

**SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING**

**DATA MINING TECHNIQUES- SWE2009**

**PROJECT REVIEW – 3**

# **TOPIC: ANALYSIS OF USER KNOWLEDGE MODELING USING CLASSIFICATION AND CLUSTERING TECHNIQUES**

**TEAM MEMBERS**

1. **Saiteja – 18MIS0360**
2. **Sandhya c – 18MIS0330**
3. **Jai swarupa santhoshini – 18MIS0155**

**Introduction :**

Data mining is the process of extracting hidden analytical information from large databases using multiple algorithms and techniques. This technology allows companies to focus on the most important information in their data warehouses . Data Mining becomes a vital aspect in data analysis. Study on data mining is very much depends on the performance of the clustering. Recently knowledge based approached has become the key forces in data classification. In this project we are going to use two classification and clustering algorithm for our User Knowledge dataset .Our topic is Analysis of user knowledge modeling using classification and clustering techniques . User knowledge includes data of their educational knowledge , their performance in exams etc . The data mining techniques we are going to use are classification and clustering techniques . In this project we have done two classification and clustering algorithm for our User Knowledge dataset. Algorithms used to classify the datasets are Decision Tree and Naïve Bayes Classifier. K-Means and Hierarchical Clustering algorithm are used to cluster the dataset.

**Objective :**

Objective is to understand and apply Naïve Bayes Classification, Decision Tree Classification , K – means clustering , Hierarchical clustering and apply them to identify the accuracy and error of the dataset

**Tool to be used : Jupyter Notebook**

**Literature review :**

**Improved K-means Algorithm for Searching Research Papers**: Clustering is one of the unsupervised learning method in which a set of essentials is separated into uniform groups. The k-means method is one of the most widely used clustering techniques for various applications. For the Searching as well as reading research papers users need more time or users spend two to three hours for searching or reading single papers, so this is more consuming process, so it is required that use enhanced search engine which is based on fastest reading algorithm which provides best output or results. So we are proposed Enhanced architecture with improved k-means algorithm, which proposes a method for making the algorithm more effective and efficient, so as to get better clustering with reduced complexity. It will search the base keyword of the content from the knowledge database. Proposed work uses the search engine based on clustering and text mining.

**Hierarchical Clustering Algorithms for Document Datasets:** Fast and high-quality document clustering algorithms play an important role in providing intuitive navigation and browsing mechanisms by organizing large amounts of information into a small number of meaningful clusters. In particular, clustering algorithms that build meaningful hierarchies out of large document collections are ideal tools for their interactive visualization and exploration as they provide data-views that are consistent, predictable, and at different levels of granularity. This paper focuses on document clustering algorithms that build such hierarchical solutions and presents a comprehensive study of partitional and agglomerative algorithms that use different criterion functions and merging schemes, and presents a new class of clustering algorithms called constrained agglomerative algorithms, which combine features from both partitional and agglomerative approaches that allows them to reduce the early-stage errors made by agglomerative methods and hence improve the quality of clustering solutions. The experimental evaluation shows that, contrary to the common belief, partitional algorithms always lead to better solutions than agglomerative algorithms; making them ideal for clustering large document collections due to not only their relatively low computational requirements, but also higher clustering quality. Furthermore, the constrained agglomerative methods consistently lead to better solutions than agglomerative methods alone and for many cases they outperform partitional methods, as well.

**Naive Bayes Classification of Uncertain Data:** Traditional machine learning algorithms assume that data are exact or precise. However, this assumption may not hold in some situations because of data uncertainty arising from measurement errors, data staleness, and repeated measurements, etc. With uncertainty, the value of each data item is represented by a probability distribution function. In this paper, we propose a novel naive Bayes classification algorithm for uncertain data with a pdf. Our key solution is to extend the class conditional probability estimation in the Bayes model to handle pdf’s. Extensive experiments on UCI datasets show that the accuracy of naive Bayes model can be improved by taking into account the uncertainty information

**Dataset info :**

It is the real dataset about the student’s knowledge status of the Electrical DC Machines subject.

Link : <https://archive.ics.uci.edu/ml/datasets/User+Knowledge+Modeling>

**Author :**

H. T. Kahraman, Sagiroglu, S., Colak, I., Developing intuitive knowledge classifier and modeling of users' domain dependent data in web, Knowledge Based Systems, vol. 37, pp. 283-295.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Set Characteristics** | Multivariate | **Area** | **Education** |
| **Attribute Characteristics** | Real | **Number of Instances** | 403 |
| **Associated tasks** | Classification, Clustering | **Number of Attributes** | 5 |

**Attribute description :**

**4.1 Input Value:**

**1.) STG** (The degree of study time for goal object materials)

Minimum Value: 0

Maximum Value: 0.99

**2.) SCG** (The degree of repetition number of user for goal object materials)

Minimum Value: 0

Maximum Value: 0.9

**3.) STR** (The degree of study time of user for related objects with goal object)

Minimum Value: 0

Maximum Value: 0.95

**4.) LPR** (The exam performance of user for related objects with goal object)

Minimum Value: 0

Maximum Value: 0.99

**5.) PEG** (The exam performance of user for goal objects)

Minimum Value: 0

Maximum Value: 0.93

**Contd.**

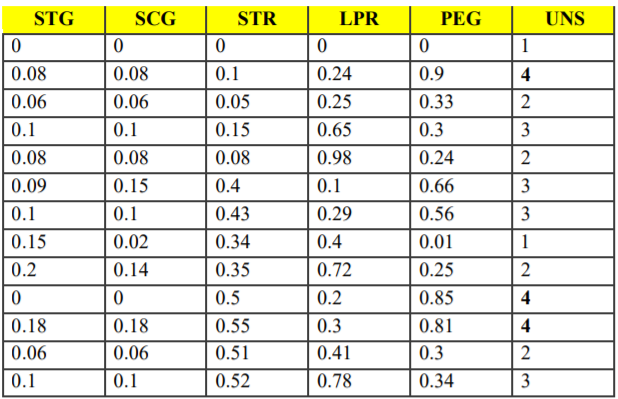
**Target Value:** (Class Distribution)

**1.) UNS (The knowledge level of user)**

* Very low: 50
* Low: 129
* Middle: 122
* High: 102

There are totally 403 instances in User Knowledge Modeling data set. Approximately there must be 60% of instances in Training data set and remaining 40% of instances in Test data set. After a model has been processed by using the training set, you test the model by making predictions against the test set. Because the data in the testing set already contains known values for the attribute that you want to predict, it is easy to determine whether the model's guesses are correct. By using similar data for training and testing, you can minimize the effects of data discrepancies and better understand the characteristics of the model.

**Sample database :**



**Classification techniques:**

1. **Naïve Bayes Classification:**

* Convert the data set into a frequency table.
* Create Likelihood table to find the probabilities.
* Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

**P(L | X) =( P(X | L) P(L) )/ P(X)**

**. Decision Tree Classification**

* Place the best attribute of the dataset at the root of the tree.
* Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
* Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

**Clustering techniques :**

**K – means clustering:**

1. Read the inputs from the dataset 'Data\_User\_Modeling.xls' by using ‘xlsread’ command.
2. 2. Set the number of clusters as 4
3. 3. Using the k-means inbuilt function perform the k-means algorithm.

The algorithm proceeds as follows:

i. Choose k initial cluster centers (centroid).

ii. Compute point-to-cluster-centroid distances of all observations to each centroid.

iii. Assign each observation to the cluster with the closest centroid.

iv. Compute the average of the observations in each cluster to obtain k new centroid locations.

v. Repeat steps 2 through 4 until cluster assignments do not change, or the maximum number of iterations is reached.

4. Plot the clusters using the plot function.

**Hierarchial clustering:**

1. **Agglomerative** – Begins with each elements as separate cluster and merge them into successively large cluster.
2. **Devisive** – Begins with the whole set and proceed to divide in into successively smaller cluster.
3. Clustering is grouping the related object into one group of cluster and group the unrelated object into another group of cluster.

S1. Convert object attributes to distance matrix.

S2. Set each object as a cluster.

S3. Repeat until number of cluster equal to 1.

* + 1. Merge two closest clusters
    2. Update distance matrix.

**Code for K-means clustering:**

import os

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import plotly as py

import plotly.graph\_objs as go

from sklearn.cluster import KMeans

import warnings

warnings.filterwarnings('ignore')

users = pd.read\_csv('data.csv', delimiter=',')

users.head()

users.shape

users.dtypes

users.isna().sum()

users.duplicated().sum()

users.describe()

plt.figure(1 , figsize = (15 , 4))

pal1 = ["#FA5858", "#58D3F7", "#704041", "#f5c3c4"]

sns.countplot(y = ' UNS' , data = users, palette=pal1)

plt.show()

# plot histograms for each variable

users.hist(figsize = (12, 12))

plt.show()

plt.figure(figsize=(10,10))

sns.heatmap(users.corr(),annot=True,fmt='.1f')

plt.show()

sns.set(style="ticks")

pal = ["#ff0000", "#1a1aff", "#660066", "#333300"]

sns.pairplot(users, hue=" UNS", palette=pal)

plt.title(" UNS")

#K-Means Clustering

X = users[['STG' , 'PEG']].iloc[: , :].values

inertia = []

for n in range(1 , 10):

models = (KMeans(n\_clusters = n ,init='k-means++', n\_init = 10 ,max\_iter=100, tol=0.0001, random\_state= 100 , algorithm='elkan') )

models.fit(X)

inertia.append(models.inertia\_)

plt.figure(1 , figsize = (15 ,6))

plt.plot(np.arange(1 , 10) , inertia , 'o')

plt.plot(np.arange(1 , 10) , inertia , '-' , alpha = 0.5)

plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')

plt.show()

models = (KMeans(n\_clusters = 3 ,init='k-means++', n\_init = 10 ,max\_iter=300,

tol=0.0001, random\_state= 111 , algorithm='elkan') )

models.fit(X)

labels = models.labels\_

centroids = models.cluster\_centers\_

print(models.cluster\_centers\_)

print(models.inertia\_)

print(models.n\_iter\_)

fig = plt.figure(figsize=(12,8))

plt.scatter(X[:,0], X[:,1], c=models.labels\_, cmap="Set1\_r", s=25)

plt.scatter(models.cluster\_centers\_[:,0] ,models.cluster\_centers\_[:,1], color='blue', marker="\*", s=250)

plt.title("Kmeans Clustering \n Finding Unknown Groups in the Population", fontsize=16)

plt.show()

models2 = (KMeans(n\_clusters = 4 ,init='k-means++', n\_init = 10 ,max\_iter=300,

tol=0.0001, random\_state= 111 , algorithm='elkan') )

models2.fit(X)

labels2 = models2.labels\_

centroids2 = models2.cluster\_centers\_

print(models2.cluster\_centers\_)

print(models2.inertia\_)

print(models2.n\_iter\_)

fig = plt.figure(figsize=(12,8))

plt.scatter(X[:,0], X[:,1], c=models2.labels\_, cmap="Set1\_r", s=25)

plt.scatter(models2.cluster\_centers\_[:,0] ,models2.cluster\_centers\_[:,1], color='blue', marker="\*", s=250)

plt.title("Kmeans Clustering \n Finding Unknown Groups in the Population", fontsize=16)

plt.show()

#3D Visualization

X1 = users[['STG' , 'LPR', 'PEG']].iloc[: , :].values

inertia = []

for n in range(1 , 10):

models3 = (KMeans(n\_clusters = n ,init='k-means++', n\_init = 10 ,max\_iter=100,

tol=0.0001, random\_state= 100 , algorithm='elkan') )

models3.fit(X)

inertia.append(models3.inertia\_)

plt.figure(1 , figsize = (15 ,6))

plt.plot(np.arange(1 , 10) , inertia , 'o')

plt.plot(np.arange(1 , 10) , inertia , '-' , alpha = 0.5)

plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')

plt.show()

**Code for Decision Tree classification:**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt # data visualization

import seaborn as sns # statistical data visualization

%matplotlib inline

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

import warnings

warnings.filterwarnings('ignore')

data = 'data.csv'

df = pd.read\_csv(data, header=None)

print(df)

df.shape

# preview the dataset

df.head()

col\_names = ['STG', 'SCG', 'STR', 'LPR', 'PEG', 'UNS']

df.columns = col\_names

col\_names

df.head()

df.info()

col\_names = ['STG', 'SCG', 'STR', 'LPR', 'PEG', 'UNS']

for col in col\_names:

print(df[col].value\_counts())

df['UNS'].value\_counts()

# check missing values in variables

df.isnull().sum()

X = df.drop(['UNS'], axis=1)

y = df['UNS']

# split X and y into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33, random\_state = 42)

# check the shape of X\_train and X\_test

X\_train.shape, X\_test.shape

# check data types in X\_train

X\_train.dtypes

X\_train.head()

pip install --upgrade category\_encoders

# import category encoders

import category\_encoders as ce

# encode variables with ordinal encoding

encoder = ce.OrdinalEncoder(cols=['STG', 'SCG', 'STR', 'LPR', 'PEG']

)

X\_train = encoder.fit\_transform(X\_train)

X\_test = encoder.transform(X\_test)

X\_train.head()

X\_test.head()

# import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier

import warnings

warnings.filterwarnings('ignore')

# instantiate the DecisionTreeClassifier model with criterion gini index

clf\_gini = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=0)

# fit the model

clf\_gini.fit(X\_train, y\_train)

y\_pred\_gini = clf\_gini.predict(X\_test)

from sklearn.metrics import accuracy\_score

print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred\_gini)))

y\_pred\_train\_gini = clf\_gini.predict(X\_train)

y\_pred\_train\_gini

print('Training-set accuracy score: {0:0.4f}'. format(accuracy\_score(y\_train, y\_pred\_train\_gini)))

#Check for overfitting and underfitting

# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf\_gini.score(X\_train, y\_train)))

print('Test set score: {:.4f}'.format(clf\_gini.score(X\_test, y\_test)))

#Visualize decision-trees

plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot\_tree(clf\_gini.fit(X\_train, y\_train))

pip install graphviz

#Decision Tree Classifier with criterion entropy

# instantiate the DecisionTreeClassifier model with criterion entropy

clf\_en = DecisionTreeClassifier(criterion='entropy', max\_depth=3, random\_state=0)

# fit the model

clf\_en.fit(X\_train, y\_train)

y\_pred\_en = clf\_en.predict(X\_test)

from sklearn.metrics import accuracy\_score

print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred\_en)))

y\_pred\_train\_en = clf\_en.predict(X\_train)

y\_pred\_train\_en

print('Training-set accuracy score: {0:0.4f}'. format(accuracy\_score(y\_train, y\_pred\_train\_en)))

##Check for overfitting and underfitting¶

# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf\_en.score(X\_train, y\_train)))

print('Test set score: {:.4f}'.format(clf\_en.score(X\_test, y\_test)))

plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot\_tree(clf\_en.fit(X\_train, y\_train))

#Confusion matrix

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred\_en)

print('Confusion matrix\n\n', cm)

#Classification Report

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred\_en))

**Code for Hierarchical clustering**

import os

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import plotly as py

import plotly.graph\_objs as go

from sklearn.cluster import KMeans

import warnings

warnings.filterwarnings('ignore')

users = pd.read\_csv('data.csv', delimiter=',')

users.head()

pip install plotly

pip install cufflinks

users.shape

users.shape

users.isna().sum()

users.duplicated().sum()

users.describe()

plt.figure(1 , figsize = (15 , 4))

pal1 = ["#FA5858", "#58D3F7", "#704041", "#f5c3c4"]

sns.countplot(y = ' UNS' , data = users, palette=pal1)

plt.show()

