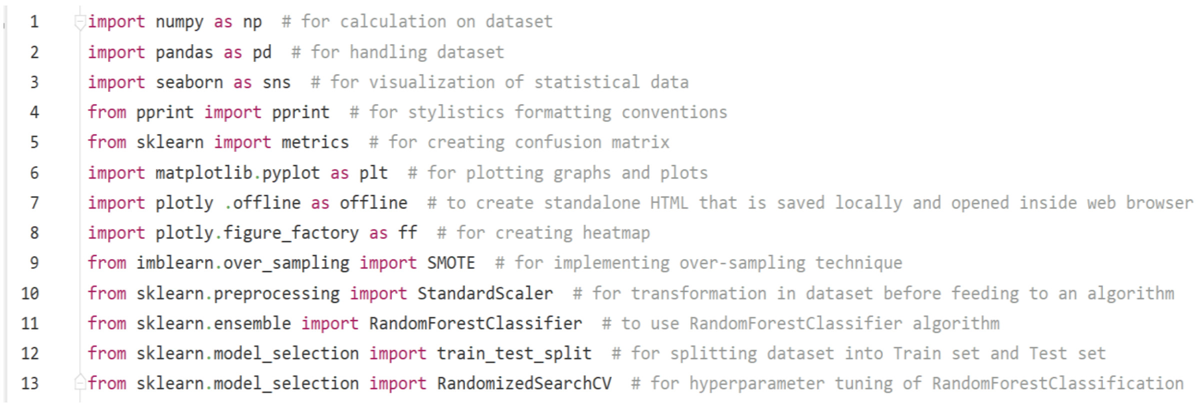
Part 1: Random Forest

1. Data Preparation:

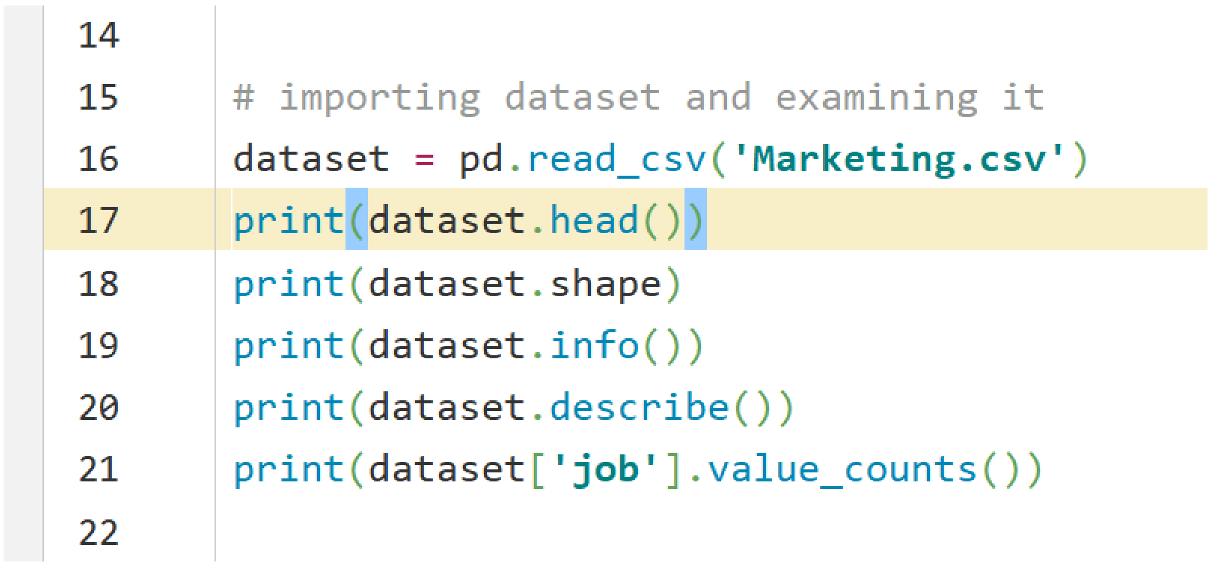
Pandas library is used for importing dataset. As Marketing.csv dataset is a csv file hence read_csv() function of pandas library is used.

- Importing all the necessary libraries and functions



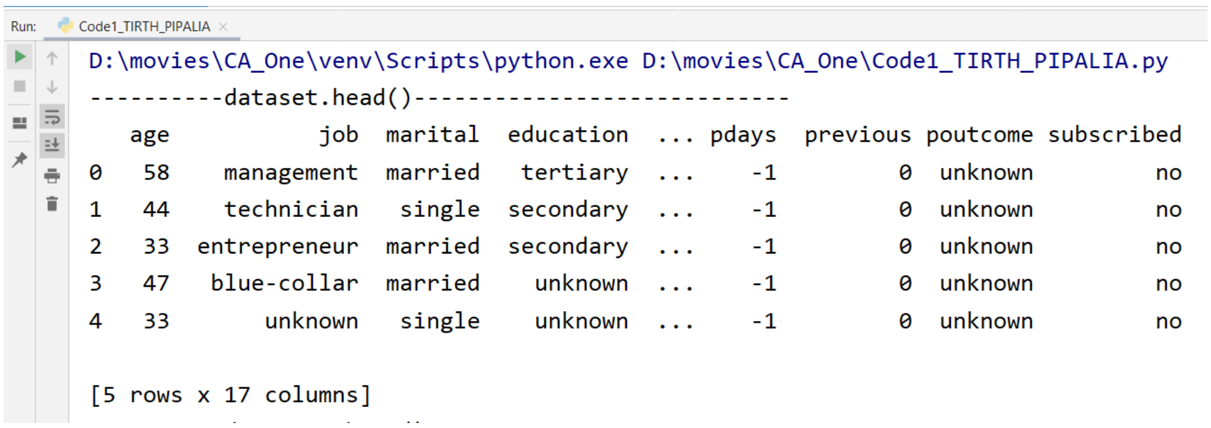
```python
import numpy as np  # for calculation on dataset
import pandas as pd  # for handling dataset
import seaborn as sns  # for visualization of statistical data
from pprint import pprint  # for stylistics formatting conventions
from sklearn import metrics  # for creating confusion matrix
import matplotlib.pyplot as plt  # for plotting graphs and plots
import plotly .offline as offline  # to create standalone HTML that is saved locally and opened inside web browser
import plotly.figure_factory as ff  # for creating heatmap
from imblearn.over_sampling import SMOTE  # for implementing over-sampling technique
from sklearn.preprocessing import StandardScaler  # for transformation in dataset before feeding to an algorithm
from sklearn.ensemble import RandomForestClassifier  # to use RandomForestClassifier algorithm
from sklearn.model_selection import train_test_split  # for splitting dataset into Train set and Test set
from sklearn.model_selection import RandomizedSearchCV  # for hyperparameter tuning of RandomForestClassification
```

- Importing dataset and examining using several pandas library functions

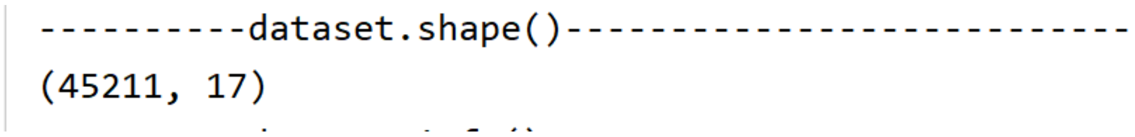


```python

# importing dataset and examining it
dataset = pd.read_csv('Marketing.csv')
print(dataset.head())
print(dataset.shape)
print(dataset.info())
print(dataset.describe())
print(dataset['job'].value_counts())

```

Output:


Run: Code1_TIRTH_PIPALIA

```
D:\movies\CA_One\venv\Scripts\python.exe D:\movies\CA_One\Code1_TIRTH_PIPALIA.py
----------dataset.head()---------------------------
   age           job  marital  education  ... pdays  previous poutcome subscribed
0   58    management  married   tertiary  ...    -1         0  unknown         no
1   44    technician   single  secondary  ...    -1         0  unknown         no
2   33  entrepreneur  married  secondary  ...    -1         0  unknown         no
3   47   blue-collar  married    unknown  ...    -1         0  unknown         no
4   33       unknown   single    unknown  ...    -1         0  unknown         no

[5 rows x 17 columns]
```



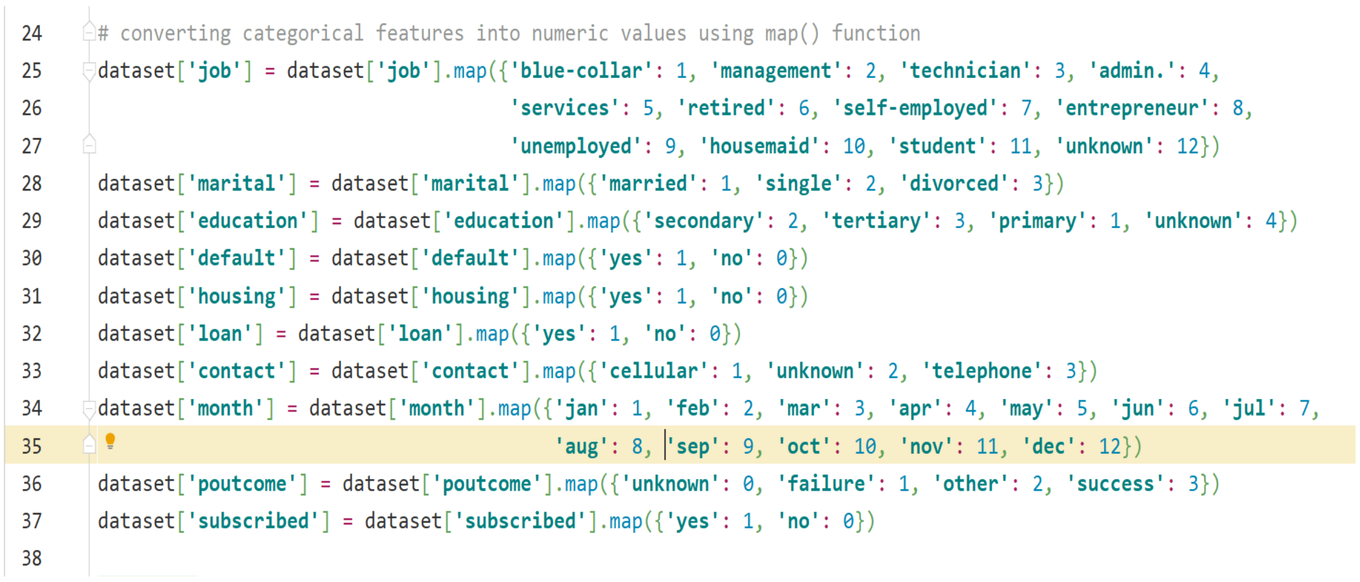
```
----------dataset.shape()---------------------------
(45211, 17)
```

```
----------dataset.info()--------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   age         45211 non-null  int64 
 1   job         45211 non-null  object
 2   marital     45211 non-null  object
 3   education   45211 non-null  object
 4   default     45211 non-null  object
 5   balance     45211 non-null  int64 
 6   housing     45211 non-null  object
 7   loan        45211 non-null  object
 8   contact     45211 non-null  object
 9   day         45211 non-null  int64 
 10  month       45211 non-null  object
 11  duration    45211 non-null  int64 
 12  campaign    45211 non-null  int64 
 13  pdays       45211 non-null  int64 
 14  previous    45211 non-null  int64 
 15  poutcome    45211 non-null  object
 16  subscribed  45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 4.1+ MB
None
```

```
----------dataset.describe()------------------------
                age        balance  ...        pdays      previous
count  45211.000000   45211.000000  ...  45211.000000  45211.000000
mean      40.936210    1362.272058  ...     40.197828      0.580323
std       10.618762    3044.765829  ...    100.128746      2.303441
min       18.000000   -8019.000000  ...     -1.000000      0.000000
25%       33.000000      72.000000  ...     -1.000000      0.000000
50%       39.000000     448.000000  ...     -1.000000      0.000000
75%       48.000000    1428.000000  ...     -1.000000      0.000000
max       95.000000  102127.000000  ...    871.000000    275.000000

[8 rows x 7 columns]
```

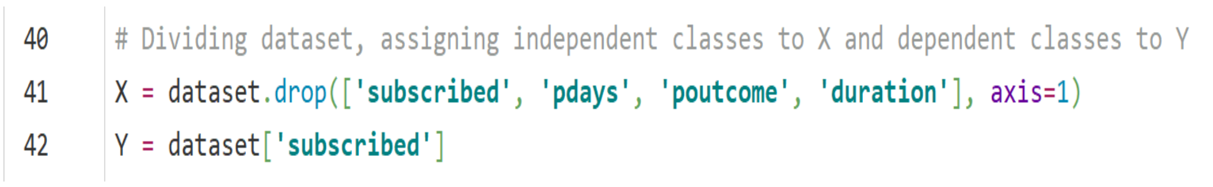
```
----------dataset['job'].value_counts()-------------
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
unknown           288
Name: job, dtype: int64
```

- Converting categorial features into numeric vales using map () function which uses dictionaries or a series. It will not create more new columns in dataset hence minimizing computation time.



