SCHOOL OF COMPUTER SCIENCES
UNIVERSITI SAINS MALAYSIA

CPC152 - FOUNDATION AND PROGRAMMING FOR DATA ANALYTICS
SEMESTER 2, 2022/2023

LECTURER:
TS.DR.CHEW XIN YING

DUE: 25 JUNE 2023
GROUP PROJECT: GROUP 28
Group Members:

NAME	STUDENT ID
AZAM TAMHEED	160610
JALAL NAIM BIN MAT YAACOB	164498
MOHAMMAD HAZIQ SURMA	160800
NORITA BINTI MUIN	160453
SHIVARSANKAR A/L THANABALAN	164525
WONG JING MING	163797

TABLE OF CONTENT

1.0 ABSTRACT

Machine learning harnesses powerful pre-programmed algorithms to efficiently process and analyze input data, allowing for the precise determination of output values that fall within a desirable range. The foundational stages of machine learning encompass classification and regression, serving as the bedrock for a wide array of algorithms. These algorithms generate sophisticated machine learning models, consisting of both model data and a predictive algorithm. By encapsulating the program within these models, machine learning algorithms empower automated programming, ushering in a new era of computational intelligence.

In this group project, our primary focus is to perform a comparative analysis and evaluate various types of machine learning models for data science that was used in machine learning algorithms using several classification processes. These machine learning algorithms are Random Forest, K-Nearest Neighbours(KNN), and Support Vector Machine (SVM). This project's secondary objective is to identify issues and critical areas that require attention. This requires a comprehensive analysis of the algorithms' strengths and weaknesses, which is accomplished by putting them into practice and employing a variety of analytical methods. We will use a provided sample dataset in USM e-learning portal and implement code and a variety of ways to assess the dataset's accuracy and explore the data by deploying each of the machine learning algorithms that focus on particular key attributes. While looking at the dataset and delving into each algorithm, we will place an emphasis on a select group of important characteristics. By doing this, we want to improve our comprehension of the data at hand and learn more about how well the algorithms work.

In this project, we used a few qualitative methods such as internet references, discussion among group members, and keeping records of the performances of each of the machine learning algorithms. We used the wine quality dataset to extract some useful insights that measure the performances of each algorithm and how accurate are they in real-life scenarios. For example, by using these machine learning models we are able to predict the quality of wine after the production is over at the brewery to classify and sell wine corresponding to their quality.

2.0 INTRODUCTION OF MACHINE LEARNING ALGORITHMS

2.1 Random Forest

The Random Forest algorithm is a powerful and adaptable machine learning method that has been widely praised in a variety of fields. Random Forest is a formidable collection of predictive models that are influenced by the collective wisdom of a wide variety of decision trees. While reducing the inherent weaknesses of each tree, this algorithm capitalizes on its strengths. Random Forest excels at tackling classification and regression issues with remarkable accuracy and robustness by combining the predictions of multiple trees. Random Forest is a steadfast pillar in the field of machine learning because it can handle high-dimensional data, identify important features, and minimize overfitting. It provides unparalleled insight and foresight in a wide range of applications.

2.2 K-Nearest Neighbours (KNN)

For classification and regression applications, the K-Nearest Neighbours (KNN) algorithm is a popular non-parametric and instance-based machine learning technique. KNN uses the similarity principle, in which predictions are based on the traits of nearby occurrences in the feature space.The training dataset for KNN is made up of labelled instances that are each characterised by a collection of input characteristics and the labels that correspond to those features. In contrast to conventional model-based methods, KNN doesn't explicitly build a model during the training stage. As opposed to doing so, it uses the full training dataset as its knowledge base, enabling tremendous flexibility and adaptability.KNN determines the K closest neighbours to the input point when given a fresh, unlabeled instance for prediction by calculating the similarity or separation between examples. KNN uses the information gathered from the K nearest neighbours to create predictions.

2.3 Support Vector Machine (SVM)

The Support Vector Machine (SVM) is a supervised machine learning algorithm commonly used for classification tasks, although it can also be applied to regression problems. Its primary objective is to identify the optimal decision boundary, known as the separating hyperplane, that maximizes the distance to the nearest data points of each class. SVM achieves this by selecting the most relevant support vectors, which are crucial in defining the hyperplane. Notably, SVM exhibits low memory requirements. However, it can be challenging for users to comprehend and analyze due to its intricacies.

Despite the inherent complexities, SVM often outperforms many other algorithms by delivering highly accurate results and faster predictions. However, its effectiveness diminishes in scenarios where the target classes overlap or when the dataset contains substantial noise. Additionally, SVM tends to have longer training times when dealing with larger datasets. Nonetheless, SVM demonstrates good performance when applied to image data.

3.0 OBJECTIVES OF THE EXPERIMENT

The propose of this project is to create a machine learning models that can predict a highly accurate wine quality using the wine quality dataset provided in the USM e-learning portal. This data set contains data about the wine contents as the features and the rating of the wine quality as the target. This also allows us to determines which models give a high prediction and a better performance by using three different machine learning algorithms which are Random Forest, KNN (K-Nearest Neighbours), and SVM (Support Vector Machine).

Firstly, we will explore the performance of each of our models using different data splitting ratios which are 60:40, 70:30, and 80:20 by allocating the greater portion of the data into the training set and the rest for the testing set. The ratio that gives a highly accurate prediction will be chosen to test different feature selection methods. Then, we will use feature selection such as feature importance, correlation matrix using heatmap, and univariate selection to enhance the model's predictive power and reduce potential noise by keeping the top four features by each feature selection technique.

4.0 JUSTIFICATIONS

4.1 Random Forest

We have chosen 3 of the most suitable machine learning algorithm for our project. First and foremost, we choose to use Random forest Classifier algorithm as one of the machine learning model for several reasons based on its suitability and advantages. One of the main reason why Random forest is suitable for this dataset is its ability to handle multiple and high number of High-Dimensional Data. Random Forest is capable of handling datasets with a high number of features, such as the 11 input features in the wine quality dataset. It uses a combination of decision trees, where each tree learns different aspects of the data. Random Forest is well-suited for capturing non-linear relationships in the data. Unlike linear models that assume a linear relationship between the features and the target variable, Random Forest can model complex and non-linear patterns. This is particularly useful in scenarios where the wine quality may depend on non-linear interactions between the input features, such as acidity, pH, and citric acid content.

By considering these factors, Random Forest Classifier emerges as a suitable choice for the wine quality dataset. It can handle high-dimensional data, capture non-linear relationships, handle outliers and imbalanced data, provide feature importance insights, and leverage ensemble learning for improved predictive performance.

4.2 K-Nearest Neighbours (KNN)

The k-Nearest Neighbors (KNN) algorithm is a suitable choice for our project due to its distinct characteristics and advantages. KNN is a non-parametric and instance-based algorithm that determines the class membership of a sample based on its proximity to the neighboring data points.One of the key advantages of KNN is its

ability to handle high-dimensional data, making it appropriate for wine quality dataset with its 11 input features. KNN is also effective in identifying non-linear correlations in the data. KNN does not make any assumptions on the underlying data distribution, in contrast to linear models, which presume a linear relationship between the features and the target variable. KNN also possesses the advantage of being a simple and interpretable algorithm. In conclusion, the KNN algorithm is a suitable choice for wine quality dataset due to its ability to handle high-dimensional data, capture non-linear relationships, and offer interpretability. By considering the proximity of samples in the feature space, KNN can effectively predict wine quality based on the characteristics of neighboring samples. However, it is crucial to carefully select the optimal k value and appropriate distance metric to ensure optimal performance.

4.3 Support Vector Machine(SVM)

We opted for the Support Vector Machine (SVM) algorithm for one of our machine learning tasks for multiple reasons. One of the primary factors is SVM's reputation for being simple yet effective in various applications. As someone well-versed in machine learning, it's crucial to have a diverse range of algorithms available, and SVM undoubtedly enriches our toolbox.One standout advantage of SVM is its ability to deliver high accuracy while demanding less computational power compared to alternative algorithms. This becomes particularly advantageous when dealing with extensive datasets or limited computational resources. Thanks to extensive research and optimization, SVM has become a reliable choice for achieving precise outcomes.

The fundamental objective of the SVM algorithm is to identify a hyperplane that effectively segregates data points into distinct classes within an N-dimensional feature space. By maximizing the margin, which denotes the separation between data points of different classes, SVM strives to create a robust decision boundary that generalizes well to unseen data. This characteristic contributes to SVM's ability to make confident predictions and handle intricate decision boundaries. To summarize, considering the simplicity, effectiveness, versatility, and proficiency in handling complex decision boundaries, we have made a well-founded decision in selecting the Support Vector Machine algorithm for our machine learning tasks. It equips us with a potent tool to tackle a wide range of problems while ensuring high accuracy and optimized computational resource usage.

5.0 STEPS TO BUILD

All of the three machine learning models has some similar steps in building them such as from importing the libraries up to feature selection for each machine learning algorithms. So these steps are merged together.

5.1 Import the library

These are the necessary libraries imported for Random forest, KNN and SVM.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
#from sklearn import metrics
#from sklearn.utils import shuffle

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

5.2 Understand and explore data

To begin understanding and exploring the dataset, we need to read the data first using the 'pd.read_csv()' method and store it in a data frame named "df". Afterwards, we can utilize various methods to gain insights from the dataset. For instance, 'df.head(10)' and 'df.tail(10)' allow us to examine the initial and last ten examples in the dataset' providing a glimpse into its contents. By employing 'df.shape', we can determine the dataset's dimensions, revealing the number of columns and rows it comprises. Additionally, we can employ 'df.describe()' to generate a numerical summary of the dataset, presenting statistics that aid in comprehending its numerical attributes.

```
In [170]: # Look through of the data set info in term of statistic
          df.describe()
```

Out[170]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

5.3 Data Preprocessing

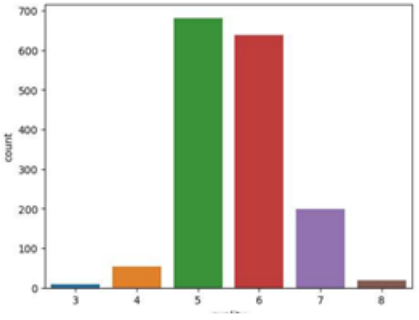
Data cleaning is an essential step to create a machine learning model. To perform this step, we utilize `df.apply(lambda x: sum(x.isnull()),axis=0)` to assess the total number of missing values. If any missing values are detected, we can proceed to address them by employing techniques like dropping the missing values or imputation. Fortunately, in this particular case, no missing values were found. Hence, we can skip this process and move on to the subsequent steps.

```
In [172]: # display target variable
          df['quality'].unique()
```

```
Out[172]: array([5, 6, 7, 4, 8, 3], dtype=int64)
```

```
In [173]: # plot graph to se the distribution of the target variable
          sns.countplot(data = df, x='quality')
```

```
Out[173]: <Axes: xlabel='quality', ylabel='count'>
```



Binarization of the target variable should be done if there are more than two target variables. Before that, we used `.unique()` function to check how many target variables we have. We plotted a graph to see the distribution of the target variable and display the counts.

