I. PROBLEM STATEMENT (5 Points)

Define and Explain the regression problem you want to solve. (Min of 100, Max of 300 Words)

```
## Insert your explanation here:

 ##The current set of data focuses on several aspects of wine, including its taste and its chemical make-up.
 ##The categorization of the quality of the wine is the output of the dataset,and the classification includes
 ##a description of the quality as either poor, good, or excellent. The researcher is attempting to solve the
 ##regression problem by determining the "excellent quality" of wine by using the composition and the other
 ##columns within the dataset as features in order to determine and forecast the "good quality" of wine.
 ##Instead of relying on the time-consuming and archaic approach of tasting wine, which is being replaced by this method,
 ##the classification of wine will now be performed using this far more effective method.
```

II. DATASET DESCRIPTION (5 Points)

Look for Public Datasets. Define and Explain the dataset you will use for classification-ensemble problem you want to solve. Include the link of where the public dataset downloaded (Min of 100, Max of 300 Words)

```
## Insert answer here:

##The red varieties of the Portuguese "Vinho Verde" wine were the focus of the dataset that was selected.
##The amount of a variety of chemicals that are present in wine as well as the effect
##that these chemicals have on the wine's quality are both described in the dataset.
##The dataset can be seen as either a classification or a regression exercise.
##The difficulty of classifying the level of quality of wine is an example of a classification-ensemble
##problem that needs to be resolved. Since the quality of the wine is the desired outcome,
##the output was initially presented as a numerical data rating on a scale from 1 to 10, and this rating was
```

```
##then translated into a word quality scale: (1-3) poor, (4-7) adequate,
```

Double-click (or enter) to edit

III. EXPLORATORY DATA ANALYSIS (EDA)

1. Read the data (5 Points)

Import of needed libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import missingno as msno
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
```

Import dataset through google drive

```
from google.colab import drive
drive.mount('/content/drive')
path = "/content/drive/MyDrive/WineQT.csv"
data = pd.read_csv('/content/drive/MyDrive/WineQT.csv')

     Mounted at /content/drive
```

2. Get an overview of the dataset: (5 Points)

Show the data types and null-count

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   object 
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 116.2+ KB
```

Number of rows and coloumns

```
data.shape
```

```
(1143, 13)
```

3. Print the first five and last five rows of the dataset (5 Points)

First Five

```
data.head()
```

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

Last Five

```
data.tail()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Id
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	Good	1592
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	Good	1593
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	Good	1594
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	Good	1595

4. Find the summary statistics of the dataset (5 Points)

```
data.describe() .T
```

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1143.0	8.311111	1.747595	4.60000	7.10000	7.90000	9.100000	15.90000
volatile acidity	1143.0	0.531339	0.179633	0.12000	0.39250	0.52000	0.640000	1.58000
citric acid	1143.0	0.268364	0.196686	0.00000	0.09000	0.25000	0.420000	1.00000
residual sugar	1143.0	2.532152	1.355917	0.90000	1.90000	2.20000	2.600000	15.50000
chlorides	1143.0	0.086933	0.047267	0.01200	0.07000	0.07900	0.090000	0.61100
free sulfur dioxide	1143.0	15.615486	10.250486	1.00000	7.00000	13.00000	21.000000	68.00000
total sulfur dioxide	1143.0	45.914698	32.782130	6.00000	21.00000	37.00000	61.000000	289.00000
density	1143.0	0.996730	0.001925	0.99007	0.99557	0.99668	0.997845	1.00369
pH	1143.0	3.311015	0.156664	2.74000	3.20500	3.31000	3.400000	4.01000
sulphates	1143.0	0.657708	0.170399	0.33000	0.55000	0.62000	0.730000	2.00000
alcohol	1143.0	10.442111	1.082196	8.40000	9.50000	10.20000	11.100000	14.90000
Id	1143.0	804.969379	463.997116	0.00000	411.00000	794.00000	1209.500000	1597.00000

5. Find the total count and total percentage of missing values in each column of the DataFrame and display them for columns having at least one null value, in descending order of missing percentages. (5 Points)