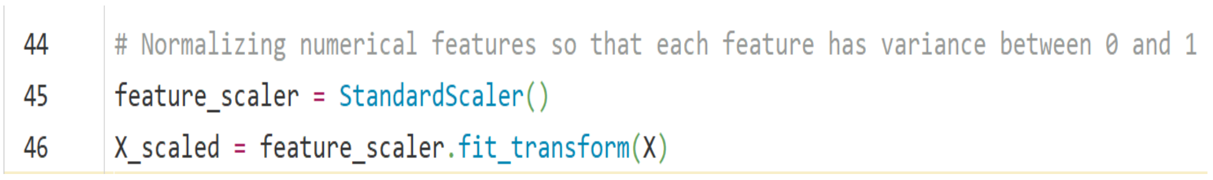
```python
# converting categorical features into numeric values using map() function
dataset['job'] = dataset['job'].map({'blue-collar': 1, 'management': 2, 'technician': 3, 'admin.': 4,
                                     'services': 5, 'retired': 6, 'self-employed': 7, 'entrepreneur': 8,
                                     'unemployed': 9, 'housemaid': 10, 'student': 11, 'unknown': 12})
dataset['marital'] = dataset['marital'].map({'married': 1, 'single': 2, 'divorced': 3})
dataset['education'] = dataset['education'].map({'secondary': 2, 'tertiary': 3, 'primary': 1, 'unknown': 4})
dataset['default'] = dataset['default'].map({'yes': 1, 'no': 0})
dataset['housing'] = dataset['housing'].map({'yes': 1, 'no': 0})
dataset['loan'] = dataset['loan'].map({'yes': 1, 'no': 0})
dataset['contact'] = dataset['contact'].map({'cellular': 1, 'unknown': 2, 'telephone': 3})
dataset['month'] = dataset['month'].map({'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4, 'may': 5, 'jun': 6, 'jul': 7,
                                         'aug': 8, 'sep': 9, 'oct': 10, 'nov': 11, 'dec': 12})
dataset['poutcome'] = dataset['poutcome'].map({'unknown': 0, 'failure': 1, 'other': 2, 'success': 3})
dataset['subscribed'] = dataset['subscribed'].map({'yes': 1, 'no': 0})
```

Dividing dataset where variable X has all the independent variables and Y has Dependent variables form main dataset (i.e. subscribed)



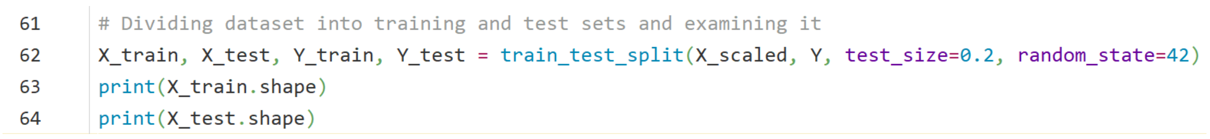
```python
# Dividing dataset, assigning independent classes to X and dependent classes to Y
X = dataset.drop(['subscribed', 'pdays', 'poutcome', 'duration'], axis=1)
Y = dataset['subscribed']
```

After dividing dataset, we need to convert mapped values into normalised form. Feature scaling is useful for precise computation using StandardScaler() function from sklearn.preprocessing package before feeding into algorithm.



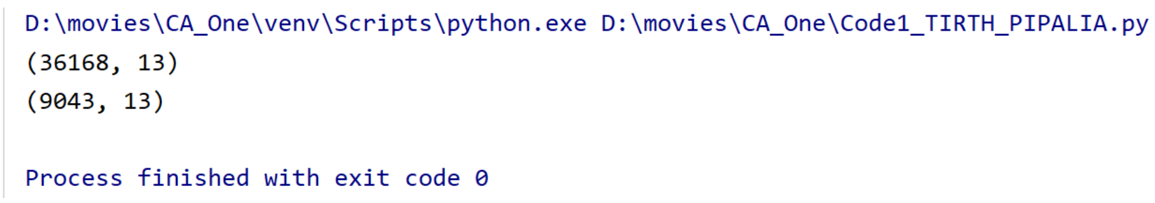
```python
# Normalizing numerical features so that each feature has variance between 0 and 1
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X)
```

Splitting the dataset into train set and test set. Here test size is 20% and 80% for train set of whole dataset. It is performed by using train_test_split() function which takes parameter as test_size where 0.2 is to have 20% of test size. random_state is seed used by random number generator in NumPy.



```python
# Dividing dataset into training and test sets and examining it
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
```

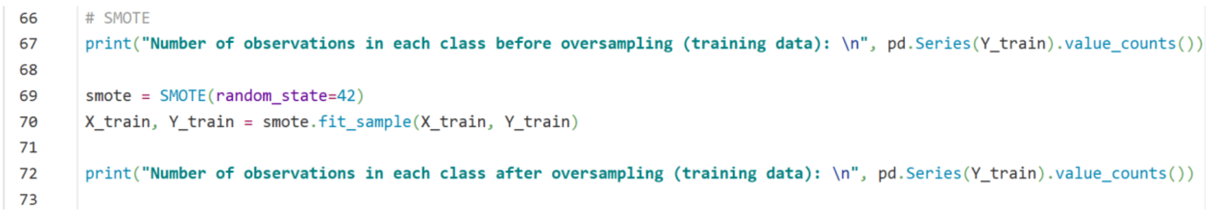
Output:



```
D:\movies\CA_One\venv\Scripts\python.exe D:\movies\CA_One\Code1_TIRTH_PIPALIA.py
(36168, 13)
(9043, 13)

Process finished with exit code 0
```

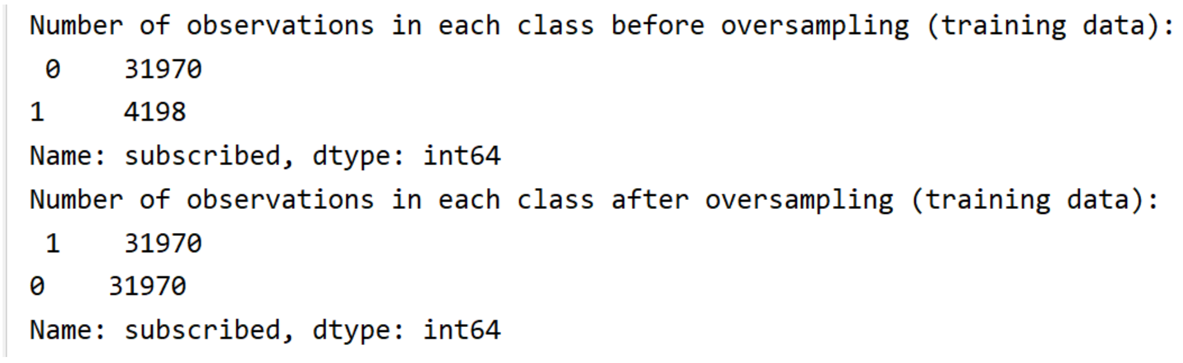
SMOTE: Synthetic Minority Over-sampling Technique

It is used to adjust the ration between the different classes/categories represented in dataset. Here it is used before cross-validation.



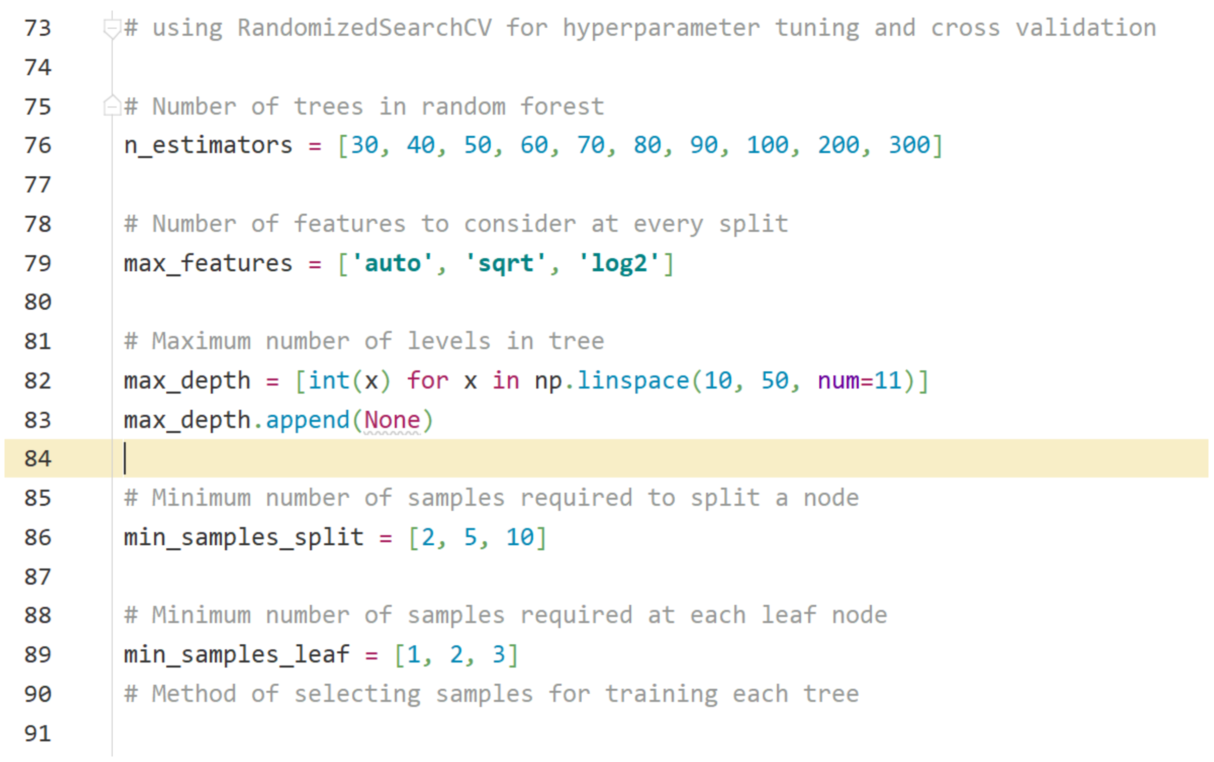
```python
# SMOTE
print("Number of observations in each class before oversampling (training data): \n", pd.Series(Y_train).value_counts())

smote = SMOTE(random_state=42)
X_train, Y_train = smote.fit_sample(X_train, Y_train)

print("Number of observations in each class after oversampling (training data): \n", pd.Series(Y_train).value_counts())

```

Output

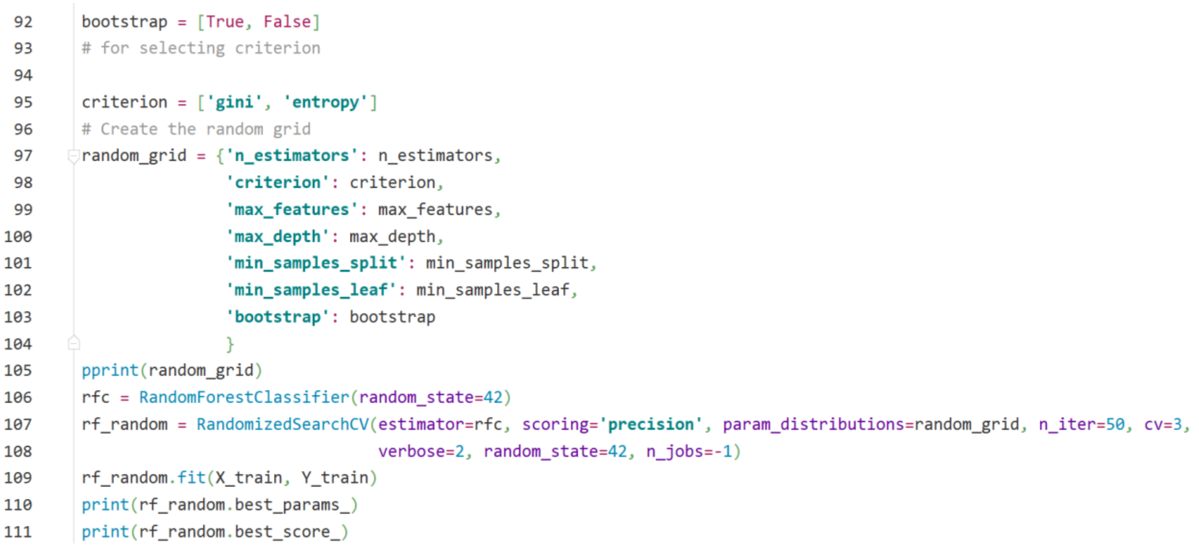


```
Number of observations in each class before oversampling (training data):
 0     31970
1      4198
Name: subscribed, dtype: int64
Number of observations in each class after oversampling (training data):
 1     31970
0    31970
Name: subscribed, dtype: int64
```

2. <u>Model Building and Testing- including hyperparameter tuning</u>

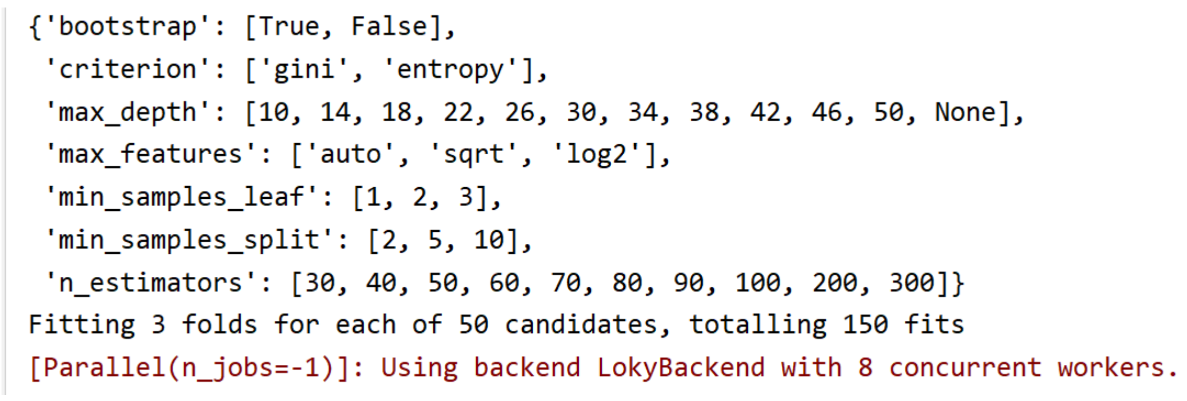


```python
# using RandomizedSearchCV for hyperparameter tuning and cross validation

# Number of trees in random forest
n_estimators = [30, 40, 50, 60, 70, 80, 90, 100, 200, 300]

# Number of features to consider at every split
max_features = ['auto', 'sqrt', 'log2']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 50, num=11)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 3]
# Method of selecting samples for training each tree

