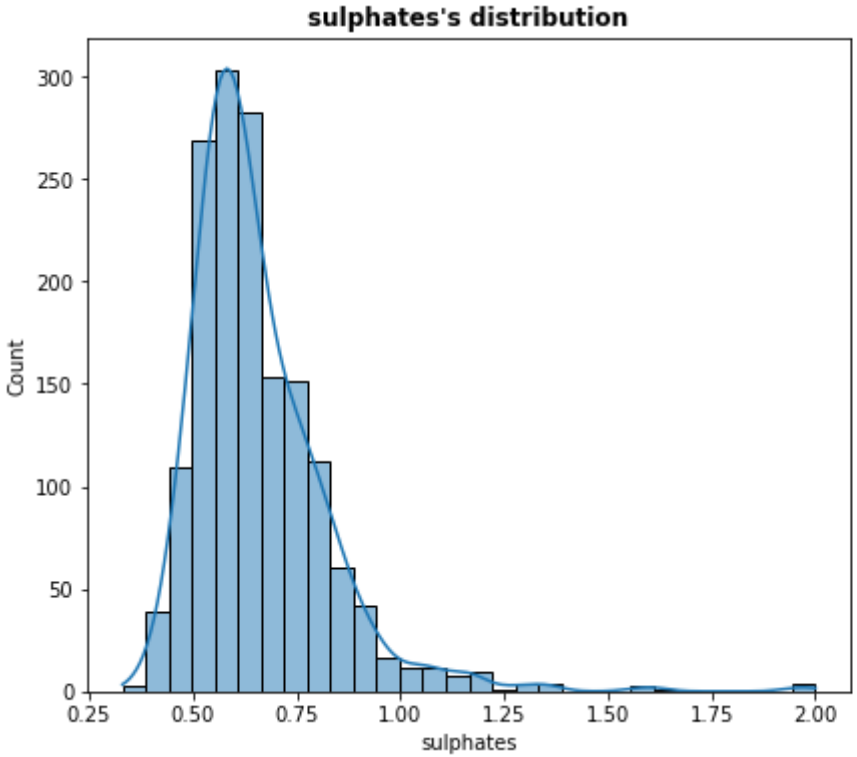

total sulfur dioxide's Q-Q Plot
Ordered Values
Theoretical quantiles
−50
0
50
100
150
200
250
300
−3
−2
−1
0
1
2
3

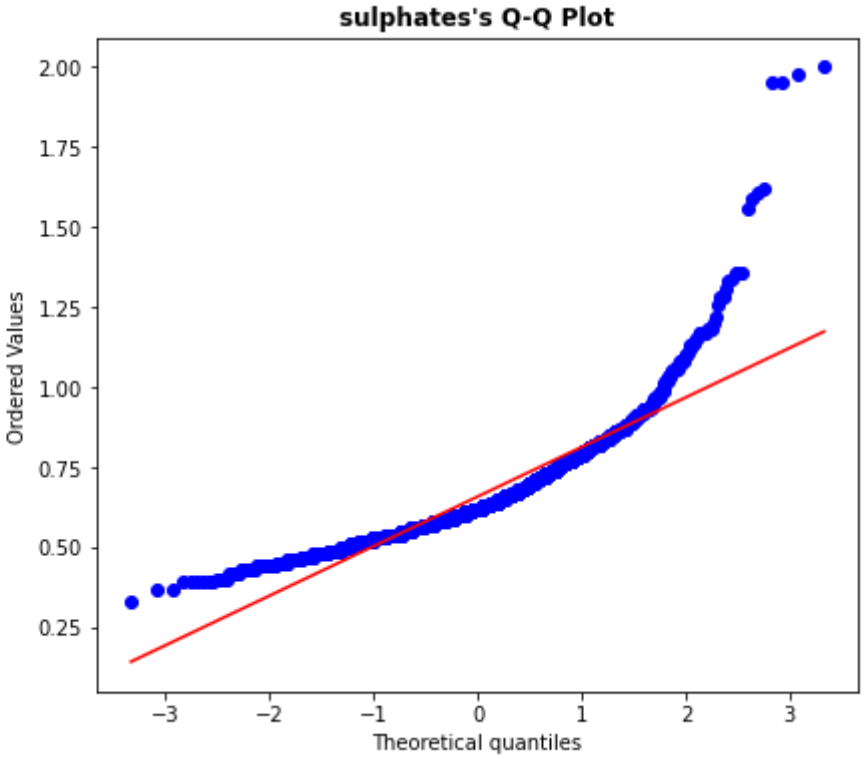

density's distribution
Count
density
0
25
50
75
100
125
150
175
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004

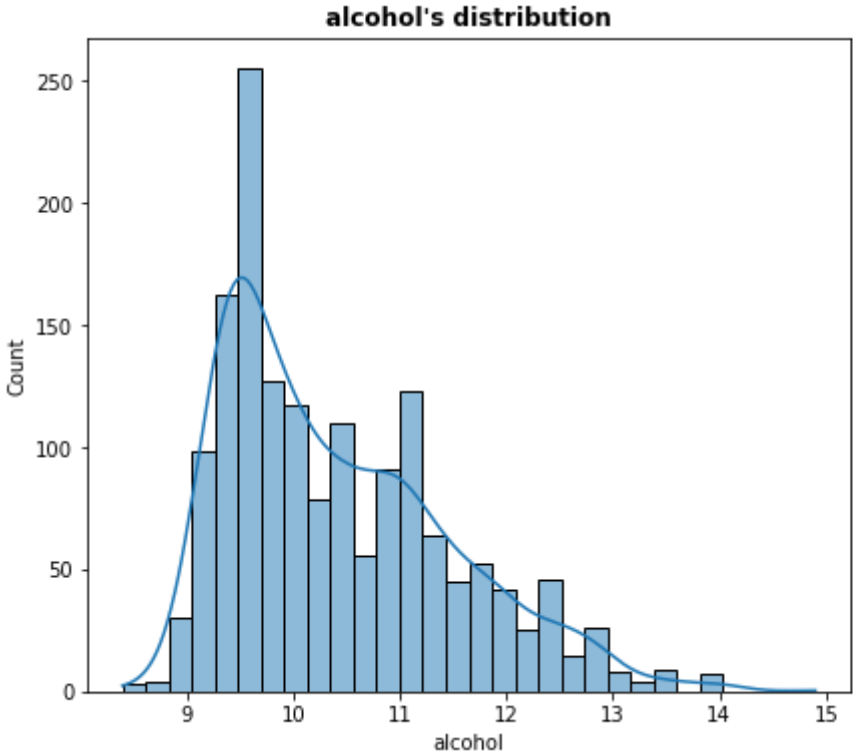

density's Q-Q Plot
Ordered Values
Theoretical quantiles
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004
−3
−2
−1
0
1
2
3

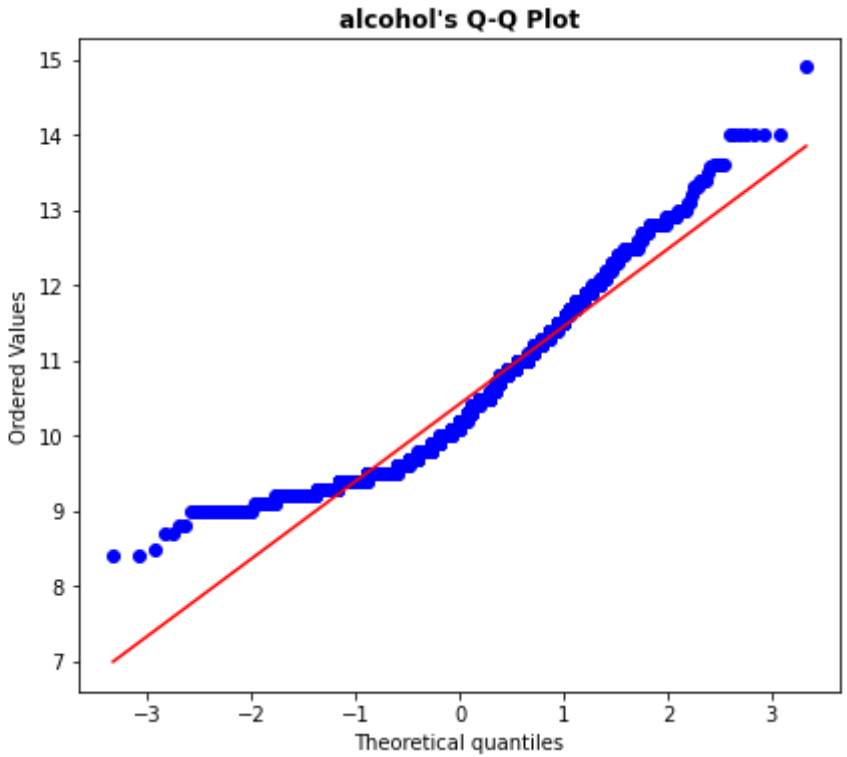

pH's distribution
Count
pH
0
25
50
75
100
125
150
175
200
2.8
3.0
3.2
3.4
3.6
3.8
4.0


pH's Q-Q Plot
Ordered Values
Theoretical quantiles
2.8
3.0
3.2
3.4
3.6
3.8
4.0
−3
−2
−1
0
1
2
3

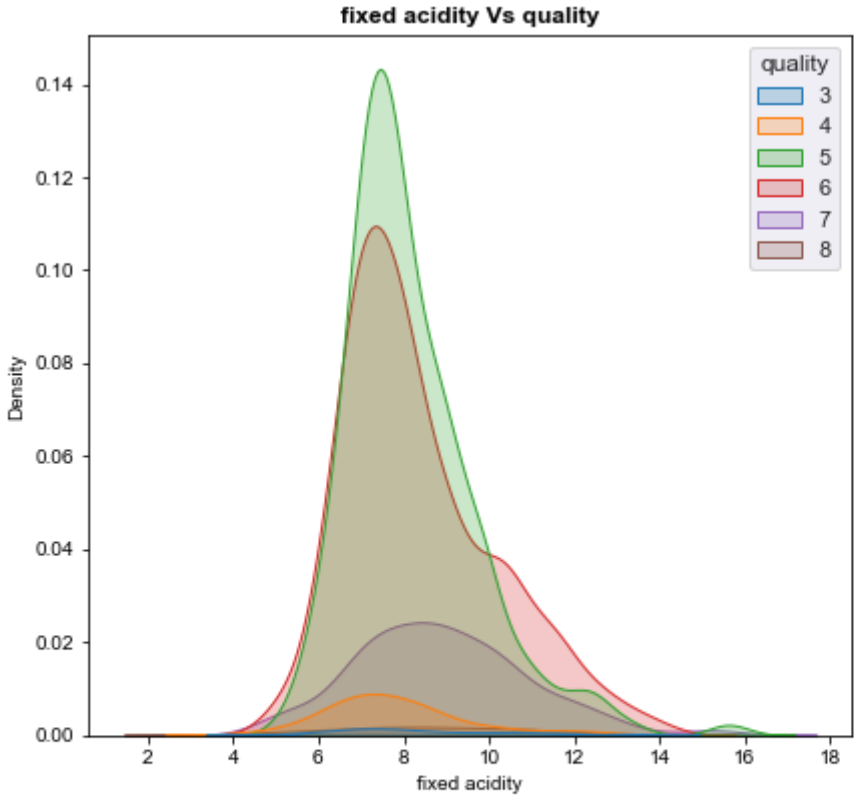

sulphates's distribution
Count
sulphates
0
50
100
150
200
250
300
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

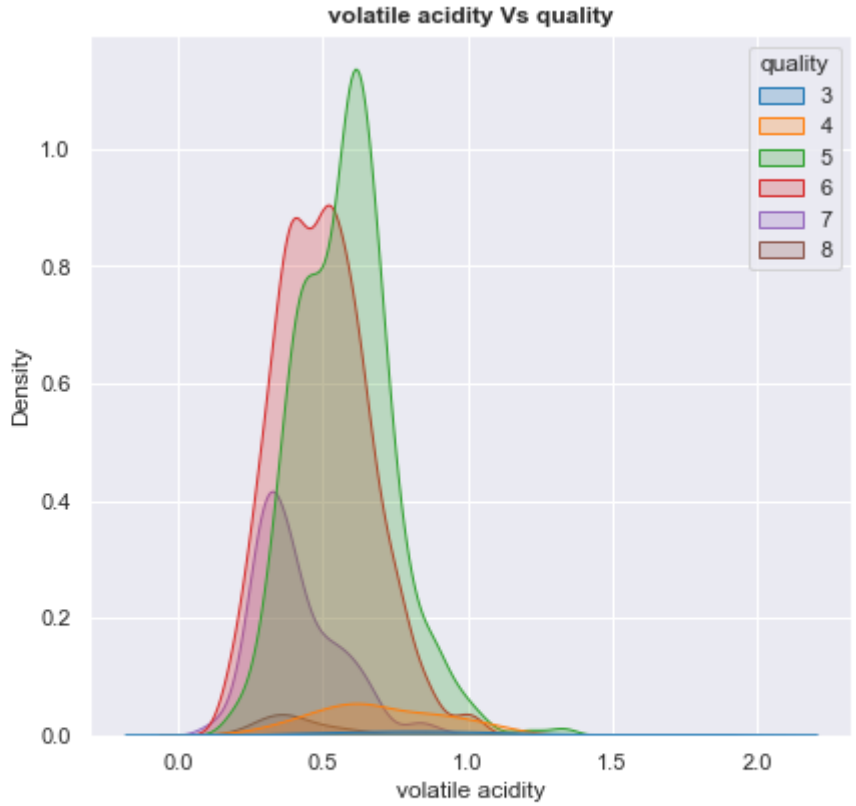

sulphates's Q-Q Plot
Ordered Values
Theoretical quantiles
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
−3
−2
−1
0
1
2
3

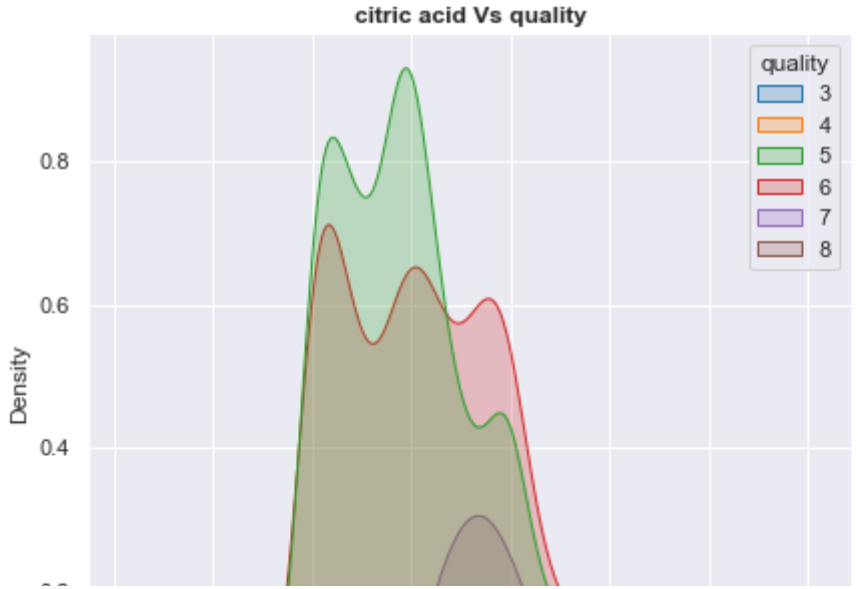




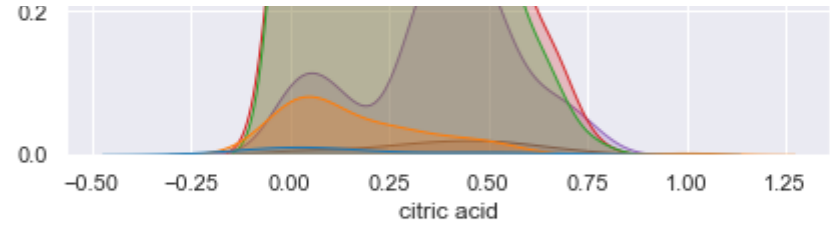
In [16]:
```
### Comparing Continuous numerical features with quality feature
palette1=sns.color_palette("tab10", 6)
plt.figure(figsize=(15,45))
for i in enumerate(continuous_features):
    plt.subplot(6, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(7,7)})
    sns.kdeplot(data=dataset, x=i[1], hue='quality', palette=palette1, fill=True)
    plt.title("{} Vs quality".format(i[1]),fontweight="bold")
```

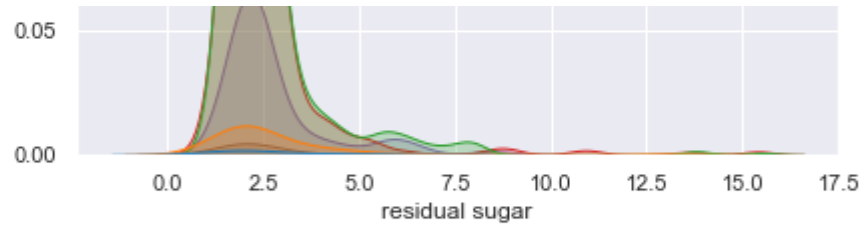

fixed acidity Vs quality
quality
3
4
5
6
7
8
Density
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2
4
6
8
10
12
14
16
18
fixed acidity

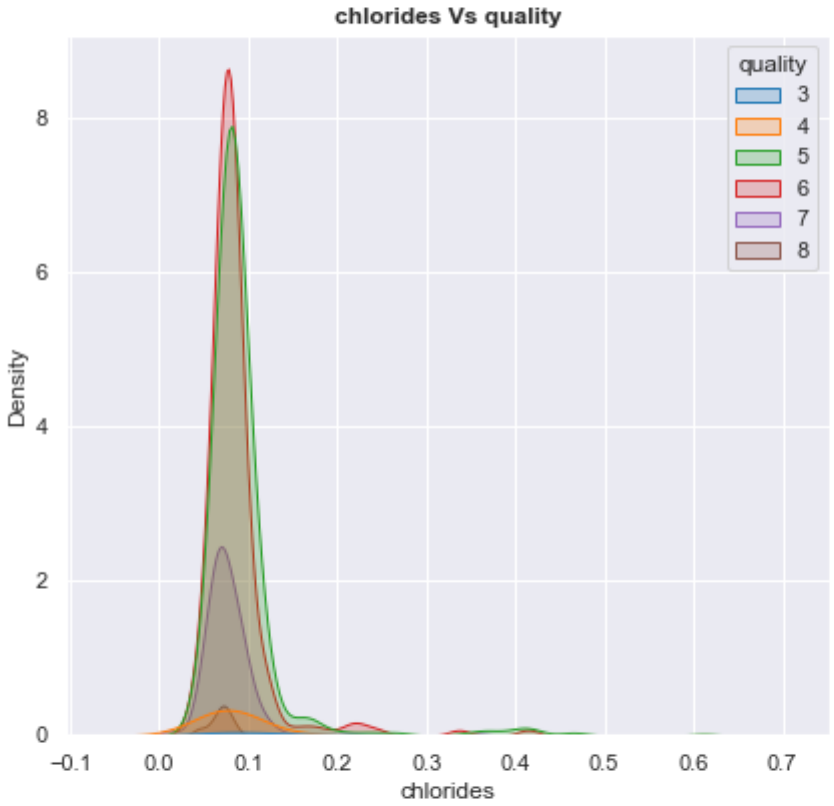

volatile acidity Vs quality
quality
3
4
5
6
7
8
Density
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
volatile acidity

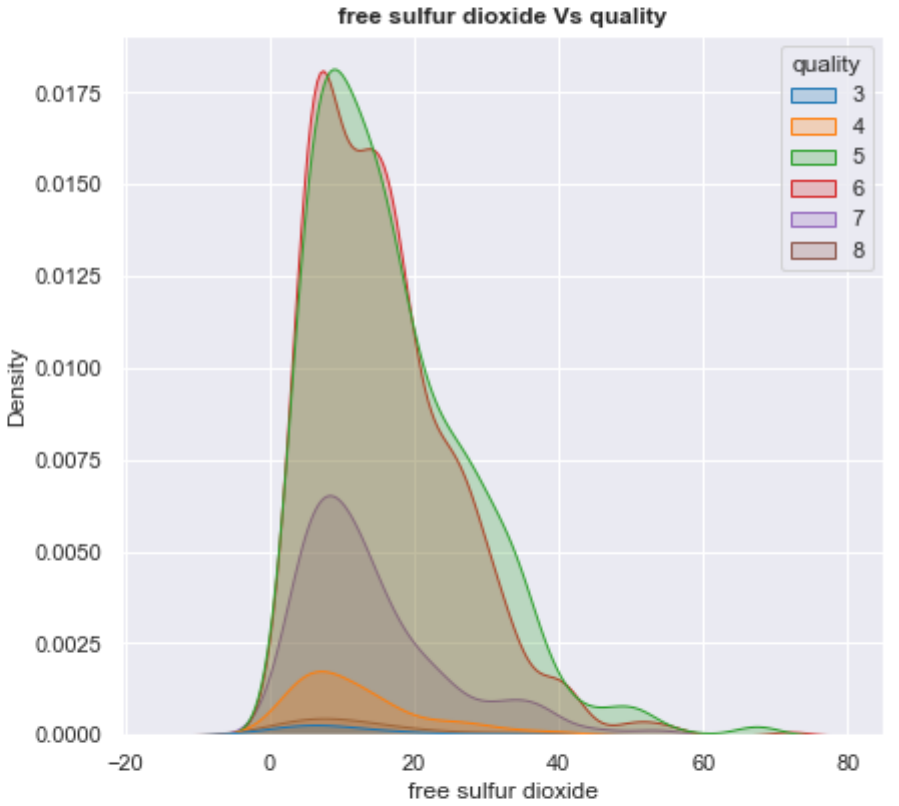

citric acid Vs quality
quality
3
4
5
6
7
8
Density
0.8
0.6
0.4

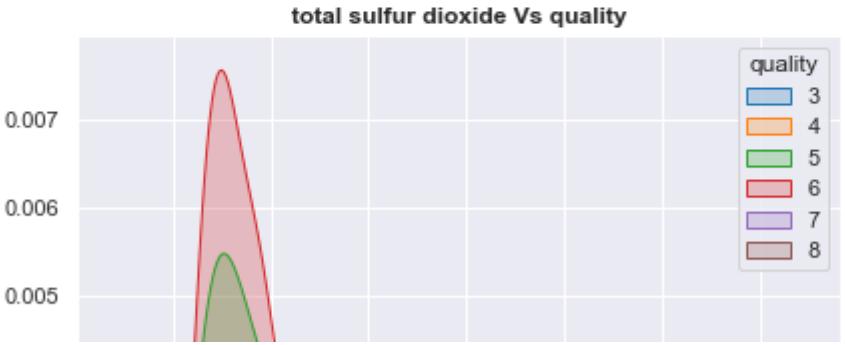

residual sugar Vs quality
quality
3
4
5
6
7
8
Density
0.25
0.20
0.15
0.10

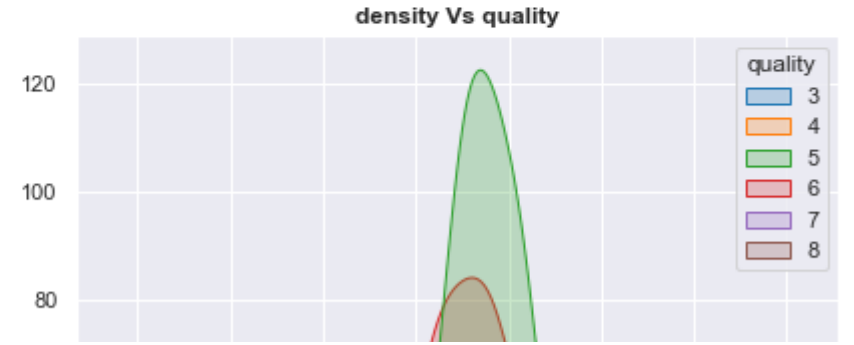

0.2
0.0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
citric acid

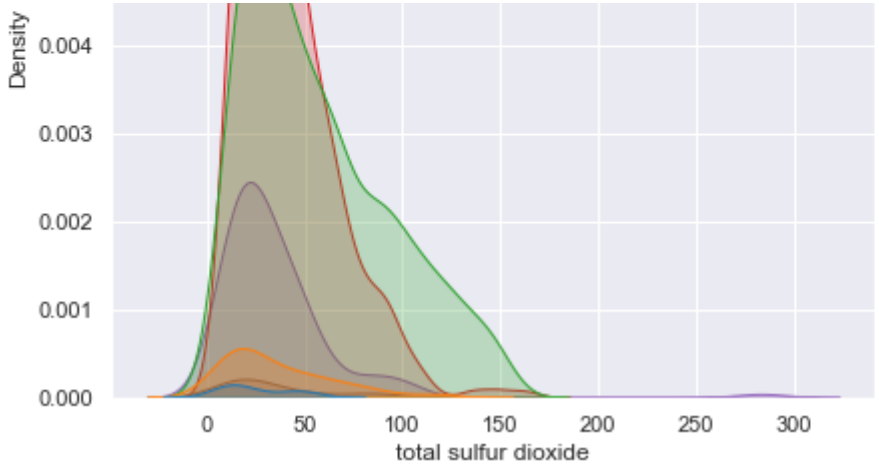

0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
residual sugar

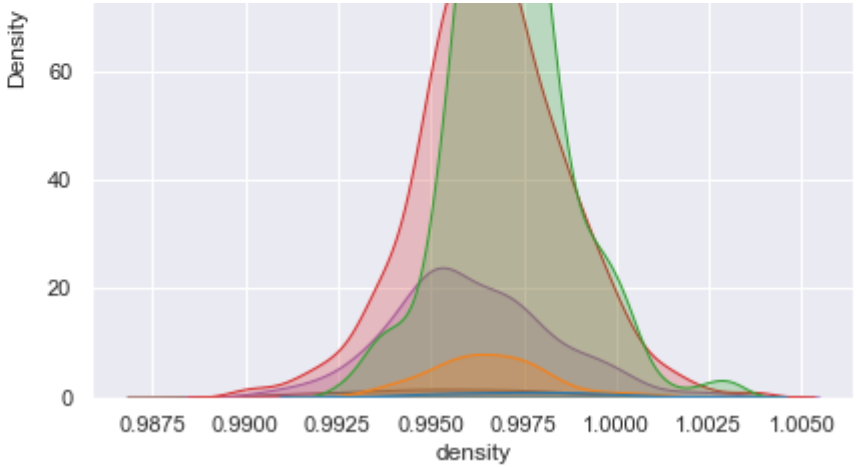

chlorides Vs quality
quality
3
4
5
6
7
8
Density
8
6
4
2
0
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
chlorides

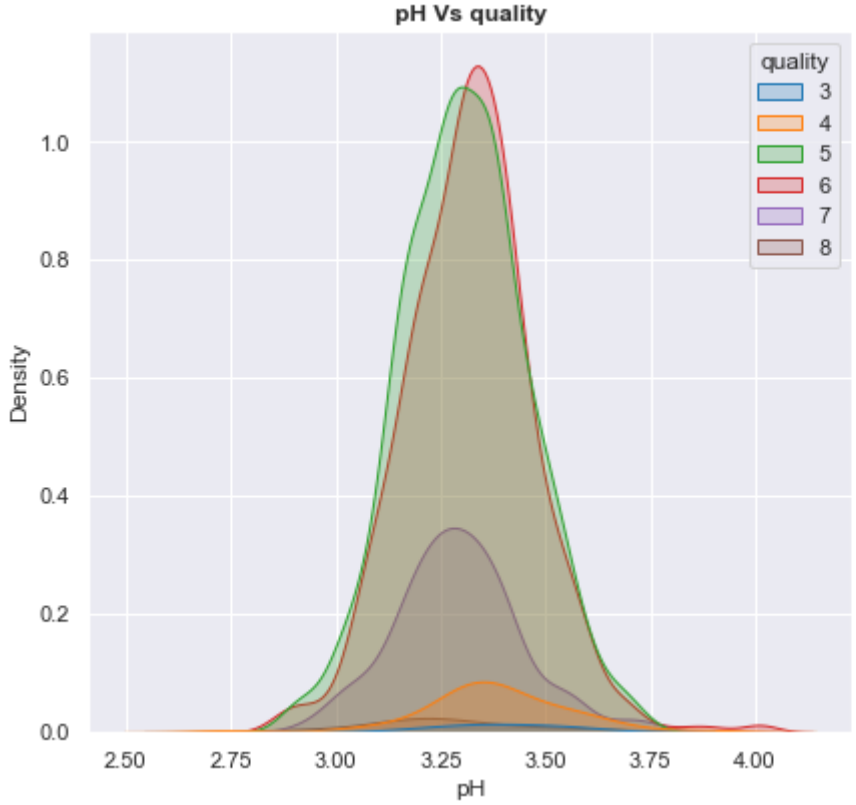

free sulfur dioxide Vs quality
quality
3
4
5
6
7
8
Density
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
−20
0
20
40
60
80
free sulfur dioxide