```
mask = data.isnull()
total = mask.sum()
percent = 100*mask.mean()

missing_data = pd.concat([total, percent], axis=1,join='outer',
                         keys=['count_missing', 'perc_missing'])
missing_data.sort_values(by='perc_missing', ascending=False, inplace=True)
missing_data
```

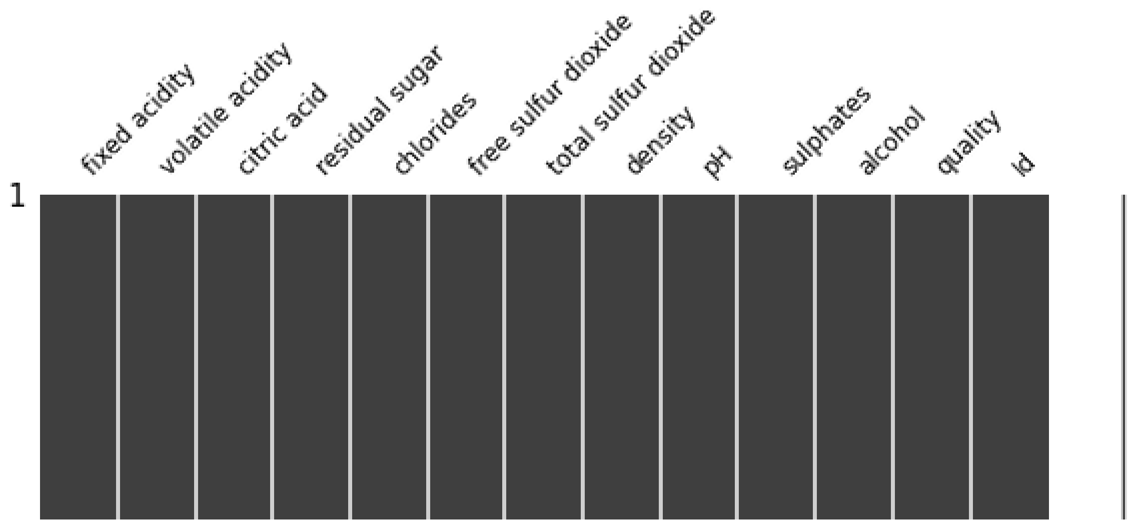
	count_missing	perc_missing
fixed acidity	0	0.0
volatile acidity	0	0.0
citric acid	0	0.0
residual sugar	0	0.0
chlorides	0	0.0
free sulfur dioxide	0	0.0
total sulfur dioxide	0	0.0
density	0	0.0
pH	0	0.0
sulphates	0	0.0
alcohol	0	0.0

Resulted in no missing values or null values within the dataset

6. Plot the nullity matrix and nullity correlation heatmap. (5 Points)

```
msno.matrix(data, figsize=(10,5), fontsize=12)
```

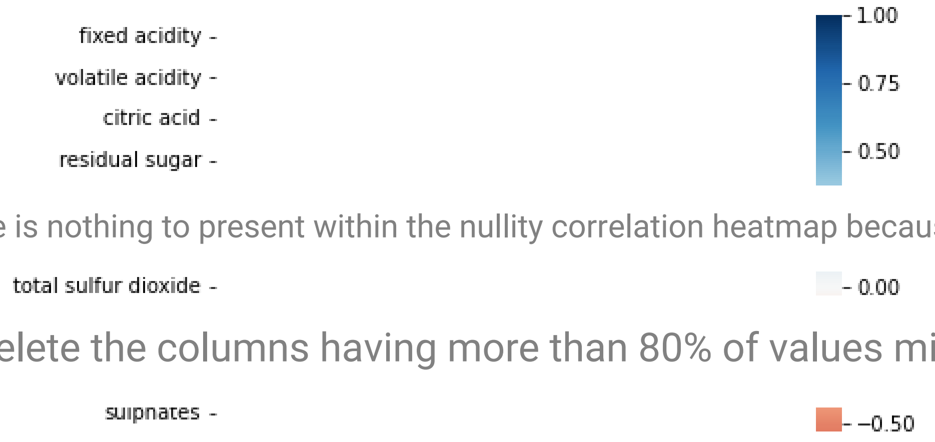
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3541978610>
```



The null matrix resulted like this, becuase there are no null values present within the dataset.

```
null = data.isnull().corr()
plt.figure(figsize = (10,5))
sns.heatmap(null, square=True, annot=True, cmap="RdBu", vmin=-1, vmax=1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f35419a6910>
```



There is nothing to present within the nullity correlation heatmap because there are no null values.