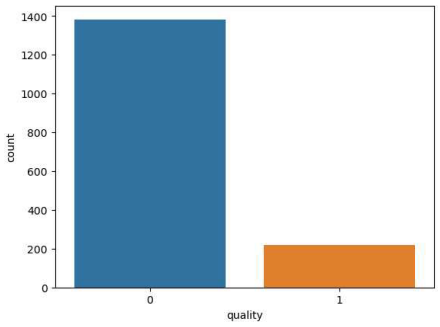
```
# perform binarization of target variable
df['quality'] = [1 if x>=7 else 0 for x in df['quality']]
```

```
df['quality'].value_counts()
```

```
0    1382
1     217
Name: quality, dtype: int64
```

```
# store feature matrix in x and response (target) in vector y
x = df.drop('quality', axis=1)
y = df['quality']
```

Binarization of the target variable is performed using the above code. We classified the feature variable into 2 classes which are 1 for good and 0 for bad. For quality greater than or equal to 7 (x > = 7) classified into 1 and else into 0. That means we picked only 7 and 8 quality ratings to be classified as good because they showed us a highly accurate prediction compared to the other classification (x > = 6). Our classification criterion was based on quality ratings, whereby any rating equal to or greater than 7 (x >= 7) was deemed as "good," while all other ratings were considered "bad."
The distribution of the target variable using x > = 7 displayed on a chart and counted.
All feature **stored** in x while target **stored** in y




```
In [179]: x = df.iloc[:,0:11] #only have 12 column
          y = df.iloc[:,-1] # last column
```

Feature scaling in machine learning is one of the most critical steps in the data preprocessing of data before building the machine learning model. Scaling can make a difference between a weak and a good machine learning model. The most common technique of feature scaling is Normalisation, Standardization, MinMax Scaler, and Binarizer. In this project, we use MinMaxScaler to scale the feature. Then, the scaled feature is converted back to the dataframe.

```
# to put over features on the same scale
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)

# Convert scaled features to DataFrame with column names
x = pd.DataFrame(x_scaled, columns=x.columns)
```

5.4 Splitting Data Set Into Different Ratios

We decided to do the test on 3 different data splitting ratios which are 60:40, 70:30, and 80:20. We allocate the largest portion of the data to the training set and the rest to the testing set. The split dataset is then stored in 2 different .csv files. The ratio that gives a highly accurate prediction is chosen and implemented in the whole code.

1. **Ratio of 60:40**

```
# split dataset into train set and test set
x_trainA, x_testA, y_trainA, y_testA = train_test_split(x, y, test_size=0.4, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_trainA, y_trainA], axis=1)
train_data.to_csv('train_A.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testA, y_testA], axis=1)
test_data.to_csv('test_A.csv', index=False)
```

2. Ratio of 70:30

```
# split dataset into train set and test set (for feature importance)
x_trainB, x_testB, y_trainB, y_testB = train_test_split(x, y, test_size=0.3, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_trainB, y_trainB], axis=1)
train_data.to_csv('train_B.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testB, y_testB], axis=1)
test_data.to_csv('test_B.csv', index=False)
```

3. Ratio of 80:20

```
# split dataset into train set and test set (for feature importance)
x_trainC, x_testC, y_trainC, y_testC = train_test_split(x, y, test_size=0.2, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_trainC, y_trainC], axis=1)
train_data.to_csv('train_C.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testC, y_testC], axis=1)
test_data.to_csv('test_C.csv', index=False)
```

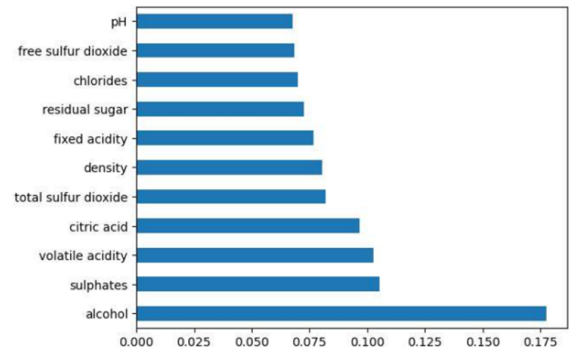
5.5 Feature Selection

Next, we implemented feature selection to build the all 3 the models. We chose 3 types of feature selection to compare the model's performance and determined which one is better to be chosen for the champion model. These are including feature importance, correlation matrix using heatmap, and univariate selection. The less related feature dropped.

5.5.1 Feature Importance

Firstly, we used Feature importance**.** We used Extra Tree Classifier to extract the top 10 features for the dataset. Then, we plotted a graph of feature importance for better visualisation.

```
# plot graph of feature importances for better visualization

feat_importances = pd.Series(model.feature_importances_, index=x.columns)
feat_importances.nlargest(12).plot(kind='barh')
plt.show()
```




```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

model = ExtraTreesClassifier()
model.fit(x, y)

#use inbuilt class feature_importances of tree based classifiers
print(model.feature_importances_)
```

```
[0.07657041 0.10246323 0.09681798 0.0726115  0.06974953 0.06839384
 0.08208658 0.08058899 0.06761535 0.10542293 0.17767967]
```

5.5.2 Univariate Selection

The second feature selection is Univariate Selection. The below code displays how to utilize univariate selection. In this code, the variable `dfscores` represents the selection scores for each feature, while `dfcolumns` corresponds to the names of the individual features. To enhance the visualization, the code concatenates these two dataframes using the statement `featureScores = pd.concat([dfcolumns, dfscores], axis=1.`

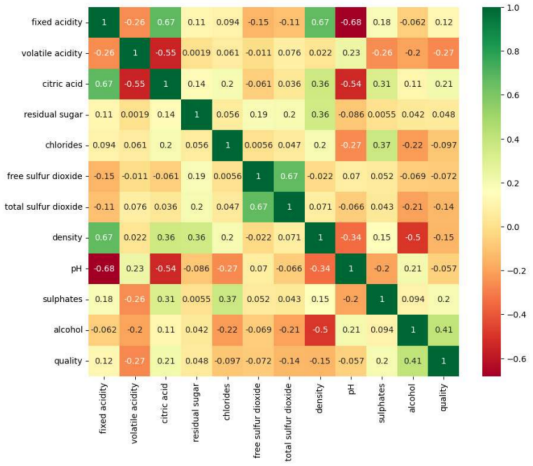
```
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)

#naming the dataframe columns
featureScores.columns = ['Specs','Score']

#print 10 best features
featureScores
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

5.5.3 Correlation Matrix using a heatmap

Visualization of the heatmap below demonstrates the relationship between each feature to the other. In this case, we can see that alcohol has the highest score followed by citric acid, sulphates. In the other word, alcohol is the most related feature that contributes to wine quality.

```
# get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
corrmat

# plot heatmap
fig, ax = plt.subplots(figsize=(10, 8))
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
plt.show()
```



The common steps present in all of the three machine learning algorithms were shown above. Now steps for each machine learning algorithm were discussed.

5.6 Random Forest

5.6.1 Test model with different data splitting ratios

To evaluate the performance of our model, we did some experiments to test which data-splitting ratios give better performance for the random forest algorithm by comparing their accuracy on predicted data.

```
In [169]: rf = RandomForestClassifier()
          rf.fit(x_trainA, y_trainA)
Out[169]: RandomForestClassifier()
```

```
In [173]: rf = RandomForestClassifier()
          rf.fit(x_trainB, y_trainB)
Out[173]: RandomForestClassifier()
```

```
In [177]: rf = RandomForestClassifier()
          rf.fit(x_trainC, y_trainC)
Out[177]: RandomForestClassifier()
```

1. **Random forest for ratio 60:40**

```
In [170]: #perform prediction
          y_predictA = rf.predict(x_testA)

          print("SCORE (60:40)")
          print("------------------------------")
          print("Accuracy: ", accuracy_score(y_testA, y_predictA))
          print("F1 Score: ", f1_score(y_testA, y_predictA, average='weighted'))
          print("Recall: ", recall_score(y_testA, y_predictA, average='weighted'))
          print("Precision: ", precision_score(y_testA, y_predictA, average='weighted', zero_division=0))
```

```
SCORE (60:40)
------------------------------
Accuracy:  0.91875
F1 Score:  0.9192854445052226
Recall:  0.91875
Precision:  0.9198542266556972
```

2. **Random forest for ratio 70:30**

```
In [174]: #perform prediction
y_predictB = rf.predict(x_testB)

print("SCORE (70:30)")
print("------------------------------")
print("Accuracy: ", accuracy_score(y_testB, y_predictB))
print("F1 Score: ", f1_score(y_testB, y_predictB, average='weighted'))
print("Recall: ", recall_score(y_testB, y_predictB, average='weighted'))
print("Precision: ", precision_score(y_testB, y_predictB, average='weighted', zero_division=0))
```

```
SCORE (70:30)
------------------------------
Accuracy:  0.91875
F1 Score:  0.9168466686948585
Recall:  0.91875
Precision:  0.9153416347381864
```

3. **Random forest for ratio 80:20**

```
In [178]: #perform prediction
y_predictC = rf.predict(x_testC)

print("SCORE (80:20)")
print("------------------------------")
print("Accuracy: ", accuracy_score(y_testC, y_predictC))
print("F1 Score: ", f1_score(y_testC, y_predictC, average='weighted'))
print("Recall: ", recall_score(y_testC, y_predictC, average='weighted'))
print("Precision: ", precision_score(y_testC, y_predictC, average='weighted', zero_division=0))
```

```
SCORE (80:20)
------------------------------
Accuracy:  0.928125
F1 Score:  0.9251651126651126
Recall:  0.928125
Precision:  0.9232415254237287
```

Thus, ratio of 80:20 selection to be used in the whole code to build random forest algorithm since it got the highest accuracy score compare to the others.

5.6.2 Test model with different feature selection

1. **Without feature selection**

```
classifier = RandomForestClassifier()
classifier.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
In [181]: #perform prediction
y_predict = classifier.predict(x_test)

print("SCORE (Without Feature Selection )")
print("------------------------------")
print("Accuracy: ", accuracy_score(y_test, y_predict))
print("F1 Score: ", f1_score(y_test, y_predict, average='weighted'))
print("Recall: ", recall_score(y_test, y_predict, average='weighted'))
print("Precision: ", precision_score(y_test, y_predict, average='weighted', zero_division=0))
```

```
SCORE (Without Feature Selection )
------------------------------
Accuracy:  0.934375
F1 Score:  0.9316724941724942
Recall:  0.934375
Precision:  0.9300635593220339
```

2. **With feature selection**

We defined selected features based on the ranking of each feature selection, we selected top 6, 4, and 2 for each feature selection. Then, we did some experiment for the selected features to compare the performance of each different number of features for the 3 feature selections. Below is the example of code for 6 selected features for feature importance.