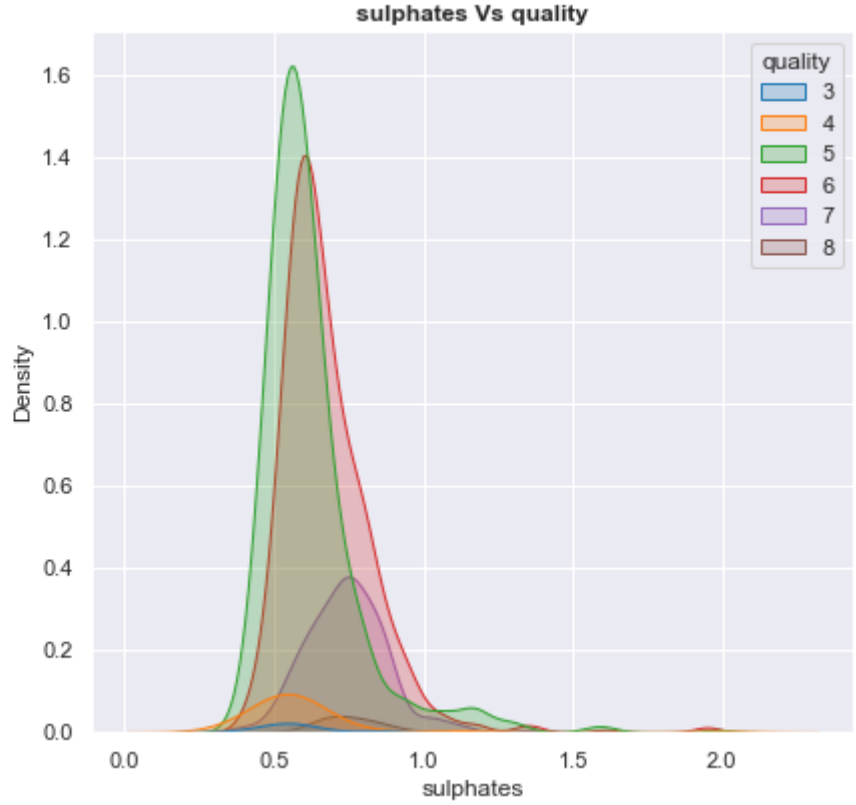

total sulfur dioxide Vs quality
quality
3
4
5
6
7
8
0.007
0.006
0.005


density Vs quality
quality
3
4
5
6
7
8
120
100
80

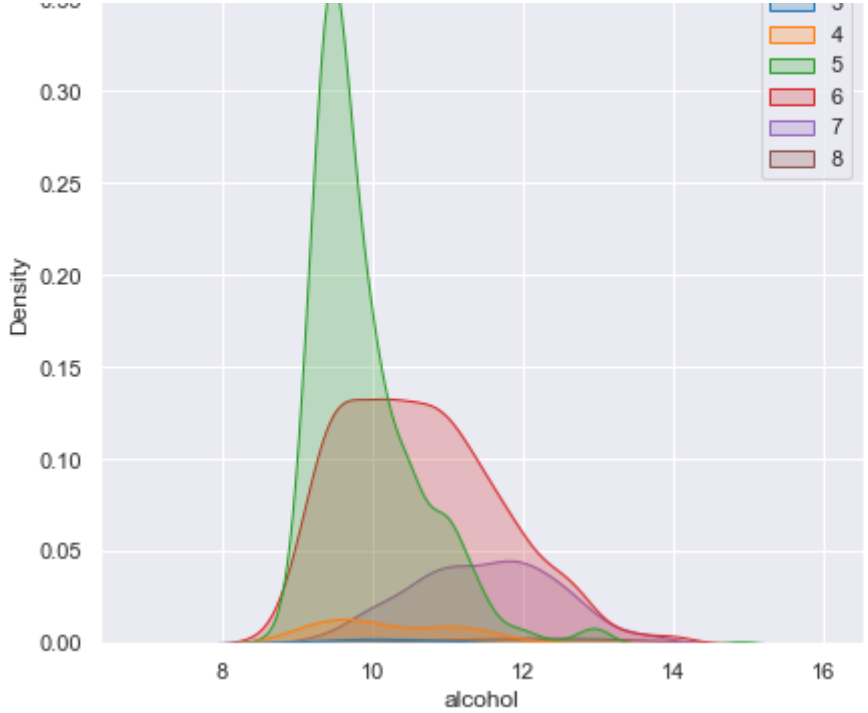

Density
0.004
0.003
0.002
0.001
0.000
0
50
100
150
200
250
300
total sulfur dioxide


Density
60
40
20
0
0.9875
0.9900
0.9925
0.9950
0.9975
1.0000
1.0025
1.0050
density

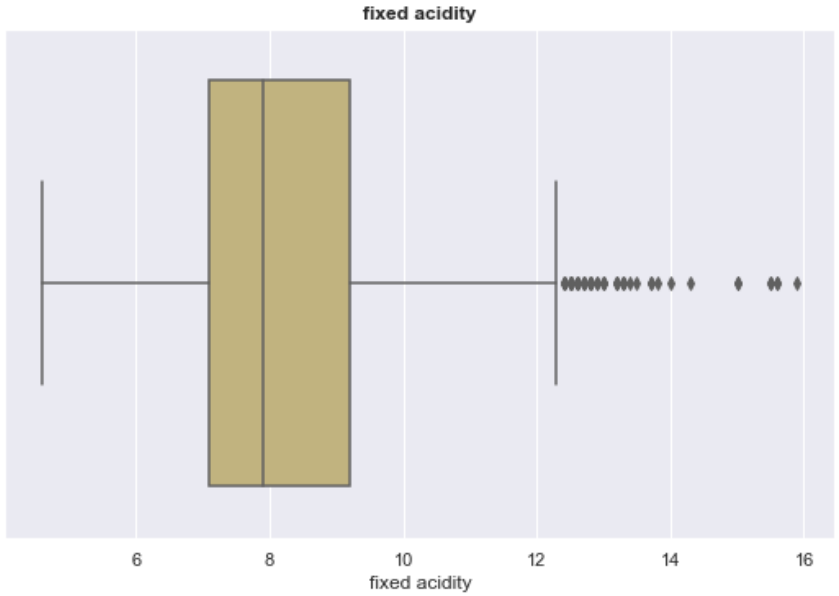

pH Vs quality
quality
3
4
5
6
7
8
Density
1.0
0.8
0.6
0.4
0.2
0.0
2.50
2.75
3.00
3.25
3.50
3.75
4.00
pH

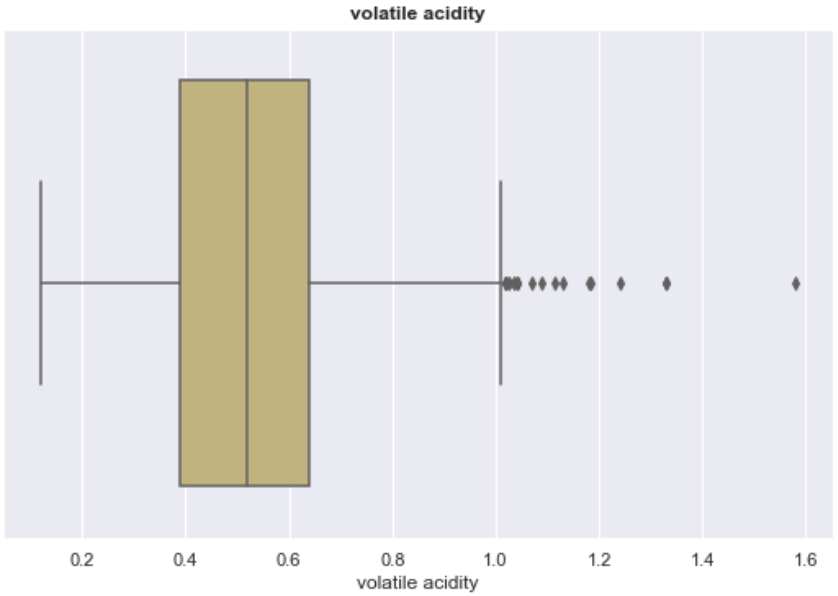

sulphates Vs quality
quality
3
4
5
6
7
8
Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
sulphates

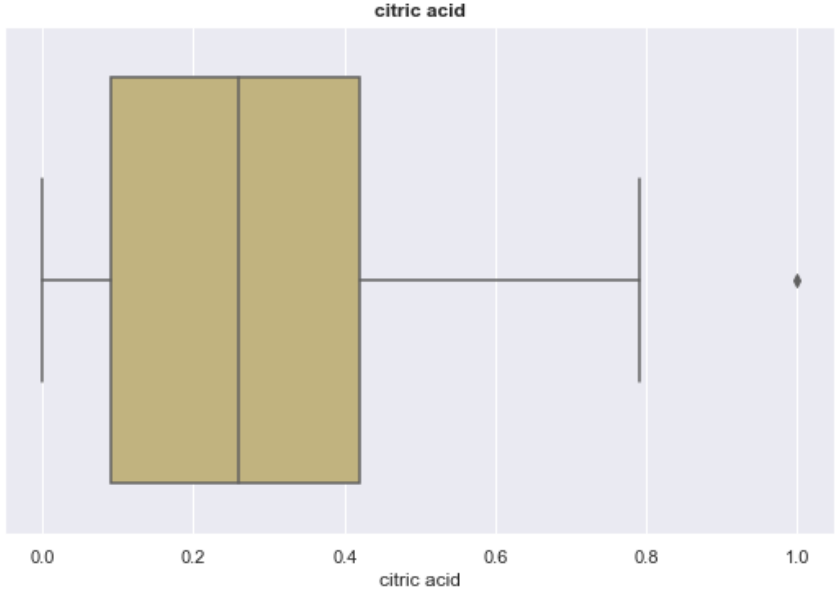

alcohol Vs quality
quality
0.35

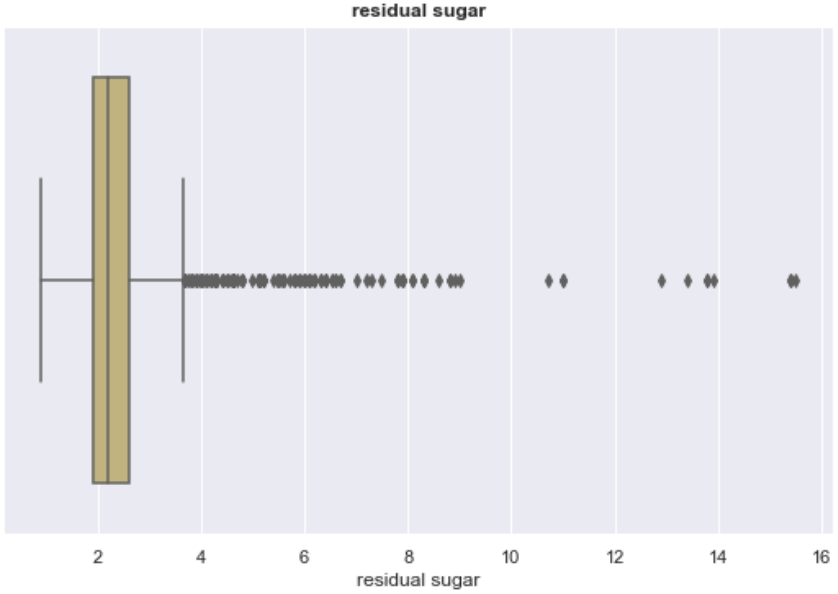


In [17]:
```
### Checking outliers in numerical features

plt.figure(figsize=(20,40))
for i in enumerate(continuous_features):
    plt.subplot(6, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(10,6)})
    sns.boxplot(data=dataset, x=i[1], color='y')
    plt.title("{}".format(i[1]), fontweight="bold")
```


fixed acidity
6
8
10
12
14
16
fixed acidity

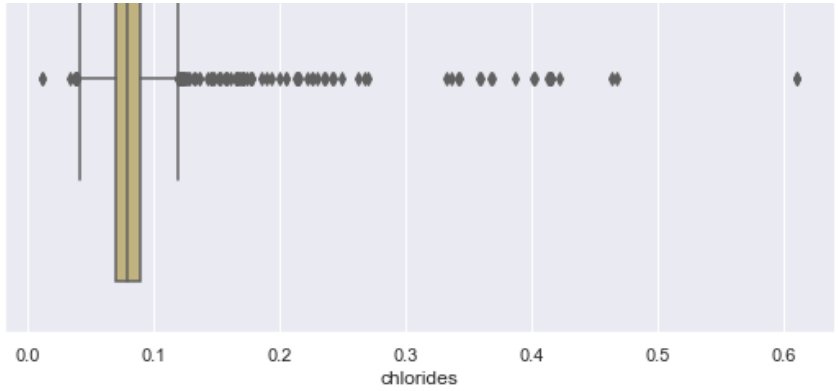

volatile acidity
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
volatile acidity

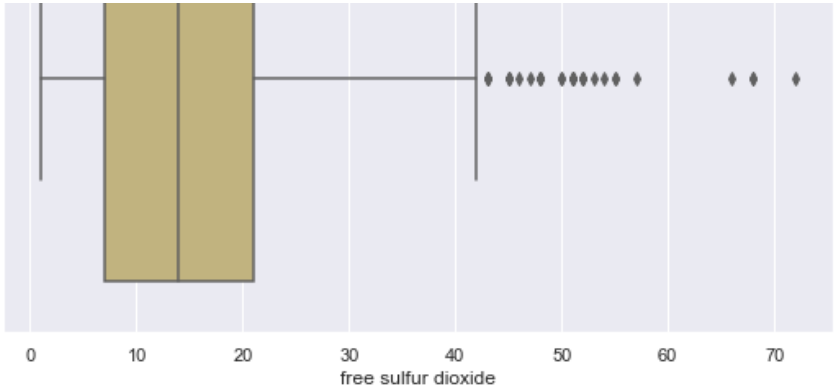

citric acid
0.0
0.2
0.4
0.6
0.8
1.0
citric acid

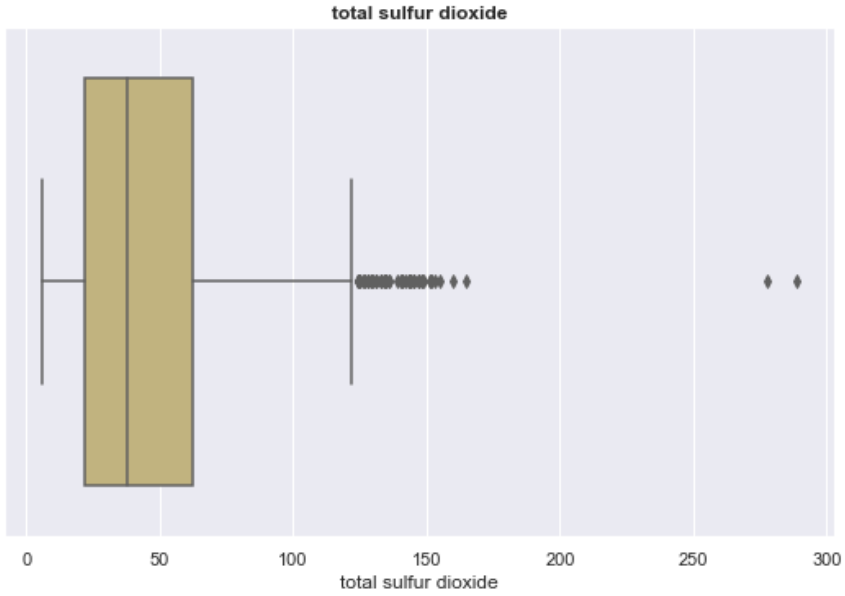

residual sugar
2
4
6
8
10
12
14
16
residual sugar

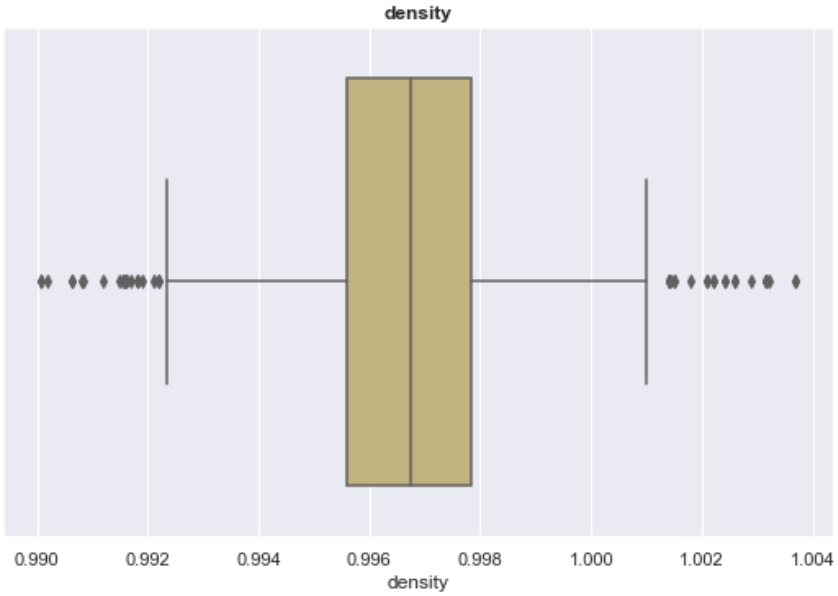

chlorides

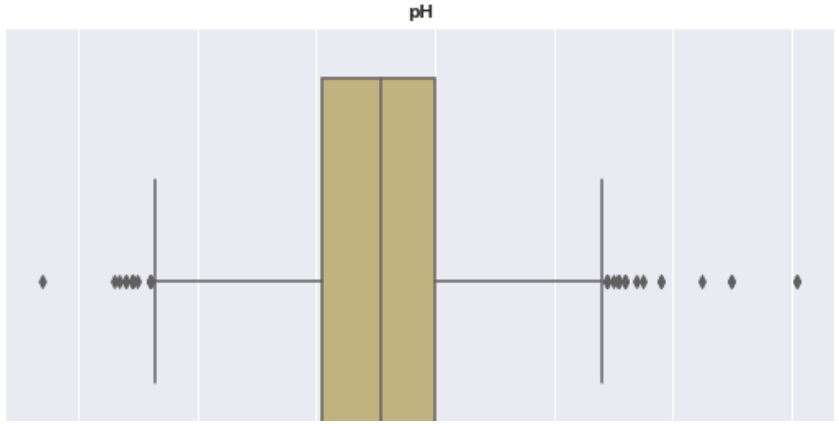

free sulfur dioxide

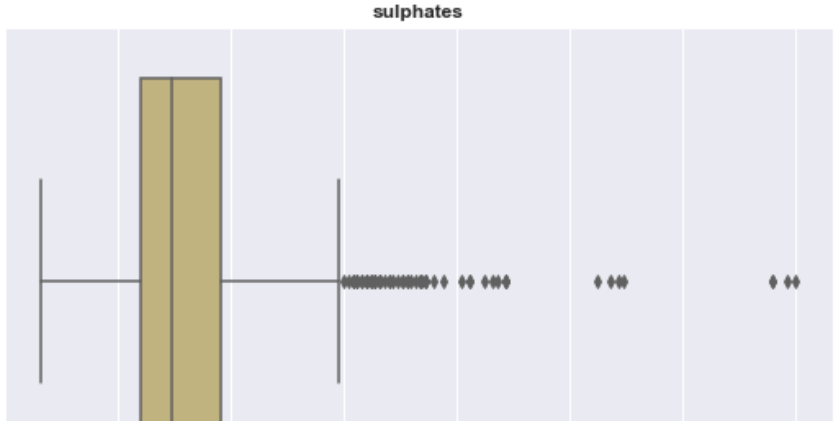

0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

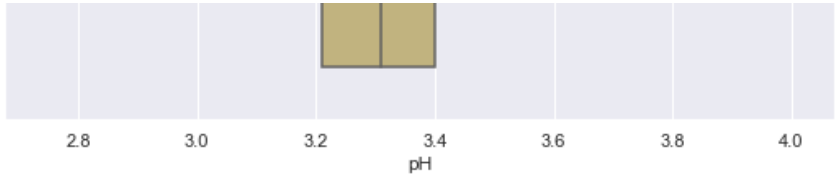

0
10
20
30
40
50
60
70
free sulfur dioxide

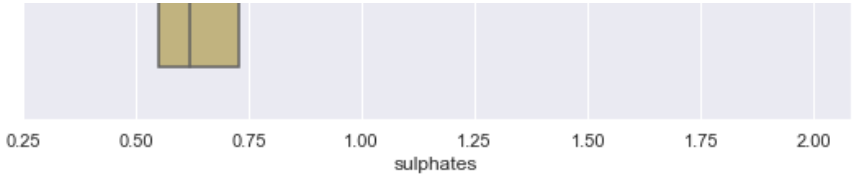

total sulfur dioxide
0
50
100
150
200
250
300
total sulfur dioxide

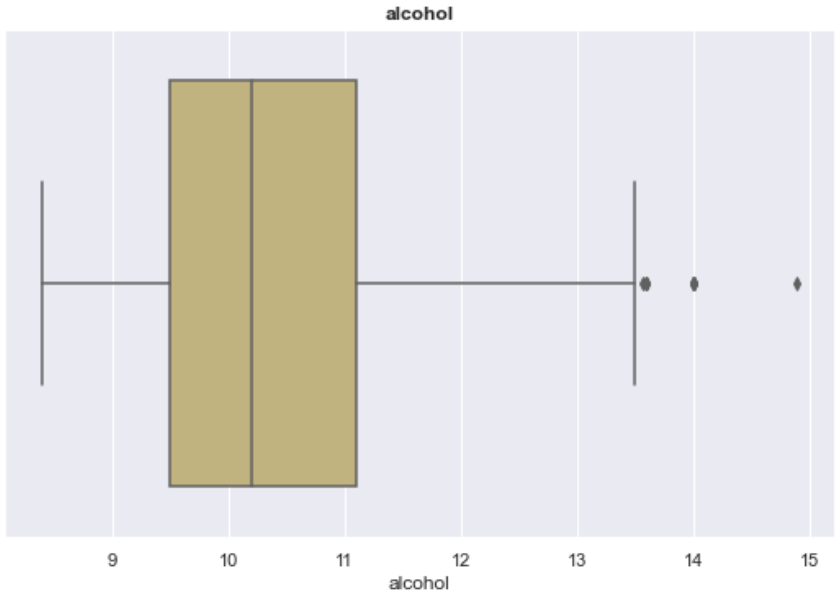

density
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004
density


pH

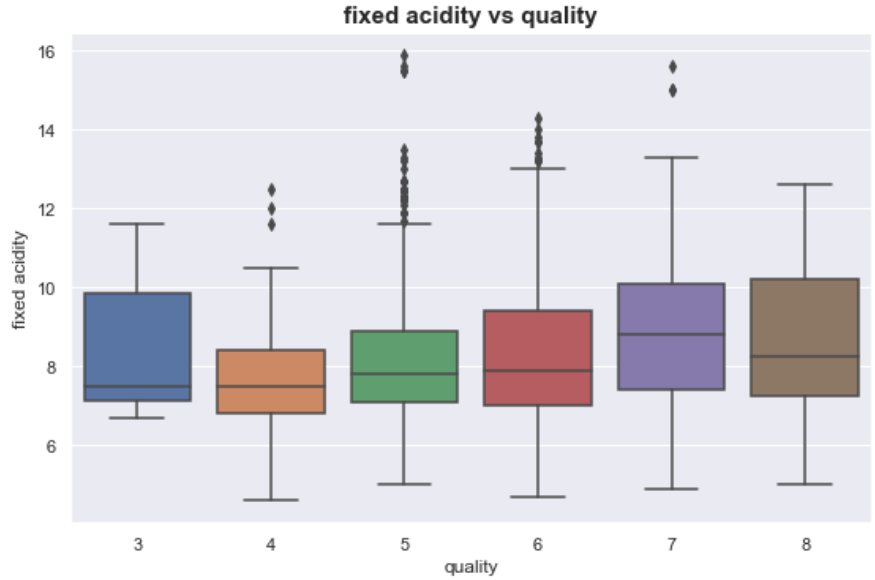

sulphates

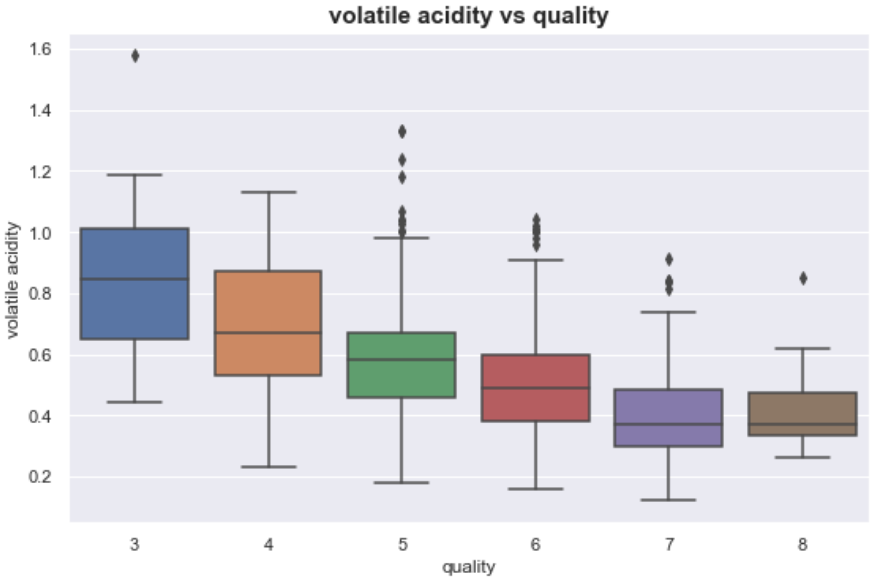


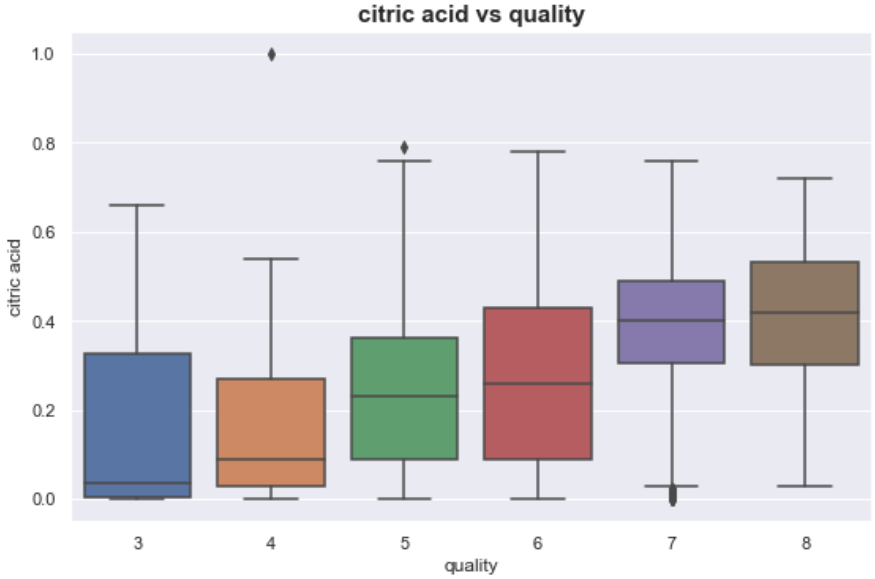


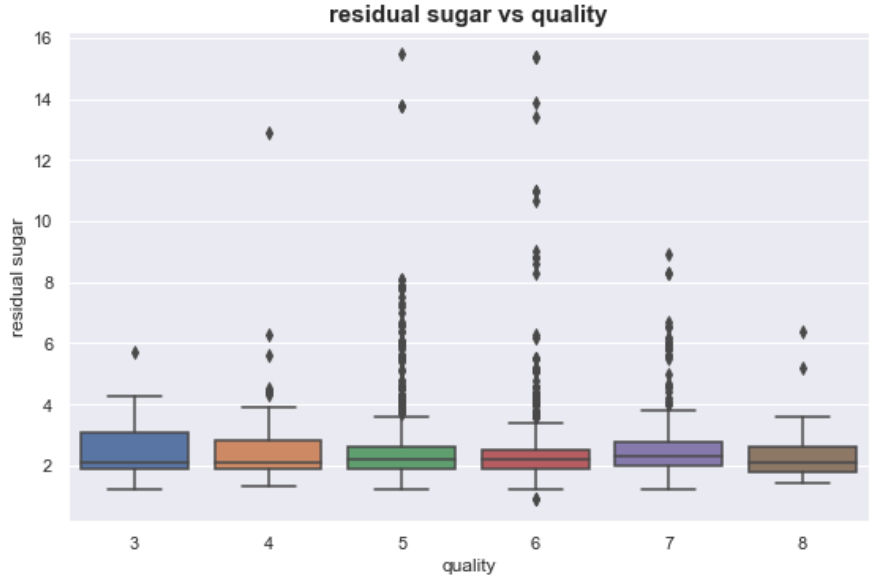


In [20]:
```
### getting outliers in features for each unique value in quality feature
plt.figure(figsize=(20,40))
for i in enumerate(continuous_features):
    plt.subplot(6, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(10,6)})
    sns.boxplot(data=dataset, y=i[1], x='quality')
    plt.title("{} vs quality".format(i[1]),fontsize=15, fontweight="bold")
```