# plot histograms for each variable

users.hist(figsize = (12, 12))

plt.show()

plt.figure(figsize=(10,10))

sns.heatmap(users.corr(),annot=True,fmt='.1f')

plt.show()

plt.rcParams['figure.figsize'] = (15, 8)

sns.heatmap(users.corr(), cmap = 'Wistia', annot = True)

plt.title('Heatmap for the Data', fontsize = 20)

plt.show()

sns.set(style="ticks")

pal = ["#ff0000", "#1a1aff", "#660066", "#333300"]

sns.pairplot(users, hue=" UNS", palette=pal)

plt.title(" UNS")

#Hierarchical clustering

import scipy.cluster.hierarchy as shc

X = users[['STG', 'SCG', 'STR', 'LPR', 'PEG']].iloc[: , :].values

plt.figure(figsize=(10, 7))

plt.title("User Knowledge Dendograms")

plt.xlabel('Users')

dend = shc.dendrogram(shc.linkage(X, method='complete'))

**Code for Naïve Bayes classification:**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt # data visualization

import seaborn as sns # statistical data visualization

%matplotlib inline

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

import warnings

warnings.filterwarnings('ignore')

data = 'data.csv'

df = pd.read\_csv(data, header=None)

print(df)

df

df.shape

# preview the dataset

df.head()

col\_names = ['STG', 'SCG', 'STR', 'LPR', 'PEG', 'UNS']

df.columns = col\_names

col\_names

df.info()

# let's again preview the dataset

df.head()

X = df.drop(['UNS'], axis=1)

y = df['UNS']

X

y

# split X and y into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 0)

# check the shape of X\_train and X\_test

X\_train.shape, X\_test.shape

# check data types in X\_train

X\_train.dtypes

# display categorical variables

categorical = [col for col in X\_train.columns if X\_train[col].dtypes == 'O']

categorical

# display numerical variables

numerical = [col for col in X\_train.columns if X\_train[col].dtypes != 'O']

numerical

# print percentage of missing values in the categorical variables in training set

X\_train[categorical].isnull().mean()

# print categorical variables with missing data

for col in categorical:

if X\_train[col].isnull().mean()>0:

print(col, (X\_train[col].isnull().mean()))

# impute missing categorical variables with most frequent value

for df2 in [X\_train, X\_test]:

df2['STG'].fillna(X\_train['STG'].mode()[0], inplace=True)

df2['SCG'].fillna(X\_train['SCG'].mode()[0], inplace=True)

df2['PEG'].fillna(X\_train['PEG'].mode()[0], inplace=True)

# check missing values in categorical variables in X\_train

X\_train[categorical].isnull().sum()

Categorical

X\_train[categorical].head()

# import category encoders

import category\_encoders as ce

# encode remaining variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['STG', 'SCG', 'STR', 'LPR', 'PEG'])

X\_train = encoder.fit\_transform(X\_train)

X\_test = encoder.transform(X\_test)

X\_train.shape

X\_test.head()

cols = X\_train.columns

from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

X\_train = pd.DataFrame(X\_train, columns=[cols])

X\_test = pd.DataFrame(X\_test, columns=[cols])

X\_train.head()

# train a Gaussian Naive Bayes classifier on the training set

from sklearn.naive\_bayes import GaussianNB

# instantiate the model

gnb = GaussianNB()

# fit the model

gnb.fit(X\_train, y\_train)

y\_pred = gnb.predict(X\_test)

y\_pred

from sklearn.metrics import accuracy\_score

print('Model accuracy score: {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))

y\_pred\_train = gnb.predict(X\_train)

y\_pred\_train

print('Training-set accuracy score: {0:0.4f}'. format(accuracy\_score(y\_train, y\_pred\_train)))

#Check for overfitting and underfitting

# print the scores on training and test set

print('Training set score: {:.4f}'.format(gnb.score(X\_train, y\_train)))

print('Test set score: {:.4f}'.format(gnb.score(X\_test, y\_test)))

# check class distribution in test set

y\_test.value\_counts()

# check null accuracy score

null\_accuracy = (7407/(7407+2362))

print('Null accuracy score: {0:0.4f}'. format(null\_accuracy))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])

# visualize confusion matrix with seaborn heatmap

cm\_matrix = pd.DataFrame(data=cm, columns=['Actual Positive', 'Actual Negative','Actual Positive','Actual Positive'],

index=['Predict Positive', 'Predict Negative','Predict Positive', 'Predict Negative'])

sns.heatmap(cm\_matrix, annot=True, fmt='d', cmap='YlGnBu')

#Classification metrices

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

# print classification accuracy

classification\_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification\_accuracy))

# print classification error

classification\_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification\_error))

# print precision score

precision = TP / float(TP + FP)

print('Precision : {0:0.4f}'.format(precision))

#recall

recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))

#True Positive Rate

#True Positive Rate is synonymous with Recall.

true\_positive\_rate = TP / float(TP + FN)

print('True Positive Rate : {0:0.4f}'.format(true\_positive\_rate))

#False Positive Rate

false\_positive\_rate = FP / float(FP + TN)

print('False Positive Rate : {0:0.4f}'.format(false\_positive\_rate))

specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))

#class probabilities

# print the first 10 predicted probabilities of two classes- 0 and 1

y\_pred\_prob = gnb.predict\_proba(X\_test)[0:10]

y\_pred\_prob

# print the first 10 predicted probabilities for class 1 - Probability of >50K

gnb.predict\_proba(X\_test)[0:10, 1]

# store the predicted probabilities for class 1 - Probability of >50K

y\_pred1 = gnb.predict\_proba(X\_test)[:, 1]

# plot histogram of predicted probabilities

# adjust the font size

plt.rcParams['font.size'] = 12

# plot histogram with 10 bins

plt.hist(y\_pred1, bins = 10)

# set the x-axis limit

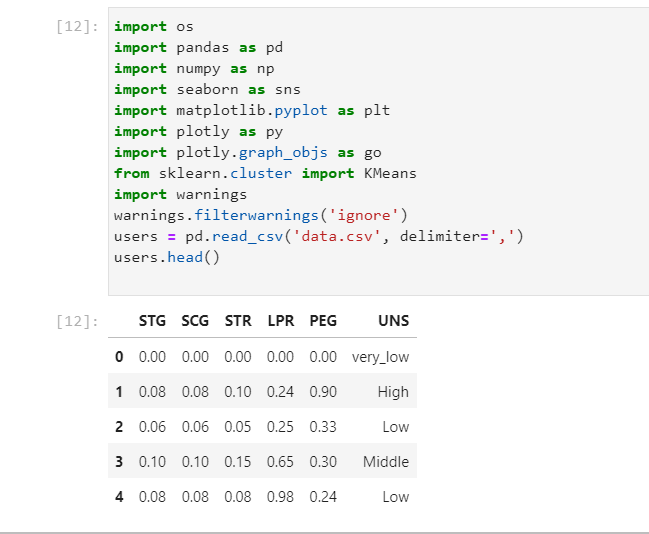
plt.xlim(0,1)

# set the title

plt.ylabel('Frequency')

