**1. Overview**

Columns:

* **RI:**refractive index
* **NA:**Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
* **Mg:**Magnesium
* **Al:**Aluminum
* **K:**Potassium
* **Ca:**Calcium
* **Ba:**Barium
* **Fe:**Iron
* **Type of glass:**1 building\_windows\_float\_processed -- 2 building\_windows\_non\_float\_processed -- 3 vehicle\_windows\_float\_processed -- 4 vehicle\_windows\_non\_float\_processed (none in this database) -- 5 containers -- 6 tableware -- 7 headlamps

-🡪 <https://www.kaggle.com/tolgahancepel/glass-classification-analysis-with-eda/comments#607012>

**2. Importing Libraries and Reading the Dataset**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import cross\_val\_score

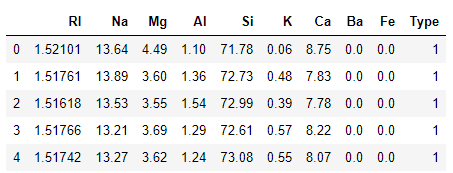
from collections import Counter

from IPython.core.display import display, HTML

sns.set\_style('darkgrid')

dataset = pd.read\_csv('C:/Users/Adinath/Desktop/Data\_Science/Assignments/KNN/glass.csv')

dataset.head()



## ****3. Data Visualization and Preprocessing****

corr = dataset.corr()

#Plot figsize

fig, ax = plt.subplots(figsize=(10, 8))

#Generate Heat Map, allow annotations and place floats in map

sns.heatmap(corr, cmap='coolwarm', annot=True, fmt=".2f")

#Apply xticks

plt.xticks(range(len(corr.columns)), corr.columns);

#Apply yticks

plt.yticks(range(len(corr.columns)), corr.columns)

#show plot

plt.show()

We then create a variable fig, and set it equal to, plt.figure(figsize=(6,3))

This creates a figure object, which has a width of 6 inches and 3 inches in height.

The values of the figsize attribute are a tuple of 2 values.

The width is the first parameter of the figsize attribute and the height is the second parameter. So it is, figsize(weight,height)

### What is a heatmap?

A heatmap is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors. The seaborn python package allows the creation of annotated heatmaps which can be tweaked using Matplotlib tools as per the creator’s requirement.

we create the heatmap using the heatmap function from the seaborn python package. The heatmap function takes the following arguments:

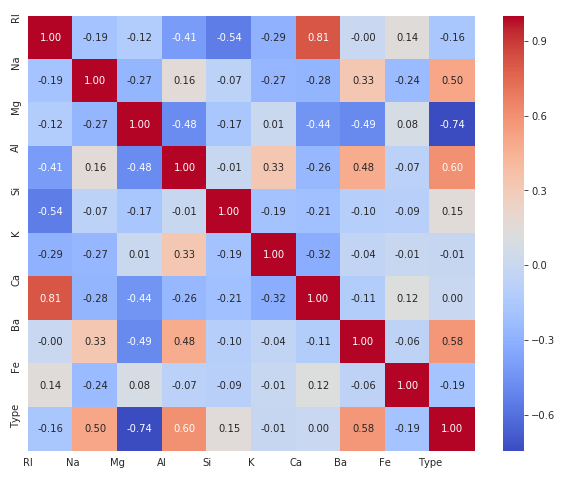
**data**– 2D dataset that can be coerced into an ndarray. If a Pandas DataFrame is provided, the index/column information will be used to label the columns and rows.

**annot** – an array of same shape as data which is used to annotate the heatmap.

**cmap** – a matplotlib colormap name or object. This maps the data values to the color space.

**fmt** – string formatting code to use when adding annotations.

**linewidths** – sets the width of the lines that will divide each cell.



f, axes = plt.subplots(1,2,figsize=(14,4))

sns.distplot(dataset['RI'], ax = axes[0])

axes[0].set\_xlabel('Refractive Index', fontsize=14)

axes[0].set\_ylabel('Count', fontsize=14)

axes[0].yaxis.tick\_left()

sns.violinplot(x = 'Type', y = 'RI', data = dataset, hue = 'Type', dodge = False, ax = axes[1])

axes[1].set\_xlabel('Type of glass', fontsize=14)