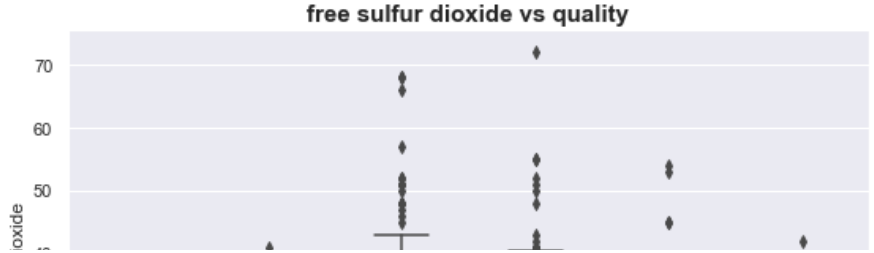

fixed acidity vs quality
fixed acidity
quality
3
4
5
6
7
8

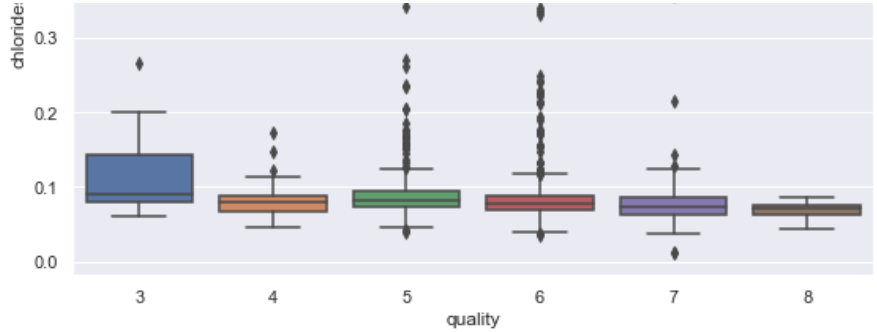

volatile acidity vs quality
volatile acidity
quality
3
4
5
6
7
8

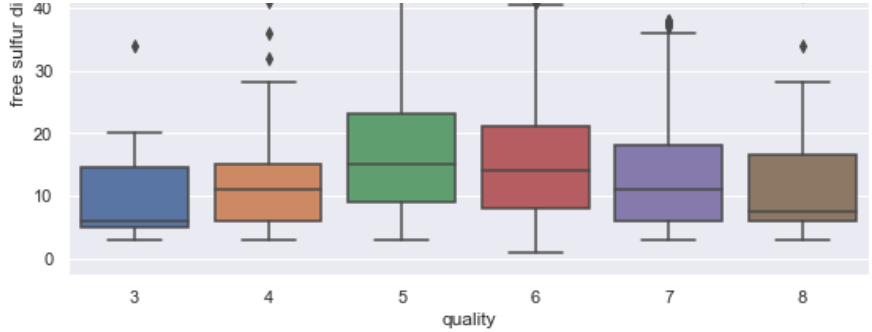

citric acid vs quality
citric acid
quality
3
4
5
6
7
8

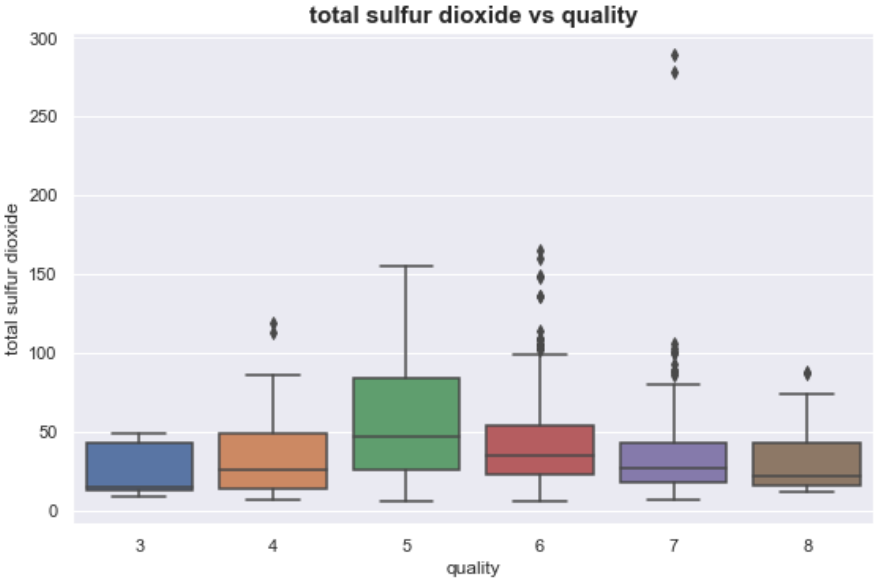

residual sugar vs quality
residual sugar
quality
3
4
5
6
7
8

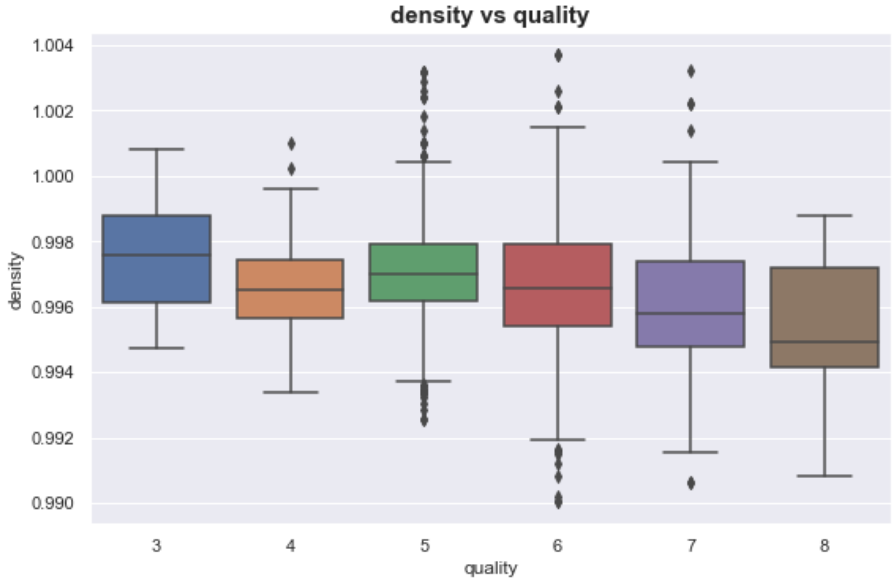

chlorides vs quality

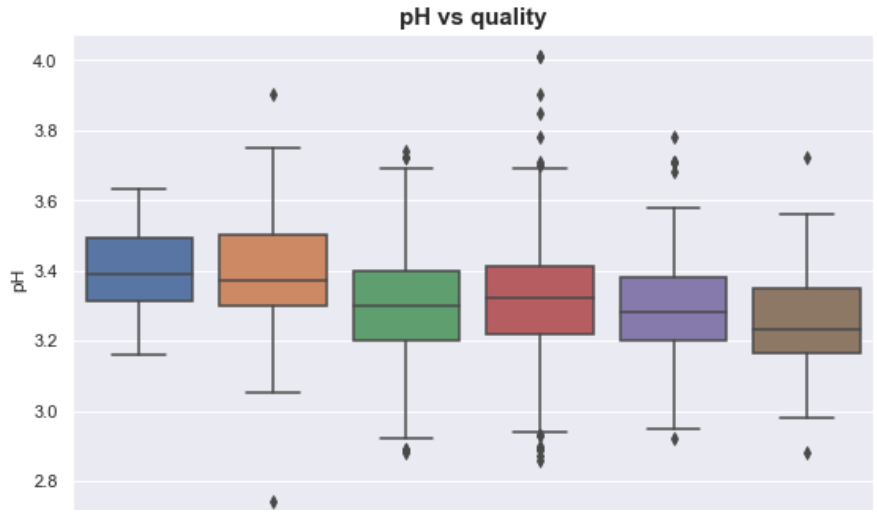

free sulfur dioxide vs quality

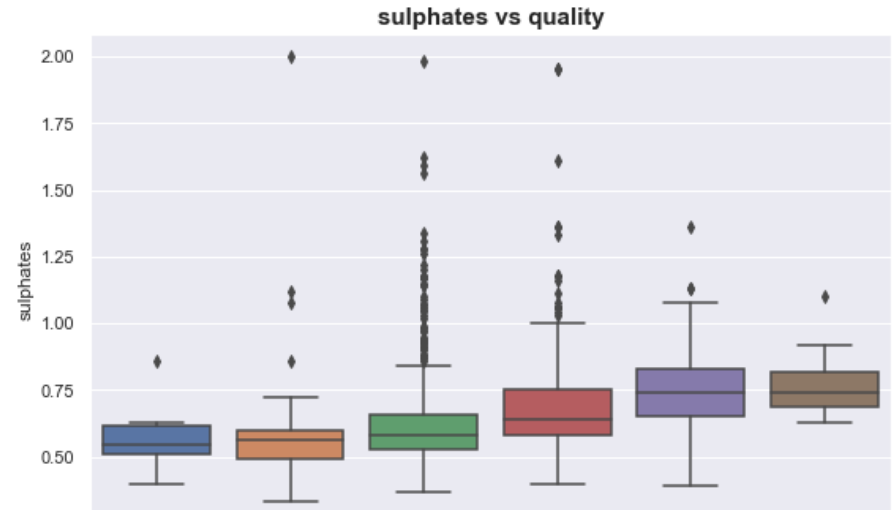

chloride
0.3
0.2
0.1
0.0
3
4
5
6
7
8
quality

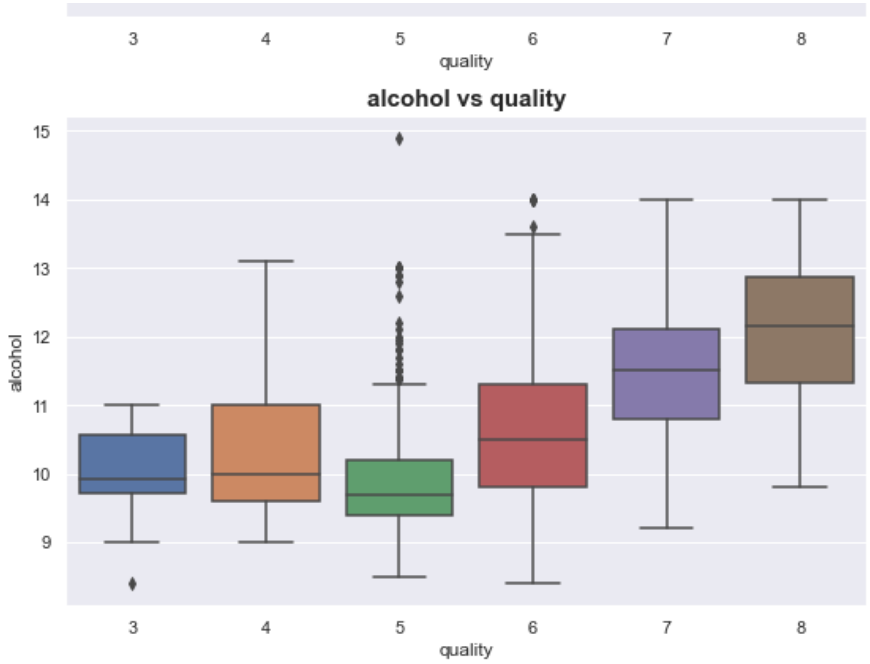

free sulfur di
40
30
20
10
0
3
4
5
6
7
8
quality

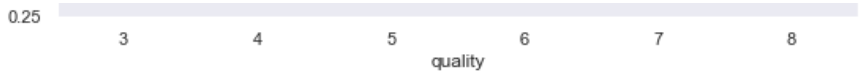

total sulfur dioxide vs quality
total sulfur dioxide
300
250
200
150
100
50
0
3
4
5
6
7
8
quality


density vs quality
density
1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.990
3
4
5
6
7
8
quality


pH vs quality
pH
4.0
3.8
3.6
3.4
3.2
3.0
2.8

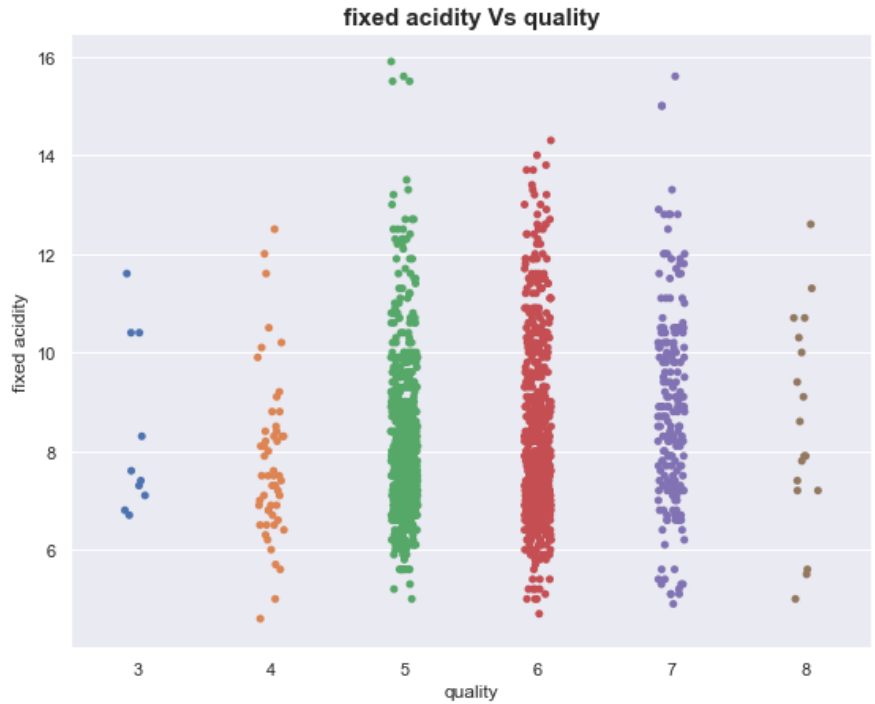

sulphates vs quality
sulphates
2.00
1.75
1.50
1.25
1.00
0.75
0.50

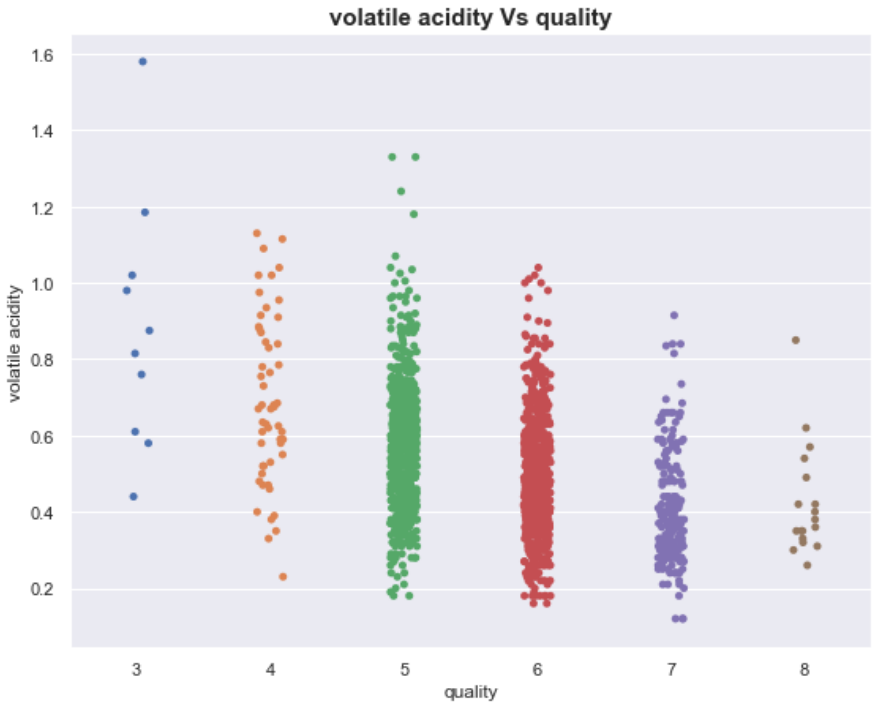


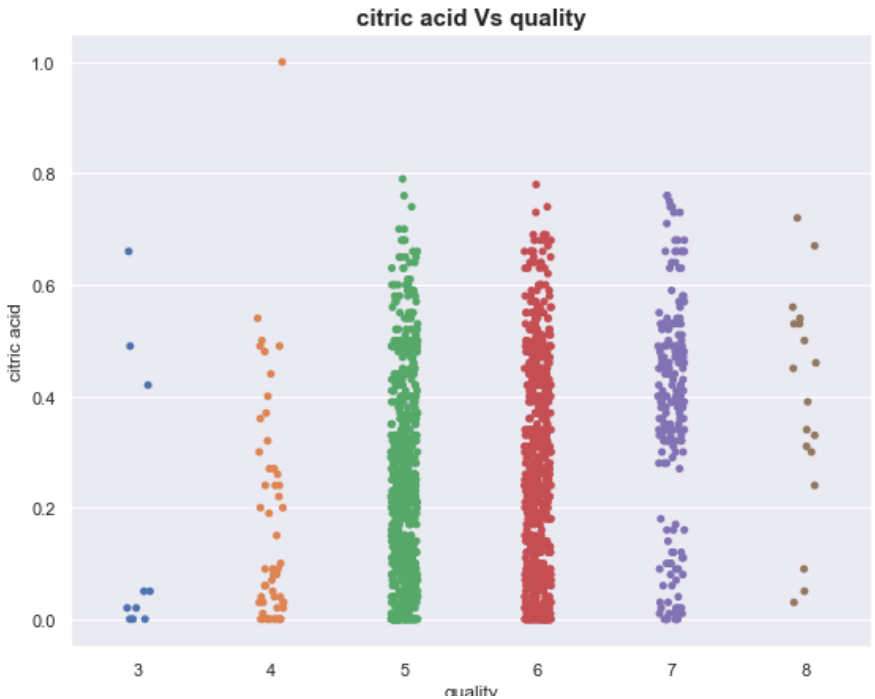


In [139…

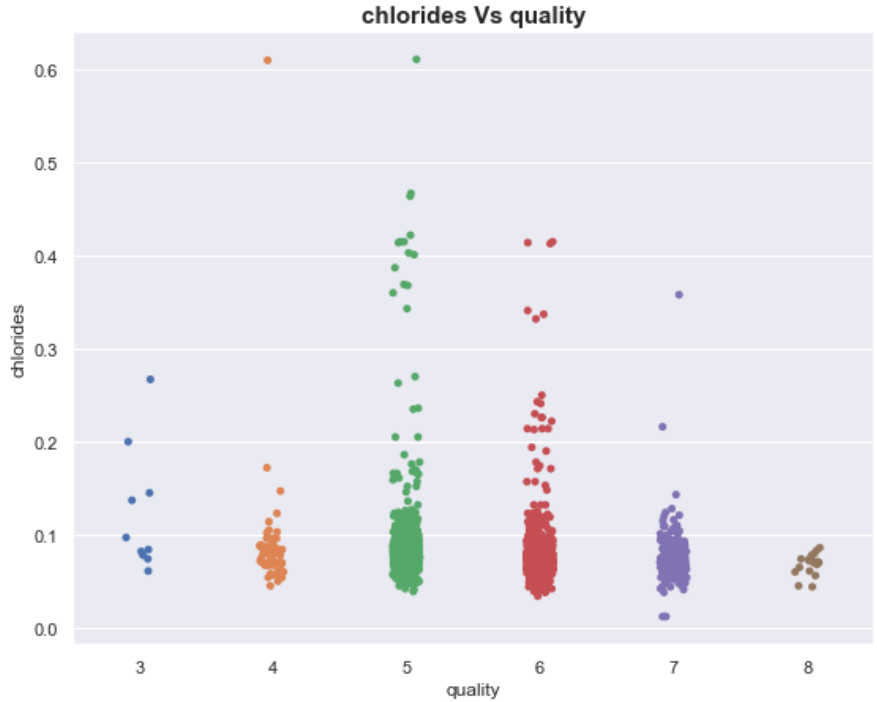
```
### visualising data scatter in each continuous feature with respect to quality
plt.figure(figsize=(20,50))
for i in enumerate(continuous_features):
    plt.subplot(6, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(7,8)})
    sns.stripplot(data=dataset, y=i[1], x='quality')
    plt.title("{} Vs quality".format(i[1]),fontsize=15, fontweight="bold")
```

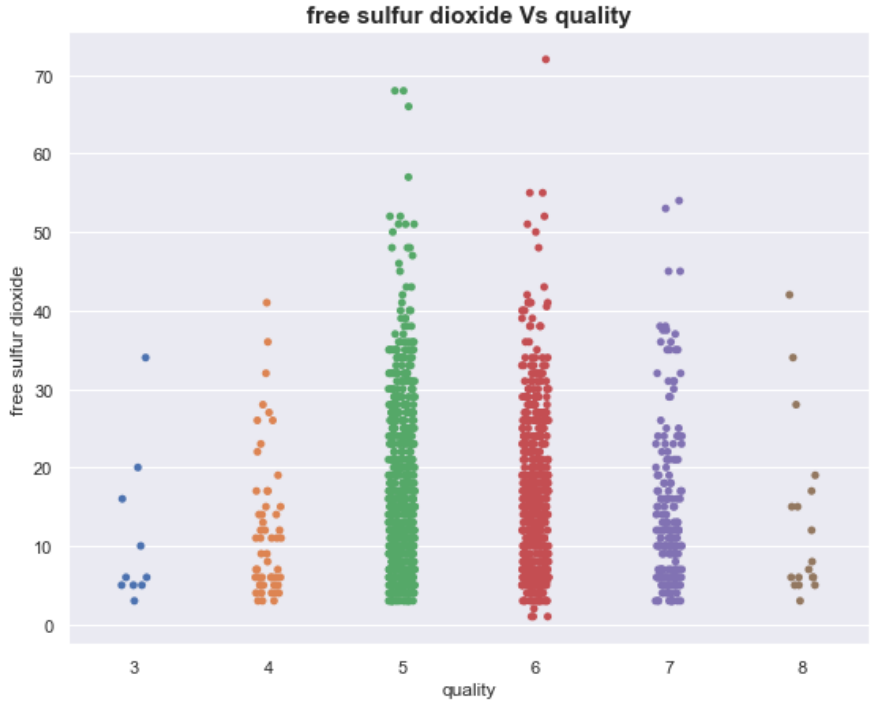

fixed acidity Vs quality
fixed acidity
quality

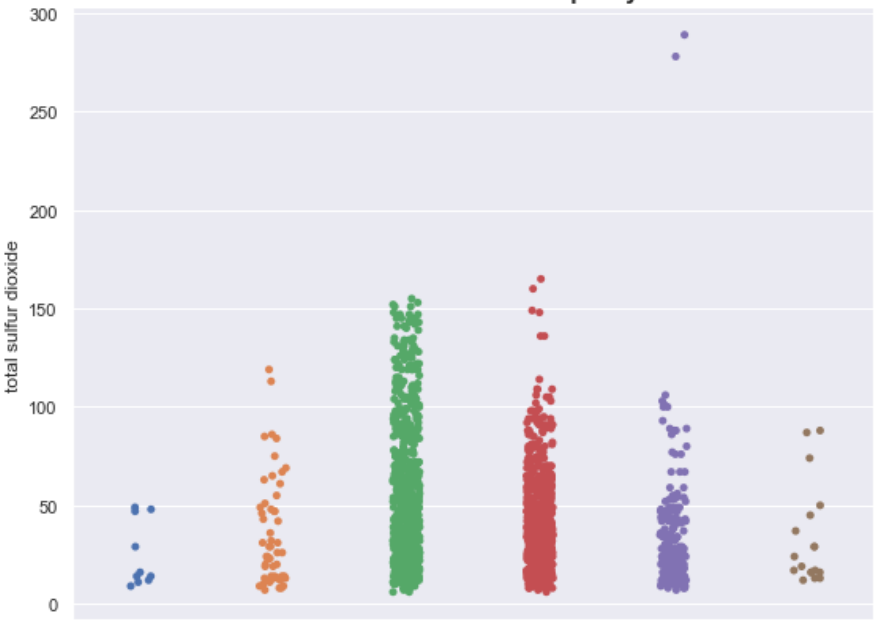

volatile acidity Vs quality
volatile acidity
quality

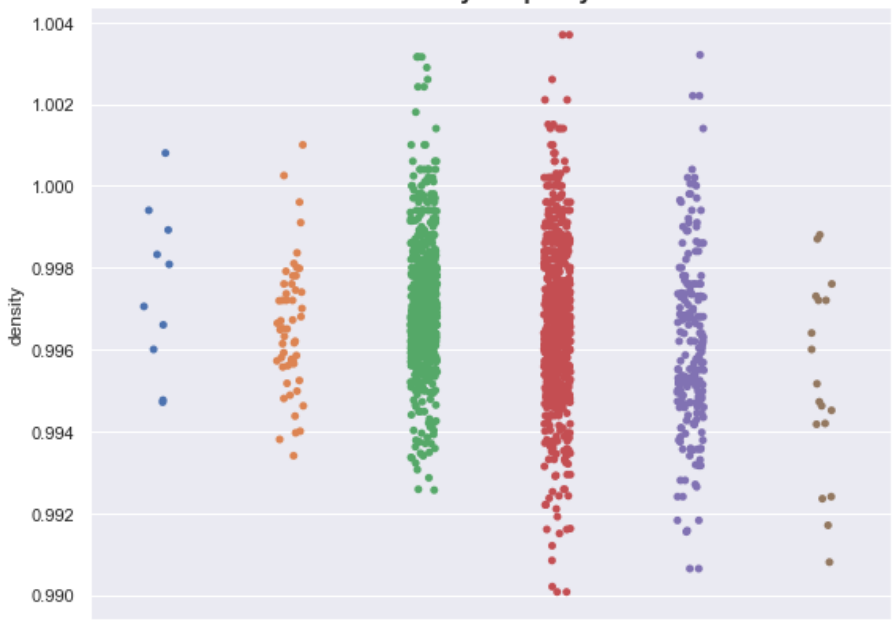

citric acid Vs quality
citric acid
quality

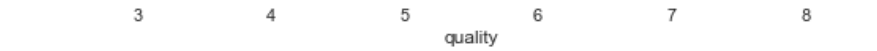

residual sugar Vs quality
residual sugar
quality

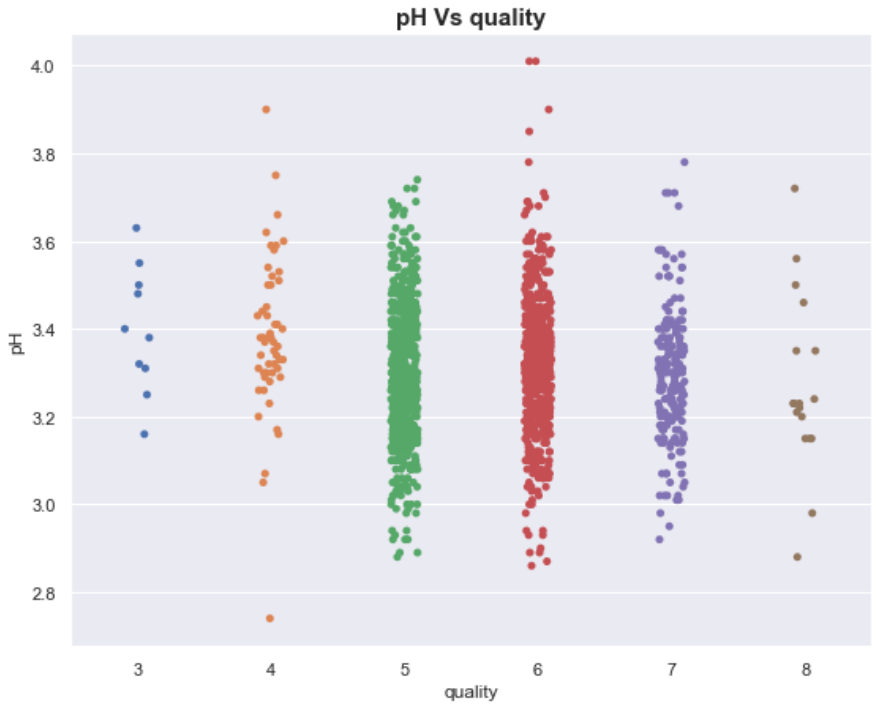

chlorides Vs quality
chlorides
quality
0.0
0.1
0.2
0.3
0.4
0.5
0.6
3
4
5
6
7
8


free sulfur dioxide Vs quality
free sulfur dioxide
quality
0
10
20
30
40
50
60
70
3
4
5
6
7
8

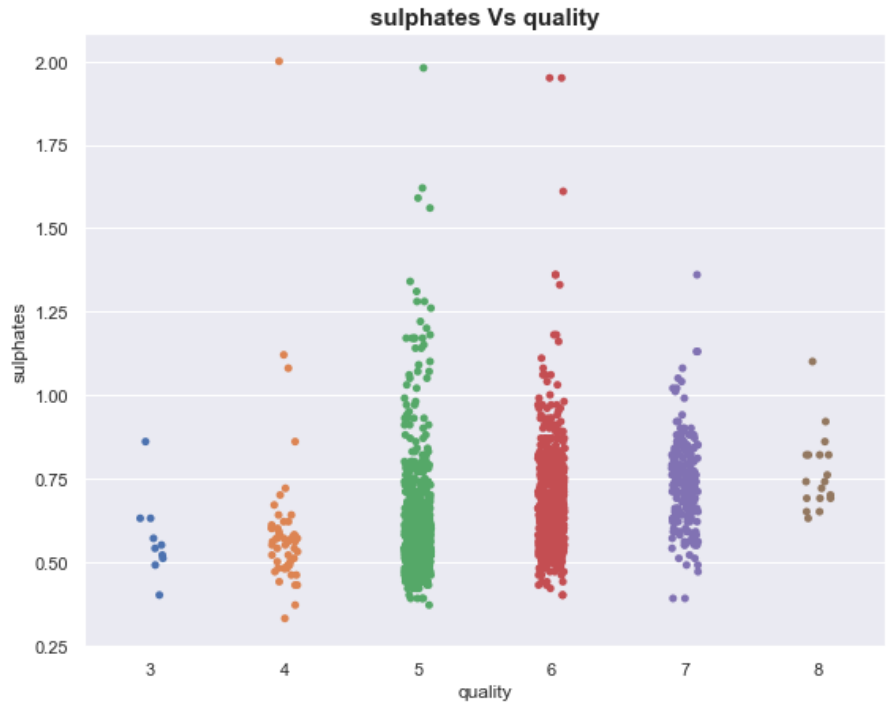

total sulfur dioxide Vs quality
total sulfur dioxide
0
50
100
150
200
250
300

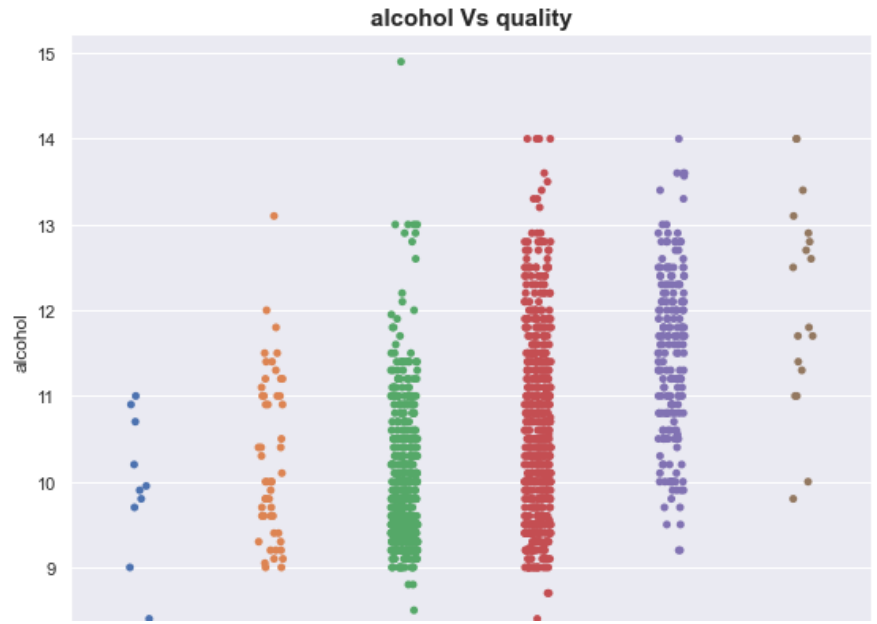

density Vs quality
density
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004