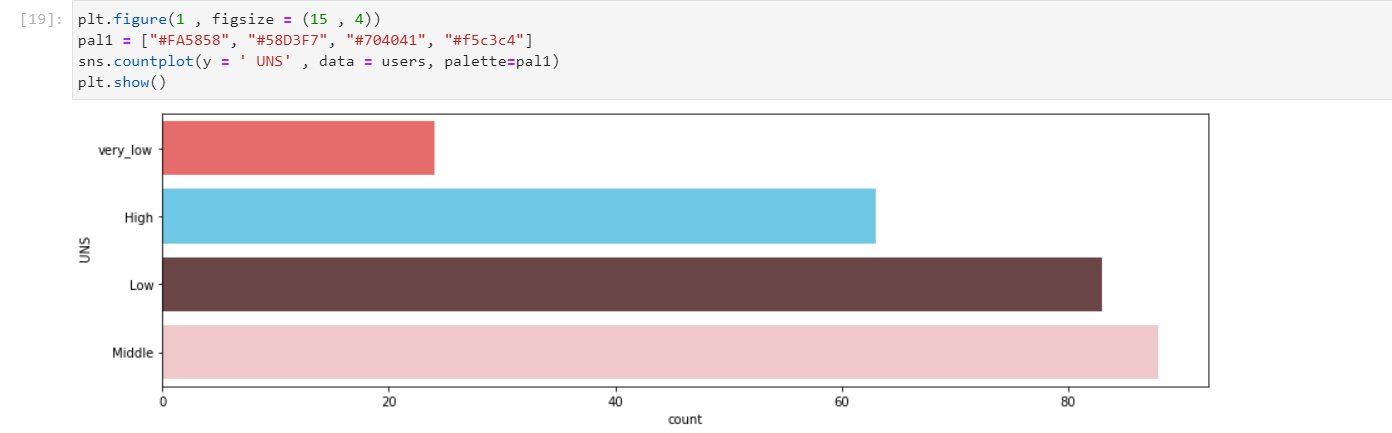
**Output with code:**

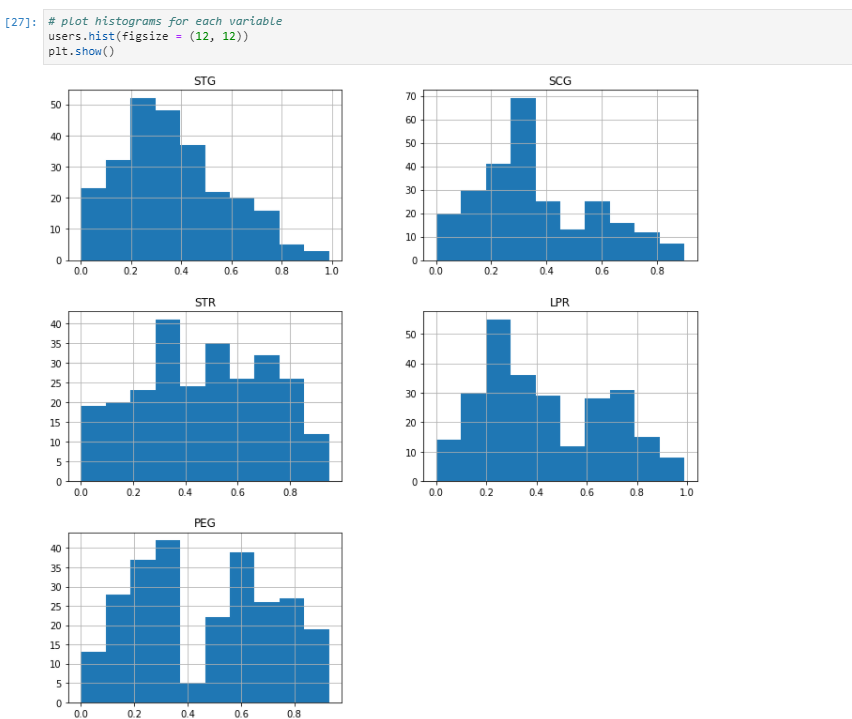
**K-means clustering:**

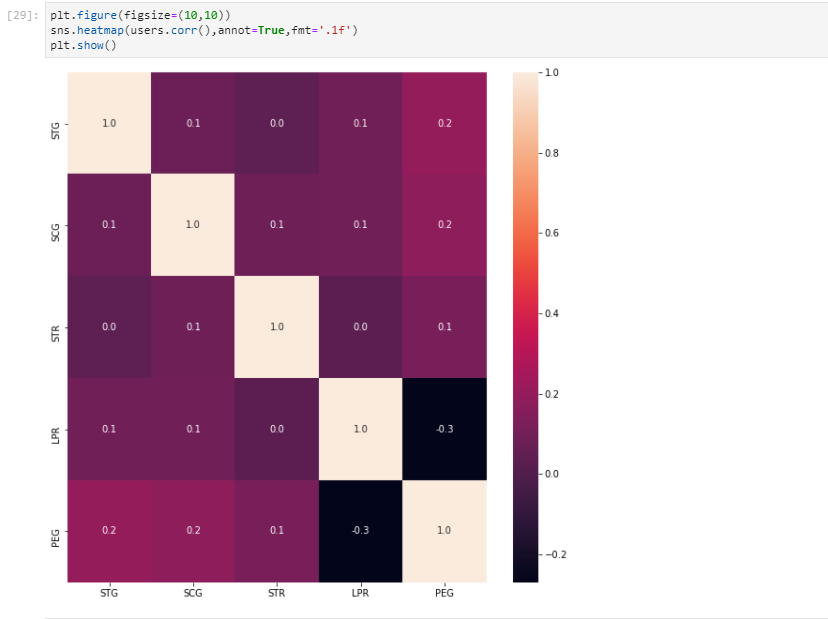


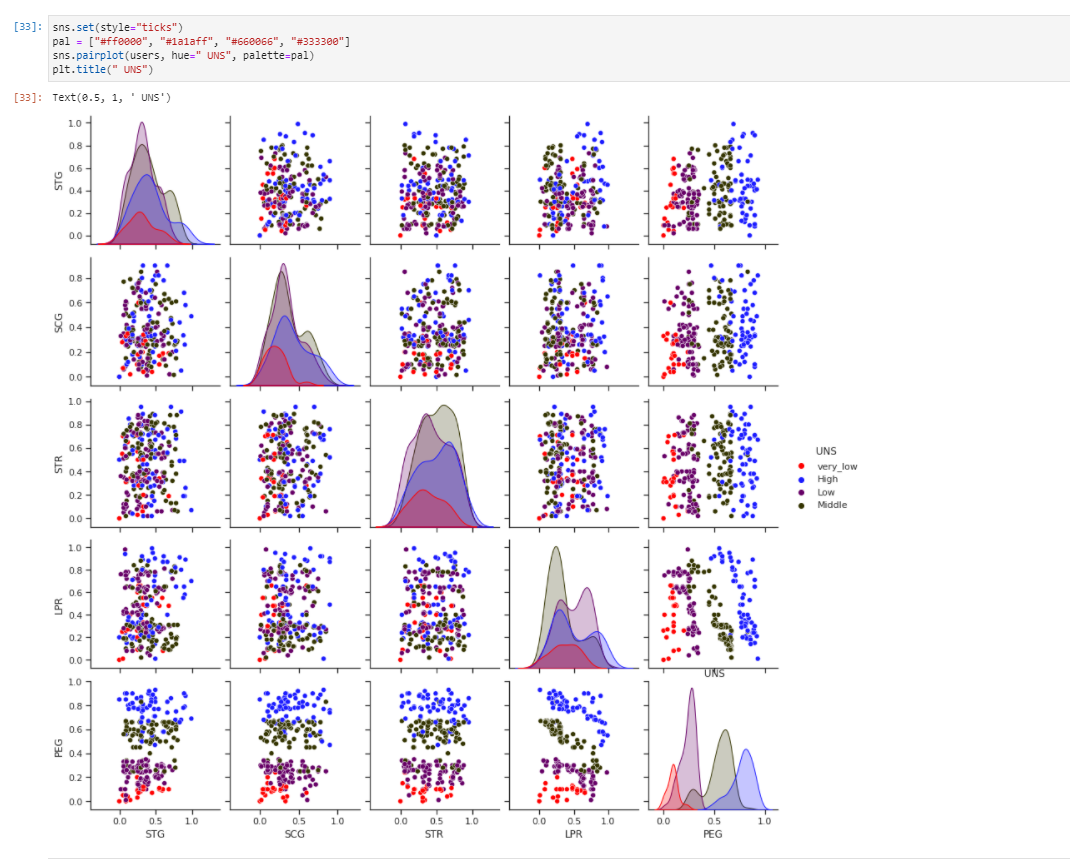


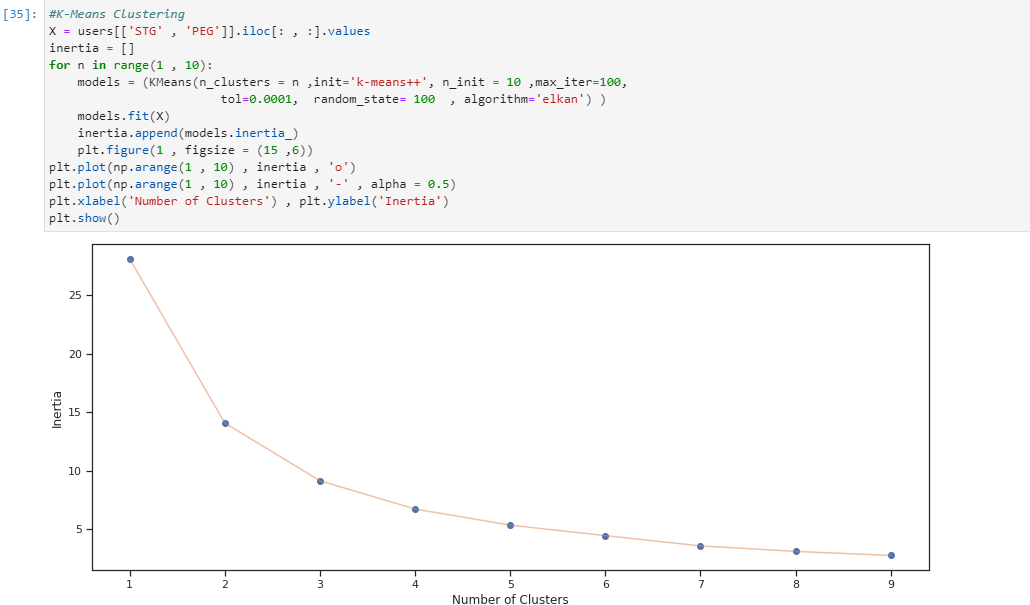




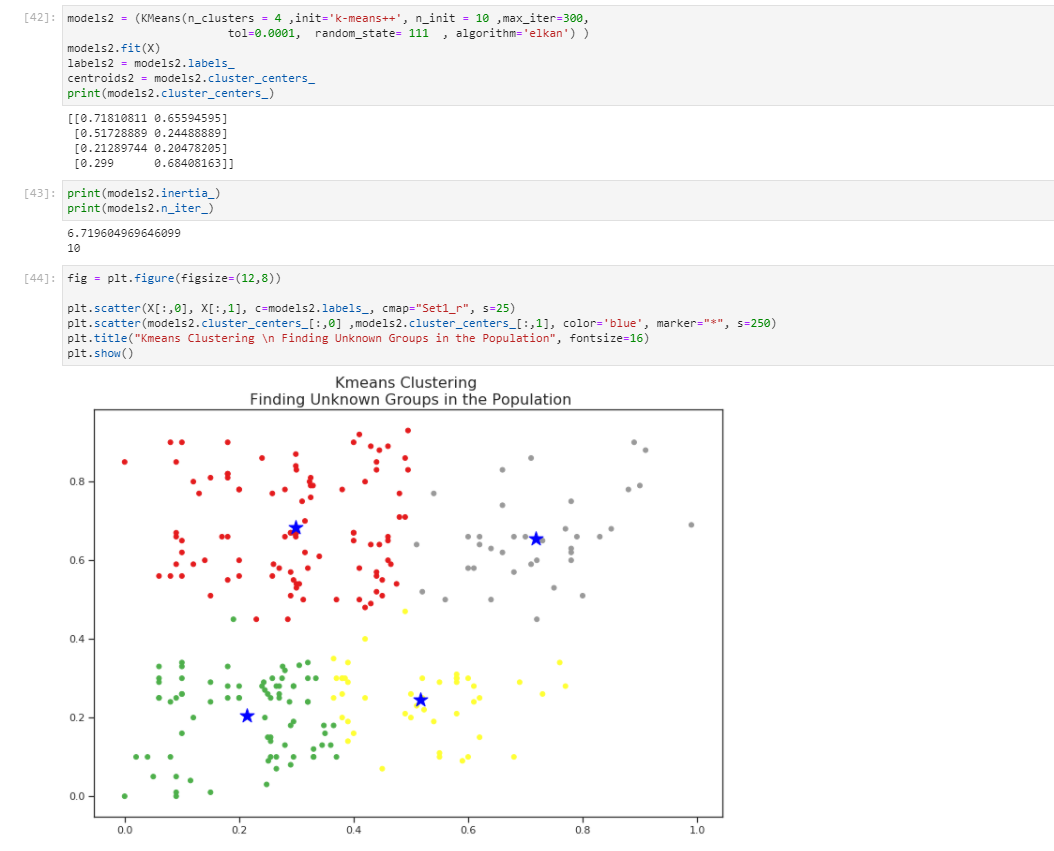


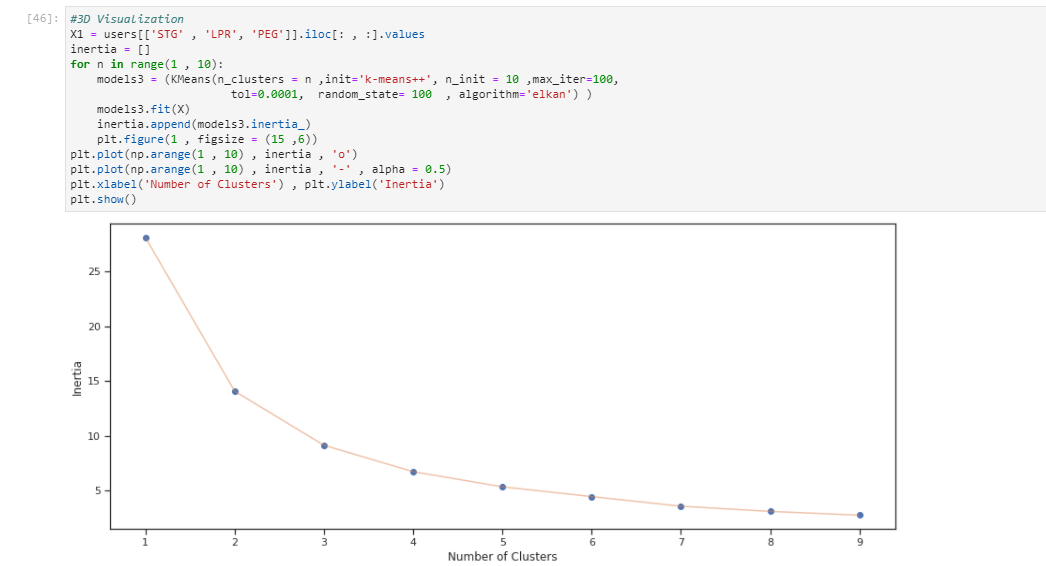




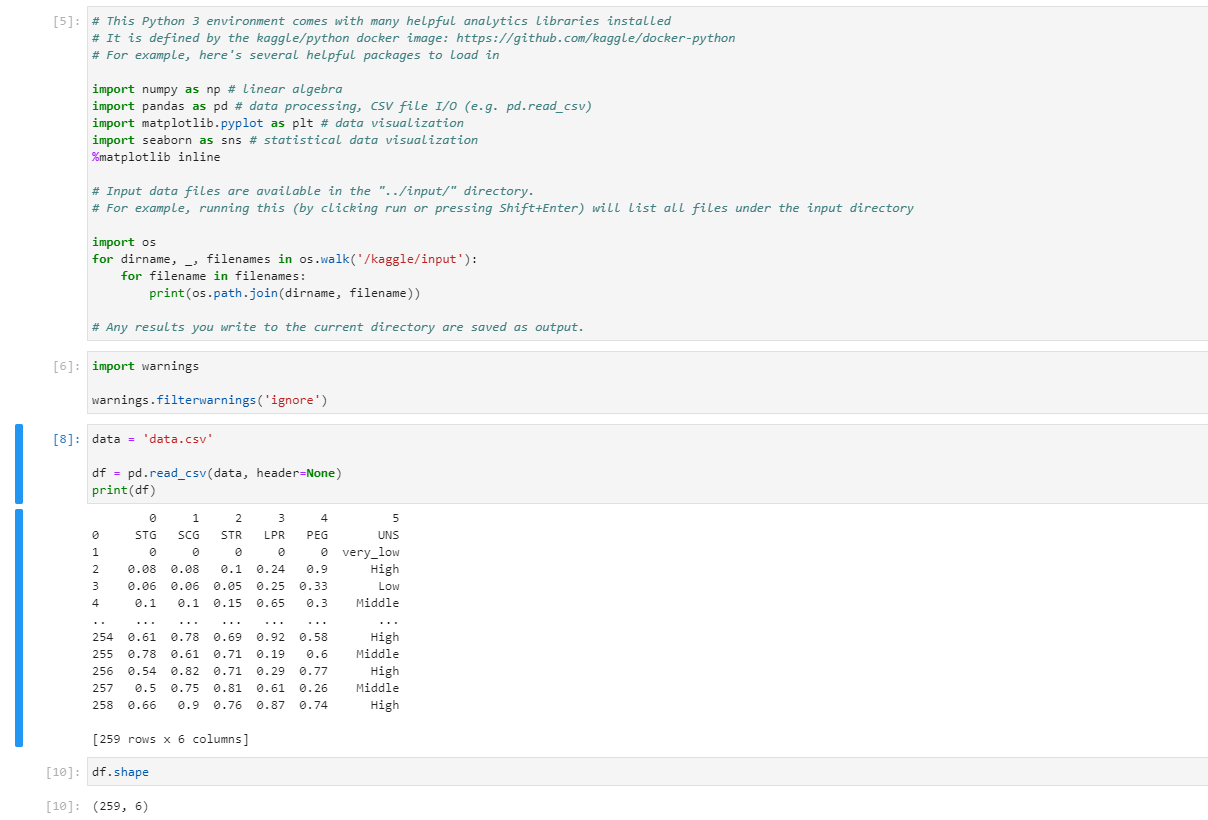