```

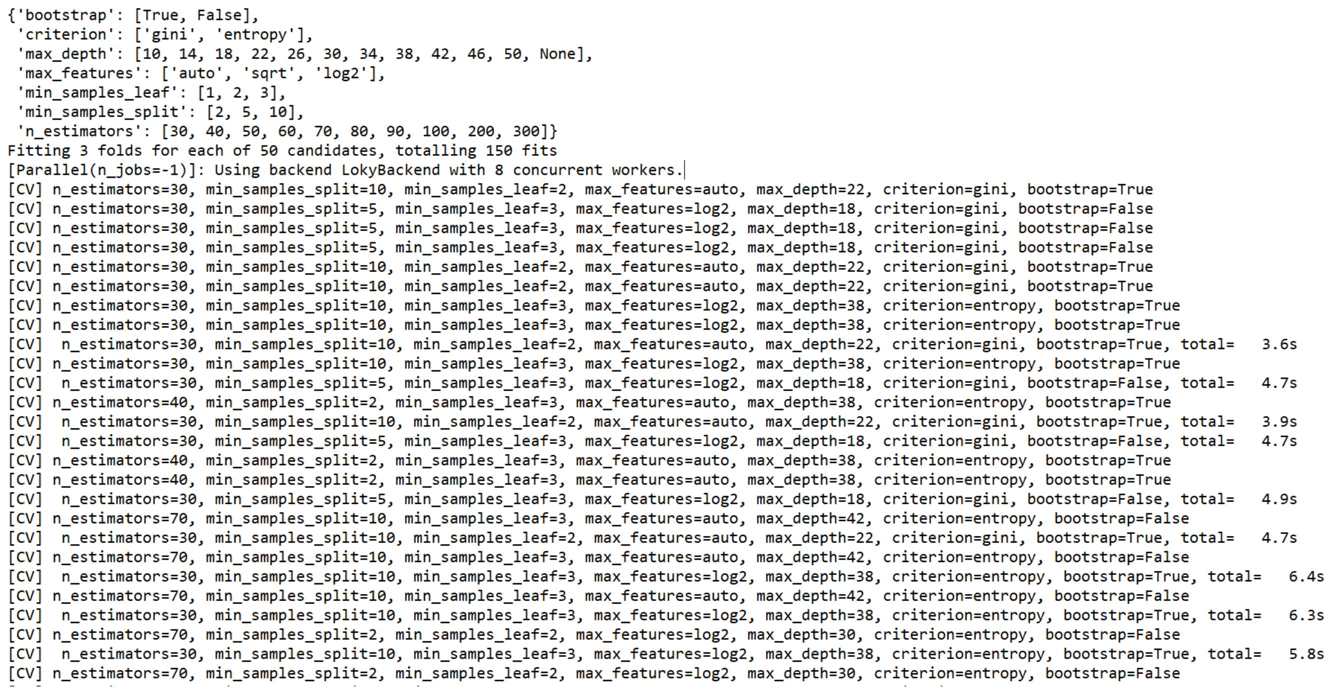


```python
bootstrap = [True, False]
# for selecting criterion

criterion = ['gini', 'entropy']
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'criterion': criterion,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap
               }
pprint(random_grid)
rfc = RandomForestClassifier(random_state=42)
rf_random = RandomizedSearchCV(estimator=rfc, scoring='precision', param_distributions=random_grid, n_iter=50, cv=3,
                               verbose=2, random_state=42, n_jobs=-1)
rf_random.fit(X_train, Y_train)
print(rf_random.best_params_)
print(rf_random.best_score_)
```

Output



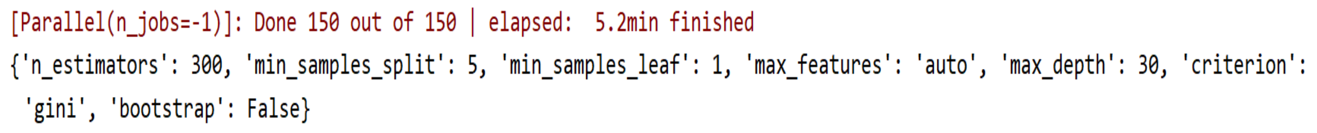
```
{'bootstrap': [True, False],
 'criterion': ['gini', 'entropy'],
 'max_depth': [10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, None],
 'max_features': ['auto', 'sqrt', 'log2'],
 'min_samples_leaf': [1, 2, 3],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [30, 40, 50, 60, 70, 80, 90, 100, 200, 300]}
Fitting 3 folds for each of 50 candidates, totalling 150 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
```



```
{'bootstrap': [True, False],
 'criterion': ['gini', 'entropy'],
 'max_depth': [10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, None],
 'max_features': ['auto', 'sqrt', 'log2'],
 'min_samples_leaf': [1, 2, 3],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [30, 40, 50, 60, 70, 80, 90, 100, 200, 300]}
Fitting 3 folds for each of 50 candidates, totalling 150 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[CV] n_estimators=30, min_samples_split=10, min_samples_leaf=2, max_features=auto, max_depth=22, criterion=gini, bootstrap=True
[CV] n_estimators=30, min_samples_split=5, min_samples_leaf=3, max_features=log2, max_depth=18, criterion=gini, bootstrap=False
[CV] n_estimators=30, min_samples_split=5, min_samples_leaf=3, max_features=log2, max_depth=18, criterion=gini, bootstrap=False
[CV] n_estimators=30, min_samples_split=5, min_samples_leaf=3, max_features=log2, max_depth=18, criterion=gini, bootstrap=False
[CV] n_estimators=30, min_samples_split=10, min_samples_leaf=2, max_features=auto, max_depth=22, criterion=gini, bootstrap=True
[CV] n_estimators=30, min_samples_split=10, min_samples_leaf=2, max_features=auto, max_depth=22, criterion=gini, bootstrap=True
[CV] n_estimators=30, min_samples_split=10, min_samples_leaf=3, max_features=log2, max_depth=38, criterion=entropy, bootstrap=True
[CV] n_estimators=30, min_samples_split=10, min_samples_leaf=3, max_features=log2, max_depth=38, criterion=entropy, bootstrap=True
[CV]  n_estimators=30, min_samples_split=10, min_samples_leaf=2, max_features=auto, max_depth=22, criterion=gini, bootstrap=True, total=   3.6s
[CV] n_estimators=30, min_samples_split=10, min_samples_leaf=3, max_features=log2, max_depth=38, criterion=entropy, bootstrap=True
[CV]  n_estimators=30, min_samples_split=5, min_samples_leaf=3, max_features=log2, max_depth=18, criterion=gini, bootstrap=False, total=   4.7s
[CV] n_estimators=40, min_samples_split=2, min_samples_leaf=3, max_features=auto, max_depth=38, criterion=entropy, bootstrap=True
[CV]  n_estimators=30, min_samples_split=10, min_samples_leaf=2, max_features=auto, max_depth=22, criterion=gini, bootstrap=True, total=   3.9s
[CV]  n_estimators=30, min_samples_split=5, min_samples_leaf=3, max_features=log2, max_depth=18, criterion=gini, bootstrap=False, total=   4.7s
[CV] n_estimators=40, min_samples_split=2, min_samples_leaf=3, max_features=auto, max_depth=38, criterion=entropy, bootstrap=True
[CV] n_estimators=40, min_samples_split=2, min_samples_leaf=3, max_features=auto, max_depth=38, criterion=entropy, bootstrap=True
[CV]  n_estimators=30, min_samples_split=5, min_samples_leaf=3, max_features=log2, max_depth=18, criterion=gini, bootstrap=False, total=   4.9s
[CV] n_estimators=70, min_samples_split=10, min_samples_leaf=3, max_features=auto, max_depth=42, criterion=entropy, bootstrap=False
[CV]  n_estimators=30, min_samples_split=10, min_samples_leaf=2, max_features=auto, max_depth=22, criterion=gini, bootstrap=True, total=   4.7s
[CV] n_estimators=70, min_samples_split=10, min_samples_leaf=3, max_features=auto, max_depth=42, criterion=entropy, bootstrap=False
[CV]  n_estimators=30, min_samples_split=10, min_samples_leaf=3, max_features=log2, max_depth=38, criterion=entropy, bootstrap=True, total=   6.4s
[CV] n_estimators=70, min_samples_split=10, min_samples_leaf=3, max_features=auto, max_depth=42, criterion=entropy, bootstrap=False
[CV]  n_estimators=30, min_samples_split=10, min_samples_leaf=3, max_features=log2, max_depth=38, criterion=entropy, bootstrap=True, total=   6.3s
[CV] n_estimators=70, min_samples_split=2, min_samples_leaf=2, max_features=log2, max_depth=30, criterion=entropy, bootstrap=False
[CV]  n_estimators=30, min_samples_split=10, min_samples_leaf=3, max_features=log2, max_depth=38, criterion=entropy, bootstrap=True, total=   5.8s
[CV] n_estimators=70, min_samples_split=2, min_samples_leaf=2, max_features=log2, max_depth=30, criterion=entropy, bootstrap=False
```

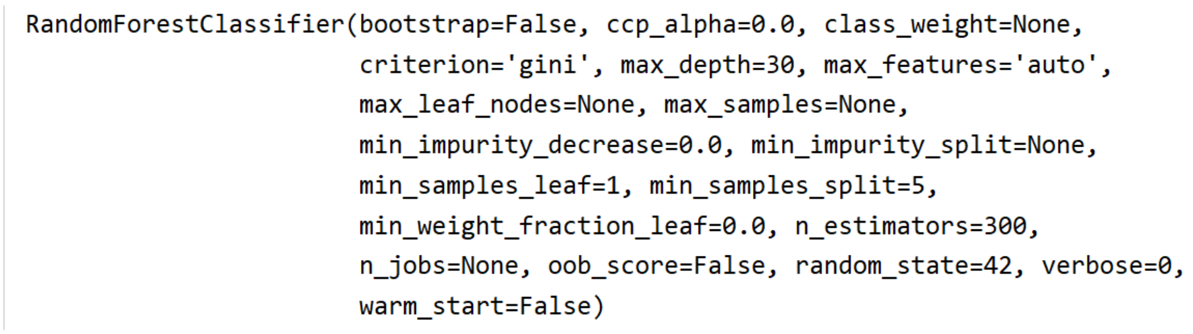
Like wise it runs every combination and shows which all combination is tried and how long it took for completion. Hence after completion it will give the best fit hyperparameters which will give the values according to you scoring value (i.e. precision/recall/accuracy/f1).

After execution using print(rf_random.best_param_) it displays list of best parameters to select.



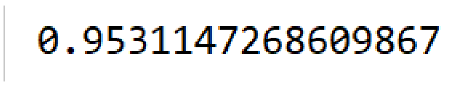
```
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:  5.2min finished
{'n_estimators': 300, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 30, 'criterion':
  'gini', 'bootstrap': False}
```

print(rf_random.best_estimator_) displays estimator which gave highest score on the left-out data.



```
RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=30, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=300,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

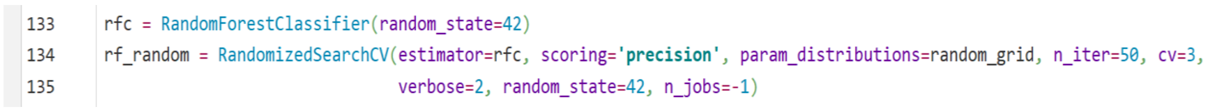
Print(rf_random.best_score_) displays mean cross-validation score of the best_estimator_



```
0.9531147268609867
```

3. Model Evaluation Strategy. Are you performing cross-validation? Are you focusing on classification accuracy or false positives or false negatives for model evaluation? You should provide logical justification behind your approach.

Yes, I have performed cross-validation within RandomizedSearchCV which has parameter as cv(cross-validation) use to assign integer value which will determine splitting strategy of cross-validation.



```
rfc = RandomForestClassifier(random_state=42)
rf_random = RandomizedSearchCV(estimator=rfc, scoring='precision', param_distributions=random_grid, n_iter=50, cv=3,
                               verbose=2, random_state=42, n_jobs=-1)
```

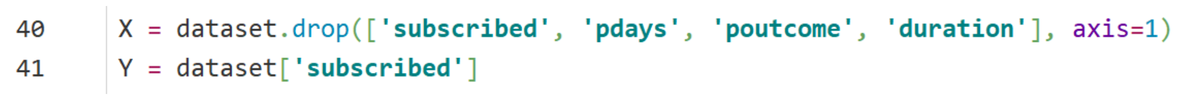
1. False Positive : This means that client is unsubscribed, and model predicts as subscribed
2. False Negative: This means that client is subscribed, and model predicts as unsubscribed

➢ Missing Information- In False Negative there are chances that single client is contacted twice and hence getting verified information whether he/she is subscribed or not
Whereas in False Positive client is left out of the reach of marketing team as it shows that client is already subscribed making a window of losing potential client. Also misleading the marketing team which could create dent in overall marketing strategy result.
So, focus should be to decrease False Positive

➢ Losing Client- In False Negative more time will be wasted as possibility is that marketing team will approach single client twice. Which depends on the company's priority whether investing time would generate loss or not. However not losing the client just getting verified with the data and prediction result.
Whilst False Positive will just create the dilemma that client is subscribed bur, he/she is not. Hence missing out the window of grabbing potential clients which always will be great loss for any marketing company. It decreases the chances of attracting a client.
Therefore, losing a customer is always a setback for company hence need to reduce false positive values.

4. Selecting Model. What is your final classification model? What are its features and parameters? How does it perform on test set?

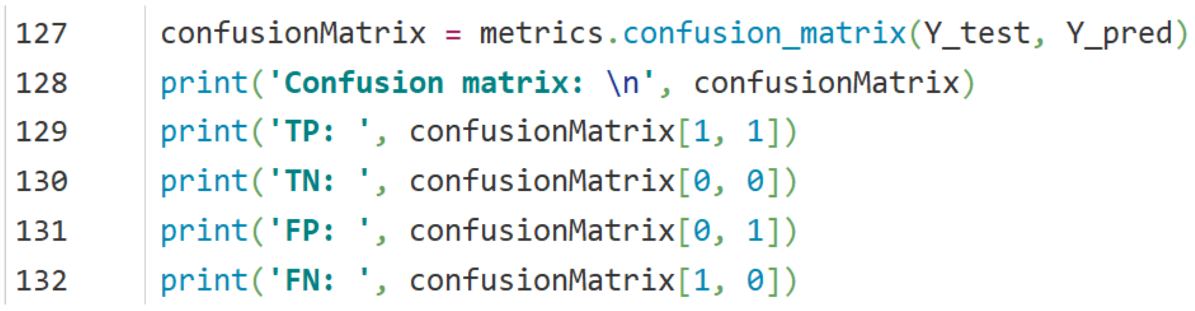
So, based on data analysis final classification model includes all the features excluding **'subscribed'**, **'pdays'**, **'poutcome'**, **'duration'** **.** for X variable(independent classes) due to above given reason.