```
In [184]: # Define the selected features
selected_features_1A = ['alcohol','citric acid' , 'volatile acidity','sulphates','total sulfur dioxide', 'density']
```

The following is the code that we used to print the result of their performance

```
In [184]: # Define the selected features
selected_features_1A = ['alcohol','citric acid' , 'volatile acidity','sulphates','total sulfur dioxide', 'density']

# Select the desired features from the training and testing sets
xtrain_selected_1A = x_train[selected_features_1A]
xtest_selected_1A = x_test[selected_features_1A]

# Create and train the Random Forest model
classifier= RandomForestClassifier()
classifier.fit(xtrain_selected_1A, y_train)

# Make predictions on the test set
y_pred_1A = classifier.predict(xtest_selected_1A)

# Print the evaluation metrics for the Random Forest model
print("SCORE (Random Forest-6)")
print("------------------------------")
print("Accuracy:", accuracy_score(y_test, y_pred_1A))
print("F1 Score:", f1_score(y_test, y_pred_1A, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_1A, average='weighted'))
print("Precision:", precision_score(y_test, y_pred_1A, average='weighted', zero_division=0))
```

A) Feature Importance

- **6 features**: The 6 highest scores of feature importance were selected. This includes *alcohol, sulphates, citric acid, volatile acidity, density, and total sulphur dioxide.*

- **4 features**: The 4 highest scores of feature importance were selected. This includes *alcohol, sulphates, citric acid, and volatile acidity*.

- **2 features**: The 2 highest scores of feature importance were selected. This includes *alcohol and sulphates.*

```
SCORE (Random Forest-6)
------------------------------
Accuracy: 0.91875
F1 Score: 0.9199199129367118
Recall: 0.91875
Precision: 0.9212239583333334
```

```
SCORE (Random Forest-4)
------------------------------
Accuracy: 0.90625
F1 Score: 0.9111091114767585
Recall: 0.90625
Precision: 0.9177748226950355
```

```
SCORE (Random Forest-2)
------------------------------
Accuracy: 0.86875
F1 Score: 0.8755527560674621
Recall: 0.86875
Precision: 0.883690276222471
```

B) Univariate Selection

Different number of selected feature using univariate selection used to be tested

- **6 features:** The 6 highest scores of univariate selection were selected. This includes *alcohol,citric acid ,volatile acidity ,sulphates ,total sulfur dioxide and ,fixed acidity* .

- **4 features:** The 4 highest scores of univariate selection were selected. This includes *alcohol,citric acid ,volatile acidity ,sulphates.*

- **2 features:** The 2 highest scores of univariate selection were selected. This includes *alcohol and citric acid.*

```
SCORE (Random Forest-6)
------------------------------
Accuracy: 0.91875
F1 Score: 0.9199199129367118
Recall: 0.91875
Precision: 0.9212239583333334
```

```
SCORE (Random Forest-4)
------------------------------
Accuracy: 0.90625
F1 Score: 0.9111091114767585
Recall: 0.90625
Precision: 0.9177748226950355
```

```
SCORE (Random Forest-2)
------------------------------
Accuracy: 0.86875
F1 Score: 0.8755527560674621
Recall: 0.86875
Precision: 0.883690276222471
```

C) Correlation Matrix using Heatmap

We also used a different number of features based on the score using a correlation matrix to determine which one performed better with highest accuracy.

- 6 features: The 6 highest scores of correlation matrix were selected. This includes *alcohol, citric acid, sulphates, fixed acidity, residual sugar and pH.*
- 4 features:This includes *alcohol, citric acid, sulphates, and fixed acidity.*
- 2 features: This includes *alcohol and citric acid.*

```
SCORE (Random Forest-6)
------------------------------
Accuracy: 0.940625
F1 Score: 0.9381798756798755
Recall: 0.940625
Precision: 0.936885593220339
```

```
SCORE (Random Forest-4)
------------------------------
Accuracy: 0.903125
F1 Score: 0.9075939022166655
Recall: 0.903125
Precision: 0.9134305462706523
```

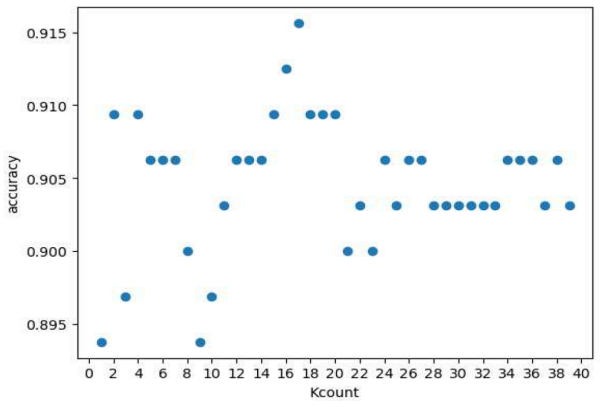
```
SCORE (Random Forest-2)
------------------------------
Accuracy: 0.871875
F1 Score: 0.8792212365796086
Recall: 0.871875
Precision: 0.8883024226663017
```

5.7 K- Nearest Neighbors (KNN)

5.7.1 Finding the optimal value of K.

In KNN, the value of k represents the number of nearest neighbors to consider when making predictions.The code iterates over a range of k values from 1 to 40 (k_range). For each k value, it creates a KNN classifier (classifier) with that number of neighbors. The classifier is then trained on the training set (x_train and y_train).The accuracy score for each value of k is stored in the scores list. By examining the scores, we can determine the value of k that results in the highest accuracy. This information can be used to select the optimal value of k for the KNN classifier. The most optimal K-value is 18.

```
# split dataset into train set and test set (for feature importance)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, randoı
#Finding k value with highest acccuracy
k_range = range(1,40)
scores = []
for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors = k)
    classifier.fit(x_train,y_train)
    scores.append(classifier.score(x_test,y_test))


plt.figure()
plt.xlabel("Kcount")
plt.ylabel("accuracy")
plt.scatter(k_range,scores)
plt.grid
plt.xticks([0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40])
plt.show()
```




```
#using root of y_train to find a suitable K value
import math
math.sqrt(len(y_train))
```

```
35.76310948449533
```

5.7.2 Test model with different data splitting ratios

To evaluate the performance of our model, we did some experiments to test which data-splitting ratios give better performance for the KNN algorithm by comparing their accuracy on predicted data.We decided to do the test on 3 different data splitting ratios which are 60:40, 70:30, and 80:20. We allocate the largest portion of the data to the training set and the rest to the testing set. The split dataset is then stored in 2 different .csv files. The ratio that gives a highly accurate prediction is chosen and implemented in the whole code.

1. KNN for 60:40 ratio

```
# split dataset into train set and test set
x_trainA, x_testA, y_trainA, y_testA = train_test_split(x, y, test_size=0.4, r

# Save training set to CSV
train_data = pd.concat([x_trainA, y_trainA], axis=1)
train_data.to_csv('train_A.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testA, y_testA], axis=1)
test_data.to_csv('test_A.csv', index=False)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
```

```
classifier.fit(x_trainA, y_trainA)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=18)
```

```
y_predA = classifier.predict(x_testA)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, c
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_testA, y_predA))

print ('Accuracy:', accuracy_score(y_testA, y_predA))
print ('Recall:', recall_score(y_testA, y_predA, average="weighted"))
print ('Precision:', precision_score(y_testA, y_predA, average="weighted"))

confusion = confusion_matrix(y_testA, y_predA)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       574
           1       0.38      0.08      0.13        66

    accuracy                           0.89       640
   macro avg       0.64      0.53      0.53       640
weighted avg       0.85      0.89      0.86       640

Accuracy: 0.8921875
Recall: 0.8921875
Precision: 0.849282680039259
Confusion matrix:
[[566   8]
 [ 61   5]]
```

2. KNN for 70:30 ratio

```
# split dataset into train set and test set (for feature importance)
x_trainB, x_testB, y_trainB, y_testB = train_test_split(x, y, test_size=0.3, r

# Save training set to CSV
train_data = pd.concat([x_trainB, y_trainB], axis=1)
train_data.to_csv('train_B.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testB, y_testB], axis=1)
test_data.to_csv('test_B.csv', index=False)
```

```
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(x_trainB, y_trainB)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=18)
```

```
y_predB = classifier.predict(x_testB)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, c
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_testB, y_predB))

print ('Accuracy:', accuracy_score(y_testB, y_predB))
print ('Recall:', recall_score(y_testB, y_predB, average="weighted"))
print ('Precision:', precision_score(y_testB, y_predB, average="weighted"))

confusion = confusion_matrix(y_testB, y_predB)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       430
           1       0.20      0.02      0.04        50

    accuracy                           0.89       480
   macro avg       0.55      0.51      0.49       480
weighted avg       0.82      0.89      0.85       480

Accuracy: 0.8895833333333333
Recall: 0.8895833333333333
Precision: 0.8242543859649122
Confusion matrix:
[[426   4]
 [ 49   1]]
```

3. KNN for 80:20

In [87]:
```
# split dataset into train set and test set (for feature importance)
x_trainC, x_testC, y_trainC, y_testC = train_test_split(x, y, test_size=0.2, random_state=0)

# Save training set to CSV
train_data = pd.concat([x_trainC, y_trainC], axis=1)
train_data.to_csv('train_C.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testC, y_testC], axis=1)
test_data.to_csv('test_C.csv', index=False)
```

In [88]:
```
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(x_trainC, y_trainC)
```

Out[88]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=18)
```

```
y_predC = classifier.predict(x_testC)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_testC, y_predC))

print ('Accuracy:', accuracy_score(y_testC, y_predC))
print ('Recall:', recall_score(y_testC, y_predC, average="weighted"))
print ('Precision:', precision_score(y_testC, y_predC, average="weighted"))

confusion = confusion_matrix(y_testC, y_predC)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       290
           1       0.60      0.10      0.17        30

    accuracy                           0.91       320
   macro avg       0.76      0.55      0.56       320
weighted avg       0.88      0.91      0.88       320

Accuracy: 0.909375
Recall: 0.909375
Precision: 0.8848214285714284
Confusion matrix:
[[288   2]
 [ 27   3]]
```

Thus, ratio of 80:20 selection to be used in the whole code to build KNN algorithm since it got the highest accuracy score compare to the others.

5.7.3 <u>Test model with different feature selection</u>

1. **Without feature selection**

The code is for without feature selection where only the dataset was split into a training set and a test set using the train_test_split function from an imported library. The test set is 20% of the entire dataset.The training set is saved to a CSV file named 'train_cm.csv' .The test set is saved to a CSV file named 'test_cm.csv'.The KNN algorithm is then applied to the data using the KNeighborsClassifier class from the same library. The n_neighbors parameter is set to 18, and the distance metric used is 'euclidean'.

```
In [35]: # Feature Scaling to x_train and x_test to classify better.
         #from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         x_train = sc.fit_transform(x_train)
         x_test = sc.transform(x_test)
```

```
In [36]: # split dataset into train set and test set (for feature importance)
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_

         # Save training set to CSV
         train_data = pd.concat([x_train, y_train], axis=1)
         train_data.to_csv('train.csv', index=False)

         # Save test set to CSV
         test_data = pd.concat([x_test, y_test], axis=1)
         test_data.to_csv('test.csv', index=False)
```

```
y_pred = classifier.predict(x_test)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred))

print ('Accuracy:', accuracy_score(y_test, y_pred))
print ('Recall:', recall_score(y_test, y_pred, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred, average="weighted"))

confusion = confusion_matrix(y_test, y_pred)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       290
           1       0.60      0.10      0.17        30

    accuracy                           0.91       320
   macro avg       0.76      0.55      0.56       320
weighted avg       0.88      0.91      0.88       320

Accuracy: 0.909375
Recall: 0.909375
Precision: 0.8848214285714284
Confusion matrix:
[[288   2]
 [ 27   3]]
```

2. **With feature selection (Feature Importance)**

We experiment using using different number of selected features and compare the performance and accuracy between them.