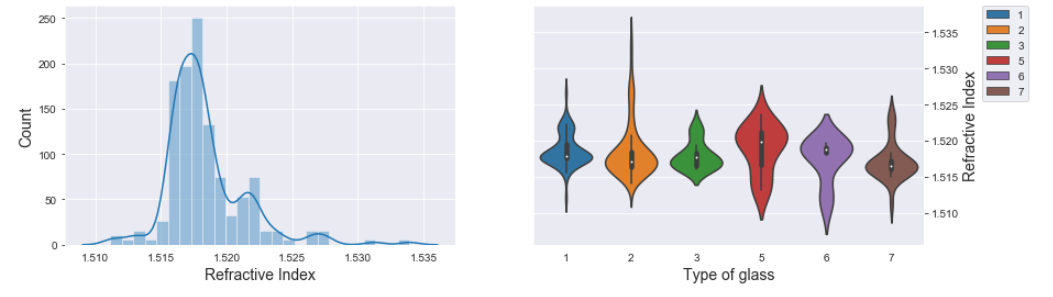
axes[1].set\_ylabel('Refractive Index', fontsize=14)

axes[1].yaxis.set\_label\_position("right")

axes[1].yaxis.tick\_right()

axes[1].legend(bbox\_to\_anchor=(1.15, 1), loc=2, borderaxespad=0.)

plt.show()



f, axes = plt.subplots(1,2,figsize=(14,4))

sns.distplot(dataset['Na'], ax = axes[0])

axes[0].set\_xlabel('Sodium', fontsize=14)

axes[0].set\_ylabel('Count', fontsize=14)

axes[0].yaxis.tick\_left()

sns.boxplot(x = 'Type', y = 'Na', data = dataset, hue = 'Type', dodge = False, ax = axes[1])

axes[1].set\_xlabel('Type of glass', fontsize=14)

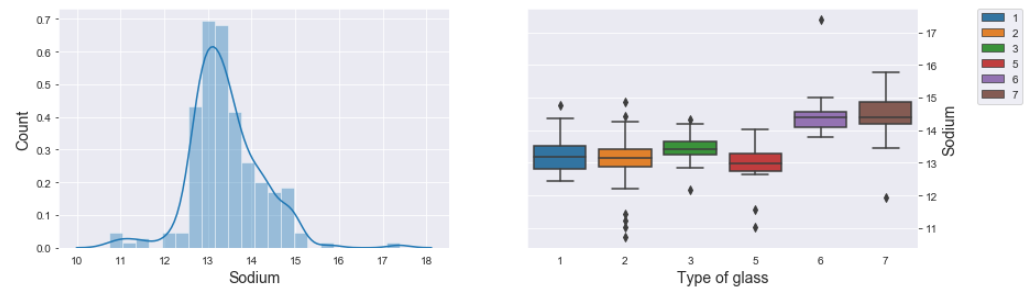
axes[1].set\_ylabel('Sodium', fontsize=14)

axes[1].yaxis.set\_label\_position("right")

axes[1].yaxis.tick\_right()

axes[1].legend(bbox\_to\_anchor=(1.15, 1), loc=2, borderaxespad=0.)

plt.show()



f, axes = plt.subplots(1,2,figsize=(14,4))

sns.distplot(dataset['Mg'], ax = axes[0])

axes[0].set\_xlabel('Magnesium', fontsize=14)

axes[0].set\_ylabel('Count', fontsize=14)

axes[0].yaxis.tick\_left()

sns.violinplot(x = 'Type', y = 'Mg', data = dataset, hue = 'Type', dodge = False, ax = axes[1])

axes[1].set\_xlabel('Type of glass', fontsize=14)

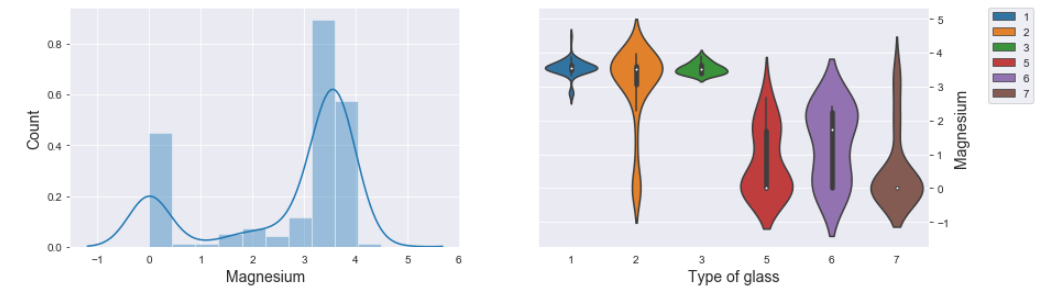
axes[1].set\_ylabel('Magnesium', fontsize=14)

axes[1].yaxis.set\_label\_position("right")

axes[1].yaxis.tick\_right()

axes[1].legend(bbox\_to\_anchor=(1.15, 1), loc=2, borderaxespad=0.)