```
X = dataset.drop(['subscribed', 'pdays', 'poutcome', 'duration'], axis=1)
Y = dataset['subscribed']
```

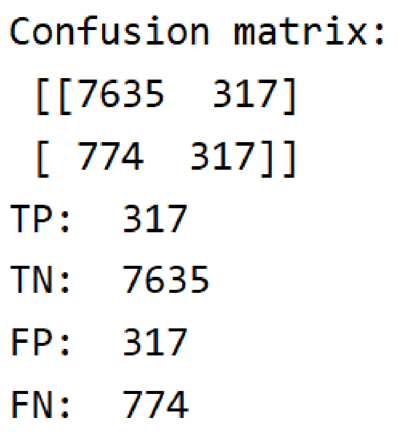
And best parameters for Random Forest Classification are:
RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
criterion='gini', max_depth=30, max_features='auto',
max_leaf_nodes=None, max_samples=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=5,
min_weight_fraction_leaf=0.0, n_estimators=300,
n_jobs=None, oob_score=False, random_state=42, verbose=0,
warm_start=False)

Confusion Matrix :

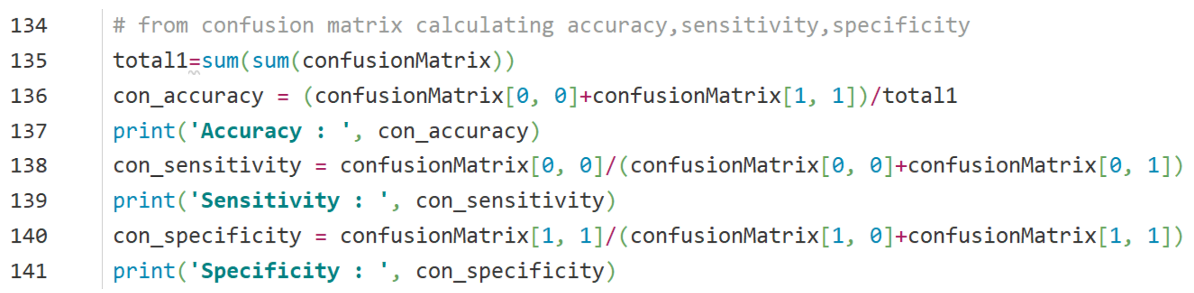


```
confusionMatrix = metrics.confusion_matrix(Y_test, Y_pred)
print('Confusion matrix: \n', confusionMatrix)
print('TP: ', confusionMatrix[1, 1])
print('TN: ', confusionMatrix[0, 0])
print('FP: ', confusionMatrix[0, 1])
print('FN: ', confusionMatrix[1, 0])
```

Output

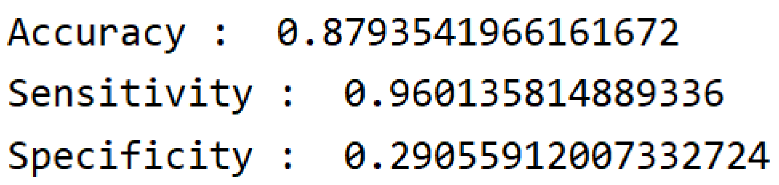


```
Confusion matrix:
 [[7635  317]
 [ 774  317]]
TP:  317
TN:  7635
FP:  317
FN:  774
```



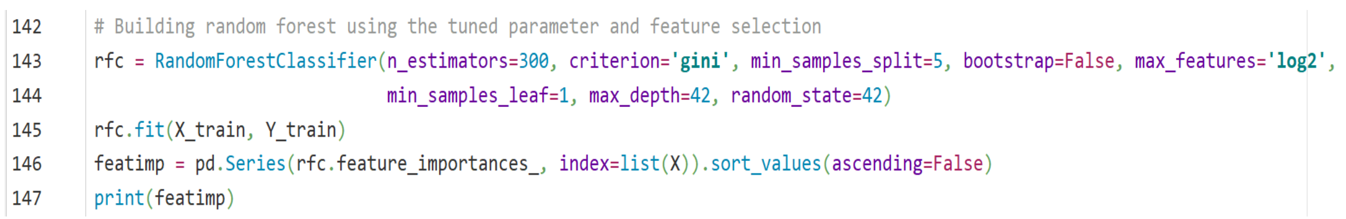
```python
# from confusion matrix calculating accuracy,sensitivity,specificity
total1=sum(sum(confusionMatrix))
con_accuracy = (confusionMatrix[0, 0]+confusionMatrix[1, 1])/total1
print('Accuracy : ', con_accuracy)
con_sensitivity = confusionMatrix[0, 0]/(confusionMatrix[0, 0]+confusionMatrix[0, 1])
print('Sensitivity : ', con_sensitivity)
con_specificity = confusionMatrix[1, 1]/(confusionMatrix[1, 0]+confusionMatrix[1, 1])
print('Specificity : ', con_specificity)
```

Output

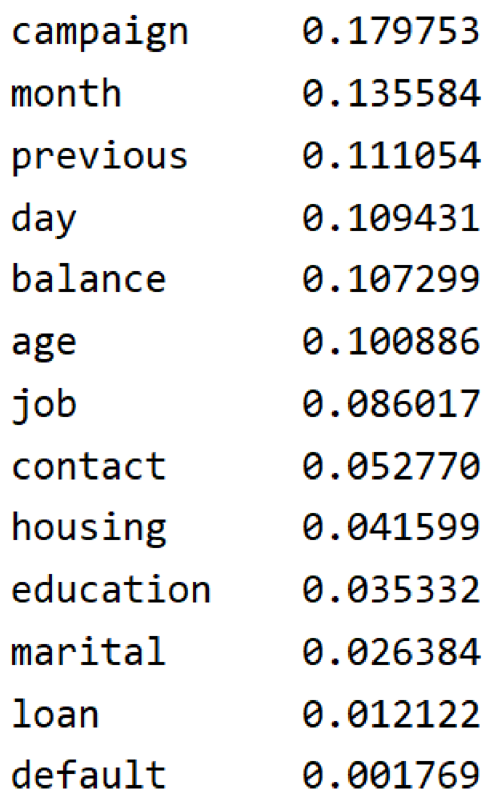


```
Accuracy :  0.8793541966161672
Sensitivity :  0.960135814889336
Specificity :  0.29055912007332724
```

5. **Generating Recommendations. Based on information about predictive power of various features used in your final model (using feature_importances), what recommendations can you make to bank's marketing department for correctly identifying a potential term deposit subscriber?**



```python
# Building random forest using the tuned parameter and feature selection
rfc = RandomForestClassifier(n_estimators=300, criterion='gini', min_samples_split=5, bootstrap=False, max_features='log2',
                             min_samples_leaf=1, max_depth=42, random_state=42)
rfc.fit(X_train, Y_train)
featimp = pd.Series(rfc.feature_importances_, index=list(X)).sort_values(ascending=False)
print(featimp)
```



```
campaign     0.179753
month        0.135584
previous     0.111054
day          0.109431
balance      0.107299
age          0.100886
job          0.086017
contact      0.052770
housing      0.041599
education    0.035332
marital      0.026384
loan         0.012122
default      0.001769
```

Group1- Education, Marital, Loan, Default

Group2- Housing, Contact, Job, Age, Day

Group3 -Previous, Month, campaign

Group4- pdays, poutcome, duration

So, for generating any kind of recommendations based on information about features used in final model using feature_importances all the classes in Group have most high significant value and hence any change in their value create major effect on overall result during prediction. For more accuracy we need to include maximum variables from Group 1.

Classes in Group 2 when, implemented along with Group 1 classes gives more accurate prediction and more information is added to machine learning model so it provides optimum result.

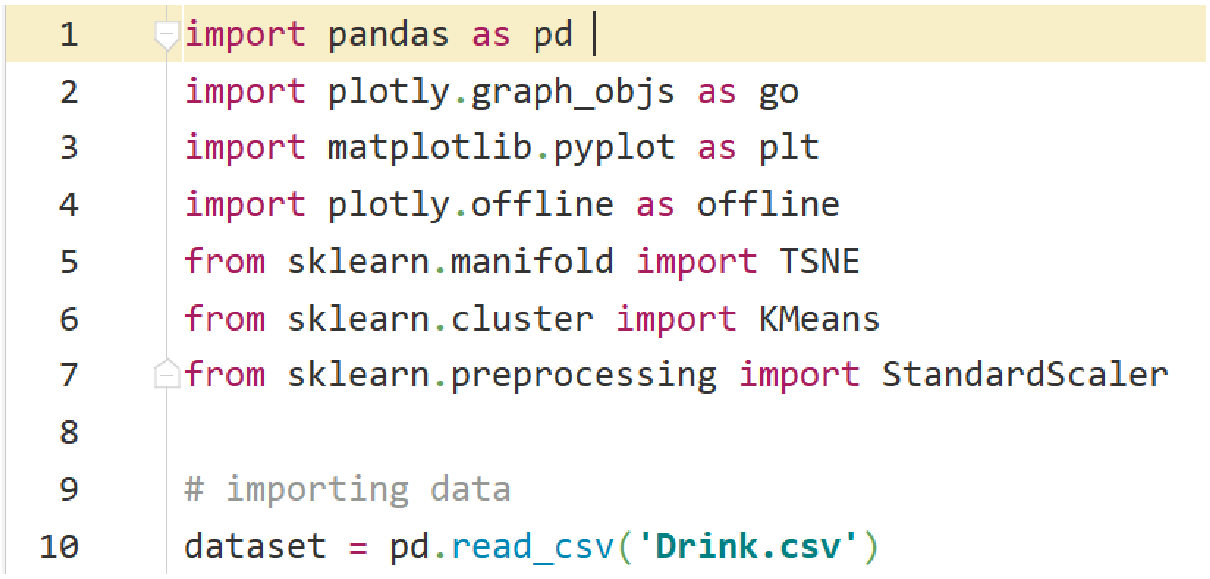
Group3 classes includes variables with least importance hence making minute distortion in output prediction.

Group4 has the variables which should be dropped as it provides corelated information or else something which is not that important for prediction. We should try not to feed all the variables in model or else it will overfit he models and we should not even remove all the variables which won’t give sufficient information to model to get trained.

<u>Part 2: t-SNE & K-Means Clustering</u>

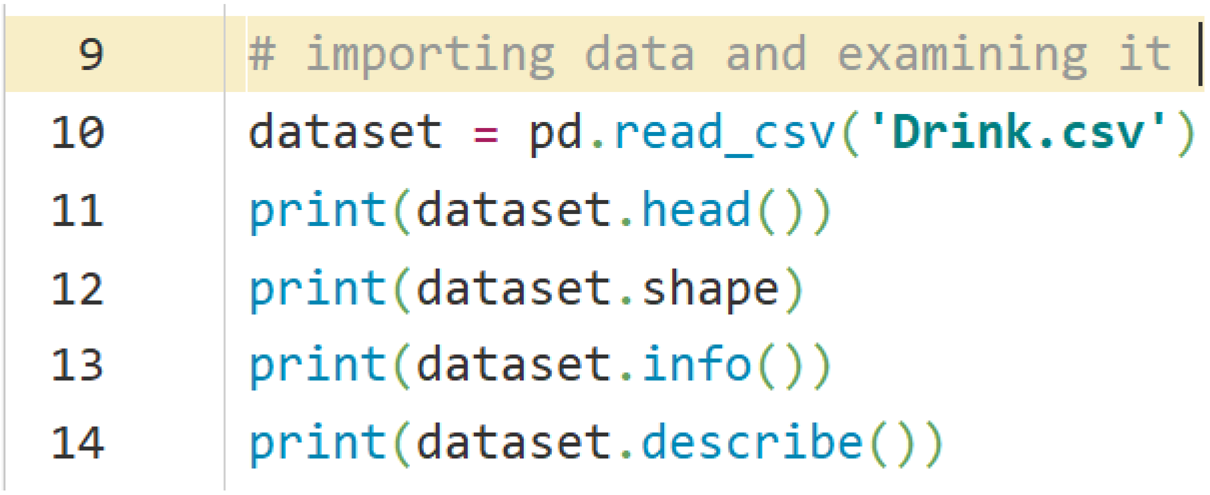
1. <u>**Data Preparation**</u>

- Importing several libraries which is useful for implementing t-SNE using k-means for this task



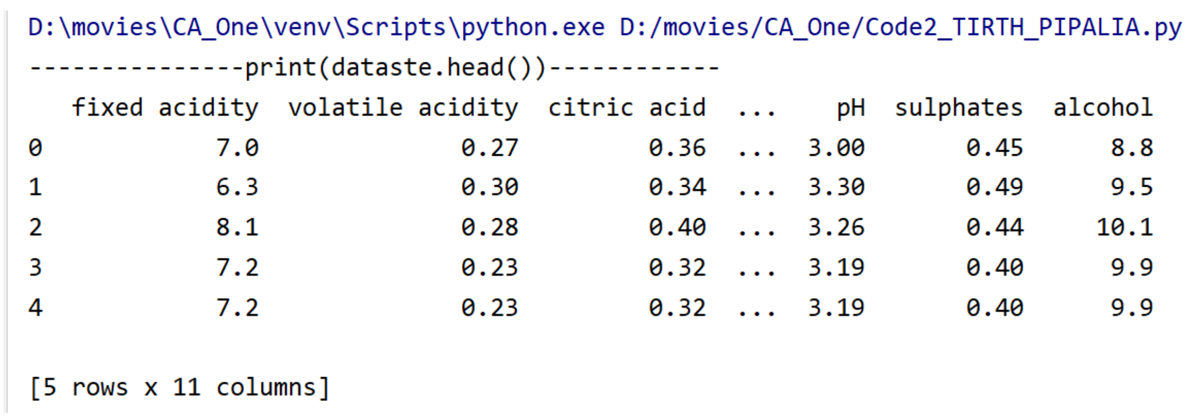
```python
import pandas as pd
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import plotly.offline as offline
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# importing data
dataset = pd.read_csv('Drink.csv')
```

- Importing dataset and examining it using pandas library functions

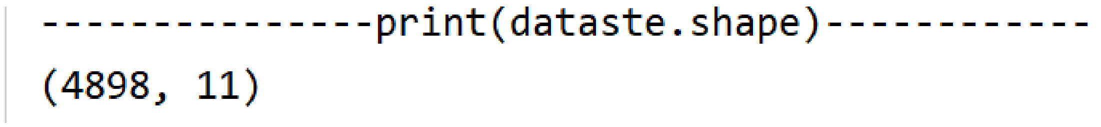