- **6 features:** The 6 highest scores of feature importance were selected. This includes *alcohol, sulphates, citric acid, volatile acidity, density, and total sulphur dioxide..*

```
# Define the selected features
selected_features_1A = ['alcohol','citric acid' , 'volatile acidity','sulphate

# Select the desired features from the training and testing sets
xtrain_selected_1A = x_train[selected_features_1A]
xtest_selected_1A = x_test[selected_features_1A]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_1A, y_train)
# Make predictions on the test set
y_pred_1A = classifier.predict(xtest_selected_1A)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_1A))

print ('Accuracy:', accuracy_score(y_test, y_pred_1A))
print ('Recall:', recall_score(y_test, y_pred_1A, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_1A, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_1A)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       290
           1       0.67      0.13      0.22        30

    accuracy                           0.91       320
   macro avg       0.79      0.56      0.59       320
weighted avg       0.89      0.91      0.89       320

Accuracy: 0.9125
Recall: 0.9125
Precision: 0.8937101910828027
Confusion matrix:
[[288   2]
 [ 26   4]]
```

- **4 features:** The 4 highest scores of feature importance were selected. This includes *alcohol, sulphates, volatile acidity, and citric acid.*

```python
# Define the selected features
selected_features_1B = ['alcohol','citric acid' , 'volatile acidity','sulphates

# Select the desired features from the training and testing sets
xtrain_selected_1B = x_train[selected_features_1B]
xtest_selected_1B = x_test[selected_features_1B]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_1B, y_train)
# Make predictions on the test set
y_pred_1B = classifier.predict(xtest_selected_1B)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_1B))

print ('Accuracy:', accuracy_score(y_test, y_pred_1B))
print ('Recall:', recall_score(y_test, y_pred_1B, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_1B, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_1B)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       290
           1       0.33      0.30      0.32        30

    accuracy                           0.88       320
   macro avg       0.63      0.62      0.62       320
weighted avg       0.87      0.88      0.88       320

Accuracy: 0.878125
Recall: 0.878125
Precision: 0.8725469283276451
Confusion matrix:
[[272  18]
 [ 21   9]]
```

- **2 features:** The 2 highest scores of feature importance were selected. This includes *alcohol and sulphates.*

```
In [99]:
```

```python
# Define the selected features
selected_features_1C = ['alcohol','sulphates' ]

# Select the desired features from the training and testing sets
xtrain_selected_1C = x_train[selected_features_1C]
xtest_selected_1C = x_test[selected_features_1C]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_1C, y_train)
# Make predictions on the test set
y_pred_1C = classifier.predict(xtest_selected_1C)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_1C))

print ('Accuracy:', accuracy_score(y_test, y_pred_1C))
print ('Recall:', recall_score(y_test, y_pred_1C, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_1C, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_1C)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       290
           1       0.45      0.30      0.36        30

    accuracy                           0.90       320
   macro avg       0.69      0.63      0.65       320
weighted avg       0.89      0.90      0.89       320

Accuracy: 0.9
Recall: 0.9
Precision: 0.885
Confusion matrix:
[[279  11]
 [ 21   9]]
```

3. **With feature selection (Univariate Selection)**

Different number of selected feature using univariate selection used to be tested and

- **6 features:** The 6 highest scores of univariate selection were selected. This includes *alcohol,citric acid ,volatile acidity ,sulphates ,total sulfur dioxide and ,fixed acidity .*

```python
# Define the selected features
selected_features_2A = ['alcohol','citric acid' , 'volatile acidity','sulphate

# Select the desired features from the training and testing sets
xtrain_selected_2A = x_train[selected_features_2A]
xtest_selected_2A = x_test[selected_features_2A]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_2A, y_train)
# Make predictions on the test set
y_pred_2A = classifier.predict(xtest_selected_2A)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_2A))

print ('Accuracy:', accuracy_score(y_test, y_pred_2A))
print ('Recall:', recall_score(y_test, y_pred_2A, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_2A, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_2A)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       290
           1       0.67      0.13      0.22        30

    accuracy                           0.91       320
   macro avg       0.79      0.56      0.59       320
weighted avg       0.89      0.91      0.89       320

Accuracy: 0.9125
Recall: 0.9125
Precision: 0.8937101910828027
Confusion matrix:
[[288   2]
 [ 26   4]]
```

- **4 features:** The 4 highest scores of univariate selection were selected. This includes *alcohol, citric acid, volatile acidity, and sulphates*

```python
# Define the selected features
selected_features_2B = ['alcohol','citric acid' , 'volatile acidity','sulphate

# Select the desired features from the training and testing sets
xtrain_selected_2B = x_train[selected_features_2B]
xtest_selected_2B = x_test[selected_features_2B]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_2B, y_train)
# Make predictions on the test set
y_pred_2B = classifier.predict(xtest_selected_2B)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_2B))

print ('Accuracy:', accuracy_score(y_test, y_pred_2B))
print ('Recall:', recall_score(y_test, y_pred_2B, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_2B, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_2B)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       290
           1       0.33      0.30      0.32        30

    accuracy                           0.88       320
   macro avg       0.63      0.62      0.62       320
weighted avg       0.87      0.88      0.88       320

Accuracy: 0.878125
Recall: 0.878125
Precision: 0.8725469283276451
Confusion matrix:
[[272  18]
 [ 21   9]]
```

- 2 features: The 2 highest scores of univariate selection were selected. This includes *alcohol and citric acid.*

```python
# Define the selected features
selected_features_2C = ['alcohol','citric acid' ]

# Select the desired features from the training and testing sets
xtrain_selected_2C = x_train[selected_features_2C]
xtest_selected_2C = x_test[selected_features_2C]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_2C, y_train)
# Make predictions on the test set
y_pred_2C = classifier.predict(xtest_selected_2C)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_2C))

print ('Accuracy:', accuracy_score(y_test, y_pred_2C))
print ('Recall:', recall_score(y_test, y_pred_2C, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_2C, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_2C)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       290
           1       0.30      0.27      0.28        30

    accuracy                           0.87       320
   macro avg       0.61      0.60      0.61       320
weighted avg       0.87      0.87      0.87       320

Accuracy: 0.871875
Recall: 0.871875
Precision: 0.8659817026924536
Confusion matrix:
[[271  19]
 [ 22   8]]
```

4. With feature selection (Correlation Matrix using Heatmap)

We also used different number of features based on the score using a correlation matrix to determine which one performed better with highest accuracy.

- 6 features: The 6 highest scores of correlation matrix were selected. This includes *alcohol, citric acid, sulphates, fixed acidity, residual sugar and pH.*

```python
# Define the selected features
selected_features_3A = ['alcohol','citric acid' , 'volatile acidity','sulphate

# Select the desired features from the training and testing sets
xtrain_selected_3A = x_train[selected_features_3A]
xtest_selected_3A = x_test[selected_features_3A]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_3A, y_train)
# Make predictions on the test set
y_pred_3A = classifier.predict(xtest_selected_3A)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_3A))

print ('Accuracy:', accuracy_score(y_test, y_pred_3A))
print ('Recall:', recall_score(y_test, y_pred_3A, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_3A, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_3A)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       290
           1       0.67      0.13      0.22        30

    accuracy                           0.91       320
   macro avg       0.79      0.56      0.59       320
weighted avg       0.89      0.91      0.89       320

Accuracy: 0.9125
Recall: 0.9125
Precision: 0.8937101910828027
Confusion matrix:
[[288   2]
 [ 26   4]]
```

- 4 features:This includes *alcohol, citric acid, sulphates, and fixed acidity.*

```
# Define the selected features
selected_features_3B = ['alcohol','citric acid' , 'volatile acidity','sulphate

# Select the desired features from the training and testing sets
xtrain_selected_3B = x_train[selected_features_3B]
xtest_selected_3B = x_test[selected_features_3B]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_3B, y_train)
# Make predictions on the test set
y_pred_3B = classifier.predict(xtest_selected_3B)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_3B))

print ('Accuracy:', accuracy_score(y_test, y_pred_3B))
print ('Recall:', recall_score(y_test, y_pred_3B, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_3B, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_3B)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       290
           1       0.33      0.30      0.32        30

    accuracy                           0.88       320
   macro avg       0.63      0.62      0.62       320
weighted avg       0.87      0.88      0.88       320

Accuracy: 0.878125
Recall: 0.878125
Precision: 0.8725469283276451
Confusion matrix:
[[272  18]
 [ 21   9]]
```

- 2 features: This includes *alcohol and citric acid.* .

```
# Define the selected features
selected_features_3C = ['alcohol','citric acid' ]

# Select the desired features from the training and testing sets
xtrain_selected_3C = x_train[selected_features_3C]
xtest_selected_3C = x_test[selected_features_3C]

# Create and train the KNN model
classifier = KNeighborsClassifier(n_neighbors=18,p=2,metric='euclidean')
classifier.fit(xtrain_selected_3C, y_train)
# Make predictions on the test set
y_pred_3C = classifier.predict(xtest_selected_3C)
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_test, y_pred_3C))

print ('Accuracy:', accuracy_score(y_test, y_pred_3C))
print ('Recall:', recall_score(y_test, y_pred_3C, average="weighted"))
print ('Precision:', precision_score(y_test, y_pred_3C, average="weighted"))

confusion = confusion_matrix(y_test, y_pred_3C)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       290
           1       0.30      0.27      0.28        30

    accuracy                           0.87       320
   macro avg       0.61      0.60      0.61       320
weighted avg       0.87      0.87      0.87       320

Accuracy: 0.871875
Recall: 0.871875
Precision: 0.8659817026924536
Confusion matrix:
[[271  19]
 [ 22   8]]
```

5.8 Support Vector Machine

5.8.1 Build the SVM Model and Show the Accuracy Based on the Different Ratios

A) SVM Model for Ratio 60:40

```
# Building Model (SVM)
model1_SVM = SVC(C=1.0,kernel='rbf', random_state=0)
model2_SVM = SVC(C=1.0,kernel='linear', random_state=0)
model3_SVM = SVC(C=1.0,kernel='poly', random_state=0)
model4_SVM = SVC(C=1.0,kernel='sigmoid', random_state=0)
```

```
model1_SVM.fit(x_trainA, y_trainA) #rbf
model2_SVM.fit(x_trainA, y_trainA) #linear
model3_SVM.fit(x_trainA, y_trainA) #poly
model4_SVM.fit(x_trainA, y_trainA) #sigmoid
```

```
SVC(kernel='sigmoid', random_state=0)
```

```
prediction1_SVM = model1_SVM.predict(x_testA) #rbf
prediction3_SVM = model3_SVM.predict(x_testA) #poly
prediction4_SVM = model4_SVM.predict(x_testA) #sigmoid
```

```
y_predC = classifier.predict(x_testC)
from sklearn.metrics import precision_score, recall_score, confusion_matrix, c
# Print the evaluation metrics for the KNN model
print('Classification Report:')
print(classification_report(y_testC, y_predC))

print ('Accuracy:', accuracy_score(y_testC, y_predC))
print ('Recall:', recall_score(y_testC, y_predC, average="weighted"))
print ('Precision:', precision_score(y_testC, y_predC, average="weighted"))

confusion = confusion_matrix(y_testC, y_predC)
print('Confusion matrix:')
print(confusion)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       290
           1       0.60      0.10      0.17        30

    accuracy                           0.91       320
   macro avg       0.76      0.55      0.56       320
weighted avg       0.88      0.91      0.88       320

Accuracy: 0.909375
Recall: 0.909375
Precision: 0.8848214285714284
Confusion matrix:
[[288   2]
 [ 27   3]]
```

To get the best machine learning model, we do some experiments which are SVM Linear Kernel, SVM RBF(Radius Basis Function) Kernel, SVM Polynomial Kernel and SVM Sigmoid Kernel.