3
4
5
6
7
8
quality


pH Vs quality
4.0
3.8
3.6
3.4
3.2
3.0
2.8
pH
3
4
5
6
7
8
quality


3
4
5
6
7
8
quality


sulphates Vs quality
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
sulphates
3
4
5
6
7
8
quality


alcohol Vs quality
15
14
13
12
11
10
9
alcohol

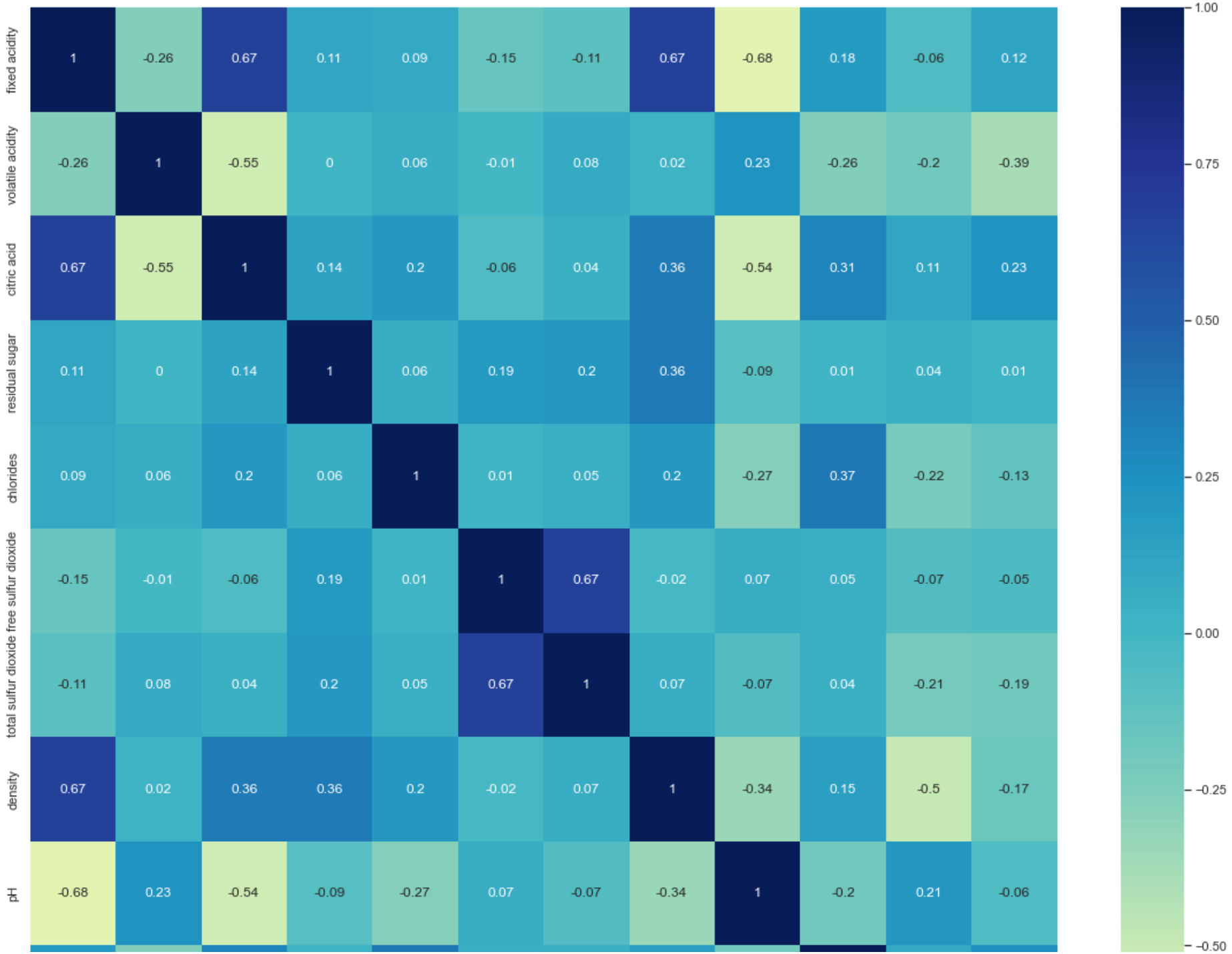
In [140…
```
### getting correlation for all the features
corr=round(dataset.corr(),2)
corr
```

Out[140]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.00	-0.26	0.67	0.11	0.09	-0.15	-0.11	0.67	-0.68	0.18	-0.06	0.12
volatile acidity	-0.26	1.00	-0.55	0.00	0.06	-0.01	0.08	0.02	0.23	-0.26	-0.20	-0.39
citric acid	0.67	-0.55	1.00	0.14	0.20	-0.06	0.04	0.36	-0.54	0.31	0.11	0.23
residual sugar	0.11	0.00	0.14	1.00	0.06	0.19	0.20	0.36	-0.09	0.01	0.04	0.01
chlorides	0.09	0.06	0.20	0.06	1.00	0.01	0.05	0.20	-0.27	0.37	-0.22	-0.13
free sulfur dioxide	-0.15	-0.01	-0.06	0.19	0.01	1.00	0.67	-0.02	0.07	0.05	-0.07	-0.05
total sulfur dioxide	-0.11	0.08	0.04	0.20	0.05	0.67	1.00	0.07	-0.07	0.04	-0.21	-0.19
density	0.67	0.02	0.36	0.36	0.20	-0.02	0.07	1.00	-0.34	0.15	-0.50	-0.17
pH	-0.68	0.23	-0.54	-0.09	-0.27	0.07	-0.07	-0.34	1.00	-0.20	0.21	-0.06
sulphates	0.18	-0.26	0.31	0.01	0.37	0.05	0.04	0.15	-0.20	1.00	0.09	0.25
alcohol	-0.06	-0.20	0.11	0.04	-0.22	-0.07	-0.21	-0.50	0.21	0.09	1.00	0.48
quality	0.12	-0.39	0.23	0.01	-0.13	-0.05	-0.19	-0.17	-0.06	0.25	0.48	1.00

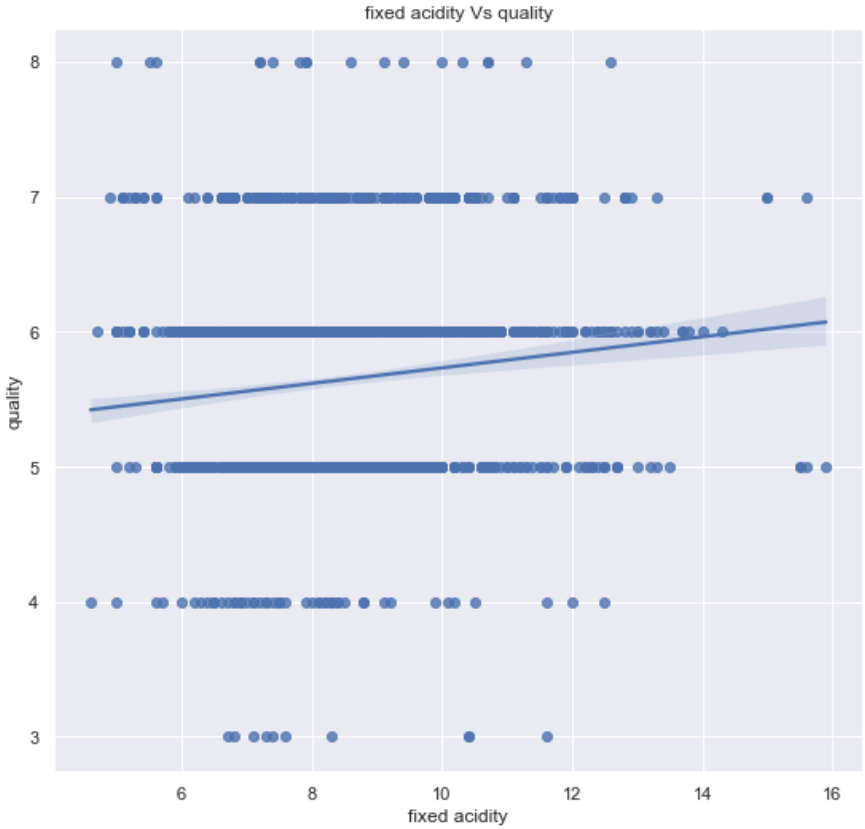
In [141…
```
### Plotting heatmap for visualising the correlation between features
sns.set(rc={'figure.figsize':(20,20)})
sns.heatmap(data=corr, annot=True,  vmin=-1, vmax=1, cmap="YlGnBu")
```

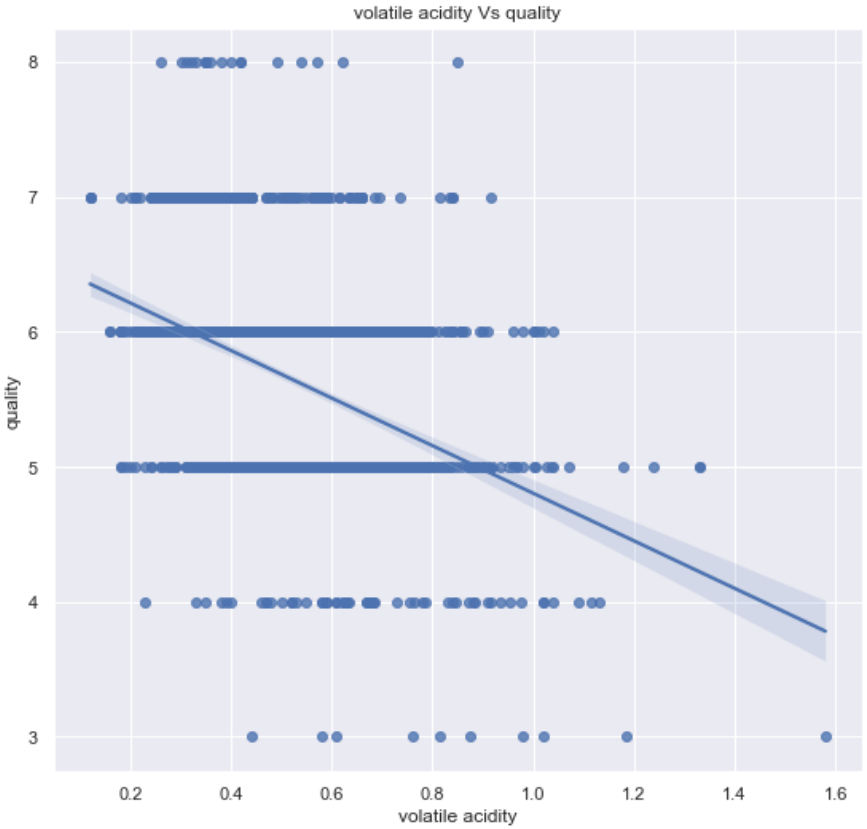
Out[141]:
```
<AxesSubplot:>
```

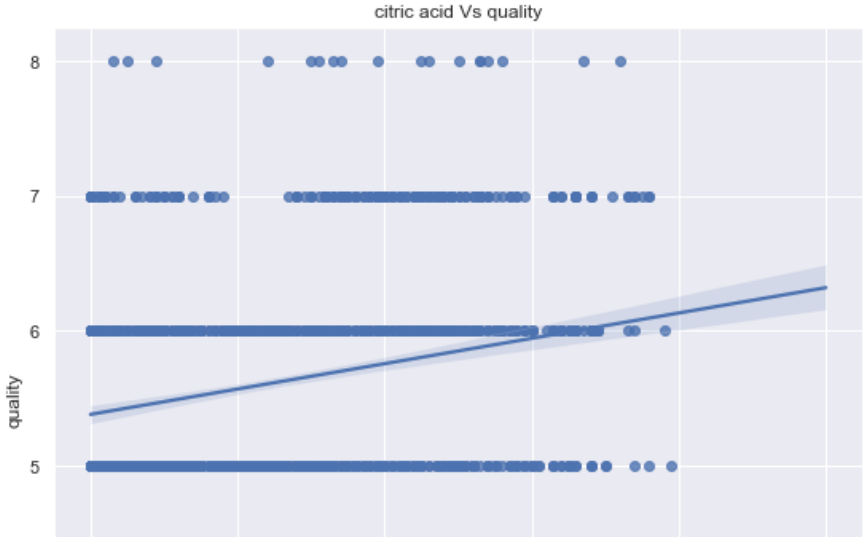

fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
1 -0.26 0.67 0.11 0.09 -0.15 -0.11 0.67 -0.68 0.18 -0.06 0.12
-0.26 1 -0.55 0 0.06 -0.01 0.08 0.02 0.23 -0.26 -0.2 -0.39
0.67 -0.55 1 0.14 0.2 -0.06 0.04 0.36 -0.54 0.31 0.11 0.23
0.11 0 0.14 1 0.06 0.19 0.2 0.36 -0.09 0.01 0.04 0.01
0.09 0.06 0.2 0.06 1 0.01 0.05 0.2 -0.27 0.37 -0.22 -0.13
-0.15 -0.01 -0.06 0.19 0.01 1 0.67 -0.02 0.07 0.05 -0.07 -0.05
-0.11 0.08 0.04 0.2 0.05 0.67 1 0.07 -0.07 0.04 -0.21 -0.19
0.67 0.02 0.36 0.36 0.2 -0.02 0.07 1 -0.34 0.15 -0.5 -0.17
-0.68 0.23 -0.54 -0.09 -0.27 0.07 -0.07 -0.34 1 -0.2 0.21 -0.06
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50

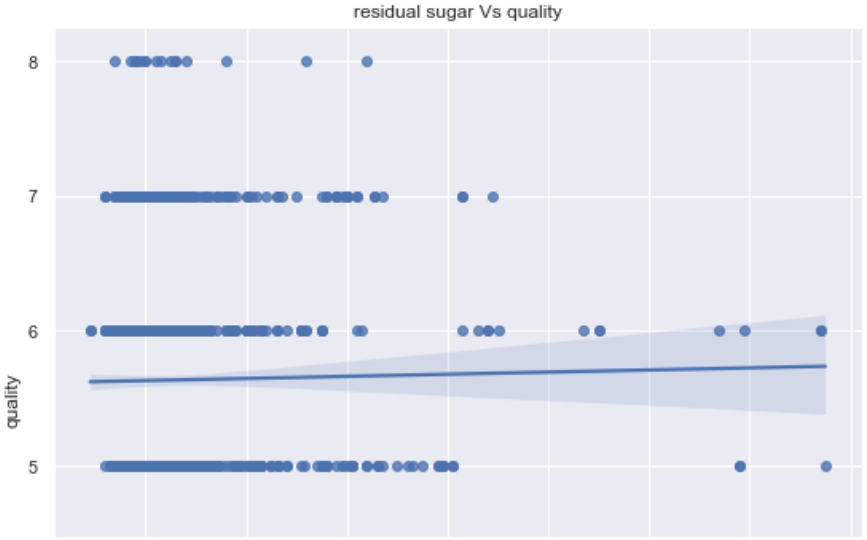
In [143…

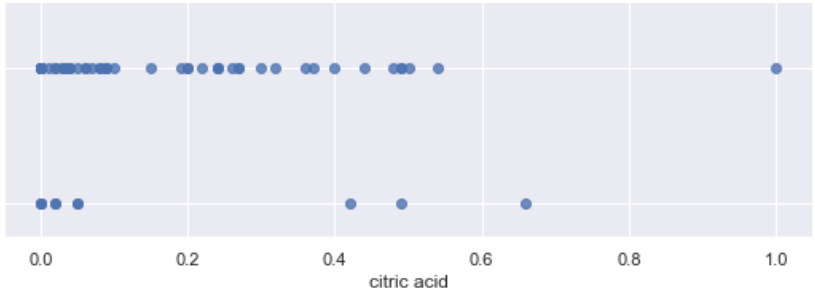
```
### plotting regplot for features vs quality
plt.figure(figsize=(20,60))
for i in enumerate(continuous_features):
    plt.subplot(6, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(8,10)})
    sns.regplot(data=dataset, x=i[1], y='quality')
    plt.xlabel(i[1])
    plt.ylabel("quality")
    plt.title("{} Vs quality".format(i[1]))
```

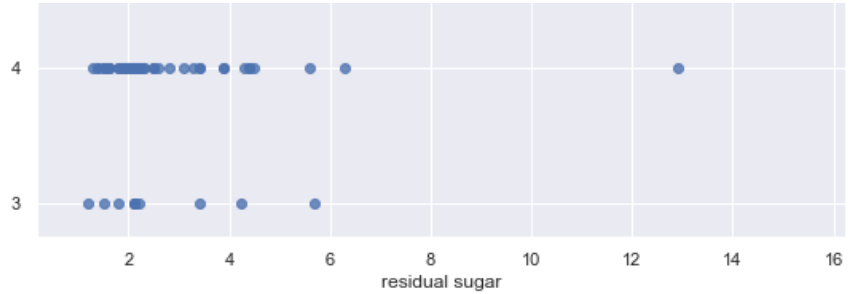

fixed acidity Vs quality
quality
fixed acidity
8
7
6
5
4
3
6
8
10
12
14
16

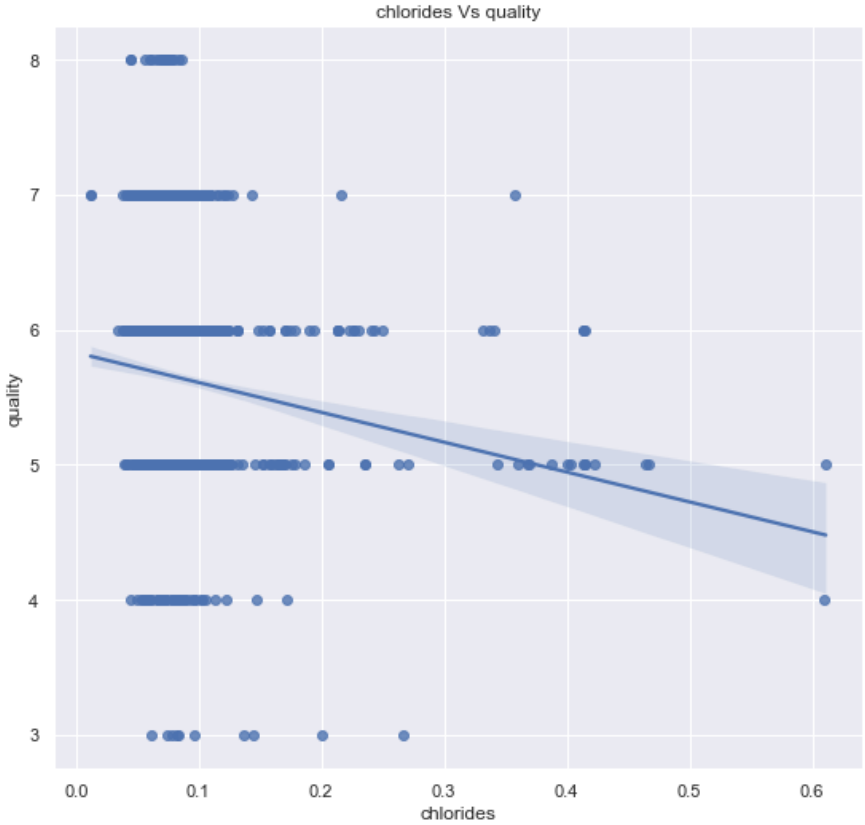

volatile acidity Vs quality
quality
volatile acidity
8
7
6
5
4
3
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

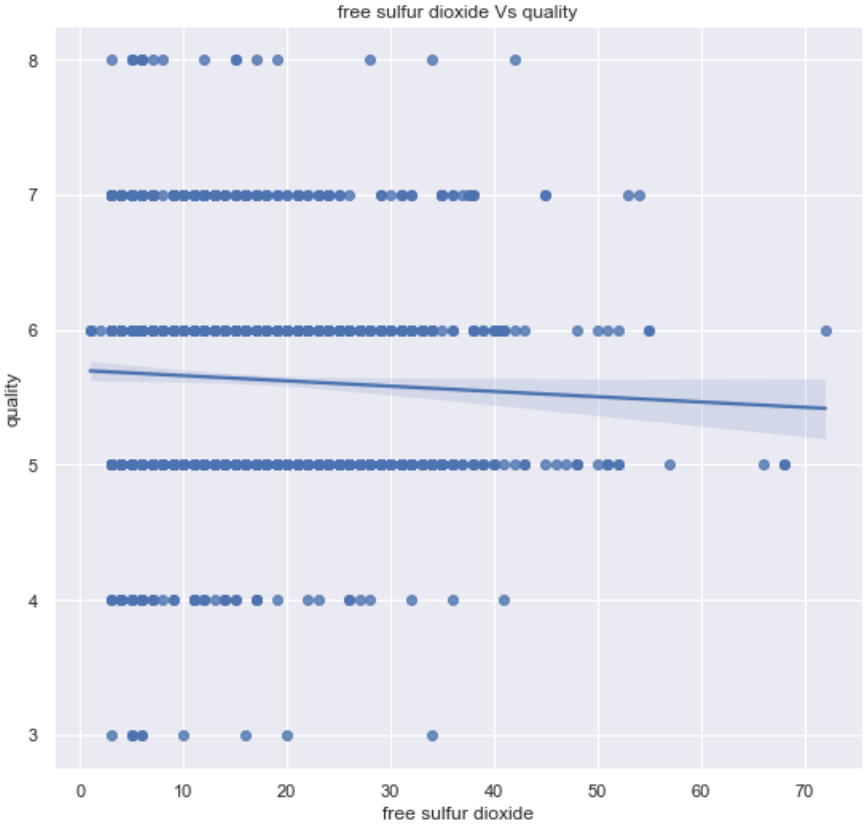

citric acid Vs quality
quality
8
7
6
5

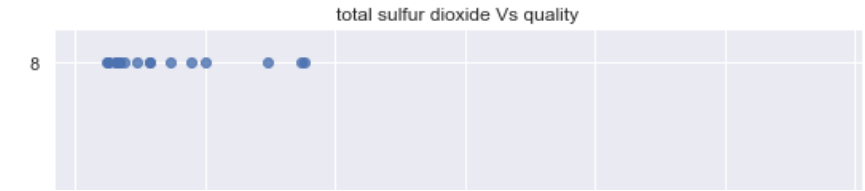

residual sugar Vs quality
quality
8
7
6
5

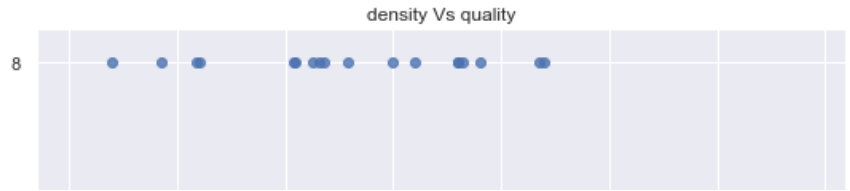

4
3
0.0
0.2
0.4
0.6
0.8
1.0
citric acid

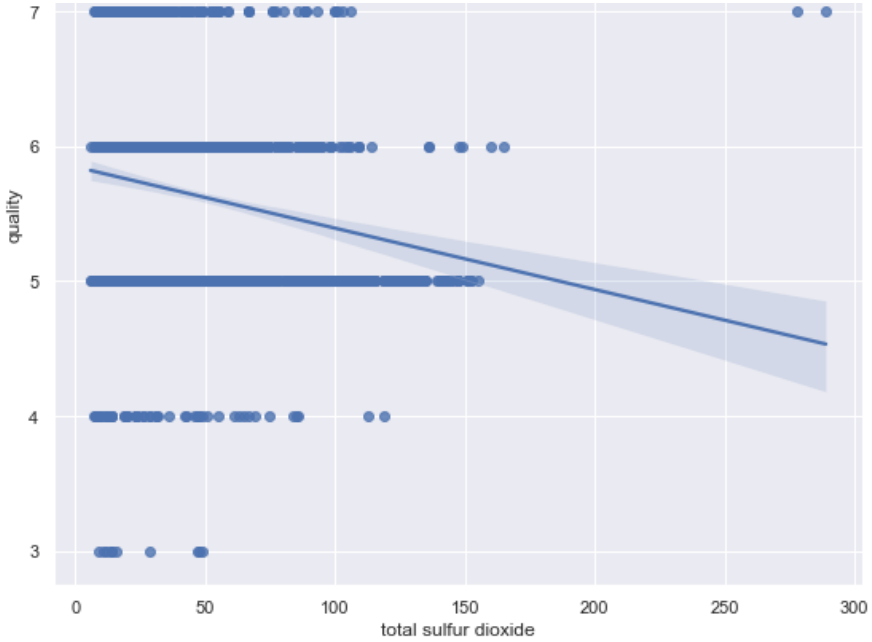

4
3
2
4
6
8
10
12
14
16
residual sugar

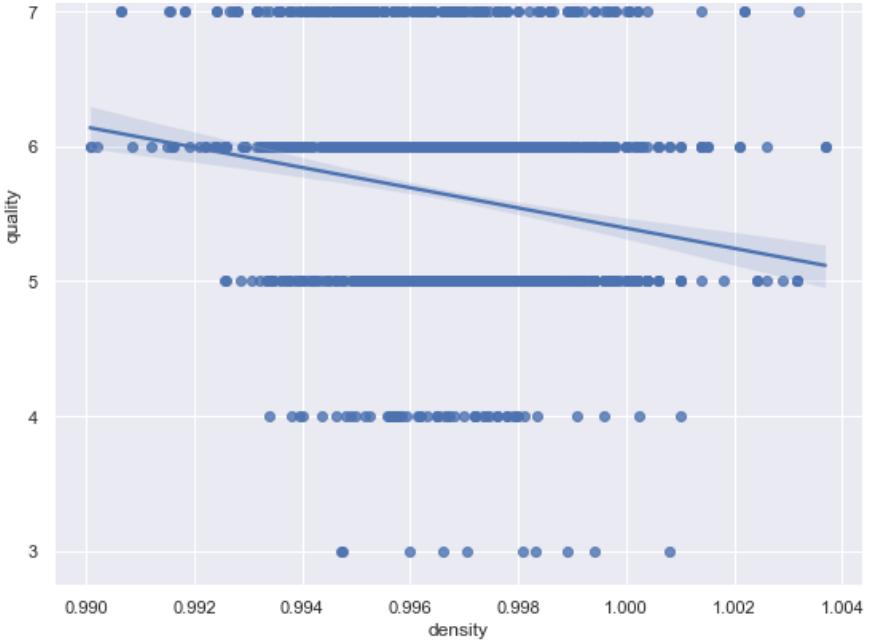

chlorides Vs quality
quality
8
7
6
5
4
3
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

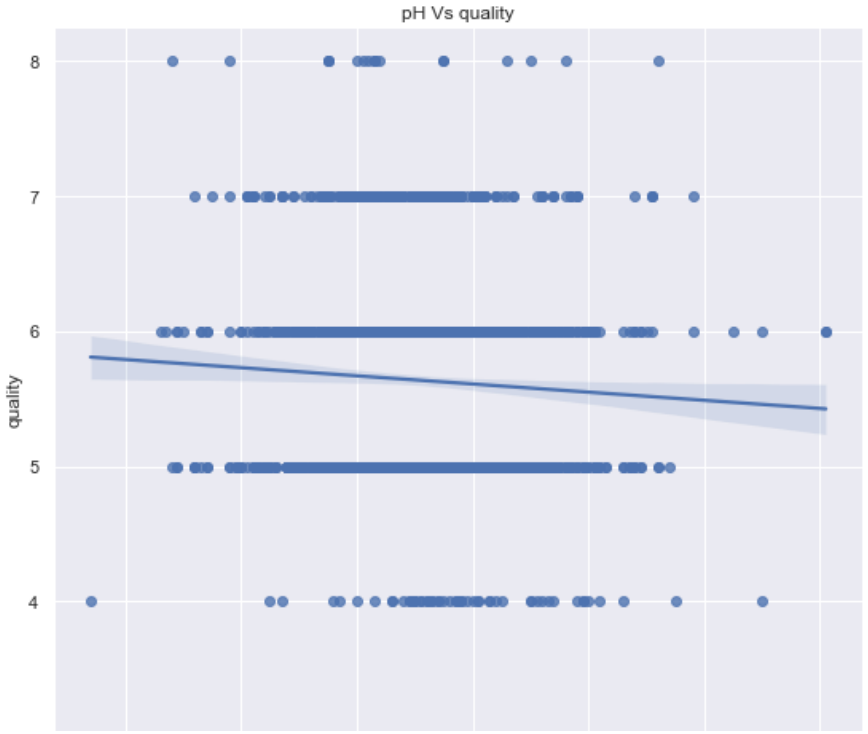

free sulfur dioxide Vs quality
quality
8
7
6
5
4
3
0
10
20
30
40
50
60
70
free sulfur dioxide

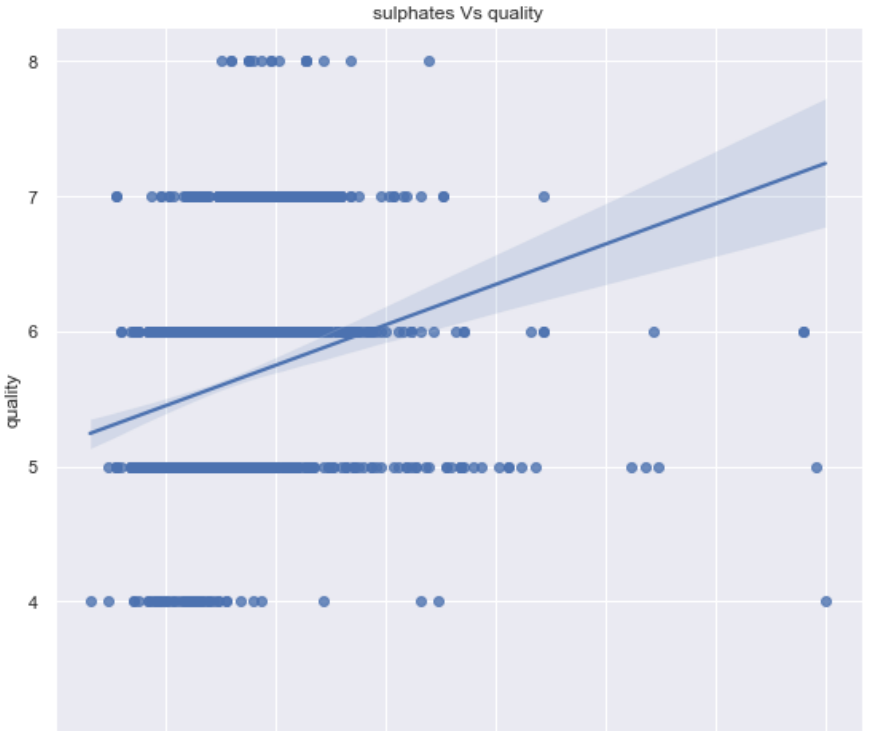

total sulfur dioxide Vs quality
8

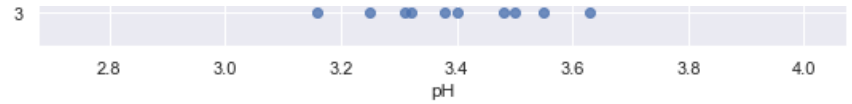

density Vs quality
8

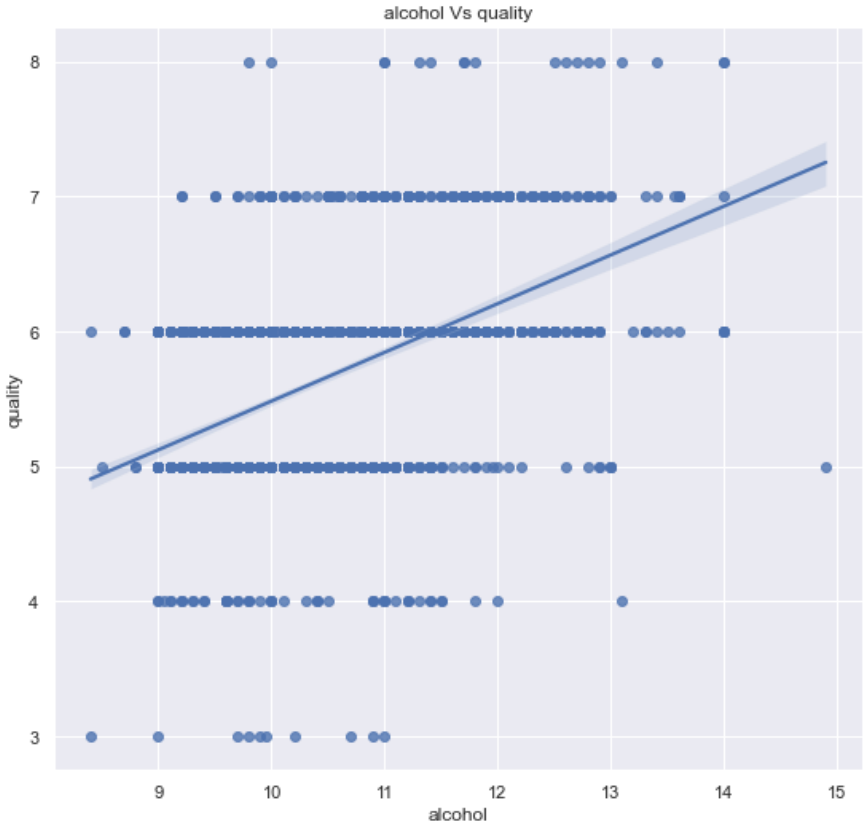

quality
total sulfur dioxide
0
50
100
150
200
250
300
3
4
5
6
7


quality
density
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004
3
4
5
6
7


pH Vs quality
quality
4
5
6
7
8


sulphates Vs quality
quality
4
5
6
7
8





2.3 Splitting data into independent and dependent features

In [144…

```
### splitting data into independent dataframe and dependent series
X=dataset.iloc[:,:-1]
y=dataset.iloc[:,-1]
X.head()
```

