TO MODEL WINE QUALITY BASED ON PHYSICOCHEMICAL TESTS

GIT HUBS: https://github.com/ManikantaMarreddy/wine_quality_accuracy

INTRODUCTION:

Abstract: Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests (see [Cortez et al., 2009], [Web Link]).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

Attribute Information:

Input variables (based on physicochemical tests):
1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
Output variable (based on sensory data):
12 - quality (score between 0 and 10)

EXPLORATORY DATA ANALYSIS:

IMPORTING REQUIRED LIBRARIES:

```
import pandas as pd              # for data manipulation
import numpy as np               # for numerical calculation
import matplotlib.pyplot as plt  # for data visualization
import seaborn as sns            # for advanced data visulization
from sklearn import metrics      #for matcine learning algorithms
from scipy import stats          # for scientific calculations
```

LOADING THE DATA SETS:

Loading red wine data:

Loading the red wine data and creating a new column (wine_type) and update the column values with 1. This wine_type column is to identify the type of wine.

```
data1=pd.read_csv("C:\\Users\\Manikanta Marreddy\\Desktop\\winequality\\New folde
r\\winequality-red.csv",sep=';')
data1['wine_type'] = 1
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 13 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
 12  wine_type             1599 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 162.5 KB
```

Loading white wine data

Loading the white wine data and creating a new column (wine_type) and update the column values with 2.

```
#loading white wine data set
data2 = pd.read_csv("C:\\Users\\Manikanta Marreddy\\Desktop\\winequality\\New fol
der\\winequality-white.csv",sep=';')
data2['wine_type'] = 2    #creating another column for red wine and placing 2
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64  
 12  wine_type             4898 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 497.6 KB
```

MERGING TWO DATA SETS:

We are having same column names in both the data sets. So it is easy to combine the data using concat.

The combined data is stored in data.

```
p = [data1,data2]
data = pd.concat(p)  #using concat
#data.head()


print(data.shape)

data = data.iloc[ :,[12,0,1,2,3,4,5,6,7,8,9,10,11]]
data.head()
```

```
(6497, 13)
```

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

CHECKING FOR NULL VALUES:

If there are any null values in the given data we need to fill the null values with logical values using mean, median for continuous data and mode for categorical data.

Here, there are no null values in the given data sets. So we can proceed to next step.

```
# checking for null valued
data.isnull().sum()
```

```
wine_type               0
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
```

IDENTIFYING OUTLIERS:

Describe is used to calculate the mean, standard deviation, 25%, 75%, 50% and minimum and maximum values for all the column attributes.

Let's have a look at the 'residual sugar' column. Count says there are 6497 rows in this column. While mean and std means respectively the average and standard deviation values of the column, 25% of the values are under 1.80 and 75% of them are under 8.10. The interesting thing is while the average value is 5.44 and minimum value is 0.60, the maximum value is 65.80. It looks like an outlier.