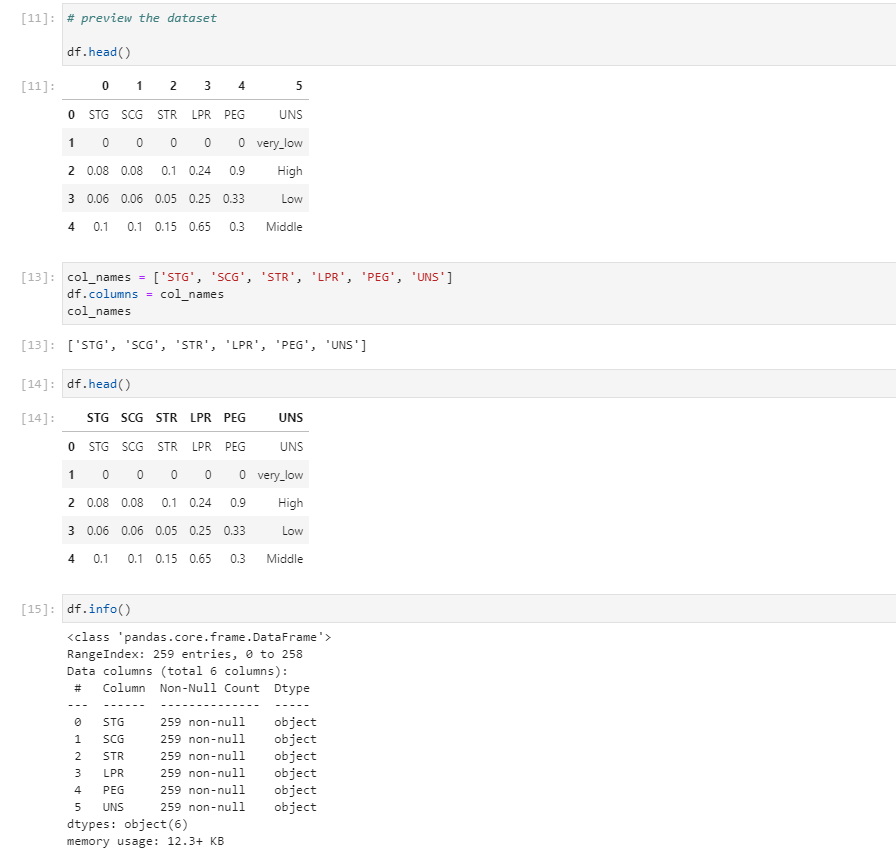


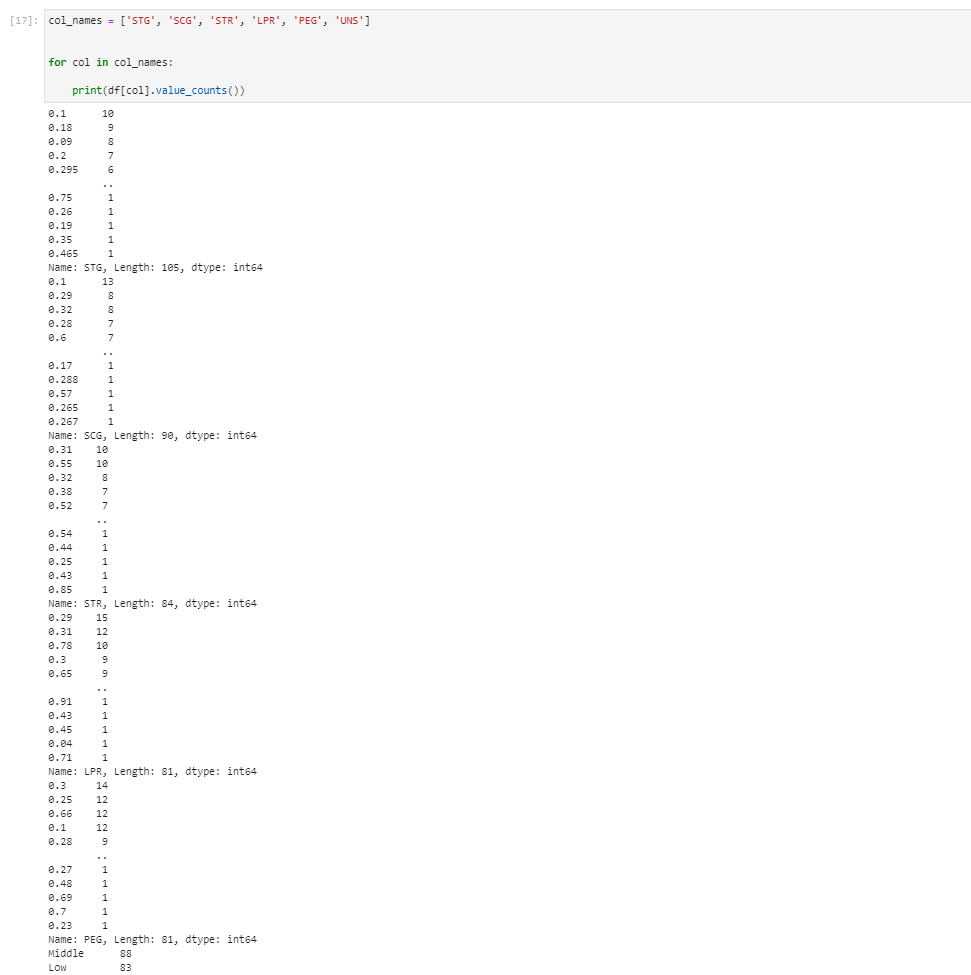


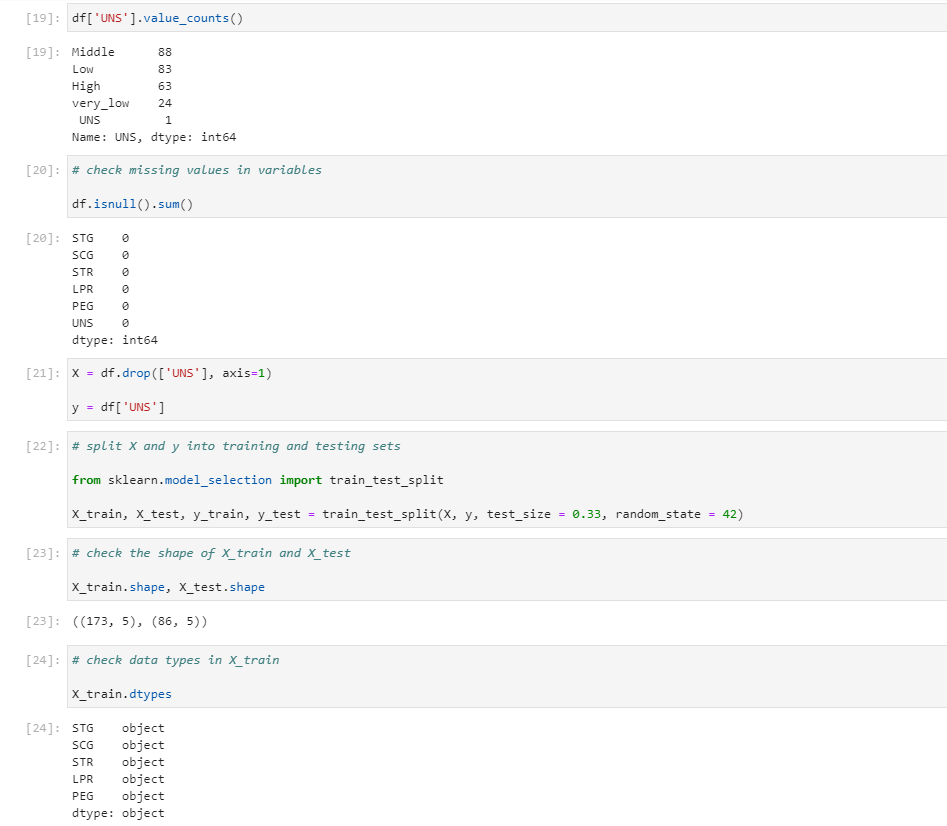


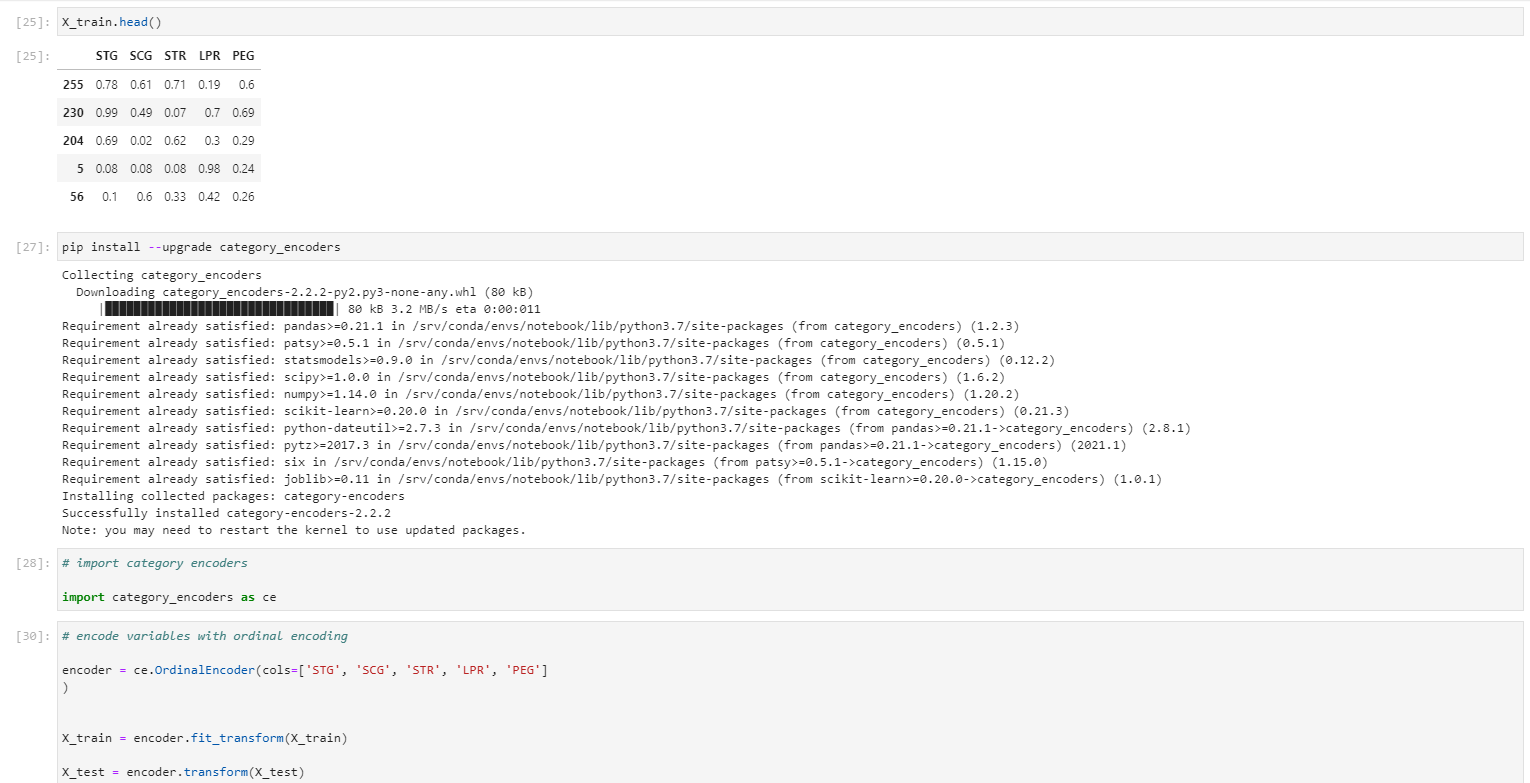
**Decision tree classification:**

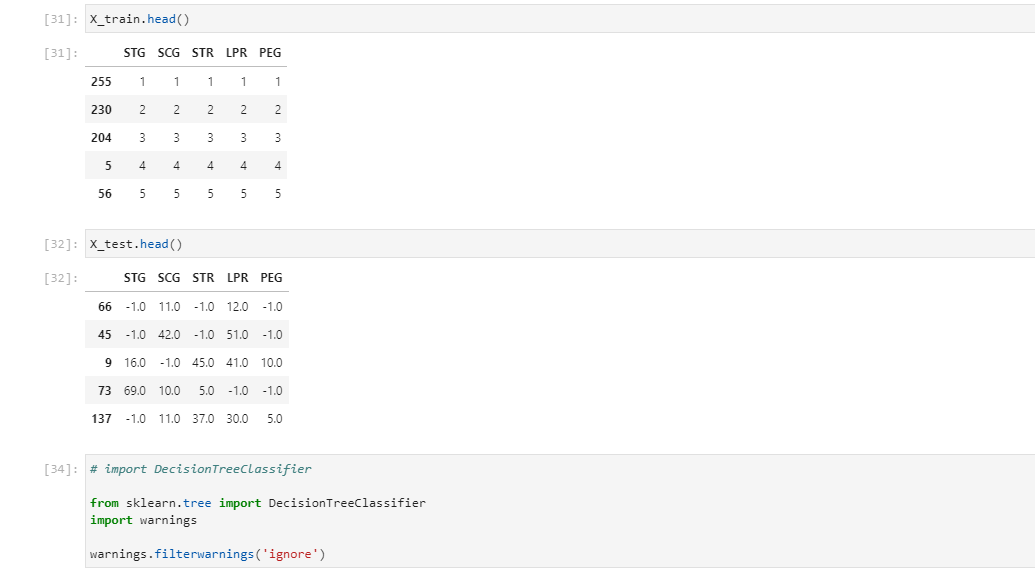


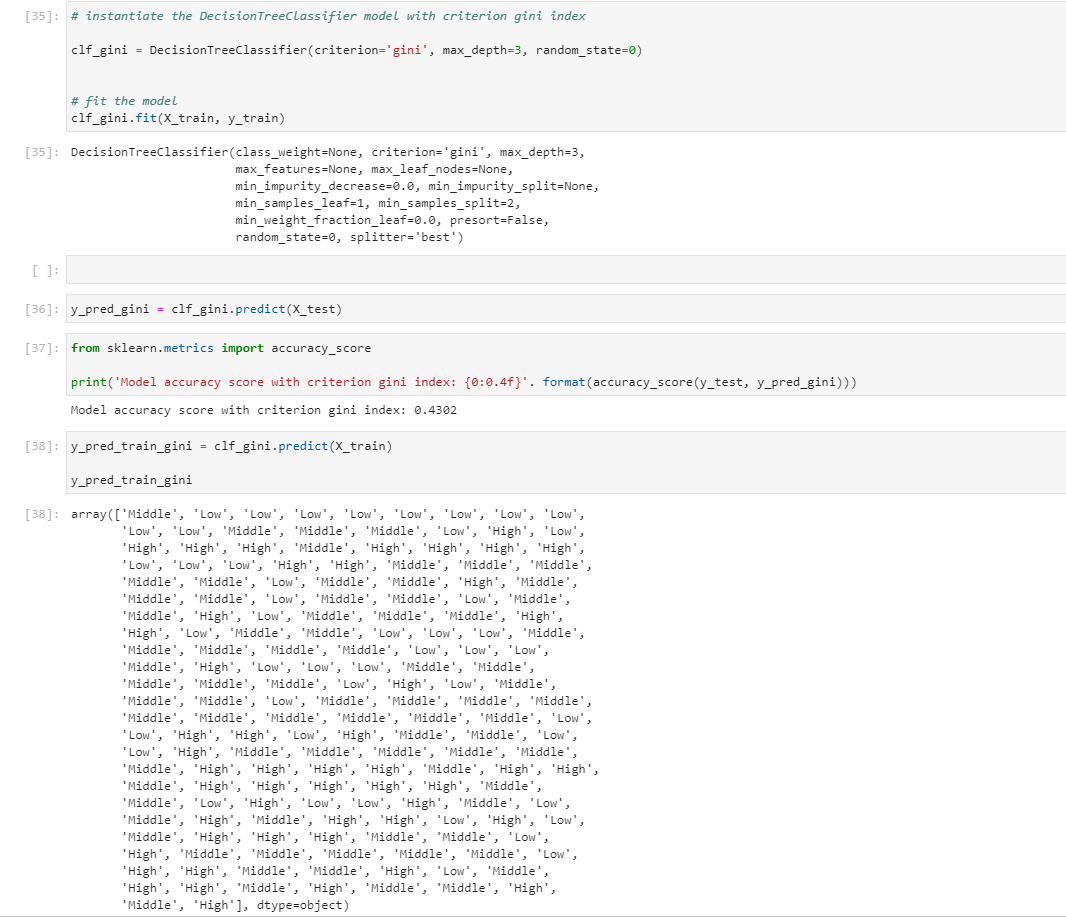




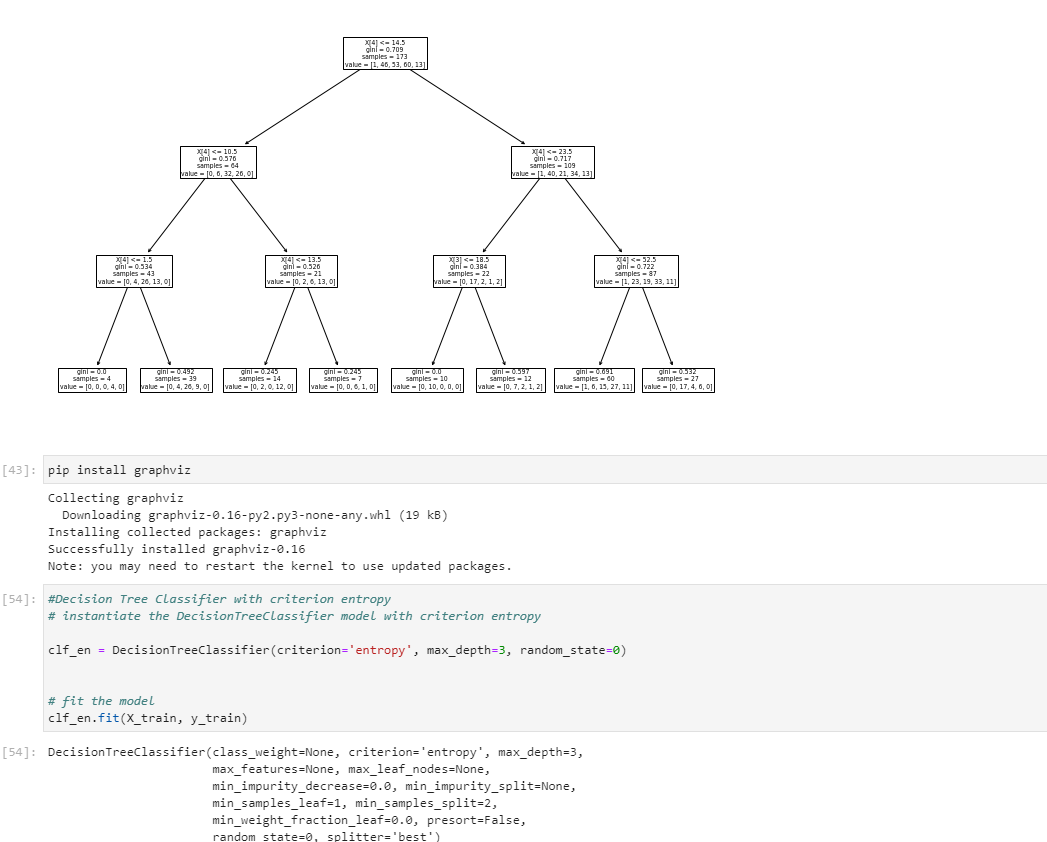




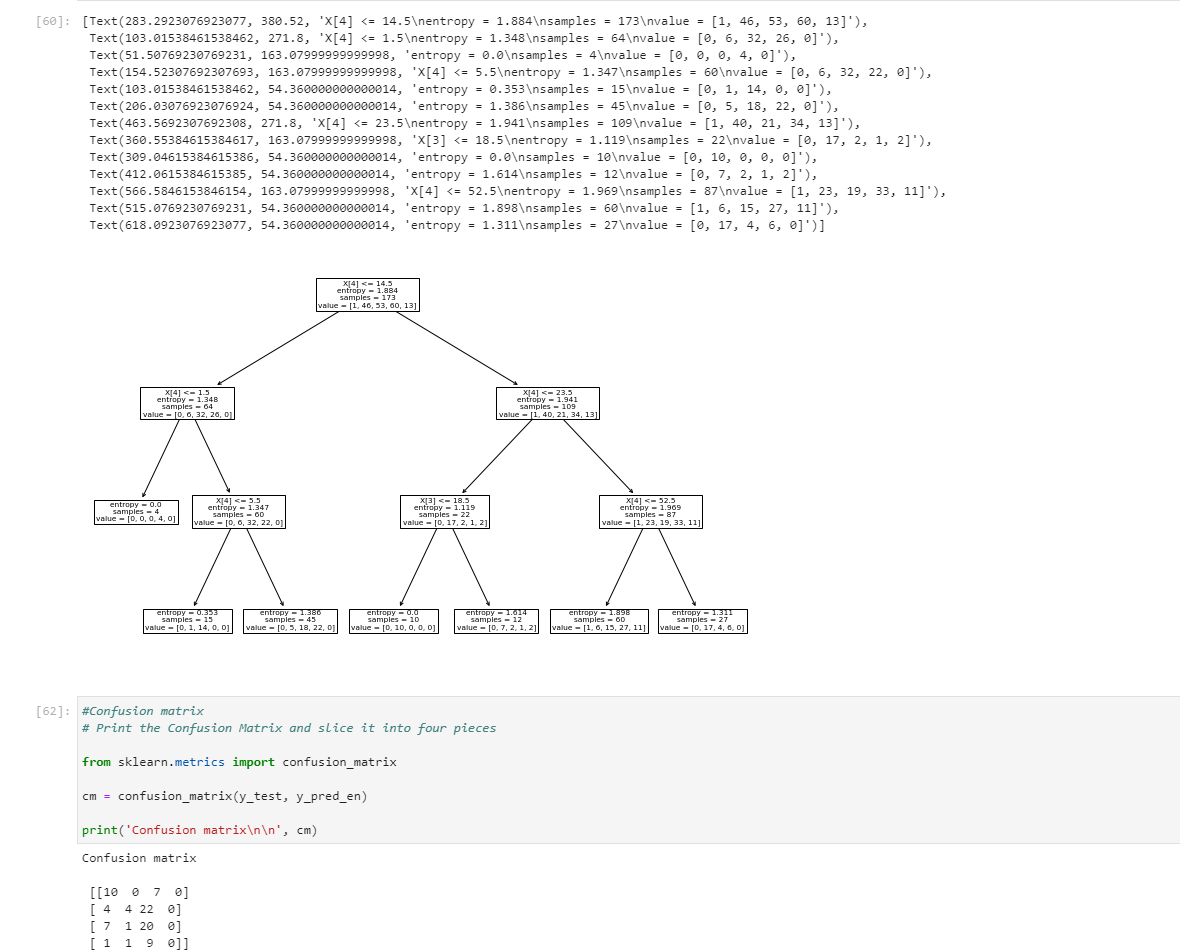