7. Delete the columns having more than 80% of values missing. (5 Points)

```
mask = data.isnull()
total = mask.sum()
percent = 100*mask.mean()

missing_data = pd.concat([total, percent], axis=1,join='outer', keys=['count_missing', 'perc_missing'])
missing_data.sort_values(by='perc_missing', ascending=False, inplace=True)
missing_data
```

	count_missing	perc_missing
fixed acidity	0	0.0
volatile acidity	0	0.0
citric acid	0	0.0

N/A, There will be nothing to delete as there are no missing values.

chlorides	0	0.0

8. Impute null values based from the Summary Statistics. Any statistical values can be used for imputation (5 Points)

density	0	0.0

```
#N/A, Nothing to impute becuase there no null values
```

sulphates	0	0.0

9. Export the Cleaned Dataset. (5 Points)

quality	0	0.0

```
data.to_csv('Maranan,Khen_Zeimmer-cleaned.csv')
```

IV. AI MODELLING

IF REGRESSION: Use Applied Regression Analysis (ACA) using ENSEMBLE

IF CLASSIFICATION: Use Applied Classification Analysis (ACA) using ENSEMBLE

1. Import the required dependencies. (5 Points)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, precision_score, accuracy_score, recall_score
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc_auc_score, roc_curve
from sklearn.ensemble import StackingClassifier
```

2. Read the cleaned data. (5 Points)

Reading the cleaned dataset through google drive

```
from google.colab import drive
drive.mount('/content/drive')
path = "/content/Maranan,Khen_Zeimmer-cleaned.csv"
df= pd.read_csv('/content/Maranan,Khen_Zeimmer-cleaned.csv')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
```

3. Divide the dataset into train and validation DataFrames. (5 Points)

Creating the linear regression model

```
object_variables = data.select_dtypes(include=[np.object])
object_variables.columns
```

```
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: `np.object` is a deprecated alias f
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecatic
      """Entry point for launching an IPython kernel.
    Index(['quality'], dtype='object')
```

No object will be created becuase there are no objects within the dataset.

```
quality= pd.get_dummies(df.quality)
quality
```

	Bad	Best	Good
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
1138	0	0	1
1139	0	0	1
1140	0	0	1
1141	0	0	1
1142	0	0	1

1143 rows × 3 columns

```
merged = pd.concat([df, quality], axis = 'columns')
merged
```

	Unnamed: 0	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	d
0	0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	(
1	1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	(
2	2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	(
3	3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	(
4	4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	(
...	...	...	...	...	...	...	...	...	
1138	1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	(
1139	1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	(
1140	1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	(

```
final = merged.drop(['Unnamed: 0', 'quality'], axis = 'columns')
final
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.

4. Construct an Ensemble model (STACKING Ensemble) using 2 base classifiers/regressor and 1 stacked model. (10 Points)

NOTE:

1. Hyperparameter Tuning: Choose a base classifier model and define the range of hyperparameter values corresponding to the model to be searched over for hyperparameter tuning. Use RandomizedSearchCV or any Hyperparameter tuning techniques.
2. Once the tuning is complete, find the position (iteration number) at which the highest mean test score was obtained. Find the corresponding hyperparameters to be used for your based models and stacked model.
3. Split the dataset into training and validation sets and train a new model using the final hyperparameters on the training dataset. Use scikit-learn's train_test_split() method to split X and y into train and test components, with test comprising 15% of the dataset or any which can attain your target accuracy:
4. Train both your base models and stacked model using the final hyperparamters used from the hyperparameter tuning done in step 2.

```
x=final.drop(["Good"],axis=True)
y=final["Good"]
```

