Electronics & ICT Academy
National Institute of Technology, Warangal

Post Graduate Program in
Artificial Intelligence and Machine Learning

Machine Learning
Question Bank

edureka!

Machine Learning

Machine Learning is the subset of AI which gives the machine the ability to use the stat model to learn from the data.

Machine Learning is used to:

- Detect patterns in the dataset
- Find hidden insights from the data
- Automate analytical and statistical model building

Steps of Machine Learning:

1. **Data collection**
2. **Data analysis**
3. **Data wrangling/Feature engineering**
4. **Train/Test Algorithms**
5. **Model Selection**
6. **Hyperparameter Tuning/Optimisation**
7. **Prediction/Deployment**

Scenario 1: Iris Prediction

You are provided the 'Iris' dataset. You have to predict species of the flowers based on the given features. You have to predict the 'Class' feature which contains 3 species namely, Iris-Setosa, Iris-Virginica, and Iris-Versicolor.

Dataset Description:

The dataset contains 5 features:

sepal length (cm): length of the sepal

sepal width (cm): width of the sepal

petal length (cm): length of the petal

petal width (cm): width of the petal

Class: species of the iris flower

Tasks to be performed:

1. Load the data, check its shape and check for null values - Beginner
2. Handle null values based on the given threshold - Beginner
3. Convert categorical to numerical feature using Label Encoder - Beginner
4. Handle outlier values - Intermediate
5. Plot and analyze Correlation - Beginner
6. Split the dataset for training and testing - Beginner
7. Perform K-Fold cross validation over different classification algorithm - Intermediate
8. Train a logistic regression model and perform prediction on test data - Beginner
9. Evaluate the model using confusion matrix and F1-score - Beginner

Topics Covered:

Data collection

Data analysis

Data wrangling/Feature engineering

Train/Test Algorithms

Predicting using the trained model

Evaluating a model: F1-score and Confusion matrix

In [2]:
```
!wget https://www.dropbox.com/s/webw4cr5dsnm3jv/iris1.csv
```

```
--2020-07-14 06:10:04--  https://www.dropbox.com/s/webw4cr5dsnm3jv
/iris1.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.1, 2620:1
00:601d:1::a27d:501
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.1|:443..
. connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/webw4cr5dsnm3jv/iris1.csv [following]
--2020-07-14 06:10:04--  https://www.dropbox.com/s/raw/webw4cr5dsn
m3jv/iris1.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucc974f28329de57876c787fd308.dl.dropboxuserconte
nt.com/cd/0/inline/A7e7sI2Jlli_GohT7cqibBoPS5R1WidlrA8UO2Kxw0eHNvq
tuSgaXSasyK_Uo_rgrCbp2UJnKtquu1Ar3zjSWC13SpX27QI6FxxFKsE7kCKJE6yg6
-DJgV4LCLnyEH4BAmM/file# [following]
--2020-07-14 06:10:05--  https://ucc974f28329de57876c787fd308.dl.d
ropboxusercontent.com/cd/0/inline/A7e7sI2Jlli_GohT7cqibBoPS5R1Widl
rA8UO2Kxw0eHNvqtuSgaXSasyK_Uo_rgrCbp2UJnKtquu1Ar3zjSWC13SpX27QI6Fx
xFKsE7kCKJE6yg6-DJgV4LCLnyEH4BAmM/file
Resolving ucc974f28329de57876c787fd308.dl.dropboxusercontent.com (
ucc974f28329de57876c787fd308.dl.dropboxusercontent.com)... 162.125
.5.15, 2620:100:601d:15::a27d:50f
Connecting to ucc974f28329de57876c787fd308.dl.dropboxusercontent.c
om (ucc974f28329de57876c787fd308.dl.dropboxusercontent.com)|162.12
5.5.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4332 (4.2K) [text/plain]
Saving to: 'iris1.csv'

iris1.csv           100%[===================>]   4.23K  --.-KB/s
in 0s

2020-07-14 06:10:05 (510 MB/s) - 'iris1.csv' saved [4332/4332]
```

Question-1: Load and analyze the data

Tasks to do:

Load the data in a pandas DataFrame

Have a look at the first five rows

Check if the dataset contains any null values

Check the shape of the dataset

In [3]:
```
import pandas as pd
import numpy as np
df=pd.read_csv('/content/iris1.csv')
df.head()
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Class
0	5.1	3.5	1.4	0.2	Iris-Setosa
1	4.9	3.0	1.4	0.2	Iris-Setosa
2	4.7	3.2	1.3	0.2	Iris-Setosa
3	4.6	3.1	1.5	0.2	Iris-Setosa
4	5.0	NaN	1.4	0.2	Iris-Setosa

In [4]:
```
print('the shape of the data is', df.shape)
```

```
the shape of the data is (150, 5)
```

In [5]:
```
print('The null values in each are:', df.isnull().sum())
```

```
The null values in each are: sepal length (cm)     0
sepal width (cm)      30
petal length (cm)     68
petal width (cm)       0
Class                  0
dtype: int64
```

Question-2: Handle null values

Tasks to do:

> Handle null values
>
> If a column has more than 40% null values, drop that column
>
> Else fill the null values with mean of that column

In [6]:
```
for column in list(df.columns):
  if df[column].isnull().sum()>(0.40*150):
    df.drop(columns=column, axis=1,inplace=True)
  elif df[column].isnull().sum():
    df[column].replace(np.nan,df[column].mean(),inplace=True)
  else:
    continue
```

In [7]:
```
df
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal width (cm)	Class
0	5.1	3.5000	0.2	Iris-Setosa
1	4.9	3.0000	0.2	Iris-Setosa
2	4.7	3.2000	0.2	Iris-Setosa
3	4.6	3.1000	0.2	Iris-Setosa
4	5.0	3.0575	0.2	Iris-Setosa
...	...	...	...	...
145	6.7	3.0000	2.3	Iris-Virginica
146	6.3	2.5000	1.9	Iris-Virginica
147	6.5	3.0000	2.0	Iris-Virginica
148	6.2	3.0575	2.3	Iris-Virginica
149	5.9	3.0000	1.8	Iris-Virginica

150 rows × 4 columns

Question-3: We can not use string objects for prediction so convert categorical feature to numerical feature

Tasks to do:

Convert the categorical features to numerical values using Label Encoder from sklearn

In [8]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Class']=le.fit_transform(df['Class'])
print(df.head())
print(df['Class'].unique())
```

```
   sepal length (cm)  sepal width (cm)  petal width (cm)  Class
0                5.1            3.5000               0.2      0
1                4.9            3.0000               0.2      0
2                4.7            3.2000               0.2      0
3                4.6            3.1000               0.2      0
4                5.0            3.0575               0.2      0
[0 1 2]
```

Question-4: Handle outlier

Tasks to do:

Check for outlier in all the columns using boxplot.

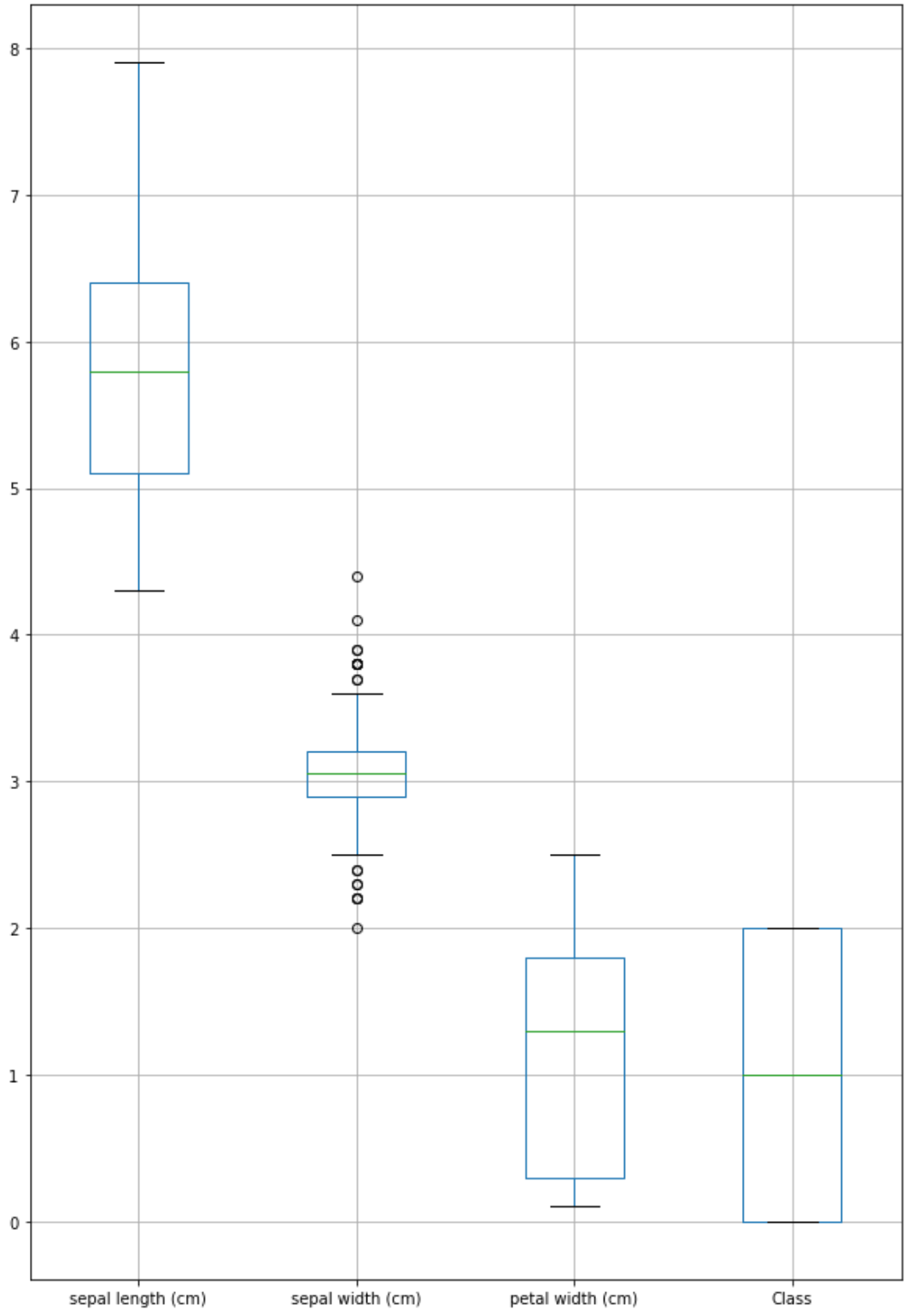
If there are outliers, clip them, lower limit will be Q1-1.5 *IQR and upper limit will be Q3+1.5* IQR.

Q1 = 1st Quartile (25%)

Q3 = 3rd Quartile (75%)

IQR = Inter-quartile range (Q3-Q1)

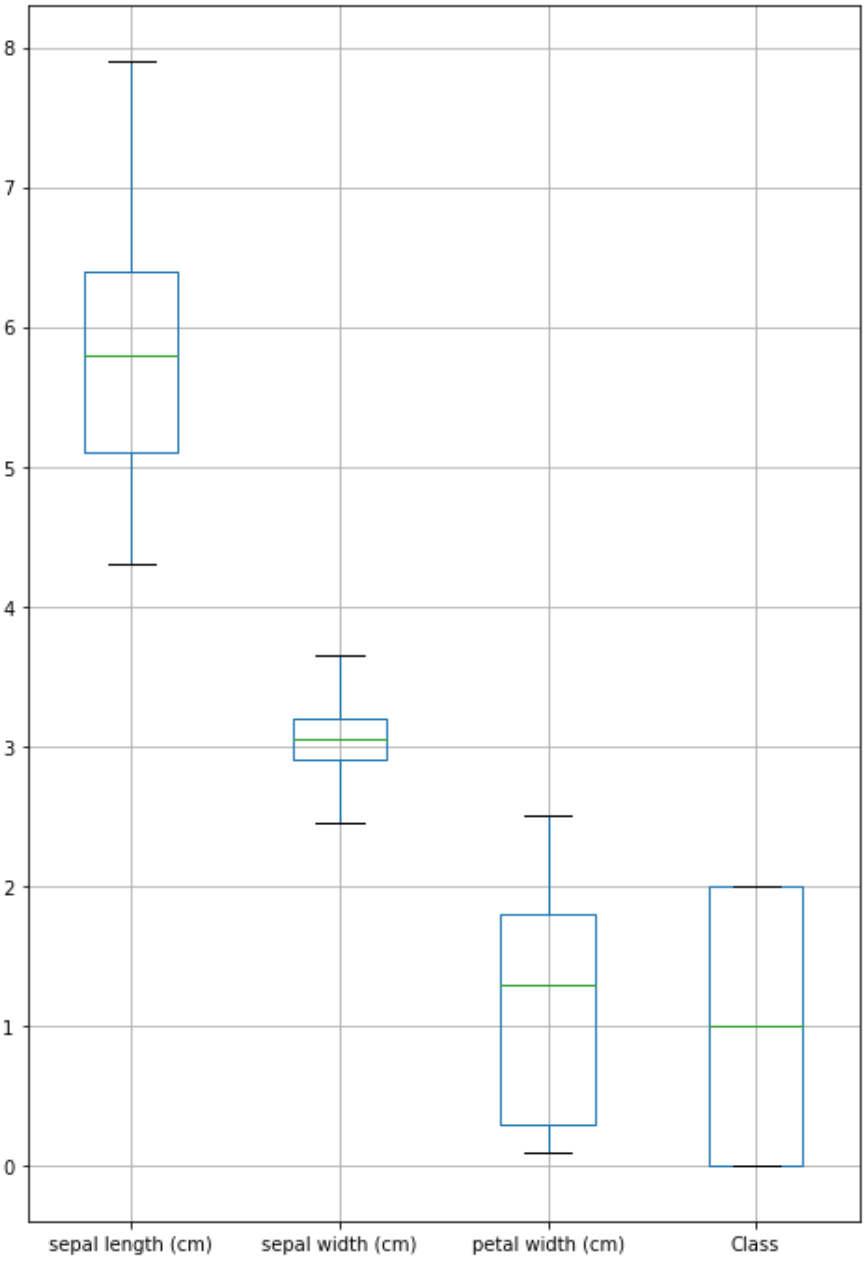
In [18]:
```
import matplotlib.pyplot as plt

df.boxplot(figsize=(10,15))

plt.show()
```



Sepal width contains outliers, so we will need to cap them.

In [20]:
```
q1 = df['sepal width (cm)'].quantile(.25)
q3 = df['sepal width (cm)'].quantile(.75)
IQR = q3 - q1
df['sepal width (cm)'] = np.clip(df['sepal width (cm)'], q1 - 1.5 *
IQR, q3 + 1.5 * IQR)
```

In [21]:
```
df.boxplot(figsize=(8,12))
plt.show()
```



Now we can see there are no outliers left

Question-5: Plot the correlation and tell which feature will help the most during prediction

Calculate correlation

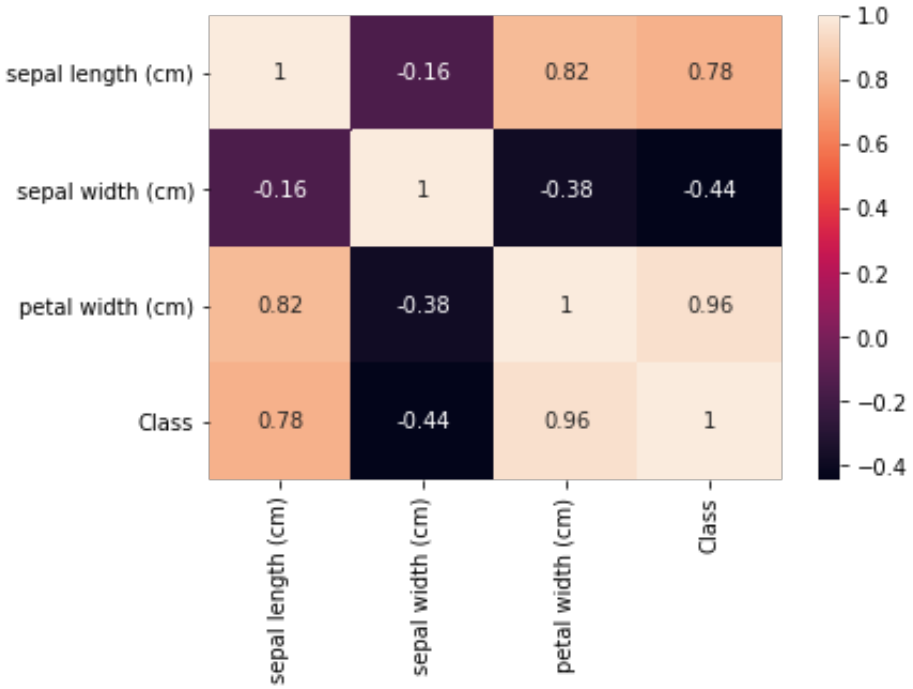
Plot the correlation

Compare the correlation

In [22]:
```
import seaborn as sns
correlation = df.corr()
sns.heatmap(correlation, annot = True)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.
py:19: FutureWarning: pandas.util.testing is deprecated. Use the f
unctions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f588180e278>



Petal and sepal width are highly correlated with feature Class.

Question-6: Split the data into training and testing datasets

Tasks to do:

> Split the dataset using sklearn, with 20% for testing with random_state=7

In [23]:
```
from sklearn.model_selection import train_test_split
X= df.iloc[:,:-1]
y= df.iloc[:,-1]
X_train,X_test,y_train,y_test=train_test_split(X, y,test_size=0.20,
random_state = 7)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(120, 3)
(30, 3)
(120,)
(30,)
```

Question-7: Perform K-Fold cross validation

Tasks to do:

> Perform K-fold with K=10 with random_state = 7
>
> Perform K-Fold with commonly used classification algorithm
>
> Calculate the mean score of each iteration

In [24]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [25]:
```
from sklearn.model_selection import cross_val_score,KFold
#machine learning algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier

models=[]
models.append(('lr',LogisticRegression()))
models.append(('decision tree',DecisionTreeClassifier()))
models.append(('svm',SVC(gamma='auto')))
models.append(('knn',KNeighborsClassifier()))
models.append(('naive bayes',GaussianNB()))
models.append(('Random Forest',RandomForestClassifier()))


for name,model in models:
  kfold=KFold(n_splits=10,random_state=7)
  cross_val_sc=cross_val_score(model,X,y,scoring='accuracy',cv=kfol
d)
  print('{} : acc: {}(standard deviation: {})'.format(name,cross_va
l_sc.mean(),cross_val_sc.std()))
```

```
lr : acc: 0.9200000000000002(standard deviation: 0.077746025264603
99)
decision tree : acc: 0.9333333333333333(standard deviation: 0.0788
8106377466154)
svm : acc: 0.9333333333333333(standard deviation: 0.07302967433402
213)
knn : acc: 0.9333333333333333(standard deviation: 0.06666666666666
665)
naive bayes : acc: 0.9400000000000001(standard deviation: 0.062893
20754704401)
Random Forest : acc: 0.9266666666666667(standard deviation: 0.0813
7703743822466)
```

Question-8: Train the model

Tasks to do:

> Train a logistic regression model for prediction
>
> Also, predict the classes for test data

In [26]:
```
model= LogisticRegression()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
```

Question-9: Evaluate the model

Print confusion matrix of the test data

Also, find the precision and recall using classification report

- Precision is the fraction of relevant instances among the retrieved instances
- While recall (also known as sensitivity) is the fraction of the total amount of relevant instances that were actually retrieved
- https://en.wikipedia.org/wiki/Precision_and_recall (https://en.wikipedia.org/wiki/Precision_and_recall) Visit this link to understand further

In [27]:
```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[7 0 0]
 [0 9 3]
 [0 2 9]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.82      0.75      0.78        12
           2       0.75      0.82      0.78        11

    accuracy                           0.83        30
   macro avg       0.86      0.86      0.86        30
weighted avg       0.84      0.83      0.83        30
```

Scenario 2: Beer Consumption Prediction

The data (sample) were collected in São Paulo, Brazil, in a university area, where there are some parties with groups of students from 18 to 28 years of age (average). The dataset used for this activity has 7 attributes, being a Target, with a period of one year. You have to predict the quantity of beer consumption based on the features that contain climate conditions of a given day.

Dataset Description:

The dataset contains 7 features:

Data: date of the record

Temperatura Media (C): Average temperature of the day in Celsius

Temperatura Minima (C): Minimum temperature of the day in Celsius

Temperatura Maxima (C): Maximum temperature of the day in Celsius

Precipitacao (mm): Percipitation in mm

Final de Semana: If the day is weekend or not

Consumo de cerveja (litros): Beer consumption in litres

Tasks to be performed:

1. Load the dataset, check its shape, Perform EDA using Pandas Profiling - Intermediate
2. Rectify the data of first four columns - Intermediate
3. Create new features using the 'Data' feature and the make 'Data' column as index - Intermediate
4. Handle null and duplicate values - Beginner
5. Check the data-type of the features and convert them to appropriate data-type - Beginner
6. Analyze features with outlier values - Intermediate
7. Plot and analyze Correlation - Beginner
8. Split the dataset for training and testing - Beginner
9. Train a linear regression model and print the intercept and coefficients - Beginner
10. Evaluate the model using R2 score, Mean Absolute Error, and root mean squared error - Beginner

Topics Covered:

Data collection

Data analysis

Data wrangling/Feature engineering

Train/Test Algorithms

Predicting using the trained model

Evaluating a model: R2-score, Mean Absolute Error, and root mean squared error

Fetch and download the data.

In [29]:
```
!wget https://www.dropbox.com/s/9tmnvhivvq4oyc7/Consumo_cerveja.csv
```

```
--2020-07-14 06:55:54--  https://www.dropbox.com/s/9tmnvhivvq4oyc7
/Consumo_cerveja.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.1, 2620:1
00:601d:1::a27d:501
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.1|:443..
. connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/9tmnvhivvq4oyc7/Consumo_cerveja.csv [following]
--2020-07-14 06:55:54--  https://www.dropbox.com/s/raw/9tmnvhivvq4
oyc7/Consumo_cerveja.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc831eb0d046bfcb6f3c7747d547.dl.dropboxuserconte
nt.com/cd/0/inline/A7c4pt-nxLwnVAc-lzJDwlkNUJbI4oLSiL4J45BjCDtqE9B
et4J9VtscMtzfNy7yAtZFbOEUvftK4hG3yDCi3EvpixfyJUSBvdGrAY-aKwkWluxKb
2PmpxA8dsDXF68zRs4/file# [following]
--2020-07-14 06:55:55--  https://uc831eb0d046bfcb6f3c7747d547.dl.d
ropboxusercontent.com/cd/0/inline/A7c4pt-nxLwnVAc-lzJDwlkNUJbI4oLS
iL4J45BjCDtqE9Bet4J9VtscMtzfNy7yAtZFbOEUvftK4hG3yDCi3EvpixfyJUSBvd
GrAY-aKwkWluxKb2PmpxA8dsDXF68zRs4/file
Resolving uc831eb0d046bfcb6f3c7747d547.dl.dropboxusercontent.com (
uc831eb0d046bfcb6f3c7747d547.dl.dropboxusercontent.com)... 162.125
.5.15, 2620:100:601d:15::a27d:50f
Connecting to uc831eb0d046bfcb6f3c7747d547.dl.dropboxusercontent.c
om (uc831eb0d046bfcb6f3c7747d547.dl.dropboxusercontent.com)|162.12
5.5.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20376 (20K) [text/plain]
Saving to: ‘Consumo_cerveja.csv’

Consumo_cerveja.csv 100%[===================>]  19.90K  --.-KB/s
in 0.02s

2020-07-14 06:55:55 (811 KB/s) - ‘Consumo_cerveja.csv’ saved [2037
6/20376]
```

Question-1: Load and anlayze the dataset

Tasks to do:

Load the data in a pandas DataFrame

Have a look at the first five rows

Perform pandas profiling and observe the columns

Check the shape of the dataset

- Pandas profiling is an open-source Python module with which we can quickly do an exploratory data analysis with just a few lines of code
- To learn more about pandas profiling visit the following link: https://towardsdatascience.com/exploratory-data-analysis-with-pandas-profiling-de3aae2ddff3#:~:text=Pandas%20profiling%20is%20an%20open,a%20few%20lines%20of%(https://towardsdatascience.com/exploratory-data-analysis-with-pandas-profiling-de3aae2ddff3#:~:text=Pandas%20profiling%20is%20an%20open,a%20few%20lines%20of%

In [32]:
```
import pandas as pd
df1 = pd.read_csv('/content/Consumo_cerveja.csv',parse_dates=['Data
'])
```

In [ ]:
```
!pip install https://github.com/pandas-profiling/pandas-profiling/a
rchive/master.zip
```

```
Collecting https://github.com/pandas-profiling/pandas-profiling/ar
chive/master.zip
  Downloading https://github.com/pandas-profiling/pandas-profiling
/archive/master.zip
     / 45.0MB 1.0MB/s
Requirement already satisfied: joblib in /usr/local/lib/python3.6/
dist-packages (from pandas-profiling==2.8.0) (0.15.1)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/pyth
on3.6/dist-packages (from pandas-profiling==2.8.0) (1.4.1)
Requirement already satisfied: pandas!=1.0.0,!=1.0.1,!=1.0.2,>=0.2
5.3 in /usr/local/lib/python3.6/dist-packages (from pandas-profili
ng==2.8.0) (1.0.5)
Requirement already satisfied: matplotlib>=3.2.0 in /usr/local/lib
/python3.6/dist-packages (from pandas-profiling==2.8.0) (3.2.2)
Collecting confuse>=1.0.0
  Downloading https://files.pythonhosted.org/packages/b5/6d/bedc0d
1068bd244cee05843313cbec6cebb9f01f925538269bababc6d887/confuse-1.3
.0-py2.py3-none-any.whl (64kB)
     |████████████████████████████████| 71kB 1.9MB/s
Requirement already satisfied: jinja2>=2.11.1 in /usr/local/lib/py
thon3.6/dist-packages (from pandas-profiling==2.8.0) (2.11.2)
Collecting visions[type_image_path]==0.4.4
  Downloading https://files.pythonhosted.org/packages/4a/03/5a45d5
```

```
42257830cf1d9da2cdc1c0bc6f55a9212937b70fdd6d7031b46d6c/visions-0.4
.4-py3-none-any.whl (59kB)
     |████████████████████████████████| 61kB 4.9MB/s
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/pyt
hon3.6/dist-packages (from pandas-profiling==2.8.0) (1.18.5)
Collecting htmlmin>=0.1.12
  Downloading https://files.pythonhosted.org/packages/b3/e7/fcd59e
12169de19f0131ff2812077f964c6b960e7c09804d30a7bf2ab461/htmlmin-0.1
.12.tar.gz
Requirement already satisfied: missingno>=0.4.2 in /usr/local/lib/
python3.6/dist-packages (from pandas-profiling==2.8.0) (0.4.2)
Collecting phik>=0.9.10
  Downloading https://files.pythonhosted.org/packages/01/5a/7ef1c0
4ce62cd72f900c06298dc2385840550d5c653a0dbc19109a5477e6/phik-0.10.0
-py3-none-any.whl (599kB)
     |████████████████████████████████| 604kB 17.6MB/s
Requirement already satisfied: astropy>=4.0 in /usr/local/lib/pyth
on3.6/dist-packages (from pandas-profiling==2.8.0) (4.0.1.post1)
Collecting tangled-up-in-unicode>=0.0.6
  Downloading https://files.pythonhosted.org/packages/4a/e2/e588ab
9298d4989ce7fdb2b97d18aac878d99dbdc379a4476a09d9271b68/tangled_up_
in_unicode-0.0.6-py3-none-any.whl (3.1MB)
     |████████████████████████████████| 3.1MB 28.1MB/s
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/
python3.6/dist-packages (from pandas-profiling==2.8.0) (2.23.0)
Collecting tqdm>=4.43.0
  Downloading https://files.pythonhosted.org/packages/46/62/766389
4f67ac5a41a0d8812d78d9d2a9404124051885af9d77dc526fb399/tqdm-4.47.0
-py2.py3-none-any.whl (66kB)
     |████████████████████████████████| 71kB 6.9MB/s
Requirement already satisfied: ipywidgets>=7.5.1 in /usr/local/lib
/python3.6/dist-packages (from pandas-profiling==2.8.0) (7.5.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/pyth
on3.6/dist-packages (from pandas!=1.0.0,!=1.0.1,!=1.0.2,>=0.25.3->
pandas-profiling==2.8.0) (2018.9)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/loca
l/lib/python3.6/dist-packages (from pandas!=1.0.0,!=1.0.1,!=1.0.2,
>=0.25.3->pandas-profiling==2.8.0) (2.8.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/pyth
on3.6/dist-packages (from matplotlib>=3.2.0->pandas-profiling==2.8
.0) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=
2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=
3.2.0->pandas-profiling==2.8.0) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib
/python3.6/dist-packages (from matplotlib>=3.2.0->pandas-profiling
==2.8.0) (1.2.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/
dist-packages (from confuse>=1.0.0->pandas-profiling==2.8.0) (3.13
)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/
python3.6/dist-packages (from jinja2>=2.11.1->pandas-profiling==2.
8.0) (1.1.1)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/pyt
hon3.6/dist-packages (from visions[type_image_path]==0.4.4->pandas
-profiling==2.8.0) (19.3.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/pyt
```

```
hon3.6/dist-packages (from visions[type_image_path]==0.4.4->pandas
-profiling==2.8.0) (2.4)
Collecting imagehash; extra == "type_image_path"
  Downloading https://files.pythonhosted.org/packages/1a/5d/cc8183
0be3c4705a46cdbca74439b67f1017881383ba0127c41c4cecb7b3/ImageHash-4
.1.0.tar.gz (291kB)
     |████████████████████████████████| 296kB 40.1MB/s
Requirement already satisfied: Pillow; extra == "type_image_path"
in /usr/local/lib/python3.6/dist-packages (from visions[type_image
_path]==0.4.4->pandas-profiling==2.8.0) (7.0.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.6
/dist-packages (from missingno>=0.4.2->pandas-profiling==2.8.0) (0
.10.1)
Requirement already satisfied: numba>=0.38.1 in /usr/local/lib/pyt
hon3.6/dist-packages (from phik>=0.9.10->pandas-profiling==2.8.0)
(0.48.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/li
b/python3.6/dist-packages (from requests>=2.23.0->pandas-profiling
==2.8.0) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib
/python3.6/dist-packages (from requests>=2.23.0->pandas-profiling=
=2.8.0) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.
21.1 in /usr/local/lib/python3.6/dist-packages (from requests>=2.2
3.0->pandas-profiling==2.8.0) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/pyth
on3.6/dist-packages (from requests>=2.23.0->pandas-profiling==2.8.
0) (2.9)
Requirement already satisfied: ipython>=4.0.0; python_version >= "
3.3" in /usr/local/lib/python3.6/dist-packages (from ipywidgets>=7
.5.1->pandas-profiling==2.8.0) (5.5.0)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/
python3.6/dist-packages (from ipywidgets>=7.5.1->pandas-profiling=
=2.8.0) (4.10.1)
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/
python3.6/dist-packages (from ipywidgets>=7.5.1->pandas-profiling=
=2.8.0) (4.3.3)
Requirement already satisfied: nbformat>=4.2.0 in /usr/local/lib/p
ython3.6/dist-packages (from ipywidgets>=7.5.1->pandas-profiling==
2.8.0) (5.0.7)
Requirement already satisfied: widgetsnbextension~=3.5.0 in /usr/l
ocal/lib/python3.6/dist-packages (from ipywidgets>=7.5.1->pandas-p
rofiling==2.8.0) (3.5.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.
6/dist-packages (from python-dateutil>=2.6.1->pandas!=1.0.0,!=1.0.
1,!=1.0.2,>=0.25.3->pandas-profiling==2.8.0) (1.12.0)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/
python3.6/dist-packages (from networkx>=2.4->visions[type_image_pa
th]==0.4.4->pandas-profiling==2.8.0) (4.4.2)
Requirement already satisfied: PyWavelets in /usr/local/lib/python
3.6/dist-packages (from imagehash; extra == "type_image_path"->vis
ions[type_image_path]==0.4.4->pandas-profiling==2.8.0) (1.1.1)
Requirement already satisfied: setuptools in /usr/local/lib/python
3.6/dist-packages (from numba>=0.38.1->phik>=0.9.10->pandas-profil
ing==2.8.0) (47.3.1)
Requirement already satisfied: llvmlite<0.32.0,>=0.31.0dev0 in /us
r/local/lib/python3.6/dist-packages (from numba>=0.38.1->phik>=0.9
```

```
.10->pandas-profiling==2.8.0) (0.31.0)
Requirement already satisfied: pickleshare in /usr/local/lib/pytho
n3.6/dist-packages (from ipython>=4.0.0; python_version >= "3.3"->
ipywidgets>=7.5.1->pandas-profiling==2.8.0) (0.7.5)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib
/python3.6/dist-packages (from ipython>=4.0.0; python_version >= "
3.3"->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (0.8.1)
Requirement already satisfied: pygments in /usr/local/lib/python3.
6/dist-packages (from ipython>=4.0.0; python_version >= "3.3"->ipy
widgets>=7.5.1->pandas-profiling==2.8.0) (2.1.3)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /us
r/local/lib/python3.6/dist-packages (from ipython>=4.0.0; python_v
ersion >= "3.3"->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (1.0.
18)
Requirement already satisfied: pexpect; sys_platform != "win32" in
/usr/local/lib/python3.6/dist-packages (from ipython>=4.0.0; pytho
n_version >= "3.3"->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (4
.8.0)
Requirement already satisfied: tornado>=4.0 in /usr/local/lib/pyth
on3.6/dist-packages (from ipykernel>=4.5.1->ipywidgets>=7.5.1->pan
das-profiling==2.8.0) (4.5.3)
Requirement already satisfied: jupyter-client in /usr/local/lib/py
thon3.6/dist-packages (from ipykernel>=4.5.1->ipywidgets>=7.5.1->p
andas-profiling==2.8.0) (5.3.4)
Requirement already satisfied: ipython-genutils in /usr/local/lib/
python3.6/dist-packages (from traitlets>=4.3.1->ipywidgets>=7.5.1-
>pandas-profiling==2.8.0) (0.2.0)
Requirement already satisfied: jupyter-core in /usr/local/lib/pyth
on3.6/dist-packages (from nbformat>=4.2.0->ipywidgets>=7.5.1->pand
as-profiling==2.8.0) (4.6.3)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /usr/loc
al/lib/python3.6/dist-packages (from nbformat>=4.2.0->ipywidgets>=
7.5.1->pandas-profiling==2.8.0) (2.6.0)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/p
ython3.6/dist-packages (from widgetsnbextension~=3.5.0->ipywidgets
>=7.5.1->pandas-profiling==2.8.0) (5.2.2)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6
/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4->ipython>=4.0.0;
python_version >= "3.3"->ipywidgets>=7.5.1->pandas-profiling==2.8.
0) (0.2.5)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/p
ython3.6/dist-packages (from pexpect; sys_platform != "win32"->ipy
thon>=4.0.0; python_version >= "3.3"->ipywidgets>=7.5.1->pandas-pr
ofiling==2.8.0) (0.6.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3
.6/dist-packages (from jupyter-client->ipykernel>=4.5.1->ipywidget
s>=7.5.1->pandas-profiling==2.8.0) (19.0.1)
Requirement already satisfied: terminado>=0.3.3; sys_platform != "
win32" in /usr/local/lib/python3.6/dist-packages (from notebook>=4
.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas-profili
ng==2.8.0) (0.8.3)
Requirement already satisfied: nbconvert in /usr/local/lib/python3
.6/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0-
>ipywidgets>=7.5.1->pandas-profiling==2.8.0) (5.6.1)
Requirement already satisfied: bleach in /usr/local/lib/python3.6/
dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension
~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (3.1.5)
```

```
Requirement already satisfied: defusedxml in /usr/local/lib/python
3.6/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbexten
sion~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (0.6.0)
Requirement already satisfied: testpath in /usr/local/lib/python3.
6/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextensi
on~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (0.4.4)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib
/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widgets
nbextension~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (0
.8.4)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/
lib/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widg
etsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0)
(1.4.2)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/li
b/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widget
snbextension~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (
0.3)
Requirement already satisfied: packaging in /usr/local/lib/python3
.6/dist-packages (from bleach->nbconvert->notebook>=4.4.1->widgets
nbextension~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0) (2
0.4)
Requirement already satisfied: webencodings in /usr/local/lib/pyth
on3.6/dist-packages (from bleach->nbconvert->notebook>=4.4.1->widg
etsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas-profiling==2.8.0)
(0.5.1)
Building wheels for collected packages: pandas-profiling, htmlmin,
imagehash
  Building wheel for pandas-profiling (setup.py) ... done
  Created wheel for pandas-profiling: filename=pandas_profiling-2.
8.0-py2.py3-none-any.whl size=259932 sha256=f12a54fce7828efb62ea85
35abe525bf68a198e461a15f8e0d29aa3ec6281f0c
  Stored in directory: /tmp/pip-ephem-wheel-cache-o6g9dhuh/wheels/
56/c2/dd/8d945b0443c35df7d5f62fa9e9ae105a2d8b286302b92e0109
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-cp36-none-any
.whl size=27084 sha256=46901dab3797200b1466da017fd934143b236d8a99c
0e016687aafdcd25f641c
  Stored in directory: /root/.cache/pip/wheels/43/07/ac/7c5a9d708d
65247ac1f94066cf1db075540b85716c30255459
  Building wheel for imagehash (setup.py) ... done
  Created wheel for imagehash: filename=ImageHash-4.1.0-py2.py3-no
ne-any.whl size=291990 sha256=7b9285ae0cbbc31171a470a4b6876a0b8343
a7177a779b444807be4cda57ffbe
  Stored in directory: /root/.cache/pip/wheels/07/1c/dc/6831446f09
feb8cc199ec73a0f2f0703253f6ae013a22f4be9
Successfully built pandas-profiling htmlmin imagehash
Installing collected packages: confuse, tangled-up-in-unicode, ima
gehash, visions, htmlmin, phik, tqdm, pandas-profiling
  Found existing installation: tqdm 4.41.1
    Uninstalling tqdm-4.41.1:
      Successfully uninstalled tqdm-4.41.1
  Found existing installation: pandas-profiling 1.4.1
    Uninstalling pandas-profiling-1.4.1:
      Successfully uninstalled pandas-profiling-1.4.1
Successfully installed confuse-1.3.0 htmlmin-0.1.12 imagehash-4.1.
0 pandas-profiling-2.8.0 phik-0.10.0 tangled-up-in-unicode-0.0.6 t
```

```
qdm-4.47.0 visions-0.4.4
```