```python
# importing data and examining it
dataset = pd.read_csv('Drink.csv')
print(dataset.head())
print(dataset.shape)
print(dataset.info())
print(dataset.describe())
```

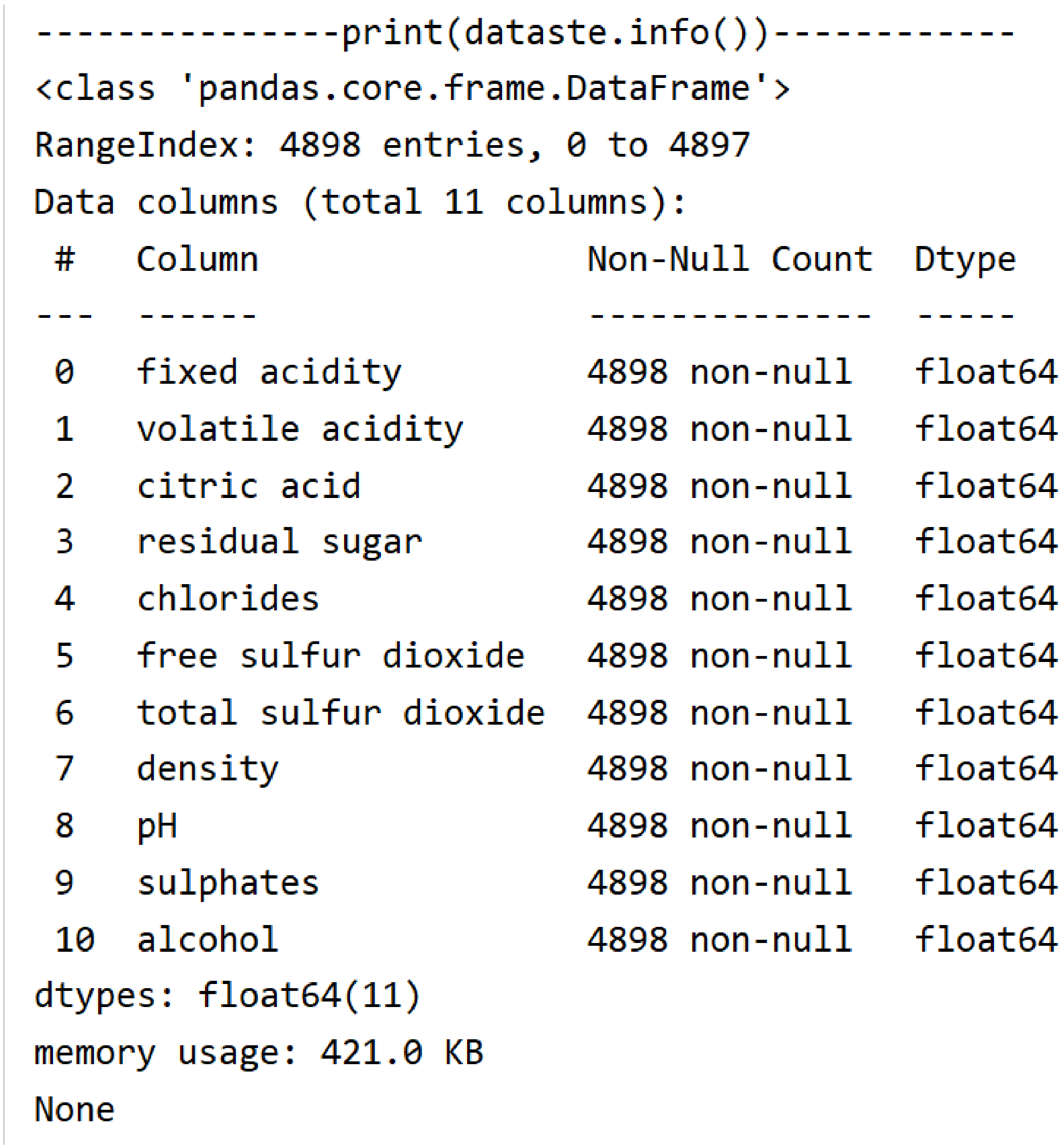
Output



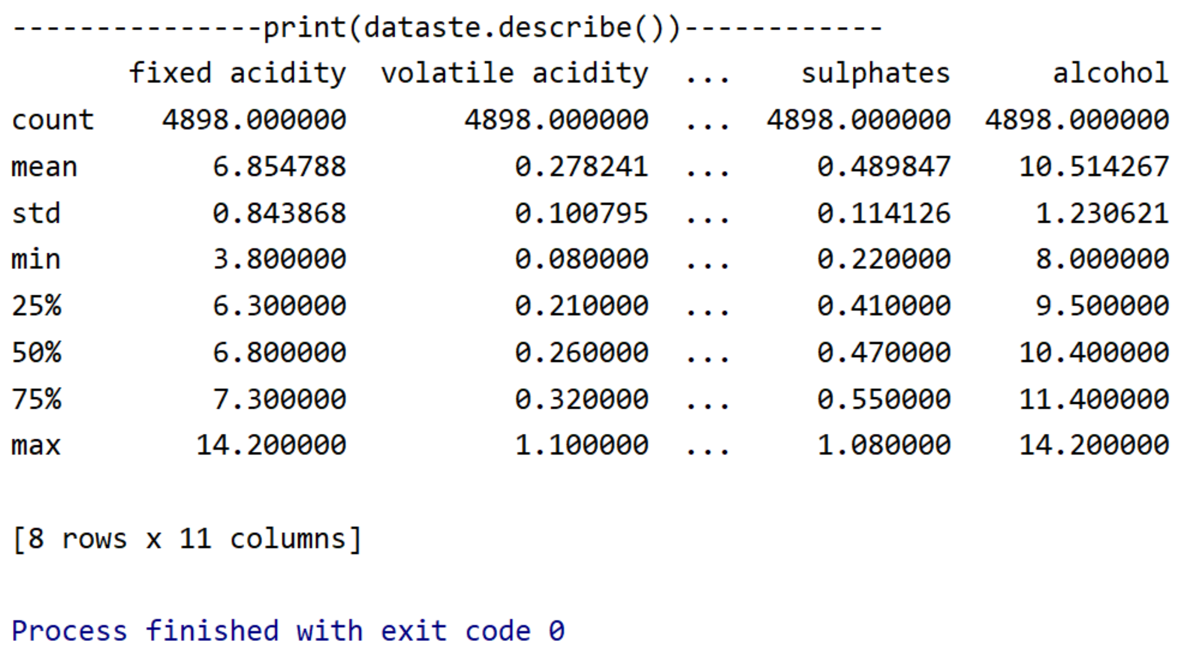
```
D:\movies\CA_One\venv\Scripts\python.exe D:/movies/CA_One/Code2_TIRTH_PIPALIA.py
---------------print(dataste.head())------------
   fixed acidity  volatile acidity  citric acid  ...    pH  sulphates  alcohol
0            7.0              0.27         0.36  ...  3.00       0.45      8.8
1            6.3              0.30         0.34  ...  3.30       0.49      9.5
2            8.1              0.28         0.40  ...  3.26       0.44     10.1
3            7.2              0.23         0.32  ...  3.19       0.40      9.9
4            7.2              0.23         0.32  ...  3.19       0.40      9.9

[5 rows x 11 columns]
```



```
---------------print(dataste.shape)------------
(4898, 11)
```

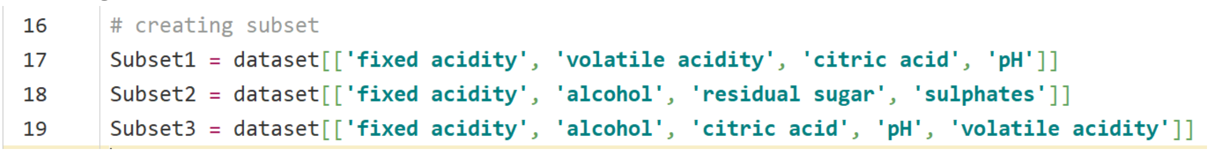


```
---------------print(dataste.info())------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
dtypes: float64(11)
memory usage: 421.0 KB
None
```



```
---------------print(dataste.describe())------------
       fixed acidity  volatile acidity  ...   sulphates     alcohol
count    4898.000000       4898.000000  ...  4898.000000  4898.000000
mean        6.854788          0.278241  ...     0.489847    10.514267
std         0.843868          0.100795  ...     0.114126     1.230621
min         3.800000          0.080000  ...     0.220000     8.000000
25%         6.300000          0.210000  ...     0.410000     9.500000
50%         6.800000          0.260000  ...     0.470000    10.400000
75%         7.300000          0.320000  ...     0.550000    11.400000
max        14.200000          1.100000  ...     1.080000    14.200000

[8 rows x 11 columns]

Process finished with exit code 0
```

- Creating Subsets

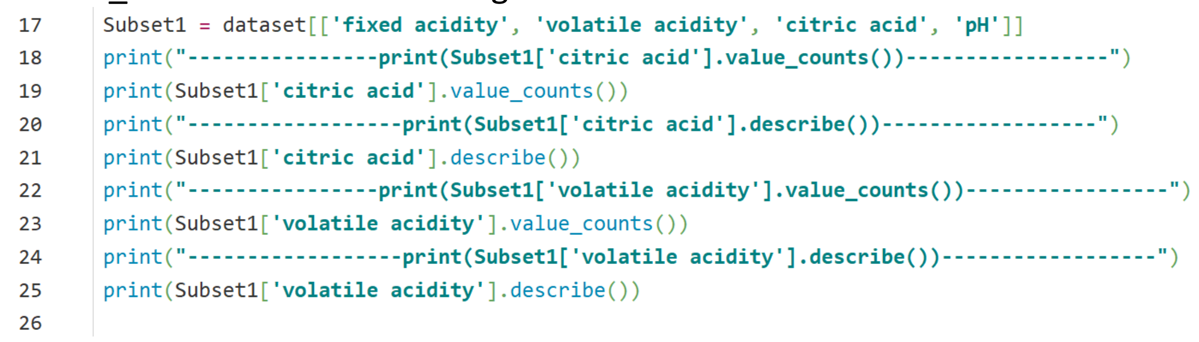


```
# creating subset
Subset1 = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'pH']]
Subset2 = dataset[['fixed acidity', 'alcohol', 'residual sugar', 'sulphates']]
Subset3 = dataset[['fixed acidity', 'alcohol', 'citric acid', 'pH', 'volatile acidity']]
```

As data of all the classes in subset is continuous which will not provide accurate clusters hence converting few classes into discrete values will help to generate better clustering using t-SNE and k-means

For subset1 which has **'fixed acidity'**, **'volatile acidity'**, **'citric acid'**, **'pH'** classes out of which I have decided to convert citric acid and volatile acid into discrete. Instead of taking any random value and then converting it into discrete I choose to take mean value of classes so that it is converted in balanced form.

Values_counts before converting and mean value



```
Subset1 = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'pH']]
print("----------------print(Subset1['citric acid'].value_counts())-----------------")
print(Subset1['citric acid'].value_counts())
print("------------------print(Subset1['citric acid'].describe())------------------")
print(Subset1['citric acid'].describe())
print("----------------print(Subset1['volatile acidity'].value_counts())-----------------")
print(Subset1['volatile acidity'].value_counts())
print("------------------print(Subset1['volatile acidity'].describe())------------------")
print(Subset1['volatile acidity'].describe())

```

Output

```
----------------print(Subset1['citric acid'].value_counts())-----------------
0.30    307
0.28    282
0.32    257
0.34    225
0.29    223
       ...
1.66      1
0.11      1
0.86      1
0.99      1
1.23      1
Name: citric acid, Length: 87, dtype: int64
Name: citric acid, Length: 87, dtype: int64
------------------print(Subset1['citric acid'].describe())------------------
count    4898.000000
mean        0.334192
std         0.121020
min         0.000000
25%         0.270000
50%         0.320000
75%         0.390000
max         1.660000
Name: citric acid, dtype: float64
----------------print(Subset1['volatile acidity'].value_counts())-----------------
0.280    263
0.240    253
0.260    240
0.250    231
0.220    229
        ...
0.355      1
0.215      1
0.740      1
0.090      1
0.405      1
Name: volatile acidity, Length: 125, dtype: int64
```

```
Name: volatile acidity, Length: 125, dtype: int64
------------------print(Subset1['volatile acidity'].describe())------------------
count    4898.000000
mean        0.278241
std         0.100795
min         0.080000
25%         0.210000
50%         0.260000
75%         0.320000
max         1.100000
Name: volatile acidity, dtype: float64
```