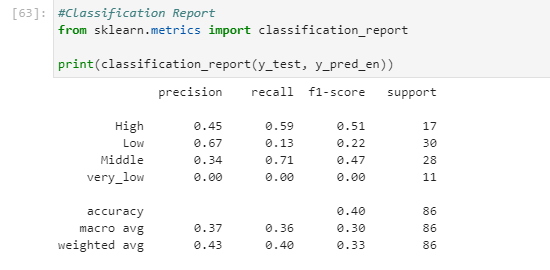




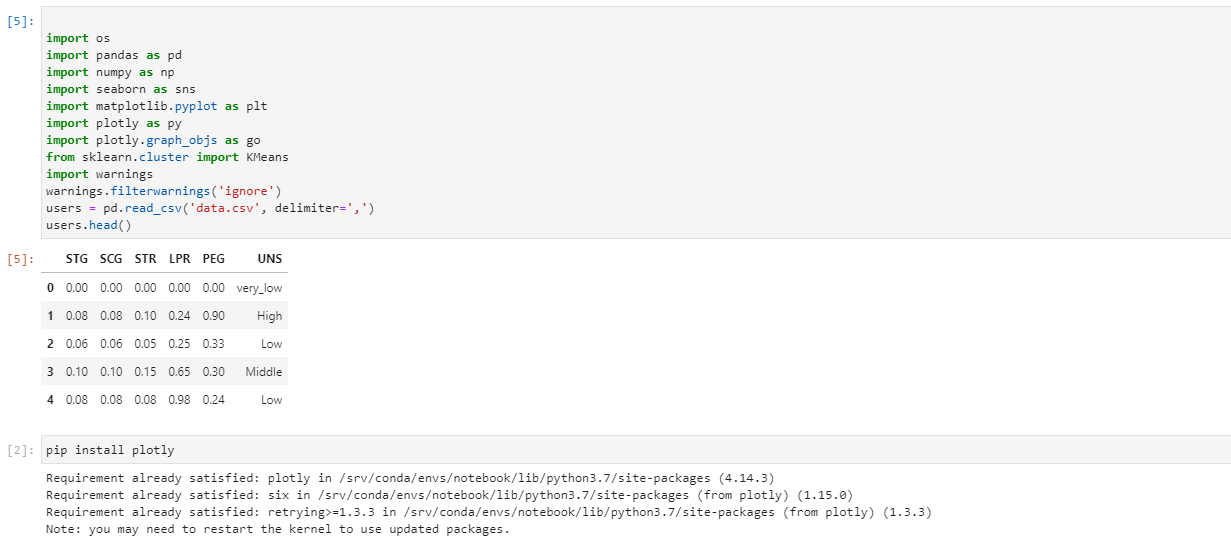


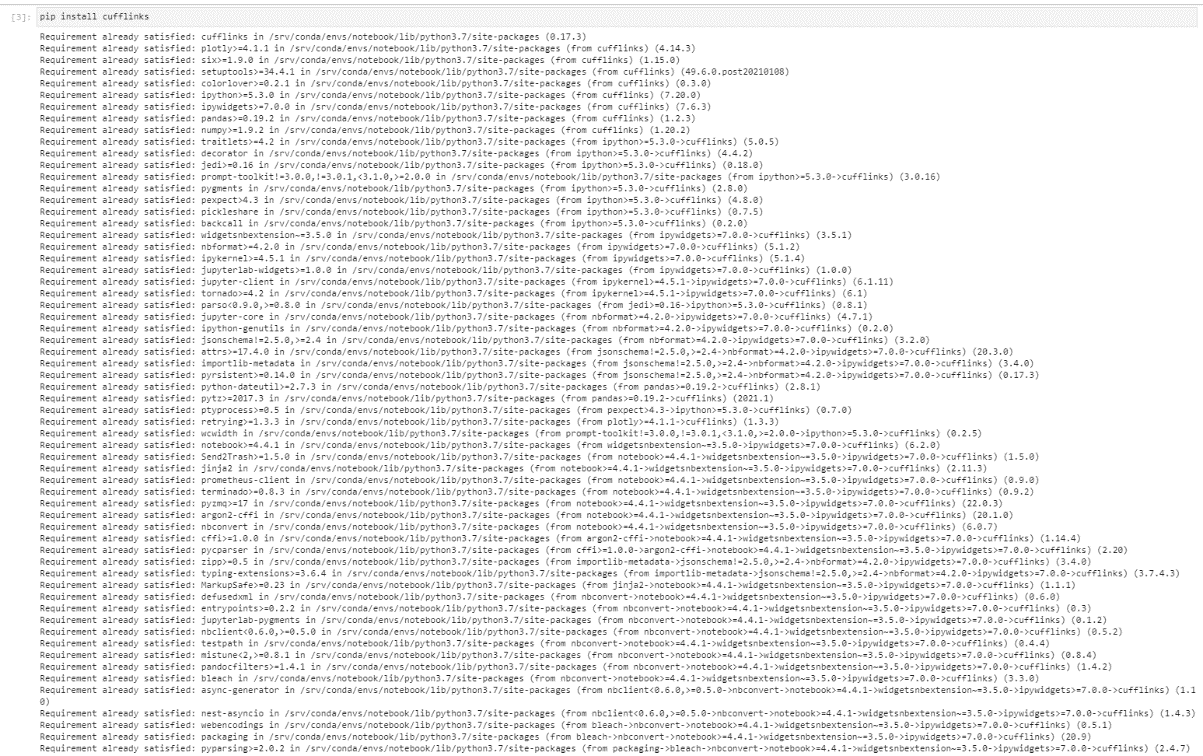


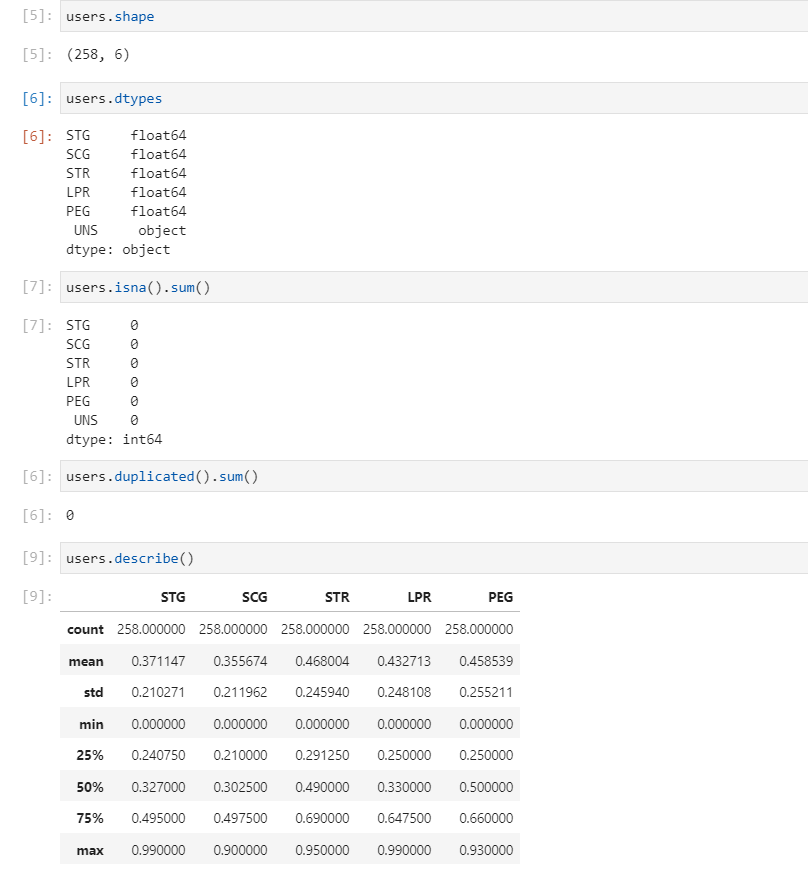


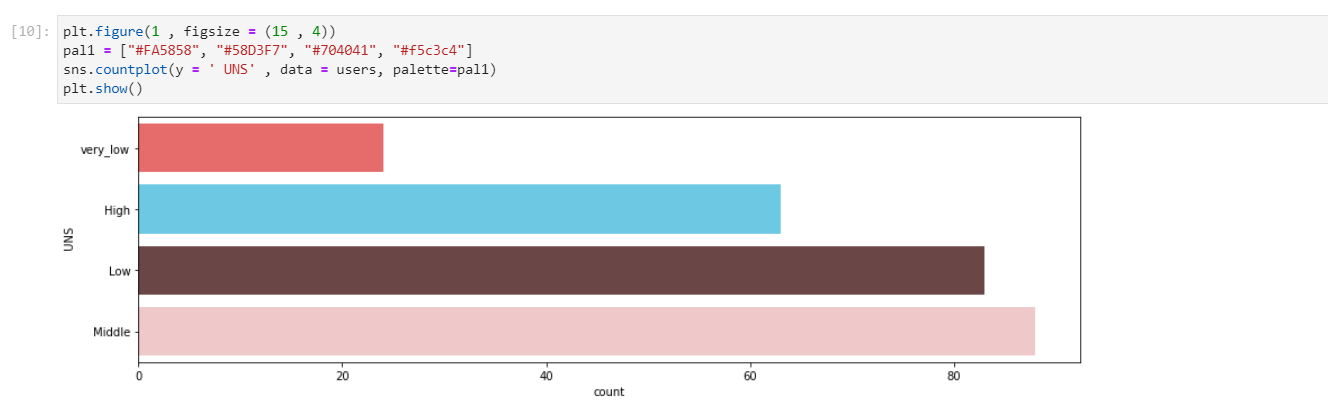


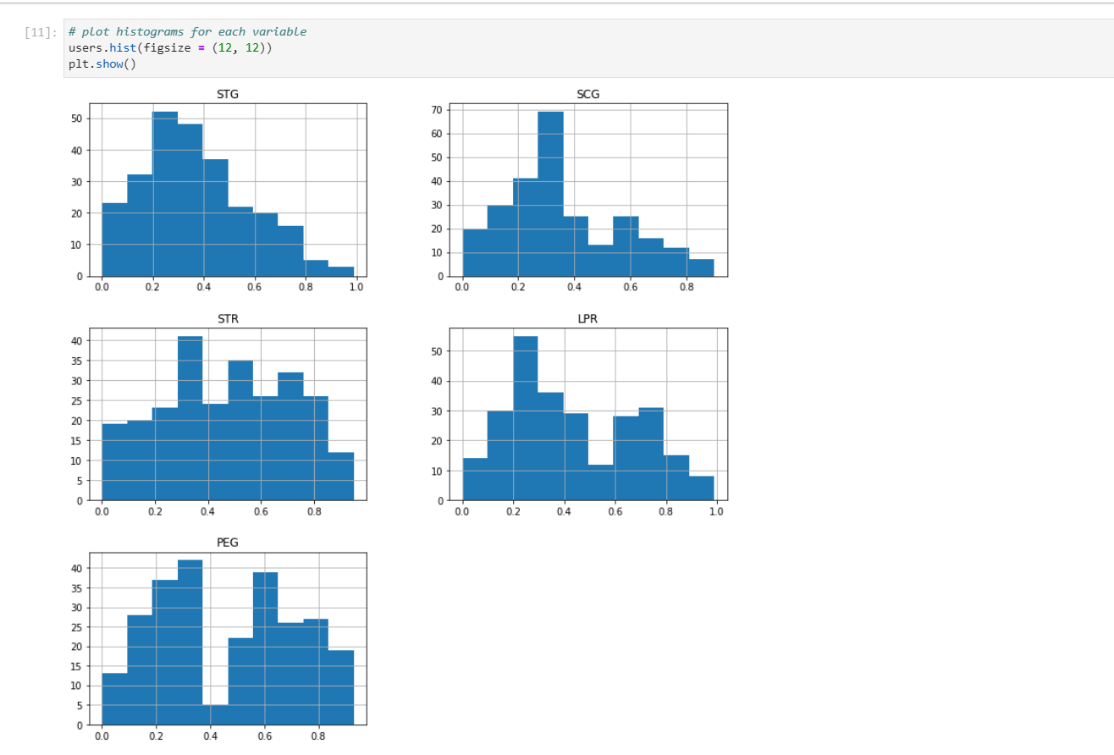
**Hierarchical clustering:**

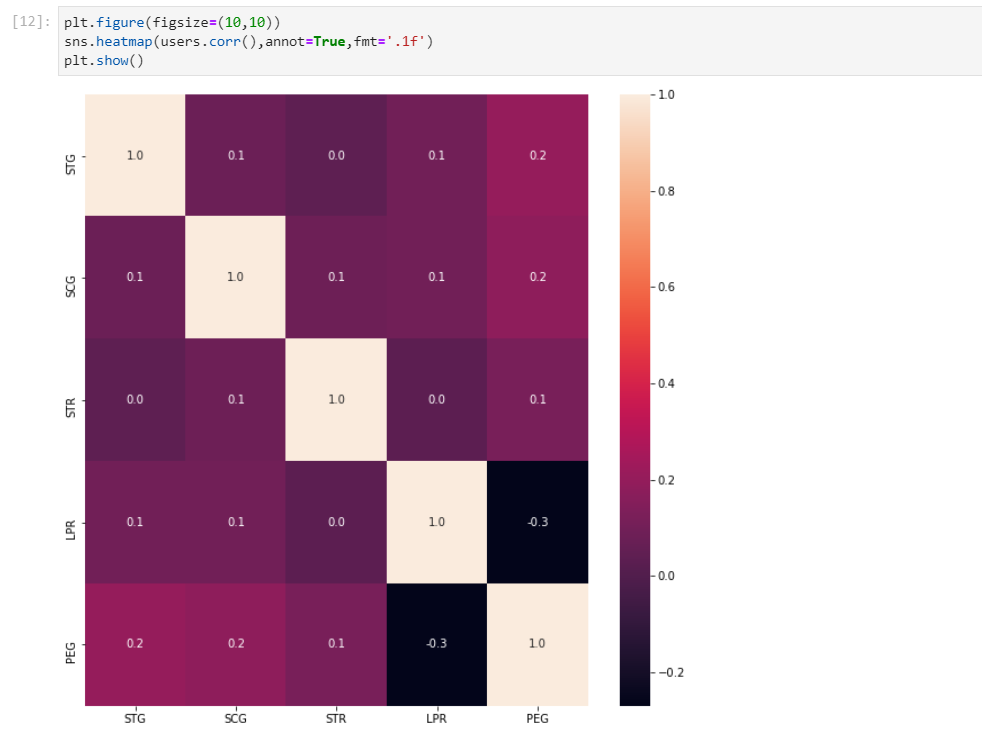


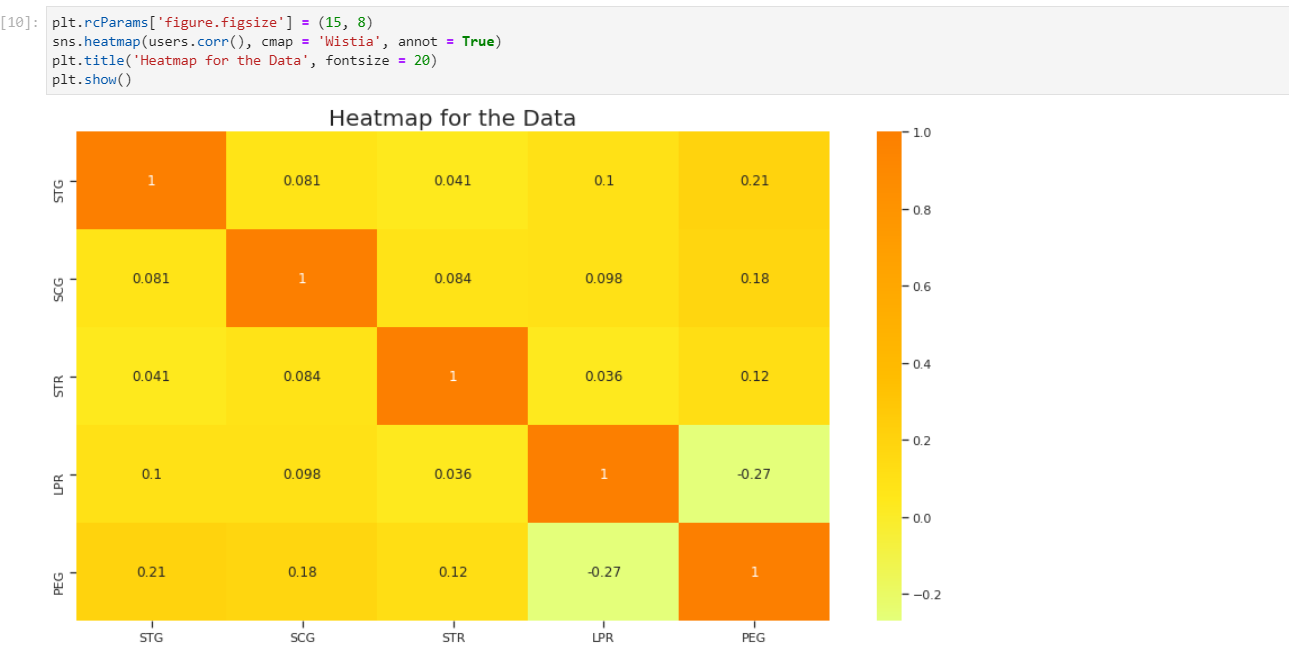