In [ ]:
```
from pandas_profiling import ProfileReport
profile = ProfileReport(df1, title="Pandas Profiling Report")
profile.to_file("your_report.html")
```

```
/usr/local/lib/python3.6/dist-packages/scipy/cluster/hierarchy.py:
2837: UserWarning: Attempting to set identical bottom == top == 0.
0 results in singular transformations; automatically expanding.
  ax.set_ylim([dvw, 0])
```

Download the file 'your_report.html' and open it, you can see a detailed analysis on your data

In [33]:
```
df1.head()
```

Out[33]:

	Data	Temperatura Media (C)	Temperatura Minima (C)	Temperatura Maxima (C)	Precipitacao (mm)	Final de Semana	Consumo de cerveja (litros)
0	2015-01-01	27,3	23,9	32,5	0	0.0	25.461
1	2015-01-02	27,02	24,5	33,5	0	0.0	28.972
2	2015-01-03	24,82	22,4	29,9	0	1.0	30.814
3	2015-01-04	23,98	21,5	28,6	1,2	1.0	29.799
4	2015-01-05	23,82	21	28,3	0	0.0	28.900

We can see from pandas profiling column number 2 to 5 seems to have inappropriate values. Instead of '.' they have ','. The columns are also having categorical values instead of continuous numerical values

Question-2: Rectify the data

Tasks to do:

> Replace ',' with '.' in columns 'Temperatura Media (C)', 'Temperatura Minima (C)', 'Temperatura Maxima (C)', and 'Precipitacao (mm)'.
>
> Then check if the data contains appropriate values now

In [34]:
```
df1['Temperatura Media (C)']=df1['Temperatura Media (C)'].str.replace(',','.')
df1['Temperatura Minima (C)'] = df1['Temperatura Minima (C)'].str.replace(',','.')
df1['Temperatura Maxima (C)'] = df1['Temperatura Maxima (C)'].str.replace(',','.')
df1['Precipitacao (mm)'] = df1['Precipitacao (mm)'].str.replace(',','.')
```

In [35]:
```
df1.head()
```

Out[35]:

	Data	Temperatura Media (C)	Temperatura Minima (C)	Temperatura Maxima (C)	Precipitacao (mm)	Final de Semana	Consumo de cerveja (litros)
0	2015-01-01	27.3	23.9	32.5	0	0.0	25.461
1	2015-01-02	27.02	24.5	33.5	0	0.0	28.972
2	2015-01-03	24.82	22.4	29.9	0	1.0	30.814
3	2015-01-04	23.98	21.5	28.6	1.2	1.0	29.799
4	2015-01-05	23.82	21	28.3	0	0.0	28.900

Question-3: Create new features using the 'Data' feature and the make 'Data' column as index

Tasks to do:

Create new feature 'Month' from the dates, consisting of the month of the year

Create new feature 'Day' from the dates, consisting of the day of the week

Set values from 'Data' column as indexes

In [36]:
```
df1['Month']=df1.Data.dt.month
df1['day']=df1.Data.dt.dayofweek
df1.iloc[335:341]
```

Out[36]:

	Data	Temperatura Media (C)	Temperatura Minima (C)	Temperatura Maxima (C)	Precipitacao (mm)	Final de Semana	Consumo de cerveja (litros)	Mo
335	2015-12-02	22.1	18.2	29.4	0	0.0	30.471	1
336	2015-12-03	22.44	20.2	26.1	0	0.0	28.405	1
337	2015-12-04	22.76	19	29.1	0	0.0	29.513	1
338	2015-12-05	24.8	19.5	30.6	0.1	1.0	32.451	1
339	2015-12-06	23.12	20.6	28	0.1	1.0	32.780	1
340	2015-12-07	20.04	18	23.9	47.8	0.0	23.375	1

In [37]:
```
df1.set_index('Data',inplace=True)
```

Question-4: Handle null values

Tasks to do:

Only drop those instances where all values are null

Also, check the duplicate values

In [38]:
```
print(df1.isnull().sum())
print(df1.shape)
```

```
Temperatura Media (C)           576
Temperatura Minima (C)          576
Temperatura Maxima (C)          576
Precipitacao (mm)               576
Final de Semana                 576
Consumo de cerveja (litros)     576
Month                           576
day                             576
dtype: int64
(941, 8)
```

In [39]:
```
print(df1.isnull().all(axis=1).sum()) # calculate the number of rows which have null values in all columns
```

```
576
```

In [40]:
```
df1.head()
```

Out[40]:

Data	Temperatura Media (C)	Temperatura Minima (C)	Temperatura Maxima (C)	Precipitacao (mm)	Final de Semana	Consumo de cerveja (litros)	Month
2015-01-01	27.3	23.9	32.5	0	0.0	25.461	1.0
2015-01-02	27.02	24.5	33.5	0	0.0	28.972	1.0
2015-01-03	24.82	22.4	29.9	0	1.0	30.814	1.0
2015-01-04	23.98	21.5	28.6	1.2	1.0	29.799	1.0
2015-01-05	23.82	21	28.3	0	0.0	28.900	1.0

We can see that the 576 instances have all null values in all columns. So we can easily drop those instances

In [41]:
```
df1.dropna(how='all',inplace=True)
```

In [42]:
```
df1.shape
```

Out[42]:
```
(365, 8)
```

In [43]:
```
print(df1.isnull().sum())
```

```
Temperatura Media (C)          0
Temperatura Minima (C)         0
Temperatura Maxima (C)         0
Precipitacao (mm)              0
Final de Semana                0
Consumo de cerveja (litros)    0
Month                          0
day                            0
dtype: int64
```

Now we don't have any null values

In [44]:
```
if df1.duplicated().any():
  print('True: duplicate instances')
else:
  print('False: No duplicate instances')
```

```
False: No duplicate instances
```

Question-5: Handle data types of each features

Tasks to do:

> Check the data-types of the features
>
> Convert them to appropriate data types

In [45]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 365 entries, 2015-01-01 to 2015-12-31
Data columns (total 8 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Temperatura Media (C)         365 non-null    object 
 1   Temperatura Minima (C)        365 non-null    object 
 2   Temperatura Maxima (C)        365 non-null    object 
 3   Precipitacao (mm)             365 non-null    object 
 4   Final de Semana               365 non-null    float64
 5   Consumo de cerveja (litros)   365 non-null    float64
 6   Month                         365 non-null    float64
 7   day                           365 non-null    float64
dtypes: float64(4), object(4)
memory usage: 25.7+ KB
```

The columns with dtype object will be needed to be converted to appropriate dtype

In [46]:
```
df1['Temperatura Media (C)']=df1[['Temperatura Media (C)']].astype(
float)
df1['Temperatura Minima (C)'] = df1['Temperatura Minima (C)'].astyp
e(float)
df1['Temperatura Maxima (C)'] = df1['Temperatura Maxima (C)'].astyp
e(float)
df1['Precipitacao (mm)'] = df1['Precipitacao (mm)'].astype(float)
# Final de semana is a categorical column(like yes or no) so it sho
uld be int, not float
df1['Final de Semana'] = df1['Final de Semana'].astype(int)
```

In [47]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 365 entries, 2015-01-01 to 2015-12-31
Data columns (total 8 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Temperatura Media (C)         365 non-null    float64
 1   Temperatura Minima (C)        365 non-null    float64
 2   Temperatura Maxima (C)        365 non-null    float64
 3   Precipitacao (mm)             365 non-null    float64
 4   Final de Semana               365 non-null    int64  
 5   Consumo de cerveja (litros)   365 non-null    float64
 6   Month                         365 non-null    float64
 7   day                           365 non-null    float64
dtypes: float64(7), int64(1)
memory usage: 25.7 KB
```

In [48]:
```
df1.describe()
```

Out[48]:

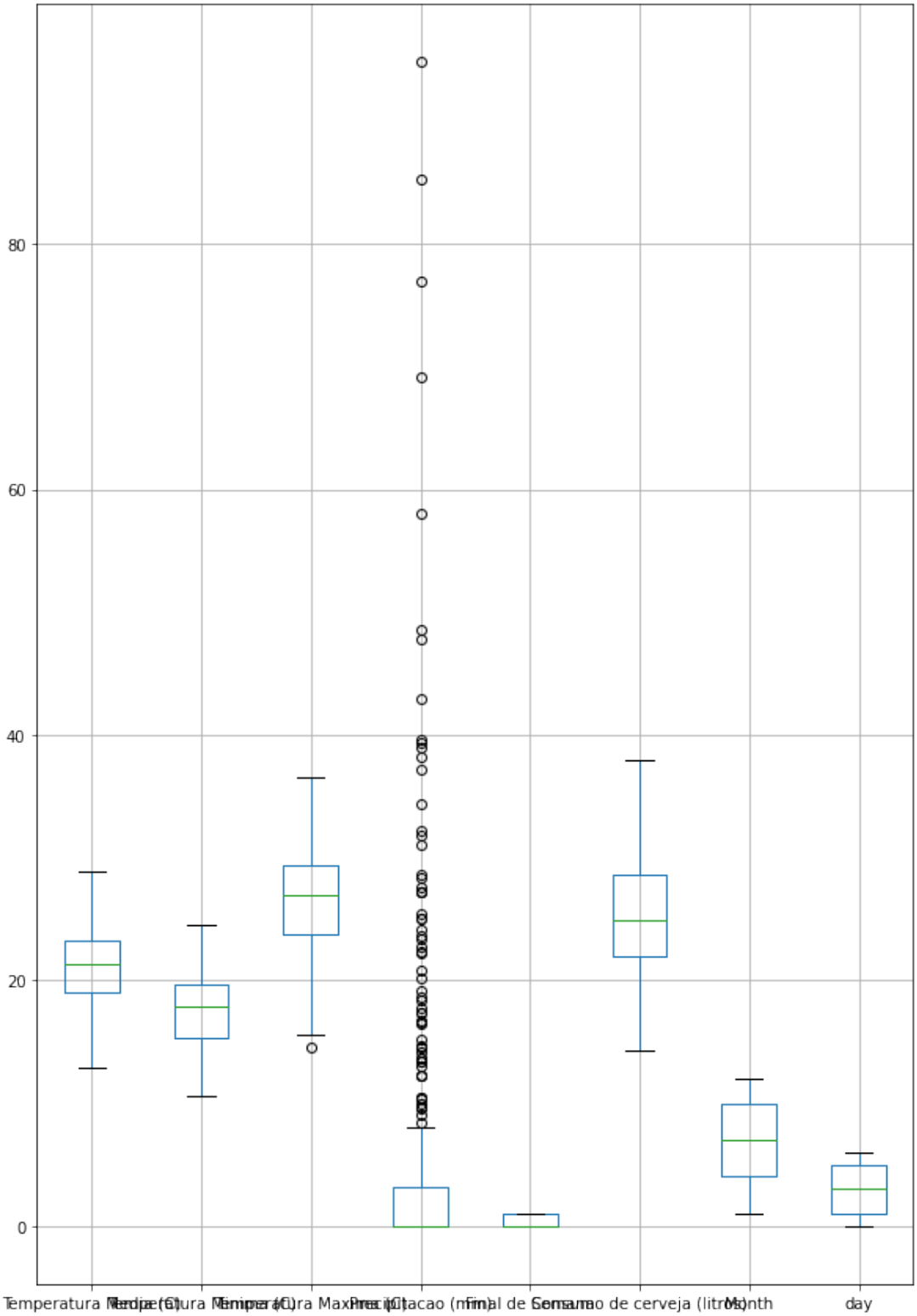
	Temperatura Media (C)	Temperatura Minima (C)	Temperatura Maxima (C)	Precipitacao (mm)	Final de Semana	Consumo de cerveja (litros)	
count	365.000000	365.000000	365.000000	365.000000	365.000000	365.000000	365.
mean	21.226356	17.461370	26.611507	5.196712	0.284932	25.401367	6.
std	3.180108	2.826185	4.317366	12.417844	0.452001	4.399143	3.
min	12.900000	10.600000	14.500000	0.000000	0.000000	14.343000	1.
25%	19.020000	15.300000	23.800000	0.000000	0.000000	22.008000	4.
50%	21.380000	17.900000	26.900000	0.000000	0.000000	24.867000	7.
75%	23.280000	19.600000	29.400000	3.200000	1.000000	28.631000	10.
max	28.860000	24.500000	36.500000	94.800000	1.000000	37.937000	12.

Question-6: Handle Outlier Data

Tasks to do:

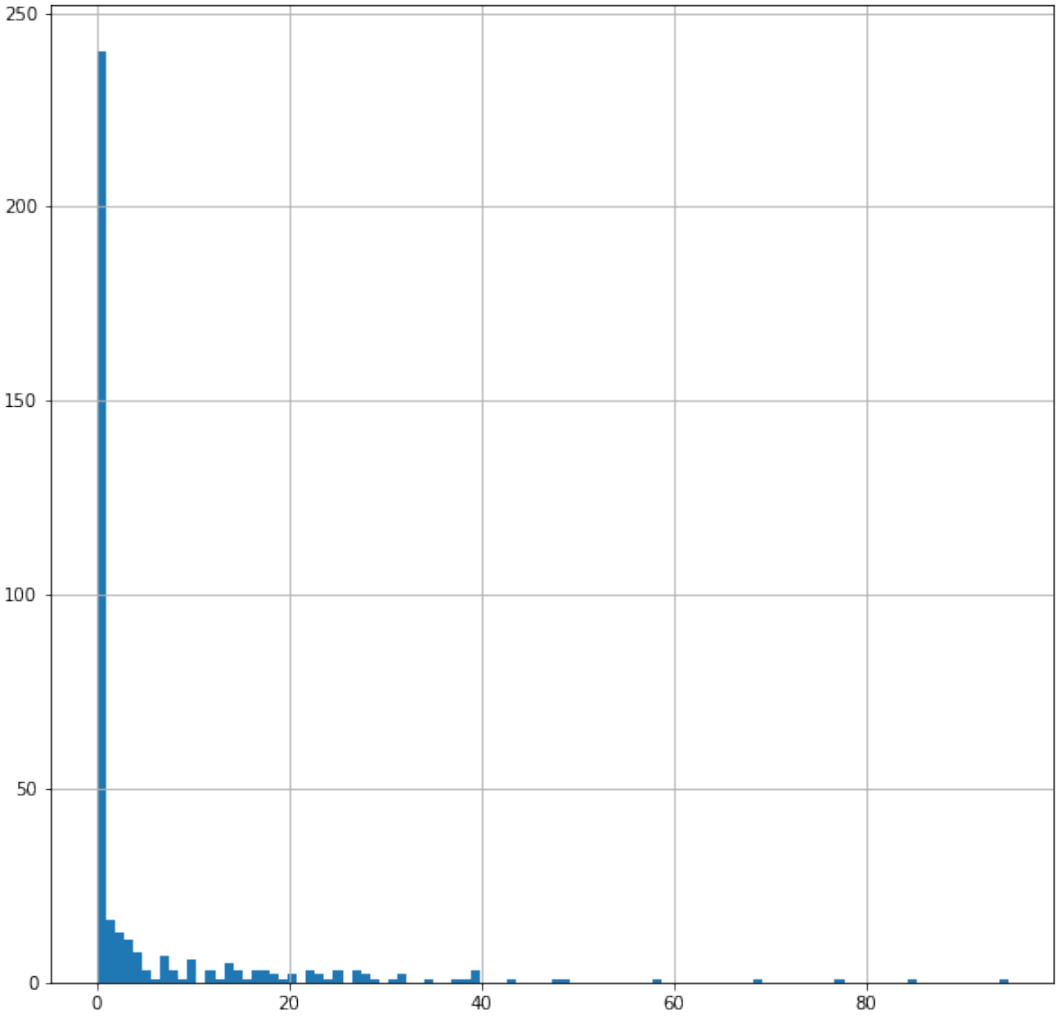
> Check for outlier in all the columns using boxplot
>
> Analyze the column with outliers

In [49]:
```
import matplotlib.pyplot as plt
df1.boxplot(figsize=(10,15))
plt.show()
```



Column 'Precipitacio' seems to have lots of outlier. Let's try to understand this using the distribution of the data

In [50]:
```
df1['Precipitacao (mm)'].hist(bins=100,figsize=(10,10))
plt.show()
```



In [51]:
```
print(df1['Precipitacao (mm)'][df1['Precipitacao (mm)']==0].value_c
ounts())
```

```
0.0    218
Name: Precipitacao (mm), dtype: int64
```

We can see out of 365, 218 values are 0

We can see how the data is largely skewed, thus having so many outliers. It can also be possible that the values with 0 percipitation are the instance where percipitation was not recorded. Lets clip all the values over 40 in column 'Precipitacao (mm)' to 40.

In [53]:
```
df1['Precipitacao (mm)'] = np.clip(df1['Precipitacao (mm)'], 0, 40)
```

Question-7: Calculate correlation and compare

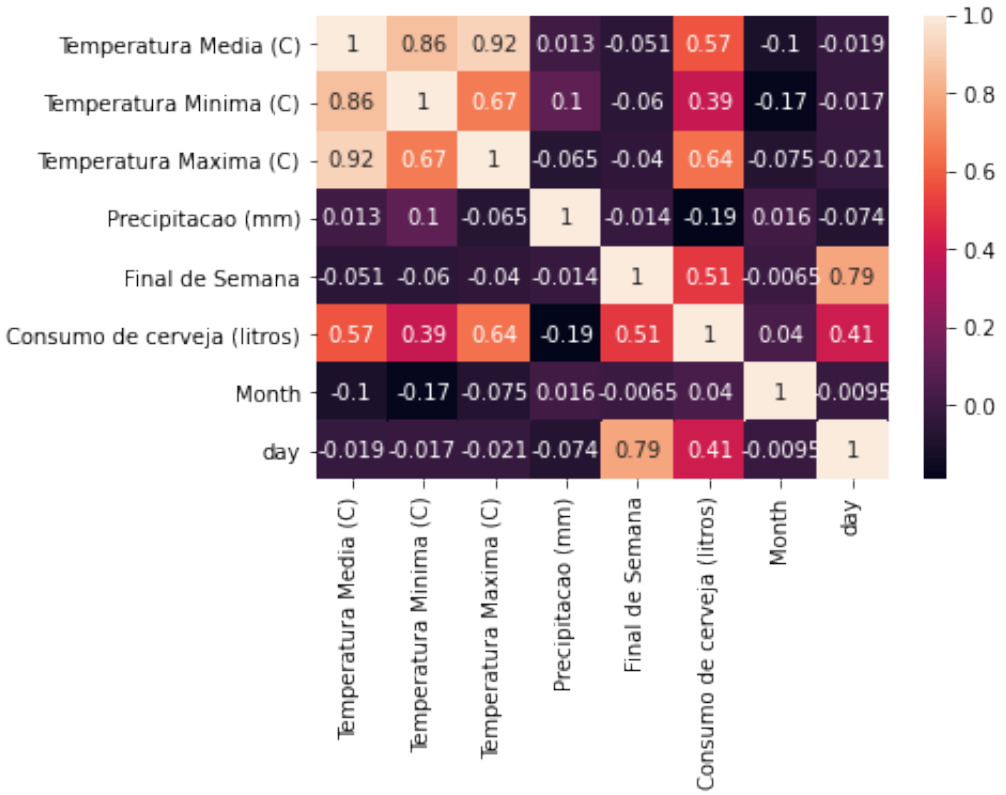
Tasks to do:

Plot the correlation between features

Analyze the correlation of independent features with respect to dependent features

In [54]:
```
import seaborn as sns
correlation = df1.corr()
sns.heatmap(correlation, annot = True)
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x7f58794ac160>



All the features are showing high correlation with the output feature except 'Month' feature. In case of 'Precipitacao (mm)' feature, which has lots of values as 0, still it is showing quite good correlation, so we will keep it

Question-8: Split the data into training and testing datasets

Tasks to do:

> Split the dataset using sklearn, with 20% for testing with random_state=7

In [55]:
```
from sklearn.model_selection import train_test_split
X= df1.drop(columns=['Consumo de cerveja (litros)'],axis=1)
y= df1['Consumo de cerveja (litros)']
X_train,X_test,y_train,y_test=train_test_split(X, y,test_size=0.20,
random_state = 7)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(292, 7)
(73, 7)
(292,)
(73,)
```

Question-9: Train the model

Tasks to do:

> Train a linear regression model for prediction
>
> Also, print the coefficients and intercept from the trained model

In [56]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,y_train)
print('The final coefficients after training is:',lr.coef_)
print('The final intercept after training is:',lr.intercept_)
```

```
The final coefficients after training is: [-1.34341349e-01  1.2205
5186e-01  7.03305117e-01 -5.80658209e-02
  5.24988993e+00  1.36610989e-01  2.42604471e-03]
The final intercept after training is: 5.299119057068435
```

Question-10: Evaluate the model

Tasks to do:

Predict the consumption for the test data

Evaluate the model using R2 score

Evaluate the model using Mean Absolute Error

Evaluate the model using Root Mean Squared Error

In [58]:
```
from sklearn.metrics import r2_score,mean_squared_error, mean_absol
ute_error
y_pred = lr.predict(X_test)
print("r2 score of our model is:", r2_score(y_test,y_pred))
print("mean absolute error of our model is:", mean_absolute_error(y
_test,y_pred))
print("root mean squared error of our model is:", mean_squared_erro
r(y_test,y_pred,squared=False))
```

```
r2 score of our model is: 0.6692125883575043
mean absolute error of our model is: 2.0233917644808708
root mean squared error of our model is: 2.4737414676598464
```

Classification

Classification is the process of grouping things according to similar features they share.

The outcome of classification is categorical, as opposed to regression where the outcome is continuous.

Decision Tree

- It uses a graphical representation of all the possible solutions to a decision
- Decisions are mainly based on some conditions

Random Forest

Random Forest is an ensemble method made of decision trees

Scenario 1: Telecom Churn Prediction

A telecom company wants you to analyze its data, to retain its customers. You are provided with the 'Telecom Churn' dataset.

Problem Statement:

You have to create a model to predict which customer will switch to other telecom service providers, based on the relevant customer data.

Dataset Description:

The dataset contains 20 features:

State: state of the customer

Account length: how long the account has been active

Area code: Area code

International plan: Does the customer have any international plan or not

Voice mail plan: Does the customer have any voice mail plan or not

Number vmail messages: Number of voice mail messages

Total day minutes: Total day minutes used

Total day calls: Total day calls made

Total day charge: Total day charge

Total eve minutes: Total evening minutes used

Total eve calls: Total evening calls made

Total eve charge: Total evening charge

Total night minutes: Total night minutes used

Total night calls: Total night calls made

Total night charge: Total night charge

Total intl minutes: Total international minutes used

Total intl calls: Total international calls made

Total intl charge: Total international charge

Customer service calls: Number of customer service calls made

Churn: Customer will switch provider or not

Tasks to be performed:

1. Load the data, check its shape and check for null values, check unique values for categorical feature - Beginner
2. Convert categorical to numerical feature, also create new feature using 'State' column - Intermediate
3. Plot and analyze Correlation - Beginner
4. Split the dataset for training and testing - Beginner
5. Perform K-Fold cross validation - Intermediate
6. Perform Grid Search cross validation - Intermediate
7. Train a Random Forest model and perform prediction on test data - Beginner
8. Evaluate the model using classification report and accuracy score - Beginner
9. Find 10 best features using trained random forest model and perform K-Fold cross validation on the new data - Intermediate

Topics Covered:

Data collection

Data analysis

Data wrangling/Feature engineering

Train/Test Algorithms

Perform K-Fold and Grid Search Cross Validation

Predicting using the trained model

Evaluating a model: acuracy score and Classification Report

In [86]:
```
!wget https://www.dropbox.com/s/55ar5v2hnvy8cx3/datasets-255093-535845-churn-bigml-80.csv
```

```
--2020-07-14 11:52:14--  https://www.dropbox.com/s/55ar5v2hnvy8cx3
/datasets-255093-535845-churn-bigml-80.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.1, 2620:1
00:601d:1::a27d:501
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.1|:443..
. connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/55ar5v2hnvy8cx3/datasets-255093-535845-churn-bigm
l-80.csv [following]
--2020-07-14 11:52:14--  https://www.dropbox.com/s/raw/55ar5v2hnvy
8cx3/datasets-255093-535845-churn-bigml-80.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc421a5065485bda56c74e6c8380.dl.dropboxuserconte
nt.com/cd/0/inline/A7jm4fsI3V8fzaKDLA2NQmpfdFdJpkEdnyo9OGi7NQ4qW1e
4vN01l3q82dofH3tX3-ZGi6NnnEDDulaeKXBPdeBzQ5ydOvk-uV4mHJG1DNOdm9nHY
6XZnaH4QIRVbZNNEcc/file# [following]
--2020-07-14 11:52:14--  https://uc421a5065485bda56c74e6c8380.dl.d
ropboxusercontent.com/cd/0/inline/A7jm4fsI3V8fzaKDLA2NQmpfdFdJpkEd
nyo9OGi7NQ4qW1e4vN01l3q82dofH3tX3-ZGi6NnnEDDulaeKXBPdeBzQ5ydOvk-uV
4mHJG1DNOdm9nHY6XZnaH4QIRVbZNNEcc/file
Resolving uc421a5065485bda56c74e6c8380.dl.dropboxusercontent.com (
uc421a5065485bda56c74e6c8380.dl.dropboxusercontent.com)... 162.125
.5.15, 2620:100:601d:15::a27d:50f
Connecting to uc421a5065485bda56c74e6c8380.dl.dropboxusercontent.c
om (uc421a5065485bda56c74e6c8380.dl.dropboxusercontent.com)|162.12
5.5.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 226663 (221K) [text/plain]
Saving to: ‘datasets-255093-535845-churn-bigml-80.csv.2’

datasets-255093-535 100%[===================>] 221.35K  --.-KB/s
in 0.09s

2020-07-14 11:52:15 (2.34 MB/s) - ‘datasets-255093-535845-churn-bi
gml-80.csv.2’ saved [226663/226663]
```

Question-1: Load and analyze the data

Tasks to do:

- Load the data in a pandas DataFrame
- Have a look at the first five rows
- Check if the dataset contains any null values
- Check the shape of the dataset
- Check the unique values of the categorical columns

In [87]:
```
import pandas as pd
df=pd.read_csv('/content/datasets-255093-535845-churn-bigml-80.csv'
)
```

In [91]:
```
df.head()
```

Out[91]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3

In [92]:
```
df.shape
```

Out[92]:
```
(2666, 20)
```

The dataset contains 2666 rows and 20 features

In [93]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2666 entries, 0 to 2665
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   State                   2666 non-null   object 
 1   Account length          2666 non-null   int64  
 2   Area code               2666 non-null   int64  
 3   International plan      2666 non-null   object 
 4   Voice mail plan         2666 non-null   object 
 5   Number vmail messages   2666 non-null   int64  
 6   Total day minutes       2666 non-null   float64
 7   Total day calls         2666 non-null   int64  
 8   Total day charge        2666 non-null   float64
 9   Total eve minutes       2666 non-null   float64
 10  Total eve calls         2666 non-null   int64  
 11  Total eve charge        2666 non-null   float64
 12  Total night minutes     2666 non-null   float64
 13  Total night calls       2666 non-null   int64  
 14  Total night charge      2666 non-null   float64
 15  Total intl minutes      2666 non-null   float64
 16  Total intl calls        2666 non-null   int64  
 17  Total intl charge       2666 non-null   float64
 18  Customer service calls  2666 non-null   int64  
 19  Churn                   2666 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 398.5+ KB
```

We can see none of the columns contain any null values.

3 rows contain dtypes object

The dependent feature 'Churn' contains Boolean values

16 columns contain numeric values

In [94]:
```
print(len(df['State'].unique()))
print(df['International plan'].unique())
print(df['Voice mail plan'].unique())
print(df['Churn'].unique())
```

```
51
['No' 'Yes']
['Yes' 'No']
[False  True]
```

Question-2: We cannot use string objects for prediction, so convert categorical feature to numerical values

Tasks to do:

Convert all the categorical features except State to numerical values

Create new feature 'state_churn' using feature 'State', it should contain the ratio of total churn in a given state and total customers of a given state

Create new feature 'avg_day_call_dur' using features 'Total day minutes' and 'Total day calls'

Drop 'Total day minutes' and 'Total day calls'

Similarly create new features 'avg_night_call_dur' and 'avg_eve_call_dur'

In [95]:
```
df.replace({'International plan':{'No':0,'Yes':1},'Voice mail plan'
:{'No':0,'Yes':1},'Churn':{False:0,True:1}},inplace=True)
```

In [96]:
```
df.head()
```

Out[96]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes
0	KS	128	415	0	1	25	265.1	110	45.07	197.4
1	OH	107	415	0	1	26	161.6	123	27.47	195.5
2	NJ	137	415	0	0	0	243.4	114	41.38	121.2
3	OH	84	408	1	0	0	299.4	71	50.90	61.9
4	OK	75	415	1	0	0	166.7	113	28.34	148.3

In [102]:
```
import numpy as np

state_churn = df.groupby(['State'])['Churn'].agg('mean')
state_churn
```

Out[102]:
```
State
AK    0.069767
AL    0.106061
AR    0.234043
AZ    0.066667
CA    0.208333
CO    0.118644
CT    0.186441
DC    0.111111
DE    0.156863
FL    0.129630
GA    0.163265
HI    0.045455
IA    0.078947
ID    0.089286
IL    0.088889
IN    0.111111
KS    0.192308
KY    0.139535
LA    0.085714
MA    0.153846
MD    0.233333
ME    0.224490
MI    0.224138
MN    0.185714
MO    0.098039
MS    0.229167
MT    0.188679
NC    0.160714
ND    0.090909
NE    0.088889
NH    0.209302
NJ    0.280000
NM    0.090909
NV    0.213115
NY    0.176471
OH    0.151515
OK    0.134615
OR    0.112903
PA    0.222222
RI    0.062500
SC    0.224490
SD    0.122449
TN    0.121951
TX    0.290909
UT    0.133333
VA    0.059701
VT    0.105263
WA    0.208333
WI    0.065574
WV    0.079545
WY    0.121212
Name: Churn, dtype: float64
```

In [98]:
```
d=dict()
for i in df.State.unique():
  if i not in d:
    d[i]=state_churn[i]
df['state_churn']=df['State'].map(d)
```

In [99]:
```
df['avg_day_call_dur']=df['Total day minutes']/df['Total day calls'
]
df.drop(columns=['Total day minutes','Total day calls'],axis=1,inpl
ace=True)
```

In [100]:
```
df['avg_night_call_dur']=df['Total night minutes']/df['Total night
calls']
df.drop(columns=['Total night minutes','Total night calls'],axis=1,
inplace=True)
df['avg_eve_call_dur']=df['Total eve minutes']/df['Total eve calls'
]
df.drop(columns=['Total eve minutes','Total eve calls'],axis=1,inpl
ace=True)
```

In [61]:
```
df.dropna(inplace=True)
df.isnull().sum()
```

Out[61]:
```
State                     0
Account length            0
Area code                 0
International plan        0
Voice mail plan           0
Number vmail messages     0
Total day charge          0
Total eve charge          0
Total night charge        0
Total intl minutes        0
Total intl calls          0
Total intl charge         0
Customer service calls    0
Churn                     0
state_churn               0
avg_day_call_dur          0
avg_night_call_dur        0
avg_eve_call_dur          0
dtype: int64
```

In [62]:
```
df.head()
```

Out[62]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day charge	Total eve charge	Total night charge	Total intl minutes
0	KS	128	415	0	1	25	45.07	16.78	11.01	10.0
1	OH	107	415	0	1	26	27.47	16.62	11.45	13.7
2	NJ	137	415	0	0	0	41.38	10.30	7.32	12.2
3	OH	84	408	1	0	0	50.90	5.26	8.86	6.6
4	OK	75	415	1	0	0	28.34	12.61	8.41	10.1

Question-3: Plot the correlation and tell which feature will help the most while prediction

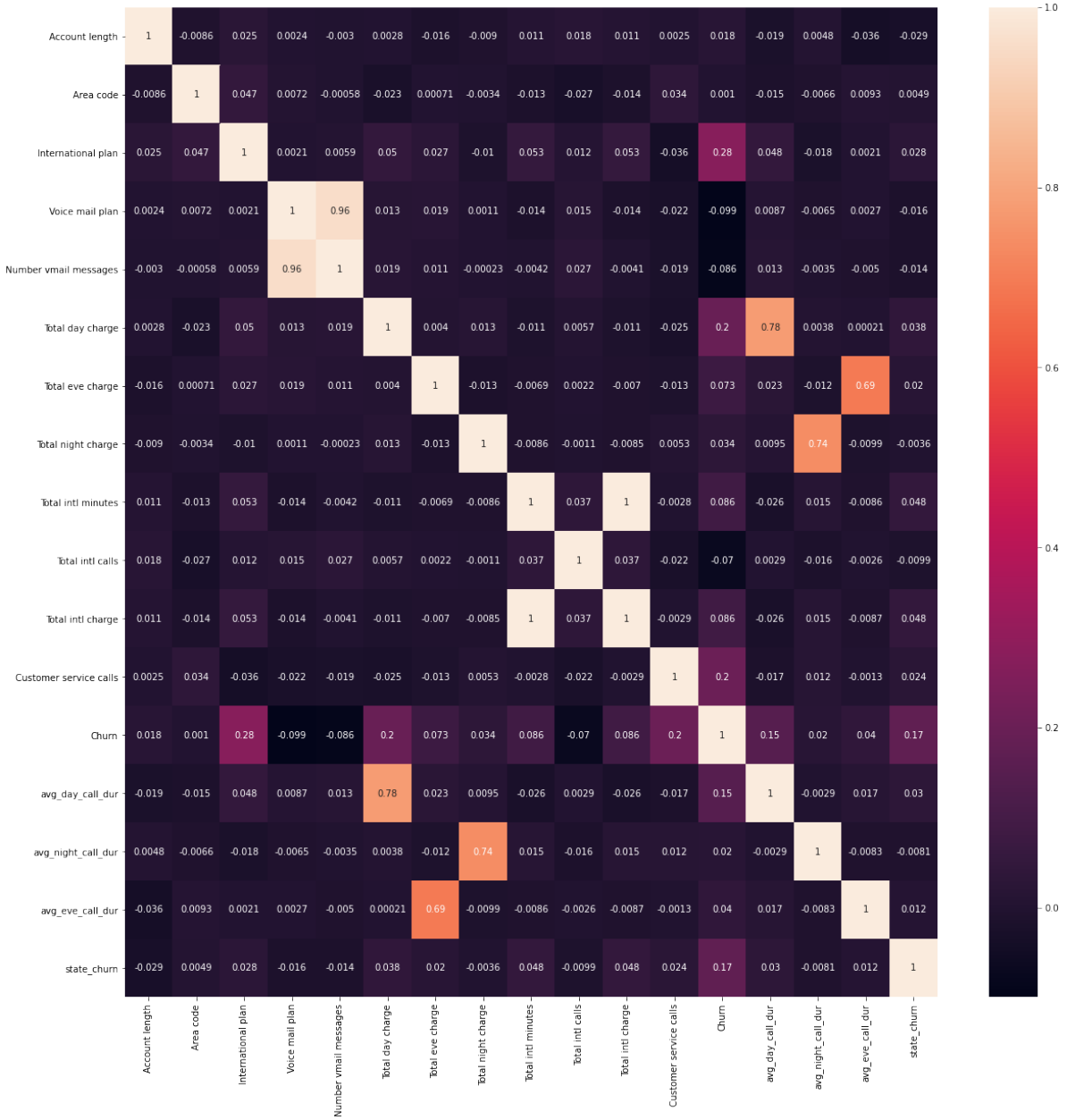
Tasks to do:

- Calculate correlation
- Plot the correlation
- Compare the correlation

In [36]:
```
import seaborn as sns
from matplotlib import pyplot as plt
fig, ax = plt.subplots(figsize=(20,20))
correlation = df.corr()
sns.heatmap(correlation,annot=True,ax=ax)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.
py:19: FutureWarning: pandas.util.testing is deprecated. Use the f
unctions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1c9064dcc0>