Target is the good quality

```
x_train,x_val,y_train,y_val=train_test_split(x,y,test_size=.20,random_state=11)
```

```
print("input - training set :", x_train.shape)
print("output - training set :", y_train.shape)
print("output - training set :", x_val.shape)
print("output - training set :", y_val.shape)
```

```
     input - training set : (914, 14)
     output - training set : (914,)
     output - training set : (229, 14)
     output - training set : (229,)
```

```
x_train,x_val,y_train,y_val=train_test_split(x,y,test_size=.20,random_state=11)
```

```
#Using KNN process

params_knn = {
    "leaf_size": list(range(1,30)),
    "n_neighbors": list(range(1,50)),
    "p": [1,2]
}


grid_search_kn = GridSearchCV(KNeighborsClassifier(),
                              params_knn,
                              verbose=1,
                              cv=5)
grid_search_kn.fit(x_train, y_train);
print ("Best Score: ", grid_search_kn.best_score_)
print ("Best Parameters: ", grid_search_kn.best_params_)
print ("Best Estimator: ", grid_search_kn.best_estimator_)
```

```
     Fitting 5 folds for each of 2842 candidates, totalling 14210 fits
     Best Score:  0.9496727316399447
     Best Parameters:  {'leaf_size': 1, 'n_neighbors': 7, 'p': 1}
     Best Estimator:  KNeighborsClassifier(leaf_size=1, n_neighbors=7, p=1)
```

```
 #Implementing DecisionTree

decision_tree = DecisionTreeClassifier()
```

```
param_dict = {
    'max_leaf_nodes': list(range(2, 100)),
    'min_samples_split': [2,3,4]
}


grid = GridSearchCV(decision_tree,
                   param_grid = param_dict,
                   cv = 5,
                   verbose = 1,
                   n_jobs = -1)
grid.fit(x_train,y_train)
print ("Best Score: ", grid.best_score_)
print ("Best Parameters: ", grid.best_params_)
print ("Best Estimator: ", grid.best_estimator_)
```

```
     Fitting 5 folds for each of 294 candidates, totalling 1470 fits
     Best Score:  1.0
     Best Parameters:  {'max_leaf_nodes': 3, 'min_samples_split': 2}
     Best Estimator:  DecisionTreeClassifier(max_leaf_nodes=3)
```

```
#LogisticRegression

params_lr = {
    "C":np.logspace(-3,3,7),
    "penalty":["l1","l2"]
}


grid_search_lr = GridSearchCV(LogisticRegression(random_state=11), params_lr, cv=10)
grid_search_lr.fit(x_train, y_train)
print ("Best Score: ", grid_search_lr.best_score_)
print ("Best Parameters: ", grid_search_lr.best_params_)
print ("Best Estimator: ", grid_search_lr.best_estimator_)
```

```
     /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to cor
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to cor
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to cor
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to cor
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to cor
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to cor
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to cor
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
kneighbors_params = {
    'leaf_size': 30,
    'n_neighbors': 50,
    'p': 4
}



decisiontree_params = {
    'max_leaf_nodes': 12,
    'min_samples_split': 6
}


logisticregression_params = {
    'C': 0.001,
    'penalty': 'l2'
}
```

5. Calculate the performance and use a metric accordingly (Target Score >= 80%): (5 Points)

```
knn = KNeighborsClassifier(**kneighbors_params)
knn.fit(x_train, y_train)
knnpred_val = knn.predict(x_val)
accuracyscore = accuracy_score(y_val, knnpred_val)
```

```
precisionscore = precision_score(y_val, knnpred_val, average='weighted')
recallscore = recall_score(y_val, knnpred_val, average='macro')
f1score = f1_score(y_val, knnpred_val, average = 'micro')
cm_rf = confusion_matrix(y_val, knnpred_val)
print("--RANDOM FOREST CLASSIFIER RESULTS--")
print("Accuracy: ", accuracyscore*100)
print("Precision: ", precisionscore*100)
print("Recall: ", recallscore*100)
print("F1-Score: ", f1score*100)
print("Confusion Matrix: \n", cm_rf)
```

```
--RANDOM FOREST CLASSIFIER RESULTS--
Accuracy:  96.06986899563319
Precision:  92.29419728838123
Recall:  50.0
F1-Score:  96.06986899563319
Confusion Matrix: 
 [[  0   9]
 [  0 220]]
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is il
  _warn_prf(average, modifier, msg_start, len(result))
```

```
dt = DecisionTreeClassifier(**decisiontree_params, random_state=11)
dt.fit(x_train, y_train)
dtpred_val = dt.predict(x_val)
accuracyscore1 = accuracy_score(y_val, dtpred_val)
precisionscore1 = precision_score(y_val, dtpred_val, average='weighted')
recallscore1 = recall_score(y_val, dtpred_val, average='weighted')
f1score1 = f1_score(y_val, dtpred_val, average = 'micro')
cm_rf1 = confusion_matrix(y_val, dtpred_val)
print("--DECISION TREE CLASSIFIER RESULTS--")
print("Accuracy: ", accuracyscore1*100)
print("Precision: ", precisionscore1*100)
print("Recall: ", recallscore1*100)
print("F1-Score: ", f1score1*100)
print("Confusion Matrix: \n", cm_rf1)
```

```
--DECISION TREE CLASSIFIER RESULTS--
```

