```
%r
library(SparkR)

data_r <- read.df("/FileStore/Day16_wine_quality.csv", source = "csv",
header="true")

display(data_r)
data_r <- as.data.frame(data_r)
```

And we can also do the same for Python:

```
import pandas as pd
data_py = pd.read_csv("/dbfs/FileStore/Day16_wine_quality.csv", sep=';')
```

We can use also Python to insert the data and get the dataset insight.

```
import matplotlib.pyplot as plt
import seaborn as sns
data_py = pd.read_csv("/dbfs/FileStore/Day16_wine_quality.csv", sep=',')
data_py.info()
```

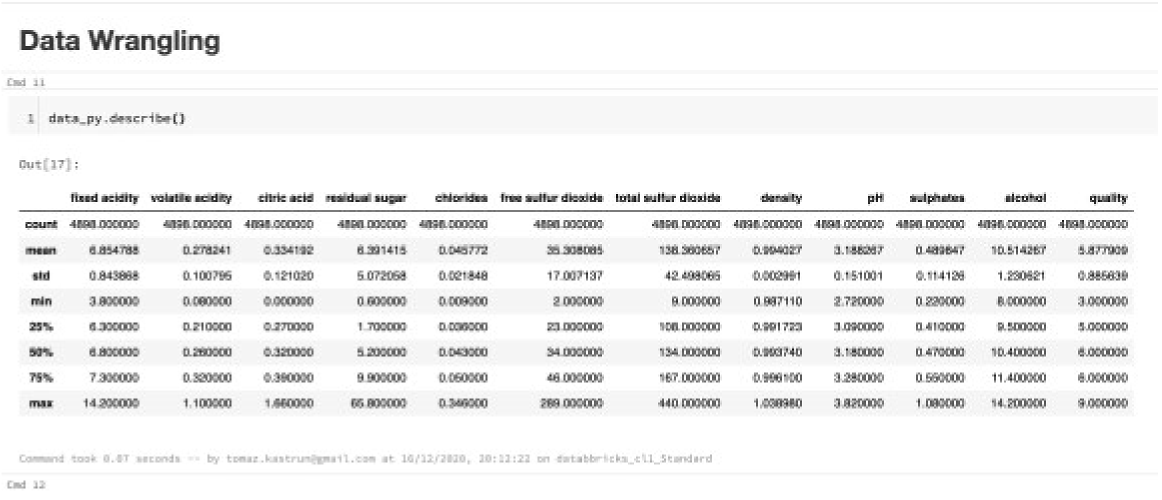
Importing also all other packages that will be relevant in following steps:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

2.Data wrangling

So let's continue using Python. You can get the sense of the dataset by using Python describe function:

```
data_py.describe()
```



Data Wrangling

Cmd 11

```
data_py.describe()
```

Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	5.877909
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621	0.885639
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	6.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000

```
Command took 0.07 seconds -- by tomaz.kastrun@gmail.com at 10/12/2020, 20:12:22 on databbricks_cl1_Standard
```