plt.show()



X = dataset.drop('Type', axis = 1).values

y = dataset['Type'].values.reshape(-1,1)

# Splitting the dataset into the Training set and Test set

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

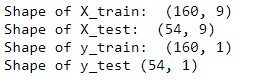
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 42)

print("Shape of X\_train: ",X\_train.shape)

print("Shape of X\_test: ", X\_test.shape)

print("Shape of y\_train: ",y\_train.shape)

print("Shape of y\_test",y\_test.shape)



## ****4. Classification Models****

### ****Logistic Regression****

*# Fitting Logistic Regression to the Training set*

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import GridSearchCV

from sklearn.linear\_model import LogisticRegression

classifier\_lr = LogisticRegression()

steps = [

('scalar', StandardScaler()),

('model', LogisticRegression())

]

lr\_pipe = Pipeline(steps)

parameters = { 'model\_\_C' : [1,10,100,1000,10000],

'model\_\_fit\_intercept' : [True],

'model\_\_multi\_class' : ['auto'],

'model\_\_tol' : [0.0001],

'model\_\_solver' : ['newton-cg', 'lbfgs', 'sag', 'saga'],

'model\_\_n\_jobs' : [-1],

'model\_\_max\_iter' : [5000],

'model\_\_random\_state': [42]

}

classifier\_lr = GridSearchCV(lr\_pipe, parameters, iid=False, cv = 3)

classifier\_lr = classifier\_lr.fit(X\_train, y\_train.ravel())

from sklearn.metrics import accuracy\_score

y\_pred\_lr\_train = classifier\_lr.predict(X\_train)

accuracy\_lr\_train = accuracy\_score(y\_train, y\_pred\_lr\_train)

print("Training set: ", accuracy\_lr\_train)

y\_pred\_lr\_test = classifier\_lr.predict(X\_test)

accuracy\_lr\_test = accuracy\_score(y\_test, y\_pred\_lr\_test)

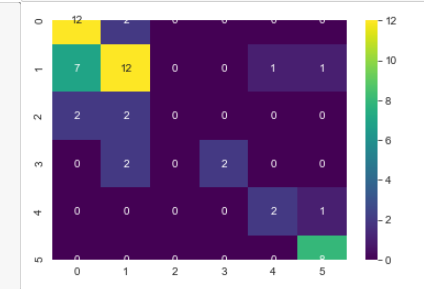
print("Test set: ", accuracy\_lr\_test)



from sklearn.metrics import confusion\_matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_lr\_test), annot=True, cmap = 'viridis', fmt='.0f')

plt.show()



### ****K-Nearest Neighbors (K-NN)****

*# Fitting classifier to the Training set*

from sklearn.neighbors import KNeighborsClassifier

classifier\_knn = KNeighborsClassifier()

steps = [

('scalar', StandardScaler()),

('model', KNeighborsClassifier())

]

knn\_pipe = Pipeline(steps)

parameters = { 'model\_\_algorithm' : ['brute'],

'model\_\_leaf\_size' : [30,50,70,90,110],

'model\_\_metric' : ['minkowski'],

'model\_\_p' : [1],

'model\_\_n\_neighbors' : [3,5,11,19],

'model\_\_weights' : ['uniform', 'distance'],

'model\_\_n\_jobs' : [-1]

}

classifier\_knn = GridSearchCV(knn\_pipe, parameters, iid=False, cv = 3)

classifier\_knn = classifier\_knn.fit(X\_train, y\_train.ravel())

y\_pred\_knn\_train = classifier\_knn.predict(X\_train)

accuracy\_knn\_train = accuracy\_score(y\_train, y\_pred\_knn\_train)

print("Training set: ", accuracy\_knn\_train)

y\_pred\_knn\_test = classifier\_knn.predict(X\_test)

accuracy\_knn\_test = accuracy\_score(y\_test, y\_pred\_knn\_test)

print("Test set: ", accuracy\_knn\_test)