Out[144]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [145…
```
y.head()
```

Out[145]:
```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

In [146…
```
### random state train test split will be same with all people using random_state=10
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=10)
```

In [147…
```
X_train.head()
```

Out[147]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
739	9.0	0.69	0.00	2.4	0.088	19.0	38.0	0.99900	3.35	0.60	9.3
663	10.1	0.28	0.46	1.8	0.050	5.0	13.0	0.99740	3.04	0.79	10.2
981	9.5	0.86	0.26	1.9	0.079	13.0	28.0	0.99712	3.25	0.62	10.0
463	8.1	0.66	0.70	2.2	0.098	25.0	129.0	0.99720	3.08	0.53	9.0
78	6.7	0.75	0.12	2.0	0.086	12.0	80.0	0.99580	3.38	0.52	10.1

In [148…
```
y_train.head()
```

Out[148]:
```
739    5
663    6
981    5
463    5
78     5
Name: quality, dtype: int64
```

In [149…
```
X_test.head()
```

Out[149]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
1518	7.4	0.47	0.46	2.2	0.114	7.0	20.0	0.99647	3.32	0.63	10.5
1246	7.4	0.74	0.07	1.7	0.086	15.0	48.0	0.99502	3.12	0.48	10.0
544	14.3	0.31	0.74	1.8	0.075	6.0	15.0	1.00080	2.86	0.79	8.4
1343	7.5	0.51	0.02	1.7	0.084	13.0	31.0	0.99538	3.36	0.54	10.5
428	9.1	0.52	0.33	1.3	0.070	9.0	30.0	0.99780	3.24	0.60	9.3

In [150…
```
y_test.head()
```

Out[150]:
```
1518    5
1246    5
544     6
1343    6
428     5
Name: quality, dtype: int64
```

In [151…
```
### both will have same shape
X_train.shape, y_train.shape
```

Out[151]:
```
((1199, 11), (1199,))
```

In [152…
```
### both will have same shape
X_test.shape, y_test.shape
```

Out[152]:
```
((400, 11), (400,))
```

2.4 Transforming data

In [153…
```
scaler=StandardScaler()
scaler
```

Out[153]:
```
StandardScaler
StandardScaler()
```

In [154…
```
X_train=scaler.fit_transform(X_train)
X_train
```

Out[154]:
```
array([[ 0.36628211,  0.9252263 , -1.41559084, ...,  0.2800327 ,
        -0.34099149, -1.03579592],
       [ 0.99530488, -1.38052243,  0.95277345, ..., -1.75526948,
         0.71993818, -0.17938734],
       [ 0.65220155,  1.88126845, -0.07695015, ..., -0.37651639,
        -0.22931468, -0.36970036],
       ...,
       [-0.77739563, -0.8181447 ,  1.10723199, ...,  0.54265233,
         0.60826138,  0.96249075],
       [ 0.93812099, -0.98685802,  1.00425963, ..., -0.50782621,
        -0.78769872,  1.5334298 ],
       [-0.77739563,  0.41908633,  0.12899457, ..., -0.04824185,
         2.84179753, -0.17938734]])
```

In [155…
```
X_test=scaler.transform(X_test)
X_test
```

Out[155]:
```
array([[-0.54866008, -0.31200473,  0.95277345, ...,  0.08306797,
        -0.17347628,  0.10608218],
       [-0.54866008,  1.20641516, -1.05518758, ..., -1.23003021,
        -1.01105233, -0.36970036],
       [ 3.39702815, -1.21180911,  2.3943865 , ..., -2.93705783,
         0.71993818, -1.89220449],
       ...,
       [-0.4914762 , -0.53695583,  0.12899457, ...,  0.87092688,
         0.32906936,  1.05764726],
       [-0.26274065, -0.14329141,  0.28345311, ..., -0.70479094,
        -0.6201835 , -0.8454829 ],
       [-0.37710842, -1.49299798, -0.07695015, ..., -1.03306548,
         0.71993818,  0.48670821]])
```

2.5 Building SVC Model, training and performance of Model

In [156…
```
svc=SVC()
svc
```

Out[156]:
SVC
SVC()

In [157…
```
svc.fit(X_train,y_train)
```

Out[157]:
SVC
SVC()

In [158…
```
svc_pred=svc.predict(X_test)
svc_pred
```

Out[158]:
```
array([6, 5, 6, 6, 5, 7, 6, 6, 6, 5, 6, 5, 5, 5, 5, 5, 5, 6, 5, 5, 5, 6,
       6, 5, 5, 6, 5, 6, 6, 5, 5, 5, 6, 6, 6, 5, 7, 5, 6, 5, 6, 6, 5, 6,
       5, 6, 6, 6, 6, 5, 6, 7, 6, 6, 6, 6, 6, 5, 5, 5, 6, 5, 6, 6, 5, 5,
       6, 6, 5, 6, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 7, 5, 6, 5, 6, 6, 5, 5,
       5, 5, 7, 6, 7, 5, 5, 5, 5, 5, 6, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6,
       5, 7, 6, 5, 5, 5, 5, 5, 6, 6, 5, 5, 6, 5, 5, 5, 6, 5, 5, 5, 6, 5,
       5, 6, 5, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6, 6, 5, 5, 6, 5, 6, 6, 6,
       6, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 6, 6, 5, 7, 6, 6, 5, 5,
       6, 6, 5, 6, 6, 6, 6, 6, 5, 5, 6, 7, 5, 5, 5, 5, 6, 6, 6, 6, 5, 6,
       7, 6, 6, 6, 6, 5, 5, 5, 5, 5, 6, 7, 5, 6, 6, 5, 6, 6, 6, 5, 5, 5,
       6, 6, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 5, 6,
       6, 7, 6, 6, 5, 5, 5, 6, 5, 6, 5, 5, 6, 5, 7, 7, 7, 6, 5, 5, 6, 5,
       5, 5, 5, 6, 6, 6, 5, 7, 6, 6, 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 5,
       6, 6, 5, 6, 6, 5, 6, 6, 5, 7, 5, 6, 5, 6, 6, 6, 6, 6, 6, 7, 5, 6,
       6, 5, 6, 5, 6, 5, 6, 6, 7, 7, 5, 6, 6, 6, 6, 7, 5, 6, 6, 5, 5, 5,
       6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 6, 5, 5, 6, 5, 5,
       7, 5, 5, 5, 6, 6, 6, 5, 5, 6, 6, 5, 5, 5, 6, 6, 5, 6, 5, 5, 5, 6,
       6, 7, 6, 5, 5, 7, 6, 5, 6, 5, 5, 6, 5, 6, 6, 5, 6, 6, 7, 6, 5, 5,
       5, 6, 5, 6], dtype=int64)
```

In [159…
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, classification_report
```

In [160…
```
confusion_mat=confusion_matrix(y_test, svc_pred)
confusion_mat
```

Out[160]:
```
array([[  0,   0,   1,   1,   0,   0],
       [  0,   0,  12,   3,   0,   0],
       [  0,   0, 124,  47,   1,   0],
       [  0,   0,  41, 105,   6,   0],
       [  0,   0,   2,  31,  17,   0],
       [  0,   0,   0,   6,   3,   0]], dtype=int64)
```

In [161…
```
acc=accuracy_score(y_test, svc_pred)
acc
```

Out[161]:
```
0.615
```

In [162…
```
report=classification_report(y_test, svc_pred, zero_division=False)
print(report)
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        15
           5       0.69      0.72      0.70       172
           6       0.54      0.69      0.61       152
           7       0.63      0.34      0.44        50
           8       0.00      0.00      0.00         9

    accuracy                           0.61       400
   macro avg       0.31      0.29      0.29       400
weighted avg       0.58      0.61      0.59       400
```

In [163…
```
from sklearn.linear_model import LogisticRegression
```

In [164…
```
log_reg=LogisticRegression()
log_reg
```

Out[164]:
```
LogisticRegression()
```

In [165…
```
log_reg.fit(X_train,y_train)
```

Out[165]:
```
LogisticRegression
LogisticRegression()
```

In [166…
```
log_pred=log_reg.predict(X_test)
log_pred
```

Out[166]:
```
array([6, 5, 5, 6, 5, 7, 6, 6, 5, 5, 6, 5, 5, 5, 5, 4, 5, 6, 5, 5, 5, 6,
       6, 6, 5, 6, 6, 6, 6, 5, 5, 5, 6, 6, 6, 5, 7, 5, 6, 5, 6, 6, 5, 7,
       5, 6, 5, 6, 7, 5, 6, 7, 6, 7, 6, 6, 6, 5, 5, 5, 6, 5, 6, 6, 5, 5,
       6, 6, 5, 6, 6, 5, 6, 5, 5, 5, 6, 6, 6, 6, 6, 5, 6, 5, 6, 6, 6, 5,
       6, 5, 7, 6, 7, 5, 5, 6, 5, 5, 6, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6,
       5, 6, 5, 5, 5, 5, 5, 5, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 5,
       5, 6, 5, 7, 6, 5, 5, 6, 6, 6, 5, 6, 5, 6, 6, 5, 5, 6, 5, 6, 6, 6,
       5, 5, 6, 5, 6, 7, 6, 6, 6, 6, 6, 6, 6, 7, 6, 6, 6, 7, 6, 6, 5, 5,
       6, 5, 5, 6, 6, 6, 6, 6, 5, 5, 6, 7, 5, 6, 5, 5, 6, 6, 5, 6, 5, 6,
       7, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 7, 5, 6, 6, 5, 6, 5, 6, 5, 5, 5,
       6, 6, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 5, 6, 5, 6, 5, 3, 5, 5, 5, 5,
       6, 7, 6, 6, 5, 5, 6, 6, 5, 6, 5, 5, 5, 5, 7, 7, 6, 6, 6, 5, 6, 5,
       5, 6, 5, 6, 6, 5, 5, 6, 6, 6, 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 5,
       6, 6, 5, 6, 6, 5, 7, 6, 5, 6, 5, 6, 5, 6, 6, 6, 6, 6, 6, 7, 5, 7,
       6, 5, 6, 5, 6, 5, 6, 7, 6, 7, 5, 5, 6, 6, 6, 7, 6, 6, 6, 5, 5, 5,
       6, 7, 6, 6, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 6, 5, 5, 6, 5, 5,
       7, 5, 5, 5, 6, 6, 6, 5, 5, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6,
       6, 7, 6, 5, 5, 6, 6, 5, 6, 5, 5, 6, 5, 6, 6, 6, 6, 6, 7, 6, 5, 6,
       5, 6, 5, 6], dtype=int64)
```

In [167…
```
report_log=classification_report(y_test, log_pred, zero_division=False)
print(report_log)
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        15
           5       0.66      0.66      0.66       172
           6       0.49      0.64      0.55       152
           7       0.54      0.30      0.38        50
           8       0.00      0.00      0.00         9

    accuracy                           0.56       400
   macro avg       0.28      0.27      0.27       400
weighted avg       0.54      0.56      0.54       400
```

2.6 Increasing performance of model (Hyper-parameter Tuning)

In [180…
```
### using different kernels to guage performanc of model for constant hyper-parameter C
kernels=['linear', 'rbf', 'poly', 'sigmoid']
for kernel in kernels:
    model=SVC(kernel=kernel, C=1.0)
    model.fit(X_train, y_train)
    print("For kernel {} modal accuracy is {}".format(kernel, model.score(X_test, y_test)))
```

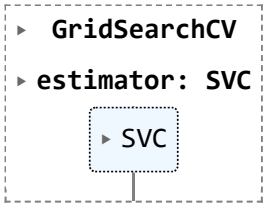
```
For kernel linear modal accuracy is 0.5575
For kernel rbf modal accuracy is 0.615
For kernel poly modal accuracy is 0.5925
For kernel sigmoid modal accuracy is 0.4475
```

In [178…
```
### for polynomial kernel with different degree of polynomial with constant hyper-parameter C
for degree in range(1,11):
    model=SVC(kernel='poly', degree=degree, C=100)
    model.fit(X_train, y_train)
    print("For degree {} modal accuracy is {}".format(degree, model.score(X_test, y_test)))
```

```
For degree 1 modal accuracy is 0.5575
For degree 2 modal accuracy is 0.485
For degree 3 modal accuracy is 0.5525
For degree 4 modal accuracy is 0.575
For degree 5 modal accuracy is 0.5875
For degree 6 modal accuracy is 0.5775
For degree 7 modal accuracy is 0.57
For degree 8 modal accuracy is 0.5925
For degree 9 modal accuracy is 0.615
For degree 10 modal accuracy is 0.53
```

In [193…
```
#### using gridsearchcv to increase model efficiency by combining above parameters
param_grid={'C':[i for i in range(1,100,10)], 'kernel':['linear', 'rbf', 'poly', 'sigmoid'], 'degree':[1,2,3,4,5,6,7]}
grid=GridSearchCV(SVC(), param_grid=param_grid)
grid.fit(X_train, y_train)
```

Out[193]:



In [194…
```
### getting best parameters after gridsearchCV
print("Best parameters are {} for optimal accuracy.".format(grid.best_params_))
```

```
Best parameters are {'C': 11, 'degree': 1, 'kernel': 'rbf'} for optimal accuracy.
```

In [196…
```
### getting best accuracy after gridsearchCV
print("Best accuracy is {}".format(grid.score(X_test, y_test)))
```

```
Best accuracy is 0.6225
```

In [202…
```
### Accuracy_score comparision
print("Accuracy of SVC without Hyperparameter Tuning is {}\nAccuracy of SVC with Hyperparameter Tuning is {}\nIncrease in accura
accuracy_score(y_test, svc_pred)*100, round(grid.score(X_test, y_test)*100,2),round(grid.score(X_test, y_test)-accuracy_score(y_t
```

```
Accuracy of SVC without Hyperparameter Tuning is 61.5
Accuracy of SVC with Hyperparameter Tuning is 62.25
Increase in accuracy is 0.75
```

3.0 Support Vector Regressor Grad Admission Dataset

In [40]:
```
from IPython import display
display.Image("student.png")
```

Out[40]:

3.1 Importing Graduate Admission Dataset and doing statistical analysis

In [21]:
```
dataset1=pd.read_csv("https://raw.githubusercontent.com/srinivasav22/Graduate-Admission-Prediction/master/Admission_Predict_Ver1
dataset1.drop('Serial No.',axis=1, inplace=True)
```

In [22]:
```
dataset1.head()
```

Out[22]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [23]:
```
dataset1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   GRE Score          500 non-null    int64  
 1   TOEFL Score        500 non-null    int64  
 2   University Rating  500 non-null    int64  
 3   SOP                500 non-null    float64
 4   LOR                500 non-null    float64
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    int64  
 7   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 31.4 KB
```

In [24]:
```
dataset1.describe().T
```

Out[24]:

	count	mean	std	min	25%	50%	75%	max
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
University Rating	500.0	3.11400	1.143512	1.00	2.0000	3.00	4.00	5.00
SOP	500.0	3.37400	0.991004	1.00	2.5000	3.50	4.00	5.00
LOR	500.0	3.48400	0.925450	1.00	3.0000	3.50	4.00	5.00
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	9.04	9.92
Research	500.0	0.56000	0.496884	0.00	0.0000	1.00	1.00	1.00
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	0.82	0.97

In [25]:
```
dataset1.isnull().sum()
```

Out[25]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

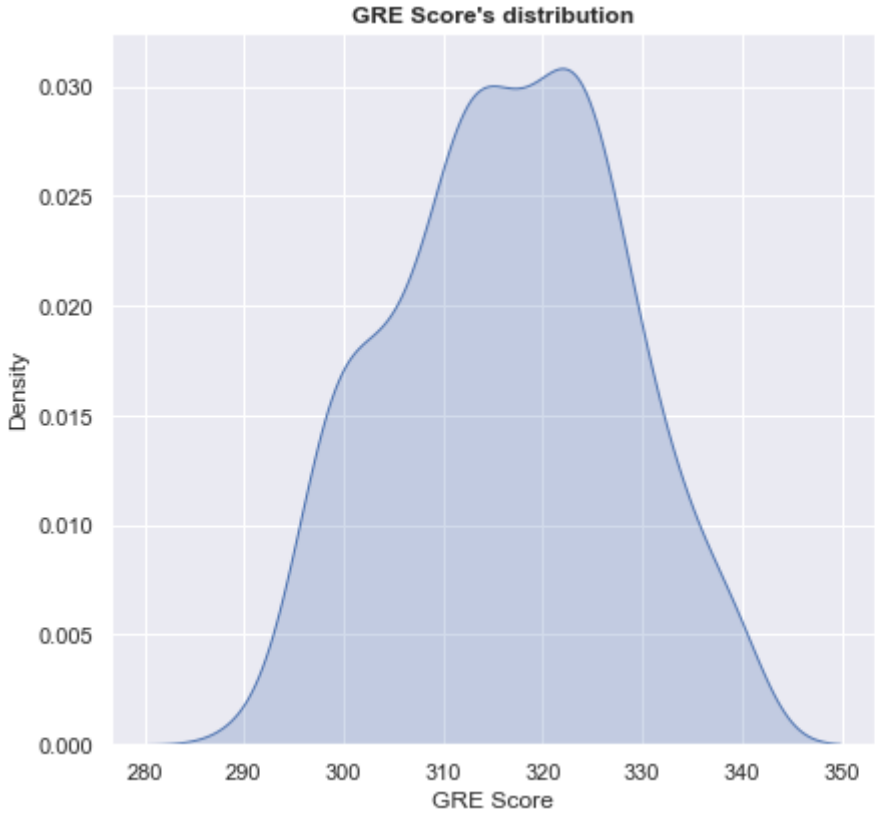
3.2 Visualising Numerical data

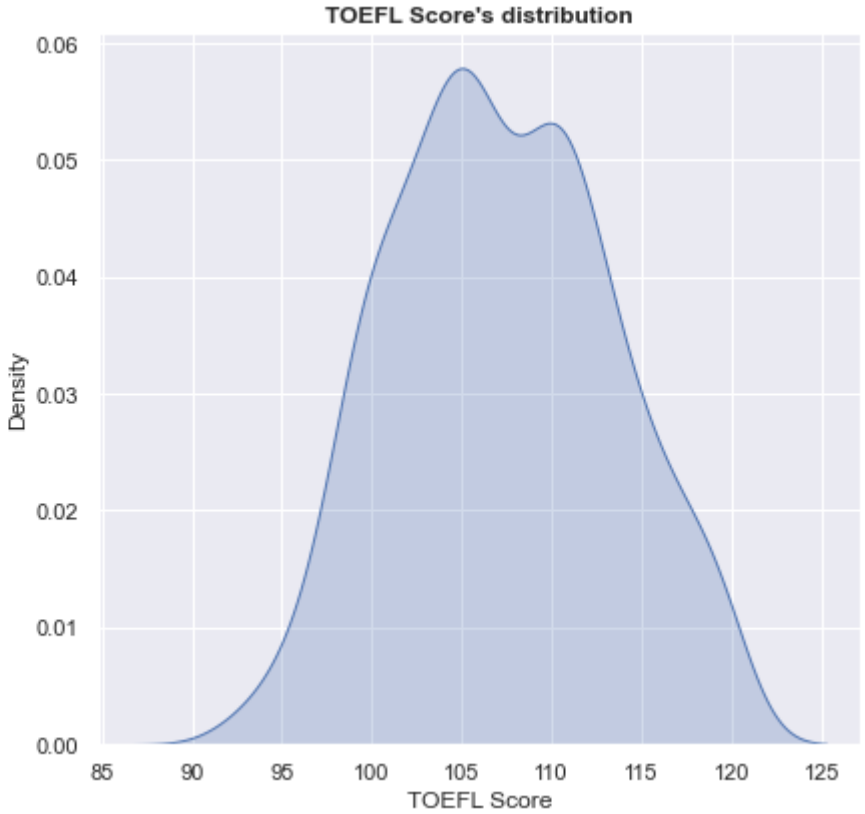
In [26]:
```
numerical_features=dataset1.columns
print(numerical_features)
```

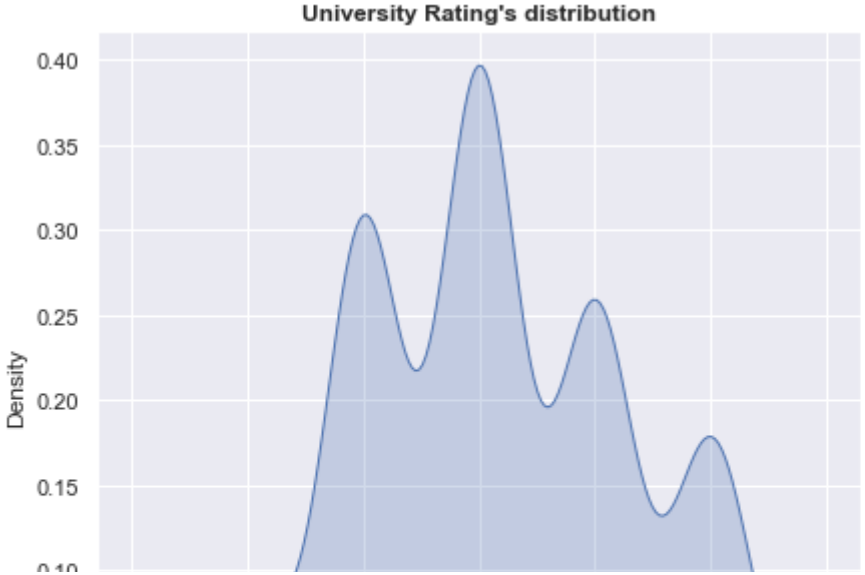
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

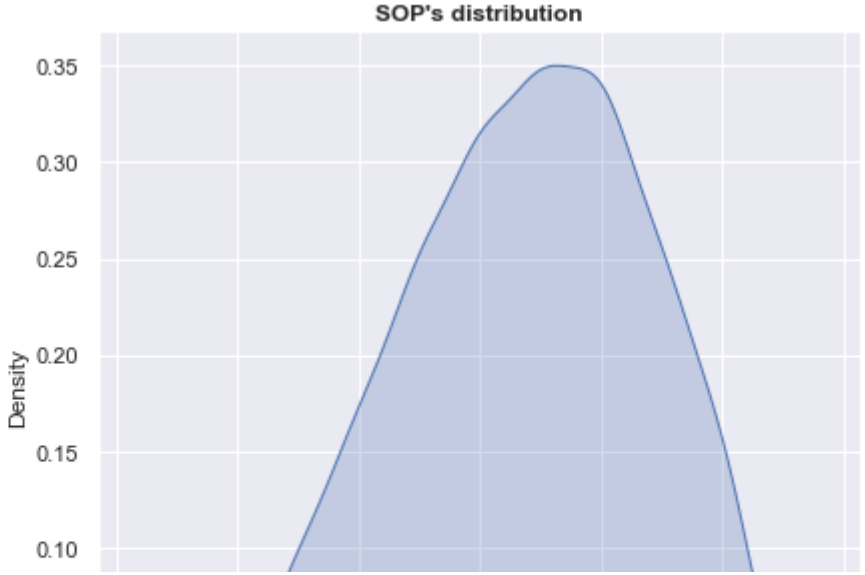
In [27]:
```
plt.figure(figsize=(15,30))
for i in enumerate(numerical_features):
    plt.subplot(4, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(7,5)})
```

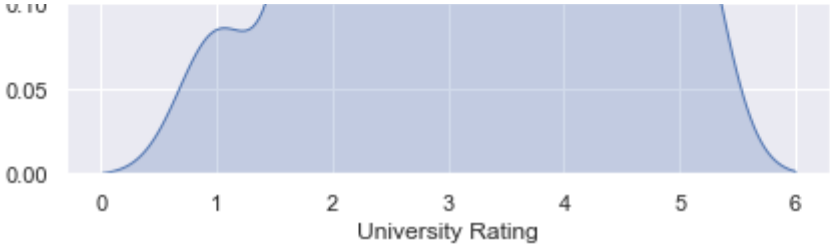
```
    sns.kdeplot(data=dataset1, x=i[1], fill=True)
    plt.title("{}'s distribution".format(i[1]),fontweight="bold")
```

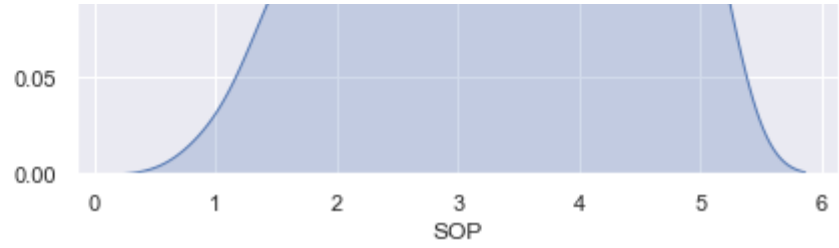

GRE Score's distribution
Density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
280
290
300
310
320
330
340
350
GRE Score

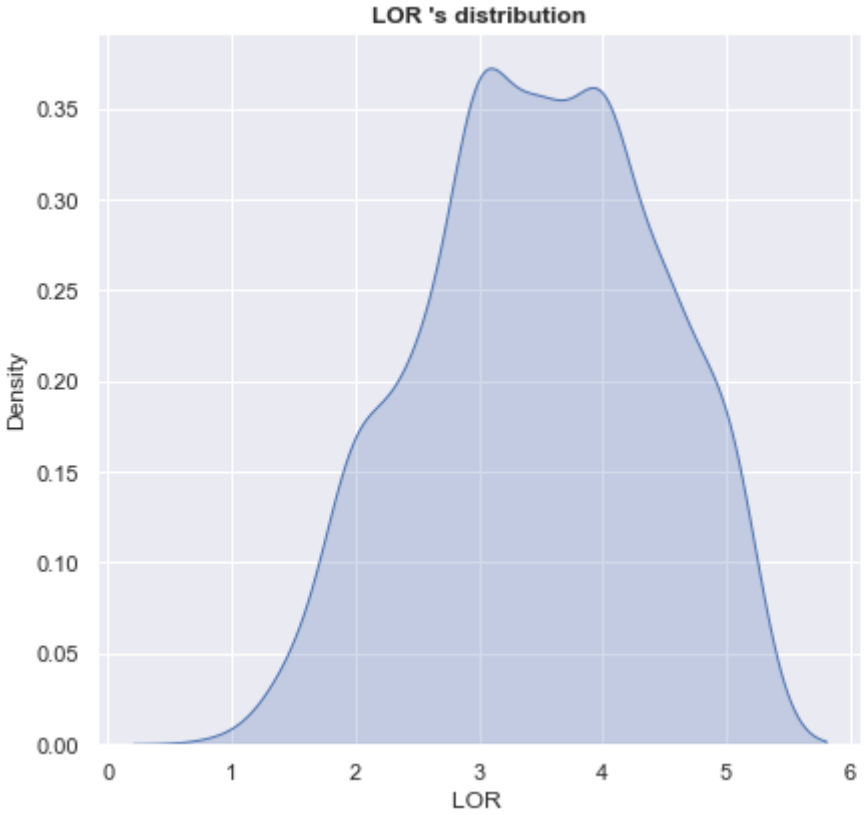

TOEFL Score's distribution
Density
0.06
0.05
0.04
0.03
0.02
0.01
0.00
85
90
95
100
105
110
115
120
125
TOEFL Score