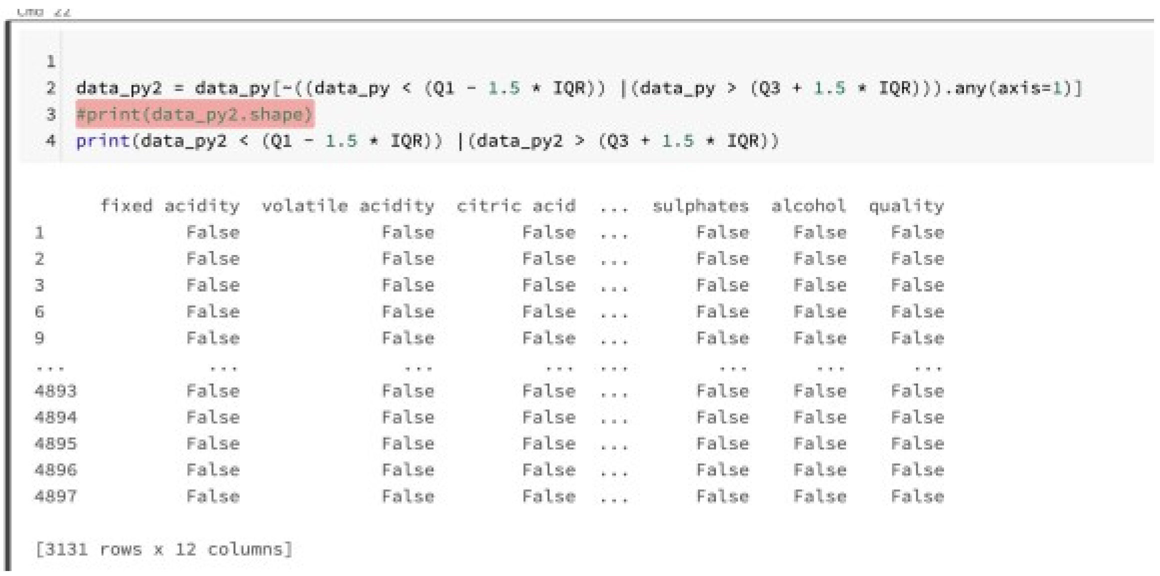
Cmd 12

And also work with duplicate values (remove them) and missing values (remove them or replace them with mean value):

```
#remove duplicates
sum(data_py.duplicated())
data_py.drop_duplicates(inplace=True)

#remove rows with empty values
data_py.isnull().sum(axis=0)
data_py.dropna(axis=0, how='any', inplace=True)

#fill the missing values with mean
data_py.fillna(0, inplace=True)
data_py['quality'].fillna(data_py['quality'].mean(), inplace=True)
data_py.apply(lambda x: x.fillna(x.mean(), inplace=True), axis=0)
```

You can also find and filter out the outlier by using IQR – Interquartile rang:

```
Q1 = data_py.quantile(0.25)
Q3 = data_py.quantile(0.75)
IQR = Q3 - Q1
data_py2 = data_py[~((data_py < (Q1 - 1.5 * IQR)) |(data_py > (Q3 + 1.5 *
IQR))).any(axis=1)]
#print(data_py2.shape)
print(data_py2 < (Q1 - 1.5 * IQR)) |(data_py2 > (Q3 + 1.5 * IQR))
```



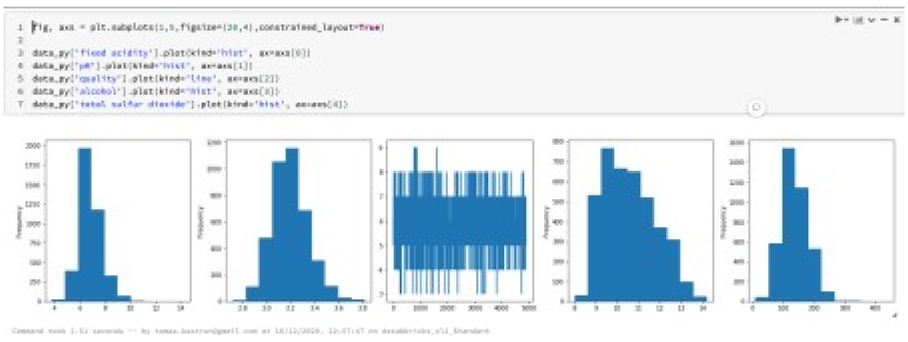
```

data_py2 = data_py[~((data_py < (Q1 - 1.5 * IQR)) |(data_py > (Q3 + 1.5 * IQR))).any(axis=1)]
#print(data_py2.shape)
print(data_py2 < (Q1 - 1.5 * IQR)) |(data_py2 > (Q3 + 1.5 * IQR))
```

```
      fixed acidity  volatile acidity  citric acid  ...  sulphates  alcohol  quality
1             False             False        False  ...      False    False    False
2             False             False        False  ...      False    False    False
3             False             False        False  ...      False    False    False
6             False             False        False  ...      False    False    False
9             False             False        False  ...      False    False    False
...             ...               ...          ...  ...        ...      ...      ...
4893          False             False        False  ...      False    False    False
4894          False             False        False  ...      False    False    False
4895          False             False        False  ...      False    False    False
4896          False             False        False  ...      False    False    False
4897          False             False        False  ...      False    False    False

[3131 rows x 12 columns]
```