We can see the new feature 'state_churn' is also highly correlated with the target variable 'churn'

Question-4: Split the data into training and testing datasets

Tasks to do:

Drop the feature 'State' as we cannot use string object for prediction

Split the dataset using sklearn, with 20% for testing with random_state=7

In [63]:
```
from sklearn.model_selection import train_test_split
X= df.drop(columns=['State','Churn'],axis=1)
y= df['Churn']
X_train,X_test,y_train,y_test=train_test_split(X, y,test_size=0.20,
random_state = 7)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2130, 16)
(533, 16)
(2130,)
(533,)
```

Question-5: Perform K-Fold cross validation

Tasks to do:

Perform Stratified K-fold with K=10

Perform Stratified K-Fold with Random Forest and Decision Tree algorithm

Calculate the mean score of each iteration

- This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

In [65]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score,StratifiedKFold
cv = StratifiedKFold(10)
model_rf = RandomForestClassifier(n_estimators=100, n_jobs=-1, rand
om_state=7)
model_dt = DecisionTreeClassifier()
score=cross_val_score(model_rf, X_train, y_train, cv=cv)
print(f'score of random forest classifier:{score.mean()}')
score=cross_val_score(model_dt, X_train, y_train, cv=cv)
print(f'score of decision tree classifier:{score.mean()}')
```

```
score of random forest classifier:0.9427230046948356
score of decision tree classifier:0.8995305164319249
```

Since we are getting higher accuracy from random forest, let us train our data random forest model

Question-6: Perform Grid Search cross validation to find the best parameters for Random Forest Classifier

Tasks to do:

Perform Grid Search CV on random forest classifier to find the best number of:

trees between 20 to 100

maximum depth for the trees, between 3 and 15

maximum features to be used, between range 4 and 17

> Use the following dictionary to implement the first 3 points:
>
> {'n_estimators':range(20,100,20), 'max_depth':range(3,15),
> 'max_features':range(4,17)}

Perform GridSearchCV with scoring = recall

Cross validate the model with 5 folds

Check the best score and best estimator found by cross validation

In [66]:
```
from sklearn.model_selection import GridSearchCV
model_params = {'n_estimators':range(20,100,20), 'max_depth':range(
3,15), 'max_features':range(4,17)}
model_cv = GridSearchCV(model_rf,model_params,cv=5,n_jobs=-1,verbos
e=True,return_train_score = True,scoring = 'recall')
model_cv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 624 candidates, totalling 3120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   10.3s
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:   46.0s
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed:  8.8min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 12.6min
[Parallel(n_jobs=-1)]: Done 3120 out of 3120 | elapsed: 16.9min fi
nished
```

Out[66]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_
alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', ma
x_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decreas
e=0.0,
                                              min_impurity_split=N
one,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_
leaf=0.0,
                                              n_estimators=100, n_
jobs=-1,
                                              oob_score=False, ran
dom_state=7,
                                              verbose=0, warm_star
t=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': range(3, 15),
                         'max_features': range(4, 17),
                         'n_estimators': range(20, 100, 20)},
             pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=True,
             scoring='recall', verbose=True)
```

In [67]:
```
model_cv.best_params_
```

Out[67]:
```
{'max_depth': 14, 'max_features': 13, 'n_estimators': 60}
```

The best parameters found by grid search are max_depth=11, max_features=8 and n_estimators =80

max_depth: maximum depth of the tree

max_features: the number of features to consider for finding best split

n_estimators: the number of trees by random forest model

In [68]:
```
model_cv.best_score_
```

Out[68]:
```
0.7484848484848484
```

In [69]:
```
model_cv.best_estimator_
```

Out[69]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight
=None,
                       criterion='gini', max_depth=14, max_feature
s=13,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_spl
it=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=
60, n_jobs=-1,
                       oob_score=False, random_state=7, verbose=0,
                       warm_start=False)
```

Question-7: Train the Random Forest model

Tasks to do:

> Train the random forest model using the parameters found by Grid Search CV
>
> Also, predict the classes for test data

In [74]:
```
final_model = model_cv.best_estimator_
final_model.fit(X_train,y_train)
y_pred=final_model.predict(X_test)
```

Question-8: Evaluate the model

Tasks to do:

Evaluate the model using accuracy score

Evaluate the model using classification report

In [75]:
```
from sklearn.metrics import accuracy_score, classification_report
print(f"Accuracy score on test data is {accuracy_score(y_test,y_pred)}")
print(classification_report(y_test,y_pred))
```

```
Accuracy score on test data is 0.9568480300187617
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       472
           1       0.93      0.67      0.78        61

    accuracy                           0.96       533
   macro avg       0.95      0.83      0.88       533
weighted avg       0.96      0.96      0.95       533
```

Question-9: Find the 10 best features and perform cross validation using just 10 features

Tasks to do:

Find the 10 best features from the random forest model

Create new dataframe with the 10 best features

Perform K-Fold cross validation on the new dataset with reduced features

Compare the score with previous K-fold cross validation

In [79]:
```
from numpy import argsort
imp = final_model.feature_importances_
x= argsort(imp)[::-1]
col_names=list(df.columns)
col_names.remove('State')
col_names.remove('Churn')
top_features = list()
for i in x[:13]:
  print(col_names[i])
  top_features.append(col_names[i])
```

```
Total day charge
Customer service calls
Total eve charge
International plan
Total intl calls
Total intl minutes
avg_eve_call_dur
Total intl charge
avg_day_call_dur
Total night charge
avg_night_call_dur
Number vmail messages
state_churn
```

In [80]:
```
X_new = df[top_features]
X_new
```

Out[80]:

	Total day charge	Customer service calls	Total eve charge	International plan	Total intl calls	Total intl minutes	avg_eve_call_dur	Total intl charge	a
0	45.07	1	16.78	0	3	10.0	1.993939	2.70	
1	27.47	1	16.62	0	3	13.7	1.898058	3.70	
2	41.38	0	10.30	0	5	12.2	1.101818	3.29	
3	50.90	2	5.26	1	7	6.6	0.703409	1.78	
4	28.34	3	12.61	1	3	10.1	1.215574	2.73	
...	...	...	...	...	...	...	...	...	
2661	22.90	2	16.12	0	5	11.8	2.789706	3.19	
2662	26.55	2	18.32	0	6	9.9	1.710317	2.67	
2663	39.29	3	13.04	0	4	9.6	2.789091	2.59	
2664	30.74	2	24.55	0	6	14.1	4.979310	3.81	
2665	39.85	0	22.60	0	4	13.7	3.242683	3.70	

2663 rows × 13 columns

In [81]:
```
new_model = model_cv.best_estimator_
score=cross_val_score(new_model, X_new, y, cv=5)
print(f'score of random forest classifier:{score.mean()}')
```

```
score of random forest classifier:0.9463069023402785
```

We can see with just 10 features accuracy is quite good as compared to previous K-Fold cross validation

> Current Score: 0.946
>
> Previous Score: 0.947

Scenario 2: Chronic Kidney Disease Prediction

The dataset is taken over 2-month period in India. It has 400 rows with 26 features like red blood cells, pedal edema, sugar, etc.

Problem Statement:

Your aim is to classify whether a patient has chronic kidney disease or not.

Dataset Description:

The dataset contains 26 features:

age - age

bp - blood pressure

sg - specific gravity

al - albumin

su - sugar

rbc - red blood cells

pc - pus cell

pcc - pus cell clumps

ba - bacteria

bgr - blood glucose random

bu - blood urea

sc - serum creatinine

sod - sodium

pot - potassium

hemo - hemoglobin

pcv - packed cell volume

wc - white blood cell count

rc - red blood cell count

htn - hypertension

dm - diabetes mellitus

cad - coronary artery disease

appet - appetite

pe - pedal edema

ane - anemia

class - class

Tasks to be performed:

1. Load the data, check its shape and check for null values, check unique values for categorical feature, Convert features into appropriate data type - Beginner
2. Convert categorical to numerical feature using Label Encoder - Intermediate
3. Plot and analyze Correlation - Beginner
4. Split the dataset for training and testing - Beginner

1. Perform Grid Search cross validation also perform prediction using the best model - Intermediate
2. Evaluate the model using classification report and confusion matrix - Beginner

Topics Covered:

Data collection

Data analysis

Data wrangling/Feature engineering

Train/Test Algorithms

Perform Grid Search Cross Validation

Predicting using the trained model

Evaluating a model: Confusion matrix and Classification Report

In [ ]:
```
!wget https://www.dropbox.com/s/hd1jir21dmlgh27/kidney_disease.csv
```

```
--2020-06-16 10:15:18--  https://www.dropbox.com/s/hd1jir21dmlgh27
/kidney_disease.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:
100:6032:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443.
.. connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/hd1jir21dmlgh27/kidney_disease.csv [following]
--2020-06-16 10:15:18--  https://www.dropbox.com/s/raw/hd1jir21dml
gh27/kidney_disease.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uce5e99a9d509ba96d4f06cb0c4c.dl.dropboxuserconte
nt.com/cd/0/inline/A5yavMjjnag7QFRLq-c1BwMPJneG9QR9K4ZEyRqvD0RxVcS
cj6IZc3re88zhi6LvtbeX4AyEWCroGL-7dVmpHPBgbpmszPaHnwhcS2ufRi5Bx6_pG
Jt_qXxlMeTt8MNr5r8/file# [following]
--2020-06-16 10:15:18--  https://uce5e99a9d509ba96d4f06cb0c4c.dl.d
ropboxusercontent.com/cd/0/inline/A5yavMjjnag7QFRLq-c1BwMPJneG9QR9
K4ZEyRqvD0RxVcScj6IZc3re88zhi6LvtbeX4AyEWCroGL-7dVmpHPBgbpmszPaHnw
hcS2ufRi5Bx6_pGJt_qXxlMeTt8MNr5r8/file
Resolving uce5e99a9d509ba96d4f06cb0c4c.dl.dropboxusercontent.com (
uce5e99a9d509ba96d4f06cb0c4c.dl.dropboxusercontent.com)... 162.125
.82.15, 2620:100:6032:15::a27d:520f
Connecting to uce5e99a9d509ba96d4f06cb0c4c.dl.dropboxusercontent.c
om (uce5e99a9d509ba96d4f06cb0c4c.dl.dropboxusercontent.com)|162.12
5.82.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 48551 (47K) [text/plain]
Saving to: ‘kidney_disease.csv’

kidney_disease.csv  100%[===================>]  47.41K  --.-KB/s
in 0.02s

2020-06-16 10:15:19 (2.94 MB/s) - ‘kidney_disease.csv’ saved [4855
1/48551]
```

Question-1: Load and analyze the data

Tasks to do:

- Load the data in a pandas DataFrame
- Have a look at the first five rows
- Check if the dataset contains any null values
- Check the shape of the dataset
- Check the unique values of the categorical columns
- Check the data-types of all the columns
- Convert features to appropriate data-types

In [ ]:
```
import pandas as pd
data = pd.read_csv('/content/kidney_disease.csv')
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(400, 26)
```

In [ ]:
```
data.head()
```

Out[ ]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              400 non-null    int64  
 1   age             391 non-null    float64
 2   bp              388 non-null    float64
 3   sg              353 non-null    float64
 4   al              354 non-null    float64
 5   su              351 non-null    float64
 6   rbc             248 non-null    object 
 7   pc              335 non-null    object 
 8   pcc             396 non-null    object 
 9   ba              396 non-null    object 
 10  bgr             356 non-null    float64
 11  bu              381 non-null    float64
 12  sc              383 non-null    float64
 13  sod             313 non-null    float64
 14  pot             312 non-null    float64
 15  hemo            348 non-null    float64
 16  pcv             330 non-null    object 
 17  wc              295 non-null    object 
 18  rc              270 non-null    object 
 19  htn             398 non-null    object 
 20  dm              398 non-null    object 
 21  cad             398 non-null    object 
 22  appet           399 non-null    object 
 23  pe              399 non-null    object 
 24  ane             399 non-null    object 
 25  classification  400 non-null    object 
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

We can see pcv, wc and rc column have numeric values but there dtypes is object. We need to fix this

In [ ]:
```
data.pcc.unique()
```

Out[ ]:
```
array(['notpresent', 'present', nan], dtype=object)
```

In [ ]:
```
clean_df = data.dropna()
```

In [ ]:
```
clean_df.isnull().sum()
```

Out[ ]:
```
id                0
age               0
bp                0
sg                0
al                0
su                0
rbc               0
pc                0
pcc               0
ba                0
bgr               0
bu                0
sc                0
sod               0
pot               0
hemo              0
pcv               0
wc                0
rc                0
htn               0
dm                0
cad               0
appet             0
pe                0
ane               0
classification    0
dtype: int64
```

In [ ]:
```
clean_df.shape
```

Out[ ]:
```
(158, 26)
```

In [ ]:
```
clean_df['pcv']=clean_df[['pcv']].astype(float)
clean_df['wc'] = clean_df['wc'].astype(float)
clean_df['rc'] = clean_df['rc'].astype(float)
```

Question-2: We cannot use string objects for prediction, so convert the categorical feature to numerical values

Tasks to do:

Convert all the categorical features to numerical values using Label Encoder from sklearn

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in clean_df.columns:
  if clean_df[i].dtype=='object':
    clean_df[i]=le.fit_transform(clean_df[i])
clean_df.head()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy
  """
```

Out[ ]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo
3	3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	56.0	3.8	111.0	2.5	11.2
9	9	53.0	90.0	1.020	2.0	0.0	0	0	1	0	70.0	107.0	7.2	114.0	3.7	9.5
11	11	63.0	70.0	1.010	3.0	0.0	0	0	1	0	380.0	60.0	2.7	131.0	4.2	10.8
14	14	68.0	80.0	1.010	3.0	2.0	1	0	1	1	157.0	90.0	4.1	130.0	6.4	5.6
20	20	61.0	80.0	1.015	2.0	0.0	0	0	0	0	173.0	148.0	3.9	135.0	5.2	7.7

Question-3: Plot the correlation and tell which feature will help the most while prediction

Tasks to do:

Calculate correlation

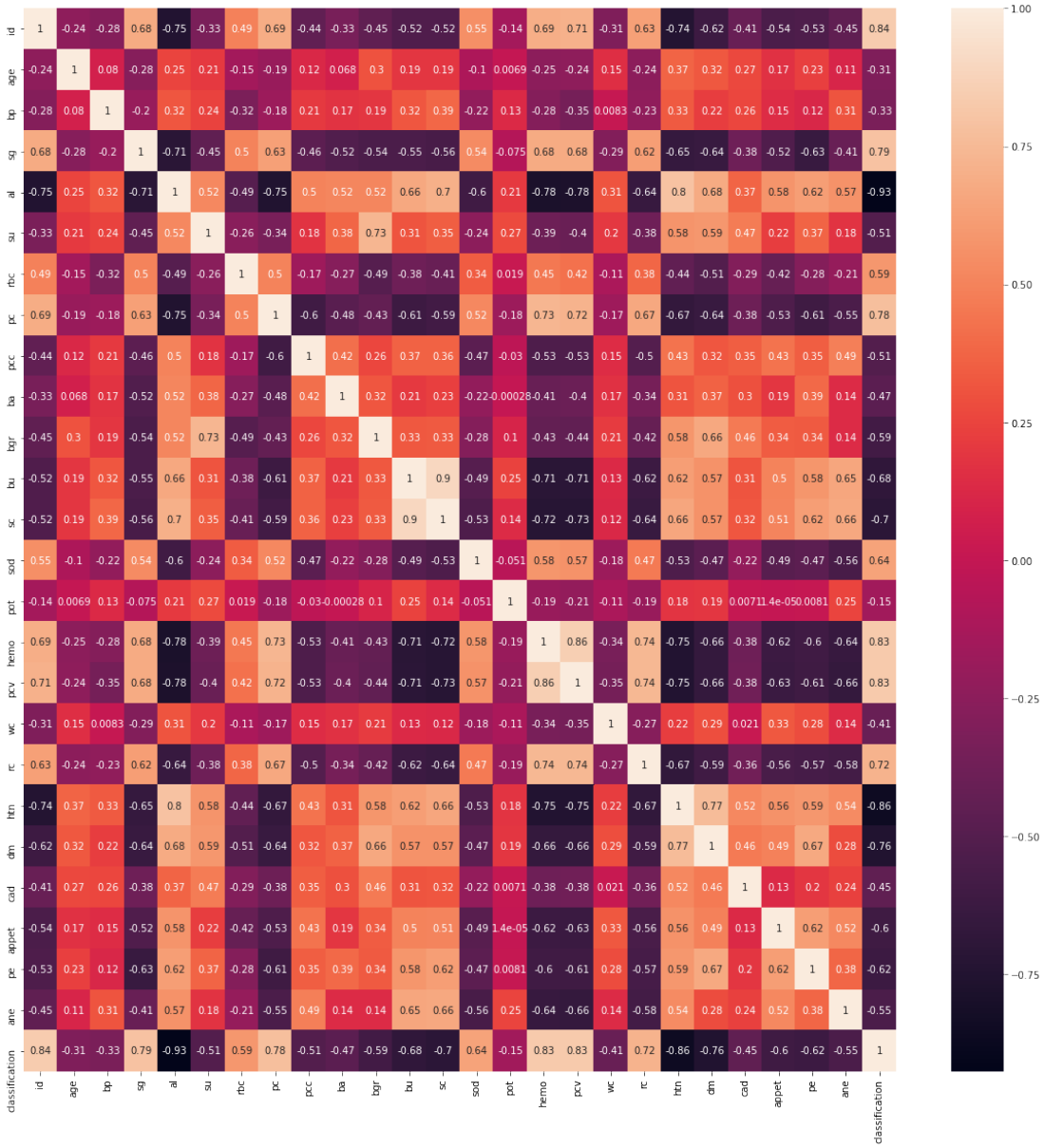
Plot the correlation

Compare the correlation

In [ ]:
```
import seaborn as sns
from matplotlib import pyplot as plt
fig, ax = plt.subplots(figsize=(20,20))
correlation = clean_df.corr()
sns.heatmap(correlation,annot=True,ax=ax)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.
py:19: FutureWarning: pandas.util.testing is deprecated. Use the f
unctions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f30ba28a588>



We can see each of the 25 independent features are showing quite good correlation with the target variable

Question-4: Split the data into training and testing datasets

Tasks to do:

Split the dataset using sklearn, with 30% for testing with random_state=1

In [ ]:
```
from sklearn.model_selection import train_test_split
X= clean_df.drop(columns=['classification'],axis=1)
y= clean_df['classification']
X_train,X_test,y_train,y_test=train_test_split(X, y,test_size=0.30,
random_state = 1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(110, 25)
(48, 25)
(110,)
(48,)
```

Question-5: Perform Grid Search cross validation to find the best parameters for Decision Tree Classifier

Tasks to do:

Perform Grid Search CV on random forest classifier to find the best number of:

maximum depth for the trees, between 3 and 25

maximum features to be used, between range 4 and 22

Cross validate the model with 5 folds

Check the best score and best estimator found by cross validation

Perform prediction using the model found in cross validation

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
model = DecisionTreeClassifier(random_state=7)
model_params = {'criterion':['gini', 'entropy'], 'max_depth':range(
3,25), 'max_features':range(4,22)}
model_cv = GridSearchCV(model,model_params,cv=5,n_jobs=-1,verbose=T
rue)
model_cv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 792 candidates, totalling 3960 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent
workers.
[Parallel(n_jobs=-1)]: Done 788 tasks      | elapsed:    2.7s
[Parallel(n_jobs=-1)]: Done 3960 out of 3960 | elapsed:   12.7s fi
nished
```

Out[ ]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class
_weight=None,
                                              criterion='gini', ma
x_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decreas
e=0.0,
                                              min_impurity_split=N
one,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_
leaf=0.0,
                                              presort='deprecated'
,
                                              random_state=7, spli
tter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(3, 25),
                         'max_features': range(4, 22)},
             pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=False,
             scoring=None, verbose=True)
```

In [ ]:
```
model_cv.best_estimator_
```

Out[ ]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion
='gini',
                       max_depth=3, max_features=4, max_leaf_nodes
=None,
                       min_impurity_decrease=0.0, min_impurity_spl
it=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='depr
ecated',
                       random_state=7, splitter='best')
```

In [ ]:
```
model_cv.best_score_
```

Out[ ]:
```
0.990909090909091
```

In [ ]:
```
model_cv.best_params_
```

Out[ ]:
```
{'criterion': 'gini', 'max_depth': 3, 'max_features': 4}
```

In [ ]:
```
y_pred=model_cv.predict(X_test)
```

Question-6: Evaluate the model

Tasks to do:

Evaluate the model using confusion matrix

Evaluate the model using classification report

In [ ]:
```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[13  0]
 [ 0 35]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00        35

    accuracy                           1.00        48
   macro avg       1.00      1.00      1.00        48
weighted avg       1.00      1.00      1.00        48
```

Module-3: Classification II

Classification is the process of classifying data in the diferent categories based on training data it was fitted. In this question bank, we will be applying Naive Bayes, SVM, and KNN to classify the data.

Getting Datasets:

In [ ]:
```
!wget https://www.dropbox.com/s/mcflbukhp33tbj2/plastics_type.csv -nv
!wget https://www.dropbox.com/s/rtkd1f4iulf8jp0/auto-mpg.csv -nv
!wget https://www.dropbox.com/s/oqsri5a5a85tlaa/guest_data.csv -nv
```

Scenario 1: Emerald Oyster Resorts

Emerald Oyster is proud to have been catering business and luxury travelers for more than 50 years. The worldwide offering of high-end hotel stays, customer services and with best in class chefs of every locus. The accommodations are designed in such a way that they reflect the surrounding environment with decor and hotel's location. Each hotel exhibits its own sense of style and individual properties with 24-hour customer service.

Problem Statement:

Recent visitors do say, you should brace yourself for the high cost of meals at multiple restaurants located in the resort. In order to resolve this issue, the amenities can be remodeled with new menu and choices based on the economical condition of the customer. As the resorts cover a wide area only rich people go in depths of the resorts who buy the shuttle services; customers with low budget only explore the outer parts of the resorts. The task is to identify economic class of the customer based on the data collected so that the board can resolve this with proper remodeling.

Tasks to be Performed:

In order to attain the above goal below tasks must be performed:

- Read the dataset with no headers; Then put respective columns names and find the missing values in each column. - **Beginner**
- Process any null values present in the data with the respective method. - **Beginner**
- Create two datasets one using dummy variable and one using LabelEncoder. - **Intermediate**
- Split both the datasets into training and testing set and apply Gaussian Naive Bayes. - **Intermediate**
- Evaluate both the model using confusion matrix. - **Advanced**
- Apply Bernoulli Naive Bayes on both the datasets and evaluate them again. - **Advanced**

Dataset Description:

The data collected has 45222 observations and 14 features along with the target variable *Class*, which identifies the income of the individual.

Features:

- **Age**: Real [17.0, 90.0]
- **Workclass**: {Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked}
- **Fnlwgt**: Real [12285.0, 1490400.0]
- **Education**: {Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool}
- **Education-num**: Real [1.0, 16.0]
- **Marital-status**: {Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse}
- **Occupation**: {Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces}
- **Relationship**: {Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried}
- **Race**: {White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black}
- **Sex**: {Female, Male}
- **Capital-gain**: Real [0.0, 99999.0]
- **Capital-loss**: Real [0.0, 4356.0]
- **Hours-per-week**: Real [1.0, 99.0]
- **Native-country**: {United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands}

Target Variable:

- **Class**: {>50K, <=50K}

Topics Covered:

- Naive Bayes
- Confusion Matrix

Question-1: Read the dataset with no headers; Then put respective columns names and find the missing values in each column.

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
cols=['Age','Workaclass','Fnlwgt','Education','Edu_num',
      'Marital_Status','Occupation','Relationship','Race','Sex','Capital_gain','Capital_loss',
     'HPW','Native_Country','Target']
```

In [3]:
```
data=pd.read_csv('guest_data.csv',header=None)
data.columns=cols
```

In [4]:
```
data.head()
```

Out[4]:

	Age	Workaclass	Fnlwgt	Education	Edu_num	Marital_Status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	HPW	Nat
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	U
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	U
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	U
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	U
4	18	NaN	103497	Some-college	10	Never-married	NaN	Own-child	White	Female	0	0	30	U

Basic Data Information

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Age             48842 non-null  int64 
 1   Workaclass      46043 non-null  object
 2   Fnlwgt          48842 non-null  int64 
 3   Education       48842 non-null  object
 4   Edu_num         48842 non-null  int64 
 5   Marital_Status  48842 non-null  object
 6   Occupation      46033 non-null  object
 7   Relationship    48842 non-null  object
 8   Race            48842 non-null  object
 9   Sex             48842 non-null  object
 10  Capital_gain    48842 non-null  int64 
 11  Capital_loss    48842 non-null  int64 
 12  HPW             48842 non-null  int64 
 13  Native_Country  47985 non-null  object
 14  Target          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

- Dataset has 48842 entries.

In [6]:
```
# Target distribution
def target_distribution(data,col):
    target_ratio=pd.DataFrame({'Counts':data[col].value_counts(),
                               'Percentage':data[col].value_counts()/len(data)})
    import matplotlib.pyplot as plt
    import seaborn as sns
    plt.figure(figsize = (10,1))
    plt.barh(target_ratio.index, target_ratio.Percentage,color='c')
    plt.xlabel('Percentage')
    plt.show()
target_distribution(data,'Target')
```

```
<Figure size 1000x100 with 1 Axes>
```

Question-2: Process any null values present in the data with the respective method.

In [7]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[7]:

	Col_name	Missing value?	Count_	Percentage
6	Occupation	True	2809	0.057512
1	Workaclass	True	2799	0.057307
13	Native_Country	True	857	0.017546
0	Age	False	0	0.000000
2	Fnlwgt	False	0	0.000000
3	Education	False	0	0.000000
4	Edu_num	False	0	0.000000
5	Marital_Status	False	0	0.000000
7	Relationship	False	0	0.000000
8	Race	False	0	0.000000
9	Sex	False	0	0.000000
10	Capital_gain	False	0	0.000000
11	Capital_loss	False	0	0.000000
12	HPW	False	0	0.000000
14	Target	False	0	0.000000

In [8]:
```
print('Total Missing values: %s'%sum(miss.Count_))
```

```
Total Missing values: 6465
```

In [9]:
```
import numpy as np
```

In [10]:
```
for i in miss[miss.Count_!=0].Col_name:
    temp=data[i].mode()
    tes=data[i].isnull()
    ind=tes[tes==True].index
    quill=list(data[i])
    for j in ind:
        quill[j]=temp[0]
    data[i]=quill
```

In [11]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?': [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns]})
```

In [12]:
```
miss.sort_values(by='Count_',ascending=False)
```

Out[12]:

	Col_name	Missing value?	Count_
0	Age	False	0
1	Workaclass	False	0
2	Fnlwgt	False	0
3	Education	False	0
4	Edu_num	False	0
5	Marital_Status	False	0
6	Occupation	False	0
7	Relationship	False	0
8	Race	False	0
9	Sex	False	0
10	Capital_gain	False	0
11	Capital_loss	False	0
12	HPW	False	0
13	Native_Country	False	0
14	Target	False	0

Question-3: Create two datasets one using dummy variable and one using labelencoder.

In [13]:
```
dummy_data=data.copy()
label_data=data.copy()
```

- Dummy Dataset

In [14]:
```
# findUnique returns a dataframe with every column name, number of unique values, minimum values, maximum values
# and total unique values present altogether in the dataset
def findUnique(data):
    unq=0
    unq_data={'Col_Name':[],'Unique_Counts':[],'Max_value':[],'Min_value':[]}
    for i in data.columns:
        unq_data['Col_Name'].append(i)
        unq_data['Unique_Counts'].append(len(dummy_data[i].unique()))
        unq_data['Max_value'].append(dummy_data[i].max())
        unq_data['Min_value'].append(dummy_data[i].min())
        unq+=len(dummy_data[i].unique())
    return pd.DataFrame(unq_data),unq
u,ud=findUnique(dummy_data)
u.sort_values(by='Unique_Counts')
```

Out[14]:

	Col_Name	Unique_Counts	Max_value	Min_value
9	Sex	2	Male	Female
14	Target	2	>50K	<=50K
8	Race	5	White	Amer-Indian-Eskimo
7	Relationship	6	Wife	Husband
5	Marital_Status	7	Widowed	Divorced
1	Workaclass	8	Without-pay	Federal-gov
6	Occupation	14	Transport-moving	Adm-clerical
3	Education	16	Some-college	10th
4	Edu_num	16	16	1
13	Native_Country	41	Yugoslavia	Cambodia
0	Age	74	90	17
12	HPW	96	99	1
11	Capital_loss	99	4356	0
10	Capital_gain	123	99999	0
2	Fnlwgt	28523	1490400	12285

In [15]:
```
print('Total unique values:',ud)
```

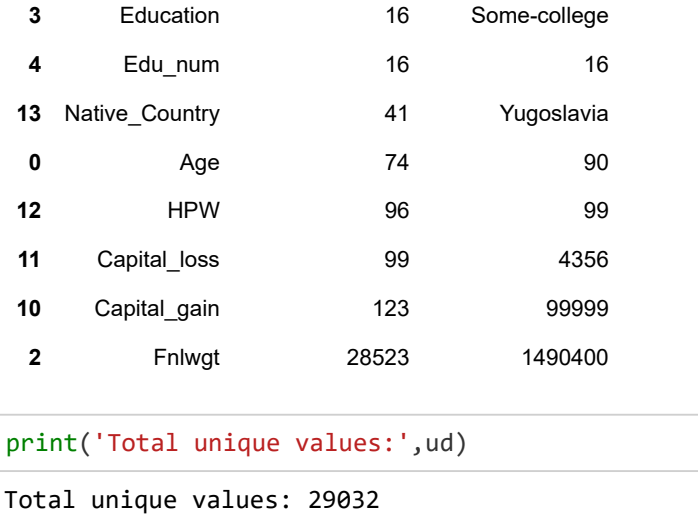
```
Total unique values: 29032
```

In [16]:
```
print('Dataset shape: %s'%str(dummy_data.shape))
```

```
Dataset shape: (48842, 15)
```

In [17]:
```
# Since Age and Education are ordinal in nature we should use label encoder on these columns
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
dummy_data.Age=lb.fit_transform(dummy_data.Age)
dummy_data.Education=lb.fit_transform(dummy_data.Education)
dummy_data.HPW=lb.fit_transform(dummy_data.HPW)
```

In [18]:
```
# All that left is 3 columns 'Capital_gain','Capital_loss' and 'Fnlwgt'
import matplotlib.pyplot as plt
dummy_data.boxplot(column=['Capital_gain','Capital_loss','Fnlwgt'])
plt.show()
```



In [19]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [20]:
```
scl=MinMaxScaler()
```

In [21]:
```
dummy_data.Capital_gain=scl.fit_transform(np.array(dummy_data.Capital_gain.values.astype(float)).reshape(1,-1))[0]
dummy_data.Capital_loss=scl.fit_transform(np.array(dummy_data.Capital_loss.values.astype(float)).reshape(1,-1))[0]
dummy_data.Fnlwgt=scl.fit_transform(np.array(dummy_data.Fnlwgt.values.astype(float)).reshape(1,-1))[0]
```

In [22]:
```
dols=['Workaclass','Edu_num',
      'Marital_Status','Occupation','Relationship','Race','Sex','Native_Country']
dummy_data=pd.get_dummies(dummy_data,columns=dols)
```

In [23]:
```
print('Dummy Dataset shape: %s'%str(dummy_data.shape))
```

```
Dummy Dataset shape: (48842, 106)
```

In [24]:
```
dummy_data.head()
```

Out[24]:

	Age	Fnlwgt	Education	Capital_gain	Capital_loss	HPW	Target	Workaclass_Federal-gov	Workaclass_Local-gov	Workaclass_Never-worked	...	Native_Coun
0	8	0.0	1	0.0	0.0	39	<=50K	0	0	0	...	
1	21	0.0	11	0.0	0.0	49	<=50K	0	0	0	...	
2	11	0.0	7	0.0	0.0	39	>50K	0	1	0	...	
3	27	0.0	15	0.0	0.0	39	>50K	0	0	0	...	
4	1	0.0	15	0.0	0.0	29	<=50K	0	0	0	...	