- Accuracy using SVM RBF Kernel:

```
#The accuracy when kernel = rbf

print("SCORE 60:40 (RBF Kernel)")
print("------------------------------")
print ('Accuracy:', accuracy_score(y_testA, prediction1_SVM))
print ('Recall:', recall_score(y_testA, prediction1_SVM, average="weighted"))
print ('Precision:', precision_score(y_testA, prediction1_SVM, average="weighted", zero_division=0))
confusion1_SVM = confusion_matrix(y_testA, prediction1_SVM)
print('Confusion matrix:')
print(confusion1_SVM)
```

```
SCORE 60:40 (RBF Kernel)
------------------------------
Accuracy: 0.9
Recall: 0.9
Precision: 0.8884712837837837
Confusion matrix:
[[551  23]
 [ 41  25]]
```

- Accuracy using SVM Linear Kernel

```
SCORE 60:40 (Linear Kernel)
------------------------------
Accuracy: 0.896875
Recall: 0.896875
Precision: 0.8043847656249999
Confusion matrix:
[[574   0]
 [ 66   0]]
```

- Accuracy using Polynomial Kernel

```
SCORE 60:40 (Poly Kernel)
------------------------------
Accuracy: 0.896875
Recall: 0.896875
Precision: 0.8914196735395189
Confusion matrix:
[[545  29]
 [ 37  29]]
```

- Accuracy using SVM Sigmoid Kernel

```
SCORE 60:40 (Sigmoid Kernel)
------------------------------
Accuracy: 0.859375
Recall: 0.859375
Precision: 0.8217609444768007
Confusion matrix:
[[545  29]
 [ 61   5]]
```

- SVM Model for Ratio 70:30

```
# Building Model (SVM)
model1_SVM = SVC(C=1.0,kernel='rbf', random_state=0)
model2_SVM = SVC(C=1.0,kernel='linear', random_state=0)
model3_SVM = SVC(C=1.0,kernel='poly', random_state=0)
model4_SVM = SVC(C=1.0,kernel='sigmoid', random_state=0)
```

```
In [30]:
```

```
model1_SVM.fit(x_trainB, y_trainB) #rbf
model2_SVM.fit(x_trainB, y_trainB) #linear
model3_SVM.fit(x_trainB, y_trainB) #poly
model4_SVM.fit(x_trainB, y_trainB) #sigmoid
```

```
Out[30]:
```

```
SVC
SVC(kernel='sigmoid', random_state=0)
```

```
In [31]:
```

```
prediction1_SVM = model1_SVM.predict(x_testB) #rbf
prediction2_SVM = model2_SVM.predict(x_testB) #linear
prediction3_SVM = model3_SVM.predict(x_testB) #poly
prediction4_SVM = model4_SVM.predict(x_testB) #sigmoid
```

We build 4 types of SVM Kernel which is Linear, RBF, Polynomial and Sigmoid to show which one have the best accuracy.

- Accuracy using RBF Kernel

```
SCORE 70:30 (RBF Kernel)
------------------------------
Accuracy: 0.9020833333333333
Recall: 0.9020833333333333
Precision: 0.8832875457875458
Confusion matrix:
[[419  11]
 [ 36  14]]
```

- Accuracy using Linear Kernel

```
SCORE 70:30 (Linear Kernel)
------------------------------
Accuracy: 0.8958333333333334
Recall: 0.8958333333333334
Precision: 0.8025173611111112
Confusion matrix:
[[430   0]
 [ 50   0]]
```

-Accuracy using Polynomial Kernel

```
SCORE 70:30 (Poly Kernel)
------------------------------
Accuracy: 0.8958333333333334
Recall: 0.8958333333333334
Precision: 0.883352102102102
Confusion matrix:
[[412  18]
 [ 32  18]]
```

-Accuracy using Sigmoid Kernel

```
SCORE 70:30 (Sigmoid Kernel)
------------------------------
Accuracy: 0.8958333333333334
Recall: 0.8958333333333334
Precision: 0.8025173611111112
Confusion matrix:
[[430   0]
 [ 50   0]]
```

- SVM Model for Ratio 80:20

```
# split dataset into train set and test set
x_trainC, x_testC, y_trainC, y_testC = train_test_split(x, y, test_size = 0.2, random_state = 0)

# Save training set to CSV
train_data = pd.concat([x_trainC, y_trainC], axis=1)
train_data.to_csv('train_C.csv', index=False)

# Save test set to CSV
test_data = pd.concat([x_testC, y_testC], axis=1)
test_data.to_csv('test_C.csv', index=False)
```

In [37]:

```
# Building Model (SVM)
model1_SVM = SVC(C=1.0,kernel='rbf', random_state=0)
model2_SVM = SVC(C=1.0,kernel='linear', random_state=0)
model3_SVM = SVC(C=1.0,kernel='poly', random_state=0)
model4_SVM = SVC(C=1.0,kernel='sigmoid', random_state=0)
```

In [38]:

```
model1_SVM.fit(x_trainC, y_trainC) #rbf
model2_SVM.fit(x_trainC, y_trainC) #linear
model3_SVM.fit(x_trainC, y_trainC) #poly
model4_SVM.fit(x_trainC, y_trainC) #sigmoid
```

Out[38]:

```
SVC
SVC(kernel='sigmoid', random_state=0)
```

```
prediction1_SVM = model1_SVM.predict(x_testC) #rbf
prediction2_SVM = model2_SVM.predict(x_testC) #linear
prediction3_SVM = model3_SVM.predict(x_testC) #poly
prediction4_SVM = model4_SVM.predict(x_testC) #sigmoid
```

4 types of SVM Kernel which is Linear, RBF, Polynomial and Sigmoid are created to get the best machine learning model.

- Accuracy using RBF Kernel

```
SCORE 80:20 (RBF Kernel)
------------------------------
Accuracy: 0.915625
Recall: 0.915625
Precision: 0.9001024590163935
Confusion matrix:
[[284   6]
 [ 21   9]]
```

- Accuracy using Linear Kernel

```
SCORE 80:20 (Linear Kernel)
------------------------------
Accuracy: 0.90625
Recall: 0.90625
Precision: 0.8212890625
Confusion matrix:
[[290   0]
 [ 30   0]]
```

- Accuracy using Polynomial Kernel

```
SCORE 80:20 (Poly Kernel)
------------------------------
Accuracy: 0.90625
Recall: 0.90625
Precision: 0.8953439597315436
Confusion matrix:
[[279  11]
 [ 19  11]]
```

- Accuracy using Sigmoid Kernel

```
SCORE 80:20 (Sigmoid Kernel)
------------------------------
Accuracy: 0.90625
Recall: 0.90625
Precision: 0.8212890625
Confusion matrix:
[[290   0]
 [ 30   0]]
```

So, as we can see, the dataset with a ratio 80:20 and using RBF Kernel has the highest score compared to others.

5.8.1 Build the SVM Model

1. Without Feature Selection

```
#SVM RBF Kernel

classfier = SVC(kernel='rbf', C=1)
classfier.fit(xtrain, ytrain)

y_predict = classfier.predict(xtest)
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_predict))
print("F1 Score: ", f1_score(ytest, y_predict, average='weighted'))
print("Recall: ", recall_score(ytest, y_predict, average='weighted'))
print("Precision: ", precision_score(ytest, y_predict, average='weighted', zero_division=0))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.915625
F1 Score:  0.9026260504201682
Recall:  0.915625
Precision:  0.9001024590163935
```

We take the highest accuracy which is ratio 80:20 and using RBF Kernel.

2. With Feature Selection

- **Using 6 Features**

I) Feature Importance

We use features the highest score six features : *alcohol, sulphates, volatile acidity, density, total sulfur dioxide.*

```
selected_features_1A = [ 'alcohol','sulphates','volatile acidity','citric acid','density','total sulfur dioxide']

# Select the desired features from the training and testing sets
xtrain_selected_1A = xtrain[selected_features_1A]
xtest_selected_1A = xtest[selected_features_1A]

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_1A, ytrain)

# Make predictions on the test set
y_pred_1A = model_linear.predict(xtest_selected_1A)

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_1A))
print("F1 Score: ", f1_score(ytest, y_pred_1A, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_1A, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_1A, average='weighted', zero_division=0))
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_1A))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.921875
F1 Score:  0.907544018542324
Recall:  0.921875
Precision:  0.9091628038085693
Confusion matrix:
 [[286   4]
 [ 21   9]]
```

II) Univariate Selection

- The 6 highest scores of univariate selection were selected. This includes*alcohol,citric acid ,volatile acidity ,sulphates ,total sulfur dioxide and ,fixed acidity* .

```
SCORE (RBF Kernel)
------------------------
Accuracy:  0.925
F1 Score:  0.9123703477
Recall:  0.925
Precision:  0.913982259
Confusion matrix:
 [[286   4]
```

III) Correlation Matrix with Heatmap

- The 6 highest scores of correlation matrix were selected. This includes *alcohol, citric acid, sulphates, fixed acidity, residual sugar and pH.*

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.921875
F1 Score:  0.905011910094059
Recall:  0.921875
Precision:  0.9099091644601354
Confusion matrix:
 [[287   3]
 [ 22   8]]
```

Then, we also use different number of features for prediction which are 4 and 2.

- **Using 4 Features**

I) Feature Importance -Four highest features which is : *alcohol, sulphates, volatile acidity, citric acid.*

```
# Perform feature selection and store the selected feature indices in a list
selected_features_1B = [ 'alcohol','sulphates','volatile acidity','citric acid']  # Replace with the selected features

# Select the desired features from the training and testing sets
xtrain_selected_1B = xtrain[selected_features_1B]
xtest_selected_1B = xtest[selected_features_1B]

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_1B, ytrain)

# Make predictions on the test set
y_pred_1B = model_linear.predict(xtest_selected_1B)

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_1B))
print("F1 Score: ", f1_score(ytest, y_pred_1B, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_1B, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_1B, average='weighted', zero_division=0))
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_1B))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.9000461859870852
Recall:  0.909375
Precision:  0.8953761584193041
Confusion matrix:
 [[281   9]
 [ 20  10]]
```

II) Univariate Selection-The 4 highest scores of univariate selection were selected. This includes *alcohol, citric acid, volatile acidity, and sulphates*

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.9000461859870852
Recall:  0.909375
Precision:  0.8953761584193041
Confusion matrix:
 [[281   9]
 [ 20  10]]
```

III) Correlation Matrix with Heatmap -The highest 4 scores of correlation matrix includes *alcohol, citric acid, sulphates, and fixed acidity.*

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.91875
F1 Score:  0.9050678767541184
Recall:  0.91875
Precision:  0.9043242296918768
Confusion matrix:
 [[285   5]
 [ 21   9]]
```

Using 2 Features

I) Feature Importance

- 2 features: This includes *alcohol and sulphates*.

```
# Perform feature selection and store the selected feature indices in a list
selected_features_1C = [ 'alcohol','sulphates']  # Replace with the selected features

# Select the desired features from the training and testing sets
xtrain_selected_1C = xtrain[selected_features_1C]
xtest_selected_1C = xtest[selected_features_1C]

# Create and train the linear SVM model
model_linear = SVC(kernel='rbf', C=1)
model_linear.fit(xtrain_selected_1C, ytrain)

# Make predictions on the test set
y_pred_1C = model_linear.predict(xtest_selected_1C)

# Print the evaluation metrics for the linear SVM model
print("SCORE (RBF Kernel)")
print("------------------------------")
print("Accuracy: ", accuracy_score(ytest, y_pred_1C))
print("F1 Score: ", f1_score(ytest, y_pred_1C, average='weighted'))
print("Recall: ", recall_score(ytest, y_pred_1C, average='weighted'))
print("Precision: ", precision_score(ytest, y_pred_1C, average='weighted', zero_division=0))
print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_1C))
```

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.8927510615090958
Recall:  0.909375
Precision:  0.8888358180907041
Confusion matrix:
 [[284   6]
 [ 23   7]]
```

II) Univariate Selection -This includes highest 2 feature of univariate selection includes *alcohol and citric acid.*

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.890625
F1 Score:  0.8793660865361372
Recall:  0.890625
Precision:  0.8715411348137787
Confusion matrix:
 [[278  12]
 [ 23   7]]
```

III) Correlation Matrix with HeatMap- This includes highest 2 feature of correlation matrix includes *alcohol and citric acid.*

```
SCORE (RBF Kernel)
------------------------------
Accuracy:  0.909375
F1 Score:  0.8927510615090958
Recall:  0.909375
Precision:  0.8888358180907041
Confusion matrix:
 [[284   6]
 [ 23   7]]
```

As we can see, six features selected using Univariate Selection have the highest score in Model Evaluation.

- **Using Different Number of Cost**

The last step, we use different number of cost which are 1,10,100 by SVM RBF Kernel.