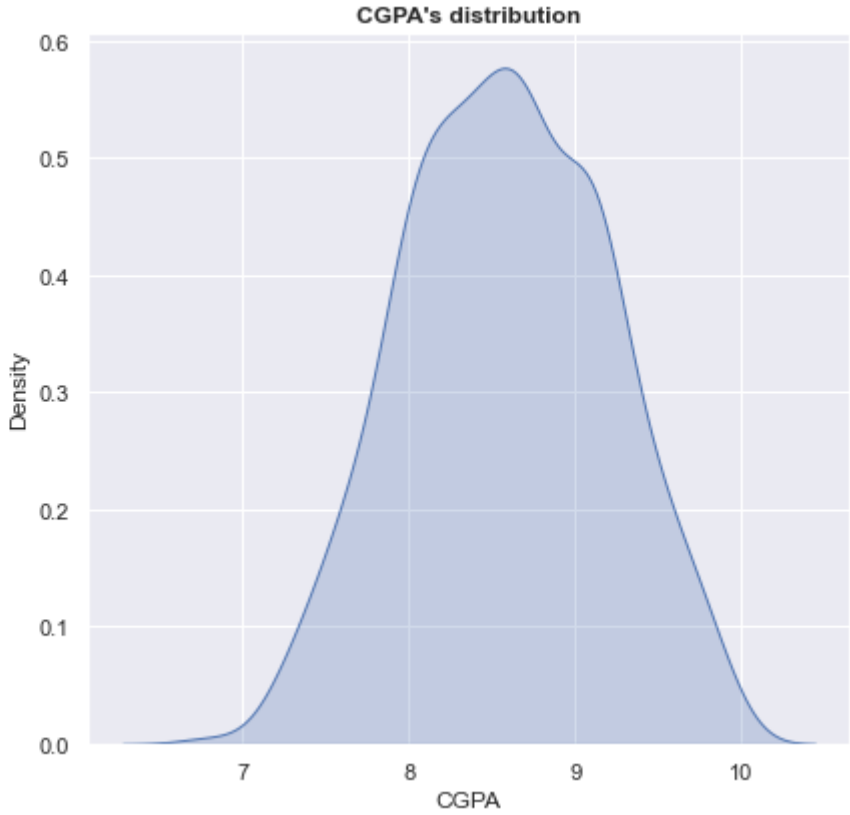

University Rating's distribution
Density
0.40
0.35
0.30
0.25
0.20
0.15

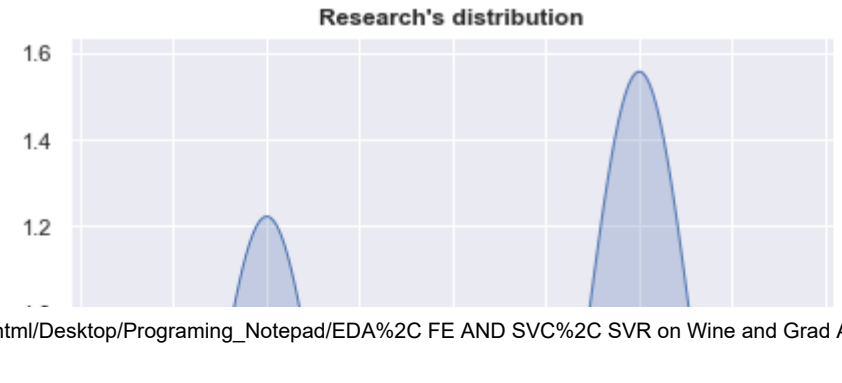

SOP's distribution
Density
0.35
0.30
0.25
0.20
0.15
0.10

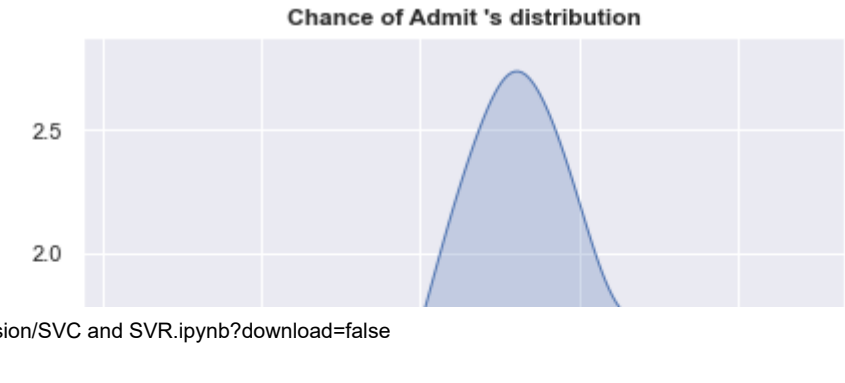

0.05
0.00
0
1
2
3
4
5
6
University Rating

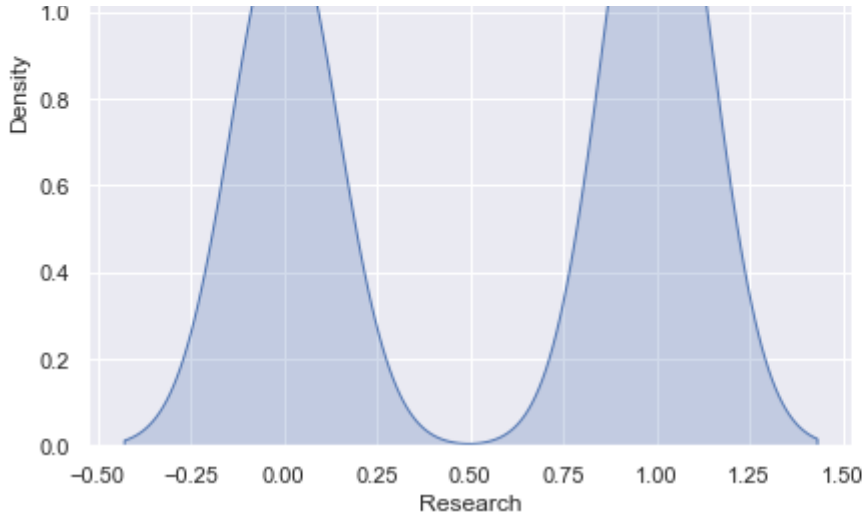

0.05
0.00
0
1
2
3
4
5
6
SOP

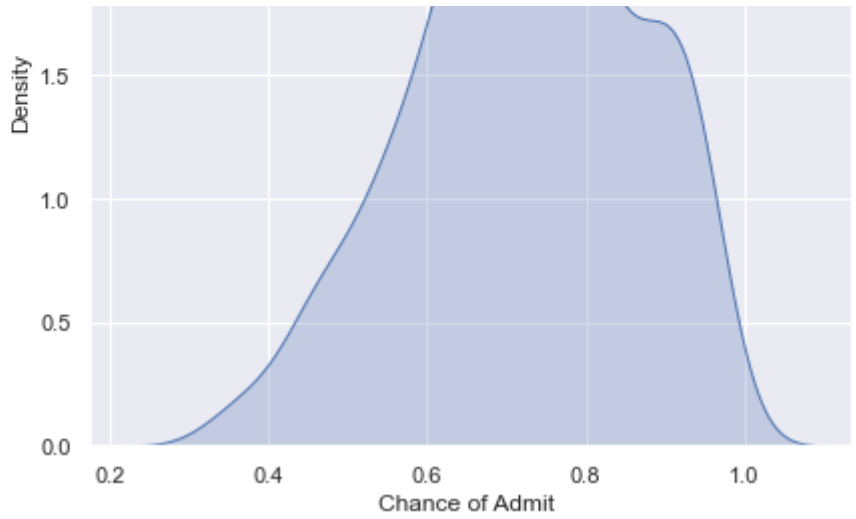

LOR 's distribution
Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
LOR


CGPA's distribution
Density
0.6
0.5
0.4
0.3
0.2
0.1
0.0
7
8
9
10
CGPA


Research's distribution
1.6
1.4
1.2


Chance of Admit 's distribution
2.5
2.0





In [28]:
```
for feature in numerical_features:
    print("'{}' has '{}' No. of unique values".format(feature, dataset1[feature].nunique()))
```

```
'GRE Score' has '49' No. of unique values
'TOEFL Score' has '29' No. of unique values
'University Rating' has '5' No. of unique values
'SOP' has '9' No. of unique values
'LOR ' has '9' No. of unique values
'CGPA' has '184' No. of unique values
'Research' has '2' No. of unique values
'Chance of Admit ' has '61' No. of unique values
```

In [29]:
```
discrete_feature=[feature for feature in numerical_features if dataset1[feature].nunique()<10]
print(discrete_feature)
```

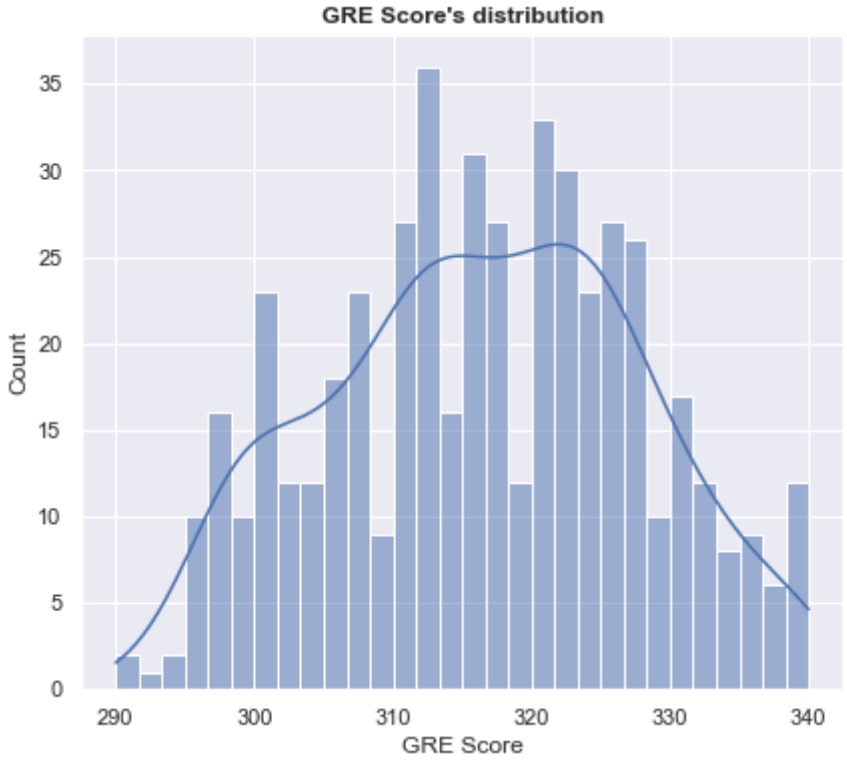
```
['University Rating', 'SOP', 'LOR ', 'Research']
```

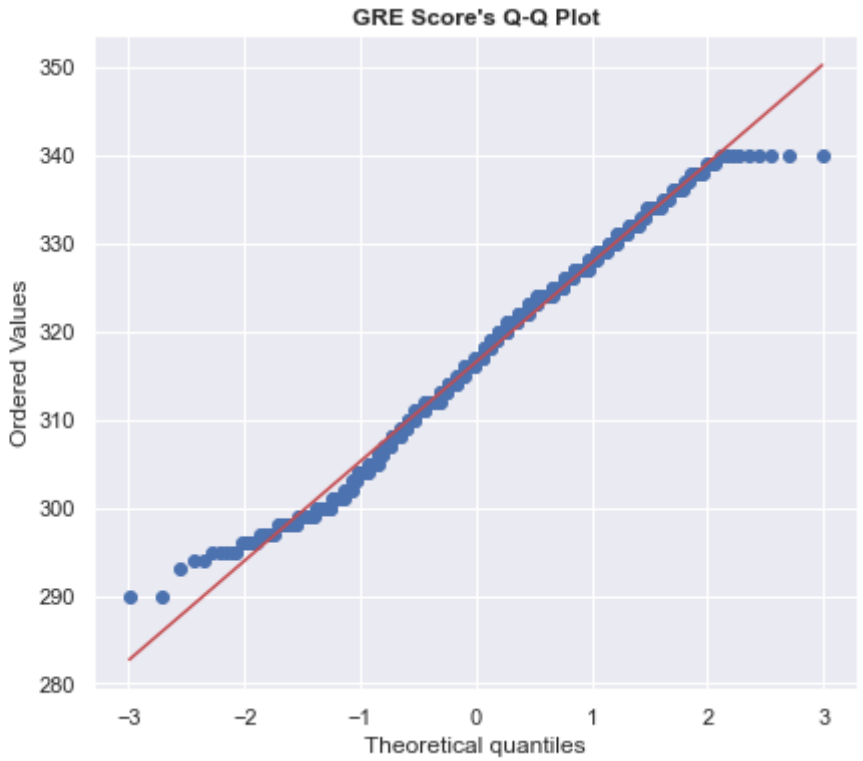
In [33]:
```
continuous_features=[feature for feature in numerical_features if feature not in discrete_feature]
print(continuous_features)
```

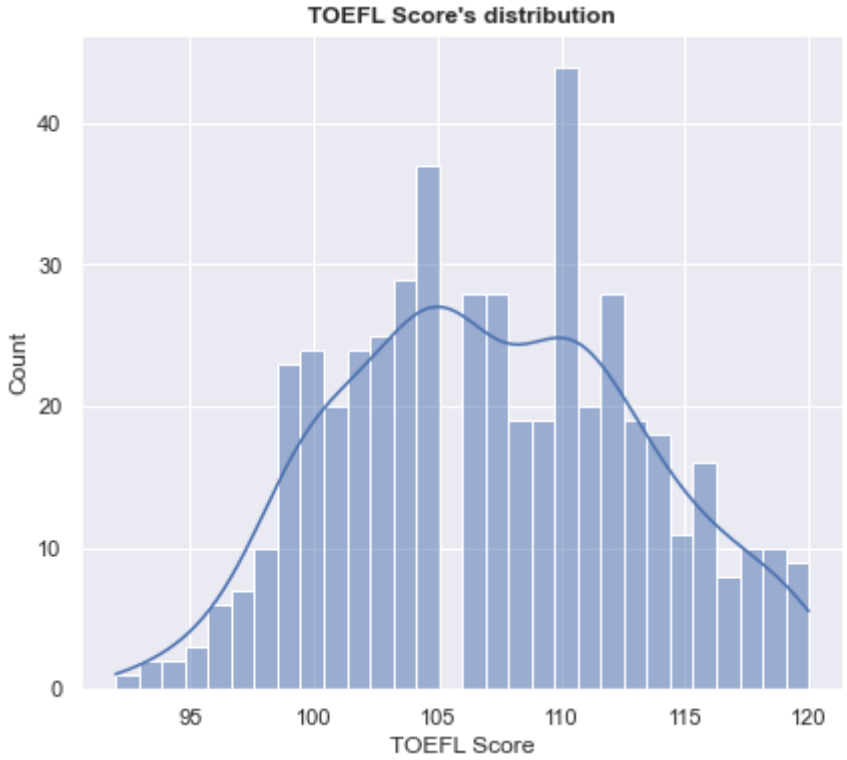
```
['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit ']
```

In [31]:
```
### Checking distribution of Continuous numerical features

for i in continuous_features:
    plt.figure(figsize=(15,6))
    plt.subplot(121)
```

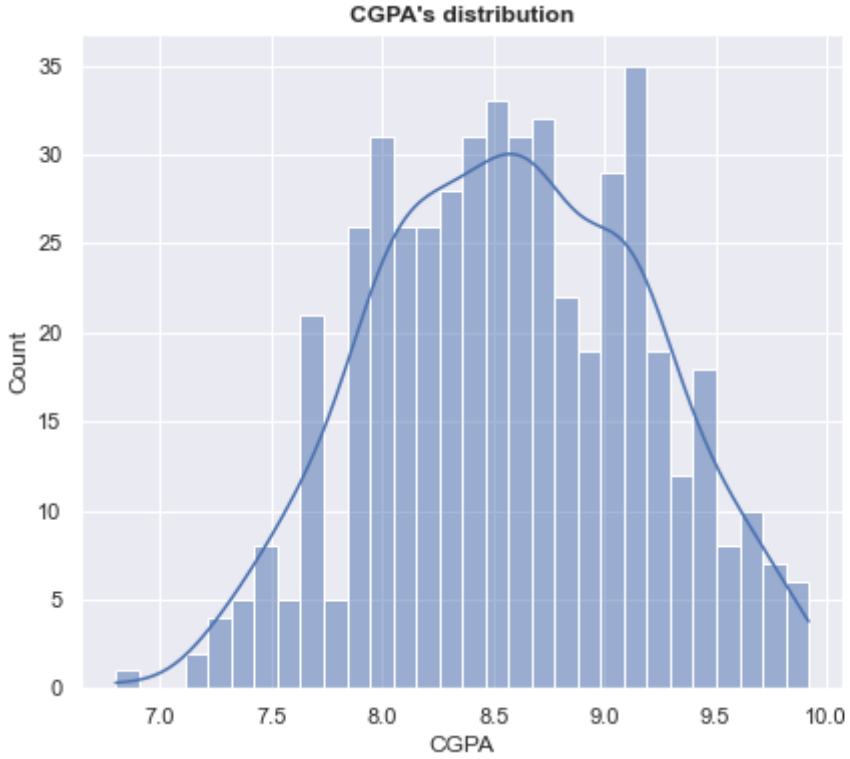
```
    sns.histplot(data=dataset1, x=i, kde=True, bins=30)
    plt.title("{}'s distribution".format(i),fontweight="bold")

    plt.subplot(122)
    stats.probplot(dataset1[i], dist='norm', plot=plt)
    plt.title("{}'s Q-Q Plot".format(i),fontweight="bold")
    plt.show();
```

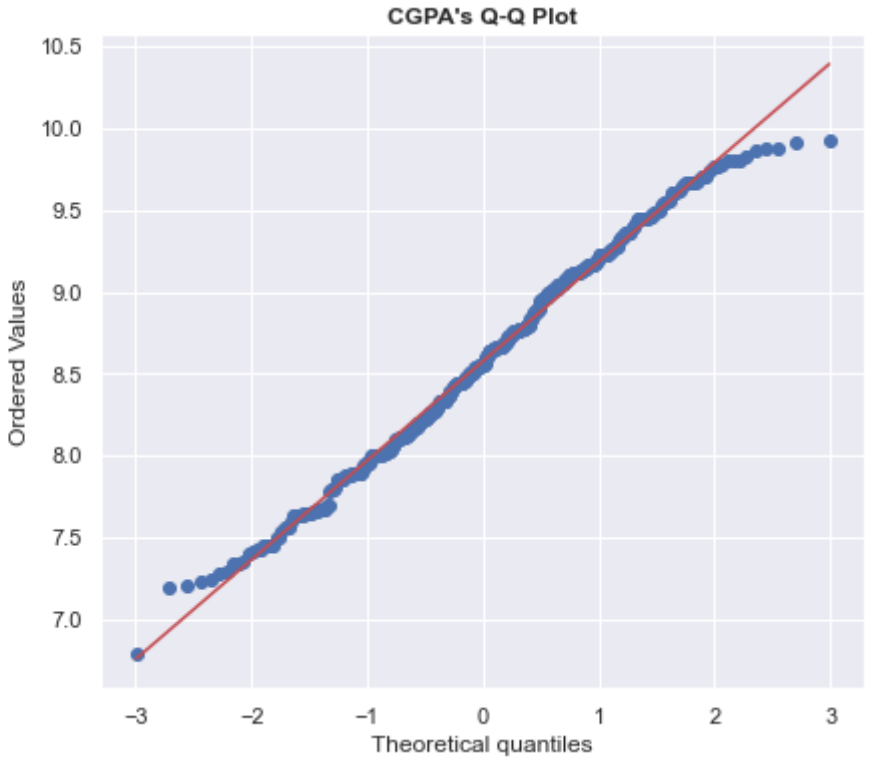


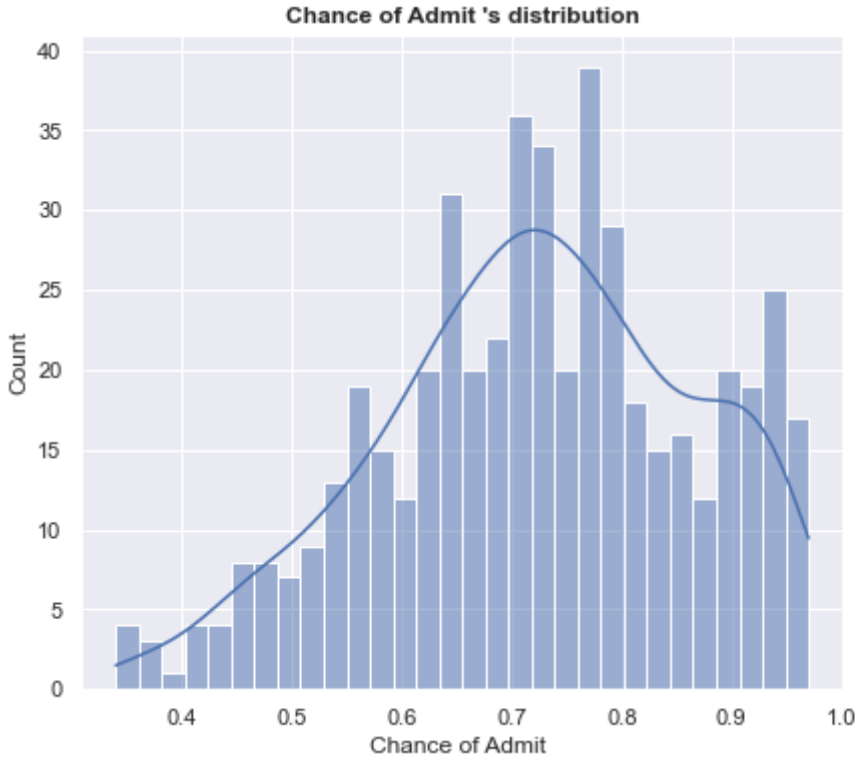


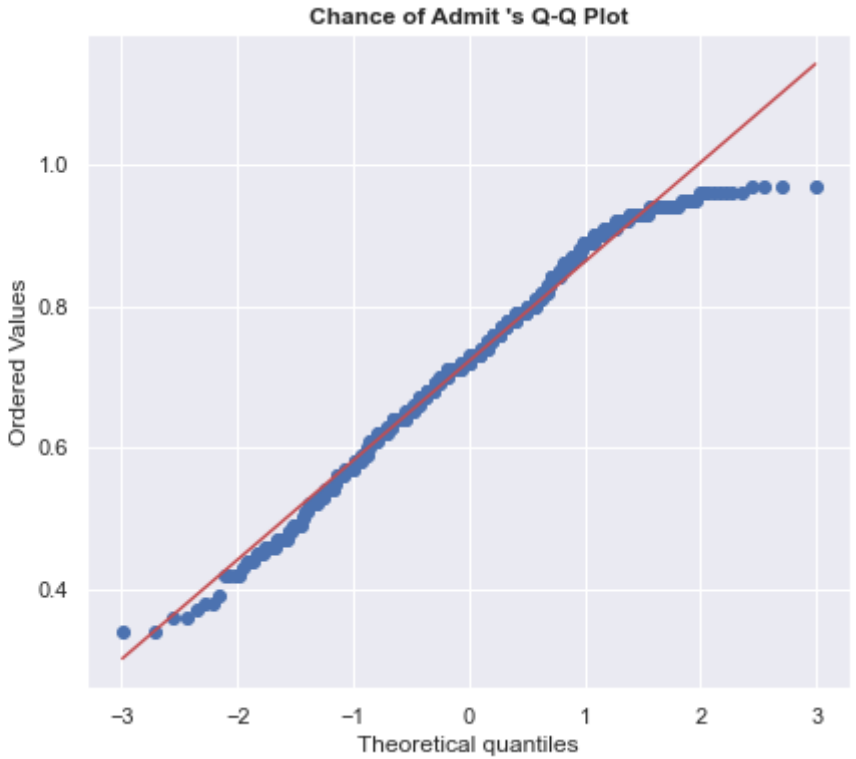

TOEFL Score's distribution
Count
TOEFL Score
0
10
20
30
40
95
100
105
110
115
120


TOEFL Score's Q-Q Plot
Ordered Values
Theoretical quantiles
90
95
100
105
110
115
120
125
−3
−2
−1
0
1
2
3

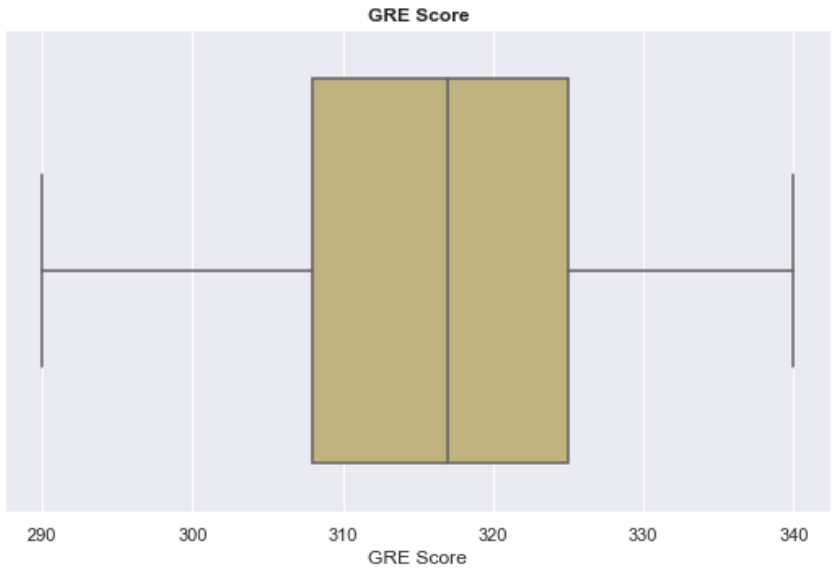

CGPA's distribution
Count
CGPA
0
5
10
15
20
25
30
35
7.0
7.5
8.0
8.5
9.0
9.5
10.0

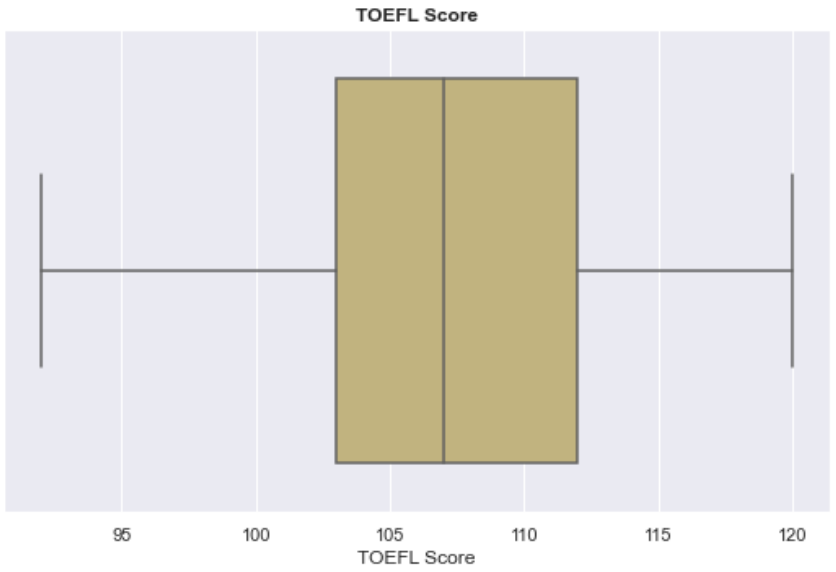

CGPA's Q-Q Plot
Ordered Values
Theoretical quantiles
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
−3
−2
−1
0
1
2
3

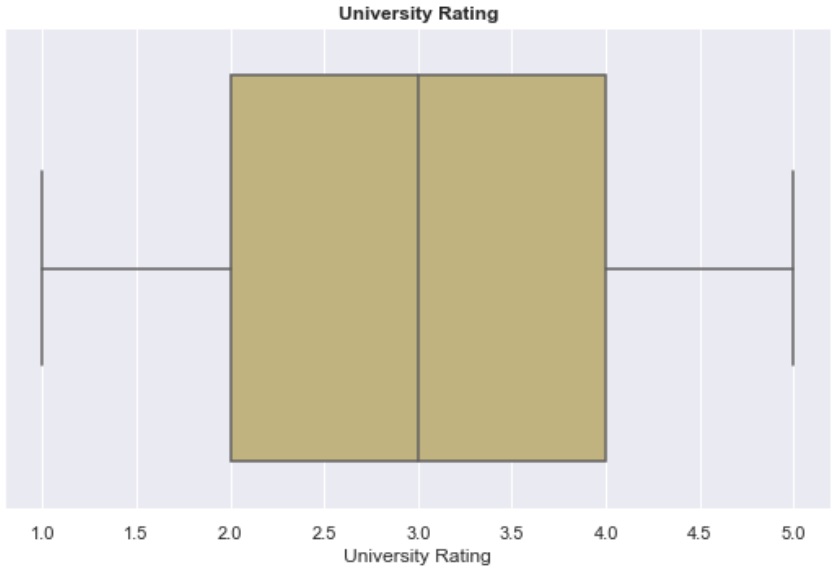


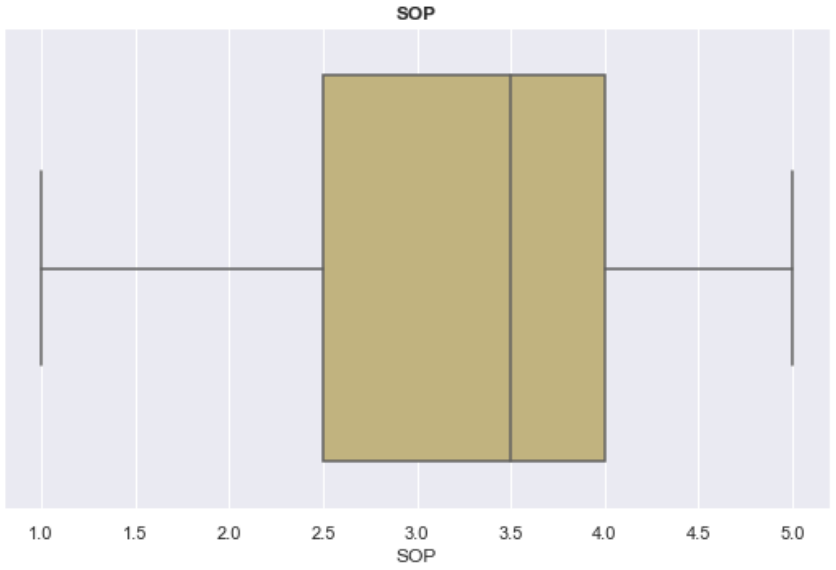


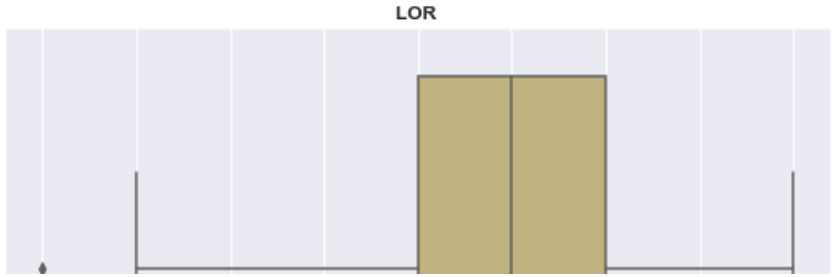
In [32]:
```
### Checking outliers in numerical features

plt.figure(figsize=(20,25))
for i in enumerate(numerical_features):
    plt.subplot(4, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(10,6)})
    sns.boxplot(data=dataset1, x=i[1], color='y')
    plt.title("{}".format(i[1]), fontweight="bold")
```