5 rows × 106 columns

LabelEncode the target variable

In [25]:
```
from sklearn.preprocessing import LabelEncoder
```

In [26]:
```
lb=LabelEncoder()
```

In [27]:
```
dummy_data.Target=lb.fit_transform(dummy_data.Target)
```

In [28]:
```
dummy_data.head()
```

Out[28]:

	Age	Fnlwgt	Education	Capital_gain	Capital_loss	HPW	Target	Workaclass_Federal-gov	Workaclass_Local-gov	Workaclass_Never-worked	...	Native_Coun
0	8	0.0	1	0.0	0.0	39	0	0	0	0	...	
1	21	0.0	11	0.0	0.0	49	0	0	0	0	...	
2	11	0.0	7	0.0	0.0	39	1	0	1	0	...	
3	27	0.0	15	0.0	0.0	39	1	0	0	0	...	
4	1	0.0	15	0.0	0.0	29	0	0	0	0	...	

5 rows × 106 columns

- Label Data

In [29]:
```
def lb_encode(x):
    lb=LabelEncoder()
    return lb.fit_transform(x)
```

In [30]:
```
label_data=label_data.apply(lb_encode)
```

In [31]:
```
label_data.head()
```

Out[31]:

	Age	Workaclass	Fnlwgt	Education	Edu_num	Marital_Status	Occupation	Relationship	Race	Sex	Capital_gain	Capital_loss	HPW	Native_
0	8	3	19329	1	6	4	6	3	2	1	0	0	39	
1	21	3	4212	11	8	2	4	0	4	1	0	0	49	
2	11	1	25340	7	11	2	10	0	4	1	0	0	39	
3	27	3	11201	15	9	2	6	0	2	1	98	0	39	
4	1	3	5411	15	9	4	9	3	4	0	0	0	29	

Question-4: Split both the datasets into training and testing set and apply Gaussian Naive Bayes.

In [32]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
```

- Dummy Data

In [33]:
```
X=dummy_data.drop('Target',axis=1)
y=dummy_data.Target
X_drain, X_dest, y_drain, y_dest = train_test_split(X, y, train_size=0.7,test_size=0.3, random_state=101)
```

In [34]:
```
dummy_gnb=GaussianNB()
dummy_gnb.fit(X_drain,y_drain)
```

Out[34]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [35]:
```
dummy_pred=dummy_gnb.predict(X_dest)
```

- Label Data

In [36]:
```
lX=label_data.drop('Target',axis=1)
ly=label_data.Target
X_train, X_test, y_train, y_test = train_test_split(lX, ly, train_size=0.7,test_size=0.3, random_state=101)
```

In [37]:
```
label_gnb=GaussianNB()
label_gnb.fit(X_train,y_train)
```

Out[37]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [38]:
```
label_pred=label_gnb.predict(X_test)
```

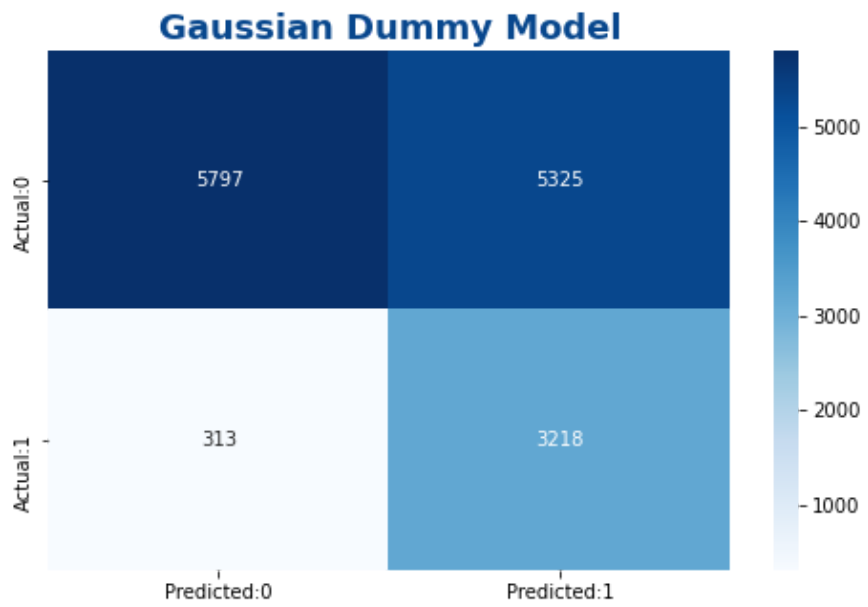
Question-5: Evaluate both the model using confusion matrix.

In [39]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
from sklearn.metrics import classification_report,f1_score
import matplotlib.pyplot as plt
import seaborn as sns
```

In [40]:
```
print('The accuracy of the Gaussian dummy model is: ',round(accuracy_score(y_dest,dummy_pred)*100,2))
print('The accuracy of the Gaussian label model is: ',round(accuracy_score(y_test,label_pred)*100,2))
```

```
The accuracy of the Gaussian dummy model is:  61.52
The accuracy of the Gaussian label model is:  81.61
```

In [41]:
```
dcm=confusion_matrix(y_dest,dummy_pred)
dconf_matrix=pd.DataFrame(data=dcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(dconf_matrix, annot=True,fmt='d',cmap='Blues')
fm={'size':18,'color':'#08478d','weight':'bold'}
plt.title('Gaussian Dummy Model',**fm)
plt.show()
```



In [42]:
```
TN=dcm[0,0]
TP=dcm[1,1]
FN=dcm[1,0]
FP=dcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 5797
True Positive 3218
False Negative 313
False Positive 5325
Sensitivity 0.9113565562163694
Specificity 0.5212192051789246
```

The model is more sensitive than specific means it predicts target *>50K* more accurately.

- **Precision**: Ability of model to distinguish between negative and positive labels.
- **Recall**: Ability of model to find all positive samples.
- **F1 Score**: Weighted harmonic mean of the precision and recall.

In [43]:
```
# Let's see precision and recall values
#Initializae the evaluation dictionary
def initialize_evaluator():
    return {'Model':[],'Accuracy':[],'Precision':[],'Recall':[],'F1_score':[]}

#Insert data in evaluation dictionary
def insert_data(test,pred,model):
    eval_data=initialize_evaluator()
    eval_data['Model'].append(model)
    eval_data['Accuracy'].append(accuracy_score(test,pred))
    eval_data['Precision'].append(precision_score(test,pred))
    eval_data['Recall'].append(recall_score(test,pred))
    eval_data['F1_score'].append(f1_score(test,pred))
    return eval_data

# Append data of one dictionary to another
def append_data(data1, data2):
    for i in data1.keys():
        data2[i].extend(data1[i])
    return data2

dummy_eval_data=insert_data(y_test,dummy_pred,'Gaussian Dummy Model')
```

In [44]:
```
pd.DataFrame(dummy_eval_data)
```

Out[44]:

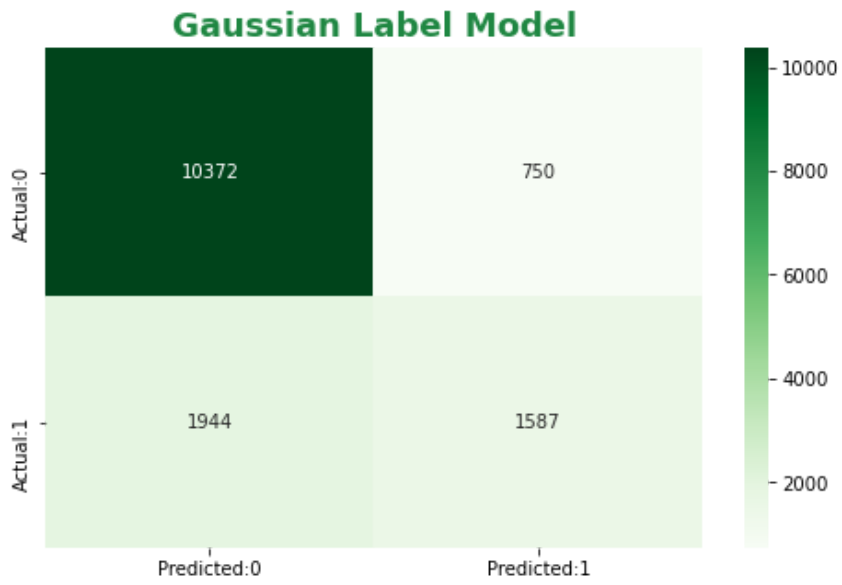
	Model	Accuracy	Precision	Recall	F1_score
0	Gaussian Dummy Model	0.615232	0.376683	0.911357	0.533046

From above we can say that our model is able to correctly classify around 91% guests with >=50K.

In [45]:
```
# Let's Check the classification report of the model
print(classification_report(y_test,dummy_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.52      0.67     11122
           1       0.38      0.91      0.53      3531

    accuracy                           0.62     14653
   macro avg       0.66      0.72      0.60     14653
weighted avg       0.81      0.62      0.64     14653
```

In [46]:
```
lcm=confusion_matrix(y_test,label_pred)
lconf_matrix=pd.DataFrame(data=lcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(lconf_matrix, annot=True,fmt='d',cmap='Greens')
fm={'size':18,'color':'#1f8742','weight':'bold'}
plt.title('Gaussian Label Model',**fm)
plt.show()
```



In [47]:
```
TN=lcm[0,0]
TP=lcm[1,1]
FN=lcm[1,0]
FP=lcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 10372
True Positive 1587
False Negative 1944
False Positive 750
Sensitivity 0.4494477485131691
Specificity 0.9325660852364682
```

The model is more specific than sensitive means it predicts target *<50K* more accurately.

In [48]:
```
label_eval_data=insert_data(y_test,label_pred,'Gaussian Label Model')
pd.DataFrame(label_eval_data)
```

Out[48]:

	Model	Accuracy	Precision	Recall	F1_score
0	Gaussian Label Model	0.816147	0.679076	0.449448	0.5409

While label model is only able to classify around 44.95% of guests >50K.

In [49]:
```
# Let's Check the classification report of the model
print(classification_report(y_test,label_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.93      0.89     11122
           1       0.68      0.45      0.54      3531

    accuracy                           0.82     14653
   macro avg       0.76      0.69      0.71     14653
weighted avg       0.80      0.82      0.80     14653
```

In [50]:
```
eval_data=append_data(label_eval_data,dummy_eval_data)
```

Question-6: Apply Bernoulli Naive Bayes on both the datasets and evaluate them again.

In [51]:
```
from sklearn.naive_bayes import BernoulliNB
```

- Dummy Data

In [52]:
```
dummy_gnb=BernoulliNB(alpha=2)
dummy_gnb.fit(X_drain,y_drain)
```

Out[52]:
```
BernoulliNB(alpha=2, binarize=0.0, class_prior=None, fit_prior=True)
```

In [53]:
```
dummy_pred=dummy_gnb.predict(X_dest)
```

- Label Data

In [54]:
```
label_gnb=BernoulliNB(alpha=2)
label_gnb.fit(X_train,y_train)
```

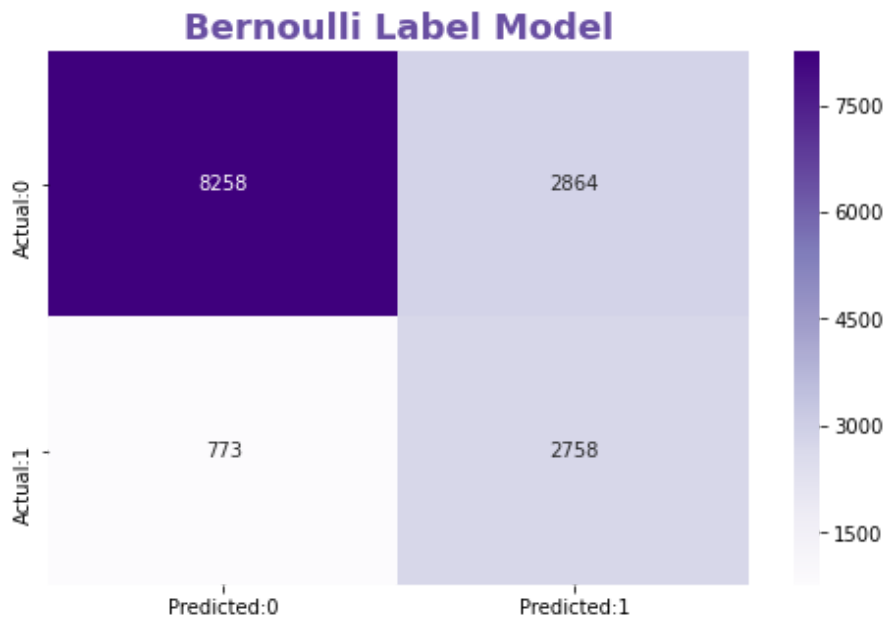
Out[54]:
```
BernoulliNB(alpha=2, binarize=0.0, class_prior=None, fit_prior=True)
```

In [55]:
```
label_pred=label_gnb.predict(X_test)
```

In [56]:
```
print('The accuracy of the bernoulli dummy model is: ',round(accuracy_score(y_dest,dummy_pred)*100,2))
print('The accuracy of the bernoulli label model is: ',round(accuracy_score(y_test,label_pred)*100,2))
```

```
The accuracy of the bernoulli dummy model is:  75.18
The accuracy of the bernoulli label model is:  73.41
```

In [57]:
```
dcm=confusion_matrix(y_dest,dummy_pred)
dconf_matrix=pd.DataFrame(data=dcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(dconf_matrix, annot=True,fmt='d',cmap='Purples')
fm={'size':18,'color':'#694fa2','weight':'bold'}
plt.title('Bernoulli Label Model',**fm)
plt.show()
```



In [58]:
```
TN=dcm[0,0]
TP=dcm[1,1]
FN=dcm[1,0]
FP=dcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 8258
True Positive 2758
False Negative 773
False Positive 2864
Sensitivity 0.7810818465024072
Specificity 0.7424923574896601
```

In [59]:
```
bern_dummy_eata=insert_data(y_test,dummy_pred,'Bernoulli Dummy Model')
pd.DataFrame(bern_dummy_eata)
```

Out[59]:

	Model	Accuracy	Precision	Recall	F1_score
0	Bernoulli Dummy Model	0.751791	0.490573	0.781082	0.602644

The model is able to correctly classify 78% of positive data i.e. >=50K.

In [60]:
```
print(classification_report(y_test,dummy_pred))
```

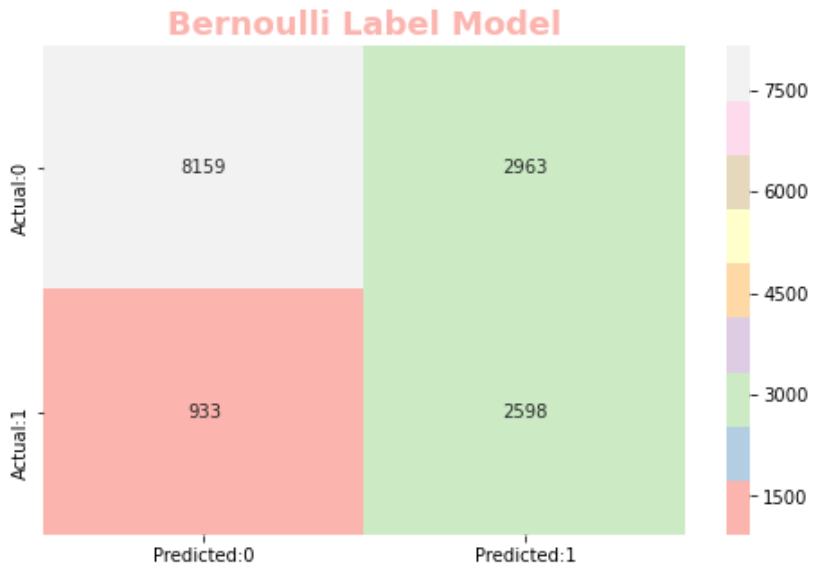
```
              precision    recall  f1-score   support

           0       0.91      0.74      0.82     11122
           1       0.49      0.78      0.60      3531

    accuracy                           0.75     14653
   macro avg       0.70      0.76      0.71     14653
weighted avg       0.81      0.75      0.77     14653
```

In [61]:
```
eval_data=append_data(bern_dummy_eata,eval_data)
```

The model is highly specific than sensitive means it predicts target *<50K* more accurately.

In [62]:
```
def format_text(size,color,weight='normal'):
    return {'size':size,'color':'#'+color,'weight':weight}
```

In [63]:
```
lcm=confusion_matrix(y_test,label_pred)
lconf_matrix=pd.DataFrame(data=lcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(lconf_matrix, annot=True,fmt='d',cmap='Pastel1')
fm={'size':18,'color':'#fbb4ae','weight':'bold'}
plt.title('Bernoulli Label Model',**fm)
plt.show()
```



In [64]:
```
TN=lcm[0,0]
TP=lcm[1,1]
FN=lcm[1,0]
FP=lcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 8159
True Positive 2598
False Negative 933
False Positive 2963
Sensitivity 0.735768903993203
Specificity 0.733591080740874
```

The model is both specific and sensitive.

In [65]:
```
bern_label_model=insert_data(y_test,label_pred,'Bernoulli Label Model')
pd.DataFrame(bern_label_model)
```

Out[65]:

	Model	Accuracy	Precision	Recall	F1_score
0	Bernoulli Label Model	0.734116	0.467182	0.735769	0.571491

In [66]:
```
eval_data=append_data(bern_label_model,eval_data)
```

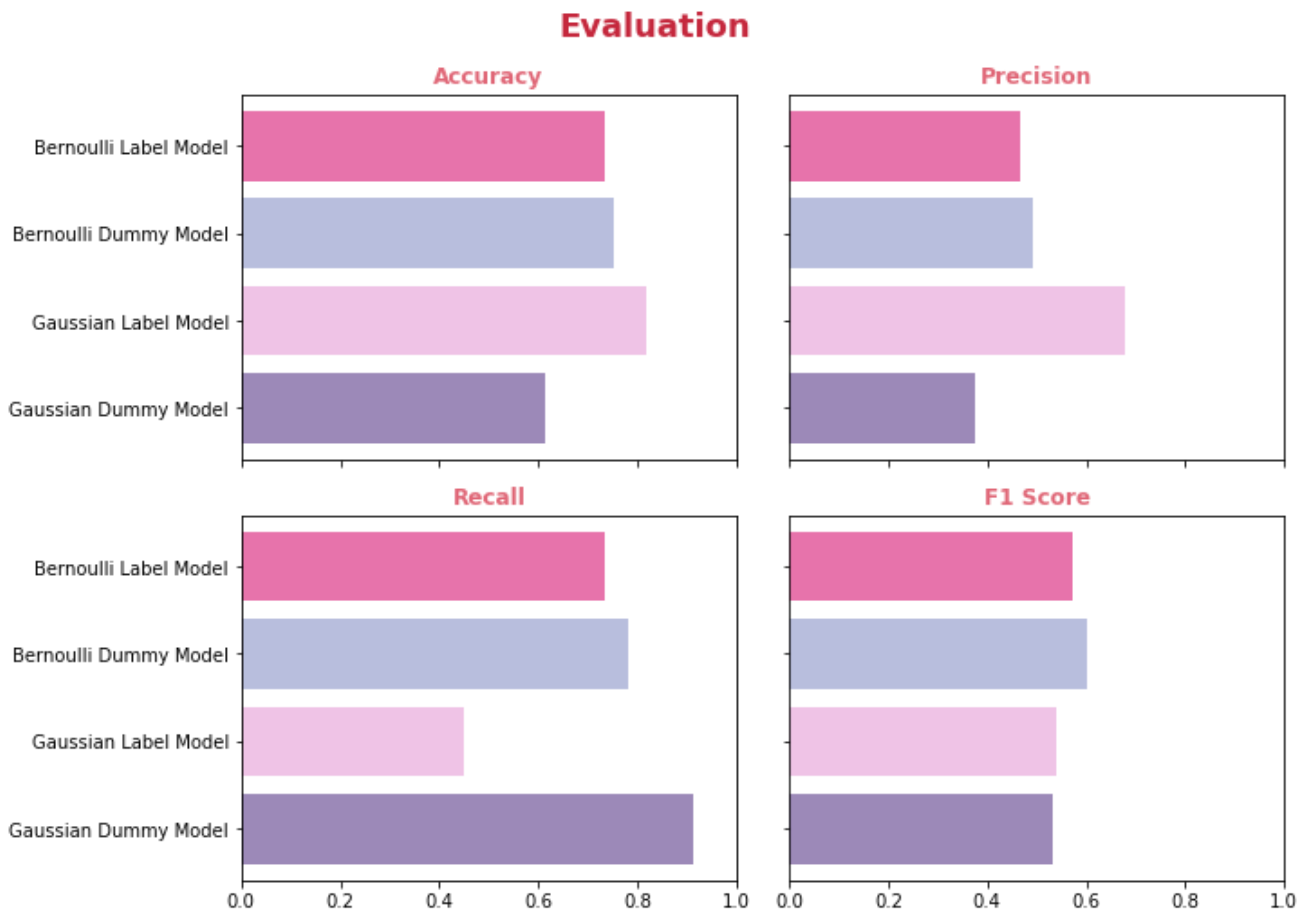
In [67]:
```
pd.DataFrame(eval_data)
```

Out[67]:

	Model	Accuracy	Precision	Recall	F1_score
0	Gaussian Dummy Model	0.615232	0.376683	0.911357	0.533046
1	Gaussian Label Model	0.816147	0.679076	0.449448	0.540900
2	Bernoulli Dummy Model	0.751791	0.490573	0.781082	0.602644
3	Bernoulli Label Model	0.734116	0.467182	0.735769	0.571491

In [68]:
```
from matplotlib.gridspec import GridSpec
def plot_models(data):
sns.set_palette(sns.color_palette("rocket"))
super_title={'size':18,'color':'#c5283d','weight':'bold'}
sub_title={'size':12,'color':'#e06777','weight':'bold'}
colors=np.array([[156, 137, 184],[239, 195, 230],[184, 190, 221],[231, 115, 171]])
colors=colors/255 #Matplotlib RGB color range is from 0-1
data=pd.DataFrame(data)
fig = plt.figure(figsize=(10,7),constrained_layout=True)
gs = GridSpec(2, 2, figure=fig)
ax1 = fig.add_subplot(gs[0, 0])
ax1.barh(data.Model,data.Accuracy,color=colors)
ax1.tick_params(labelbottom=False, labelleft=True)
ax1.set_xlim(0,1)
ax1.set_title('Accuracy',**sub_title)
ax2 = fig.add_subplot(gs[0, 1])
ax2.barh(data.Model,data.Precision,color=colors)
ax2.tick_params(labelbottom=False, labelleft=False)
ax2.set_xlim(0,1)
ax2.set_title('Precision',**sub_title)
ax3 = fig.add_subplot(gs[1, 0])
ax3.barh(data.Model,data.Recall,color=colors)
ax3.tick_params(labelbottom=True, labelleft=True)
ax3.set_xlim(0,1)
ax3.set_title('Recall',**sub_title)
ax4 = fig.add_subplot(gs[1, 1])
ax4.barh(data.Model,data.F1_score,color=colors)
ax4.tick_params(labelbottom=False, labelleft=False)
ax4.set_xlim(0,1)
ax4.set_title('F1 Score',**sub_title)
fig.suptitle("Evaluation",**super_title)
ax4.tick_params(labelbottom=True, labelleft=False)
plt.show()
```

In [69]:
```
plot_models(eval_data)
```



From the above graphs, we can determine the best model. Overall, Gaussian Dummy Model has best recall but accuracy is pretty low as well as precision score. Even Gaussian Label Model has best accuracy but the truth is that the model is performing really bad as recall is worst. On an average sense, we can see that our Bernoulli Label model is performing well enough to be selected.

Conclusion: Even though the dummy Bernoulli model produced better accuracy it would not appreciated if it used as it is highly specific. As Bernoulli label model has a good accuracy of 73.41%, it would best if this is used. As discussed in the module Gaussian Naive Bayes model is always best used when data is continous and normally distributed which is why dummy Gaussian naive bayes failed to produce better accuracy than label Gaussian model. Bernoulli model is best used when we have independent features and have a binary outcome.

Scenario-2: Gnar Automobiles

Gnar Automobiles engages in the distribution and sale of automobiles and light commercial vehicles. The owner of the Gnar Automobiles deals with a number of distributors across countries in different origins.

Problem Statement

As every orgin sends cars with various specifications. The owner wants to determine the origin of the cars based on the specifications of the cars to further increase business opportunities.

Dataset Description

"The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes." Attribute Information:

- **mpg**: continuous
- **cylinders**: multi-valued discrete
- **displacement**: continuous
- **horsepower**: continuous
- **weight**: continuous
- **acceleration**: continuous
- **model year**: multi-valued discrete
- **origin**: multi-valued discrete
- **car name**: string (unique for each instance)

Tasks to be Performed:

In order to attain the above goal below tasks must be performed:

- Read the dataset and process the missing values in each column. Use pandas_profiling to get a report of the data. - **Beginner**
- Scale the data using MinMaxScaler and plot the boxplot before and after scaling the data. - **Beginner**
- Explore the data relations with target variable *origin*. - **Intermediate**
- Split the data into training and testing set and apply KNN model with k=3,9,12. - **Intermediate**
- Evaluate the model and find the accuracy score. - **Intermediate**
- Plot the visualizations for the models. - **Advanced**

Topics Covered:

- K-Nearest Neighbor

Importing Required Libraries

In [70]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from mlxtend.plotting import plot_decision_regions
%matplotlib inline
```

Question-1: Read the dataset and process the missing values in each column. Use pandas_profiling to get a report of the data.

In [139]:
```
data=pd.read_csv('auto-mpg.csv')
```

In [140]:
```
data.head()
```

Out[140]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino

In [141]:
```
data.describe(include='all')
```

Out[141]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000	398.000000	398
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	305
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	ford pinto
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050	1.572864	NaN
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627	0.802055	NaN
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000	NaN
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000	1.000000	NaN
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000	1.000000	NaN
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000	2.000000	NaN
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000	NaN

In [142]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[142]:

	Col_name	Missing value?	Count_	Percentage
3	horsepower	True	6	0.015075
0	mpg	False	0	0.000000
1	cylinders	False	0	0.000000
2	displacement	False	0	0.000000
4	weight	False	0	0.000000
5	acceleration	False	0	0.000000
6	model year	False	0	0.000000
7	origin	False	0	0.000000
8	car name	False	0	0.000000

In [143]:
```
print('Total Missing values: %s'%sum(miss.Count_))
```

```
Total Missing values: 6
```

In [144]:
```
data.horsepower=data.horsepower.fillna(data.horsepower.mean())
```

In [145]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[145]:

	Col_name	Missing value?	Count_	Percentage
0	mpg	False	0	0.0
1	cylinders	False	0	0.0
2	displacement	False	0	0.0
3	horsepower	False	0	0.0
4	weight	False	0	0.0
5	acceleration	False	0	0.0
6	model year	False	0	0.0
7	origin	False	0	0.0
8	car name	False	0	0.0

In [146]:
```
import pandas_profiling
```

In [147]:
```
rpt=pandas_profiling.ProfileReport(data)
```

In [148]:
```
rpt
```

Overview

Dataset statistics

Number of variables	9
Number of observations	398
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	28.1 KiB
Average record size in memory	72.3 B

Variable types

NUM	7
CAT	2

Reproduction

Analysis started	2020-07-14 14:25:21.444493
Analysis finished	2020-07-14 14:25:41.407574
Duration	19.96 seconds
Version	pandas-profiling v2.8.0 (https://github.com/pandas-profiling/pandas-profiling)
Command line	`pandas_profiling --config_file config.yaml [YOUR_FILE.csv]`

Out[148]:

ProfileReport functions generate a report of dataset which includes missing values, correlations and other important functions required for data exploration and understanding.

Question-2: Scale the data using MinMaxScaler and plot the boxplot before and after scaling the data.

In [149]:
```
feat=data.columns
feat=feat.drop(['car name', 'origin'])
```

- **IQR Score**: Measure of spread, IQR = Q3 − Q1

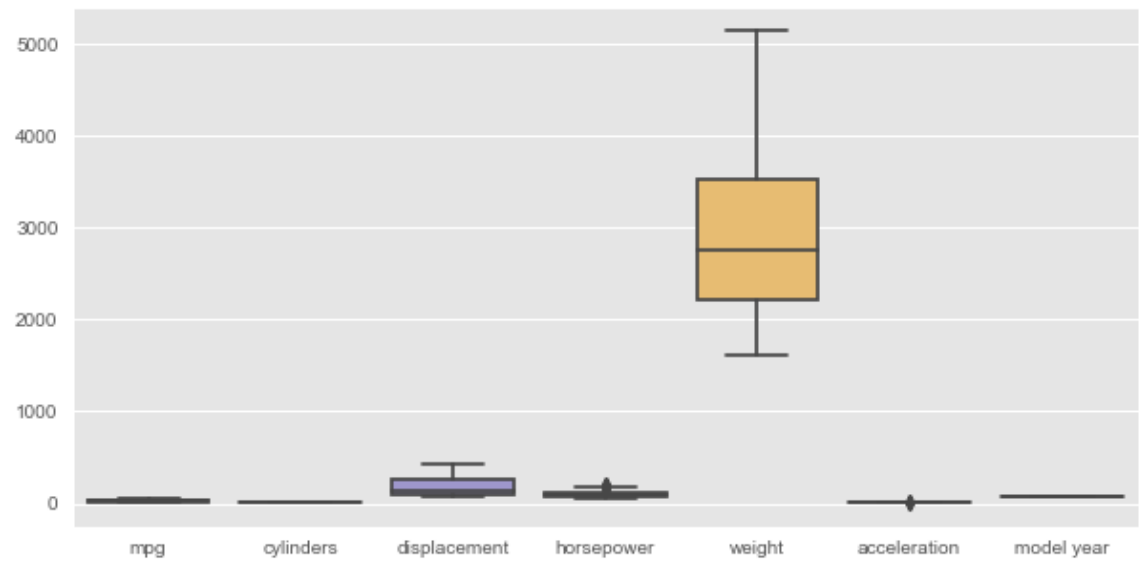
In [150]:
```
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
mpg              11.50
cylinders         4.00
displacement    157.75
horsepower       49.00
weight         1384.25
acceleration      3.35
model year        6.00
origin            1.00
dtype: float64
```

In [151]:
```
data_o=data[~((data < (Q1 - 1.5 * IQR)) |(data > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [152]:
```
X=data_o.drop(['car name', 'origin'],axis=1)
y=data_o.origin
```

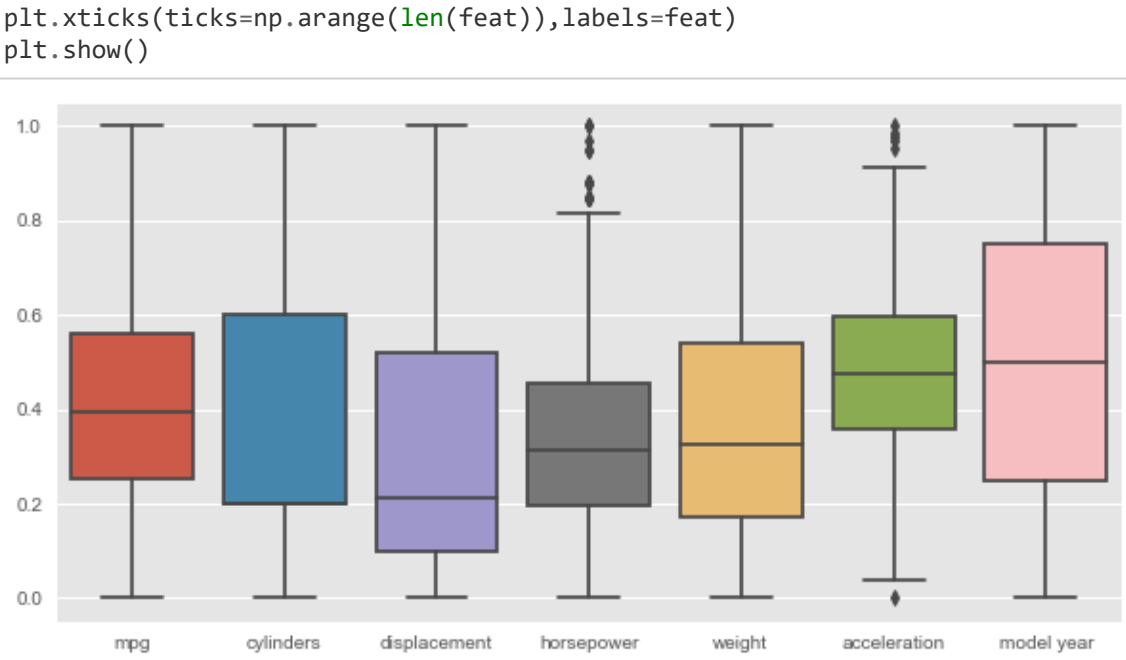
In [153]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,5))
sns.boxplot(data=X)
plt.xticks(ticks=np.arange(len(feat)),labels=feat)
plt.show()
```



In [154]:
```
# Scaling data after outlier removal
scl=MinMaxScaler()
```

In [155]:
```
X=scl.fit_transform(X)
```

In [156]:
```
plt.figure(figsize=(10,5))
sns.boxplot(data=X)
plt.xticks(ticks=np.arange(len(feat)),labels=feat)
plt.show()
```

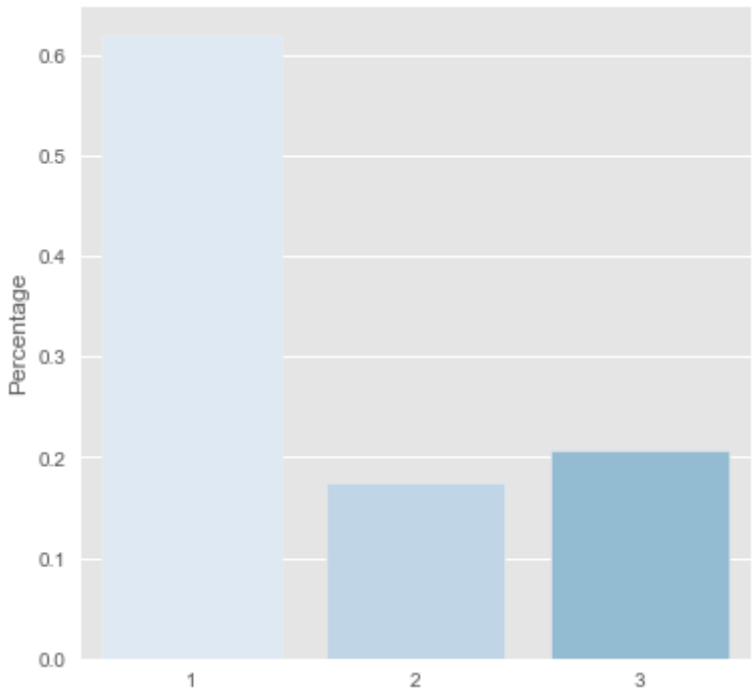


Question-3: Explore the data relations with target variable *origin*.

In [157]:
```
Target_ratio=y.value_counts()/len(y)
print(Target_ratio)
```