```
# Perform feature selection and store the selected feature indices in a list
selected_features_2A = [ 'alcohol','citric acid','volatile acidity','sulphates','total sulfur dioxide','fixed acidity

# Select the desired features from the training and testing sets
xtrain_selected_2A = xtrain[selected_features_2A]
xtest_selected_2A = xtest[selected_features_2A]

costs= [1,100,1000]#using different cost value

for cost in costs:
    model_linear = SVC(kernel='rbf', C=cost)
    model_linear.fit(xtrain_selected_2A, ytrain)
    #perform prediction
    # Make predictions on the test set
    y_pred_2A = model_linear.predict(xtest_selected_2A)

    # Print the evaluation metrics for the linear SVM model
    print(f"SCORE (RBF Kernel) when cost= {cost}")
    print("------------------------------")
    print("Accuracy: ", accuracy_score(ytest, y_pred_2A))
    print("F1 Score: ", f1_score(ytest, y_pred_2A, average='weighted'))
    print("Recall: ", recall_score(ytest, y_pred_2A, average='weighted'))
    print("Precision: ", precision_score(ytest, y_pred_2A, average='weighted', zero_division=0))
    print('Confusion matrix: \n',confusion_matrix(ytest, y_pred_2A))
```

```
SCORE (RBF Kernel) when cost= 1
------------------------------
Accuracy:  0.925
F1 Score:  0.9123703477730322
Recall:  0.925
Precision:  0.913982259570495
Confusion matrix:
 [[286   4]
 [ 20  10]]
SCORE (RBF Kernel) when cost= 100
------------------------------
Accuracy:  0.915625
F1 Score:  0.9195172696725795
Recall:  0.915625
Precision:  0.9249027074777958
Confusion matrix:
 [[273  17]
 [ 10  20]]
SCORE (RBF Kernel) when cost= 1000
------------------------------
Accuracy:  0.9125
F1 Score:  0.9160199556541018
Recall:  0.9125
Precision:  0.9206279342723004
Confusion matrix:
 [[273  17]
 [ 11  19]]
```

We can conclude that, accuracy not getting higher if we used different values o

6.0 RESULTS AND DISCUSSION

6.1 Division Data Set

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

To prepare the training and test datasets, we utilized the "train_test_split" library from "sklearn.model_selection". This library enables the division of the original dataframe into two separate dataframes, with the split ratio determined by the user. The performance of the machine learning model can vary based on the chosen ratio, as the model learns from different examples provided during training. Therefore, having an adequate sample size is crucial for ensuring the model's effectiveness.

In our project, we experimented with three different ratios to partition the original dataframe: 60/40, where 60% of the data was used for the training set and 40% for the test set; 70/30, with 70% for training and 30% for testing; and finally, 80/20, where 80% of the data was assigned to the training set and 20% to the test set.

1. ***Ratio of 60:40***

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
117	7.8	0.560	0.12	2.0	0.082	7.0	28.0	0.99700	3.37	0.50	9.4
991	7.1	0.340	0.28	2.0	0.082	31.0	68.0	0.99694	3.45	0.48	9.4
252	11.1	0.350	0.48	3.1	0.090	5.0	21.0	0.99860	3.17	0.53	10.5
507	11.2	0.670	0.55	2.3	0.084	6.0	13.0	1.00000	3.17	0.71	9.5
1587	5.8	0.610	0.11	1.8	0.066	18.0	28.0	0.99483	3.55	0.66	10.9
...	...	...	...	...	...	...	...	...	...	...	...
763	9.3	0.655	0.26	2.0	0.096	5.0	35.0	0.99738	3.25	0.42	9.6
835	7.6	0.665	0.10	1.5	0.066	27.0	55.0	0.99655	3.39	0.51	9.3
1216	7.9	0.570	0.31	2.0	0.079	10.0	79.0	0.99677	3.29	0.69	9.5
559	13.0	0.470	0.49	4.3	0.085	6.0	47.0	1.00210	3.30	0.68	12.7
684	9.8	0.980	0.32	2.3	0.078	35.0	152.0	0.99800	3.25	0.48	9.4

959 rows × 11 columns

959 rows for train set

1109	10.8	0.470	0.43	2.10	0.171	27.0	66.0	0.99820	3.17	0.76	10.8
1032	8.1	0.820	0.00	4.10	0.095	5.0	14.0	0.99854	3.36	0.53	9.6
1002	9.1	0.290	0.33	2.05	0.063	13.0	27.0	0.99516	3.26	0.84	11.7
487	10.2	0.645	0.36	1.80	0.053	5.0	14.0	0.99820	3.17	0.42	10.0
979	12.2	0.450	0.49	1.40	0.075	3.0	6.0	0.99690	3.13	0.63	10.4
...	...	...	...	...	...	...	...	...	...	...	...
912	10.0	0.460	0.44	2.90	0.065	4.0	8.0	0.99674	3.33	0.62	12.2
349	9.1	0.785	0.00	2.60	0.093	11.0	28.0	0.99940	3.36	0.86	9.4
185	8.9	0.310	0.57	2.00	0.111	26.0	85.0	0.99710	3.26	0.53	9.7
1433	6.1	0.400	0.16	1.80	0.069	11.0	25.0	0.99550	3.42	0.74	10.1
371	7.9	0.240	0.40	1.60	0.056	11.0	25.0	0.99670	3.32	0.87	8.7

640 rows × 11 columns

640 rows for test set

2. ***Ratio of 70:30***

Out[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
92	8.6	0.490	0.29	2.0	0.110	19.0	133.0	0.99720	2.93	1.98	9.8
1017	8.0	0.180	0.37	0.9	0.049	36.0	109.0	0.99007	2.89	0.44	12.7
1447	6.8	0.670	0.00	1.9	0.080	22.0	39.0	0.99701	3.40	0.74	9.7
838	10.1	0.310	0.35	1.6	0.075	9.0	28.0	0.99672	3.24	0.83	11.2
40	7.3	0.450	0.36	5.9	0.074	12.0	87.0	0.99780	3.33	0.83	10.5
...	...	...	...	...	...	...	...	...	...	...	...
763	9.3	0.655	0.26	2.0	0.096	5.0	35.0	0.99738	3.25	0.42	9.6
835	7.6	0.665	0.10	1.5	0.066	27.0	55.0	0.99655	3.39	0.51	9.3
1216	7.9	0.570	0.31	2.0	0.079	10.0	79.0	0.99677	3.29	0.69	9.5
559	13.0	0.470	0.49	4.3	0.085	6.0	47.0	1.00210	3.30	0.68	12.7
684	9.8	0.980	0.32	2.3	0.078	35.0	152.0	0.99800	3.25	0.48	9.4

1119 rows × 11 columns

1119 rows for train set

1109	10.8	0.470	0.43	2.10	0.171	27.0	66.0	0.99820	3.17	0.76	10.8
1032	8.1	0.820	0.00	4.10	0.095	5.0	14.0	0.99854	3.36	0.53	9.6
1002	9.1	0.290	0.33	2.05	0.063	13.0	27.0	0.99516	3.26	0.84	11.7
487	10.2	0.645	0.36	1.80	0.053	5.0	14.0	0.99820	3.17	0.42	10.0
979	12.2	0.450	0.49	1.40	0.075	3.0	6.0	0.99690	3.13	0.63	10.4
...	...	...	...	...	...	...	...	...	...	...	...
801	8.6	0.550	0.09	3.30	0.068	8.0	17.0	0.99735	3.23	0.44	10.0
61	7.7	0.690	0.49	1.80	0.115	20.0	112.0	0.99680	3.21	0.71	9.3
431	7.8	0.550	0.35	2.20	0.074	21.0	66.0	0.99740	3.25	0.56	9.2
1210	6.8	0.650	0.02	2.10	0.078	8.0	15.0	0.99498	3.35	0.62	10.4
713	8.0	0.430	0.36	2.30	0.075	10.0	48.0	0.99760	3.34	0.46	9.4

480 rows × 11 columns

480 rows for test set

3. ***Ratio of 80:20***

Out[84]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
642	9.9	0.540	0.45	2.3	0.071	16.0	40.0	0.99910	3.39	0.62	9.4
679	10.8	0.260	0.45	3.3	0.060	20.0	49.0	0.99720	3.13	0.54	9.6
473	9.9	0.350	0.55	2.1	0.062	5.0	14.0	0.99710	3.26	0.79	10.6
390	5.6	0.850	0.05	1.4	0.045	12.0	88.0	0.99240	3.56	0.82	12.9
1096	6.6	0.725	0.09	5.5	0.117	9.0	17.0	0.99655	3.35	0.49	10.8
...	...	...	...	...	...	...	...	...	...	...	...
763	9.3	0.655	0.26	2.0	0.096	5.0	35.0	0.99738	3.25	0.42	9.6
835	7.6	0.665	0.10	1.5	0.066	27.0	55.0	0.99655	3.39	0.51	9.3
1216	7.9	0.570	0.31	2.0	0.079	10.0	79.0	0.99677	3.29	0.69	9.5
559	13.0	0.470	0.49	4.3	0.085	6.0	47.0	1.00210	3.30	0.68	12.7
684	9.8	0.980	0.32	2.3	0.078	35.0	152.0	0.99800	3.25	0.48	9.4

1279 rows × 11 columns

1279 rows for train set

Out[85]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
1109	10.8	0.470	0.43	2.10	0.171	27.0	66.0	0.99820	3.17	0.76	10.8
1032	8.1	0.820	0.00	4.10	0.095	5.0	14.0	0.99854	3.36	0.53	9.6
1002	9.1	0.290	0.33	2.05	0.063	13.0	27.0	0.99516	3.26	0.84	11.7
487	10.2	0.645	0.36	1.80	0.053	5.0	14.0	0.99820	3.17	0.42	10.0
979	12.2	0.450	0.49	1.40	0.075	3.0	6.0	0.99690	3.13	0.63	10.4
...	...	...	...	...	...	...	...	...	...	...	...
794	10.1	0.270	0.54	2.30	0.065	7.0	26.0	0.99531	3.17	0.53	12.5
813	6.9	0.390	0.24	2.10	0.102	4.0	7.0	0.99462	3.44	0.58	11.4
1322	9.1	0.340	0.42	1.80	0.058	9.0	18.0	0.99392	3.18	0.55	11.4
704	9.1	0.765	0.04	1.60	0.078	4.0	14.0	0.99800	3.29	0.54	9.7
1023	8.2	0.320	0.42	2.30	0.098	3.0	9.0	0.99506	3.27	0.55	12.3

320 rows × 11 columns

320 rows for test set

To find out the most optimal ratio for the data splitting, we had undergone different experiment under 3 different model, and the manipulating value of those experiments is a different ratio. For instance, we use 80/20, 70/30, and also 60/40.

Ratio for data splitting	Random Forest	SVM	KNN
Condition	Random state: 0	C-value: 1 Kernel:Radius Basis Function	K-value: 18

60/40	Accuracy: 0.91875 Recall: 0.91875 Precision: 0.9198	Accuracy: 0.9 Recall: 0.9 Precision: 0.8885	Accuracy: 0.8921 Recall:0.8921 Precision: 0.8492
70/30	Accuracy: 0.91875 Recall: 0.91875 Precision: 0.9153	Accuracy: 0.9021 Recall: 0.9021 Precision: 0.8833	Accuracy:0.8895 Recall:0.8895 Precision: 0.8242
80/20	Accuracy: 0.9281 Recall: 0.9281 Precision: 0.9232	Accuracy: 0.9156 Recall: 0.9156 Precision: 0.9001	Accuracy:0.9093 Recall: 0.9093 Precision: 0.8848

6.2 Feature Selection

Feature selection is a technique employed in machine learning to reduce the number of input variables when constructing predictive models. Its purpose is to identify and retain only the most relevant features, thereby enhancing model performance and reducing the computational burden.

1. ***Feature Importance***

Prior to evaluating the accuracy of each machine learning model on the dataset, we employed Feature Importance to select the appropriate features. By determining the significance of each feature, we gained insight into their relative importance, enabling us to make informed decisions about accuracy assessment. The table below presents a description of each feature, its corresponding feature importance score, and its ranking.

Feature	Description	Feature Importance Score	Ranking
alcohol	the percentage of ethanol present in a beverage	0.1777	1
sulphates	as preservatives and antioxidants to prevent spoilage and maintain freshness	0.1054	2
volatile acidity	the concentration of volatile acids	0.1025	3
citric acid	the overall acidity and freshness of the flavor profile	0.0968	4
total sulfur dioxide	the total concentration of both free and bound forms of sulfur dioxide	0.0820	5
density	the level of concentration or richness	0.0806	6
fixed acidity	the overall tartness and acidity of the wine	0.0767	7

residual sugar	the perceived sweetness of the wine	0.0726	8
chlorides	can affect the taste and perception of saltiness	0.0697	9
free sulfur dioxide	the total concentration of free forms of sulfur dioxide	0.0684	10
pH	pH of the wine(acidic or alkaline)	0.0663	11

Hence, the top 6 features with the highest feature importance value that we use to construct the machine learning model which is alcohol, sulphates,volatile acidity, citric acid, total sulfur dioxide and density.

2. Correlation Matrix Using Heatmap

Feature	Correlation matrix Score	Ranking
alcohol	0.41	1
citric acid	0.21	2
sulphates	0.2	3
fixed acidity	0.12	4
residual sugar	0.048	5
pH	-0.057	6
free sulfur dioxide	-0.072	7
chlorides	-0.097	8
total sulfur dioxide	-0.14	9
density	-0.15	10

From the table, the 6 features that we choose is alcohol, citric acid, sulphates, fixed acidity, residual sugar, and pH.

3. Univariate Selection