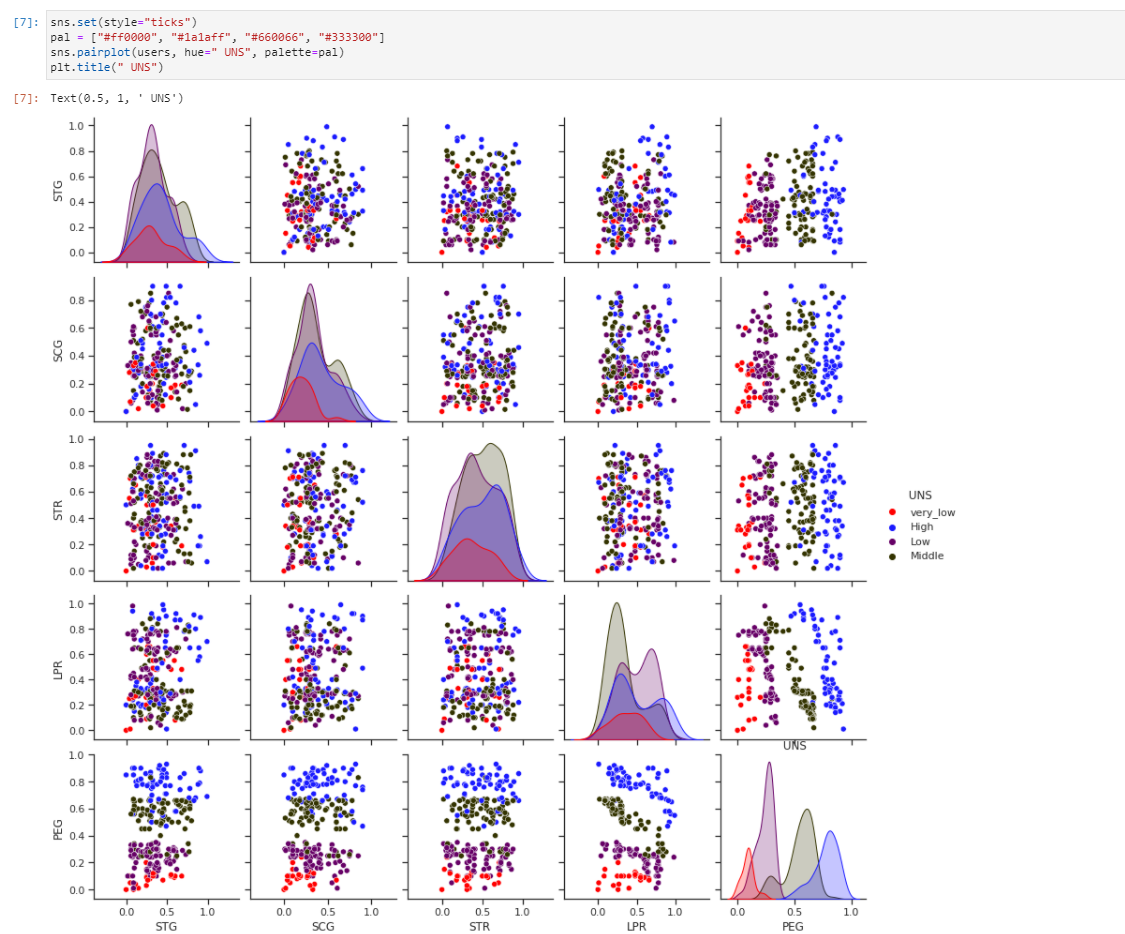


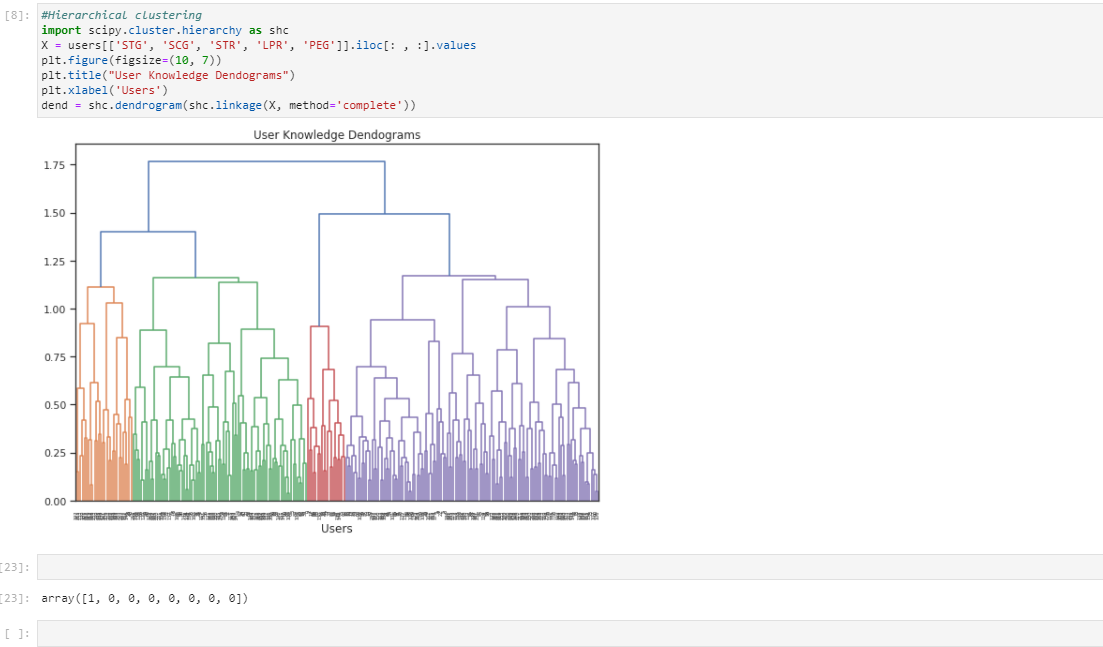




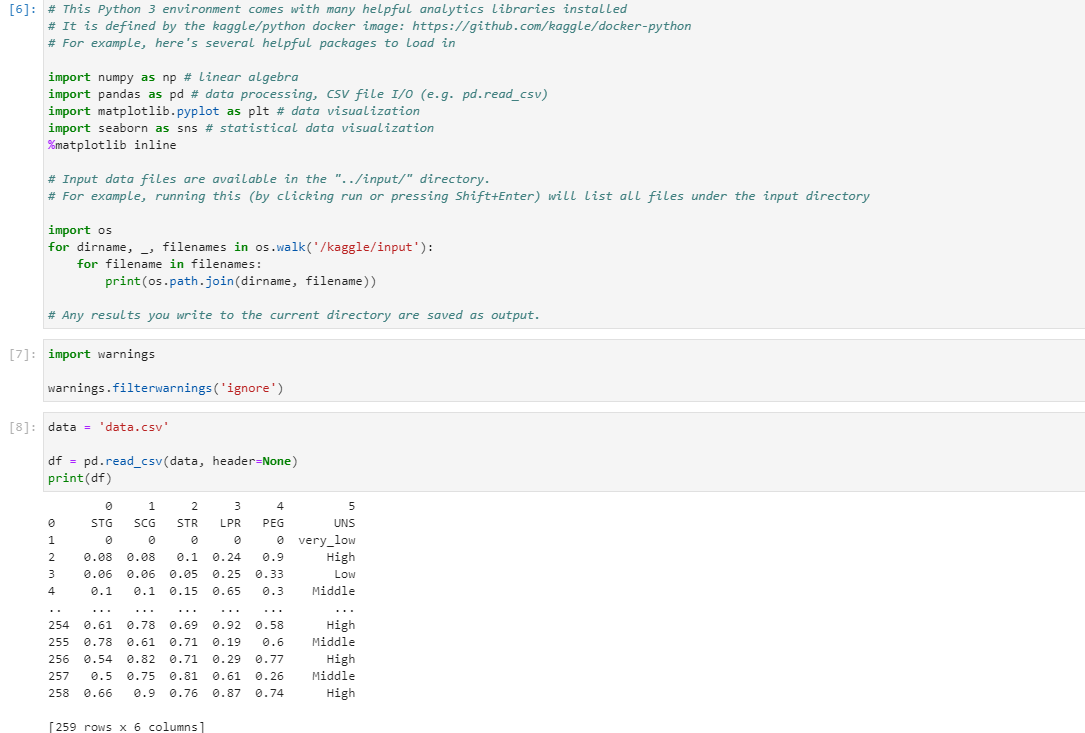


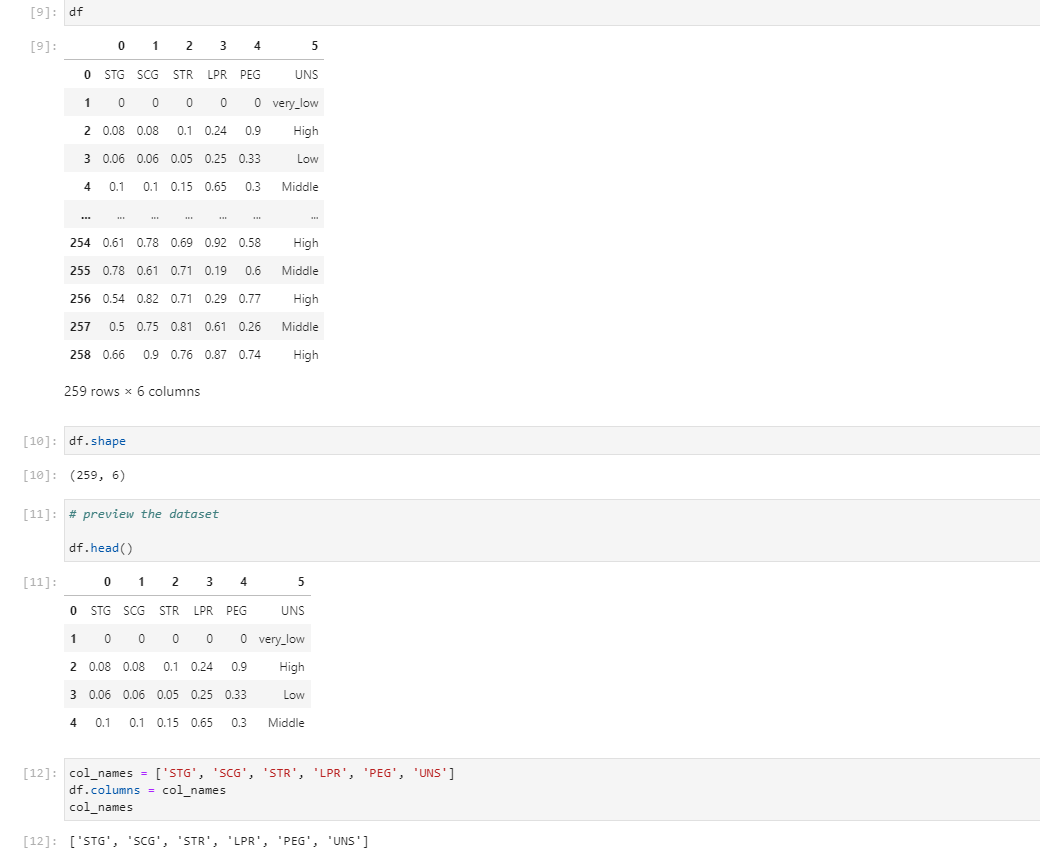


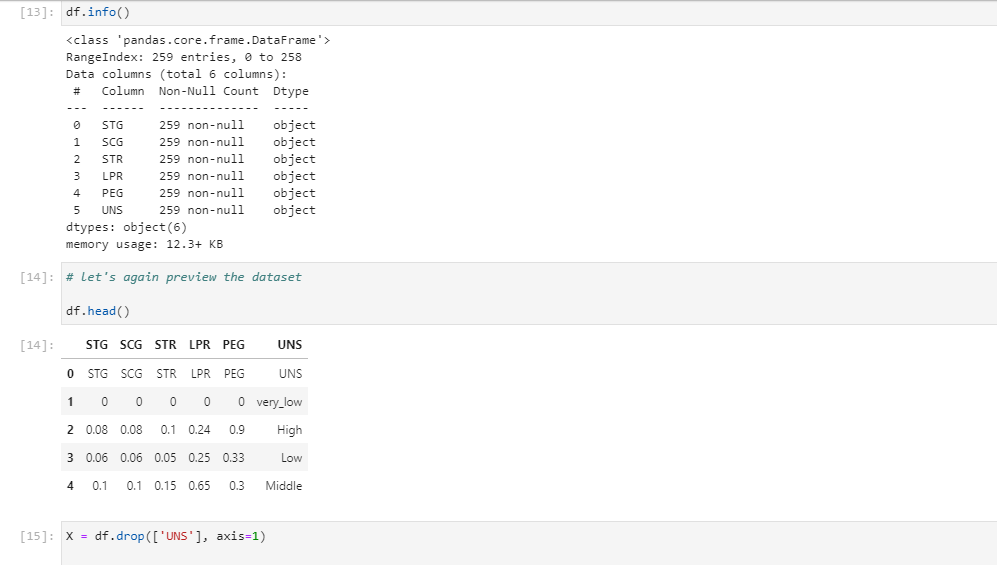


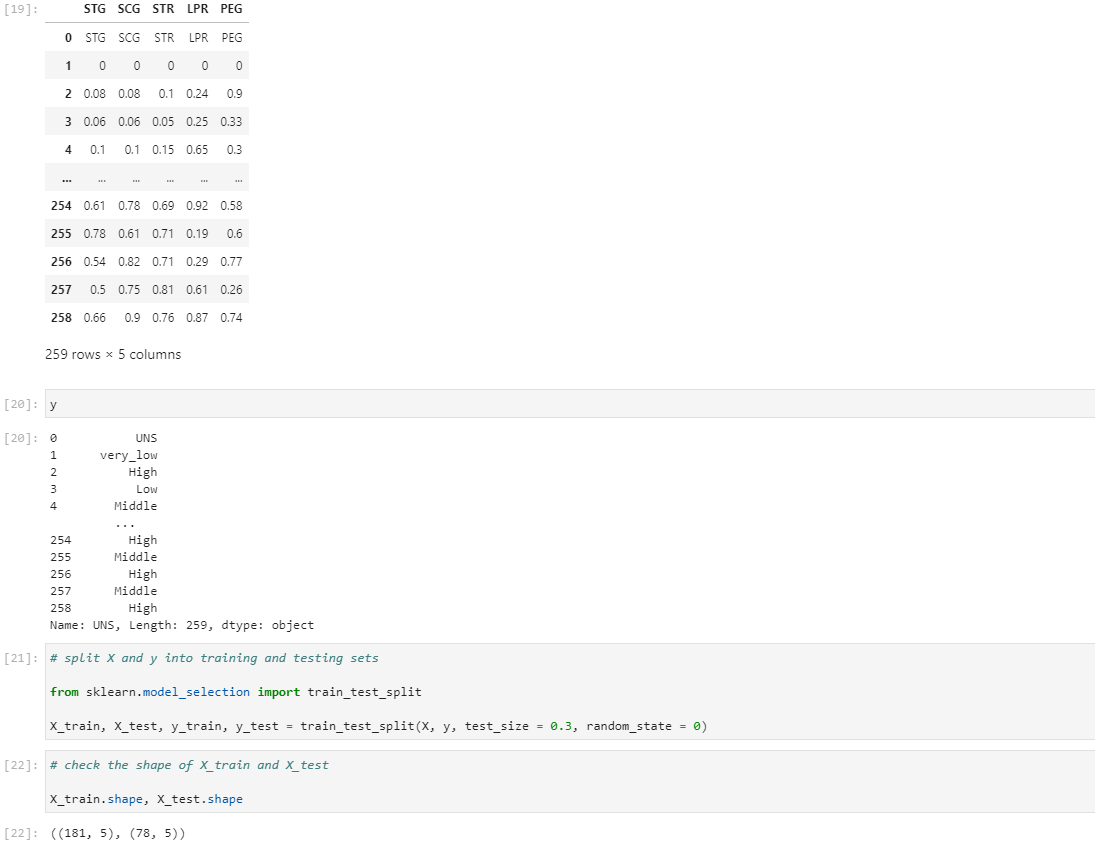


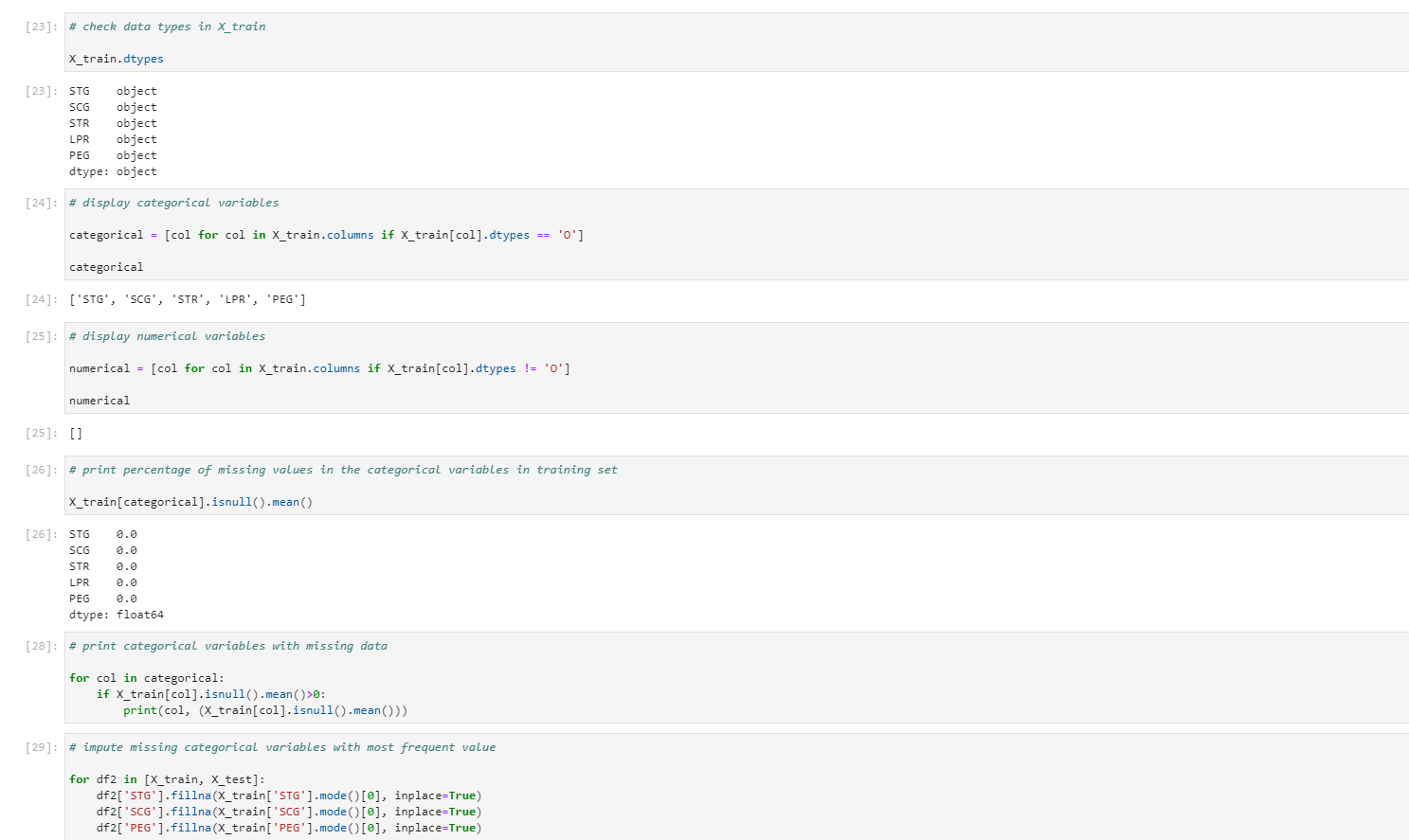
**Naïve Bayes classification:**

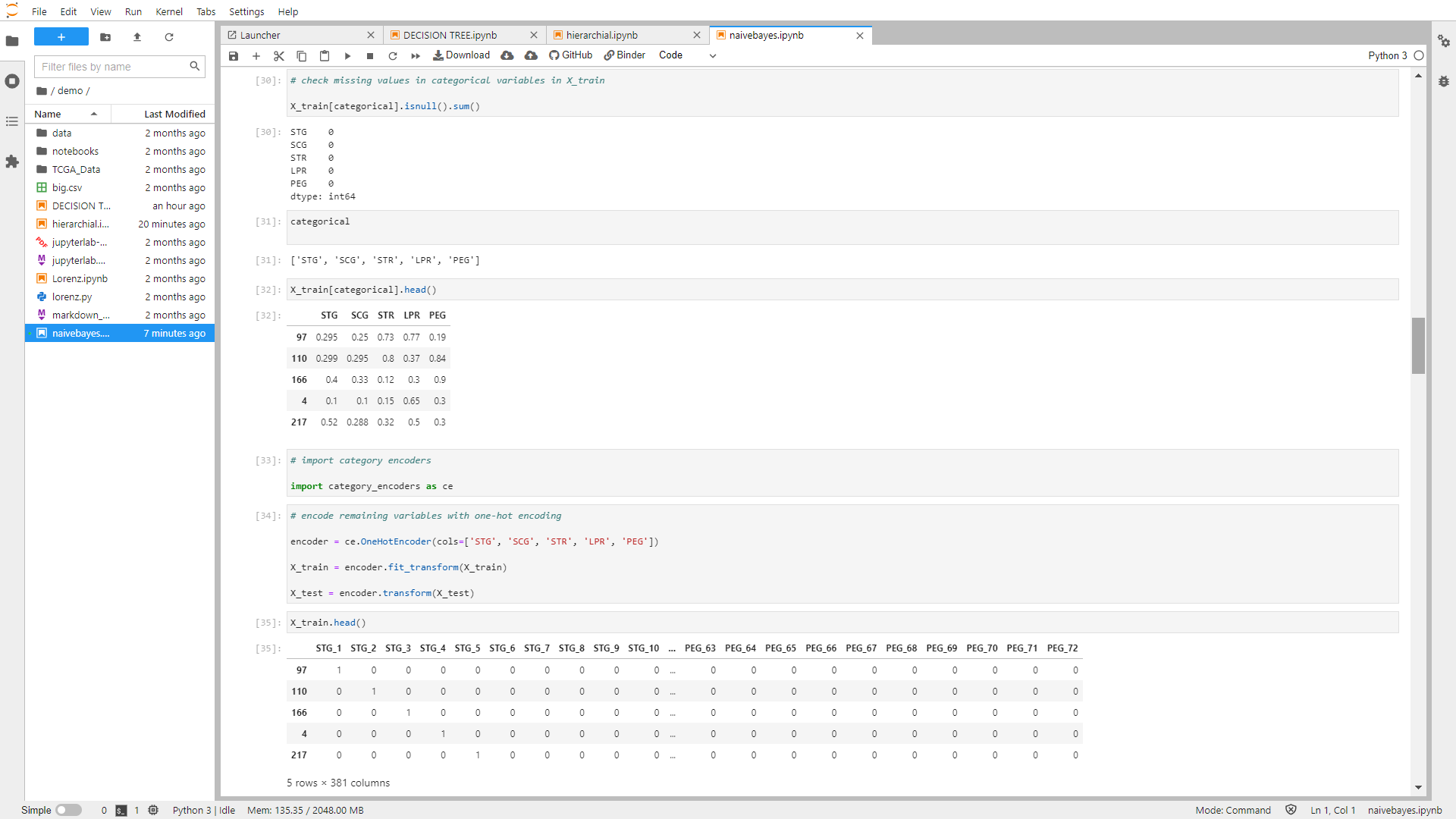