```
1    0.619048
3    0.206349
2    0.174603
Name: origin, dtype: float64
```

In [158]:
```
## Checking for data unbalance
plt.figure(figsize = (6,6))
sns.set_palette(sns.color_palette("Blues"))
sns.barplot(Target_ratio.index,Target_ratio,)
plt.ylabel('Percentage')
plt.show()
```

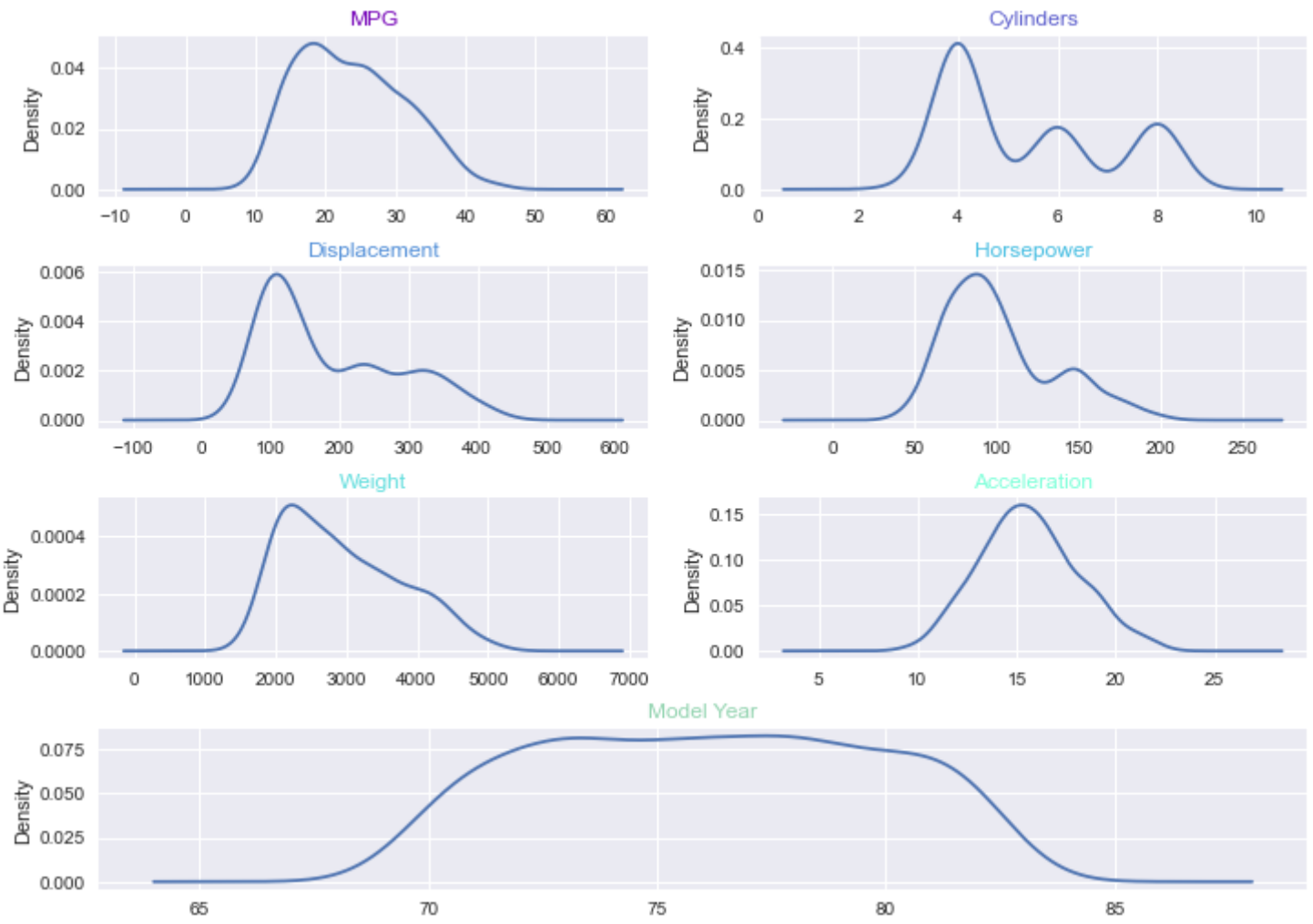


In [159]:
```
data_o.head()
```

Out[159]:

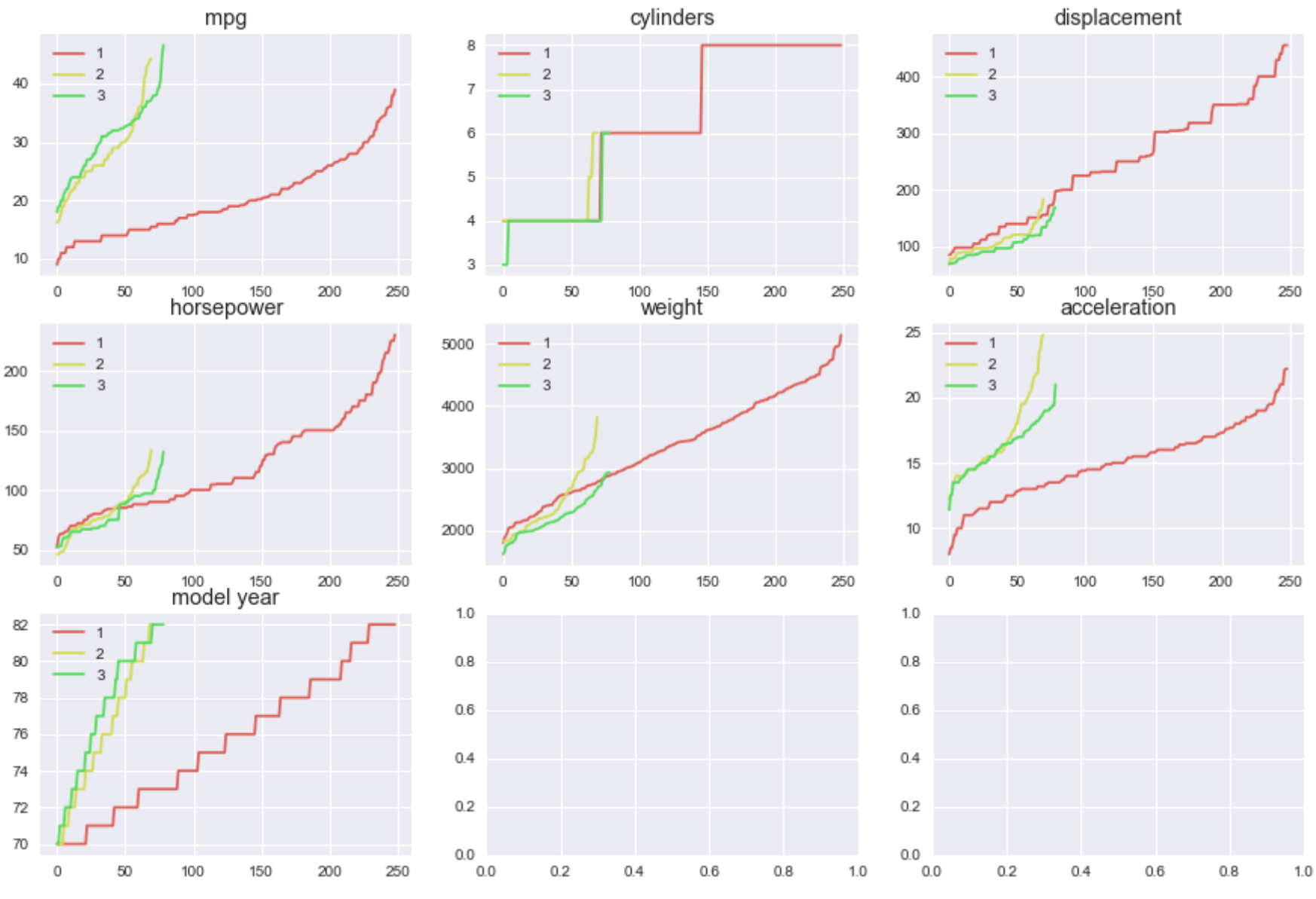
	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino

In [160]:
```
# KDE Plots
from matplotlib.gridspec import GridSpec
plt.style.use('seaborn')
fig = plt.figure(figsize=(10,7),constrained_layout=True)
gs = GridSpec(4, 2, figure=fig)
ax1 = fig.add_subplot(gs[0, 0])
data_o.mpg.plot.kde()
ax1.set_title('MPG',**format_text(12,'7400b8'))

ax2 = fig.add_subplot(gs[0,1])
data_o.cylinders.plot.kde()
ax2.set_title('Cylinders',**format_text(12,'5e60ce'))

ax3 = fig.add_subplot(gs[1,0])
data_o.displacement.plot.kde()
ax3.set_title('Displacement',**format_text(12,'5390d9'))


ax4 = fig.add_subplot(gs[1,1])
data_o.horsepower.plot.kde()
ax4.set_title('Horsepower',**format_text(12,'48bfe3'))

ax5 = fig.add_subplot(gs[2,0])
data_o.weight.plot.kde()
ax5.set_title('Weight',**format_text(12,'64dfdf'))

ax6 = fig.add_subplot(gs[2,1])
data_o.acceleration.plot.kde()
ax6.set_title('Acceleration',**format_text(12,'80ffdb'))

ax7 = fig.add_subplot(gs[3,:])
data_o['model year'].plot.kde()
ax7.set_title('Model Year',**format_text(12,'95d5b2'))
plt.show()
```



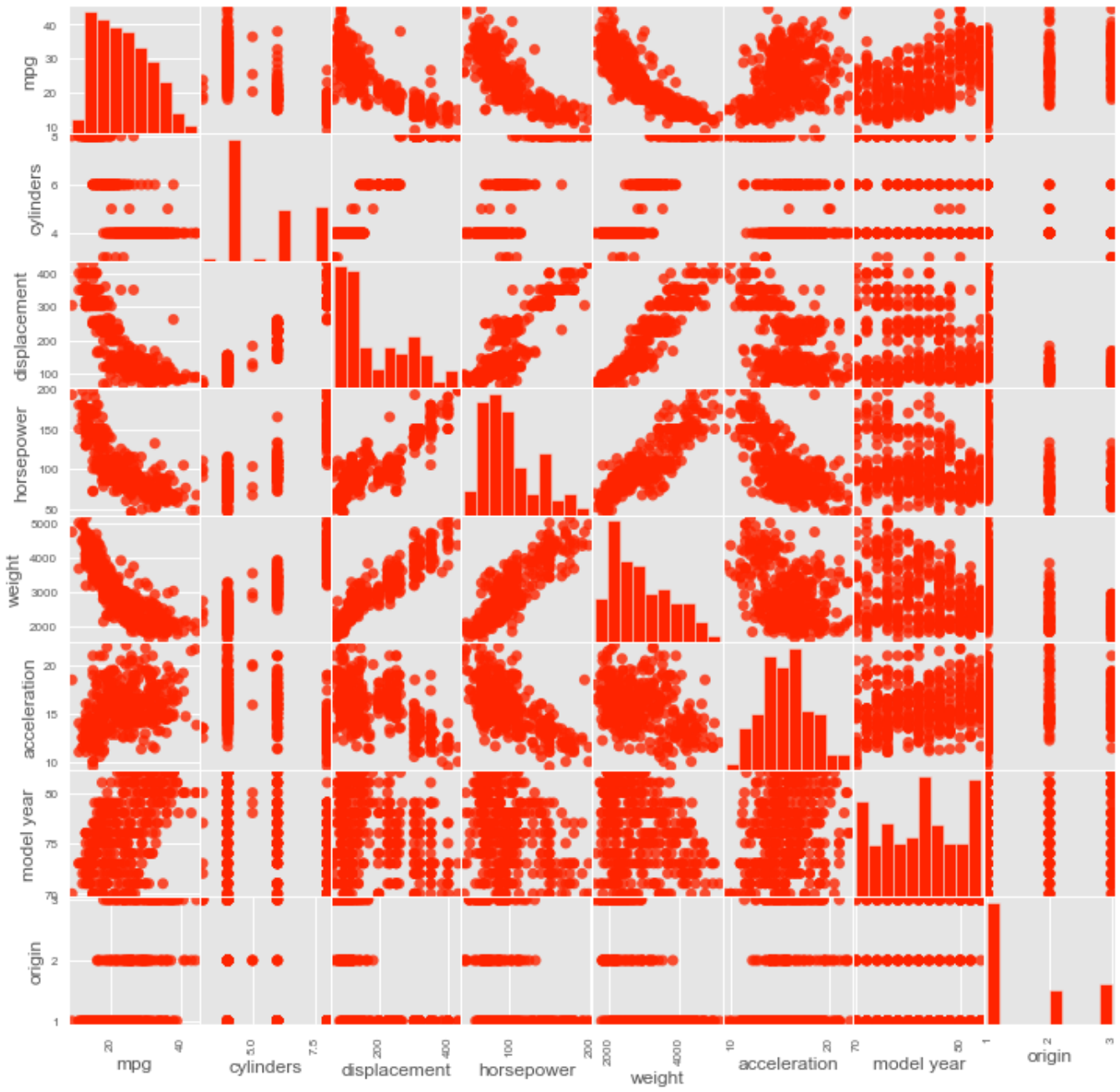
From the above plot we can see that data is not normal in nature.

In [161]:
```
sns.set_palette(sns.color_palette('hls'))
fig,ax=plt.subplots(3,3,figsize=(15,10))
col=data.columns
q=0
plt.style.use('ggplot')
for i in ax:
    for j in i:
        for grp,tata in data.groupby('origin'):
            j.set_title(col[q])
            j.plot(sorted(tata[col[q]]))
            j.legend(np.arange(3)+1)
        if q==6:
            break
        q+=1
```



From the above data we can clearly see that origin 1 has high stats for every feature while origin 3 and 2 have pretty low stats.

In [162]:
```
# Pairplots
plt.style.use('fast')
sns.set_palette(sns.color_palette('autumn'))
pd.plotting.scatter_matrix(data_o, figsize=(12,12), marker = 'o', hist_kwds = {'bins': 10}, alpha = 0.8)
plt.show()
```



With the above graph, we can see the distribution of features as well as the relation between them. From the above graph, we can see that the weight is proportional to displacement and horsepower but inversely proportional with mpg. We can derive that the data is left skewed from the above graph.

Question-4: Apply KNN model with k=3,9,12.

In [95]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=0, test_size=0.3)
```

In [96]:
```
knn9=KNeighborsClassifier(n_neighbors=9)
knn12=KNeighborsClassifier(n_neighbors=12)
knn3=KNeighborsClassifier(n_neighbors=3)
knn3.fit(X_train,y_train)
knn12.fit(X_train,y_train)
knn9.fit(X_train,y_train)
knn3_pred=knn3.predict(X_test)
knn12_pred=knn12.predict(X_test)
knn9_pred=knn9.predict(X_test)
```

Question-5: Evaluate the models and find the accuracy.

In [97]:
```
print('Accuracy score when k=3:',accuracy_score(y_test,knn3_pred)*100)
print('Accuracy score when k=9:',accuracy_score(y_test,knn9_pred)*100)
print('Accuracy score when k=12:',accuracy_score(y_test,knn12_pred)*100)
```

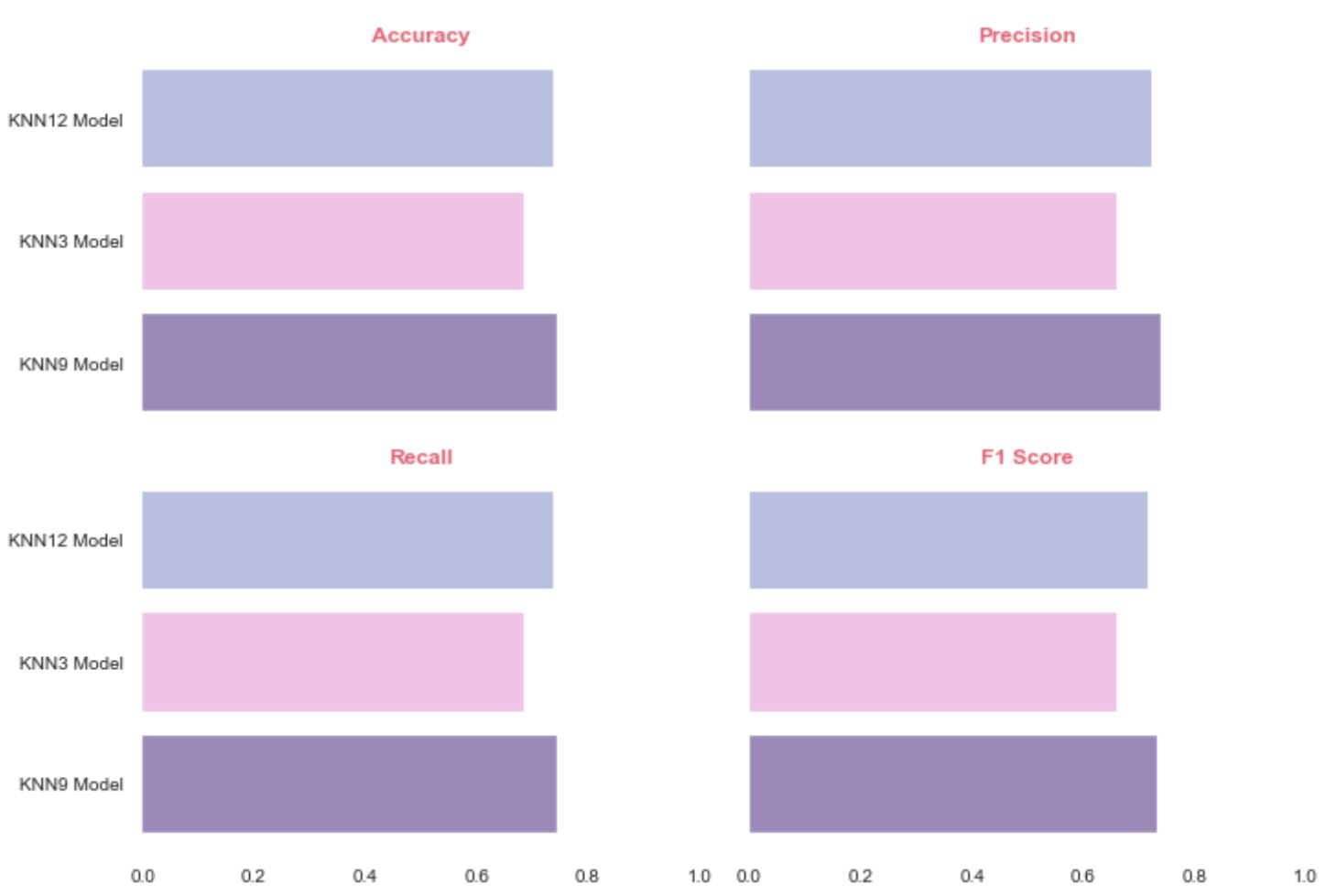
```
Accuracy score when k=3: 68.42105263157895
Accuracy score when k=9: 74.56140350877193
Accuracy score when k=12: 73.68421052631578
```

In [98]:
```
# Let's see precision and recall values
#Insert data in evaluation dictionary
def insert_data_k(test,pred,model,average):
    eval_data=initialize_evaluator()
    eval_data['Model'].append(model)
    eval_data['Accuracy'].append(accuracy_score(test,pred))
    eval_data['Precision'].append(precision_score(test,pred,average=average))
    eval_data['Recall'].append(recall_score(test,pred,average=average))
    eval_data['F1_score'].append(f1_score(test,pred,average=average))
    return eval_data
```

In [99]:
```
# Check prcision score from sklearn to learn more about average
eval3=insert_data_k(y_test,knn3_pred,'KNN3 Model','weighted')
eval9=insert_data_k(y_test,knn9_pred,'KNN9 Model','weighted')
eval12=insert_data_k(y_test,knn12_pred,'KNN12 Model','weighted')
```

In [100]:
```
eval_knn=append_data(eval3,eval9)
eval_knn=append_data(eval12,eval_knn)
```

In [101]:
```
plot_models(eval_knn)
```



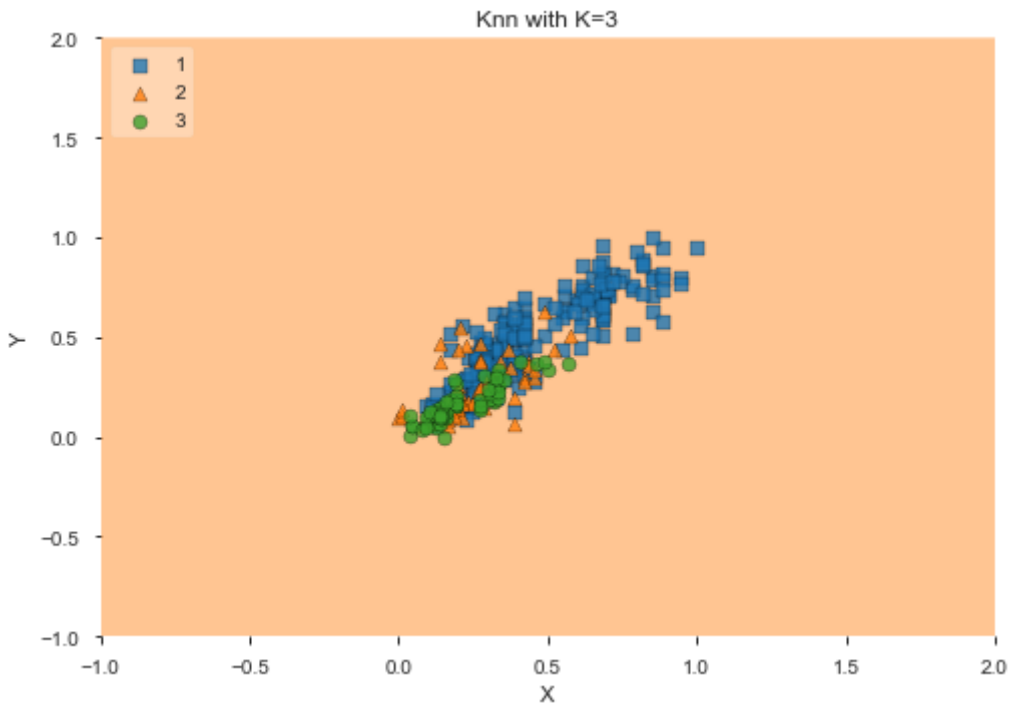
From the above scores we can see that KNN12 has been performing well enough.

Question-6: Plot the visualizations for the models.

plot_decision_regions function for plotting decision regions of classifiers in 1 or 2 dimensions. Since we have 7 dimensions, we can only plot for at most 2 dimensions at a time so we will be using attributes feature_index, filler_feature_ranges and filler_feature_values to compensate.

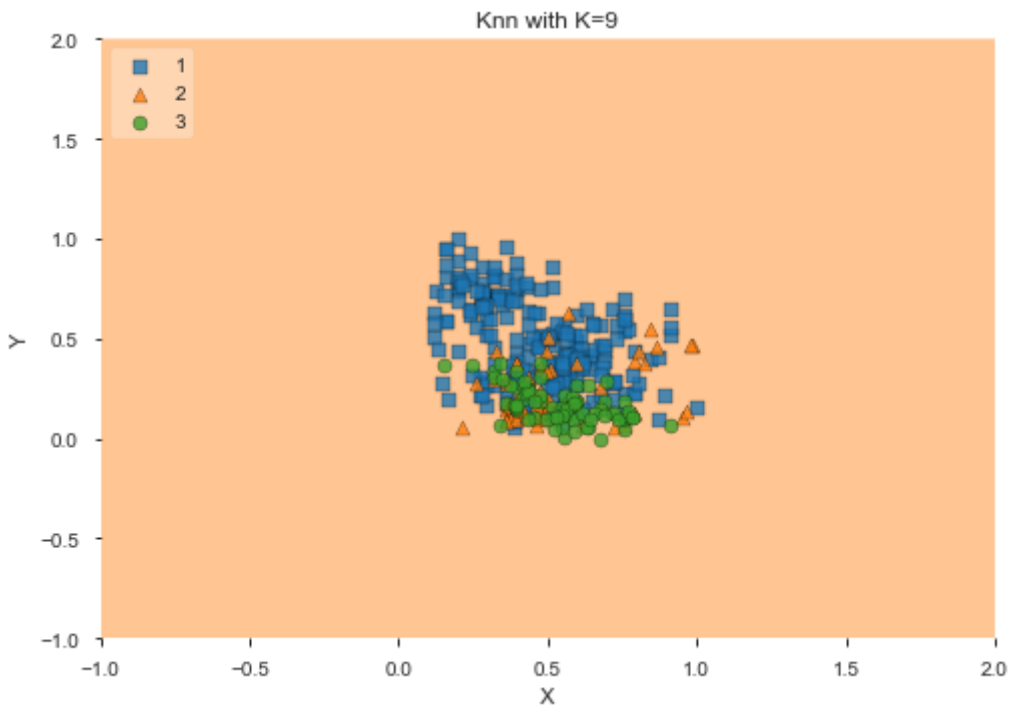
In [102]:
```
value=4
width=4
plot_decision_regions(np.array(X), np.array(y), clf=knn3, legend=2,feature_index=[3,4],
              filler_feature_values={6: value, 1:value,2:value,5:value,0:value},
              filler_feature_ranges={6: width, 1: width,2: width,5: width,0: width})
# Adding axes annotations
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Knn with K='+ str(3))
plt.show()
```

```
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:247: UserWarning: No contour levels were found within the data range.
  antialiased=True)
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:249: MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis() will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



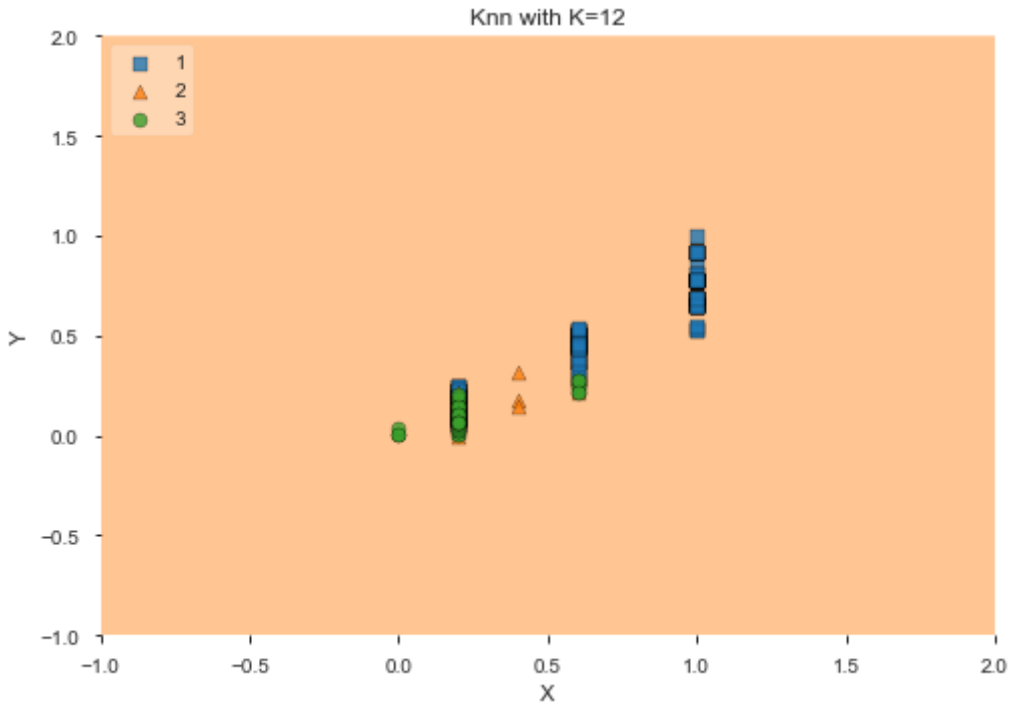
In [103]:
```
value=4
width=4
plot_decision_regions(np.array(X), np.array(y), clf=knn9, legend=2,feature_index=[5,4],
              filler_feature_values={6: value, 1:value,2:value,3:value,0:value},
              filler_feature_ranges={6: width, 1: width,2: width,3: width,0: width})
# Adding axes annotations
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Knn with K='+ str(9))
plt.show()
```

```
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:247: UserWarning: No contour levels were found within the data range.
  antialiased=True)
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:249: MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis() will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



In [104]:
```
value=4
width=4
plot_decision_regions(np.array(X), np.array(y), clf=knn12, legend=2,feature_index=[1,2],
                      filler_feature_values={6: value, 3:value,4:value,5:value,0:value},
                      filler_feature_ranges={6: width, 3: width,4: width,5: width,0: width})
# Adding axes annotations
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Knn with K='+ str(12))
plt.show()
```

```
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:247: UserWarning: No contour levels were found within the data range.
  antialiased=True)
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:249: MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis() will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



Scenario-3: Ergo Plastics

Ergo plastics was found in 1973, one of the oldest and largest plastic manufacturing industry across the world. They have been developing in a new plastic materials which can withstand high pressure and temperature. They have created a 2 new type of materials and based on their elasticity and ductility the data has been collected. The data has been collected based on the material strength.

Problem Statement:

They have collected the data in a csv format. Task is to classify a given piece of plastic based on it's strength.

Dataset Description:

Attributes:

- **elasticity**: real
- **ductility**: real

Target Variable:

- **type**: tensil, brittle

Tasks to be Performed:

In order to attain the above goal below tasks must be performed:

- Read the dataset and process all the missing values. - **Beginner**
- Split the data in training and testing set; then apply support vector classifier. - **Beginner**
- Find the accuracy of the model and plot the confusion matrix. - **Intermediate**
- Plot the visualization of the model to check the boundaries. - **Advanced**

Topics Covered:

- Support Vector Machine

Question-1: Read the dataset and process all the missing values.

In [105]:
```
import pandas as pd
import numpy as np
```

In [106]:
```
data=pd.read_csv('plastics_type.csv')
```

In [107]:
```
data.head()
```

Out[107]:

	elasticity	ductility	type
0	29.19393	8.443208	tensil
1	28.04907	7.237035	tensil
2	26.90421	7.719504	tensil
3	26.33178	7.478270	tensil
4	28.62150	8.684442	tensil

In [108]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 155 entries, 0 to 154
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   elasticity  154 non-null    float64
 1   ductility   152 non-null    float64
 2   type        155 non-null    object 
dtypes: float64(2), object(1)
memory usage: 3.8+ KB
```

In [109]:
```
data.describe(include='all')
```

Out[109]:

	elasticity	ductility	type
count	154.000000	152.000000	155
unique	NaN	NaN	2
top	NaN	NaN	brittle
freq	NaN	NaN	79
mean	33.427861	7.357653	NaN
std	4.702495	1.049866	NaN
min	24.614490	4.824690	NaN
25%	29.193930	6.754566	NaN
50%	33.200940	7.237035	NaN
75%	36.635520	7.960739	NaN
max	45.221970	10.614318	NaN

- Checking and processin Null Values

In [110]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[110]:

	Col_name	Missing value?	Count_	Percentage
1	ductility	True	3	0.019355
0	elasticity	True	1	0.006452
2	type	False	0	0.000000

In [111]:
```
print('Total Missing values: %s'%sum(miss.Count_))
```

```
Total Missing values: 4
```

In [112]:
```
data.ductility.fillna(data.ductility.mean(),inplace=True)
data.elasticity.fillna(data.elasticity.mean(),inplace=True)
```

In [113]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[113]:

	Col_name	Missing value?	Count_	Percentage
0	elasticity	False	0	0.0
1	ductility	False	0	0.0
2	type	False	0	0.0

Question-2: Split the data in training and testing set; then apply support vector classifier.