3.Exploring dataset

We can check the distribution of some variables and best way is to show it with graphs:

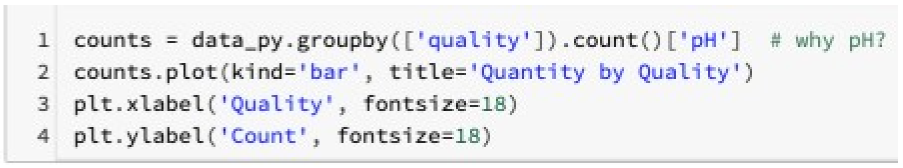
```
fig, axs = plt.subplots(1,5,figsize=(20,4),constrained_layout=True)

data_py['fixed acidity'].plot(kind='hist', ax=axs[0])
data_py['pH'].plot(kind='hist', ax=axs[1])
data_py['quality'].plot(kind='line', ax=axs[2])
data_py['alcohol'].plot(kind='hist', ax=axs[3])
data_py['total sulfur dioxide'].plot(kind='hist', ax=axs[4])
```



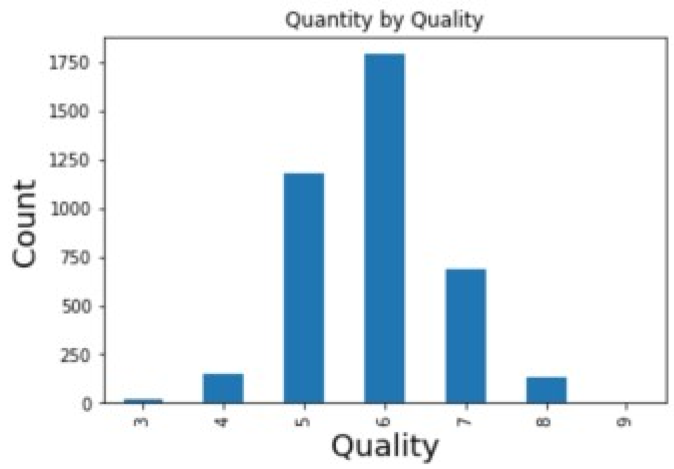
```
fig, axs = plt.subplots(1,5,figsize=(20,4),constrained_layout=True)

data_py['fixed acidity'].plot(kind='hist', ax=axs[0])
data_py['pH'].plot(kind='hist', ax=axs[1])
data_py['quality'].plot(kind='line', ax=axs[2])
data_py['alcohol'].plot(kind='hist', ax=axs[3])
data_py['total sulfur dioxide'].plot(kind='hist', ax=axs[4])
```

Adding also a plot of counts per quality:

```
counts = data_py.groupby(['quality']).count()['pH']  # pH or anything else - 
just for count
counts.plot(kind='bar', title='Quantity by Quality')
plt.xlabel('Quality', fontsize=18)
plt.ylabel('Count', fontsize=18)
```

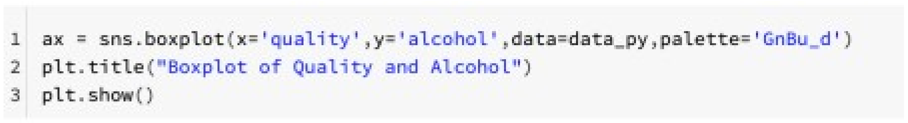


```
counts = data_py.groupby(['quality']).count()['pH']  # why pH?
counts.plot(kind='bar', title='Quantity by Quality')
plt.xlabel('Quality', fontsize=18)
plt.ylabel('Count', fontsize=18)
```

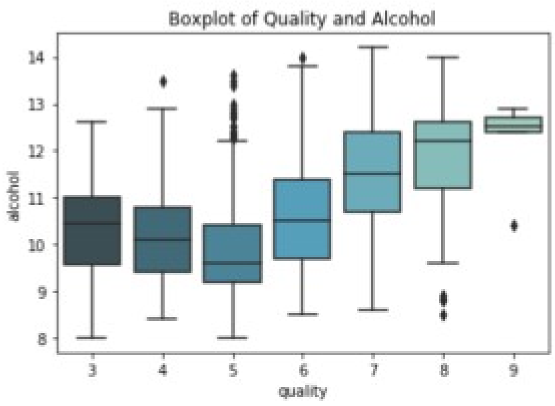


Adding some boxplots will also give a great understanding of the data and statistics of particular variable. So, let’s take pH and Quality

```
sns.boxplot(x='quality',y='pH',data=data_py,palette='GnBu_d')
plt.title("Boxplot - Quality and pH")
plt.show()
```



```
ax = sns.boxplot(x='quality',y='alcohol',data=data_py,palette='GnBu_d')
plt.title("Boxplot of Quality and Alcohol")
plt.show()
```