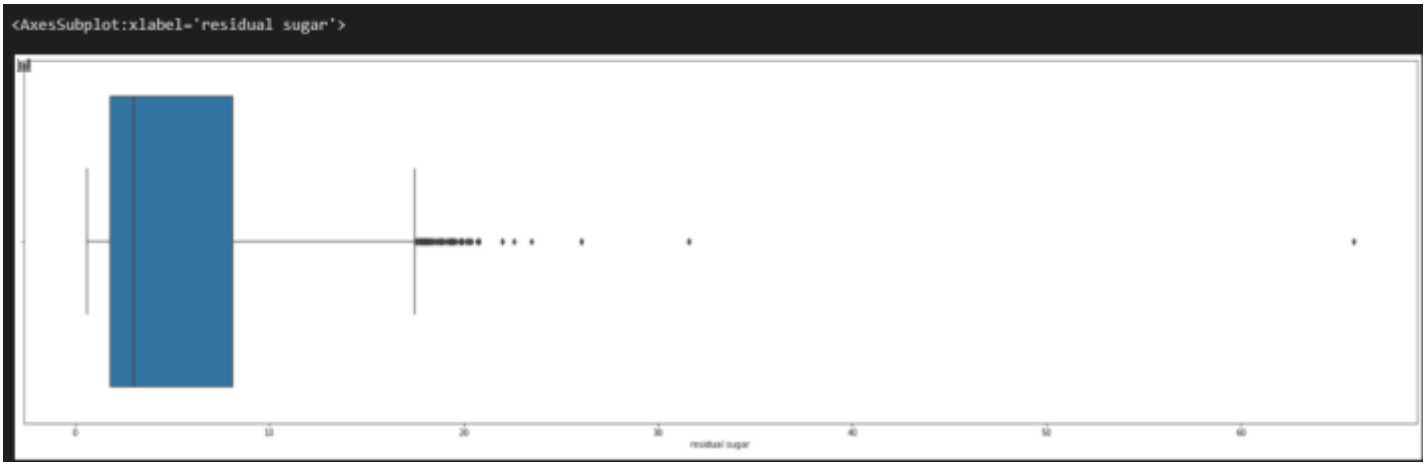
```
data.describe()
```

	wine_type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000
mean	1.753886	7.215307	0.339666	0.318633	5.443235	0.056034	30.525319	115.744574	0.994697	3.218501	0.531268	10.491801	5.818378
std	0.430779	1.296434	0.164636	0.145318	4.757804	0.035034	17.749400	56.521855	0.002999	0.160787	0.148806	1.192712	0.873255
min	1.000000	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	2.000000	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.000000
50%	2.000000	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
75%	2.000000	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.000000
max	2.000000	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

IDENTIFYING OUTLIERS USING BOX PLOT:

I used boxplot to visualize the distribution of the values in 'residual sugar' column to get a better insight. The actual maximum value is around 18 and the values bigger than that are outliers since they are not included in the box of observation.

```
plt.figure(figsize=(30,8))
sns.boxplot(data['residual sugar'])  #using boxplot
```



USING Z-SCORE REMOVING OUTLIERS:

It is important to remove outliers because they would likely affect the performance of machine learning models. But 30% of our dataset are outliers. Then it may not be wise to remove them all because probably there is something more going on and it needs to be inspected further. In order to find and remove the outliers, I used z-score.

After using Z-score the shape of the data set is reduced to (5989, 1)

```
# remove outliers from the data using z-score function
z = np.abs(stats.zscore(data))
data = data[(z<3).all(axis = 1)]    #removing extreme outlers only (z<3)
data.shape
```