To enhance the accuracy of the results, we incorporate Univariate Selection as an additional form of feature selection. This technique involves selecting the most optimal feature based on a statistical test that examines each feature individually. Through the implementation of Univariate Selection, we obtain a new set of feature scores. The following table displays the score assigned to each feature during the univariate selection process.

Feature	Univariate selection score	Ranking
alcohol	22.896820	1
citric acid	10.317077	2
volatile acidity	6.306147	3

sulphates	3.334157	4
total sulfur dioxide	2.939007	5
fixed acidity	1.661298	6
density	1.417023	7
free sulfur dioxide	0.852219	8
chlorides	0.741466	9
residual sugar	0.303084	10

We also chose 6 Features from the table above, which are alcohol, citric acid, volatile acidity, sulphates, total sulfur dioxide, fixed acidity.

6.2.4 Result under Feature Selection

The various techniques employed for feature selection, such as Feature Importance, Correlation Matrix with Heatmap, and Univariate Selection, bring differents result on accuracy. We conducted three experiments for each feature selection method, utilizing KNN, Random Forest, and SVM as the machine learning algorithms. Each algorithm produced distinct results from one another. The table below presents the outcomes for the three ML models.

Feature Selection	SVM	KNN	Random Forest
Condition	C-value: 1 Kernel: RBF	k-value=18	Random state=0
Feature Importance	Accuracy: 0.9219 F1 Score:0.9075 Recall: 0.9219 Precision: 0.9092	Accuracy: 0.9125 F1 Score: 0.89 Recall: 0.9125 Precision: 0.8983	Accuracy: 0.91875 F1 Score: 0.91991 Recall: 0.91875 Precision: 0.92122
Correlation Matrix with Heatmap	Accuracy: 0.925 F1 Score: 0.9124 Recall: 0.925 Precision: 0.9140	Accuracy: 0.9125 F1 Score: 0.91 Recall: 0.9125 Precision: 0.8983	Accuracy: 0.940625 F1 Score: 0.93817 Recall: 0.940625 Precision: 0.93688
Univariate Selection	Accuracy: 0.9219 F1 Score: 0.9050 Recall: 0.9219 Precision:0.9099	Accuracy: 0.9125 F1 Score: 0.91 Recall: 0.9125 Precision: 0.8983	Accuracy: 0.925 F1 Score: 0.9238 Recall: 0.925 Precision: 0.9228

3. Results And Discussion for Random Forest

For the first part of the experiment, we tested the model without feature selection using 3 different ratios which are 60:40, 70:30, and 80:20. The largest portion of the set is used for the training set and the rest for the testing set. The training set is used to train the machine learning model while the testing set is used to evaluate the model's performance. Among these 3 different ratios, ratio of 80:20 recorded with the highest score of accuracy which is 0.9281. Thus, we can sum up that the best random forest model without feature selection recorded when the data splitting ratio used is 80:20.

After that, we tested the model by implementing feature selection with different number of features to enhance the model's performance. For this part, we finalized to use data splitting ratio of 80:20 since it gives the best performance based on the accuracy and recall score. The experiment was conducted with 3 different feature selections including feature importance, univariate selection, and correlation matrix using heatmap. Among the 3 feature selections, correlation matrix using heatmap with 6 number of features recorded as the best model which gives the higest accuracy score and recall score which are 0.940625 both.

In a nutshell, this experiment successfully identifies the best optimal model for random forest algorithm. By considering the accuracy and recall score, we can conclude that the machine learning algorithm that utilizing correlation matrix using heatmap, with 6 selected features give the best performance.

Condition	Feature selection	Ratio	Number of features	Result
Without feature selection	-	60:40	-	Accuracy: 0.9187 F1 Score: 0.9192 Recall: 0.9187 Precision: 0.9198
		70:30		Accuracy: 0.9187 F1 Score: 0.9168 Recall: 0.9187 Precision: 0.9153
		80:20		Accuracy: 0.9281 F1 Score: 0.9251 Recall: 0.9281 Precision: 0.9232
	Feature importance	80:20	2	Accuracy: 0.8687 F1 Score: 0.8755 Recall: 0.86875 Precision: 0.8836

With feature selection			4	Accuracy: 0.9062 F1 Score: 0.9111 Recall: 0.9062 Precision: 0.9177
			6	Accuracy: 0.9187 F1 Score: 0.9199 Recall: 0.9187 Precision: 0.9212
	Univariate selection	80:20	2	Accuracy: 0.8687 F1 Score: 0.8769 Recall: 0.8687 Precision: 0.8872
			4	Accuracy: 0.9062 F1 Score: 0.9111 Recall: 0.9062 Precision: 0.9177
			6	Accuracy: 0.925 F1 Score: 0.9238 Recall: 0.925 Precision: 0.9228
	Correlation matrix with heatmap	80:20	2	Accuracy: 0.8687 F1 Score: 0.8769 Recall: 0.8687 Precision: 0.8872
			4	Accuracy: 0.9031 F1 Score: 0.9075 Recall: 0.9031 Precision: 0.9134
			6	Accuracy: 0.9406 F1 Score: 0.9381 Recall: 0.9406 Precision: 0.9368

4. ***Results And Discussion for KNN (K-Nearest Neighbours)***

For K-Nearest Neighbours, we had done a few experiment to test and find which model presented the highest performance. Firstly, we have done the experiment without using any feature selection. Before that we found the K value with the highest accuracy using iteration in which each iteration creates a KNN classifier with the current K value. The accuracy score

for each value of k is stored in the scores list. From the test and the information we determined the optimal value of k for the KNN classifier is 18.

Next we tested the model with a different number of data splitting ratios which are 60:40, 70:30 and 80:20. The largest portion of the data ratio is stored in the training set while the rest in the testing set. At the end, we found that the ratio of 80:20 gave the best performance of the model.

After that, we conducted experiments using feature selection which are feature importance, univariate selection, and correlation matrix using heatmap to boost the performance of the model. Since the ratio of 80:20 performed well, we use the ratio in order to test the implementation of feature selection in the model.

Condition	Feature selection	K-value	Ratio	Number of features	Results
Without feature selection	-	18	60:40	-	Accuracy: 0.89218 F1 Score: 0.89 Recall: 0.89218 Precision: 0.84928
			70:30		Accuracy: 0.88958 F1 Score: 0.89 Recall: 0.88958 Precision: 0.82425
			80:20		Accuracy: 0.90937 F1 Score: 0.91 Recall: 0.90937 Precision: 0.88482
With feature selection	Feature importance	18	80:20	6	Accuracy: 0.9125 F1 Score: 0.91 Recall: 0.9125 Precision: 0.89371
		18	80:20	4	Accuracy: 0.878125 F1: Score: 0.88 Recall: 0.878125 Precision: 0.87254
		18	80:20	2	Accuracy: 0.9

					F1 Score: 0.9 Recall: 0.9 Precision: 0.8885
	Univariate Selection	18	80:20	6	Accuracy: 0.91562 F1 Score: 0.92 Recall: 0.91562 Precision: 0.92281
		18	80:20	4	Accuracy: 0.909375 F1 Score: 0.91 Recall: 0.909375 Precision: 0.88870
		18	80:20	2	Accuracy: 0.90625 F1 Score: 0.91 Recall: 0.90625 Precision: 0.87047
	Correlation matrix using heatmap	18	80:20	6	Accuracy: 0.903125 F Score: 0.90 Recall: 0.903125 Precision: 0.89027
		18	80:20	4	Accuracy: 0.903125 F1 Score: 0.90 Recall: 0.903125 Precision: 0.89311
		18	80:20	2	Accuracy: 0.871875 F1 Score: 0.87 Recall: 0.871875 Precision: 0.865981

5. Results And Discussion for SVM (Support Vector Machine)

In Support Vector Machine, firstly, we start with the experiment without doing feature selection. We start the experiment by using the kernel tricks and different ratios. In order to find the most optimal model in this kernel SVM, we did 4 types of kernel which is RBF, Linear, Polynomial, Sigmoid for each three types ratios. Among these 3 Ratios and 4 types kernel, RBF Kernel with ratio 80:20 has the best accuracy value(0.9156) and recall value(0.9156). Hence, with the result above, we can draw the conclusion that the best model

that in SVM without feature selection is the model that uses the radius basis function as the kernel, the value of 1 as the C-value and used the ratio 80:20.

In the second part of the experiment, we involve feature selection in the experiment as we would like to boost the accuracy of the model again. As the RBF Kernel with ratio 80:20 is most accurate without feature selection, so we chose it to remake on involving the feature selection. The experiment can be divided into 3 parts, one of part uses a feature importance, the second part uses univariate selection while the last part uses correlation matrix with heatmap. Some changes occurred in all this feature selection invloving experiment. This time, the model with the highest performance falls on the experiment invloving feature selection Univariate Selection with 6 Features. They recorded the highest accuracy among the 3 feature selection, which are 0.925, and their recall value is 0.925.

In the third part of the experiment, we use the different number of cost which is c=1, c=100, c=1000 using 6 features and RBF kernel since it has a highest performance among the other kernels. However, the accuracy does not improve. In fact, it shows a lower accuracy value. For the 3 experiments, C=1 has the highest accuracy and recall number, which are 0.925 and 0.925 respectively.

In Conclusion, we can find the best model for SVM. The model that uses radius basis function as kernel and 1 for C-value for univariate selection category is the model with the best performance.

Kernel	Ratio/Number Features	C-Value	Result
WITHOUT FEATURE SELECTION			
RBF	Ratio 60:40	1	Accuracy: 0.9 Recall: 0.9 Precision: 0.8885
Linear		1	Accuracy: 0.8969 Recall: 0.8969 Precision: 0.8044
Polynomial		1	Accuracy: 0.8594 Recall: 0.8594 Precision: 0.8218
Sigmoid		1	Accuracy: 0.8594 Recall: 0.8594 Precision: 0.8218
RBF	Ratio 70:30	1	Accuracy: 0.9021 Recall: 0.9021 Precision: 0.8833

Linear		1	Accuracy: 0.8958 Recall: 0.8958 Precision: 0.8025
Polynomial		1	Accuracy: 0.8958 Recall: 0.8958 Precision: 0.8834
Sigmoid		1	Accuracy: 0.8958 Recall: 0.8958 Precision: 0.8025
RBF	Ratio 80:20	1	Accuracy: 0.9156 Recall: 0.9156 Precision: 0.9001
Linear		1	Accuracy: 0.9063 Recall: 0.9063 Precision: 0.8129
Polynomial		1	Accuracy: 0.9063 Recall: 0.9063 Precision: 0.8953
Sigmoid		1	Accuracy: 0.9063 Recall: 0.9063 Precision: 0.8213
FEATURE SELECTION (FEATURE IMPORTANCES)			
RBF	6 Features	1	Accuracy: 0.9219 F1 Score: 0.9075 Recall: 0.9219 Precision: 0.9092
	4 Features	1	Accuracy: 0.9094 F1 Score: 0.9000 Recall: 0.9094 Precision: 0.8954

	2 Features	1	Accuracy: 0.9094 F1 Score: 0.8928 Recall: 0.9094 Precision: 0.8888
FEATURE SELECTION (UNIVARIATE SELECTION)			
RBF	6 Features	1	Accuracy: 0.925 F1 Score: 0.9124 Recall: 0.925 Precision: 0.9140
	4 features	1	Accuracy: 0.9094 F1 Score: 0.9000 Recall: 0.9094 Precision: 0.8954