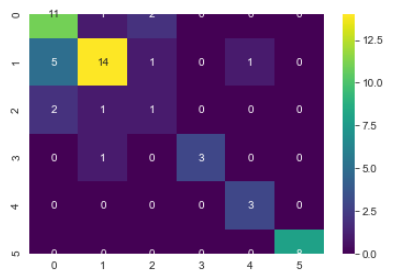
Training set: 1.0

Test set: 0.7407407407407407

from sklearn.metrics import confusion\_matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_knn\_test), annot=True, cmap = 'viridis', fmt='.0f')

plt.show()



### ****Support Vector Machine (SVM - Linear)****

from sklearn.svm import SVC

classifier\_svm = SVC()

steps = [

('scalar', StandardScaler()),

('model', SVC())

]

svm\_linear\_pipe = Pipeline(steps)

parameters = { 'model\_\_kernel' : ['linear'],

'model\_\_C' : [1,10,100,1000,10000],

'model\_\_random\_state' : [42]

}

classifier\_svm\_linear = GridSearchCV(svm\_linear\_pipe, parameters, iid=False, cv = 3)

classifier\_svm\_linear = classifier\_svm\_linear.fit(X\_train, y\_train.ravel())

y\_pred\_svm\_linear\_train = classifier\_svm\_linear.predict(X\_train)

accuracy\_svm\_linear\_train = accuracy\_score(y\_train, y\_pred\_svm\_linear\_train)

print("Training set: ", accuracy\_svm\_linear\_train)

y\_pred\_svm\_linear\_test = classifier\_svm\_linear.predict(X\_test)

accuracy\_svm\_linear\_test = accuracy\_score(y\_test, y\_pred\_svm\_linear\_test)

print("Test set: ", accuracy\_svm\_linear\_test)

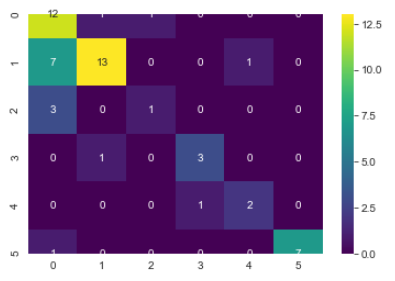
Training set: 0.75

Test set: 0.7037037037037037

from sklearn.metrics import confusion\_matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_svm\_linear\_test), annot=True, cmap = 'viridis', fmt='.0f')

plt.show()



### ****Support Vector Machine (SVM - Kernel)****

# Fitting classifier to the Training set

from sklearn.svm import SVC

classifier\_svm\_kernel = SVC()

steps = [

('scalar', StandardScaler()),

('model', SVC())

]

svm\_kernel\_pipe = Pipeline(steps)

parameters = { 'model\_\_kernel' : ['rbf', 'poly', 'sigmoid'],

'model\_\_C' : [1,10,100,1000,10000],

'model\_\_gamma' : [0.001, 0.01, 0.1, 1, 'scale'],

'model\_\_random\_state' : [42],

'model\_\_degree' : [1,2,3]

}

classifier\_svm\_kernel = GridSearchCV(svm\_kernel\_pipe, parameters, iid=False, cv = 3)

classifier\_svm\_kernel = classifier\_svm\_kernel.fit(X\_train, y\_train.ravel())

y\_pred\_svm\_kernel\_train = classifier\_svm\_kernel.predict(X\_train)

accuracy\_svm\_kernel\_train = accuracy\_score(y\_train, y\_pred\_svm\_kernel\_train)

print("Training set: ", accuracy\_svm\_kernel\_train)

y\_pred\_svm\_kernel\_test = classifier\_svm\_kernel.predict(X\_test)

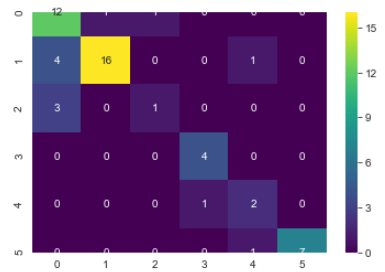
accuracy\_svm\_kernel\_test = accuracy\_score(y\_test, y\_pred\_svm\_kernel\_test)

print("Test set: ", accuracy\_svm\_kernel\_test)

from sklearn.metrics import confusion\_matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_svm\_kernel\_test), annot=True, cmap = 'viridis', fmt='.0f')

plt.show()