GRE Score
290
300
310
320
330
340
GRE Score

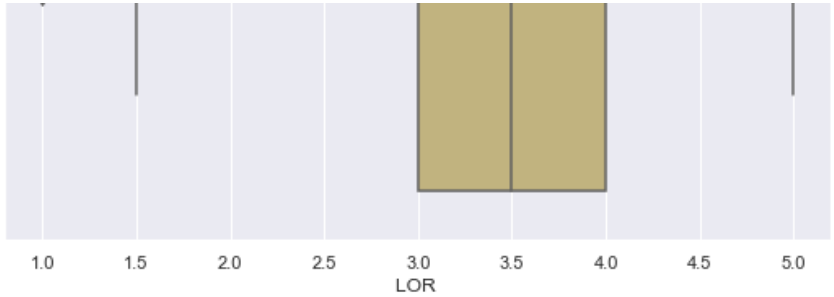

TOEFL Score
95
100
105
110
115
120
TOEFL Score

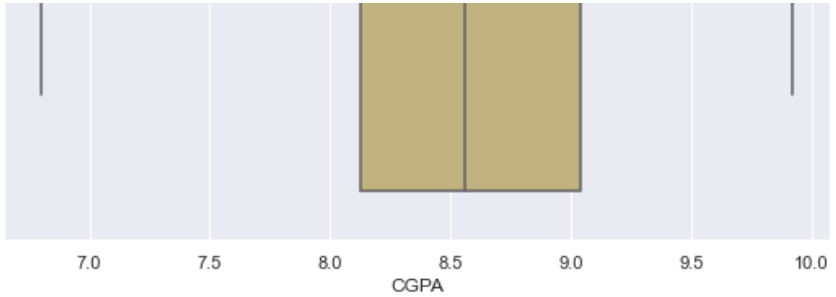

University Rating
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating

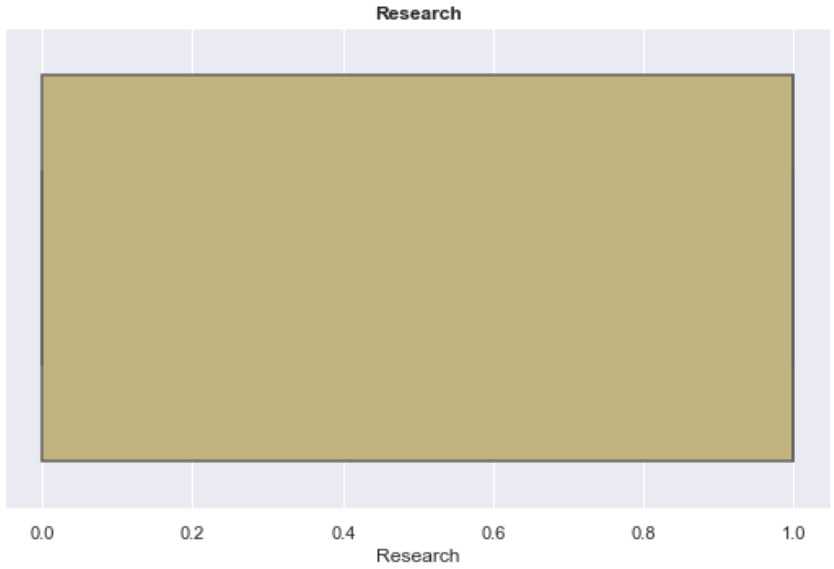

SOP
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
SOP

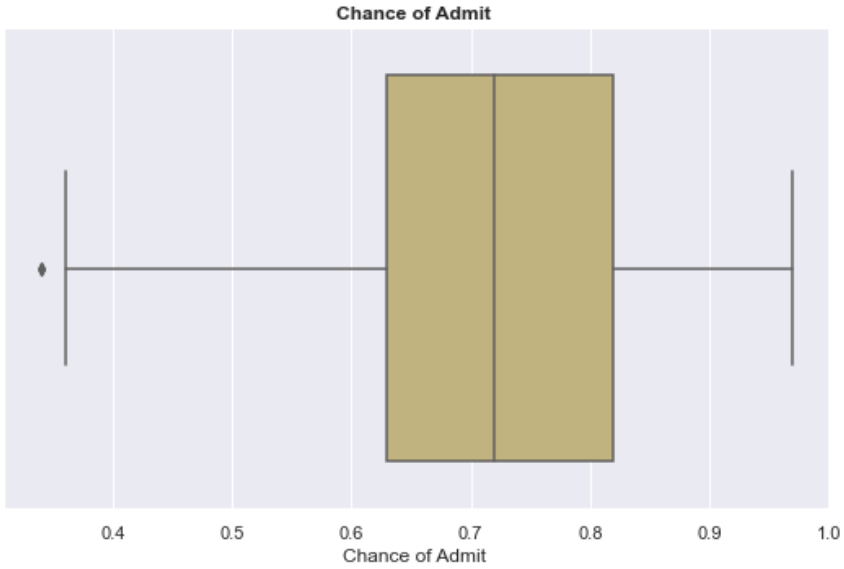

LOR


CGPA

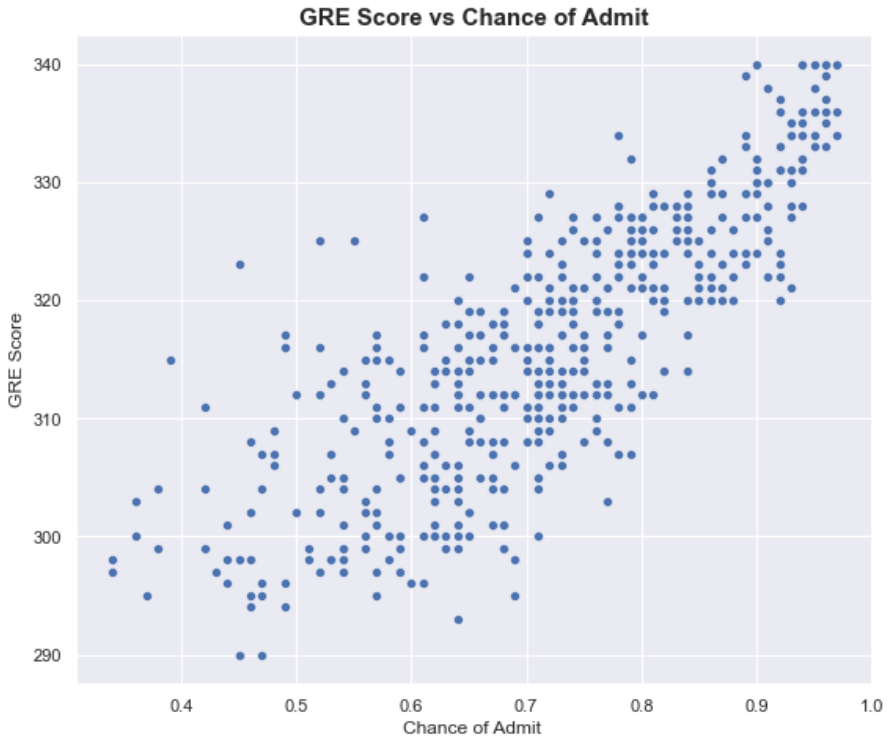


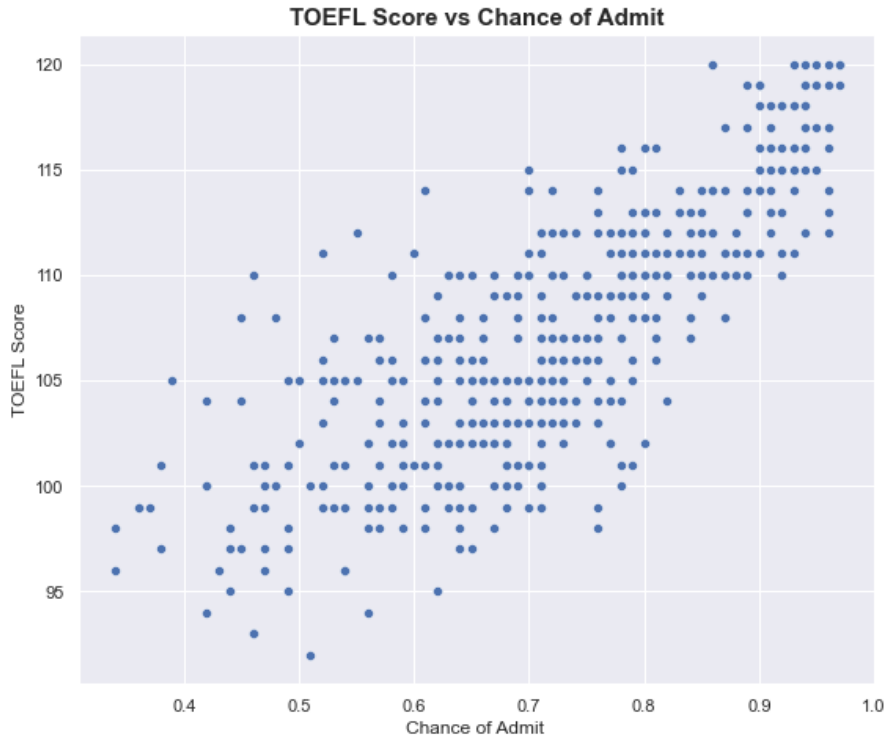


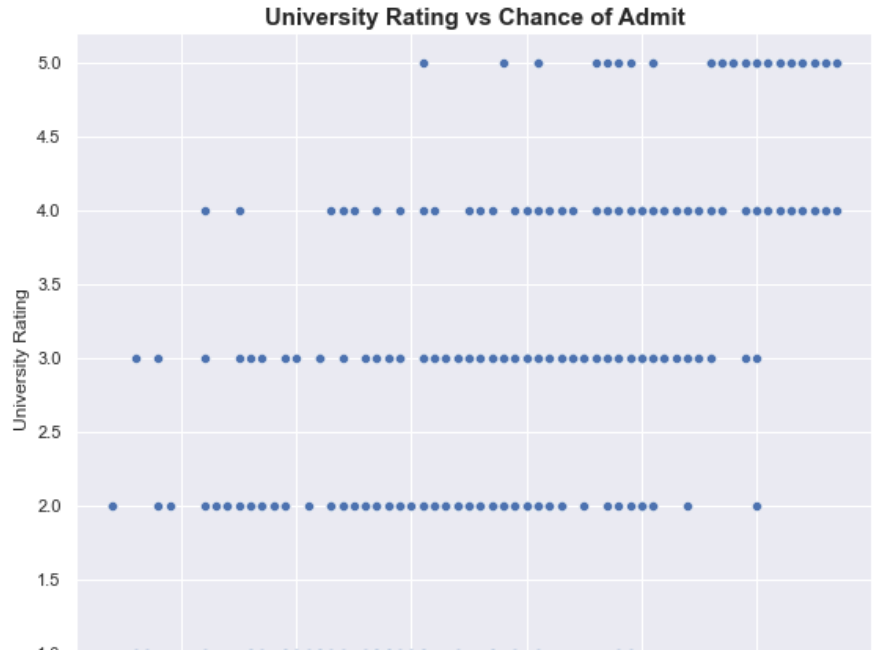


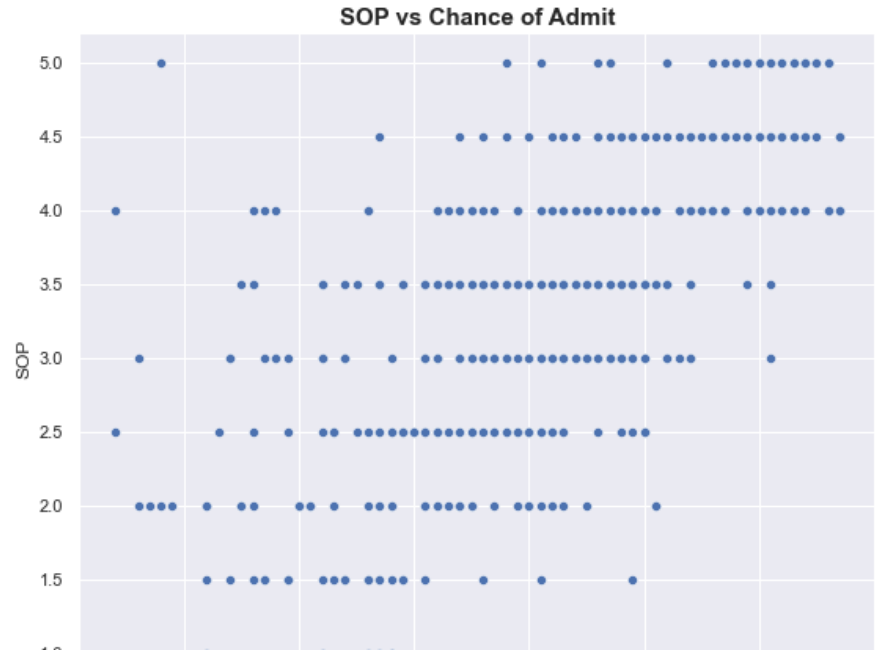


In [36]:
```
plt.figure(figsize=(20,35))
for i in enumerate(numerical_features):
    plt.subplot(4, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(10,8)})
    sns.scatterplot(data=dataset1, y=i[1], x='Chance of Admit ')
    plt.title("{} vs Chance of Admit".format(i[1]),fontsize=15, fontweight="bold")
```

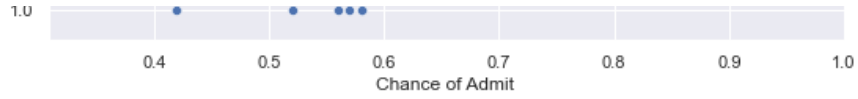

GRE Score vs Chance of Admit
GRE Score
Chance of Admit
340
330
320
310
300
290
0.4
0.5
0.6
0.7
0.8
0.9
1.0

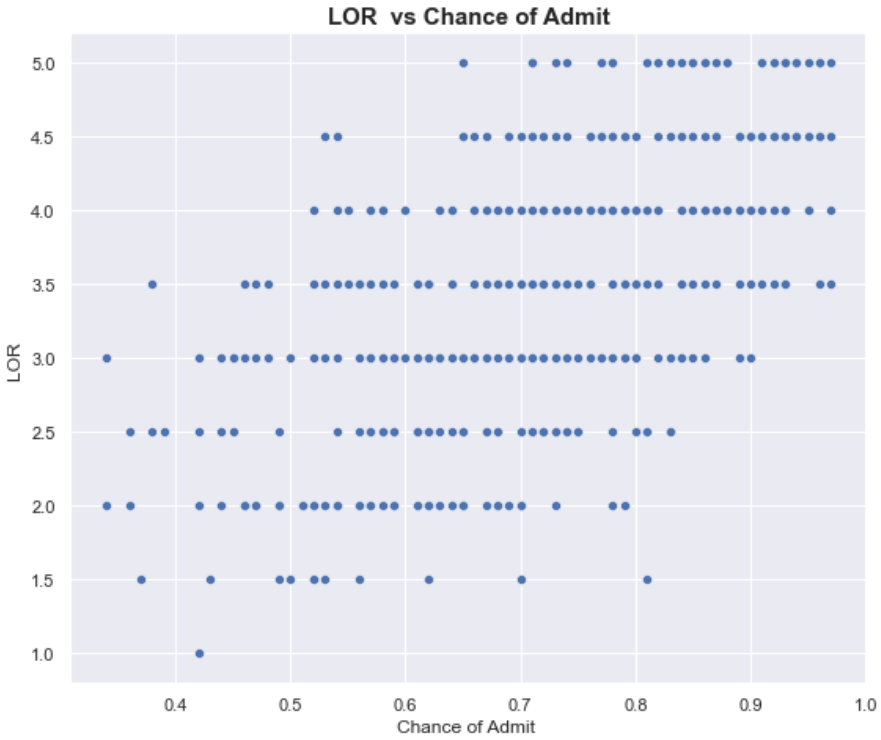

TOEFL Score vs Chance of Admit
TOEFL Score
Chance of Admit
120
115
110
105
100
95
0.4
0.5
0.6
0.7
0.8
0.9
1.0

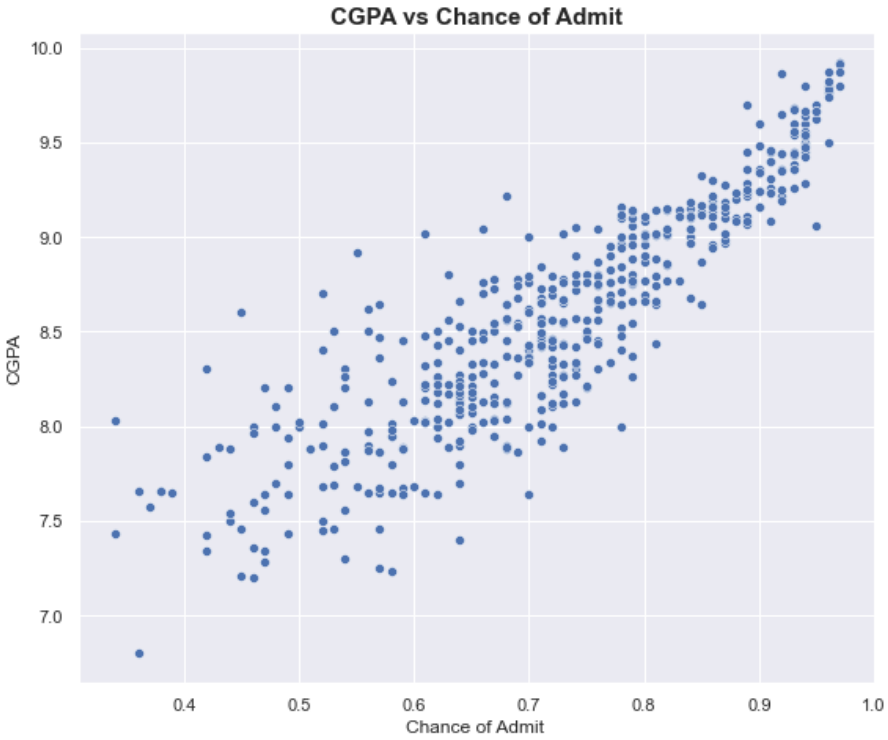

University Rating vs Chance of Admit
University Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5

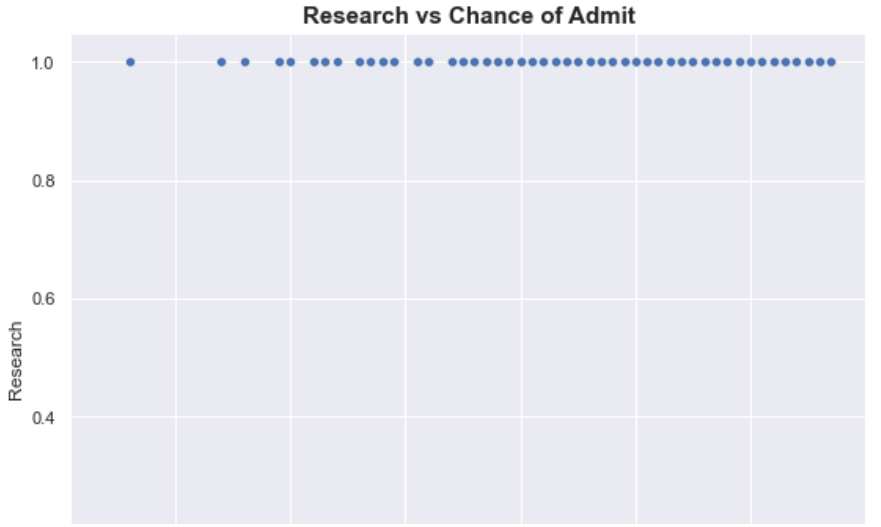

SOP vs Chance of Admit
SOP
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5

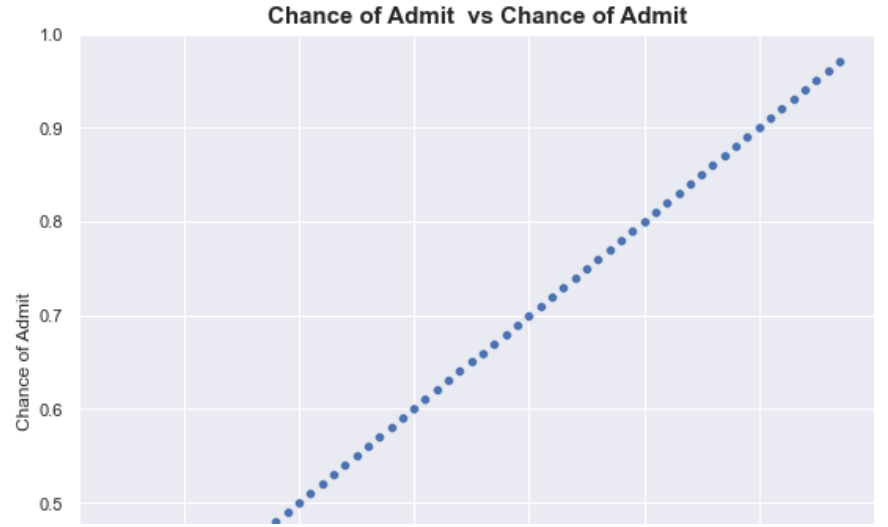

1.0
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Chance of Admit

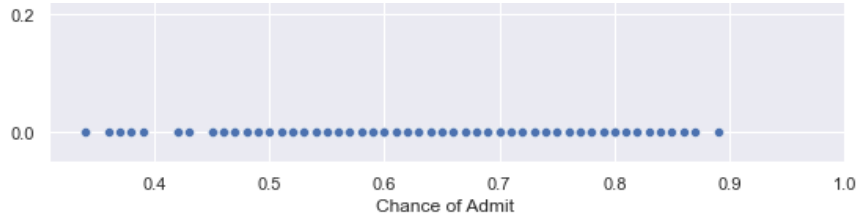

1.0
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Chance of Admit

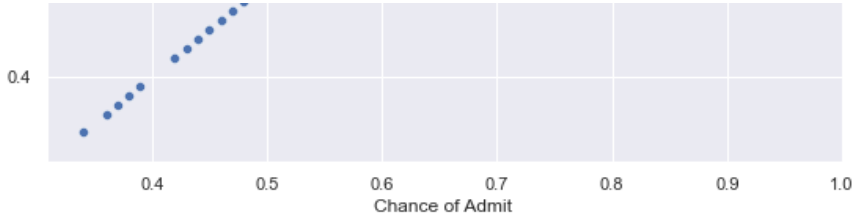

LOR vs Chance of Admit
LOR
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Chance of Admit


CGPA vs Chance of Admit
CGPA
10.0
9.5
9.0
8.5
8.0
7.5
7.0
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Chance of Admit


Research vs Chance of Admit
Research
1.0
0.8
0.6
0.4


Chance of Admit vs Chance of Admit
Chance of Admit
1.0
0.9
0.8
0.7
0.6
0.5





In [34]:
```
dataset1.columns
```

Out[34]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

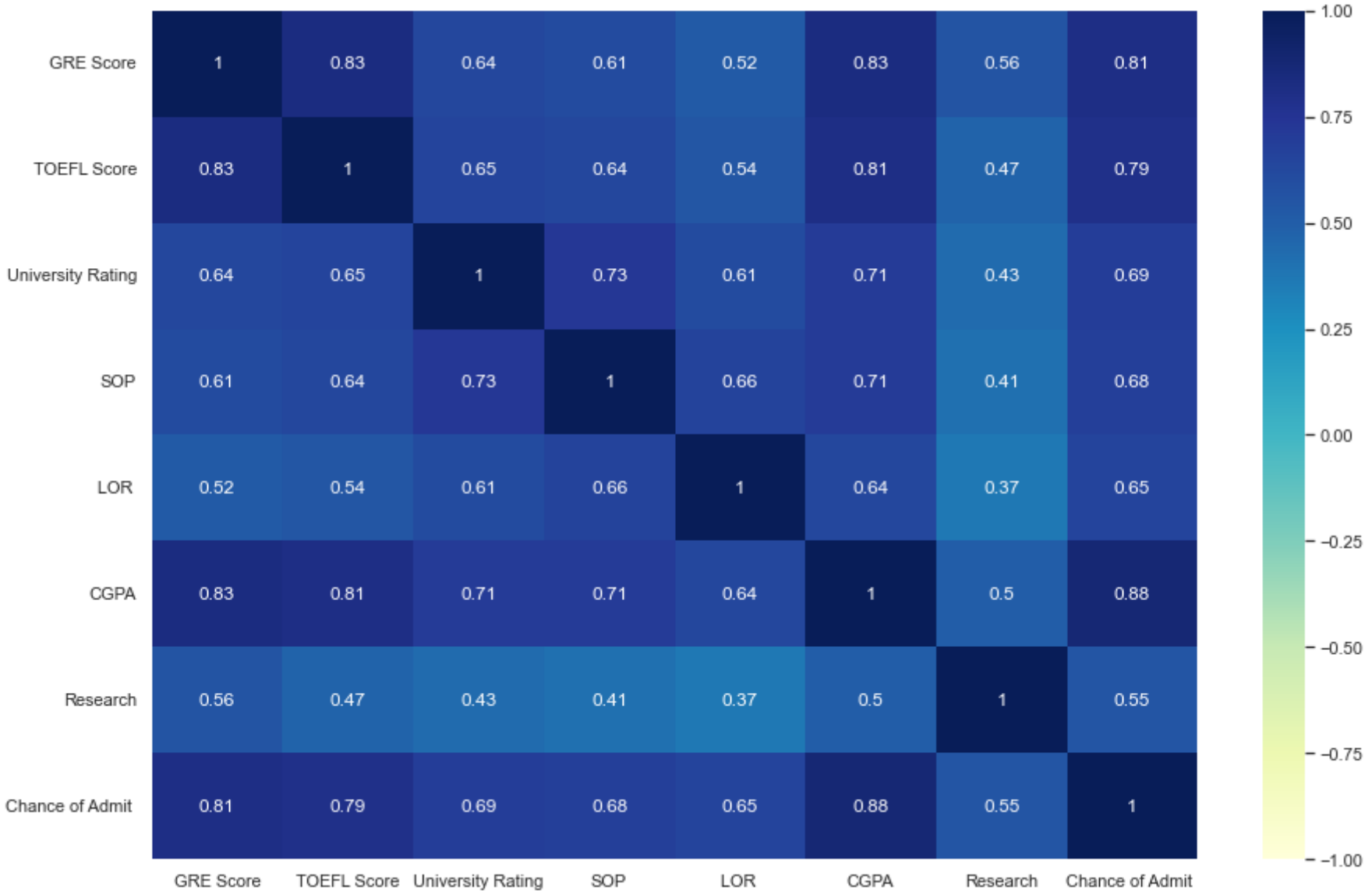
In [233…]:
```
corr1=round(dataset1.corr(),2)
corr1
```

Out[233]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.00	0.83	0.64	0.61	0.52	0.83	0.56	0.81
TOEFL Score	0.83	1.00	0.65	0.64	0.54	0.81	0.47	0.79
University Rating	0.64	0.65	1.00	0.73	0.61	0.71	0.43	0.69
SOP	0.61	0.64	0.73	1.00	0.66	0.71	0.41	0.68
LOR	0.52	0.54	0.61	0.66	1.00	0.64	0.37	0.65
CGPA	0.83	0.81	0.71	0.71	0.64	1.00	0.50	0.88
Research	0.56	0.47	0.43	0.41	0.37	0.50	1.00	0.55
Chance of Admit	0.81	0.79	0.69	0.68	0.65	0.88	0.55	1.00

In [236…]:
```
### Plotting heatmap for visualising the correlation between features
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(data=corr1, annot=True,  vmin=-1, vmax=1, cmap="YlGnBu")
```

Out[236]:
```
<AxesSubplot:>
```



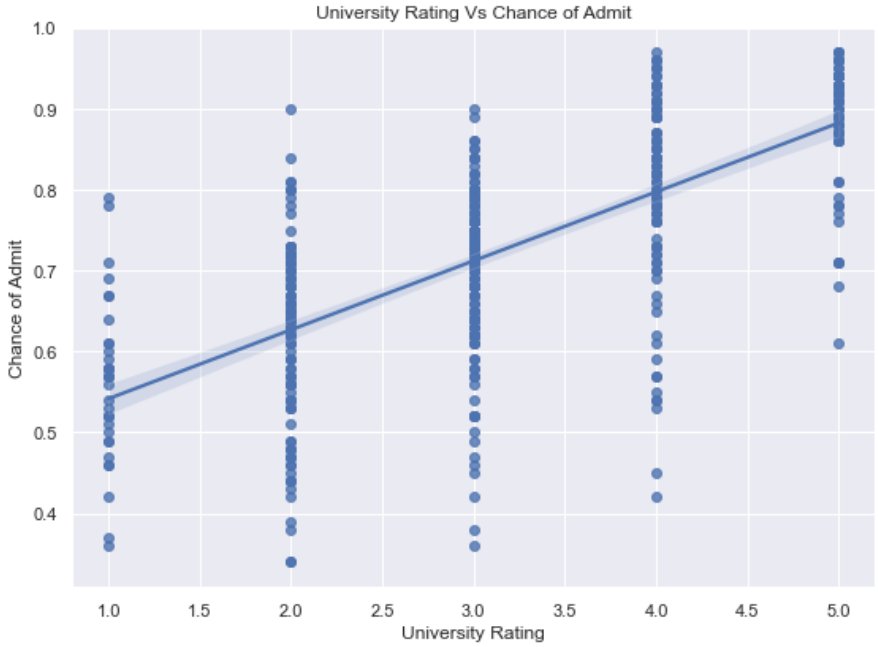
In [239…

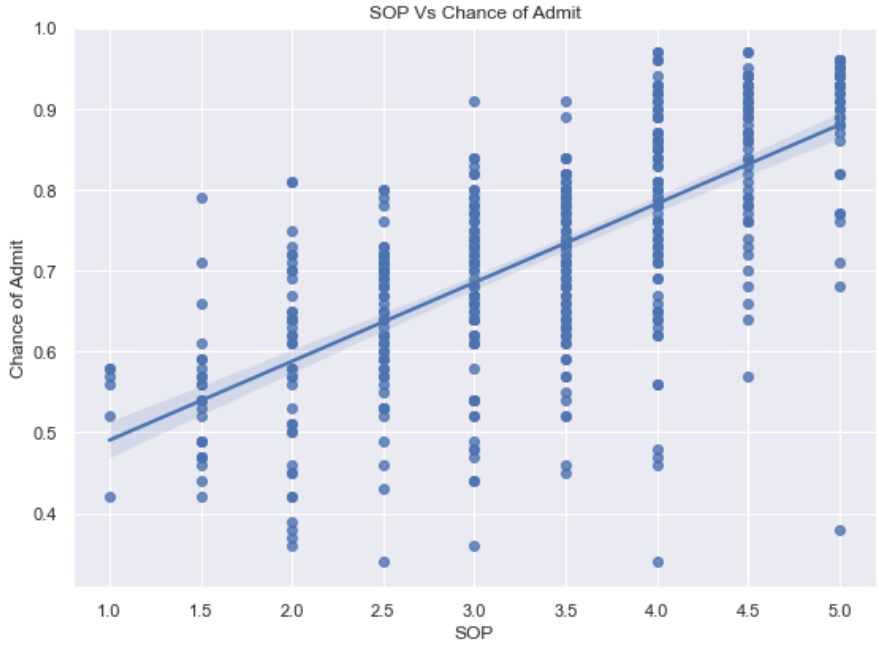
```
plt.figure(figsize=(20,30))
for i in enumerate([feature for feature in numerical_features if feature not in ['Chance of Admit ']]):
    plt.subplot(4, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(10,6)})
    sns.regplot(data=dataset1, x=i[1], y='Chance of Admit ')
    plt.xlabel(i[1])
```