In [114]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
```

In [115]:
```
lb=LabelEncoder()
data.type=lb.fit_transform(data.type)
```

In [116]:
```
X=data.drop('type',axis=1)
y=data.type
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=0, test_size=0.3)
```

In [117]:
```
model_rbf=SVC(kernel='rbf')
model_rbf.fit(X_train,y_train)
```

Out[117]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [118]:
```
rbf_pred=model_rbf.predict(X_test)
```

In [119]:
```
model_linear=SVC(kernel='linear')
model_linear.fit(X_train,y_train)
```

Out[119]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [120]:
```
linear_pred=model_linear.predict(X_test)
```

In [121]:
```
model_sigmoid=SVC(kernel='sigmoid')
model_sigmoid.fit(X_train,y_train)
```

Out[121]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [122]:
```
sigmoid_pred=model_sigmoid.predict(X_test)
```

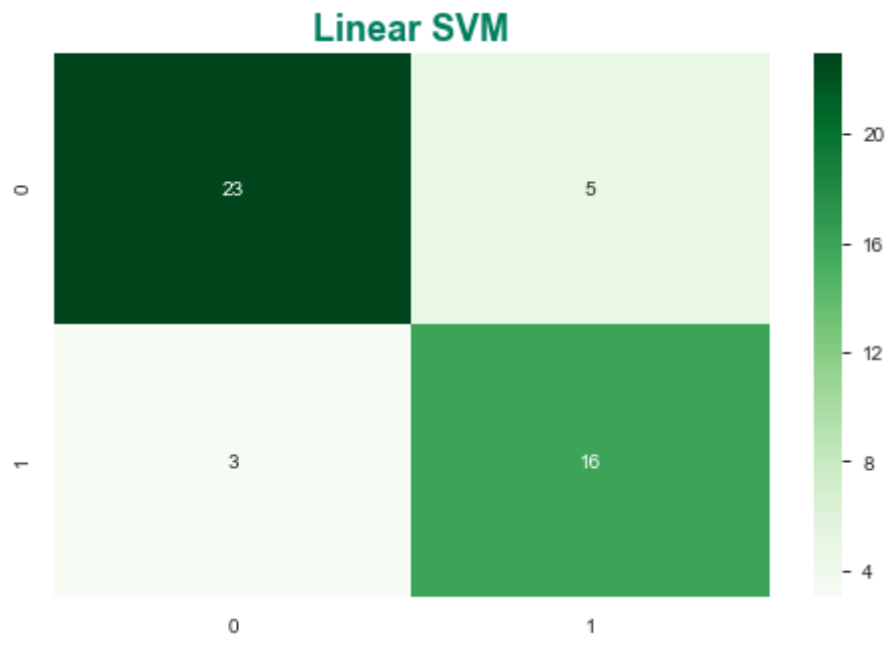
Question-3: Find the accuracy of the model and plot the confusion matrix.

In [123]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap
```

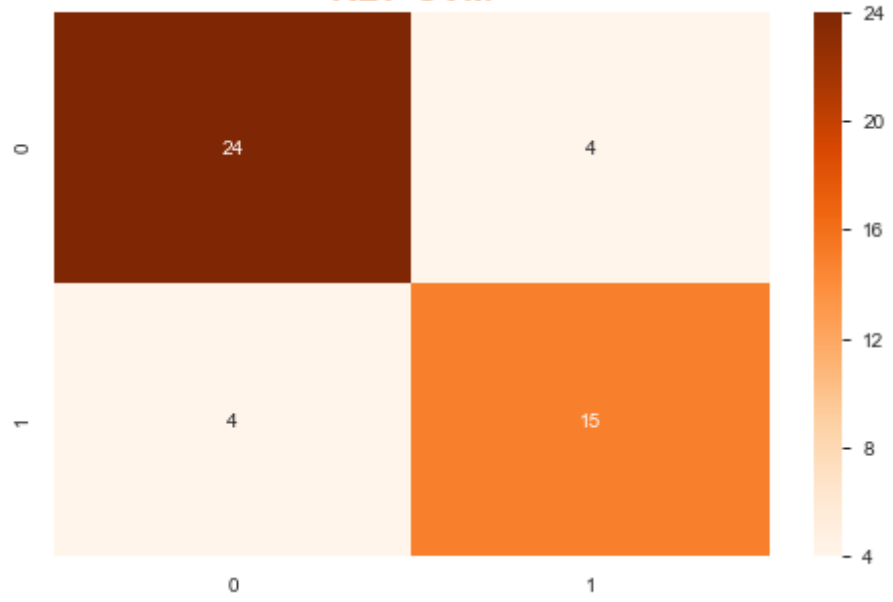
In [124]:
```
print('The accuracy of the linear model is: ',round(accuracy_score(y_test,linear_pred)*100,2))
print('The accuracy of the rbf model is: ',round(accuracy_score(y_test,rbf_pred)*100,2))
print('The accuracy of the sigmoid model is: ',round(accuracy_score(y_test,sigmoid_pred)*100,2))
```

```
The accuracy of the linear model is:  82.98
The accuracy of the rbf model is:  82.98
The accuracy of the sigmoid model is:  40.43
```

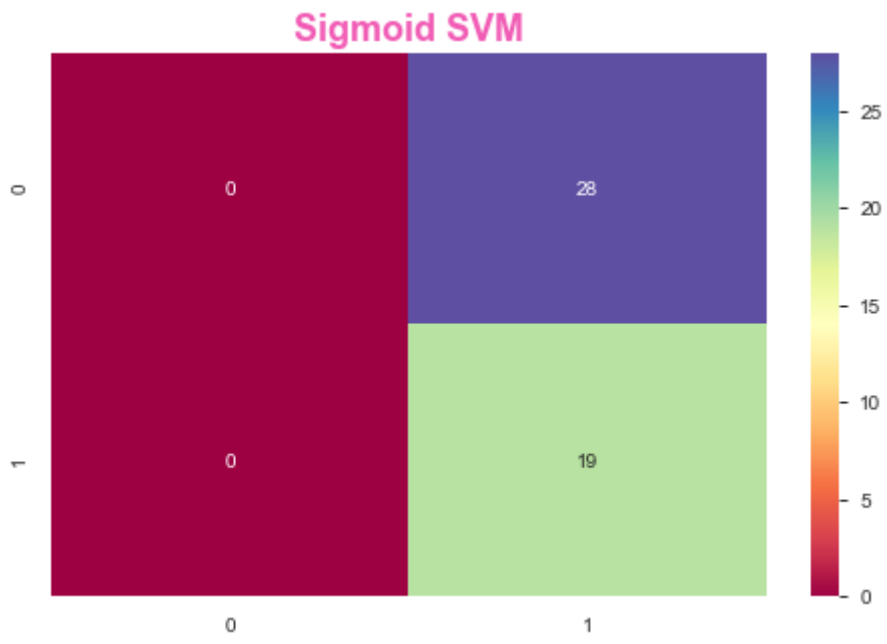
In [125]:
```
lcm=confusion_matrix(y_test,linear_pred)
lconf_matrix=pd.DataFrame(data=lcm)
plt.figure(figsize = (8,5))
sns.heatmap(lconf_matrix, annot=True,fmt='d',cmap='Greens')
plt.title('Linear SVM',**format_text(18,'007F5F','bold'))
plt.show()
```



In [126]:
```
lcm=confusion_matrix(y_test,rbf_pred)
lconf_matrix=pd.DataFrame(data=lcm)
plt.figure(figsize = (8,5))
sns.heatmap(lconf_matrix, annot=True,fmt='d',cmap='Oranges')
plt.title('RBF SVM',**format_text(18,'E8751C','heavy'))
plt.show()
```

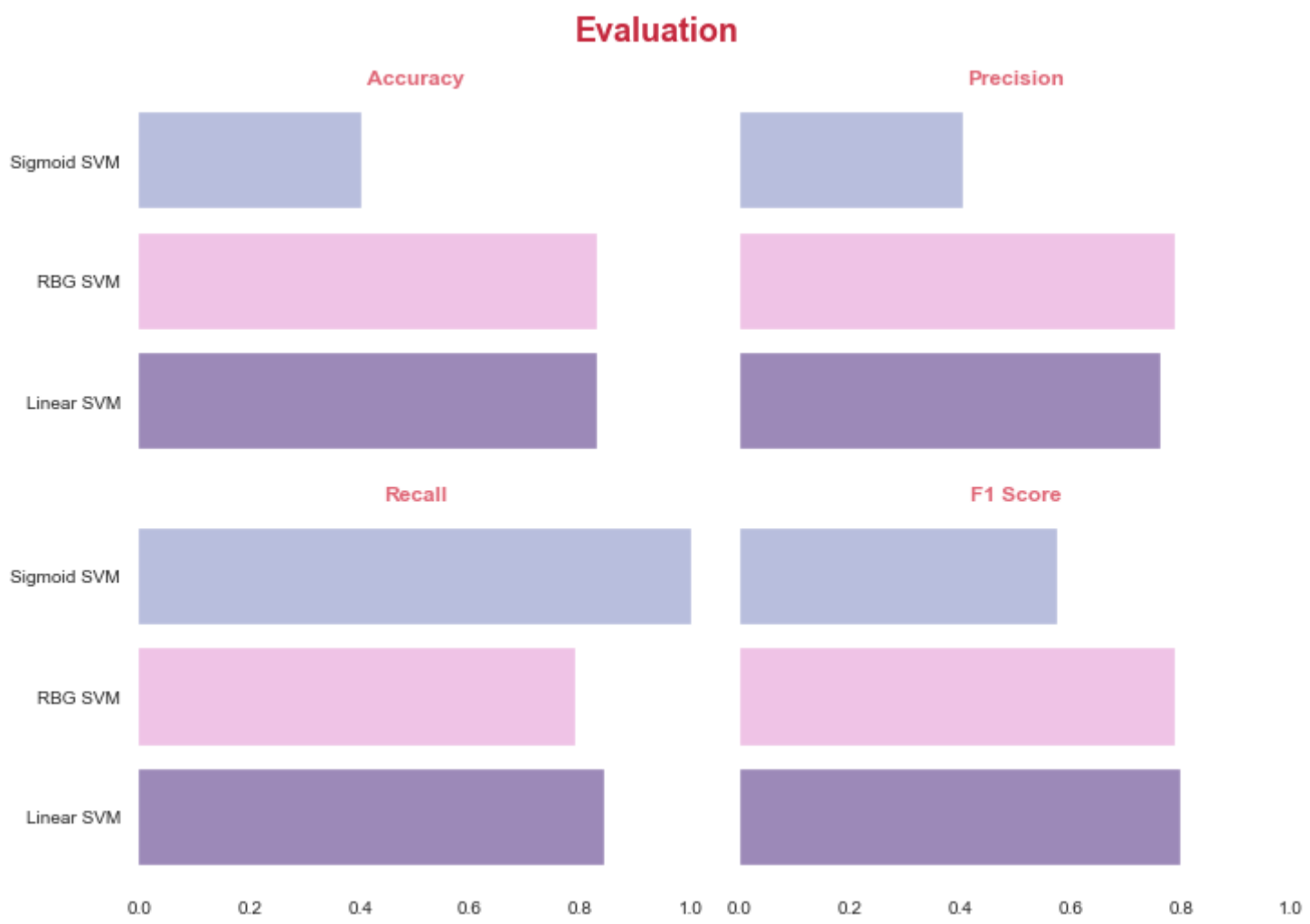


In [127]:
```
lcm=confusion_matrix(y_test,sigmoid_pred)
lconf_matrix=pd.DataFrame(data=lcm)
plt.figure(figsize = (8,5))
sns.heatmap(lconf_matrix, annot=True,fmt='d',cmap='Spectral')
plt.title('Sigmoid SVM',**format_text(18,'F15BB5','heavy'))
plt.show()
```



In [128]:
```
eval_rbf=insert_data(y_test,rbf_pred,'RBG SVM')
eval_lin=insert_data(y_test,linear_pred,'Linear SVM')
eval_sig=insert_data(y_test,sigmoid_pred,'Sigmoid SVM')
eval_svm=append_data(eval_rbf,eval_lin)
eval_svm=append_data(eval_sig,eval_svm)
```

In [129]:
```
plot_models(eval_svm)
```



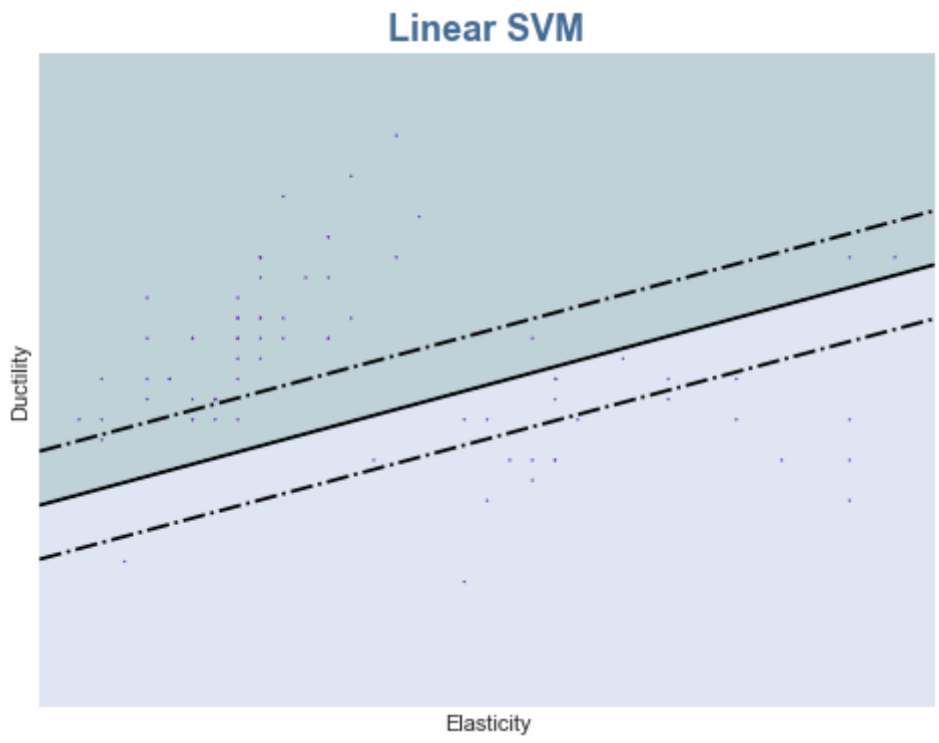
From the above graph we can conclude that Linear SVM model is best.

Question-4: Plot the visualization of the model to check the boundaries.

In [130]:
```
def plot(X,y,model,m):
    p=['tensil','brittle']
    X=np.array(X)
    X1, X2 = np.meshgrid(np.arange(start = X[:, 0].min() - 1, stop = X[:, 0].max() + 1,   step = 0.01),
                         np.arange(start = X[:, 1].min() - 1, stop = X[:, 1].max() + 1, step = 0.01))

    #To plot boundaries
    #In general, the space is divided into decision boundaries
    plt.figure(figsize=(8,6))
    Z=model.decision_function(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape)
    plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
                 alpha = 0.25, cmap = ListedColormap(('#809bce', '#004e64')))
    plt.contour(X1, X2, Z, colors=['k', 'k', 'k'],linestyles=['-.', '-', '-.'],levels=[-.7,0,.7])
    plt.xlim(X1.min(), X1.max())
    plt.ylim(X2.min(), X2.max())
    X=X.ravel()
    clr=[]
    for i in y:
        if i==0:
            clr.append('#ff499e')
        else:
            clr.append('#6f2dbd')
    plt.scatter(X[::2], X[1::2],y,c=clr,cmap= ListedColormap(('#ffaabb', '#23807A')))
    fm={'size':18,'color':'#436B95','weight':'bold'}
    plt.title(m,**fm)
    plt.xlabel('Elasticity')
    plt.ylabel('Ductility')
    plt.xticks([])
    plt.yticks([])
    return plt
```

- Linear

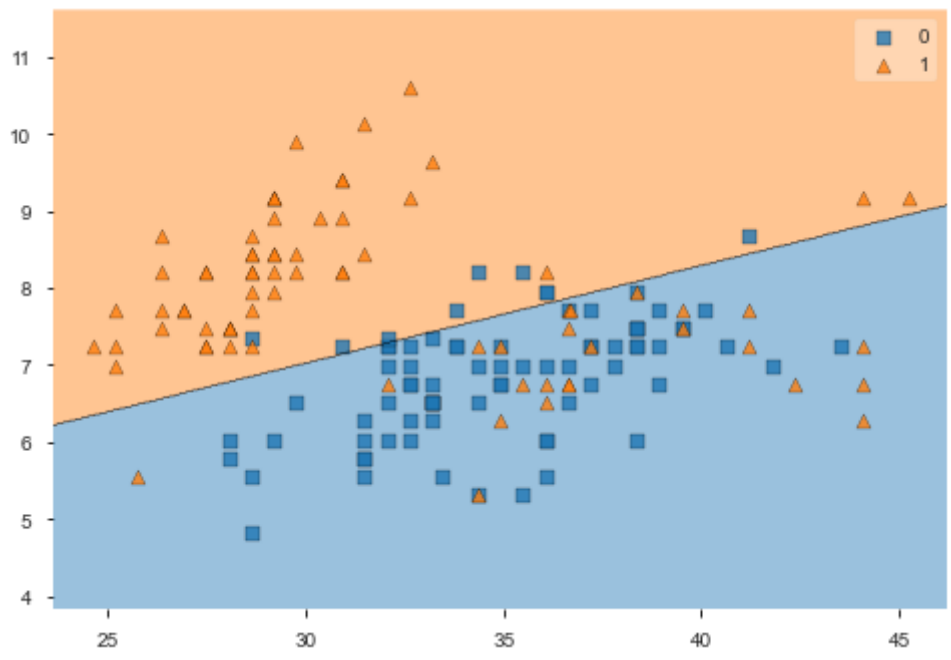
In [131]:
```
plot(np.array(X),np.array(y),model_linear,'Linear SVM')
plt.show()
```



In [132]:
```
# we can also use plot_decision_regions function
plot_decision_regions(np.array(X),np.array(y),clf=model_linear)
```

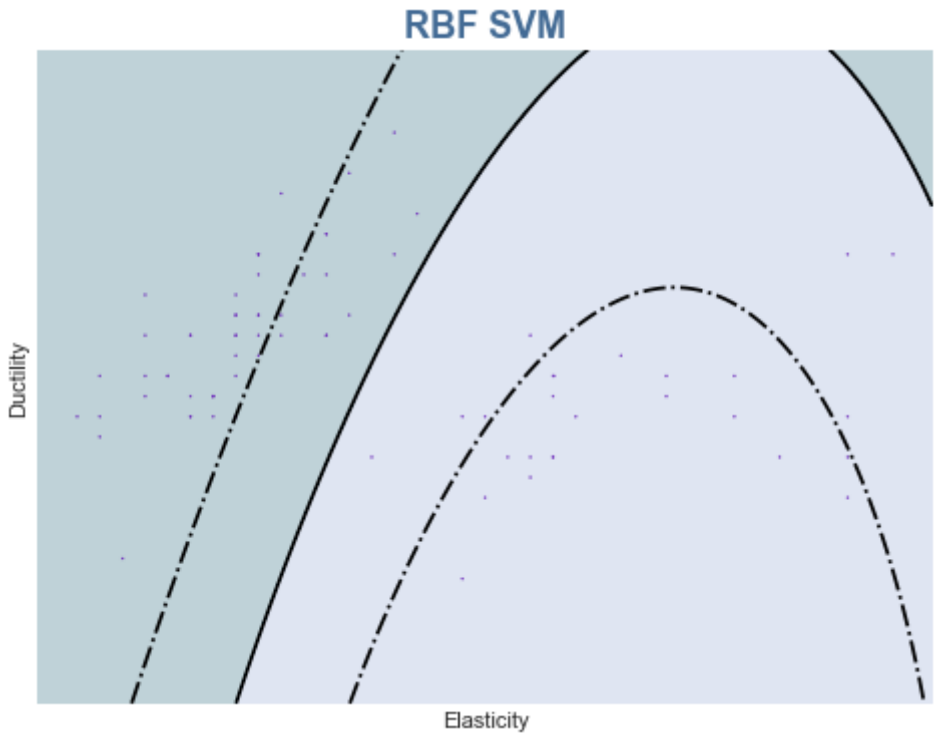
```
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:249: MatplotlibDeprecationWarning: Pa
ssing unsupported keyword arguments to axis() will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```

Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x210918b2748>



- RBF

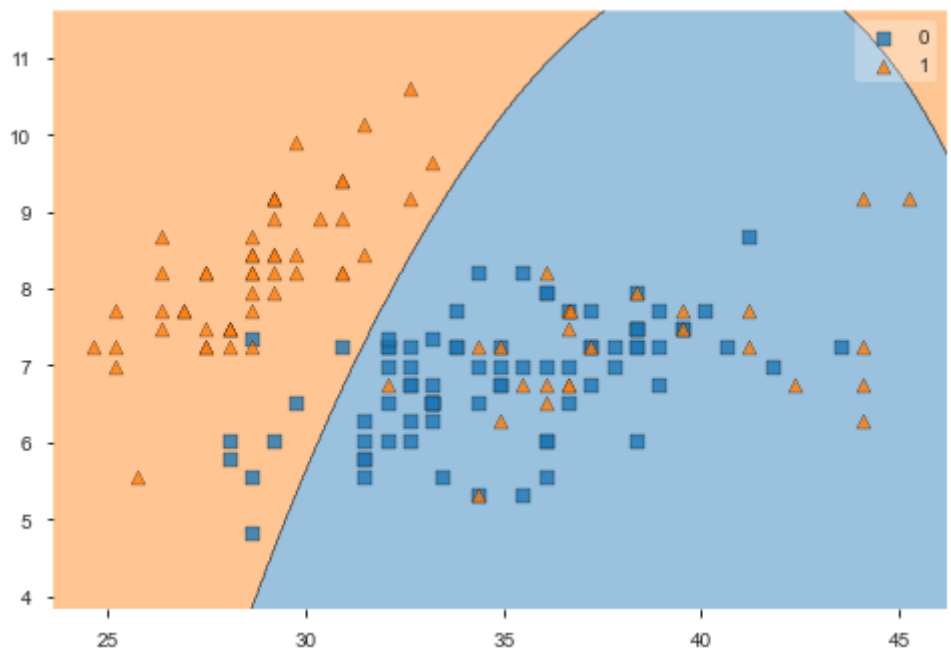
In [133]:
```
plot(np.array(X),np.array(y),model_rbf,'RBF SVM')
plt.show()
```



In [134]:
```
# we can also use plot_decision_regions function
plot_decision_regions(np.array(X),np.array(y),clf=model_rbf)
```

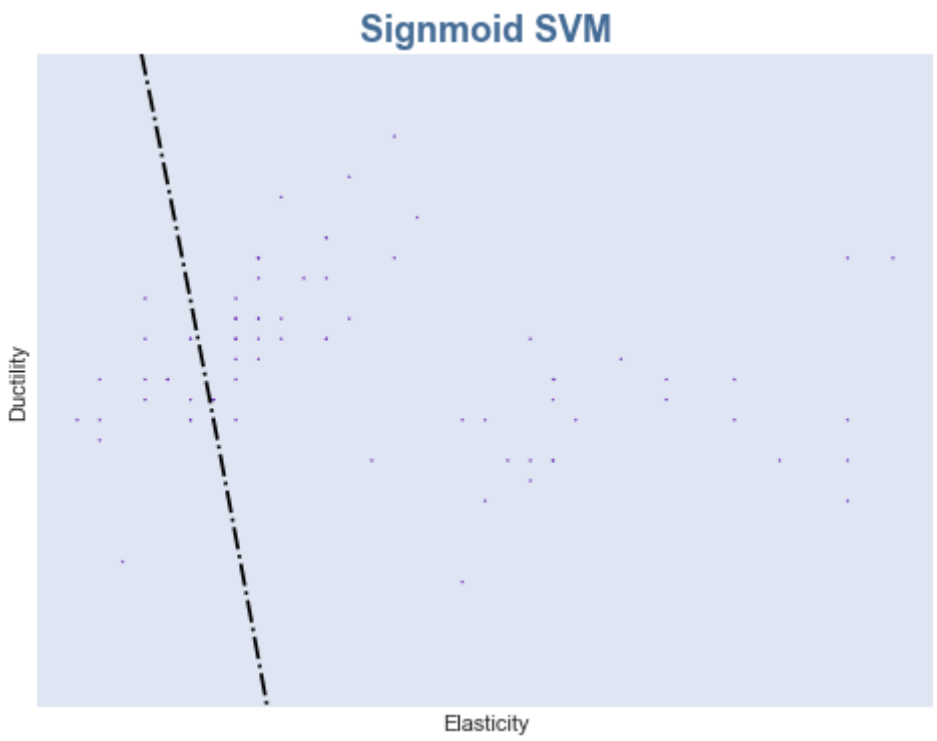
```
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:249: MatplotlibDeprecationWarning: Pa
ssing unsupported keyword arguments to axis() will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```

Out[134]: <matplotlib.axes._subplots.AxesSubplot at 0x21091a3b5f8>



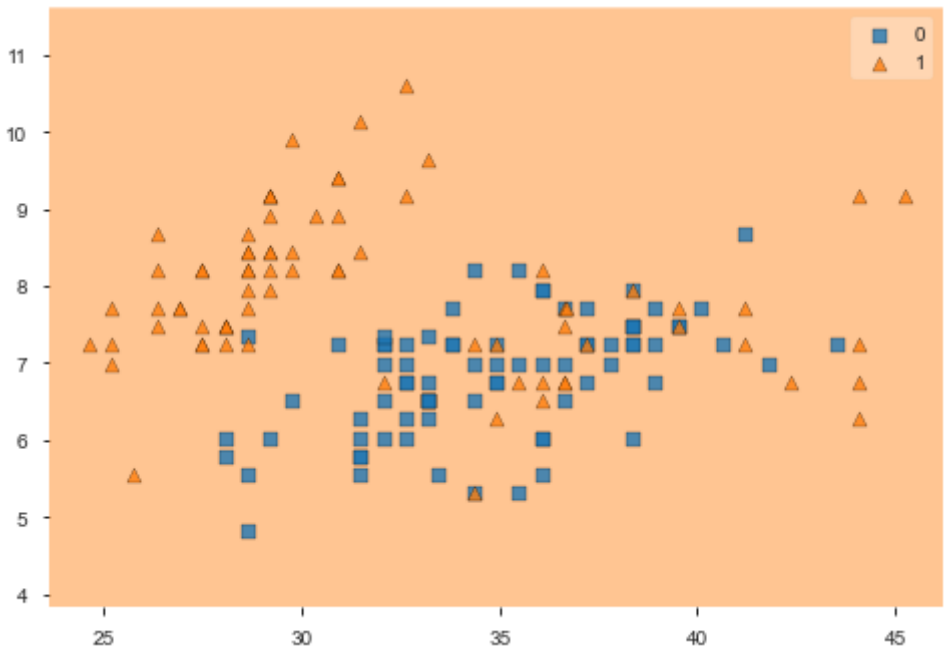
- Sigmoid

In [135]:
```
plot(np.array(X),np.array(y),model_sigmoid,'Signmoid SVM')
plt.show()
```



In [136]:
```
# we can also use plot_decision_regions function
plot_decision_regions(np.array(X),np.array(y),clf=model_sigmoid)
plt.show()
```

```
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:247: UserWarning: No contour levels w
ere found within the data range.
  antialiased=True)
C:\Users\sandh\Anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:249: MatplotlibDeprecationWarning: Pa
ssing unsupported keyword arguments to axis() will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



Module-4: Model Selection, Tuning & Boosting

The Process of choosing the best model that satisfies the requirements needed based on the problem statement is known as a model selection. Tuning the parameters of the model so that it produces the best results is know as model tuning. To do such, we will be required to evaluate the models based on different metrics like R-Square, RMSE, Mean Absolute Error for regressional models and confusion matrix, accuracy score for classification models. We will also be applying the boosting models over the data to get better results.

Getting Datasets:

In [ ]:
```
!wget https://www.dropbox.com/s/5x30kmmqk9gdfrf/concrete.csv -nv
!wget https://www.dropbox.com/s/ahfj6zfskjirxei/Depressed.csv -nv
!wget https://www.dropbox.com/s/cuh3asfjpj9fgzi/housevotes.csv -nv
!wget https://www.dropbox.com/s/v10zn2rodytj5mb/NBA.csv -nv
```

Scenario-1: Congressional Votes

Based on 16 key votes identified by CQA the data collected has votes from each U.S. House of Representatives Congressmen. There are 9 different types of votes listed by CQA: voted against, paired against, voted for, paired for, announced for, announced against, voted present, voted present to avoid conflict of interest, and no vote or otherwise.

Problem Statement:

Based on the votes given the aim is to classify either a congressman is republican or a democrat.

Dataset Description:

Attributes:

- **Handicapped_infants**:{y,n}
- **Water_project_cost**:{y,n}
- **Adoption_budget_resolution**:{y,n}
- **Physician_fee_freeze**:{y,n}
- **El_salvador_aid**:{y,n}
- **Religious_groups_in_schools**:{y,n}
- **Anti_satellite_test_ban**:{y,n}
- **Aid_to_nicaraguan_contras**:{y,n}
- **Mx_missile**:{y,n}
- **Immigration**:{y,n}
- **Synfuels_corporation_cutback**:{y,n}
- **Education_spending**:{y,n}
- **Superfund_right_to_sue**:{y,n}
- **Crime**:{y,n}
- **Duty_free_exports**:{y,n}
- **Export_south_africa**:{y,n}

Target Variable:

- **Class**: {republican,democrat}

Tasks to be Performed:

In order to attain the above goal below tasks must be performed:

- Read the dataset with no headers; Then put respective columns names. Use pandas_profiling to generate a report of dataset. - **Beginner**
- Preprocess the data. - **Beginner**
- Split the data into training and testing set and apply Bernoulli Naive Bayes, logistic regression, random forest, and decision tree models. - **Intermediate**
- Evaluate the models using confusion matrix. - **Intermediate**
- Using LOOCV fit a Logistic Regression model and Bernoulli model; Calculate the average score using accuracy as scoring parameter. - **Advanced**

Topics Covered:

- Naive Bayes
- Logistic Regression
- LOOCV

Question-1: Read the dataset with no headers; then put respective columns names. Use pandas_profiling to generate a report of dataset.

In [1]:
```
cols=['Handicapped_infants','Water_project_cost','Adoption_budget_resolution','Physician_fee_freeze',
'El_salvador_aid','Religious_groups_in_schools','Anti_satellite_test_ban','Aid_to_nicaraguan_contras',
'Mx_missile','Immigration','Synfuels_corporation_cutback','Education_spending','Superfund_right_to_sue',
'Crime','Duty_free_exports','Export_south_africa','Target']
```

In [2]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
data=pd.read_csv('housevotes.csv',header=None)
```

In [4]:
```
data.columns=cols
```

In [5]:
```
data.head()
```

Out[5]:

	Handicapped_infants	Water_project_cost	Adoption_budget_resolution	Physician_fee_freeze	El_salvador_aid	Religious_groups_in_schools	Anti_
0	n	y	n	y	y	y	
1	n	y	n	y	y	y	
2	NaN	y	y	NaN	y	y	
3	n	y	y	n	NaN	y	
4	y	y	y	n	y	y	

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Handicapped_infants             423 non-null    object
 1   Water_project_cost              387 non-null    object
 2   Adoption_budget_resolution      424 non-null    object
 3   Physician_fee_freeze            424 non-null    object
 4   El_salvador_aid                 420 non-null    object
 5   Religious_groups_in_schools     424 non-null    object
 6   Anti_satellite_test_ban         421 non-null    object
 7   Aid_to_nicaraguan_contras       420 non-null    object
 8   Mx_missile                      413 non-null    object
 9   Immigration                     428 non-null    object
 10  Synfuels_corporation_cutback    414 non-null    object
 11  Education_spending              404 non-null    object
 12  Superfund_right_to_sue          410 non-null    object
 13  Crime                           418 non-null    object
 14  Duty_free_exports               407 non-null    object
 15  Export_south_africa             331 non-null    object
 16  Target                          435 non-null    object
dtypes: object(17)
memory usage: 57.9+ KB
```

In [7]:
```
data.describe(include='all')
```

Out[7]:

	Handicapped_infants	Water_project_cost	Adoption_budget_resolution	Physician_fee_freeze	El_salvador_aid	Religious_groups_in_schools
count	423	387	424	424	420	424
unique	2	2	2	2	2	2
top	n	y	y	n	y	y
freq	236	195	253	247	212	272

```
In [8]: import pandas_profiling

In [9]: rpt=pandas_profiling.ProfileReport(data)

In [10]: rpt
```

Overview

Dataset statistics

Number of variables	17
Number of observations	435
Missing cells	392
Missing cells (%)	5.3%
Duplicate rows	93
Duplicate rows (%)	21.4%
Total size in memory	57.9 KiB
Average record size in memory	136.3 B

Variable types

BOOL	16
CAT	1

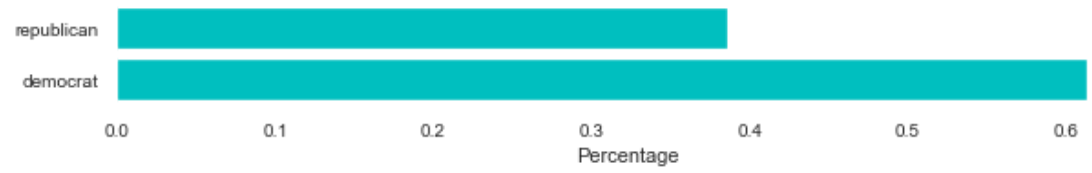
Reproduction

Analysis started	2020-07-14 14:18:31.087097
Analysis finished	2020-07-14 14:18:43.910956
Duration	12.82 seconds
Version	pandas-profiling v2.8.0 (https://github.com/pandas-profiling/pandas-profiling)
Command line	`pandas_profiling --config_file config.yaml [YOUR_FILE.csv]`

Out[10]:

Question-2: Pre-process the data.