```python
Subset2 = dataset[['fixed acidity', 'alcohol', 'residual sugar', 'sulphates']]
print("----------------print(Subset2['residual sugar'].value_counts())----------------")
print(Subset2['residual sugar'].value_counts())
print("------------------print(Subset2['residual sugar'].describe())------------------")
print(Subset2['residual sugar'].describe())
print("----------------print(Subset2['fixed acidity'].value_counts())----------------")
print(Subset2['fixed acidity'].value_counts())
print("------------------print(Subset2['fixed acidity'].describe())------------------")
print(Subset2['fixed acidity'].describe())

```

Output

```
----------------print(Subset2['residual sugar'].value_counts())----------------
1.20     187
1.40     184
1.60     165
1.30     147
1.10     146
        ...
12.75      1
6.55       1
8.55       1
5.55       1
7.85       1
Name: residual sugar, Length: 310, dtype: int64
```

```
-----------------print(Subset2['residual sugar'].describe())-----------------
count    4898.000000
mean        6.391415
std         5.072058
min         0.600000
25%         1.700000
50%         5.200000
75%         9.900000
max        65.800000
Name: residual sugar, dtype: float64
```

```
---------------print(Subset2['fixed acidity'].value_counts())----------------
6.80     308
6.60     290
6.40     280
6.90     241
6.70     236
        ...
6.45       1
3.80       1
14.20      1
10.20      1
3.90       1
Name: fixed acidity, Length: 68, dtype: int64
```

```
-----------------print(Subset2['fixed acidity'].describe())-----------------
count    4898.000000
mean        6.854788
std         0.843868
min         3.800000
25%         6.300000
50%         6.800000
75%         7.300000
max        14.200000
Name: fixed acidity, dtype: float64
```

```python
Subset3 = dataset[['fixed acidity', 'alcohol', 'citric acid', 'pH', 'volatile acidity']]
print("---------------print(Subset3['volatile acidity'].value_counts())----------------")
print(Subset3['volatile acidity'].value_counts())
print("-----------------print(Subset3['volatile acidity'].describe())-----------------")
print(Subset3['volatile acidity'].describe())
print("---------------print(Subset3['alcohol'].value_counts())----------------")
print(Subset3['alcohol'].value_counts())
print("-----------------print(Subset3['alcohol'].describe())-----------------")
print(Subset3['alcohol'].describe())
```

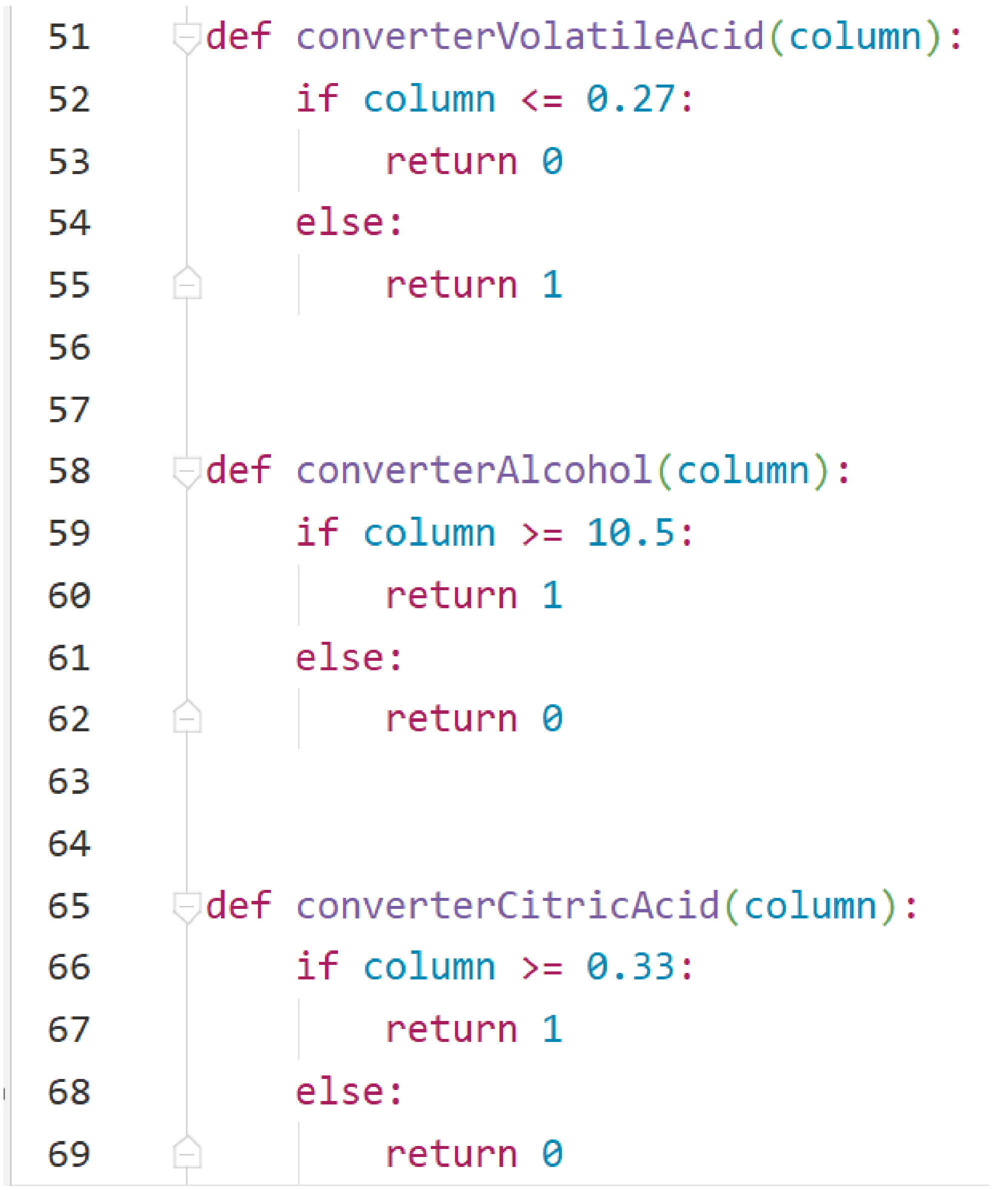
Output

```
----------------print(Subset3['volatile acidity'].value_counts())-----------------
0.280    263
0.240    253
0.260    240
0.250    231
0.220    229
        ...
0.355      1
0.215      1
0.740      1
0.090      1
0.405      1
Name: volatile acidity, Length: 125, dtype: int64

-----------------print(Subset3['volatile acidity'].describe())-----------------
count    4898.000000
mean        0.278241
std         0.100795
min         0.080000
25%         0.210000
50%         0.260000
75%         0.320000
max         1.100000
Name: volatile acidity, dtype: float64
```

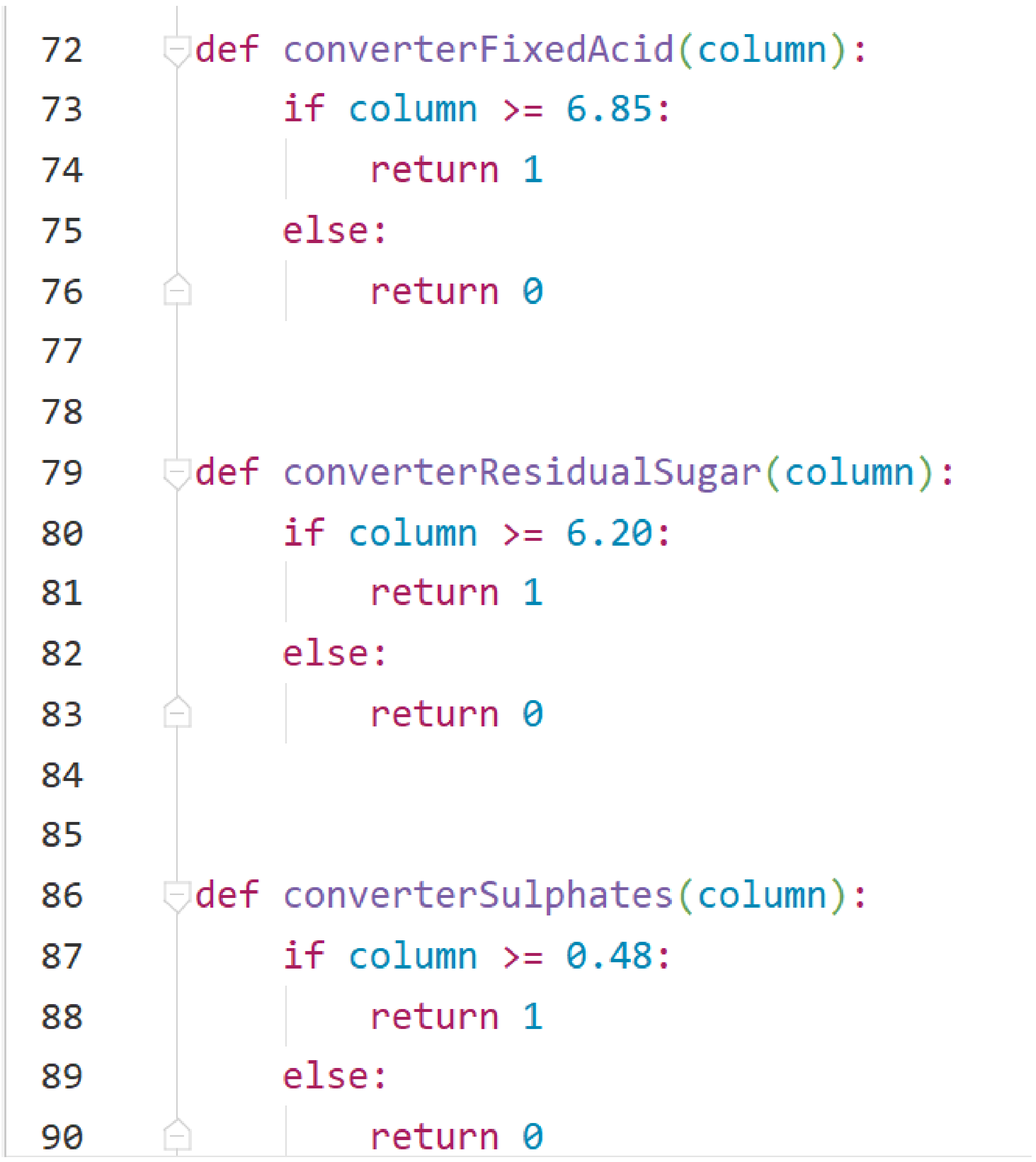
```
----------------print(Subset3['alcohol'].value_counts())-----------------
9.400000     229
9.500000     228
9.200000     199
9.000000     185
10.000000    162
            ...
11.850000      1
14.050000      1
12.066667      1
12.250000      1
11.266667      1
Name: alcohol, Length: 103, dtype: int64
```

```
------------------print(Subset3['alcohol'].describe())------------------
count    4898.000000
mean       10.514267
std         1.230621
min         8.000000
25%         9.500000
50%        10.400000
75%        11.400000
max        14.200000
Name: alcohol, dtype: float64
```

Based on this output using functions I have converted classes in discrete using mean value as splitting value



```python
def converterVolatileAcid(column):
    if column <= 0.27:
        return 0
    else:
        return 1


def converterAlcohol(column):
    if column >= 10.5:
        return 1
    else:
        return 0


def converterCitricAcid(column):
    if column >= 0.33:
        return 1
    else:
        return 0
```



```python
def converterFixedAcid(column):
    if column >= 6.85:
        return 1
    else:
        return 0


def converterResidualSugar(column):
    if column >= 6.20:
        return 1
    else:
        return 0


def converterSulphates(column):
    if column >= 0.48:
        return 1
    else:
        return 0
```

Hence applying this functions to convert specific class into discrete

```python
Subset1['citric acid'] = Subset1['citric acid'].apply(converterCitricAcid)
print(Subset1['citric acid'].value_counts())

Subset1['volatile acidity'] = Subset1['volatile acidity'].apply(converterVolatileAcid)
print(Subset1['volatile acidity'].value_counts())

Subset2['residual sugar'] = Subset2['residual sugar'].apply(converterResidualSugar)
print(Subset2['residual sugar'].value_counts())

Subset2['fixed acidity'] = Subset2['fixed acidity'].apply(converterFixedAcid)
print(Subset2['fixed acidity'].value_counts())

Subset3['alcohol'] = Subset3['alcohol'].apply(converterAlcohol)
print(Subset3['alcohol'].value_counts())

Subset3['volatile acidity'] = Subset3['volatile acidity'].apply(converterVolatileAcid)
print(Subset3['volatile acidity'].value_counts())

```

Output:

```
0    2687
1    2211
Name: citric acid, dtype: int64

0    2729
1    2169
Name: volatile acidity, dtype: int64

0    2666
1    2232
Name: residual sugar, dtype: int64

0    2635
1    2263
Name: fixed acidity, dtype: int64

0    2575
1    2323
Name: alcohol, dtype: int64

0    2729
1    2169
Name: volatile acidity, dtype: int64
```

- Feature scaling the dataset for optimal output

```python
# Normalizing numerical features so that each feature has mean 0 and variance 1
feature_scaler = StandardScaler()
scaledSubset1 = feature_scaler.fit_transform(Subset1)
scaledSubset2 = feature_scaler.fit_transform(Subset2)
scaledSubset3 = feature_scaler.fit_transform(Subset3)
```

2. t-SNE Implementation. Why are you using t-SNE? Are you implementing it on the entire dataset or its subsets?

t- distributed Stochastic Neighbour Embedding is a machine learning algorithm. It is non-linear dimensionality reduction technique well suited for embedding high-dimension data for visualization in a low-dimension space of two or three dimensions. Like several other unsupervised learning algorithm t-SNE often provides early insights on whether the date is separable or not. Because in unsupervised learning we don't know the target variable and there can be several unknown classes about which information is or knowledge is not available is minimum. So, for drinks dataset as it has 11 dimensions (i.e. 11 classes) having high dimensions hence t-SE is used for visualization. However, applying t-SNE to whole dataset was not a best choice as more the dimension inaccurate the cluster formation. Hence, I divided main dataset into 3 different subsets which are

Subset1 – Fixed Acidity, Volatile Acidity, Citric Acid, pH