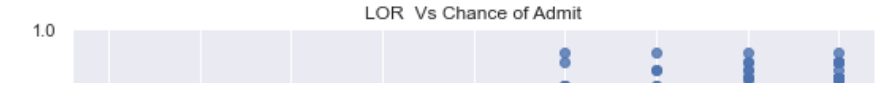
```
    plt.ylabel("Chance of Admit")
    plt.title("{} Vs Chance of Admit".format(i[1]))
```

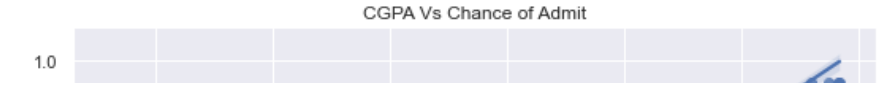

GRE Score Vs Chance of Admit
Chance of Admit
GRE Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
290
300
310
320
330
340

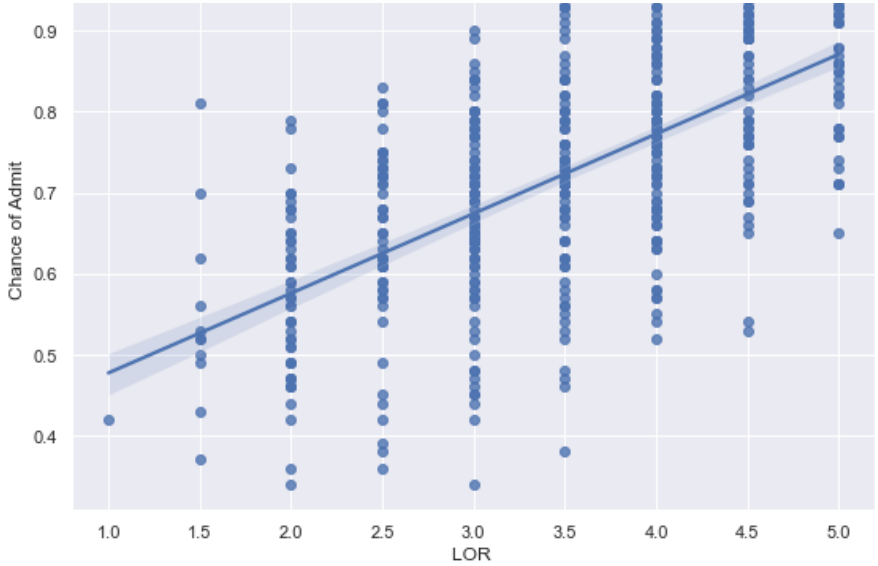

TOEFL Score Vs Chance of Admit
Chance of Admit
TOEFL Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
95
100
105
110
115
120

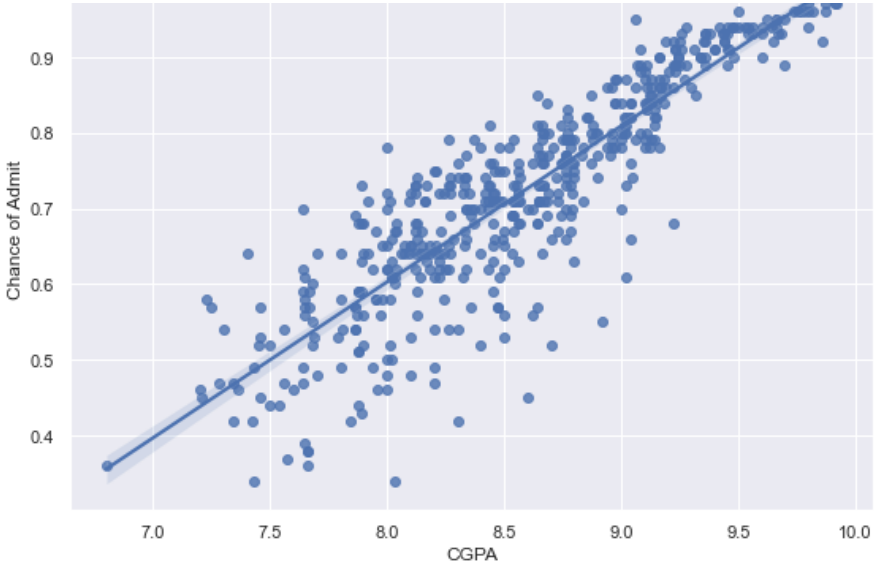

University Rating Vs Chance of Admit
Chance of Admit
University Rating
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

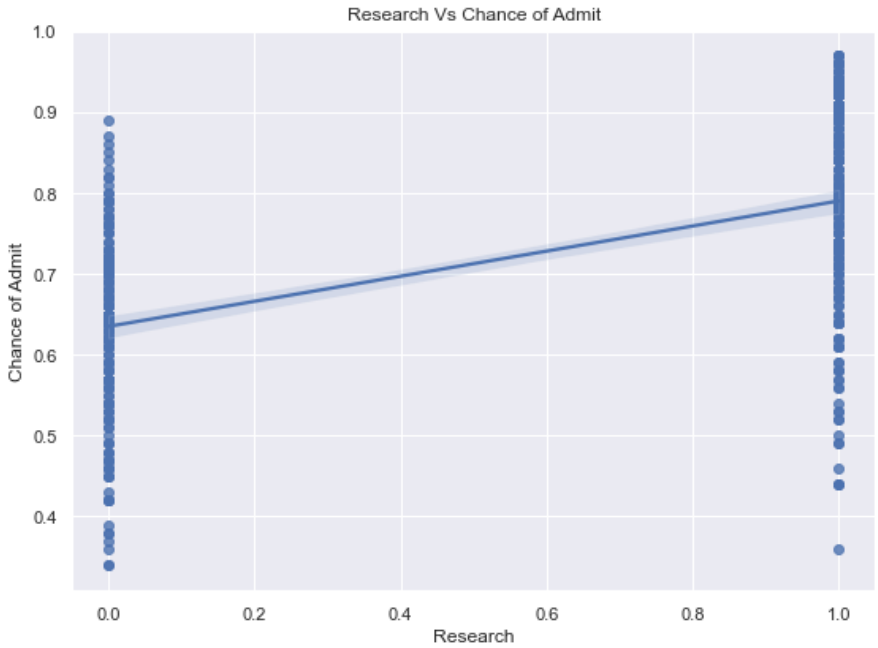

SOP Vs Chance of Admit
Chance of Admit
SOP
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


LOR Vs Chance of Admit
1.0


CGPA Vs Chance of Admit
1.0







3.3 Splitting data into independent and dependent features

In [203…
```
X1=dataset1.iloc[:,:-1]
```

```
y1=dataset1.iloc[:,-1]
X1.head()
```

Out[203]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0

In [204…
```
y1.head()
```

Out[204]:
```
0    0.92
1    0.76
2    0.72
3    0.80
4    0.65
Name: Chance of Admit , dtype: float64
```

In [205…
```
### random state train test split will be same with all people using random_state=10
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.25, random_state=10)
X_train1.head()
```

Out[205]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
324	315	104	3	3.0	2.5	8.33	0
252	318	100	2	2.5	3.5	8.54	1
441	332	112	1	1.5	3.0	8.66	1
427	310	101	3	3.5	5.0	8.65	1
70	332	118	5	5.0	5.0	9.64	1

In [206…
```
y_train1.head()
```

Out[206]:
```
324    0.67
252    0.71
441    0.79
427    0.71
70     0.94
Name: Chance of Admit , dtype: float64
```

In [207…
```
X_test1.head()
```

Out[207]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
151	332	116	5	5.0	5.0	9.28	1
424	325	114	5	4.0	5.0	9.46	1
154	326	108	3	3.0	3.5	8.89	0
190	324	111	5	4.5	4.0	9.16	1
131	303	105	5	5.0	4.5	8.65	0

In [208…
```
y_test1.head()
```

Out[208]:
```
151    0.94
424    0.91
154    0.80
190    0.90
131    0.77
Name: Chance of Admit , dtype: float64
```

In [209…
```
### both will have same shape
X_train1.shape, y_train1.shape
```

Out[209]:
```
((375, 7), (375,))
```

In [210…
```
### both will have same shape
X_test1.shape, y_test1.shape
```

Out[210]:
```
((125, 7), (125,))
```

3.4 Transforming data

In [211…
```
X_train1=scaler.fit_transform(X_train1)
X_train1
```

Out[211]:
```
array([[-0.10006876, -0.50453716, -0.0560763 , ..., -1.00222248,
        -0.36783128, -1.13427746],
       [ 0.16173904, -1.16205736, -0.93226844, ...,  0.07005549,
        -0.0297116 ,  0.8816185 ],
       [ 1.38350874,  0.81050323, -1.80846058, ..., -0.46608349,
         0.16349965,  0.8816185 ],
       ...,
       [ 0.07446977, -0.17577707, -0.0560763 , ..., -0.46608349,
        -1.07627251,  0.8816185 ],
       [-1.40910772, -1.16205736, -0.0560763 , ..., -0.46608349,
         0.16349965,  0.8816185 ],
       [-0.27460729, -0.83329726, -0.0560763 , ..., -1.00222248,
         0.19570152, -1.13427746]])
```

In [212…
```
X_test1=scaler.transform(X_test1)
X_test1
```

Out[212]:
```
array([[ 1.38350874,  1.46802342,  1.69630798,  1.66146257,  1.67847246,
         1.16175776,  0.8816185 ],
       [ 0.77262389,  1.13926332,  1.69630798,  0.66378689,  1.67847246,
         1.45157463,  0.8816185 ],
       [ 0.85989315,  0.15298303, -0.0560763 , -0.3338888 ,  0.07005549,
         0.53382121, -1.13427746],
       [ 0.68535462,  0.64612318,  1.69630798,  1.16262473,  0.60619448,
         0.96854651,  0.8816185 ],
       [-1.14729993, -0.34015711,  1.69630798,  1.66146257,  1.14233347,
         0.14739871, -1.13427746],
       [ 0.33627757, -0.50453716, -0.0560763 , -0.3338888 ,  0.07005549,
         0.29230715,  0.8816185 ],
       [-0.36187655, -0.34015711, -0.93226844, -1.33156448, -1.00222248,
        -0.17462003, -1.13427746],
       [ 1.38350874,  0.15298303,  1.69630798,  1.16262473,  0.60619448,
         0.74313339,  0.8816185 ],
       [-0.62368434, -0.50453716, -0.93226844, -1.33156448, -1.00222248,
        -0.48053784, -1.13427746],
       [-0.36187655, -0.17577707, -0.0560763 ,  0.66378689,  0.07005549,
         0.37281183,  0.8816185 ],
       [ 1.03443168,  1.46802342,  0.82011584,  1.66146257,  0.07005549,
         1.67698775,  0.8816185 ],
       [ 0.42354683,  0.81050323,  1.69630798,  1.66146257,  1.67847246,
         0.80753714,  0.8816185 ],
       [ 1.12170095,  0.64612318,  0.82011584,  1.16262473,  0.60619448,
         0.72703245,  0.8816185 ],
       [-0.36187655, -0.50453716, -0.0560763 ,  0.16494905,  0.07005549,
        -0.22292284, -1.13427746],
       [ 0.77262389,  0.48174313,  0.82011584,  0.16494905,  0.60619448,
         0.17960059,  0.8816185 ],
       [-0.01279949,  0.64612318,  0.82011584,  0.66378689,  1.67847246,
        -0.0297116 , -1.13427746],
       [ 1.38350874,  1.63240347,  0.82011584,  1.16262473,  0.60619448,
         0.87194089, -1.13427746],
       [ 1.99439359,  1.96116357,  1.69630798,  1.16262473,  0.60619448,
         1.83799712, -1.13427746],
       [ 0.16173904, -0.99767731,  1.69630798,  0.16494905,  1.67847246,
         0.3567109 ,  0.8816185 ],
       [-0.9727614 , -0.17577707, -0.93226844, -0.3338888 , -0.46608349,
        -0.64154721, -1.13427746],
       [-0.36187655, -0.66891721, -0.0560763 ,  0.16494905,  0.60619448,
         0.3567109 , -1.13427746],
       [ 0.07446977, -0.17577707, -1.80846058, -1.83040232,  0.07005549,
        -1.46269501,  0.8816185 ],
```

```
[-0.71095361, -0.34015711,  0.82011584, -0.3338888 , -1.00222248,
 -0.97966689,  0.8816185 ],
[ 0.77262389,  0.81050323,  0.82011584,  0.66378689,  1.14233347,
  0.98464745,  0.8816185 ],
[-1.58364625, -2.47709775, -1.80846058, -1.33156448, -1.53836147,
 -1.09237345, -1.13427746],
[ 1.470778  ,  1.79678352,  1.69630798,  1.66146257,  1.67847246,
  1.27446432,  0.8816185 ],
[-0.27460729, -0.50453716, -0.0560763 ,  0.66378689,  1.14233347,
  0.14739871, -1.13427746],
[-0.01279949, -1.49081745, -1.80846058, -1.83040232, -1.53836147,
 -1.81691563, -1.13427746],
[ 0.07446977, -0.34015711, -0.0560763 ,  0.16494905, -0.46608349,
  0.00249028, -1.13427746],
[ 0.2490083 ,  0.48174313, -0.0560763 , -0.3338888 , -1.00222248,
  0.37281183, -1.13427746],
[ 1.7325858 ,  1.79678352,  1.69630798,  1.16262473,  1.67847246,
  1.56428119,  0.8816185 ],
[ 1.03443168,  0.97488328,  0.82011584,  0.66378689, -1.00222248,
  0.34060996,  0.8816185 ],
[-1.67091552, -0.99767731, -0.0560763 , -1.33156448,  0.60619448,
 -1.43049313,  0.8816185 ],
[-0.44914581, -0.99767731, -0.93226844, -0.83272664,  0.07005549,
 -0.35173034,  0.8816185 ],
[ 0.51081609, -0.01139702, -0.0560763 ,  0.16494905,  0.07005549,
 -0.15851909,  0.8816185 ],
[ 1.12170095,  1.96116357,  0.82011584,  1.16262473,  1.14233347,
  0.96854651,  0.8816185 ],
[ 0.68535462,  0.48174313,  0.82011584, -0.3338888 ,  0.07005549,
  0.6626287 ,  0.8816185 ],
[-1.49637699, -0.83329726, -0.0560763 ,  0.66378689,  0.07005549,
  0.0990959 , -1.13427746],
[-0.71095361, -0.66891721, -0.93226844, -0.3338888 ,  0.07005549,
 -0.11021628, -1.13427746],
[-0.44914581, -0.83329726, -0.0560763 ,  1.16262473,  0.60619448,
  0.13129778,  0.8816185 ],
[ 1.38350874,  1.96116357,  0.82011584,  1.66146257,  1.14233347,
  1.09735401,  0.8816185 ],
[-0.18733802, -0.83329726, -0.93226844, -1.33156448, -1.00222248,
 -0.51273971, -1.13427746],
[-0.44914581, -0.34015711, -0.0560763 ,  0.16494905, -0.46608349,
 -0.17462003,  0.8816185 ],
[ 1.12170095,  0.97488328,  1.69630798,  1.66146257,  1.14233347,
  1.43547369,  0.8816185 ],
```

```
[ 1.7325858 ,  1.96116357,  1.69630798,  0.66378689,  0.07005549,
  1.99900649,  0.8816185 ],
[-1.49637699, -1.6551975 , -0.0560763 ,  1.66146257,  0.07005549,
 -1.44659407, -1.13427746],
[-1.06003067, -1.16205736, -0.93226844,  0.16494905, -0.46608349,
 -0.54494159, -1.13427746],
[ 0.77262389,  0.81050323,  0.82011584,  0.16494905,  0.07005549,
  0.58212402, -1.13427746],
[ 0.94716242, -0.50453716,  1.69630798, -0.3338888 ,  0.07005549,
  0.45331652,  0.8816185 ],
[-0.36187655, -0.66891721,  0.82011584,  0.66378689,  1.67847246,
 -0.20682191, -1.13427746],
[ 0.68535462, -0.01139702,  1.69630798,  0.16494905,  0.60619448,
  0.16349965,  0.8816185 ],
[ 0.2490083 ,  0.15298303, -0.93226844, -0.83272664, -0.46608349,
  0.32450902, -1.13427746],
[-0.27460729, -2.14833765, -0.93226844, -0.83272664, -2.07450046,
 -0.68985002, -1.13427746],
[-0.62368434, -0.17577707, -0.93226844, -0.83272664, -1.00222248,
 -0.89916221, -1.13427746],
[ 0.51081609, -0.34015711, -0.93226844, -0.3338888 , -0.46608349,
 -0.17462003,  0.8816185 ],
[ 0.42354683,  0.31736308,  0.82011584,  0.66378689,  0.60619448,
  0.19570152,  0.8816185 ],
[ 0.68535462,  1.13926332,  1.69630798,  1.66146257,  1.14233347,
  0.83973901,  0.8816185 ],
[-0.71095361, -0.99767731, -0.93226844, -0.3338888 ,  0.60619448,
 -1.06017158, -1.13427746],
[ 0.16173904,  0.31736308, -1.80846058,  0.16494905,  0.07005549,
  0.90414276, -1.13427746],
[ 1.29623948,  1.46802342,  1.69630798,  1.66146257,  1.67847246,
  1.32276713,  0.8816185 ],
[-0.18733802, -0.17577707, -0.93226844,  0.66378689,  0.07005549,
 -0.49663878, -1.13427746],
[ 0.42354683,  0.64612318,  1.69630798,  1.66146257,  1.67847246,
  1.43547369,  0.8816185 ],
[ 0.42354683,  0.48174313,  0.82011584,  0.16494905,  0.60619448,
 -0.3356294 ,  0.8816185 ],
[-1.84545405, -0.99767731, -0.93226844, -0.83272664, -1.53836147,
 -1.12457533, -1.13427746],
[ 0.42354683,  0.64612318,  0.82011584,  0.66378689,  0.60619448,
  0.6626287 ,  0.8816185 ],
[-1.40910772, -0.83329726, -0.93226844, -1.83040232, -1.53836147,
 -1.10847439, -1.13427746],
```

```
[ 0.59808536, -0.01139702,  0.82011584, -0.3338888 , -1.00222248,
 -0.12631722,  0.8816185 ],
[-0.71095361,  0.15298303,  0.82011584,  1.16262473,  1.67847246,
 -0.35173034, -1.13427746],
[-1.75818478, -0.99767731, -1.80846058, -0.83272664, -0.46608349,
 -1.4143922 , -1.13427746],
[ 0.68535462,  0.97488328,  0.82011584,  1.16262473,  0.60619448,
  0.37281183, -1.13427746],
[ 1.29623948,  1.46802342,  0.82011584,  1.16262473,  1.14233347,
  1.41937275,  0.8816185 ],
[-1.40910772, -0.99767731, -0.0560763 ,  0.16494905, -1.00222248,
 -1.09237345, -1.13427746],
[-0.62368434, -1.16205736, -0.93226844, -0.3338888 , -0.46608349,
 -0.73815283, -1.13427746],
[-0.18733802,  0.48174313, -0.0560763 ,  0.66378689,  0.60619448,
  0.38891277, -1.13427746],
[ 0.42354683,  0.31736308, -0.0560763 , -0.3338888 ,  0.60619448,
 -0.57714346,  0.8816185 ],
[-0.53641508, -0.50453716, -0.0560763 , -1.33156448,  0.07005549,
 -0.30342753, -1.13427746],
[-0.27460729, -0.17577707, -0.93226844, -0.83272664, -1.53836147,
 -0.20682191, -1.13427746],
[ 0.33627757,  2.12554362, -0.0560763 ,  0.66378689,  1.14233347,
  0.88804182, -1.13427746],
[ 0.42354683,  0.31736308,  0.82011584,  0.66378689,  0.60619448,
  0.9202437 ,  0.8816185 ],
[ 0.33627757,  0.15298303, -0.0560763 ,  0.16494905,  0.60619448,
 -0.19072097,  0.8816185 ],
[-0.36187655, -0.34015711, -0.93226844, -1.83040232, -0.46608349,
 -0.15851909, -1.13427746],
[-0.36187655, -0.50453716, -0.0560763 ,  0.16494905,  0.60619448,
 -0.75425377, -1.13427746],
[-0.18733802,  0.15298303, -0.0560763 ,  1.16262473,  0.07005549,
 -0.67374908, -1.13427746],
[-0.10006876, -0.99767731, -0.0560763 ,  0.16494905,  1.14233347,
  0.9202437 , -1.13427746],
[ 0.94716242,  0.97488328,  0.82011584,  0.16494905, -0.46608349,
  0.21180246,  0.8816185 ],
[ 0.68535462, -0.01139702,  0.82011584,  0.66378689,  1.14233347,
  0.50161933,  0.8816185 ],
[ 0.51081609,  1.13926332,  1.69630798,  1.16262473,  0.60619448,
  0.61432589,  0.8816185 ],
[-0.79822287, -0.99767731, -0.0560763 ,  0.66378689, -0.46608349,
 -0.57714346, -1.13427746],
```

```
[ 0.77262389, -0.17577707, -0.0560763 ,  0.16494905,  0.60619448,
 -0.25512472,  0.8816185 ],
[ 2.08166286,  2.12554362,  0.82011584,  1.66146257,  1.67847246,
  1.51597838,  0.8816185 ],
[ 0.51081609,  0.48174313,  0.82011584,  0.66378689,  1.67847246,
  0.9202437 ,  0.8816185 ],
[-0.27460729, -0.83329726, -0.0560763 , -1.33156448, -0.46608349,
 -0.4644369 , -1.13427746],
[-1.58364625, -1.32643741, -1.80846058, -1.83040232, -0.46608349,
 -1.76861281, -1.13427746],
[ 0.51081609,  0.81050323, -0.0560763 , -0.3338888 ,  0.60619448,
  0.0990959 ,  0.8816185 ],
[ 0.51081609, -0.50453716, -0.0560763 ,  0.16494905,  0.60619448,
  0.45331652,  0.8816185 ],
[-0.27460729, -1.49081745, -0.0560763 , -0.83272664,  1.14233347,
 -0.41613409,  0.8816185 ],
[ 0.77262389,  0.48174313, -0.93226844, -0.3338888 , -1.00222248,
  0.32450902,  0.8816185 ],
[ 1.29623948,  0.81050323,  1.69630798,  0.66378689,  1.67847246,
  1.99900649,  0.8816185 ],
[-1.75818478, -1.9839576 , -0.93226844, -0.3338888 , -1.53836147,
 -1.63980532,  0.8816185 ],
[-0.71095361,  0.31736308, -0.93226844, -0.3338888 ,  0.60619448,
 -0.17462003, -1.13427746],
[ 0.42354683,  0.31736308, -0.0560763 ,  0.16494905,  0.07005549,
  0.38891277,  0.8816185 ],
[ 1.90712433,  1.63240347,  0.82011584,  0.16494905,  1.14233347,
  1.45157463,  0.8816185 ],
[ 0.16173904, -0.01139702, -0.0560763 , -0.3338888 ,  0.07005549,
 -0.4644369 ,  0.8816185 ],
[-0.36187655, -0.01139702,  0.82011584,  1.16262473,  0.60619448,
  0.14739871,  0.8816185 ],
[ 1.29623948,  1.30364337,  1.69630798,  0.66378689,  0.07005549,
  1.41937275,  0.8816185 ],
[-1.67091552, -1.81957755, -0.93226844, -0.83272664, -2.07450046,
 -1.07627251, -1.13427746],
[ 0.42354683,  0.64612318, -0.0560763 ,  0.16494905,  0.60619448,
  0.43721558,  0.8816185 ],
[-2.28180037, -1.16205736, -1.80846058, -1.83040232, -1.53836147,
 -1.60760344, -1.13427746],
[ 0.85989315,  0.48174313, -0.0560763 ,  0.16494905,  0.07005549,
  0.32450902,  0.8816185 ],
[ 0.16173904,  0.31736308, -0.0560763 ,  0.16494905,  0.60619448,
  1.06515213,  0.8816185 ],
```

```
[-1.58364625, -1.6551975 , -0.93226844, -1.33156448, -0.46608349,
 -2.17113625, -1.13427746],
[ 0.2490083 ,  0.15298303, -0.0560763 , -0.3338888 ,  0.07005549,
 -0.0297116 ,  0.8816185 ],
[-0.88549214,  0.48174313, -0.93226844,  0.16494905,  0.60619448,
 -0.17462003, -1.13427746],
[-0.44914581, -0.66891721, -0.0560763 , -1.33156448,  0.60619448,
 -0.75425377, -1.13427746],
[ 0.94716242,  1.30364337,  0.82011584,  0.16494905,  0.60619448,
  0.93634464, -1.13427746],
[-0.71095361, -0.17577707, -0.0560763 ,  0.16494905, -1.00222248,
 -0.56104252,  0.8816185 ],
[ 0.51081609,  0.31736308,  1.69630798,  1.16262473,  0.07005549,
  0.38891277, -1.13427746],
[-1.58364625, -0.99767731, -0.93226844, -1.83040232, -1.53836147,
 -1.12457533, -1.13427746],
[ 0.68535462,  1.30364337, -0.0560763 ,  0.16494905, -0.46608349,
  0.32450902,  0.8816185 ],
[ 0.59808536, -0.01139702, -0.0560763 ,  0.16494905,  0.07005549,
 -0.01361066,  0.8816185 ],
[ 0.07446977, -1.16205736, -0.93226844, -0.3338888 , -1.00222248,
  0.01859122, -1.13427746],
[ 1.29623948,  1.63240347,  0.82011584,  1.16262473,  1.67847246,
  1.38717088,  0.8816185 ],
[ 0.59808536,  0.97488328, -0.0560763 ,  0.66378689,  0.60619448,
  0.51772027,  0.8816185 ],
[ 1.03443168,  0.48174313,  0.82011584,  0.66378689, -1.00222248,
  0.74313339,  0.8816185 ],
[ 0.59808536,  0.48174313,  0.82011584,  0.66378689,  1.67847246,
  0.51772027,  0.8816185 ]])
```

3.5 Building SVR Model, training and performance of Model

In [213…]
```
svr=SVR()
svr
```

Out[213]:
```
SVR()
```

In [214…]
```
svr.fit(X_train1,y_train1)
```

Out[214]:
```
SVR
SVR()
```

In [215…
```
svr_pred=svr.predict(X_test1)
svr_pred
```

Out[215]:
```
array([0.86448478, 0.86417389, 0.64993814, 0.85127578, 0.70546036,
       0.7574019 , 0.64562856, 0.8374946 , 0.60540491, 0.7499413 ,
       0.87245167, 0.80905978, 0.8501926 , 0.67828992, 0.79563367,
       0.73813638, 0.70599962, 0.74942901, 0.74397623, 0.62461368,
       0.74524207, 0.62023823, 0.59939887, 0.86498548, 0.47403454,
       0.86577212, 0.73739421, 0.5349166 , 0.66114659, 0.6681512 ,
       0.88063063, 0.75792351, 0.59757618, 0.659533  , 0.72195655,
       0.86222102, 0.81603033, 0.66700444, 0.66986721, 0.70308992,
       0.86541166, 0.59529725, 0.69816638, 0.87297663, 0.85464852,
       0.50589332, 0.56266899, 0.65426092, 0.77984391, 0.72964663,
       0.79144637, 0.68053493, 0.54289505, 0.60234504, 0.65093636,
       0.79311027, 0.83658814, 0.61545878, 0.68815676, 0.86796428,
       0.61411511, 0.82414663, 0.73731365, 0.50901921, 0.83202811,
       0.5044696 , 0.70643466, 0.71803499, 0.52797734, 0.69684074,
       0.89743609, 0.55063192, 0.57081914, 0.73056104, 0.69386262,
       0.65412584, 0.67213476, 0.73045228, 0.8521948 , 0.72469225,
       0.64563757, 0.66509614, 0.61956285, 0.7728156 , 0.76073394,
       0.82042555, 0.81901496, 0.60273855, 0.73071117, 0.85829731,
       0.83211108, 0.60919266, 0.52142804, 0.74445065, 0.77577819,
       0.70713967, 0.69439547, 0.86941895, 0.46678478, 0.69624509,
       0.76897481, 0.86613133, 0.68693638, 0.74092936, 0.86513894,
       0.4753147 , 0.77694577, 0.49680713, 0.76204858, 0.82616497,
       0.47006965, 0.72691457, 0.68481788, 0.64006911, 0.69851436,
       0.64063286, 0.69331741, 0.49445663, 0.74667101, 0.73464907,
       0.63246319, 0.88207478, 0.78364637, 0.79318457, 0.81335894])
```

In [218…
```
svr_r2_score=r2_score(y_test1, svr_pred)
print("Our Support Vector Regressor model has {} % accuracy".format(round(svr_r2_score*100,3)))
```

```
Our Support Vector Regressor model has 75.701 % accuracy
```

In [219…
```
adjusted_r2_score=1-((1-svr_r2_score)*(len(y_test1)-1)/(len(y_test1)-X_test1.shape[1]-1))
print("Adjusted R square accuracy is {} % ".format(round(adjusted_r2_score*100,3)))
```

```
Adjusted R square accuracy is 74.247 %
```