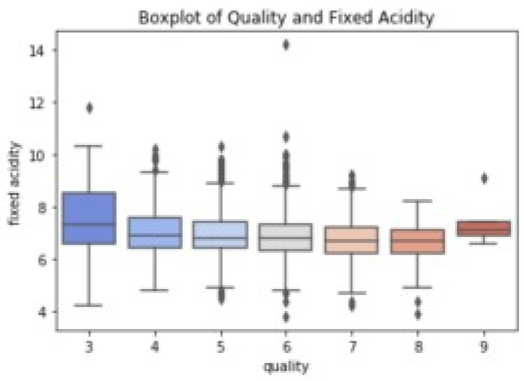


```
ommand took 0.70 seconds -- by tomaz.kastrun@gmail.com at 16/12/2020, 20:35:23 on databbricks
```

or quality with fixed acidity:

```
sns.boxplot(x="quality",y="fixed acidity",data=data_py,palette="coolwarm")
plt.title("Boxplot of Quality and Fixed Acidity")
plt.show()
```

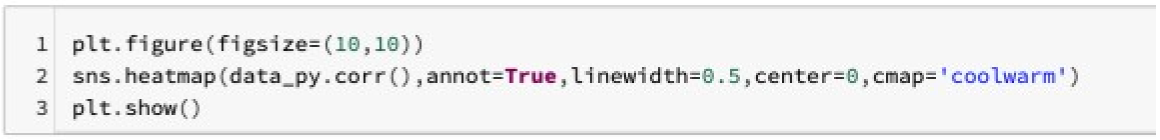
```
sns.boxplot(x="quality",y="fixed acidity",data=data_py,palette="coolwarm")
plt.title("Boxplot of Quality and Fixed Acidity")
plt.show()
```