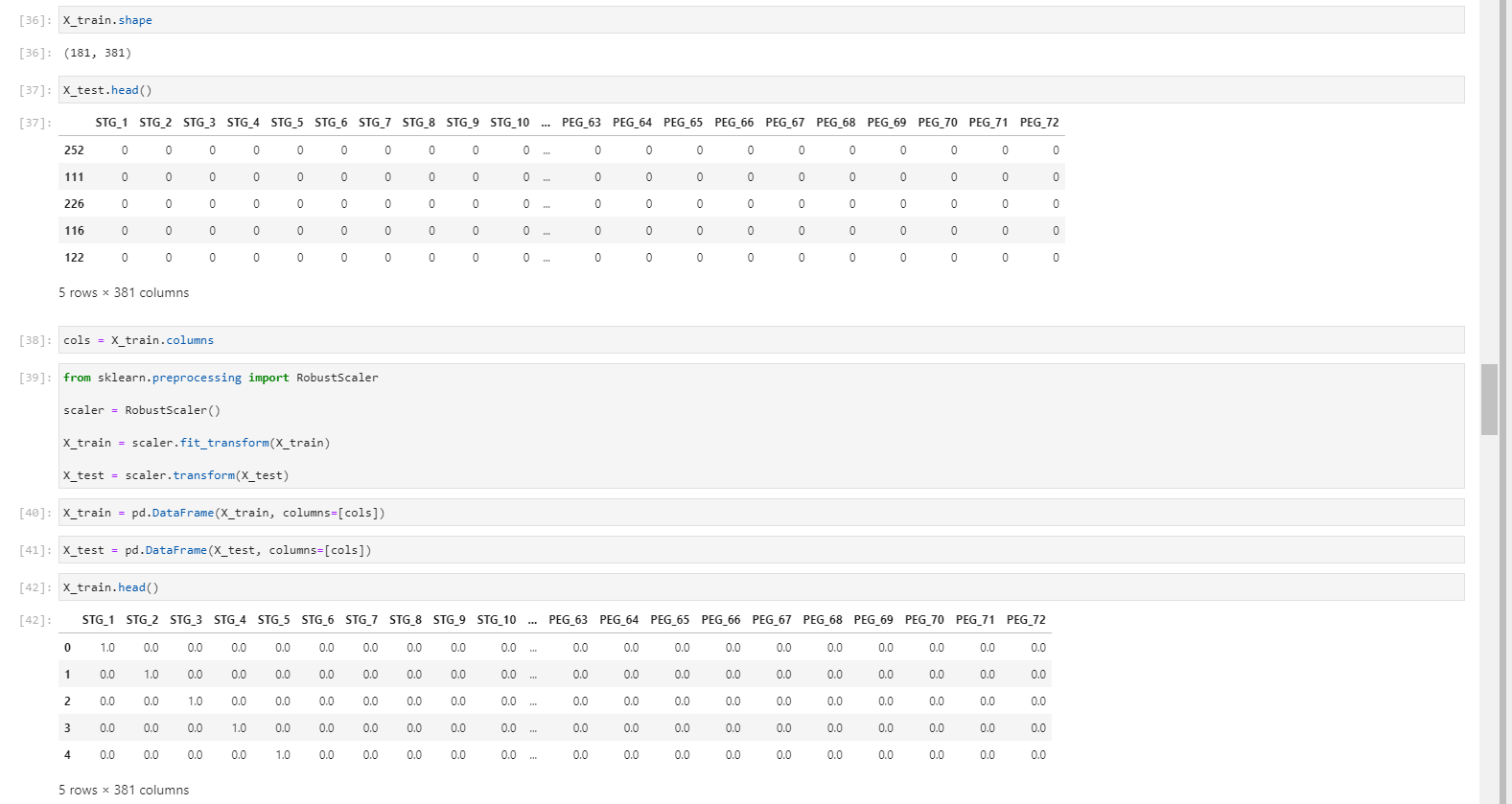




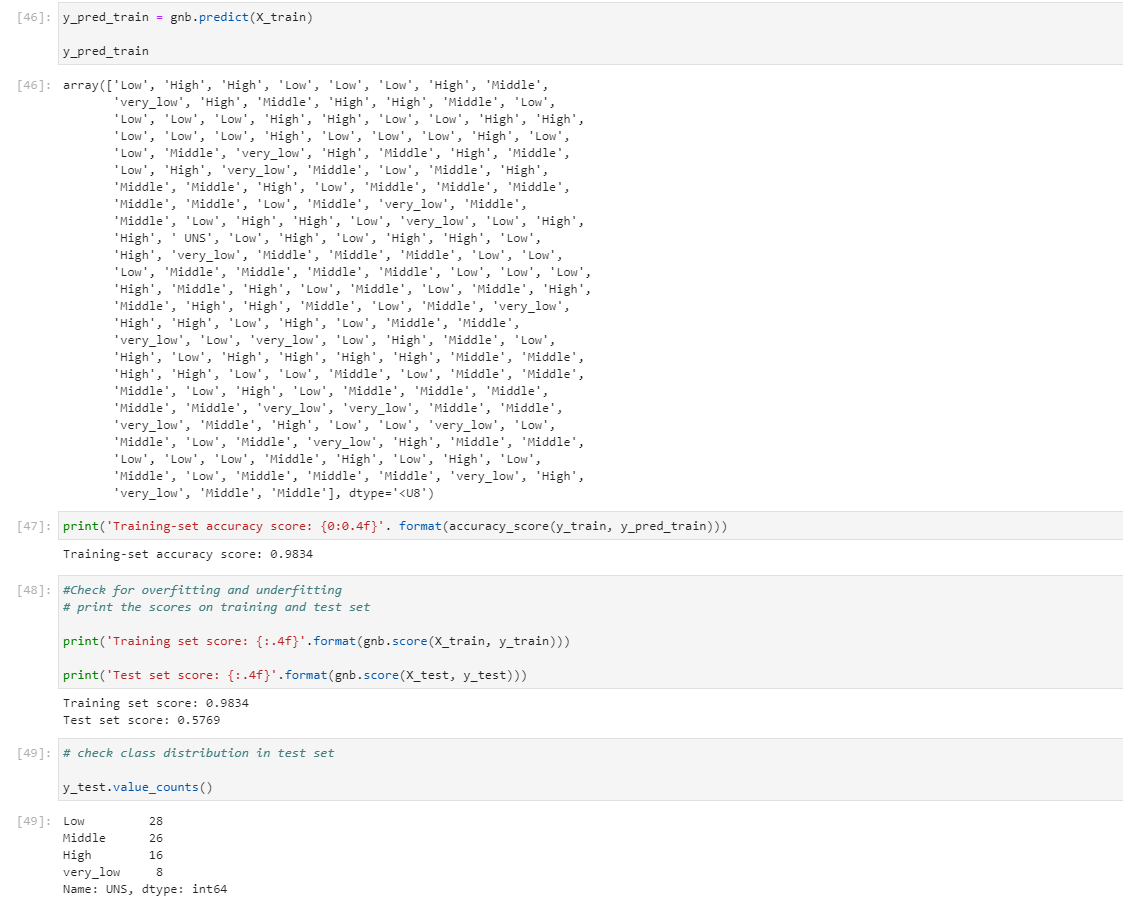




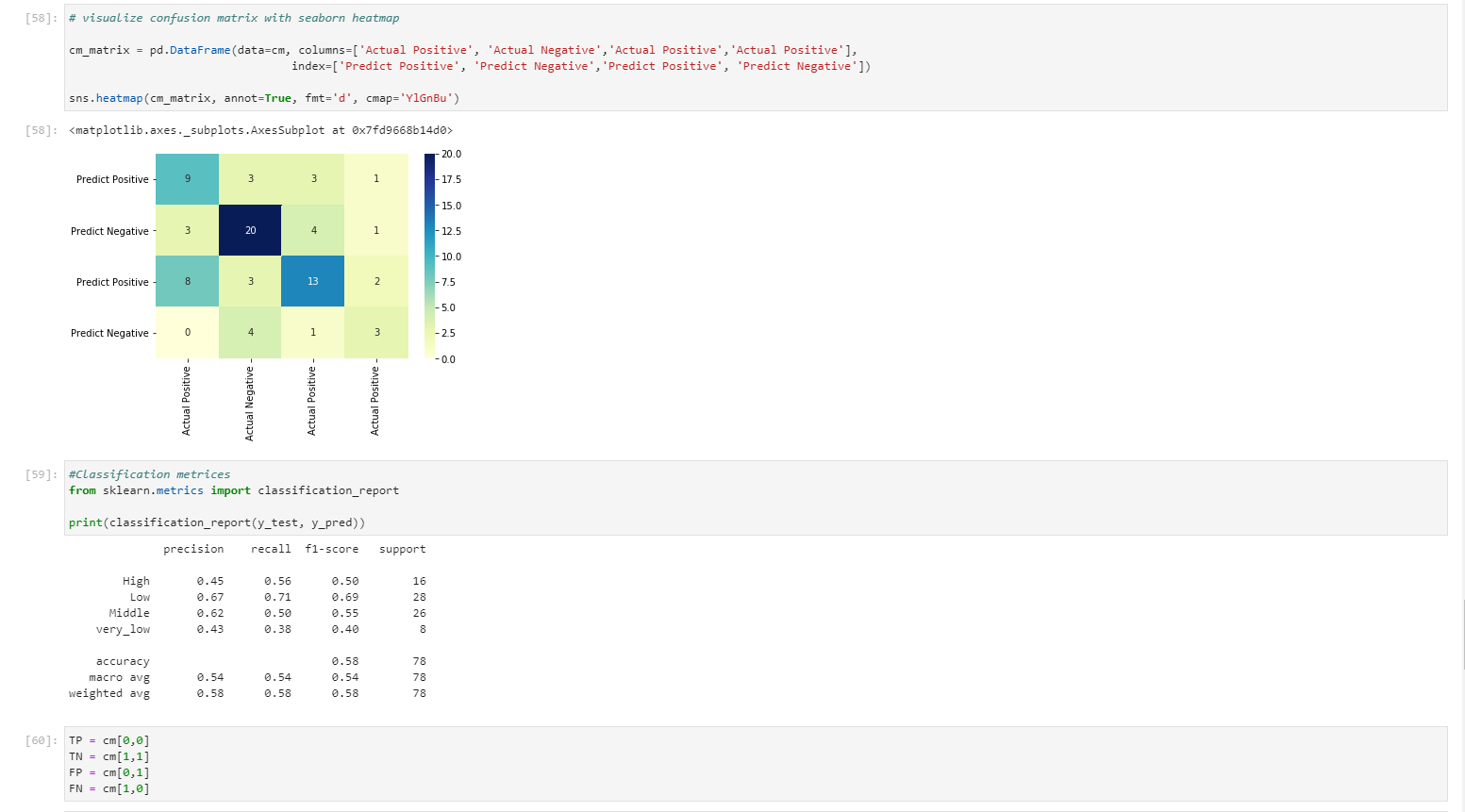


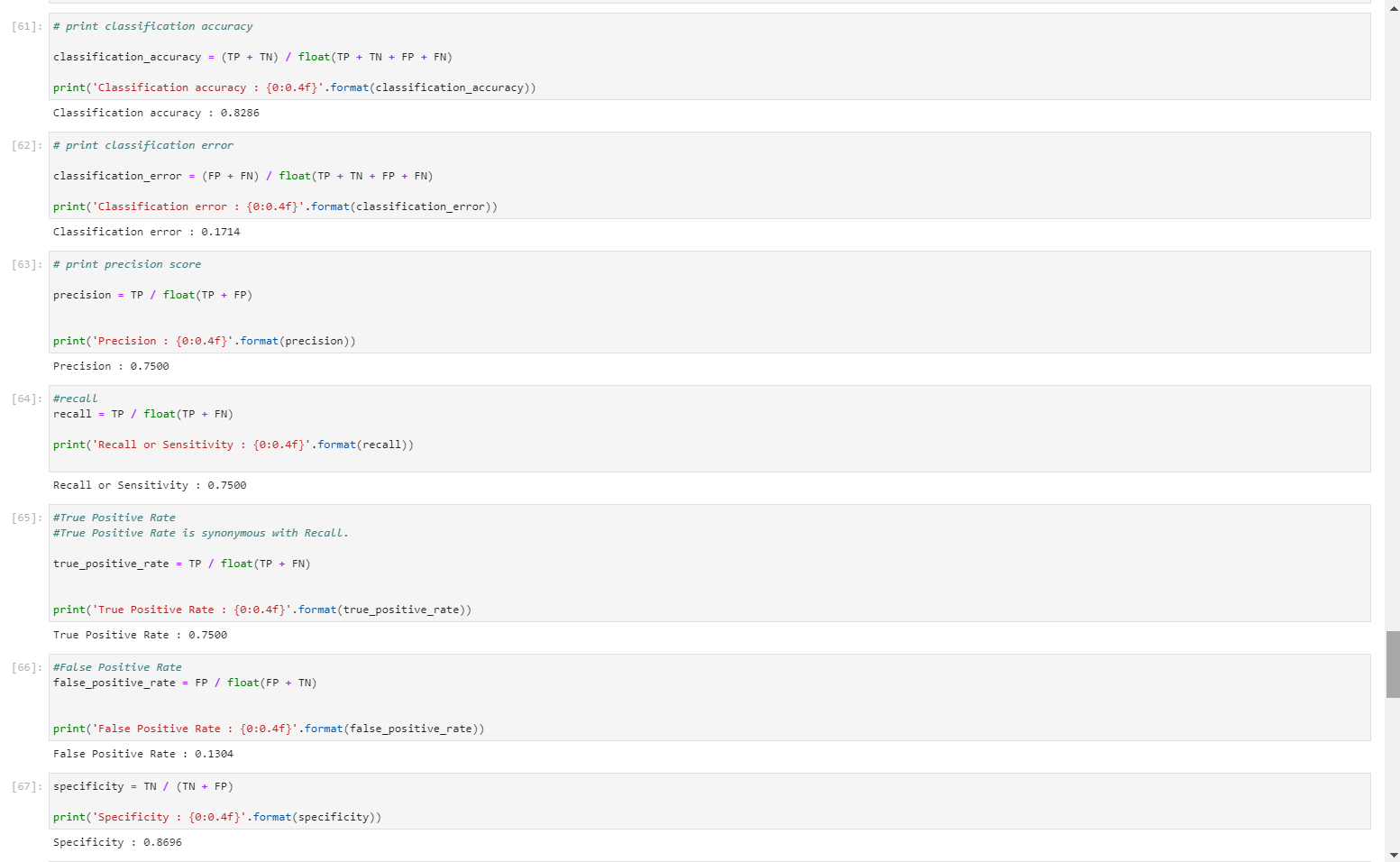




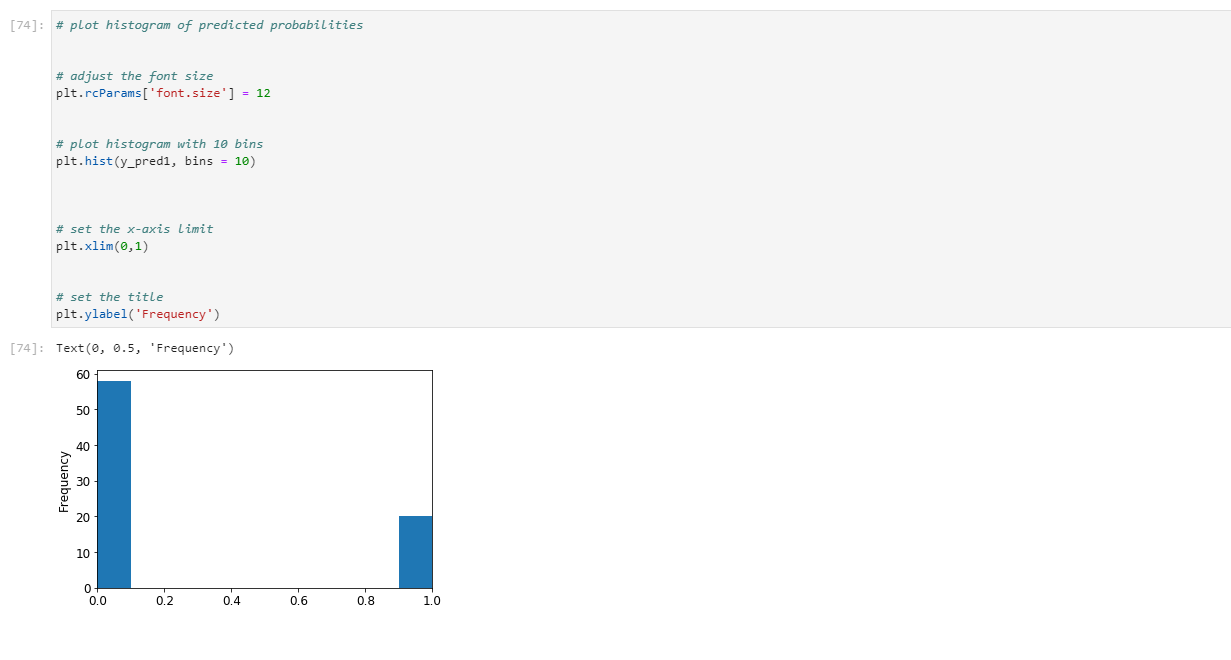












**Conclusion:**

We have used the Naïve Bayes Classification and Decision Tree algorithm for classifying the dataset and concluded that Naïve Bayes Classifier is the best algorithm for classifying this User level Knowledge dataset. The accuracy we got is 90.31%. We have used the K-means clustering and Hierarchical clustering algorithm for grouping the dataset and concluded that Hierarchical clustering algorithm is the best algorithm for clustering this User level Knowledge dataset. The accuracy we got is 32.41%.