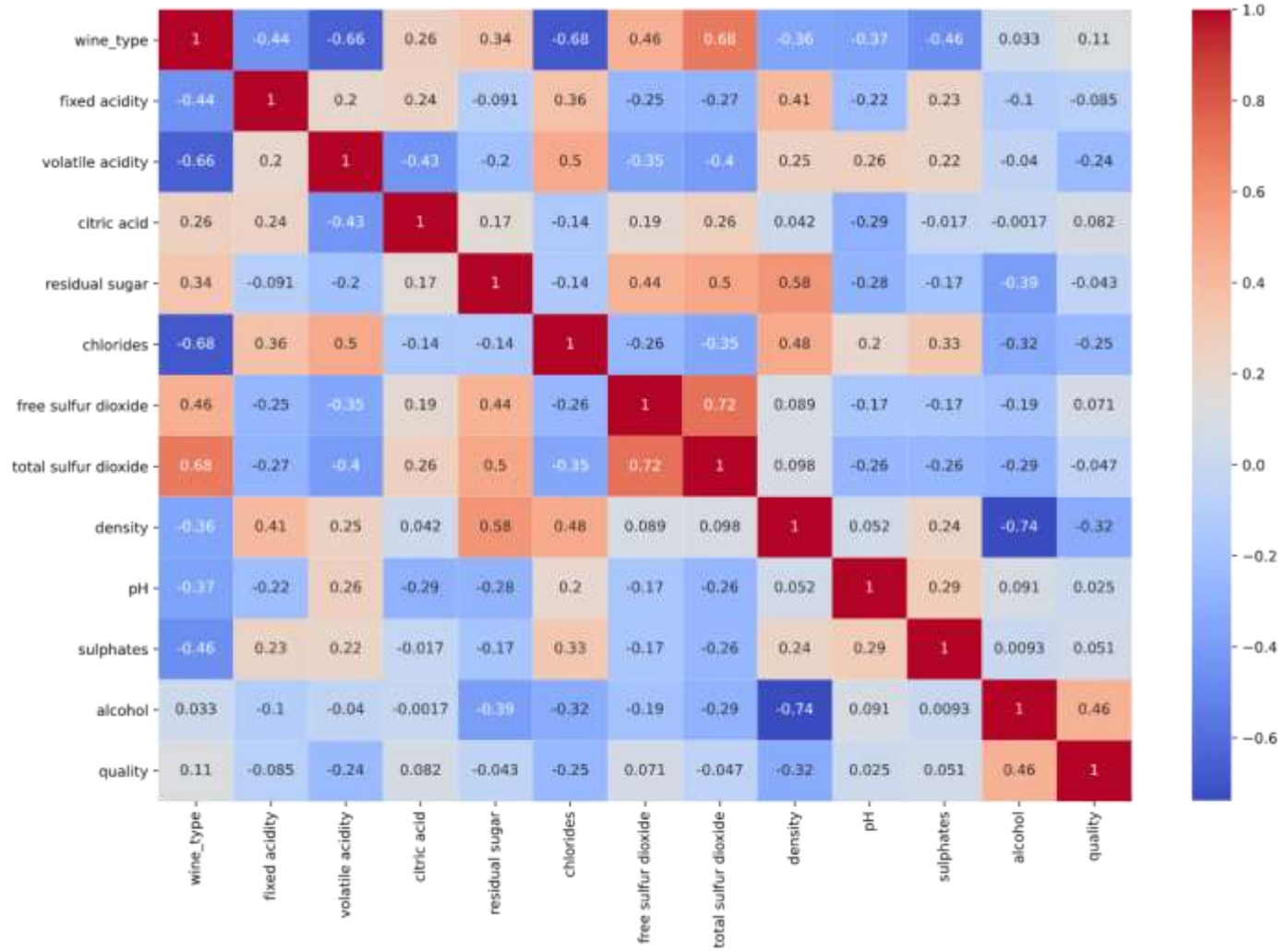
```
(5989, 13)
```

HEAT MAP:

Mainly heat map is used to find the correlation between the attributes. If you observe the correlation between the input variables and target variable (quality) is strong.

```
plt.subplots(figsize = (15,10))
sns.heatmap(data.corr(),annot =True,cmap='coolwarm')  #using correlation coeffici
ent

#by observing the heat map we can say that almost all input variables are strongl
y correlated with target variable.
#by using feature selection (backward method)
```



CHECKING FOR DATA IMBALANCE:

I wanted to check the class imbalance. And it seems like there is a high class imbalance where the minority classes are less represented than the majority classes. This will be a crucial part of the modeling in the later steps.

```
# checking the class imbalance
(data['quality'].value_counts()/(data.shape[0]))*100
```

```
6    44.014026
5    32.693271
7    17.148105
4     3.072299
8     3.072299
Name: quality, dtype: float64
```

DATA PRE-PROCESSING:

In this step of the analysis I defined the features to train and test the machine learning model and the target to predict which is 'quality'. And then I did standardization(also called z-score normalization) for the features because different scales of features may impact the performance of the machine learning models. For this purpose, I used StandardScaler() function defined in Scikit-learn. And finally I split the dataset into training and test sets 80% and 20% respectively.

FEATURE SELECTION:

```
#Adding constant column of ones, mandatory for sm.OLS model
import statsmodels.api as sm
#import python-constraint
X_1 = sm.tools.tools.add_constant(x)
#Fitting sm.OLS model
model = sm.OLS(y,X_1).fit()
model.pvalues

# here all variables are stronly correlated to target variable(quality).
```

```
array([7.60740359e-15, 5.83755920e-11, 1.05149694e-10, 7.42556202e-55,
       7.88615882e-02, 2.85692040e-26, 4.64133356e-01, 3.41933578e-15,
       3.22404020e-04, 1.83716702e-14, 1.10124460e-11, 1.03607162e-20,
       2.81171376e-18])
```

SPLITTING INPUT AND TARGET VARIABLES:

```
#splitting the data(variables) in to input and target variables
x = np.asarray(data.iloc[:,:-1])

y = np.asarray(data['quality'])

# x = data.drop('quality',1)
# y = data['quality']
```