### ****Naive Bayes****

*# Fitting classifier to the Training set*

from sklearn.naive\_bayes import GaussianNB

classifier\_nb = GaussianNB()

classifier\_nb.fit(X\_train, y\_train.ravel())

GaussianNB(priors=None, var\_smoothing=1e-09)

y\_pred\_nb\_train = classifier\_nb.predict(X\_train)

accuracy\_nb\_train = accuracy\_score(y\_train, y\_pred\_nb\_train)

print("Training set: ", accuracy\_nb\_train)

y\_pred\_nb\_test = classifier\_nb.predict(X\_test)

accuracy\_nb\_test = accuracy\_score(y\_test, y\_pred\_nb\_test)

print("Test set: ", accuracy\_nb\_test)

Training set: 0.3125

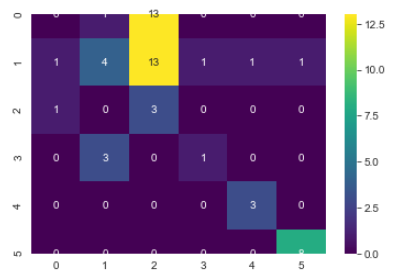
Training set: 0.3125

Test set: 0.35185185185185186

from sklearn.metrics import confusion\_matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_nb\_test), annot=True, cmap = 'viridis', fmt='.0f')

plt.show()



### ****Decision Tree Classification****

*# Fitting classifier to the Training set*

from sklearn.tree import DecisionTreeClassifier

classifier\_dt = DecisionTreeClassifier()

steps = [

('scalar', StandardScaler()),

('model', DecisionTreeClassifier())

]

dt\_pipe = Pipeline(steps)

*# Applying Grid Search to find the best model and the best parameters*

parameters = [ { "model\_\_max\_depth": np.arange(1,21),

"model\_\_min\_samples\_leaf": [1, 5, 10, 20, 50, 100],

"model\_\_min\_samples\_split": np.arange(2, 11),

"model\_\_criterion": ["gini"],

"model\_\_random\_state" : [42]}

]

classifier\_dt = GridSearchCV(estimator = dt\_pipe,

param\_grid = parameters,

cv = 3,

iid = False,

n\_jobs = -1)

classifier\_dt = classifier\_dt.fit(X\_train, y\_train.ravel())

y\_pred\_dt\_train = classifier\_dt.predict(X\_train)

accuracy\_dt\_train = accuracy\_score(y\_train, y\_pred\_dt\_train)

print("Training set: ", accuracy\_dt\_train)

y\_pred\_dt\_test = classifier\_dt.predict(X\_test)

accuracy\_dt\_test = accuracy\_score(y\_test, y\_pred\_dt\_test)

print("Test set: ", accuracy\_dt\_test)

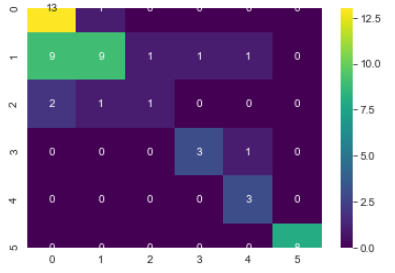
Training set: 0.9

Test set: 0.6851851851851852

from sklearn.metrics import confusion\_matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_dt\_test), annot=True, cmap = 'viridis', fmt='.0f')

plt.show()



### ****Random Forest Classification****

# Fitting Random Forest Classification to the Training set

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import RandomizedSearchCV

classifier\_rf = RandomForestClassifier()

steps = [

('scalar', StandardScaler()),

('model', RandomForestClassifier())

]

rf\_pipe = Pipeline(steps)

parameters = { "model\_\_n\_estimators": [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)],

"model\_\_max\_features": ["auto", "sqrt"],

"model\_\_max\_depth": np.linspace(10, 110, num = 11),

"model\_\_min\_samples\_split": [2, 5, 10],

"model\_\_min\_samples\_leaf": [1, 2, 4],

"model\_\_bootstrap": [True, False],

"model\_\_criterion": ["gini"],

"model\_\_random\_state" : [42] }

classifier\_rf = RandomizedSearchCV(estimator = rf\_pipe,

param\_distributions = parameters,

n\_iter = 100,

cv = 3,

random\_state=42,

verbose = 4,

n\_jobs = -1)

classifier\_rf = classifier\_rf.fit(X\_train, y\_train.ravel())

o/p:-

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 17 tasks | elapsed: 21.2s

[Parallel(n\_jobs=-1)]: Done 90 tasks | elapsed: 1.2min

[Parallel(n\_jobs=-1)]: Done 213 tasks | elapsed: 2.7min

[Parallel(n\_jobs=-1)]: Done 300 out of 300 | elapsed: 3.7min finished

C:\Users\Adinath\Anaconda3\lib\site-packages\sklearn\model\_selection\\_search.py:814: DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.