```
Subset1 = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'pH']]
```

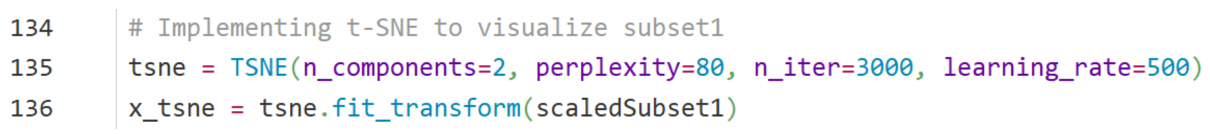
Subset2 – Fixed Acidity, Alcohol, Residual Sugar, Sulphates

```
Subset2 = dataset[['fixed acidity', 'alcohol', 'residual sugar', 'sulphates']]
```

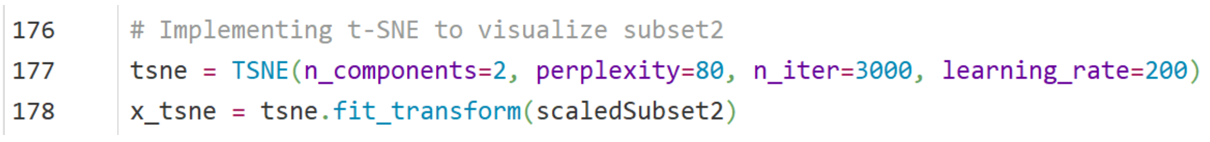
Subset3 – Fixed Acidity, Alcohol, Citric Acid, pH, Volatile Acid

```
Subset3 = dataset[['fixed acidity', 'alcohol', 'citric acid', 'pH', 'volatile acidity']]
```

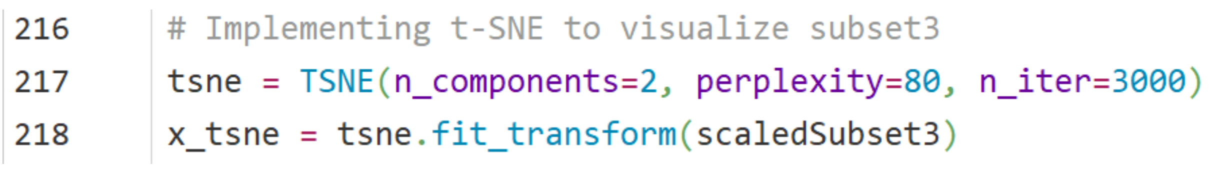
- t-SNE implementation on subsets



```
# Implementing t-SNE to visualize subset1
tsne = TSNE(n_components=2, perplexity=80, n_iter=3000, learning_rate=500)
x_tsne = tsne.fit_transform(scaledSubset1)
```



```
# Implementing t-SNE to visualize subset2
tsne = TSNE(n_components=2, perplexity=80, n_iter=3000, learning_rate=200)
x_tsne = tsne.fit_transform(scaledSubset2)
```



```
# Implementing t-SNE to visualize subset3
tsne = TSNE(n_components=2, perplexity=80, n_iter=3000)
x_tsne = tsne.fit_transform(scaledSubset3)
```

3. <u>K-Means Implementation. What does the elbow plot tell you about the number of clusters? What exactly is k-means being used for? What role is it playing in the t-SNE visualizations?</u>

k-means is a simple unsupervised machine learning algorithm that groups a dataset into a user specifies number (k) of clusters. One method to validate the number for clusters is the elbow method.

The idea of elbow plot is to run k-means clustering on the dataset for a range of values of k(say value of k from 1 to 10) and for each value of k, calculate the sum of squared errors(SSE). Then plotting line chart of the SSE for each value of k. So, as k increases, average distortion will decrease and hence each cluster will be closer to their respective centroids. The value of k at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters.

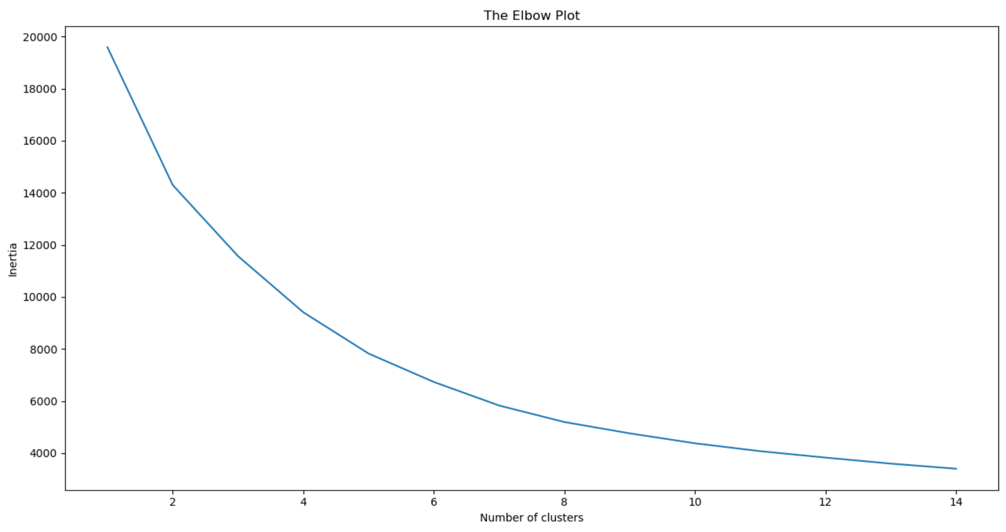
k-means along with elbow plot helps in getting labels which will be main factor for knowing the insights of data when visualizing or else it will be time consuming task to identify the clusters in unsupervised dataset. It helps in displaying visually separable clusters more feasible than just implementing t-SNE.

- Implementing k-means and elbow plot

Subset1:

```python
# Finding the number of clusters (K) - using Elbow Plot Method for Subset1
inertia = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters=i, random_state=100)
    kmeans.fit(scaledSubset1)
    inertia.append(kmeans.inertia_)
plt.plot(range(1, 15), inertia)
plt.title('The Elbow Plot')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

# Running KMeans to generate labels for subset1
kmeans = KMeans(n_clusters=4)
kmeans.fit(scaledSubset1)
```

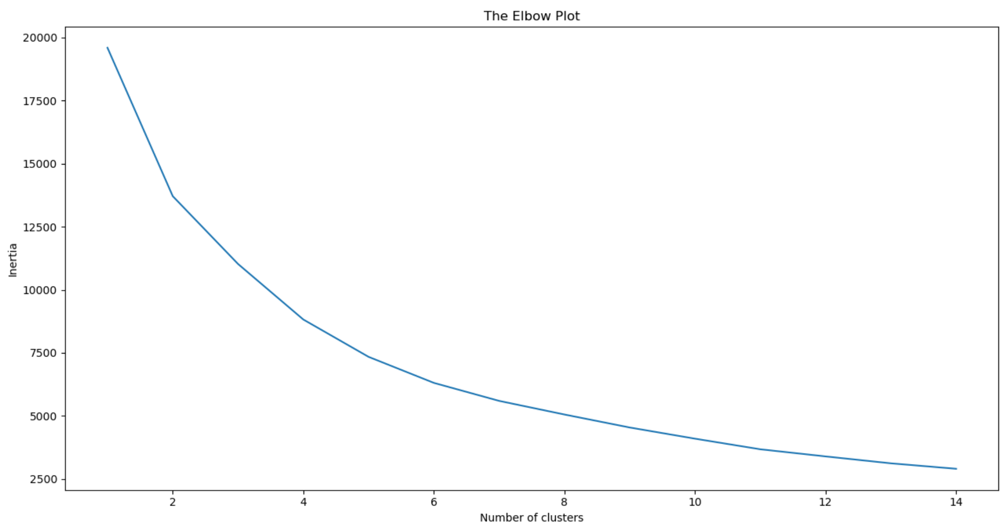
Elbow plot:



Subset2:

```python
# Finding the number of clusters (K) - using Elbow Plot Method for Subset2
inertia = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters=i, random_state=100)
    kmeans.fit(scaledSubset2)
    inertia.append(kmeans.inertia_)
plt.plot(range(1, 15), inertia)
plt.title('The Elbow Plot')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

# Running KMeans to generate labels for subset2
kmeans = KMeans(n_clusters=4)
kmeans.fit(scaledSubset2)
```

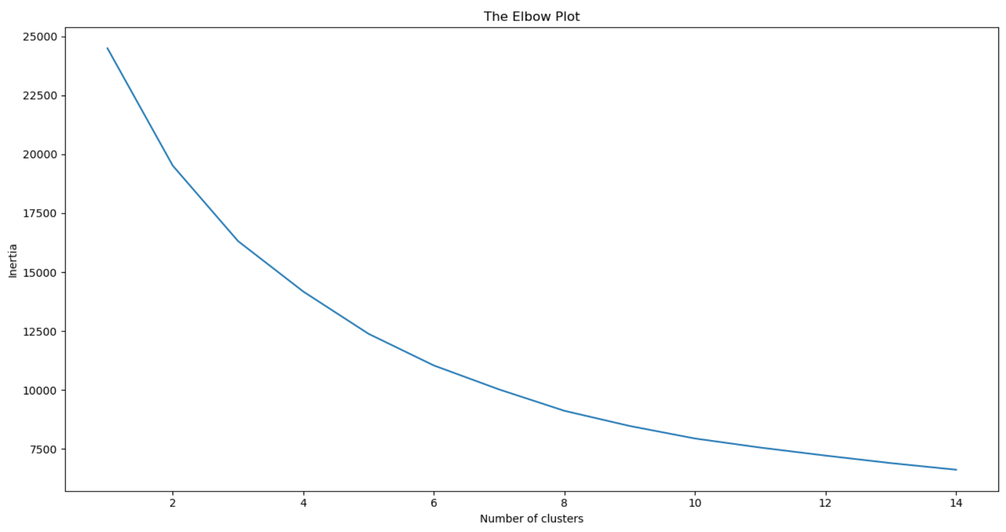
Elbow plot:



Subset3:

```python
# Finding the number of clusters (K) - using Elbow Plot Method for Subset3
inertia = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters=i, random_state=100)
    kmeans.fit(scaledSubset3)
    inertia.append(kmeans.inertia_)
plt.plot(range(1, 15), inertia)
plt.title('The Elbow Plot')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

# Running KMeans to generate labels for subset3
kmeans = KMeans(n_clusters=3)
kmeans.fit(scaledSubset3)
```

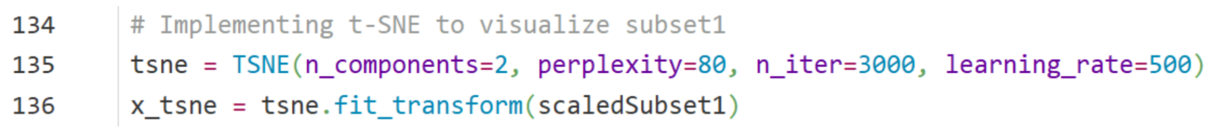
Elbow plot:



4. t-SNE Tuning. What are the optimal values for hyperparameters ‘perplexity’ and ‘number of iterations’?

Subset1:

```python
Subset1 = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'pH']]


Subset1['citric acid'] = Subset1['citric acid'].apply(converterCitricAcid)
Subset1['volatile acidity'] = Subset1['volatile
acidity'].apply(converterVolatileAcid)
```

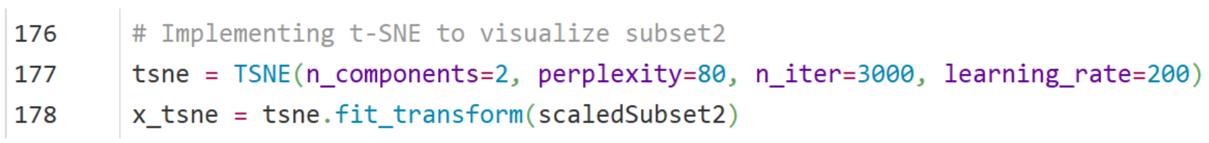


```python
# Implementing t-SNE to visualize subset1
tsne = TSNE(n_components=2, perplexity=80, n_iter=3000, learning_rate=500)
x_tsne = tsne.fit_transform(scaledSubset1)
```

Perplexity=80 Number of iteration

Subset2:

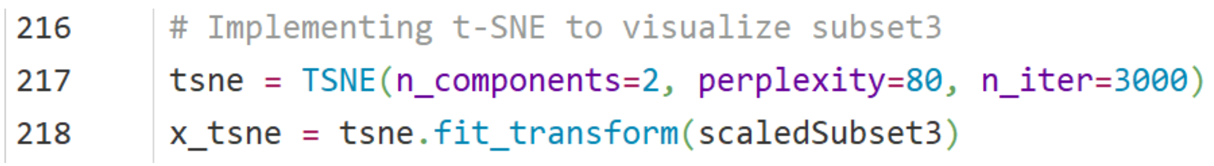
```python
Subset2 = dataset[['fixed acidity', 'alcohol', 'residual sugar', 'sulphates']]

Subset2['residual sugar'] = Subset2['residual sugar'].apply(converterResidualSugar)
Subset2['fixed acidity'] = Subset2['fixed acidity'].apply(converterFixedAcid)
```