STANDARDIZING INPUT VARIABLES:

```
# standardizing the data set
from sklearn import preprocessing
x = preprocessing.StandardScaler().fit(x).transform(x)
```

SPLITTING INTO TRAIN AND TEST DATA SETS:

```
#splitting the data into training and testing data sets
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_state=
0)
print('training dataset :' ,x_train.shape ,y_train.shape)

print('testing dataset :' ,x_test.shape ,y_test.shape)
```

```
training dataset : (4791, 12) (4791,)
testing dataset : (1198, 12) (1198,)
```

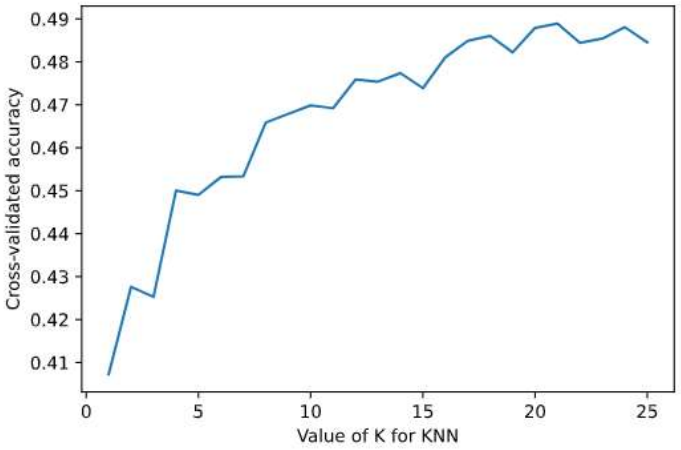
VALIDATION AND MODEL SELECTION:

KNN ACCURACY PLOT:

I started with K-Nearest Neighbors classification algorithm. What this algorithm does is that it takes a data point and selects K number of observations in the training data which are the nearest to the data point and then predicts the response of the data point regarding the most popular response value from the K-nearest neighbors.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
# Number of k from 1 to 26
k_range = range(1, 26)
k_scores= []
# Calculate cross validation score for every k number from 1 to 26
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
# It’s 10 fold cross validation with ‘accuracy’ scoring


%matplotlib inline
# Plot accuracy for every k number between 1 and 26
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-validated accuracy')
```



CROSS VALIDATION OF KNN:

```
# Train the model and predict for k=19
knn = KNeighborsClassifier(n_neighbors=19)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
# classification report for test set
print(metrics.classification_report(y_test, y_pred, digits=3, zero_division = 1))
# Calculate cv score with 'accuracy' scoring and 10 folds
accuracy = cross_val_score(knn, x, y, scoring = 'accuracy',cv=10)
print('cross validation score',accuracy.mean())
# Calculate cv score with 'roc_auc_ovr' scoring and 10 folds
accuracy = cross_val_score(knn, x, y, scoring = 'roc_auc_ovr',cv=10)
print('cross validation score with roc_auc',accuracy.mean())
# Calculate roc_auc score with multiclass parameter
print('roc_auc_score',roc_auc_score(y_test,knn.predict_proba(x_test), multi_class
='ovr'))
```

```
              precision    recall  f1-score   support

           4      0.333     0.029     0.053        35
           5      0.616     0.650     0.632       377
           6      0.589     0.680     0.631       543
           7      0.515     0.429     0.468       203
           8      0.000     0.000     0.000        40

    accuracy                          0.586      1198
   macro avg      0.411     0.357     0.357      1198
weighted avg      0.558     0.586     0.566      1198

cross validation score 0.4822192505904489
cross validation score with roc_auc 0.6975580970843452
roc_auc_score 0.7992712914875132
```

When I look at the classification report I immediately see that classes 8 have not been taken into consideration when training because their recall results are zero. This means, of all class 8 members, it did not predict any of them correctly. So, it wouldn't be a good model for our dataset.

LOGISTIC REGRESSION:

In order to use it as a multi-class classification algorithm, I used multi_class='multinomial', solver ='newton-cg' parameters.

And considering it's a multi-class classification problem, I used 'roc_auc_ovr' scoring parameter instead of 'accuracy' when calculating cross validation score. I also calculated roc_auc_score with multi_class='ovr' parameter. I will explain these later in conclusion.