DeprecationWarning)

y\_pred\_rf\_train = classifier\_rf.predict(X\_train)

accuracy\_rf\_train = accuracy\_score(y\_train, y\_pred\_rf\_train)

print("Training set: ", accuracy\_rf\_train)

y\_pred\_rf\_test = classifier\_rf.predict(X\_test)

accuracy\_rf\_test = accuracy\_score(y\_test, y\_pred\_rf\_test)

print("Test set: ", accuracy\_rf\_test)

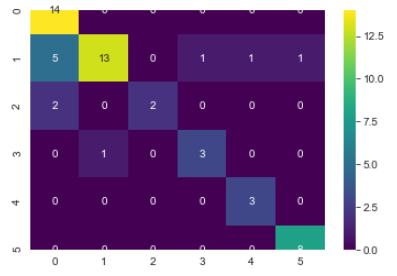
Training set: 1.0

Test set: 0.7962962962962963

from sklearn.metrics import confusion\_matrix

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_rf\_test), annot=True, cmap = 'viridis', fmt='.0f')

plt.show()



### ****Artificial Neural Network (ANN)****

##Artificial Neural Network (ANN)

# Importing the Keras libraries and packages

import keras

from keras.models import Sequential

from keras.layers import Dense

# Feature Scaling

sc\_X = StandardScaler()

X\_train\_scaled = sc\_X.fit\_transform(X\_train)

X\_test\_scaled = sc\_X.fit\_transform(X\_test)

print(X\_train\_scaled.shape)

print(X\_test\_scaled.shape)

o/p:-

(160, 9)

(54, 9)

*# Defining a function to encode output column*

from keras.utils import to\_categorical

def encode(data):

print('Shape of data (BEFORE encode): **%s**' % str(data.shape))

encoded = to\_categorical(data)

print('Shape of data (AFTER encode): **%s\n**' % str(encoded.shape))

return encoded

y\_train\_encoded = encode(y\_train)

Shape of data (BEFORE encode): (160, 1)

Shape of data (AFTER encode): (160, 8)

y\_test\_encoded = encode(y\_test)

Shape of data (BEFORE encode): (54, 1)

Shape of data (AFTER encode): (54, 8)

y\_train\_encoded = np.delete(y\_train\_encoded, [0,4], axis = 1)

y\_test\_encoded = np.delete(y\_test\_encoded, [0,4], axis = 1)

print(y\_train\_encoded[2])

print(y\_test\_encoded[2])

[0. 0. 0. 0. 0. 1.]

[1. 0. 0. 0. 0. 0.]

*# Initialising the ANN*

classifier = Sequential()

*# Adding the input layer and the first hidden layer*

classifier.add(Dense(units = 9, kernel\_initializer = 'uniform', activation = 'relu'))

*# Adding the second hidden layer*

classifier.add(Dense(units = 5, kernel\_initializer = 'uniform', activation = 'relu'))

*# Adding the output layer*

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'softmax'))

*# Compiling the ANN*

classifier.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

*# Fitting the ANN to the Training set*

history = classifier.fit(X\_train\_scaled, y\_train\_encoded, validation\_data=(X\_test\_scaled, y\_test\_encoded), batch\_size = 100, epochs = 1150)

axes[0].plot(history.history['loss'])

axes[0].plot(history.history['val\_loss'])

axes[0].set\_xlabel('Loss', fontsize=14)

axes[0].set\_ylabel('Epuch', fontsize=14)

axes[0].yaxis.tick\_left()

axes[0].legend(['Train', 'Test'], loc='upper left')

axes[1].plot(history.history['acc'])

axes[1].plot(history.history['val\_acc'])

axes[1].set\_xlabel('Accuracy', fontsize=14)