```
In [11]: # Target distribution
def target_distribution(data,col):
    target_ratio=pd.DataFrame({'Counts':data[col].value_counts(),
                               'Percentage':data[col].value_counts()/len(data)})
    import matplotlib.pyplot as plt
    import seaborn as sns
    plt.figure(figsize = (10,1))
    plt.barh(target_ratio.index, target_ratio.Percentage,color='c')
    plt.xlabel('Percentage')
    plt.show()
target_distribution(data,'Target')
```



In [12]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[12]:

	Col_name	Missing value?	Count_	Percentage
15	Export_south_africa	True	104	0.239080
1	Water_project_cost	True	48	0.110345
11	Education_spending	True	31	0.071264
14	Duty_free_exports	True	28	0.064368
12	Superfund_right_to_sue	True	25	0.057471
8	Mx_missile	True	22	0.050575
10	Synfuels_corporation_cutback	True	21	0.048276
13	Crime	True	17	0.039080
4	El_salvador_aid	True	15	0.034483
7	Aid_to_nicaraguan_contras	True	15	0.034483
6	Anti_satellite_test_ban	True	14	0.032184
0	Handicapped_infants	True	12	0.027586
5	Religious_groups_in_schools	True	11	0.025287
3	Physician_fee_freeze	True	11	0.025287
2	Adoption_budget_resolution	True	11	0.025287
9	Immigration	True	7	0.016092
16	Target	False	0	0.000000

In [13]:
```
print('Total Missing values: %s'%sum(miss.Count_))
```

```
Total Missing values: 392
```

In [14]:
```
import numpy as np
```

In [15]:
```
for i in miss[miss.Count_!=0].Col_name:
    temp=data[i].mode()
    tes=data[i].isnull()
    ind=tes[tes==True].index
    quill=list(data[i])
    for j in ind:
        quill[j]=temp[0]
    data[i]=quill
```

In [16]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                        [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                        'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[16]:

	Col_name	Missing value?	Count_	Percentage
0	Handicapped_infants	False	0	0.0
9	Immigration	False	0	0.0
15	Export_south_africa	False	0	0.0
14	Duty_free_exports	False	0	0.0
13	Crime	False	0	0.0
12	Superfund_right_to_sue	False	0	0.0
11	Education_spending	False	0	0.0
10	Synfuels_corporation_cutback	False	0	0.0
8	Mx_missile	False	0	0.0
1	Water_project_cost	False	0	0.0
7	Aid_to_nicaraguan_contras	False	0	0.0
6	Anti_satellite_test_ban	False	0	0.0
5	Religious_groups_in_schools	False	0	0.0
4	El_salvador_aid	False	0	0.0
3	Physician_fee_freeze	False	0	0.0
2	Adoption_budget_resolution	False	0	0.0
16	Target	False	0	0.0

In [17]:
```
data.head()
```

Out[17]:

	Handicapped_infants	Water_project_cost	Adoption_budget_resolution	Physician_fee_freeze	El_salvador_aid	Religious_groups_in_schools	Anti_
0	n	y	n	y	y	y	
1	n	y	n	y	y	y	
2	n	y	y	n	y	y	
3	n	y	y	n	y	y	
4	y	y	y	n	y	y	

In [18]:
```
from sklearn.preprocessing import LabelEncoder
```

In [19]:
```
def lb_encode(x):
    lb=LabelEncoder()
    return lb.fit_transform(x)
```

In [20]:
```
label_data=data.apply(lb_encode)
```

In [21]:
```
label_data.head()
```

Out[21]:

	Handicapped_infants	Water_project_cost	Adoption_budget_resolution	Physician_fee_freeze	El_salvador_aid	Religious_groups_in_schools	Anti_
0	0	1	0	1	1	1	
1	0	1	0	1	1	1	
2	0	1	1	0	1	1	
3	0	1	1	0	1	1	
4	1	1	1	0	1	1	

Question-3: Split the data into training and testing set and apply Bernoulli Naive Bayes, logistic regression, random forest, and decision tree models.

In [22]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
X=label_data.drop('Target',axis=1)
y=label_data.Target
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75,test_size=0.25, random_state=101)
```

In [24]:
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
```

In [25]:
```
bnb=BernoulliNB(alpha=5)
bnb.fit(X_train,y_train)
```

Out[25]:
```
BernoulliNB(alpha=5, binarize=0.0, class_prior=None, fit_prior=True)
```

In [26]:
```
lgm=LogisticRegression(C=2)
lgm.fit(X_train,y_train)
```

Out[26]:
```
LogisticRegression(C=2, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [28]:
```
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

Out[28]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [29]:
```
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
```

Out[29]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, 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=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [30]:
```
rf_pred=rf.predict(X_test)
dt_pred=dt.predict(X_test)
```

In [31]:
```
b_pred=bnb.predict(X_test)
```

In [32]:
```
l_pred=lgm.predict(X_test)
```

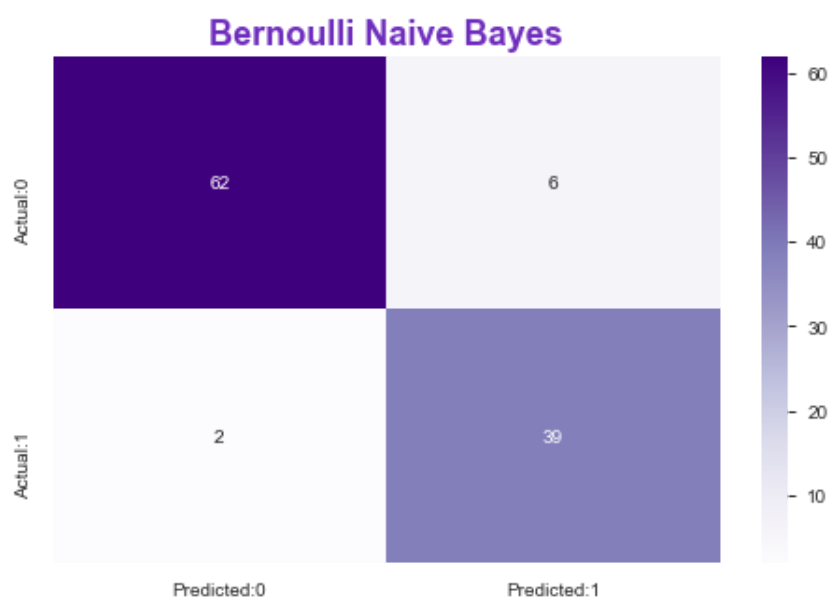
Question-4: Evaluate the models using confusion matrix.

In [33]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

In [34]:
```
print('The accuracy of the bernoulli  model is: ',round(accuracy_score(y_test,b_pred)*100,2))
print('The accuracy of the logistic  model is: ',round(accuracy_score(y_test,l_pred)*100,2))
```

```
The accuracy of the bernoulli  model is:  92.66
The accuracy of the logistic  model is:  96.33
```

In [35]:
```
dcm=confusion_matrix(y_test,b_pred)
dconf_matrix=pd.DataFrame(data=dcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(dconf_matrix, annot=True,fmt='d',cmap='Purples')
fm={'size':18,'color':'#6f2dbd','weight':'bold'}
plt.title('Bernoulli Naive Bayes',**fm)
plt.show()
```



In [36]:
```
TN=dcm[0,0]
TP=dcm[1,1]
FN=dcm[1,0]
FP=dcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 62
True Positive 39
False Negative 2
False Positive 6
Sensitivity 0.9512195121951219
Specificity 0.9117647058823529
```

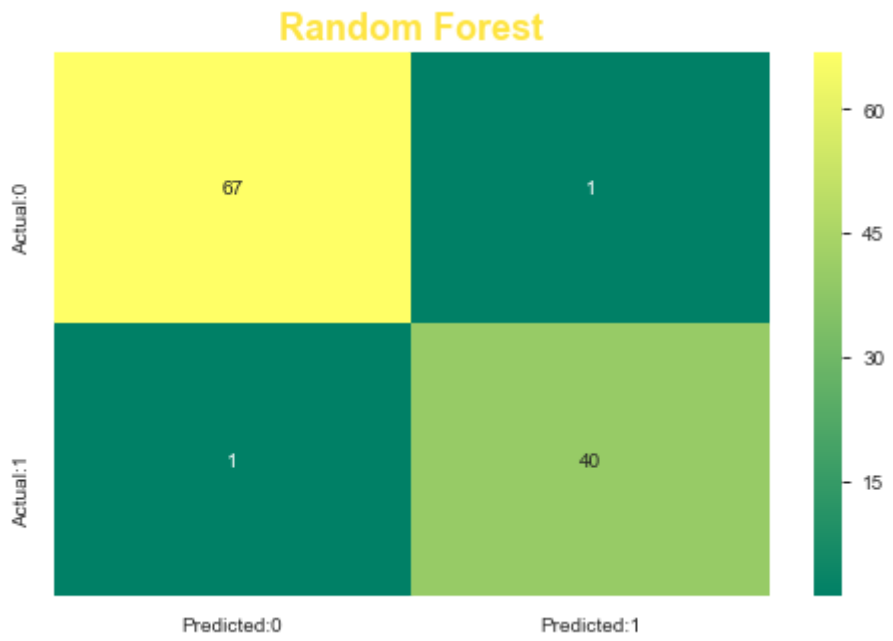
Bernoulli model is slightly more sensitive.

In [37]:
```
dcm=confusion_matrix(y_test,l_pred)
dconf_matrix=pd.DataFrame(data=dcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(dconf_matrix, annot=True,fmt='d',cmap='Greens')
fm={'size':18,'color':'#38a700','weight':'bold'}
plt.title('Logistic Regression',**fm)
plt.show()
```

In [38]:
```
TN=dcm[0,0]
TP=dcm[1,1]
FN=dcm[1,0]
FP=dcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 66
True Positive 39
False Negative 2
False Positive 2
Sensitivity 0.9512195121951219
Specificity 0.9705882352941176
```

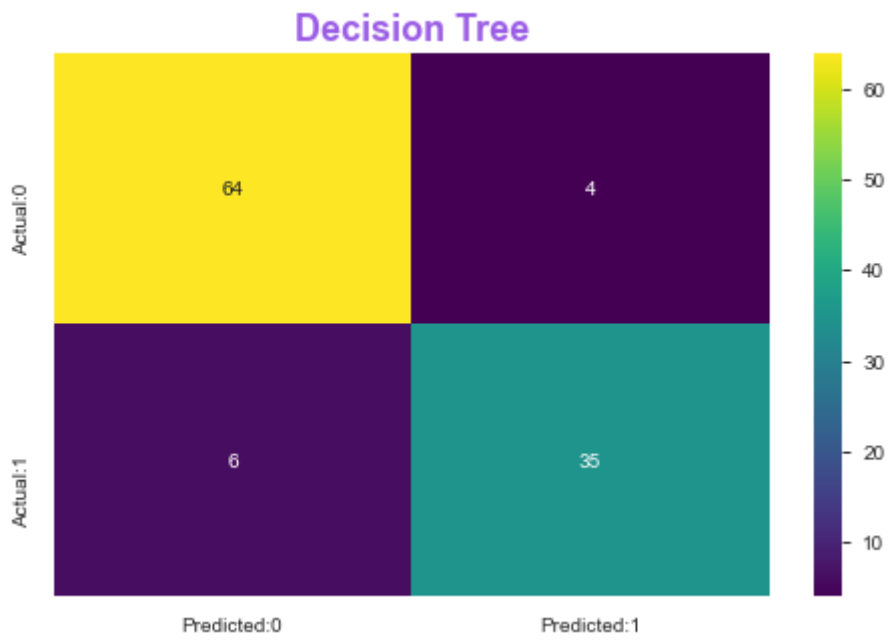
In [39]:
```
dcm=confusion_matrix(y_test,rf_pred)
dconf_matrix=pd.DataFrame(data=dcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(dconf_matrix, annot=True,fmt='d',cmap='summer')
fm={'size':18,'color':'#FEE440','weight':'bold'}
plt.title('Random Forest',**fm)
plt.show()
```



In [40]:
```
TN=dcm[0,0]
TP=dcm[1,1]
FN=dcm[1,0]
FP=dcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 67
True Positive 40
False Negative 1
False Positive 1
Sensitivity 0.975609756097561
Specificity 0.9852941176470589
```

In [41]:
```
dcm=confusion_matrix(y_test,dt_pred)
dconf_matrix=pd.DataFrame(data=dcm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(dconf_matrix, annot=True,fmt='d',cmap='viridis')
fm={'size':18,'color':'#9B5DE5','weight':'bold'}
plt.title('Decision Tree',**fm)
plt.show()
```



In [42]:
```
TN=dcm[0,0]
TP=dcm[1,1]
FN=dcm[1,0]
FP=dcm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 64
True Positive 35
False Negative 6
False Positive 4
Sensitivity 0.8536585365853658
Specificity 0.9411764705882353
```

In [43]:
```
from sklearn.metrics import precision_score, classification_report,f1_score,recall_score
```

In [44]:
```
print('\t\tClassification Report for Logistic Model')
print(classification_report(y_test,l_pred))
```

```
                Classification Report for Logistic Model
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        68
           1       0.95      0.95      0.95        41

    accuracy                           0.96       109
   macro avg       0.96      0.96      0.96       109
weighted avg       0.96      0.96      0.96       109
```

In [45]:
```
print('\t\tClassification Report for Bernoulli Naive Bayes Model')
print(classification_report(y_test,b_pred))
```

```
              Classification Report for Bernoulli Naive Bayes Model
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        68
           1       0.87      0.95      0.91        41

    accuracy                           0.93       109
   macro avg       0.92      0.93      0.92       109
weighted avg       0.93      0.93      0.93       109
```

In [46]:
```
print('\t\tClassification Report for Random Forest Model')
print(classification_report(y_test,rf_pred))
```

```
                Classification Report for Random Forest Model
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        68
           1       0.98      0.98      0.98        41

    accuracy                           0.98       109
   macro avg       0.98      0.98      0.98       109
weighted avg       0.98      0.98      0.98       109
```

In [47]:
```
print('\t\tClassification Report for Decision Tree Model')
print(classification_report(y_test,dt_pred))
```

```
                Classification Report for Decision Tree Model
              precision    recall  f1-score   support

           0       0.91      0.94      0.93        68
           1       0.90      0.85      0.88        41

    accuracy                           0.91       109
   macro avg       0.91      0.90      0.90       109
weighted avg       0.91      0.91      0.91       109
```

In [48]:
```
# Let's see precision and recall values
#Initializae the evaluation dictionary
def initialize_evaluator():
    return {'Model':[],'Accuracy':[],'Precision':[],'Recall':[],'F1_score':[]}

#Insert data in evaluation dictionary
def insert_data(test,pred,model):
    eval_data=initialize_evaluator()
    eval_data['Model'].append(model)
    eval_data['Accuracy'].append(accuracy_score(test,pred))
    eval_data['Precision'].append(precision_score(test,pred))
    eval_data['Recall'].append(recall_score(test,pred))
    eval_data['F1_score'].append(f1_score(test,pred))
    return eval_data

# Append data of one dictionary to another
def append_data(data1, data2):
    for i in data1.keys():
        data2[i].extend(data1[i])
    return data2
```

In [49]:
```
eval_l=insert_data(y_test,l_pred,'Logistic Regression')
eval_b=insert_data(y_test,b_pred,'Bernoulli Naive Bayes')
eval_rf=insert_data(y_test,rf_pred,'Random Forest')
eval_dt=insert_data(y_test,dt_pred,'Decision Tree')
eval_all=append_data(eval_l,eval_b)
eval_all=append_data(eval_rf,eval_all)
eval_all=append_data(eval_dt,eval_all)
```

In [50]:
```
# Let's Plot the model
from matplotlib.gridspec import GridSpec
def plot_models(data):
    sns.set_palette(sns.color_palette("rocket"))
    super_title={'size':18,'color':'#c5283d','weight':'bold'}
    sub_title={'size':12,'color':'#e06777','weight':'bold'}
    colors=np.array([[156, 137, 184],[239, 195, 230],[184, 190, 221],[231, 115, 171]])
    colors=colors/255 #Matplotlib RGB color range is from 0-1
    data=pd.DataFrame(data)
    fig = plt.figure(figsize=(10,7),constrained_layout=True)
    gs = GridSpec(2, 2, figure=fig)
    ax1 = fig.add_subplot(gs[0, 0])
    ax1.barh(data.Model,data.Accuracy,color=colors)
    ax1.tick_params(labelbottom=False, labelleft=True)
    ax1.set_xlim(0,1)
    ax1.set_title('Accuracy',**sub_title)
    ax2 = fig.add_subplot(gs[0, 1])
    ax2.barh(data.Model,data.Precision,color=colors)
    ax2.tick_params(labelbottom=False, labelleft=False)
    ax2.set_xlim(0,1)
    ax2.set_title('Precision',**sub_title)
    ax3 = fig.add_subplot(gs[1, 0])
    ax3.barh(data.Model,data.Recall,color=colors)
    ax3.tick_params(labelbottom=True, labelleft=True)
    ax3.set_xlim(0,1)
    ax3.set_title('Recall',**sub_title)
    ax4 = fig.add_subplot(gs[1, 1])
    ax4.barh(data.Model,data.F1_score,color=colors)
    ax4.tick_params(labelbottom=False, labelleft=False)
    ax4.set_xlim(0,1)
    ax4.set_title('F1 Score',**sub_title)
    fig.suptitle("Evaluation",**super_title)
    ax4.tick_params(labelbottom=True, labelleft=False)
    plt.show()
```

In [51]:
```
plot_models(eval_all)
```

From the above, we can see that our random forest model is performing best in every aspect.

Question-5: Using LOOCV fit a Logistic Regression model and Bernoulli model; Calculate the average score using accuracy as scoring parameter.

In [52]:
```
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import accuracy_score,mean_absolute_error
```

In [53]:
```
#the leave one out
loo = LeaveOneOut()
loo.get_n_splits(X)
scores=[]
for train_index, test_index in loo.split(X):
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    model=LogisticRegression(C=2)
    model.fit(X_train,y_train)
    scores.append(mean_absolute_error(y_test,model.predict(X_test)))
```

In [54]:
```
print('Average Score: %.2f'%np.mean(scores))
```

```
Average Score: 0.04
```

In [55]:
```
loo = LeaveOneOut()
loo.get_n_splits(X)
scores=[]
for train_index, test_index in loo.split(X):
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    model=BernoulliNB(alpha=5)
    model.fit(X_train,y_train)
    scores.append(mean_absolute_error(y_test,model.predict(X_test)))
```

In [56]:
```
print('Average Score: %.2f'%np.mean(scores))
```

```
Average Score: 0.10
```

- Let's try random forest

In [57]:
```
loo = LeaveOneOut()
loo.get_n_splits(X)
scores=[]
for train_index, test_index in loo.split(X):
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    model=RandomForestClassifier()
    model.fit(X_train,y_train)
    scores.append(mean_absolute_error(y_test,model.predict(X_test)))

print('Average Score: %.2f'%np.mean(scores))
```

```
Average Score: 0.04
```

The above accuracy is due to the fact that the dataset is very small and imbalanced.

Scenario-2: Crayola Cement Foundation

Established in 1996, Crayola Cement Foundation is one of the top competitors in the cement industries. They are proud of an exceptional record of efficiency, safety, and productivity while always seeking the newest techniques, systems, and equipment. They make precast and prestressed concrete products such as columns, raker beams, wall panels, spandrels, seat risers, double and inverted tees, hollow-core, and stairs.

Problem Statement:

Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. CCF wants to find out the compressive strength of the concrete based on these attributes.

Dataset Description:

Attributes:

- **Cement**: real [102.0, 540.0]
- **BlastFurnaceSlag**: real [0.0, 359.4]
- **FlyAsh**: real [0.0, 200.100006]
- **Water**: real [121.8, 247.0]
- **Superplasticizer**: real [0.0, 32.200001]
- **CoarseAggregate**: real [801.0, 1145.0]
- **FineAggregate**: real [594.0, 992.6]
- **Age**: integer [1, 365]

Target Variable:

- **ConcreteCompressiveStrength**: real [2.33, 82.6]

Tasks to be Performed:

In order to attain the above goal below tasks must be performed:

- Read the dataset with no headers; Then put respective columns names and find the correlation between the features - **Beginner**
- Fit a Linear Regression model by applying 5-fold cross validation using scikit and calculate the scores and average accuracy - **Intermediate**
- Fit a Linear Regression model by applying 5-fold cross validation explicitly using KFold scikit and calculate the scores using mean absolute error as scoring parameter - **Advanced**
- Using LOOCV fit a Linear Regression model and calculate the average score using mean absolute error as scoring parameter **Advanced**
- Apply Lasso Regression over the model and find R2_score and MAE - **Beginner**
- Apply Ridge Regression over the model and find R2_score and MAE - **Beginner**
- Apply RandomForestRegressor and Linear Regression over the data and evaluate. **Advanced**

Topics Covered:

- Linear Regression
- Lasso Regression
- Ridge Regression
- K-Fold
- RMSE
- Mean Absolute Error
- R-Square
- LOOCV
- Random Forest Regressor

Question-1: Read the dataset with no headers; Then put respective columns names and find the correlation between the features.

In [58]:
```
cols=['Cement','BlastFurnaceSlag','FlyAsh','Water','Superplasticizer',
'CoarseAggregate','FineAggregate','Age','ConcreteCompressiveStrength']
```

In [59]:
```
import pandas as pd
```

In [60]:
```
data=pd.read_csv('concrete.csv',header=None)
data.columns=cols
data.head()
```

Out[60]:

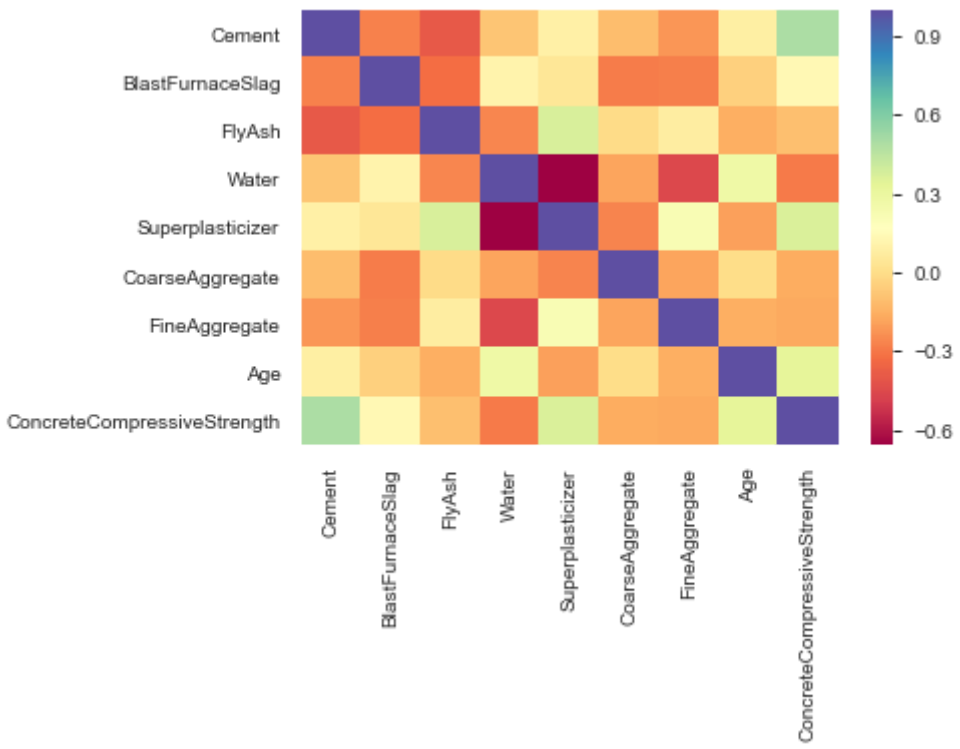
	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	ConcreteCompressiveStrength
0	252.000000	0.0	0.000000	185.000000	0.0	1111.000000	784.000000	7.0	13.710000
1	295.799988	0.0	0.000000	185.699997	0.0	1091.400024	769.299988	7.0	14.840000
2	252.300003	0.0	98.800003	146.300003	14.2	987.799988	889.000000	3.0	21.780001
3	172.399994	13.6	172.399994	156.800003	4.1	1006.299988	856.400024	28.0	33.689999
4	162.000000	214.0	164.000000	202.000000	10.0	820.000000	680.000000	28.0	30.650000

In [61]:
```
import seaborn as sns
data.corr()
```

Out[61]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	Co
Cement	1.000000	-0.275216	-0.397467	-0.081587	0.092386	-0.109349	-0.222718	0.081946	
BlastFurnaceSlag	-0.275216	1.000000	-0.323580	0.107252	0.043270	-0.283999	-0.281603	-0.044246	
FlyAsh	-0.397467	-0.323580	1.000000	-0.256984	0.377503	-0.009961	0.079109	-0.154371	
Water	-0.081587	0.107252	-0.256984	1.000000	-0.657533	-0.182294	-0.450661	0.277618	
Superplasticizer	0.092386	0.043270	0.377503	-0.657533	1.000000	-0.265999	0.222691	-0.192700	
CoarseAggregate	-0.109349	-0.283999	-0.009961	-0.182294	-0.265999	1.000000	-0.178481	-0.003016	
FineAggregate	-0.222718	-0.281603	0.079109	-0.450661	0.222691	-0.178481	1.000000	-0.156095	
Age	0.081946	-0.044246	-0.154371	0.277618	-0.192700	-0.003016	-0.156095	1.000000	
ConcreteCompressiveStrength	0.497832	0.134829	-0.105755	-0.289633	0.366079	-0.164935	-0.167241	0.328873	

In [62]:
```
sns.heatmap(data.corr(),cmap='Spectral')
plt.show()
```



In [63]:
```
# Let's use pandas profiling over the dataset to generate a report
import pandas_profiling
rpt=pandas_profiling.ProfileReport(data)
rpt
```

Overview

Dataset statistics

Number of variables	9
Number of observations	1030
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	25
Duplicate rows (%)	2.4%
Total size in memory	72.5 KiB
Average record size in memory	72.1 B

Variable types

NUM	9

Reproduction

Analysis started	2020-07-14 14:20:18.624091
Analysis finished	2020-07-14 14:20:32.193256
Duration	13.57 seconds
Version	pandas-profiling v2.8.0 (https://github.com/pandas-profiling/pandas-profiling)
Command line	`pandas_profiling --config_file config.yaml [YOUR_FILE.csv]`

Out[63]:

In [64]:
```
# Preprocessing the data
data.head()
```

Out[64]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	ConcreteCompressiveStrength
0	252.000000	0.0	0.000000	185.000000	0.0	1111.000000	784.000000	7.0	13.710000
1	295.799988	0.0	0.000000	185.699997	0.0	1091.400024	769.299988	7.0	14.840000
2	252.300003	0.0	98.800003	146.300003	14.2	987.799988	889.000000	3.0	21.780001
3	172.399994	13.6	172.399994	156.800003	4.1	1006.299988	856.400024	28.0	33.689999
4	162.000000	214.0	164.000000	202.000000	10.0	820.000000	680.000000	28.0	30.650000

In [65]:
```
data.isna().any()
```

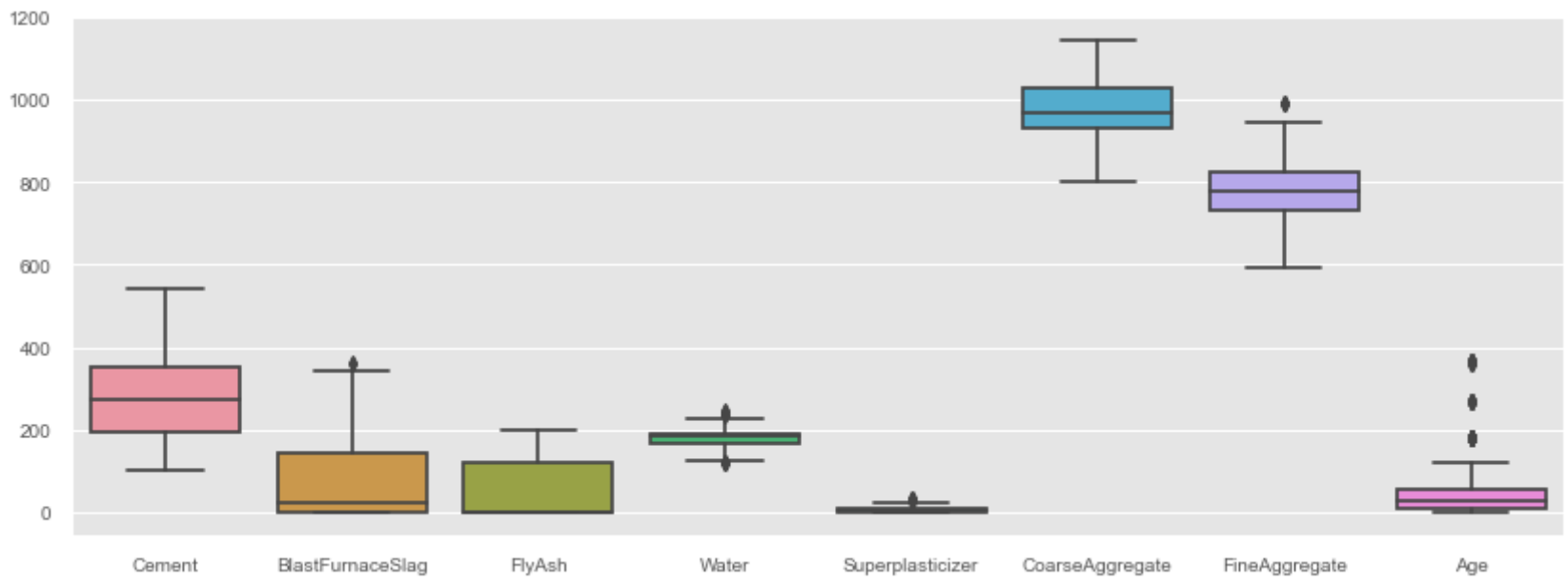
Out[65]:
```
Cement                         False
BlastFurnaceSlag               False
FlyAsh                         False
Water                          False
Superplasticizer               False
CoarseAggregate                False
FineAggregate                  False
Age                            False
ConcreteCompressiveStrength    False
dtype: bool
```

In [66]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [67]:
```
X=data.drop('ConcreteCompressiveStrength',axis=1)
y=data.ConcreteCompressiveStrength
```

In [68]:
```
feat=X.columns
```

In [69]:
```
plt.style.use('ggplot')
plt.figure(figsize=(14,5))
sns.boxplot(data=X)
plt.xticks(ticks=np.arange(len(feat)),labels=feat)
plt.show()
```



Data is highly imbalanced.

In [70]:
```
from sklearn.preprocessing import MinMaxScaler
```

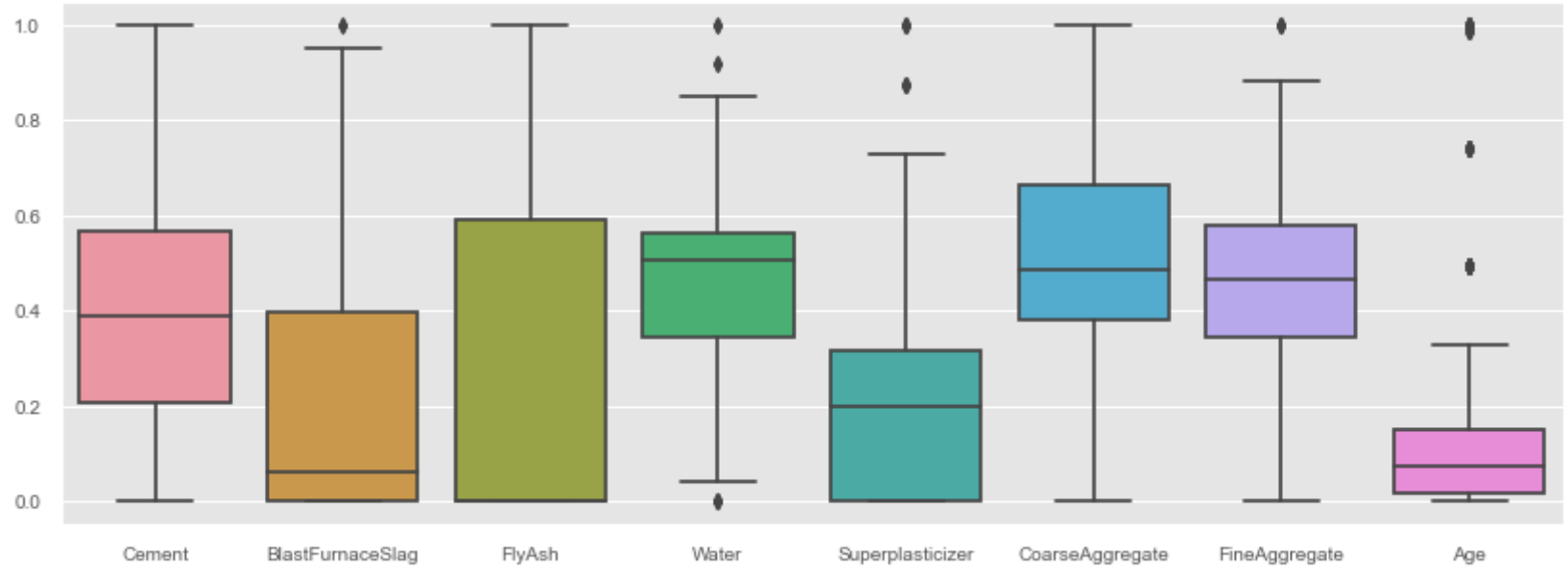
In [71]:
```
# Scaling data after outlier removal
scl=MinMaxScaler()
scl_data=scl.fit_transform(data)
```

In [72]:
```
scl_data.shape
```

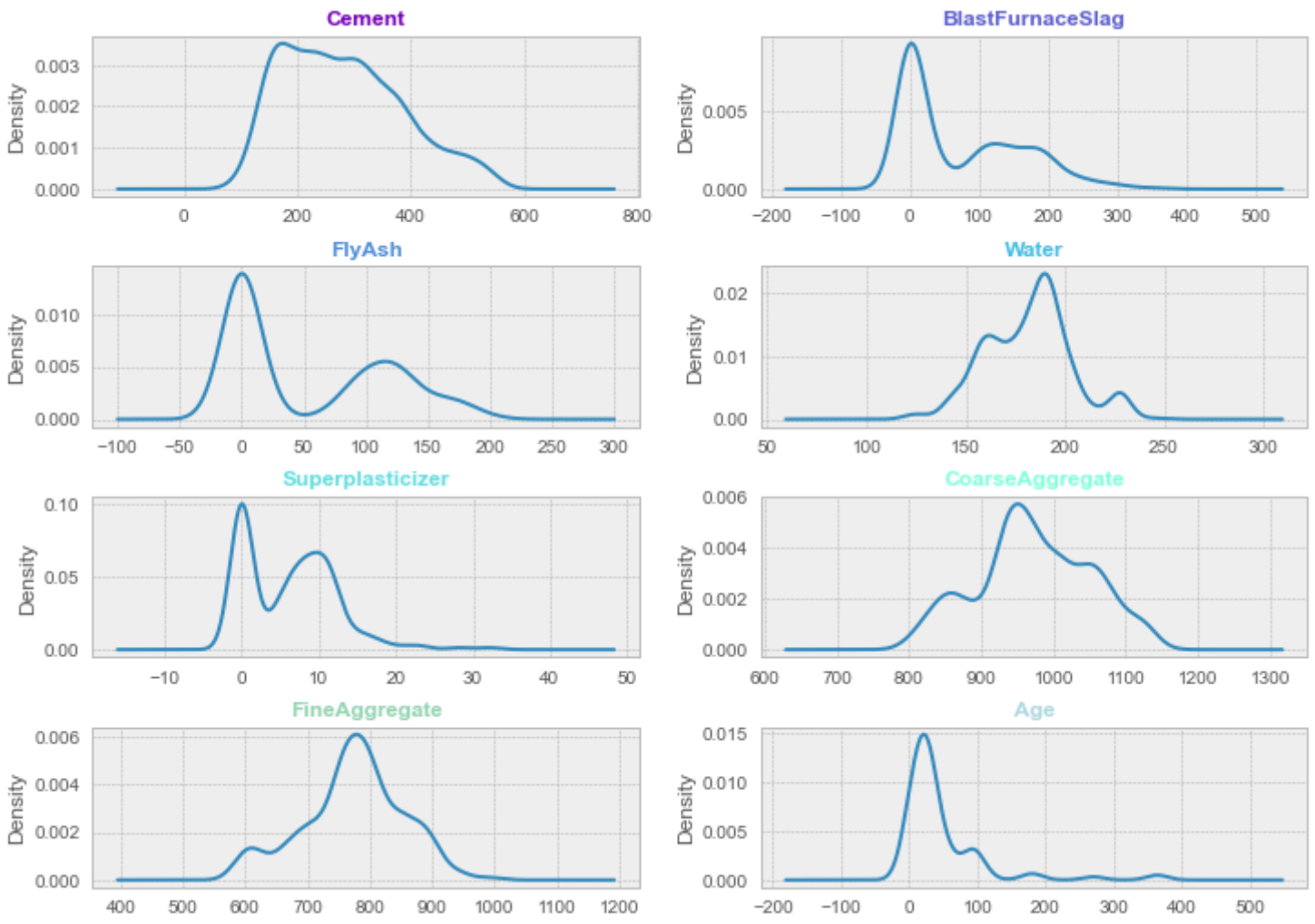
Out[72]:
```
(1030, 9)
```

In [73]:
```
X=scl_data[:,:8]
y=scl_data[:,8]
```

In [74]:
```
plt.style.use('ggplot')
plt.figure(figsize=(14,5))
sns.boxplot(data=X)
plt.xticks(ticks=np.arange(len(feat)),labels=feat)
plt.show()
```



In [75]:
```
# KDE Plots
def format_text(size,color,weight='heavy'):
    return {'size':size,'color':'#'+color,'weight':weight}

from matplotlib.gridspec import GridSpec
plt.style.use('bmh')
fig = plt.figure(figsize=(10,7),constrained_layout=True)
gs = GridSpec(4, 2, figure=fig)
ax1 = fig.add_subplot(gs[0, 0])
data.Cement.plot.kde()
ax1.set_title('Cement',**format_text(12,'7400b8'))
ax2 = fig.add_subplot(gs[0,1])
data.BlastFurnaceSlag.plot.kde()
ax2.set_title('BlastFurnaceSlag',**format_text(12,'5e60ce'))
ax3 = fig.add_subplot(gs[1,0])
data.FlyAsh.plot.kde()
ax3.set_title('FlyAsh',**format_text(12,'5390d9'))
ax4 = fig.add_subplot(gs[1,1])
data.Water.plot.kde()
ax4.set_title('Water',**format_text(12,'48bfe3'))
ax5 = fig.add_subplot(gs[2,0])
data.Superplasticizer.plot.kde()
ax5.set_title('Superplasticizer',**format_text(12,'64dfdf'))
ax6 = fig.add_subplot(gs[2,1])
data.CoarseAggregate.plot.kde()
ax6.set_title('CoarseAggregate',**format_text(12,'80ffdb'))
ax7 = fig.add_subplot(gs[3,0])
data['FineAggregate'].plot.kde()
ax7.set_title('FineAggregate',**format_text(12,'95d5b2'))
ax7 = fig.add_subplot(gs[3,1])
data['Age'].plot.kde()
ax7.set_title('Age',**format_text(12,'AED9E0'))
plt.show()
```



In [76]:
```
# findUnique returns a dataframe with every column name, number of unique values, minimum values, maximum values
# and total unique values present altogether in the dataset
def findUnique(data):
    unq=0
    unq_data={'Col_Name':[],'Unique_Counts':[],'Max_value':[],'Min_value':[]}
    for i in data.columns:
        unq_data['Col_Name'].append(i)
        unq_data['Unique_Counts'].append(len(data[i].unique()))
        unq_data['Max_value'].append(data[i].max())
        unq_data['Min_value'].append(data[i].min())
        unq+=len(data[i].unique())
    return pd.DataFrame(unq_data),unq
u,ud=findUnique(data)
u.sort_values(by='Unique_Counts')
```

Out[76]:

	Col_Name	Unique_Counts	Max_value	Min_value
7	Age	14	365.000000	1.000000
4	Superplasticizer	111	32.200001	0.000000
2	FlyAsh	156	200.100006	0.000000
1	BlastFurnaceSlag	185	359.399994	0.000000
3	Water	195	247.000000	121.800003
0	Cement	278	540.000000	102.000000
5	CoarseAggregate	284	1145.000000	801.000000
6	FineAggregate	302	992.599976	594.000000
8	ConcreteCompressiveStrength	845	82.599998	2.330000

Question-2: Fit a Linear Regression model by applying 5-fold cross validation using scikit and calculate the scores and average accuracy.

In [77]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [78]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,test_size=0.3, random_state=101)
```

In [79]:
```
from sklearn.model_selection import cross_val_score
lin_model = LinearRegression()

# fitting a model and computing the score 5 consecutive times (with different splits each time)
scores = cross_val_score(lin_model,X ,y , cv=5)

print("Scores: %s"%scores)
print("Accuracy:  %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Scores: [       nan 0.64538043 0.56344555 0.6113636  0.51208946]
Accuracy:  nan (+/- nan)
```

Question-3: Fit a Linear Regression model by applying 5-fold cross validation explicitly using KFold scikit and calculate the scores using mean absolute error as scoring parameter.