```
#LOISTIC REGRESSION MODEL

from sklearn.linear_model import LogisticRegression
# Train and fit model
logreg = LogisticRegression(multi_class='multinomial',solver ='newton-cg')
logreg.fit(x_train, y_train)
# Predict out-of-sample test set
y_pred = logreg.predict(x_test)
# classification report
print(metrics.classification_report(y_test, y_pred, digits=3, zero_division = 1))
print('accuracy',accuracy_score(y_test, y_pred))
# Calculate cv score with 'roc_auc_ovr' scoring and 10 folds
accuracy = cross_val_score(logreg, x, y, scoring = 'roc_auc_ovr',cv=10)
print('cross validation score with roc_auc',accuracy.mean())
# Calculate roc_auc score with multiclass parameter
print('roc_auc_score',roc_auc_score(y_test,logreg.predict_proba(x_test), multi_cl
ass='ovr'))
```

```
              precision    recall  f1-score   support

           4      0.500     0.029     0.054        35
           5      0.582     0.650     0.614       377
           6      0.549     0.687     0.610       543
           7      0.510     0.241     0.328       203
           8      1.000     0.000     0.000        40

    accuracy                          0.558      1198
   macro avg      0.628     0.321     0.321      1198
weighted avg      0.567     0.558     0.527      1198

accuracy 0.5575959933222037
cross validation score with roc_auc 0.7506892285656838
roc_auc_score 0.7651649237431469
```

ADDING PLOYNOMIAL FEATURES TO LOGISTIC REGRESSION:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
# Add polynomial features to the logistic regression model
def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
    LogisticRegression(multi_class='multinomial',solver ='newton-cg', **kwargs))
```

3RD DEGREE:

Now I tried adding 3rd degree polynomial features to the logistic regression model.

```
# Train and fit the 3rd degree polynomial regression model
poly = PolynomialRegression(3)
poly.fit(x_train,y_train)
# Test out-of-sample test set
y_pred = poly.predict(x_test)
# Classification report
print(metrics.classification_report(y_test, y_pred, digits=3))
# Calculate cv score with 'roc_auc_ovr' scoring and 10 folds
accuracy = cross_val_score(poly, x, y, scoring = 'roc_auc_ovr',cv=10)
print('cross validation score with roc_auc_ovr scoring',accuracy.mean())
# Calculate roc_auc score with multiclass parameter
print('roc_auc_score',roc_auc_score(y_test,poly.predict_proba(x_test), multi_clas
s='ovr'))
```

```
              precision    recall  f1-score   support

           4      0.250     0.086     0.128        35
           5      0.591     0.613     0.602       377
           6      0.559     0.610     0.583       543
           7      0.494     0.419     0.453       203
           8      0.419     0.325     0.366        40

    accuracy                          0.553      1198
   macro avg      0.463     0.410     0.426      1198
weighted avg      0.544     0.553     0.546      1198

cross validation score with roc_auc_ovr scoring 0.6953083890028516
roc_auc_score 0.7822519328531152
```

After doing 3rd degree polynomial to logistic regression, we got lower value for roc_auc_ovr scoring (cross validation score) compared to before. So let's try it using decision tree model.

DECISION TREE:

```
from sklearn.tree import DecisionTreeClassifier
```

```
# Train and fit the Decision Tree Classification model
tree = DecisionTreeClassifier(random_state=0)
tree.fit(x_train, y_train)
# Evaluate the model with out-of-sample test set
y_pred = tree.predict(x_test)
# Classification report
print(metrics.classification_report(y_test, y_pred.round(), digits=3))
# Calculate cv score with 'roc_auc_ovr' scoring and 10 folds
accuracy = cross_val_score(tree, x, y,scoring = 'roc_auc_ovr',cv=10)
print('cross validation score with roc_auc_ovr scoring',accuracy.mean())
# Calculate roc_auc score with multiclass parameter
print('roc_auc_score',roc_auc_score(y_test,tree.predict_proba(x_test), multi_clas
s='ovr'))
```

```
              precision    recall  f1-score   support

           4      0.333     0.229     0.271        35
           5      0.639     0.645     0.642       377
           6      0.634     0.615     0.624       543
           7      0.535     0.571     0.552       203
           8      0.420     0.525     0.467        40

    accuracy                          0.603      1198
   macro avg      0.512     0.517     0.511      1198
weighted avg      0.603     0.603     0.602      1198

cross validation score with roc_auc_ovr scoring 0.5505786227086912
roc_auc_score 0.6982829224442457
```

When I applied it to my dataset, there is an increase in recall results but the cross validation score decreased. So let's go for another model such as random forest model.