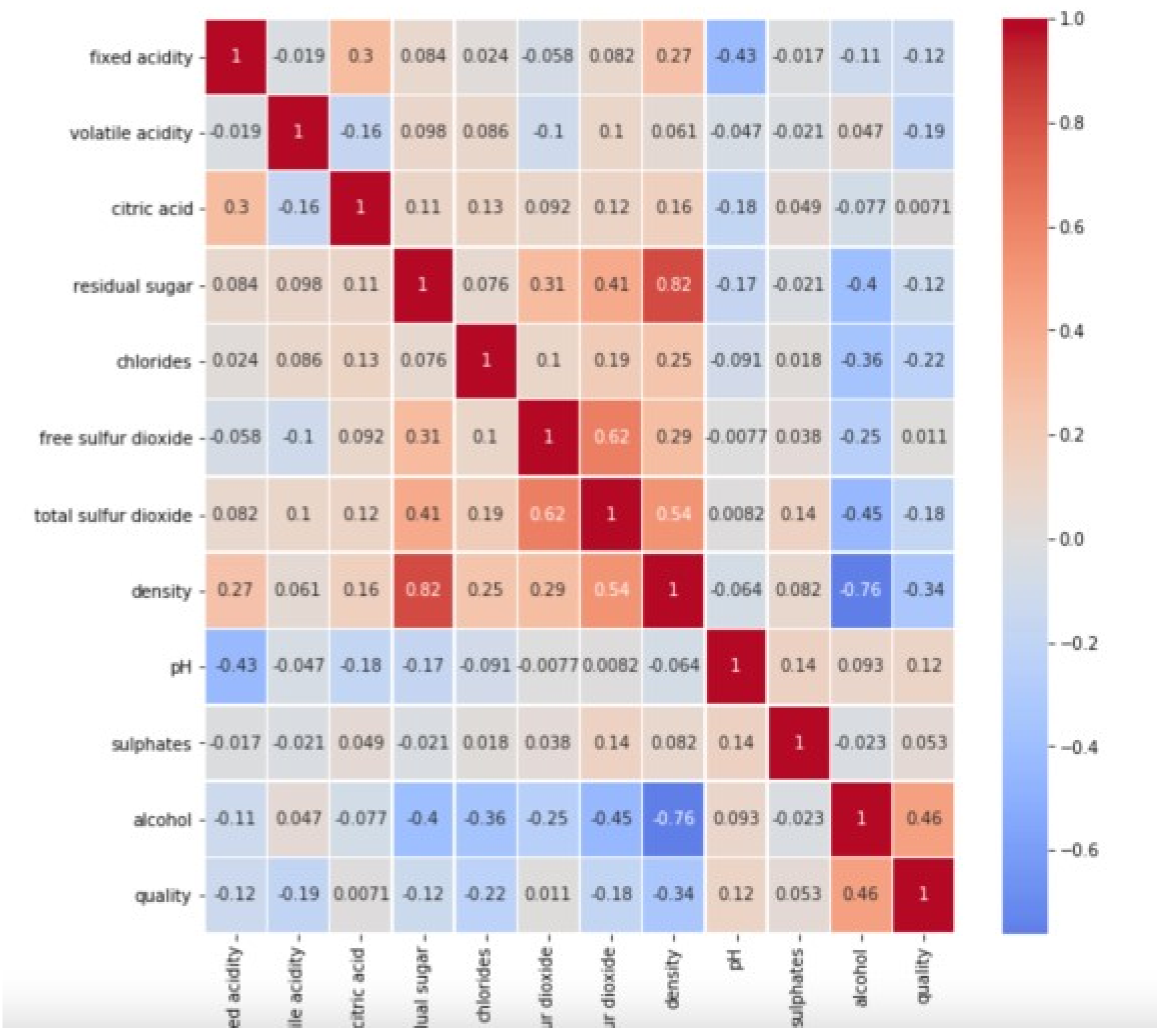
```
Command took 0.37 seconds -- by tomaz.kastrun@gmail.com at 16/12/2020, 20:36:54 on databbricks_cl
```

And also add some correlation among all the variables in dataset:

```
plt.figure(figsize=(10,10))
sns.heatmap(data_py.corr(),annot=True,linewidth=0.5,center=0,cmap='coolwarm')
plt.show()
```



```
plt.figure(figsize=(10,10))
sns.heatmap(data_py.corr(),annot=True,linewidth=0.5,center=0,cmap='coolwarm')
plt.show()
```



4.Modeling

We will split the dataset into Y-set – our predict variable and X-set – all the other variables. After that, we will do splitting of the y-set and x-set into train and test subset.

```
X = data_py.iloc[:,:11].values
Y = data_py.iloc[:,-1].values

#Splitting the dataset into training and test set
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_
size=0.25,random_state=0)
```

We will also to the feature scaling

```
#Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

And get the general understanding of explained variance:

```
# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 3)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
```

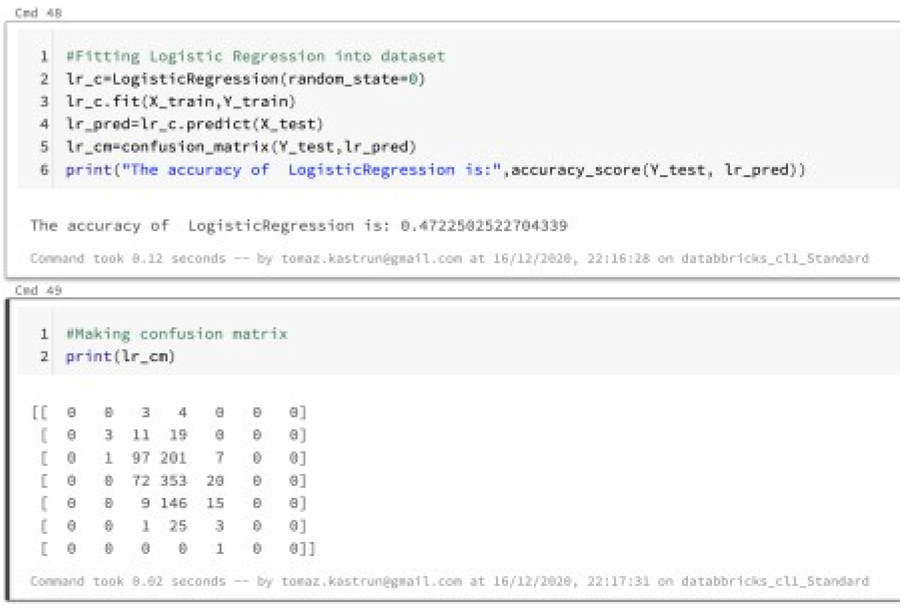
You will see, that three variables together contribute more than 50% of all variance of the model.

Based on the train and test test, let us now fit the different type of model into the dataset. Using Logistic regression:

```
#Fitting Logistic Regression into dataset
lr_c=LogisticRegression(random_state=0)
lr_c.fit(X_train,Y_train)
lr_pred=lr_c.predict(X_test)
lr_cm=confusion_matrix(Y_test,lr_pred)
print("The accuracy of  LogisticRegression is:",accuracy_score(Y_test,
lr_pred))
```

and create a confusion matrix to see the correctly predicted values per category.

```
#Making confusion matrix
print(lr_cm)
```



Cmd 48

```
#Fitting Logistic Regression into dataset
lr_c=LogisticRegression(random_state=0)
lr_c.fit(X_train,Y_train)
lr_pred=lr_c.predict(X_test)
lr_cm=confusion_matrix(Y_test,lr_pred)
print("The accuracy of  LogisticRegression is:",accuracy_score(Y_test, lr_pred))
```

```
The accuracy of  LogisticRegression is: 0.4722502522704339
```

Command took 0.12 seconds -- by tomaz.kastrun@gmail.com at 16/12/2020, 22:16:28 on databbricks_cl1_Standard

Cmd 49

```
#Making confusion matrix
print(lr_cm)
```

```
[[  0   0   3   4   0   0   0]
 [  0   3  11  19   0   0   0]
 [  0   1  97 201   7   0   0]
 [  0   0  72 353  20   0   0]
 [  0   0   9 146  15   0   0]
 [  0   0   1  25   3   0   0]
 [  0   0   0   0   1   0   0]]
```

Command took 0.02 seconds -- by tomaz.kastrun@gmail.com at 16/12/2020, 22:17:31 on databbricks_cl1_Standard

I will repeat this for the following algorithms: SVM, RandomForest, KNN, Naive Bayes and I will make a comparison at the end.

SVM

```
#Fitting SVM into dataset
cl = SVC(kernel="rbf")
cl.fit(X_train,Y_train)
svm_pred=cl.predict(X_test)
svm_cm = confusion_matrix(Y_test,cl.predict(X_test))
print("The accuracy of  SVM is:",accuracy_score(Y_test, svm_pred))
```

RandomForest

```
#Fitting Randomforest into dataset
rdf_c=RandomForestClassifier(n_estimators=10,criterion='
entropy',random_state=0)
rdf_c.fit(X_train,Y_train)
rdf_pred=rdf_c.predict(X_test)
rdf_cm=confusion_matrix(Y_test,rdf_pred)
print("The accuracy of RandomForestClassifier is:",accuracy_score(rdf_pred,
Y_test))
```

KNN

```
#Fitting KNN into dataset
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,Y_train)
knn_pred=knn.predict(X_test)
knn_cm=confusion_matrix(Y_test,knn_pred)
print("The accuracy of KNeighborsClassifier is:",accuracy_score(knn_pred,
Y_test))
```

and Naive Bayes

```
#Fitting Naive bayes into dataset
gaussian=GaussianNB()
gaussian.fit(X_train,Y_train)
bayes_pred=gaussian.predict(X_test)
bayes_cm=confusion_matrix(Y_test,bayes_pred)
print("The accuracy of naives bayes is:",accuracy_score(bayes_pred,Y_test))
```

And the accuracy for all the model fitting is the following:

- LogisticRegression is: 0.4722502522704339
- SVM is: **0.48335015136226034**
- KNeighborsClassifier is: 0.39455095862764883
- naives bayes is: 0.46316851664984865

It is clear which model would give improvements,