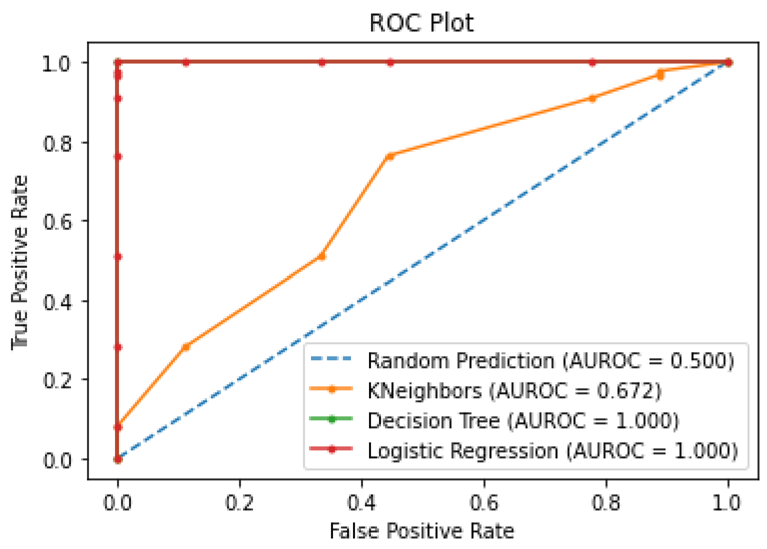
```
Accuracy:  100.0
Precision:  100.0
Recall:  100.0
F1-Score:  100.0
Confusion Matrix:
 [[  9   0]
 [  0 220]]
```

```
estimator_list = [
    ('knn', knn),
    ('dt', dt),
     ]
stacked_model = StackingClassifier(estimators=estimator_list, final_estimator=LogisticRegression(**logisticregression_params
stacked_model.fit(x_train, y_train)
stacked_preds_val = stacked_model.predict(x_val)
accuracyscore2 = accuracy_score(y_val, stacked_preds_val)
precisionscore2 = precision_score(y_val, stacked_preds_val, average='micro')
recallscore2 = recall_score(y_val, stacked_preds_val, average='micro')
f1score2 = f1_score(y_val, stacked_preds_val, average = 'micro')
cm_rf2 = confusion_matrix(y_val, stacked_preds_val)
print("--STACKED MODEL RESULTS--")
print("Accuracy: ", accuracyscore2*100)
print("Precision: ", precisionscore2*100)
print("Recall: ", recallscore2*100)
print("F1-Score: ", f1score2*100)
print("Confusion Matrix: \n", cm_rf2)
```

```
--STACKED MODEL RESULTS--
Accuracy:  96.06986899563319
Precision:  96.06986899563319
Recall:  96.06986899563319
F1-Score:  96.06986899563319
Confusion Matrix:
 [[  0   9]
 [  0 220]]
```

6. Plot the performane accordingly, use the appropriate plotting: (5 Points)

```
y_val.replace([2],[1],inplace=True)


r_probs = [0 for _ in range(len(y_val))]
knearestneighbors_probs = knn.predict_proba(x_val)
decisiontree_probs = dt.predict_proba(x_val)
stackedmodel_probs = stacked_model.predict_proba(x_val)


knearestneighbors_probs = knearestneighbors_probs[:, 1]
decisiontree_probs = decisiontree_probs[:, 1]
stackedmodel_probs = stackedmodel_probs[:, 1]


r_auc = roc_auc_score(y_val, r_probs, multi_class = 'b')
knearetneighbors_auc = roc_auc_score(y_val, knearestneighbors_probs)
decisiontree_auc = roc_auc_score(y_val, decisiontree_probs)
stackedmodel_auc = roc_auc_score(y_val, stackedmodel_probs)


r_fpr, r_tpr, _ = roc_curve(y_val, r_probs)
knn_fpr, knn_tpr, _ = roc_curve(y_val, knearestneighbors_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_val, decisiontree_probs)
sm_fpr, sm_tpr, _ = roc_curve(y_val, stackedmodel_probs)


plt.plot(r_fpr, r_tpr, linestyle='--', label='Random Prediction (AUROC = %0.3f)' % r_auc)
plt.plot(knn_fpr, knn_tpr, marker='.', label='KNeighbors (AUROC = %0.3f)' % knearetneighbors_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC = %0.3f)' % decisiontree_auc)
plt.plot(sm_fpr, sm_tpr, marker='.', label='Logistic Regression (AUROC = %0.3f)' % stackedmodel_auc)

# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```



7. Test the final values on the test dataset. (5 Points)

```
results = x_val.copy()
results ['Actual'] = y_val
results ['Predicted'] = stacked_model.predict(x_val)
results = results[['Actual', 'Predicted']]
results[:10]
```

```
     Actual  Predicted
 60       1          1
326       1          1
671       1          1
```

8. Export the Final Model using PICKLE Library. (5 Points)

```
import pickle
filename = 'GoodQualityWine.pkl'
pickle.dump(stacked_model,open(filename,'wb'))
```

V. FINAL DOCUMENTATION

Publishable Paper Article (50 Points)

```
## Insert your GDrive Link of the Word file of your Final Documentation here, following the IEEE format template.

## https://docs.google.com/document/d/1XZbYXH0VkFNdHf57tFsuhNXNq-LZvd_w/edit?usp=sharing&ouid=116848938120228726362&rtpof=tr
```