RANDOM FOREST:

```
from sklearn.ensemble import RandomForestClassifier
# Train and fit the Random Forest Classification model
forest = RandomForestClassifier(n_estimators=100,random_state = 0)
forest.fit(x_train, y_train)
# Test out-of-sample test set
y_pred = forest.predict(x_test)
# Classification report
print(metrics.classification_report(y_test, y_pred.round(), digits=3))
# Calculate cv score with 'roc_auc_ovr' scoring and 10 folds
accuracy = cross_val_score(forest, x, y,scoring = 'roc_auc_ovr',cv=10)
print('cross validation score with roc_auc_ovr scoring',accuracy.mean())
# Calculate roc_auc score with multiclass parameter
print('roc_auc_score',roc_auc_score(y_test,forest.predict_proba(x_test), multi_cl
ass='ovr'))
```

```
              precision    recall  f1-score   support

           4      1.000     0.143     0.250        35
           5      0.731     0.727     0.729       377
           6      0.651     0.775     0.708       543
           7      0.693     0.512     0.589       203
           8      0.857     0.450     0.590        40

    accuracy                          0.686      1198
   macro avg      0.786     0.521     0.573      1198
weighted avg      0.700     0.686     0.677      1198

cross validation score with roc_auc_ovr scoring 0.7392343125072626
roc_auc_score 0.8671299619547327
```

By seeing above output, The roc_auc_score is good, the cross-validation score is the best so far and there are some recall results even for the minority classes. But it's not enough yet. **So one of the things that can be done to increase recall is oversampling the minority classes. For this purpose I used the random forest algorithm with SMOTE algorithm implementation.**

SMOTE:

```
# Import SMOTE module
from imblearn.over_sampling import SMOTE
# Create model and fit the training set to create a new training set
sm = SMOTE(random_state = 2)
x_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel())
# Create random forest model
forest = RandomForestClassifier(n_estimators=100,random_state = 0)
# Fit the model to the new train set
forest.fit(x_train_res, y_train_res.ravel())
# # Test out-of-sample test set
y_pred = forest.predict(x_test)
# Classification report
print(metrics.classification_report(y_test, y_pred.round(), digits=3))
# Calculate cv score with 'roc_auc_ovr' scoring and 10 folds
accuracy = cross_val_score(forest, x, y,scoring = 'roc_auc_ovr',cv=10)
print('cross validation score with roc_auc_ovr scoring',accuracy.mean())
# Calculate roc_auc score with multiclass parameter
print('roc_auc_score',roc_auc_score(y_test,forest.predict_proba(x_test), multi_cl
ass='ovr'))
```

```
              precision    recall  f1-score   support

           4      0.406     0.371     0.388        35
           5      0.730     0.753     0.742       377
           6      0.719     0.646     0.681       543
           7      0.571     0.655     0.610       203
           8      0.446     0.625     0.521        40

    accuracy                          0.673      1198
   macro avg      0.575     0.610     0.588      1198
weighted avg      0.679     0.673     0.674      1198

cross validation score with roc_auc_ovr scoring 0.7392343125072626
roc_auc_score 0.8828608853587653
```

The accuracy is same as the previous but we can observe that increase in recall values.

CONCLUSION:

For this assignment, I used K-Nearest Neighbors, Logistic Regression with polynomial features, Decision Tree, and Random Forest. With the roc_auc_score in Scikit-learn I calculated the AUC score for each model. Also using cross_val_score method I found AUC score using cross validation method by passing roc_auc_ovr parameter.

If we compare the cross validation scores and recall results of all models we can see that the best results were obtained with Random Forest Classifier with SMOTE method with 0.739234 cross validation score. Since we had highly imbalanced classes, SMOTE created synthetic minority sampling to balance the samples. And trained the model as if we had balanced classes in our dataset.

Even though we got the highest cross validation score which is around 0.7392, it is still not an ideal model because the recall results are not representative enough for any of the classes. In this case, in order to improve the model we need more data to train it.