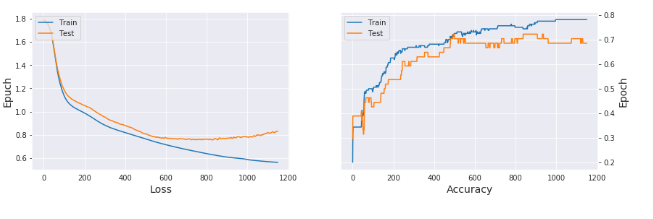
axes[1].set\_ylabel('Epoch', fontsize=14)

axes[1].yaxis.set\_label\_position("right")

axes[1].yaxis.tick\_right()

axes[1].legend(['Train', 'Test'], loc='upper left')

plt.show()



print("Training set: ", history.history.get('acc')[-1])

print("Test set: ", history.history.get('val\_acc')[-1])

Training set: 0.7812499850988388

Test set: 0.6851851940155029

## ****5. Comparing the Results****

models = [('Logistic Regression', accuracy\_lr\_train, accuracy\_lr\_test),

('KNN', accuracy\_knn\_train, accuracy\_knn\_test),

('SVM (Linear)', accuracy\_svm\_linear\_train, accuracy\_svm\_linear\_test),

('SVM (Kernel)', accuracy\_svm\_kernel\_train, accuracy\_svm\_kernel\_test),

('Naive Bayes', accuracy\_nb\_train, accuracy\_nb\_test),

('Decision Tree Classification', accuracy\_dt\_train, accuracy\_dt\_test),

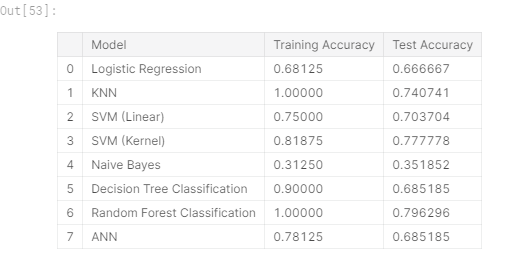
('Random Forest Classification', accuracy\_rf\_train, accuracy\_rf\_test),

('ANN', history.history.get('acc')[-1], history.history.get('val\_acc')[-1]),

]

predict = pd.DataFrame(data = models, columns=['Model', 'Training Accuracy', 'Test Accuracy'])

predict



### ****Visualizing Models Performance****

f, axes = plt.subplots(2,1, figsize=(14,10))

predict.sort\_values(by=['Training Accuracy'], ascending=False, inplace=True)

sns.barplot(x='Training Accuracy', y='Model', data = predict, palette='Blues\_d', ax = axes[0])

*#axes[0].set(xlabel='Region', ylabel='Charges')*

axes[0].set\_xlabel('Training Accuracy', size=16)

axes[0].set\_ylabel('Model')

axes[0].set\_xlim(0,1.0)

axes[0].set\_xticks(np.arange(0, 1.1, 0.1))

predict.sort\_values(by=['Test Accuracy'], ascending=False, inplace=True)

sns.barplot(x='Test Accuracy', y='Model', data = predict, palette='Greens\_d', ax = axes[1])

*#axes[0].set(xlabel='Region', ylabel='Charges')*

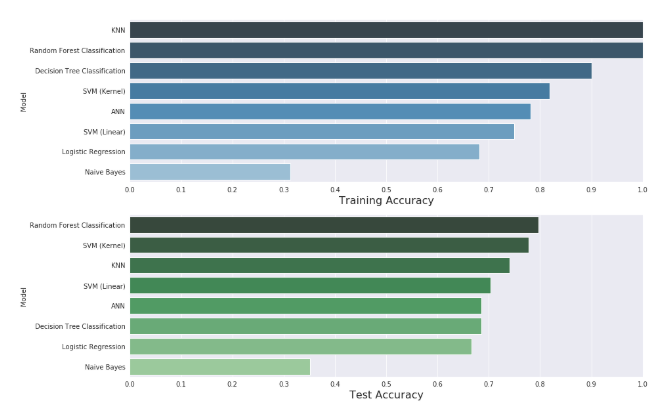
axes[1].set\_xlabel('Test Accuracy', size=16)

axes[1].set\_ylabel('Model')

axes[1].set\_xlim(0,1.0)

axes[1].set\_xticks(np.arange(0, 1.1, 0.1))

plt.show()



## ****6. Conclusion****

In this kernel, I have built 8 classification models using Glass Identification Dataset. These are logistic, k-nn, svm(linear), svm(kernel), naive bayes, decision tree, random forest and artificial neural network. Then measured and visualized the performance of the models. Please make a comment and let me know how to improve model performance, visualization or something in this kernel. This will also help me on my future works.