```python
# Implementing t-SNE to visualize subset2
tsne = TSNE(n_components=2, perplexity=80, n_iter=3000, learning_rate=200)
x_tsne = tsne.fit_transform(scaledSubset2)
```

Subset3:

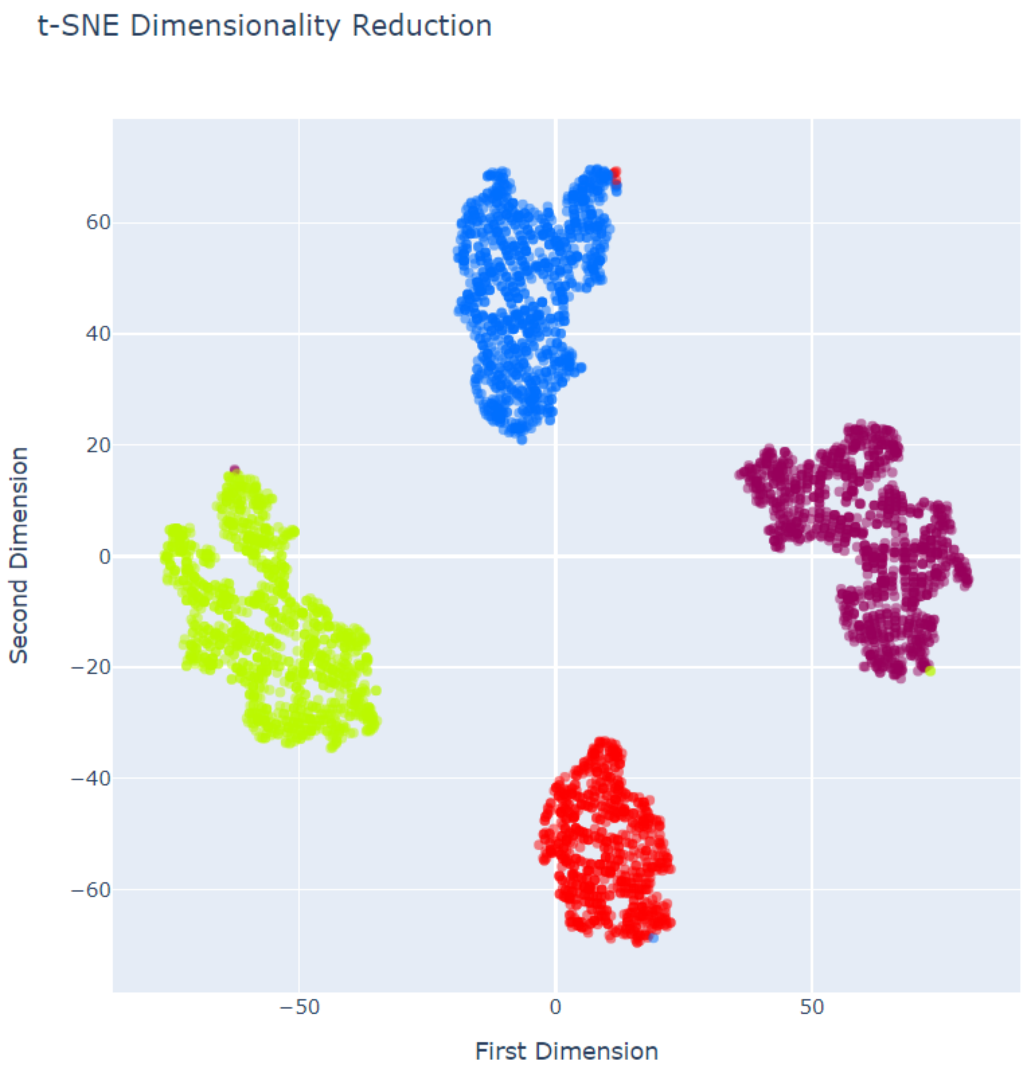
```python
Subset3 = dataset[['fixed acidity', 'alcohol', 'citric acid', 'pH', 'volatile acidity']]

Subset3['alcohol'] = Subset3['alcohol'].apply(converterAlcohol)
Subset3['volatile acidity'] = Subset3['volatile acidity'].apply(converterVolatileAcid)
```



```python
# Implementing t-SNE to visualize subset3
tsne = TSNE(n_components=2, perplexity=80, n_iter=3000)
x_tsne = tsne.fit_transform(scaledSubset3)
```

5. **<u>Cluster Interpretations. Can you assign any labels to resultant clusters? Can you find any target variable for this dataset or its subsets? If so, then what can the resultant dataset/ subset(s) be used for?</u>**

Subset1:



For Subset1 we can see that combination of all the acid with pH makes more perfect cluster as they are separable if citric acid and volatile acidity is converted into discrete class

Hence,

Yellow Cluster:

Label1 - citric acid < 0.33(0) and volatile acidity(0) <= 0.27

Blue Cluster:

Label2 – citric acid < 0.33(0) and volatile acidity(1) > 0.27

Red Cluster:

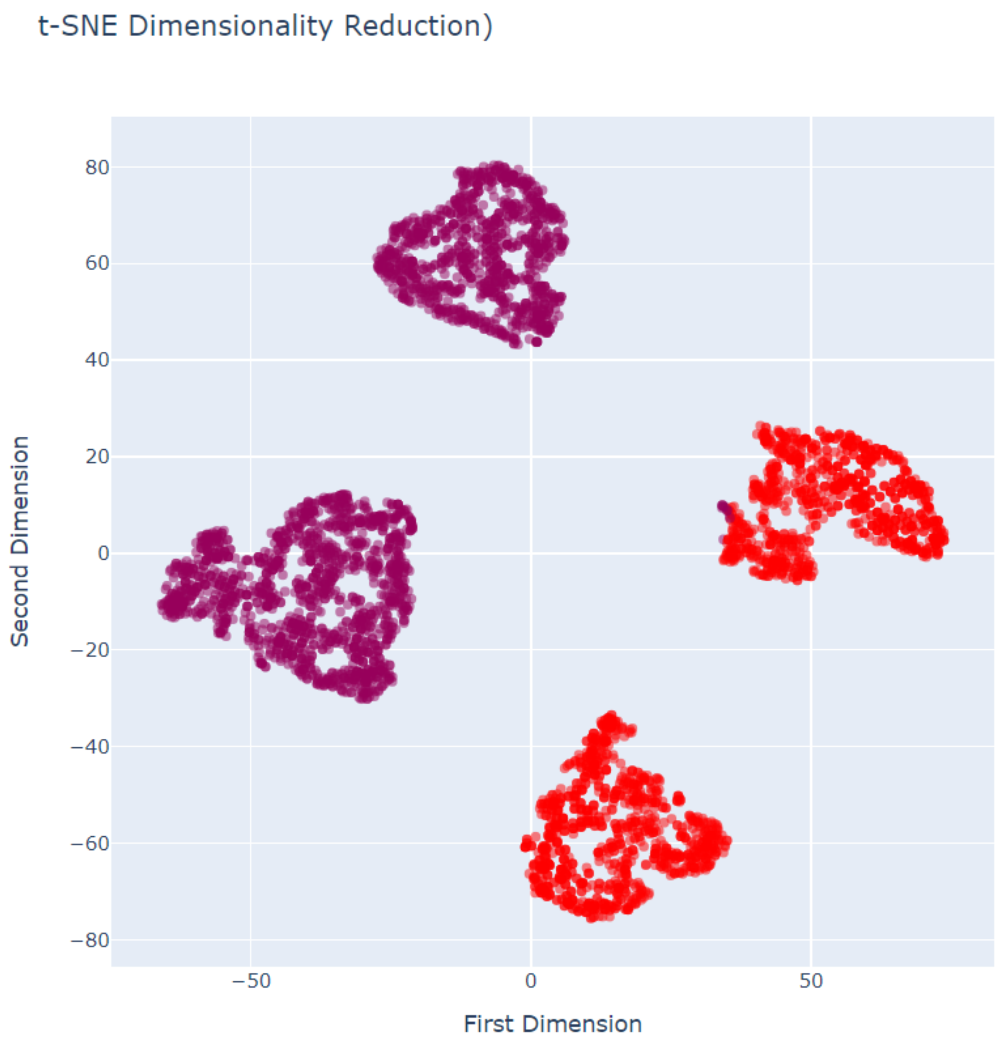
Label3 – citric acid >= 0.33(1) and volatile acidity(1) > 0.27

Purple Cluster:

Label4 – citric acid(1) >= 0.33 and volatile acidity(0) <= 0.27

Colo	Colour	Citric Acid	Volatile Acidity
Yellow	Subset1Y1	0	0
Blue	Subset1B1	0	1
Red	Subset1R1	1	1
Purple	Subset1P1	1	0

Subtset2:



Red Cluster: It has two sub clusters

Label-R1: fixed acidity(1) >= 6.85 and residual sugar(1) >= 6.20

Label-R2: fixed acidity(0) < 6.85 and residual sugar(1) >= 6.20

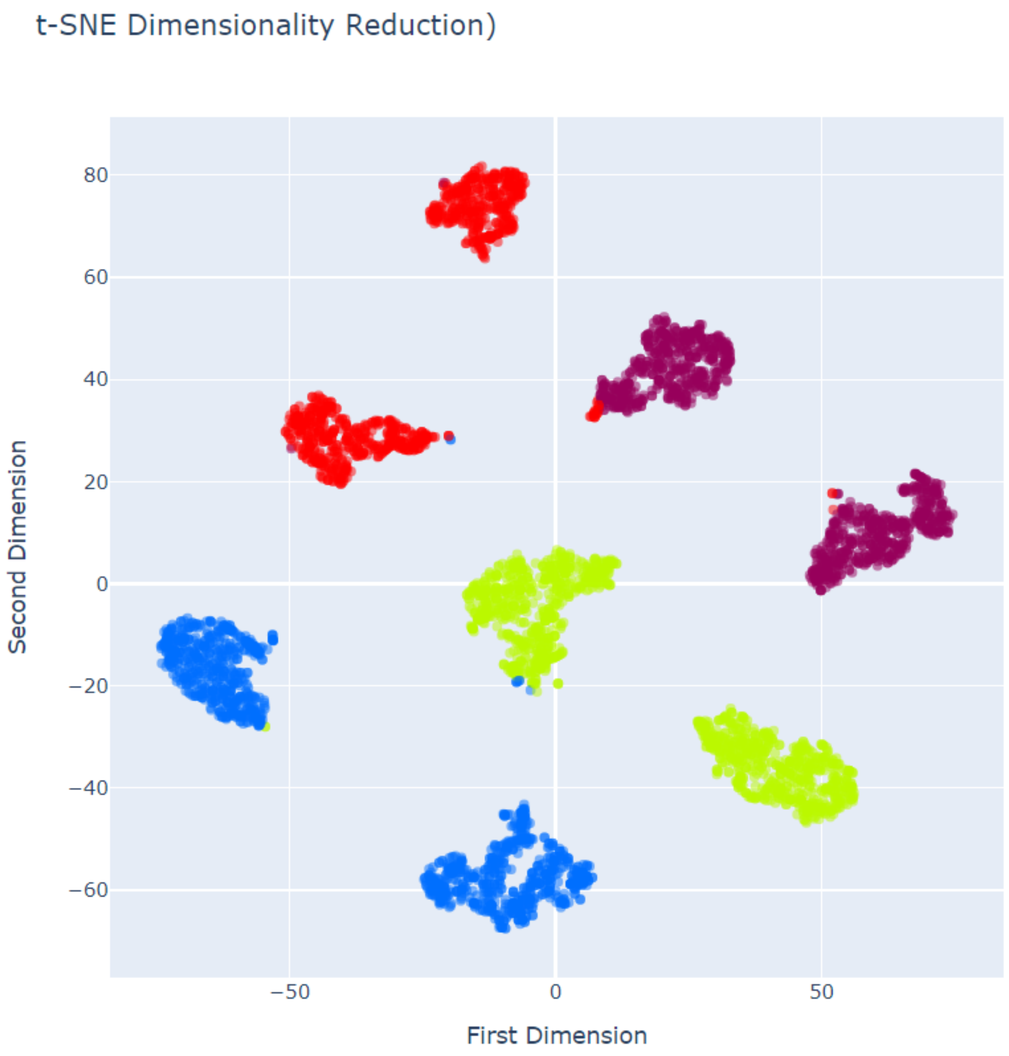
Purple Cluster: It has two sub clusters

Label-P1: fixed acidity(1) >=6.85 and residual sugar(0) <6.20

Label-P2: fixed acidity(0) < 6.85 and residual sugar(0) < 6.20

Colour	Label	Fixed Acidity	Residual Sugar
Red	Subset2P1	1	0
	Subset2P2	0	0
Purple	Subset2R1	1	1
	Subset2R2	0	1

Subset3:



Red Clusters: It has two sub clusters

LabelR1: fixed acidity(1) >= 6.85 volatile acidity(1) > 0.27 alcohol(1) >=10.5

LabelR2: fixed acidity(1) >=6.85 volatile acidity(1) > 0.27 alcohol(0) < 10.5

Purple Clusters: It has two sub clusters

LabelP1: fixed acidity(0) <6.85 volatile acidity(1) >0.27 alcohol(0) < 10.5

LabelP2: fixed acidity(0) <6.85 volatile acidity(1) >0.27 alcohol(1)>=10.5

Yellow Clusters: It has two sub clusters

LabelY1: fixed acidity(0) <6.85 volatile acidity(0) <=0.27 alcohol(0) <10.5

LabelY2: fixed acidity(0) <6.58 volatile acidity(0) <=0.27 alcohol(1) >= 10.5

Blue Clusters: It has two sub clusters

LabelB1: fixed acidity(1) >=6.85 volatile acidity(0) <=0.27 alcohol(1) >= 10.5

LabelB2: fixed acidity(1) >=6.85 volatile acidity(0) <=0.27 alcohol(0) <10.5

Colour	Labels	fixed acidity	volatile acidity	alcohol
Red	Subset3R1	1	1	1
	Subset3R2	1	1	0
Purple	Subset3P1	0	1	0
	Subset3P2	0	1	1
Yellow	Subset3Y1	0	0	0
	Subset3Y2	0	0	1
Blue	Subset3B1	1	0	1
	Subset3B2	1	0	0

Hence any new value from drink dataset arrives or needs to be shortlisted we need to check the value and according to that place in appropriate cluster using the tabular representation of the labels.

During my analysis it was easier to cluster if converting fixed acidity into discrete hence which shows that it has major impact with above used combination of different variables. As all the subsets and t-SNE visualization are developed by try and error methodology I choose random variables and tried to tune it by adjusting parameters of t-SNE and trying which classes will be best to convert into discrete values.

Therefore, depending on the values available of any new data, it will be separated accordingly.