	2 Features	1	Accuracy: 0.8906 F1 Score: 0.8794 Recall: 0.8906 Precision: 0.8715
FEATURE SELECTION (CORRELATION MATRIX WITH HEATMAP)			
RBF	6 Features	1	Accuracy: 0.9219 F1 Score: 0.9050 Recall: 0.9219 Precision: 0.9099
	4 Features	1	Accuracy: 0.9188 F1 Score: 0.9051 Recall: 0.9188 Precision: 0.9043
	2 Features	1	Accuracy: 0.9094 F1 Score: 0.8928 Recall: 0.9094 Precision: 0.8888

DIFFERENT NUMBER OF COST (C = 1, 100, 1000)			
RBF	6 Features	1	Accuracy: 0.925 F1 Score: 0.9124 Recall: 0.925 Precision: 0.9140
		100	Accuracy: 0.9156 F1 Score: 0.9195 Recall: 0.9156 Precision: 0.9249
		1000	Accuracy: 0.9125 F1 Score: 0.9160 Recall: 0.9125 Precision: 0.9206

7.0 COMPARISON AND RECOMMENDATION

Algorithm	**SVM**	**KNN**	**Random Forest**
Feature selection	Univariate and its selection 6-features	Univariate selection and its top 6-features	Correlation Heatmap and its Top 6-features
	C=1,RBF kernel	k-value=18	Random state = 0
Dataset Split ratio	80:20	80:20	80:20
Accuracy	Accuracy: 0.925	Accuracy: 0.91562	Accuracy: 0.9406
Recall	Recall: 0.925	Recall: 0.91562	Recall: 0.9406
F1-score	F1 Score: 0.9124	F1 Score: 0.92	F1 Score: 0.9382
Precision	Precision: 0.9140	Precision: 0.92281	Precision: 0.9369

From the overall result, we can see that 2 of the algorithms, SVM and KNN shows the best accuracy and overall scores when using univariate selection as their feature selection. But Random forest algorithm shows best overall accuracy and scores when using the correlation map as the feature selection. All of the algorithm shows best accuracy when the number of feature selected was the top 6 features. The top 6 features will vary for every feature selection methods because each machine learning algorithm has its own unique method to identify top 6 features. We tested all the algorithms using 320 rows of data that has 6 features including, alcohol, citric acid, sulphates, fixed acidity, residual sugar and pH to get fair results from

each algorithm. In all 3 algorithms, the dataset was split into train set and test set with a ratio of 80: 20 which means 80% train set and 20% test set. The highest accuracy for the KNN algorithm is 0.9215 and the recall is 0.94 when the K value is equal to 18. For the SVM algorithm, its highest accuracy is 0.925 and its recall is 0.925 at C equal to 1. Lastly, for the random forest algorithm, the highest accuracy is 0.9406 and the recall is 0.9406 when the random state is equal to 0.

When we classify the quality of the wine from 3 to 8 into 1(good) and 0(bad), we able to make our machine learning algorithms to give higher overall scores ex. accuracy , prediction, recall and f1 score. This is because the model easily able to identify the binary numbers and process it to give the best results possible. We can evaluate the models for the wine quality dataset using metrics like accuracy, precision, recall, and F1 score. These metrics provide a comprehensive comprehension of the model's performance across multiple classes. We can also think about methods like feature selection with correlation matrices to find the most crucial characteristics for predicting wine quality.

Random Forest emerged as the champion model in our analysis of the wine quality dataset. This is due to its high accuracy of 0.9406 when using the correlation matrix for feature selection. Random Forest is a good choice for this job because it can handle complex relationships and find non-linear patterns in the data.For wine quality prediction, we recommend the Random Forest algorithm. Based on the features of the dataset, it can make predictions that are both accurate and reliable. Additionally, utilising the correlation matrix for feature selection can assist in determining the most significant factors influencing wine quality, facilitating a better comprehension and interpretation of the findings than univariate selection.

There are some recommendation to do discussed so that our model can produce even more accurate and reliable predictions. First is Hyperparameter tuning. Optimize the Random Forest algorithm's performance by fine-tuning its hyperparameters. This can be accomplished by adjusting parameters like the number of trees in the forest, the maximum depth of each tree, or the minimum number of samples required to split a node using methods like grid search or random search. Cross validation comes next. Cross-validation methods like k-fold cross-validation are used to get estimates of the model's performance that are more accurate. To create a more reliable evaluation metric, the data are divided into multiple subsets, the model is trained and evaluated on various combinations of these subsets, and the results are averaged to create a metric.

8.0 CONCLUDING REMARKS

Based on the evaluation and comparison of Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Random Forest on the wine quality dataset, it can be concluded that Random Forest is the best machine learning model for predicting wine quality.Random Forest, as an ensemble learning method, combines multiple decision trees to make predictions. This approach allows the model to capture complex relationships and non-linear

patterns in the data, making it highly effective for predicting wine quality. The combination of multiple decision trees also helps to reduce overfitting and improve the generalisation ability of the model.While SVM and KNN are valuable algorithms, Random Forest has demonstrated superior performance in the context of the wine quality dataset. It provides robust and reliable predictions, making it the champion model for this specific task.
However, it's worth noting that the selection of the best machine learning model depends on various factors, including the specific dataset, its characteristics, and the evaluation metrics used. It's always recommended to thoroughly evaluate multiple algorithms and consider the unique requirements of the problem at hand before making a final decision. We also need to take note that the features selected is also vary for all the feature selection methods.

9.0 LESSON LEARN FROM THE PROJECT

During the analysis of the wine quality dataset, several key findings emerged. Firstly, it was observed that as the number of features increased, the data became more prone to overfitting. Overfitting occurs when a model becomes too complex and captures noise or irrelevant patterns in the data, leading to reduced generalisation performance. Therefore, it is crucial to carefully consider the number of features used in order to balance accuracy and overfitting.To address the overfitting issue and enhance the performance of the model, feature selection techniques can be applied. Several methods can be utilised, such as plotting and visualising feature importance, analysing correlation matrix with heatmaps, and employing univariate selection.Plotting and visualising feature importance can help identify the most influential features in predicting wine quality. This analysis can guide feature selection by prioritising the most relevant variables and discarding less significant ones.Correlation heatmaps provide insights into the relationships between different features in the dataset. By identifying highly correlated features, redundant or irrelevant variables can be eliminated to improve the model's performance.

Univariate selection involves evaluating each feature individually based on statistical measures, such as chi-square tests . This method helps identify features that have a strong relationship with the target variable and should be considered for inclusion in the model.After performing feature selection, the Random Forest model can be utilised to predict wine quality. Random Forest is particularly suitable for this dataset as it effectively handles overfitting, provides higher accuracy, recall, and precision, and captures complex relationships between features and the target variable.Throughout the project, teamwork skills were developed as team members supported each other and sought input from one another. Effective communication played a vital role in fostering strong teamwork, which was essential for completing the assignment within the set deadline.In conclusion, by carefully selecting relevant features, utilising Random Forest, and employing techniques such as feature scaling, plotting and visualising feature importance, correlation matrix with heatmaps, and univariate selection, it is possible to mitigate overfitting and achieve accurate predictions for wine quality.

10.0 CONCLUSION

In conclusion, a number of results can be summarized following the evaluation of various machine learning models for predicting wine quality, including K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest. KNN, SVM, and Arbitrary Woodland are suitable calculations for wine quality expectation. However, Random Forest emerged victorious in a comparison of their results. When the correlation matrix was used for feature selection, it had the highest accuracy of 0.9406 In this setting, Random Forest performed better because it was able to handle complex relationships and identify non-linear patterns in the data. Despite the fact that Random Forest outperformed KNN and SVM in terms of accuracy, the best model is chosen based on a variety of factors, such as the dataset and evaluation metrics used. In certain circumstances, KNN and SVM may still have advantages, and additional research and experimentation may be beneficial. In general, the developed machine learning model that makes use of KNN, SVM, and Random Forest has the potential to predict the quality of wine. Winemakers and professionals in the industry can improve quality control procedures and gain valuable insights by making use of the strengths of each algorithm.

11.0 REFERENCES

S. Yıldırım, “11 Most Common Machine Learning Algorithms Explained in a Nutshell,” *Medium*, Jul. 27, 2020. https://towardsdatascience.com/11-most-common-machine-learning-algorithms-explained-in-a-nutshell-cc6e98df93be

n.a., “Machine Learning Algorithms - Javatpoint,” *www.javatpoint.com*. https://www.javatpoint.com/machine-learning-algorithms

T. Yiu, “Understanding Random Forest,” *Medium*, Jun. 12, 2019. https://towardsdatascience.com/understanding-random-forest-58381e0602d2

N. Donges, “A Complete Guide to the Random Forest Algorithm,” *Built in*, Jul. 22, 2021. https://builtin.com/data-science/random-forest-algorithm

Support Vector Machine (SVM) Algorithm - Javatpoint. (2022). https://www.javatpoint.com/machine-learning-supportvector-machine-algorithm