In [80]:
```
from sklearn.model_selection import KFold
k_fold  = KFold(n_splits=5)
scores = cross_val_score(lin_model,X, y, cv=k_fold, scoring='neg_mean_absolute_error')

print("Scores:",scores)
print("MAE: %0.2f (+/- %0.2f)" % (-scores.mean(), scores.std() * 2))
```

```
Scores: [-0.10042234 -0.10178749 -0.10261833 -0.09825851 -0.11293229]
MAE: 0.10 (+/- 0.01)
```

Question-4: Using LOOCV fit a Linear Regression model and calculate the average score using mean absolute error as scoring parameter.

In [81]:
```
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import mean_absolute_error

# creating the leave one out function
loo = LeaveOneOut()
loo.get_n_splits(X)
scores=[]
# printing the training and validation data
for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index,:], X[test_index,:]
    y_train, y_test = y[train_index], y[test_index]
    model=LinearRegression()
    model.fit(X_train,y_train)
    scores.append(mean_absolute_error(y_test,model.predict(X_test)))

print('Average Score: %.2f'%np.mean(scores))
```

```
Average Score: 0.10
```

Question-5: Apply Lasso Regression over the model and find r2_score and MAE.

In [82]:
```
from sklearn.linear_model import Lasso
```

In [83]:
```
X.shape,y.shape
```

Out[83]:
```
((1030, 8), (1030,))
```

In [84]:
```
lasso=Lasso(alpha=1e-5)
lasso.fit(X,y)
```

Out[84]:
```
Lasso(alpha=1e-05, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [85]:
```
print('coeficients: %(coef)s, intercept: %(intercept).2f'%{'coef':lasso.coef_,'intercept':lasso.intercept_})
```

```
coeficients: [ 0.64942553  0.4607512   0.21645739 -0.23830273  0.11668036  0.07397172
  0.09538162  0.51728002], intercept: -0.05
```

In [86]:
```
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
```

In [87]:
```
print('Mean Absolute Error: %.2f'%mean_absolute_error(y,lasso.predict(X)))
print('Mean Squared Error: %.2f'%mean_squared_error(y,lasso.predict(X)))
print('R2-Score: %.2f'%r2_score(y, lasso.predict(X)))
```

```
Mean Absolute Error: 0.10
Mean Squared Error: 0.02
R2-Score: 0.62
```

Question-6: Apply Ridge Regression over the model and find r2_score and MAE.

In [88]:
```
from sklearn.linear_model import Ridge
```

In [89]:
```
X.shape,y.shape
```

Out[89]:
```
((1030, 8), (1030,))
```

In [90]:
```
ridge=Lasso(alpha=1e-5)
ridge.fit(X,y)
```

Out[90]:
```
Lasso(alpha=1e-05, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [91]:
```
print('coeficients: %(coef)s, intercept: %(intercept).2f'%{'coef':ridge.coef_,'intercept':ridge.intercept_})
```

```
coeficients: [ 0.64942553  0.4607512   0.21645739 -0.23830273  0.11668036  0.07397172
  0.09538162  0.51728002], intercept: -0.05
```

In [92]:
```
from sklearn.metrics import mean_absolute_error, r2_score,mean_squared_error
```

In [93]:
```
print('Mean Absolute Error: %.2f'%mean_absolute_error(y,ridge.predict(X)))
print('R2-Score: %.2f'%r2_score(y, ridge.predict(X)))
print('Mean Squared Error: %.2f'%mean_squared_error(y,ridge.predict(X)))
```

```
Mean Absolute Error: 0.10
R2-Score: 0.62
Mean Squared Error: 0.02
```

Lasso Regression has R2 Score of 0.62 which is better than Ridge regression.

Question-7: Apply RandomForestRegressor and Linear Regression over the data and evaluate.

In [94]:
```
def evaluate_model(X,y,model_name,model):
    """
    X: Features
    y: Target Variable
    model_name: String, model name
    model: model object created
    """
    evl={'Model':[],'R2_Score':[],'MSE':[],'MAE':[]}
    evl['Model'].append(model_name)
    evl['R2_Score'].append(r2_score(y, model.predict(X)))
    evl['MSE'].append(mean_absolute_error(y, model.predict(X)))
    evl['MAE'].append(mean_squared_error(y, model.predict(X)))
    return evl
```

In [95]:
```
revl=evaluate_model(X,y,'Ridge Model',ridge)
levl=evaluate_model(X,y,'Lasso Model',lasso)
```

In [96]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
```

In [97]:
```
rfr=RandomForestRegressor()
rfr.fit(X,y)
```

Out[97]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, 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=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [98]:
```
print('Mean Absolute Error: %.2f'%mean_absolute_error(y,rfr.predict(X)))
print('R2-Score: %.2f'%r2_score(y, rfr.predict(X)))
print('Mean Squared Error: %.3f'%mean_squared_error(y,rfr.predict(X)))
```

```
Mean Absolute Error: 0.01
R2-Score: 0.99
Mean Squared Error: 0.001
```

In [99]:
```
lr=LinearRegression()
lr.fit(X,y)
```

Out[99]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

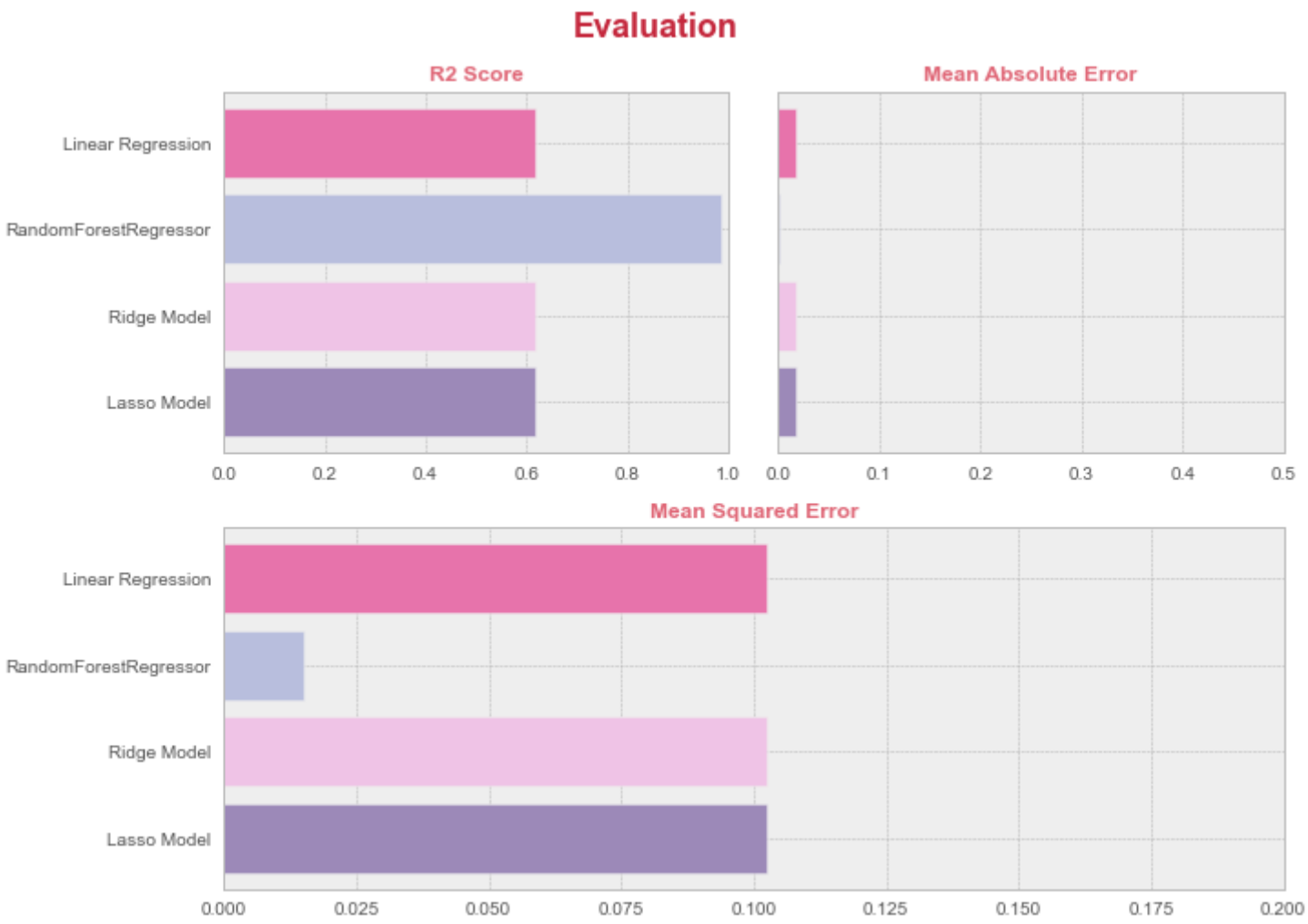
In [100]:
```
print('Mean Absolute Error: %.2f'%mean_absolute_error(y,lr.predict(X)))
print('R2-Score: %.2f'%r2_score(y, lr.predict(X)))
print('Mean Squared Error: %.2f'%mean_squared_error(y,lr.predict(X)))
```

```
Mean Absolute Error: 0.10
R2-Score: 0.62
Mean Squared Error: 0.02
```

In [101]:
```
evl=append_data(revl,levl)
evl=append_data(evaluate_model(X,y,'RandomForestRegressor',rfr),evl)
evl=append_data(evaluate_model(X,y,'Linear Regression',lr),evl)
```

In [102]:
```
# Let's Plot the model
from matplotlib.gridspec import GridSpec
def plot_regress(data):
    sns.set_palette(sns.color_palette("rocket"))
    super_title={'size':18,'color':'#c5283d','weight':'extra bold'}
    sub_title={'size':12,'color':'#e06777','weight':'bold'}
    colors=np.array([[156, 137, 184],[239, 195, 230],[184, 190, 221],[231, 115, 171]])
    colors=colors/255 #Matplotlib RGB color range is from 0-1
    data=pd.DataFrame(data)
    fig = plt.figure(figsize=(10,7),constrained_layout=True)
    gs = GridSpec(2, 2, figure=fig)
    ax1 = fig.add_subplot(gs[0, 0])
    ax1.barh(data.Model,data.R2_Score,color=colors)
    ax1.tick_params(labelbottom=True, labelleft=True)
    ax1.set_xlim(0,1)
    ax1.set_title('R2 Score',**sub_title)
    ax2 = fig.add_subplot(gs[0, 1])
    ax2.barh(data.Model,data.MAE,color=colors)
    ax2.tick_params(labelbottom=True, labelleft=False)
    ax2.set_xlim(0,0.5)
    ax2.set_title('Mean Absolute Error',**sub_title)
    ax3 = fig.add_subplot(gs[1, :])
    ax3.barh(data.Model,data.MSE,color=colors)
    ax3.tick_params(labelbottom=True, labelleft=True)
    ax3.set_xlim(0,0.2)
    ax3.set_title('Mean Squared Error',**sub_title)
    fig.suptitle("Evaluation",**super_title)
    ax4.tick_params(labelbottom=True, labelleft=False)
    plt.show()
```

In [103]:
```
plot_regress(evl)
```



conclusion: Overall, RandomForestRegressor is best.

Scenario-3: Toronto Rookies

The National Basketball Association (NBA) is a men's professional basketball league in North America, composed of 30 teams. It is one of the four major sports leagues in USA and Canada. The NBA is an active member of USA Basketball (USAB), which is recognized by FIBA (also known as the International Basketball Federation) as the national governing body for basketball in the United States.

Problem Statement:

The data contains the details of NBA rookies who have recently been part of NBA. We have to classify the career longevity of NBA rookies more than 5 years or not.

Dataset Description:

Attributes:

The dataset has details of NBA rookies along with their career longevity either more than 5 years or not:

- **Name**: Name of the players
- **GP**: Games played
- **MIN**: Total minutes played
- **PTS**: Points per game
- **FGM**: Field goals made
- **FGA**: Field goals attempted
- **FG%**: Field goals percent
- **3P Made**: 3 Points made
- **3PA**: 3 Points attempted
- **3P%**: 3 Points percentage
- **FTM**: Free throw made
- **FTA**: Free throw attempted
- **FT%**: Free throw percentage
- **OREB**: Offensive rebounds
- **DREB**: Defensive rebounds
- **REB**: Rebounds
- **AST**: Assists
- **STL**: Steals
- **BLK**: Blocks
- **TOV**: Turnovers

Target Variable:

- **TARGET_5Yrs**: 0 if career years played < 5 | 1 if career years played >= 5

Tasks to be Performed:

In order to attain the above goal below tasks must be performed:

- Read the dataset and pre-process the data. - **Beginner**
- Split the data into training and testing set; Apply Gaussian Naive Bayes and Decision Tree over the model. - **Intermediate**
- Evaluate the models using confusion matrix. - **Intermediate**
- Apply XGBoost Classifier over the data and evaluate the model. - **Advanced**
- Apply GridSearchCV over XGboost Classifier. **Advanced**

Topics Covered:

- XGBoost
- GridSearchCV
- Decision Tree
- Naive Bayes

Question-1: Read the dataset and pre-process the data.

In [104]:
```
data=pd.read_csv('NBA.csv')
```

In [105]:
```
data.head()
```

Out[105]:

	Name	GP	MIN	PTS	FGM	FGA	FG%	3P Made	3PA	3P%	...	FTA	FT%	OREB	DREB	REB	AST	STL	BLK	TOV	TARGET_5Yrs
0	Brandon Ingram	36	27.4	7.4	2.6	7.6	34.7	0.5	2.1	25.0	...	2.3	69.9	0.7	3.4	4.1	1.9	0.4	0.4	1.3	0.0
1	Andrew Harrison	35	26.9	7.2	2.0	6.7	29.6	0.7	2.8	23.5	...	3.4	76.5	0.5	2.0	2.4	3.7	1.1	0.5	1.6	0.0
2	JaKarr Sampson	74	15.3	5.2	2.0	4.7	42.2	0.4	1.7	24.4	...	1.3	67.0	0.5	1.7	2.2	1.0	0.5	0.3	1.0	0.0
3	Malik Sealy	58	11.6	5.7	2.3	5.5	42.6	0.1	0.5	22.6	...	1.3	68.9	1.0	0.9	1.9	0.8	0.6	0.1	1.0	1.0
4	Matt Geiger	48	11.5	4.5	1.6	3.0	52.4	0.0	0.1	0.0	...	1.9	67.4	1.0	1.5	2.5	0.3	0.3	0.4	0.8	1.0

5 rows × 21 columns

In [106]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1340 entries, 0 to 1339
Data columns (total 21 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Name         1340 non-null   object
 1   GP           1340 non-null   int64
 2   MIN          1340 non-null   float64
 3   PTS          1340 non-null   float64
 4   FGM          1340 non-null   float64
 5   FGA          1340 non-null   float64
 6   FG%          1340 non-null   float64
 7   3P Made      1340 non-null   float64
 8   3PA          1340 non-null   float64
 9   3P%          1329 non-null   float64
 10  FTM          1340 non-null   float64
 11  FTA          1340 non-null   float64
 12  FT%          1340 non-null   float64
 13  OREB         1340 non-null   float64
 14  DREB         1340 non-null   float64
 15  REB          1340 non-null   float64
 16  AST          1340 non-null   float64
 17  STL          1340 non-null   float64
 18  BLK          1340 non-null   float64
 19  TOV          1340 non-null   float64
 20  TARGET_5Yrs  1340 non-null   float64
dtypes: float64(19), int64(1), object(1)
memory usage: 220.0+ KB
```

In [107]:
```
data.describe(include='all')
```

Out[107]:

	Name	GP	MIN	PTS	FGM	FGA	FG%	3P Made	3PA	3P%	...
count	1340	1340.000000	1340.000000	1340.000000	1340.000000	1340.000000	1340.000000	1340.000000	1340.000000	1329.000000	...
unique	1294	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
top	Charles Smith	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
freq	9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
mean	NaN	60.414179	17.624627	6.801493	2.629104	5.885299	44.169403	0.247612	0.779179	19.308126	...
std	NaN	17.433992	8.307964	4.357545	1.683555	3.593488	6.137679	0.383688	1.061847	16.022916	...
min	NaN	11.000000	3.100000	0.700000	0.300000	0.800000	23.800000	0.000000	0.000000	0.000000	...
25%	NaN	47.000000	10.875000	3.700000	1.400000	3.300000	40.200000	0.000000	0.000000	0.000000	...
50%	NaN	63.000000	16.100000	5.550000	2.100000	4.800000	44.100000	0.100000	0.300000	22.400000	...
75%	NaN	77.000000	22.900000	8.800000	3.400000	7.500000	47.900000	0.400000	1.200000	32.500000	...
max	NaN	82.000000	40.900000	28.200000	10.200000	19.800000	73.700000	2.300000	6.500000	100.000000	...

	...
count	1340
unique	
top	
freq	
mean	1
std	1
min	0
25%	0
50%	1
75%	2
max	10

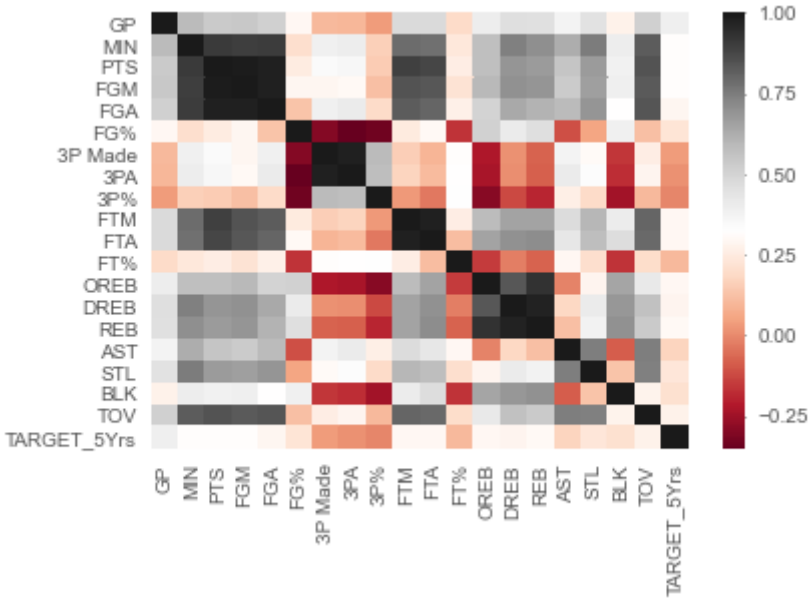
11 rows × 21 columns

In [108]:
```
import seaborn as sns
data.corr()
```

Out[108]:

	GP	MIN	PTS	FGM	FGA	FG%	3P Made	3PA	3P%	FTM	FTA	FT%	O
GP	1.000000	0.590240	0.538471	0.542724	0.516625	0.296289	0.107423	0.098772	0.038209	0.482123	0.479487	0.196299	0.40
MIN	0.590240	1.000000	0.911822	0.903060	0.910247	0.203901	0.389920	0.403258	0.165997	0.791000	0.779609	0.239878	0.57
PTS	0.538471	0.911822	1.000000	0.990834	0.979733	0.255333	0.346682	0.356751	0.151072	0.896297	0.880703	0.258931	0.57
FGM	0.542724	0.903060	0.990834	1.000000	0.980050	0.291693	0.289007	0.299057	0.119493	0.848019	0.840408	0.223566	0.59
FGA	0.516625	0.910247	0.979733	0.980050	1.000000	0.129798	0.390253	0.413560	0.197160	0.826616	0.805559	0.269614	0.50
FG%	0.296289	0.203901	0.255333	0.291693	0.129798	1.000000	-0.294471	-0.350658	-0.330690	0.245776	0.300154	-0.161183	0.51
3P Made	0.107423	0.389920	0.346682	0.289007	0.390253	-0.294471	1.000000	0.982616	0.589855	0.158472	0.095396	0.314355	-0.21
3PA	0.098772	0.403258	0.356751	0.299057	0.413560	-0.350658	0.982616	1.000000	0.582337	0.173533	0.108388	0.323612	-0.23
3P%	0.038209	0.165997	0.151072	0.119493	0.197160	-0.330690	0.589855	0.582337	1.000000	0.030320	-0.032333	0.326372	-0.28
FTM	0.482123	0.791000	0.896297	0.848019	0.826616	0.245776	0.158472	0.173533	0.030320	1.000000	0.980505	0.257818	0.58
FTA	0.479487	0.779609	0.880703	0.840408	0.805559	0.300154	0.095396	0.108388	-0.032333	0.980505	1.000000	0.114872	0.65
FT%	0.196299	0.239878	0.258931	0.223566	0.269614	-0.161183	0.314355	0.323612	0.326372	0.257818	0.114872	1.000000	-0.14
OREB	0.401136	0.573062	0.575106	0.596687	0.504212	0.511367	-0.219010	-0.231897	-0.288759	0.583865	0.653445	-0.146786	1.00
DREB	0.466840	0.745513	0.693934	0.703278	0.640123	0.410555	0.016570	0.011226	-0.122949	0.653823	0.700863	-0.022905	0.83
REB	0.460406	0.709707	0.676849	0.691186	0.614328	0.465423	-0.072503	-0.080939	-0.191071	0.653833	0.711425	-0.071105	0.93
AST	0.372749	0.629015	0.552338	0.532534	0.589818	-0.108797	0.376604	0.410531	0.262120	0.476214	0.428624	0.296315	-0.01
STL	0.451137	0.757034	0.675341	0.662640	0.690168	0.056658	0.306908	0.338631	0.194329	0.600158	0.580065	0.207205	0.28
BLK	0.276498	0.399088	0.387043	0.398125	0.322184	0.391626	-0.158535	-0.172150	-0.242274	0.407466	0.468974	-0.161152	0.64
TOV	0.518167	0.826500	0.850366	0.834352	0.845989	0.121806	0.258369	0.283925	0.108277	0.804990	0.798936	0.199742	0.42
TARGET_5Yrs	0.396833	0.317805	0.315981	0.317594	0.292660	0.227134	0.036619	0.018110	-0.003411	0.296841	0.296089	0.106706	0.29

In [109]:
```
sns.heatmap(data.corr(),cmap='RdGy')
plt.show()
```



- MIN column is highly correlated with PTS, FGA, FGM
- FTM and FTA columns are highly correlated

In [110]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[110]:

	Col_name	Missing value?	Count_	Percentage
9	3P%	True	11	0.008209
0	Name	False	0	0.000000
11	FTA	False	0	0.000000
19	TOV	False	0	0.000000
18	BLK	False	0	0.000000
17	STL	False	0	0.000000
16	AST	False	0	0.000000
15	REB	False	0	0.000000
14	DREB	False	0	0.000000
13	OREB	False	0	0.000000
12	FT%	False	0	0.000000
10	FTM	False	0	0.000000
1	GP	False	0	0.000000
8	3PA	False	0	0.000000
7	3P Made	False	0	0.000000
6	FG%	False	0	0.000000
5	FGA	False	0	0.000000
4	FGM	False	0	0.000000
3	PTS	False	0	0.000000
2	MIN	False	0	0.000000
20	TARGET_5Yrs	False	0	0.000000

In [111]:
```
data['3P%'].fillna(data['3P%'].mean(),inplace=True)
```

In [112]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[112]:

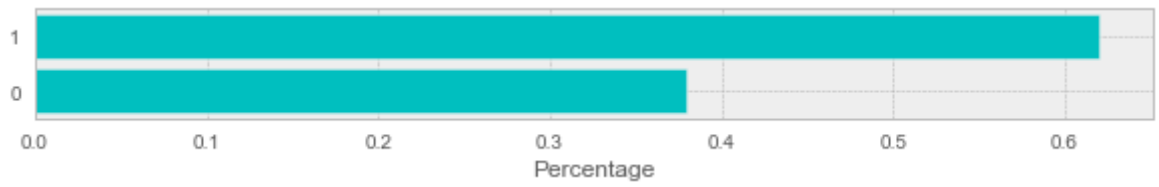
	Col_name	Missing value?	Count_	Percentage
0	Name	False	0	0.0
11	FTA	False	0	0.0
19	TOV	False	0	0.0
18	BLK	False	0	0.0
17	STL	False	0	0.0
16	AST	False	0	0.0
15	REB	False	0	0.0
14	DREB	False	0	0.0
13	OREB	False	0	0.0
12	FT%	False	0	0.0
10	FTM	False	0	0.0
1	GP	False	0	0.0
9	3P%	False	0	0.0
8	3PA	False	0	0.0
7	3P Made	False	0	0.0
6	FG%	False	0	0.0
5	FGA	False	0	0.0
4	FGM	False	0	0.0
3	PTS	False	0	0.0
2	MIN	False	0	0.0
20	TARGET_5Yrs	False	0	0.0

In [113]:
```
target_ratio=pd.DataFrame({'Counts':data.TARGET_5Yrs.value_counts(),
                           'Percentage':data.TARGET_5Yrs.value_counts()/len(data)})
target_ratio
```

Out[113]:

	Counts	Percentage
1.0	831	0.620149
0.0	509	0.379851

In [114]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (10,1))
plt.barh(target_ratio.index, target_ratio.Percentage,color='c')
plt.xlabel('Percentage')
plt.show()
```



In [115]:
```
import pandas_profiling
rpt=pandas_profiling.ProfileReport(data)
rpt
```

Overview

Dataset statistics

Number of variables	21
Number of observations	1340
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	12
Duplicate rows (%)	0.9%
Total size in memory	220.0 KiB
Average record size in memory	168.1 B

Variable types

NUM	19
BOOL	1
CAT	1

Reproduction

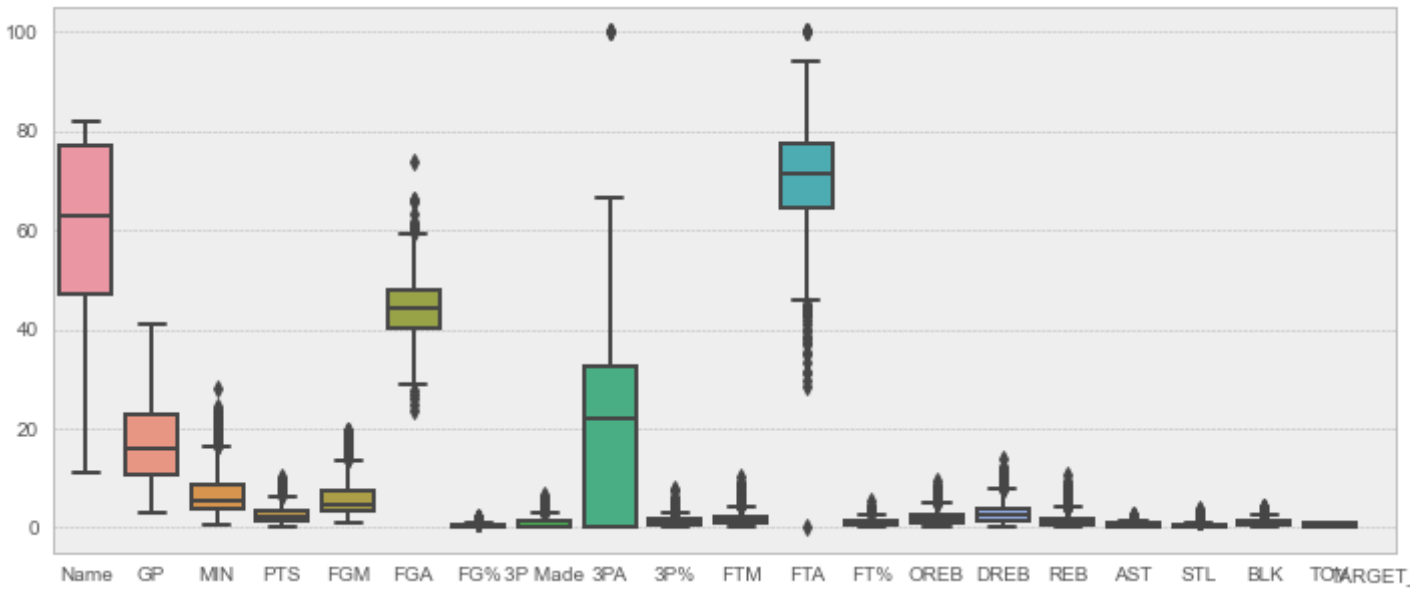
Analysis started	2020-07-14 14:20:52.753239
Analysis finished	2020-07-14 14:21:53.523884
Duration	1 minute and 0.77 seconds
Version	pandas-profiling v2.8.0 (https://github.com/pandas-profiling/pandas-profiling)
Command line	`pandas_profiling --config_file config.yaml [YOUR_FILE.csv]`

Out[115]:

Question-2: Split the data into training and testing set; Apply Gaussian Naive Bayes and Decision Tree over the model.

In [116]:
```
feat=data.columns
```

In [117]:
```
# Before splitting lets see the data distribution.
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12,5))
sns.boxplot(data=data.drop('Name',axis=1))
plt.xticks(ticks=np.arange(len(feat)),labels=feat)
plt.show()
```



In [118]:
```
data.shape
```

Out[118]: (1340, 21)

In [119]:
```
# Removing outliers
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
data_o=data[~((data < (Q1 - 1.5 * IQR)) |(data > (Q3 + 1.5 * IQR))).any(axis=1)]
```

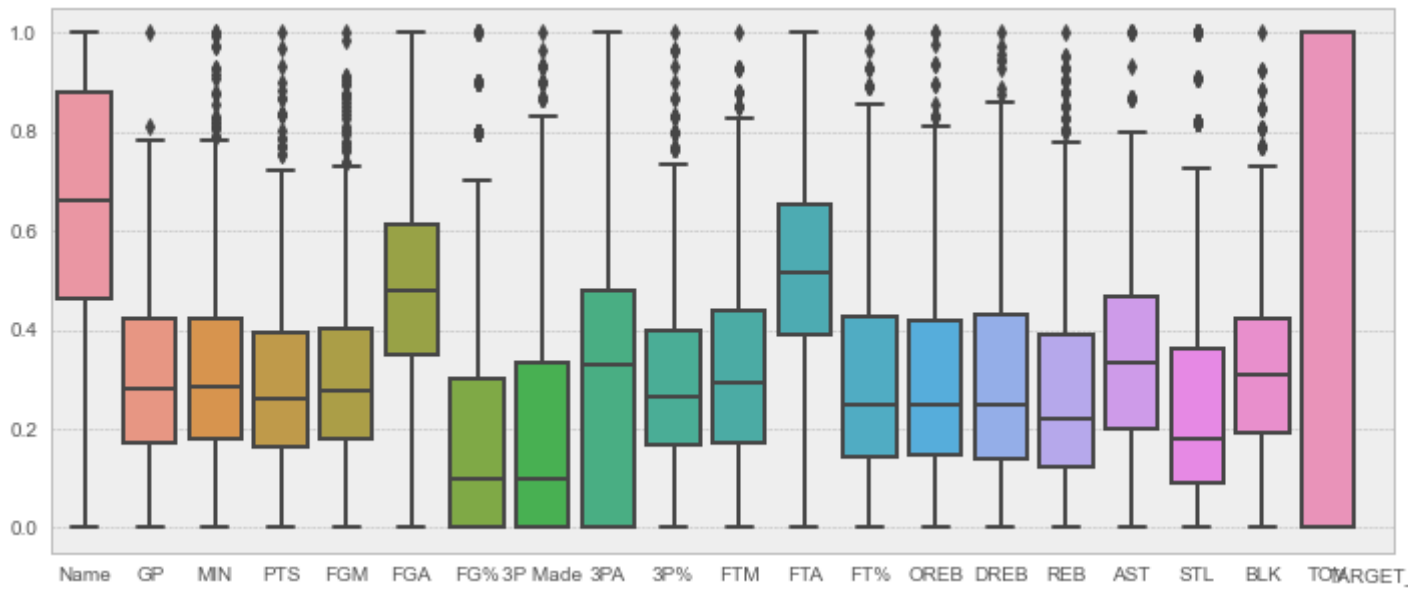
```
GP            30.000
MIN           12.025
PTS            5.100
FGM            2.000
FGA            4.200
FG%            7.700
3P Made        0.400
3PA            1.200
3P%           32.500
FTM            1.000
FTA            1.400
FT%           12.900
OREB           1.000
DREB           1.600
REB            2.500
AST            1.400
STL            0.500
BLK            0.400
TOV            0.800
TARGET_5Yrs    1.000
dtype: float64
```

In [120]:
```
data_o.shape
```

Out[120]: (991, 21)

Data is highly unbalanced.

In [121]:
```
# Scaling data after outlier removal
scl=MinMaxScaler()

scl_data=scl.fit_transform(data_o.drop('Name',axis=1))

plt.figure(figsize=(12,5))
sns.boxplot(data=scl_data)
plt.xticks(ticks=np.arange(len(feat)),labels=feat)
plt.show()
```



In [122]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

In [123]:
```
X=scl_data[:,:19]
y=scl_data[:,19]
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,test_size=0.3, random_state=101)
```

In [124]:
```
gnb=GaussianNB()
gnb.fit(X_train,y_train)
g_pred=gnb.predict(X_test)
```

In [125]:
```
dnb=DecisionTreeClassifier()
dnb.fit(X_train,y_train)
d_pred=dnb.predict(X_test)
```

In [126]:
```
# Let's Try using RandomForest
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
r_pred=rf.predict(X_test)
```

In [127]:
```
# Let's try using Logistic Model as well
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(X_train,y_train)
l_pred=lr.predict(X_test)
```

Question-3: Evaluate the models using confusion matrix.

In [128]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [129]:
```
print('The accuracy of the Gaussian Naive Bayes model is: ',round(accuracy_score(y_test,g_pred)*100,2))
print('The accuracy of the Decision Tree model is: ',round(accuracy_score(y_test,d_pred)*100,2))
print('The accuracy of the Random Forest model is: ',round(accuracy_score(y_test,r_pred)*100,2))
print('The accuracy of the Logistic Regression model is: ',round(accuracy_score(y_test,l_pred)*100,2))
```

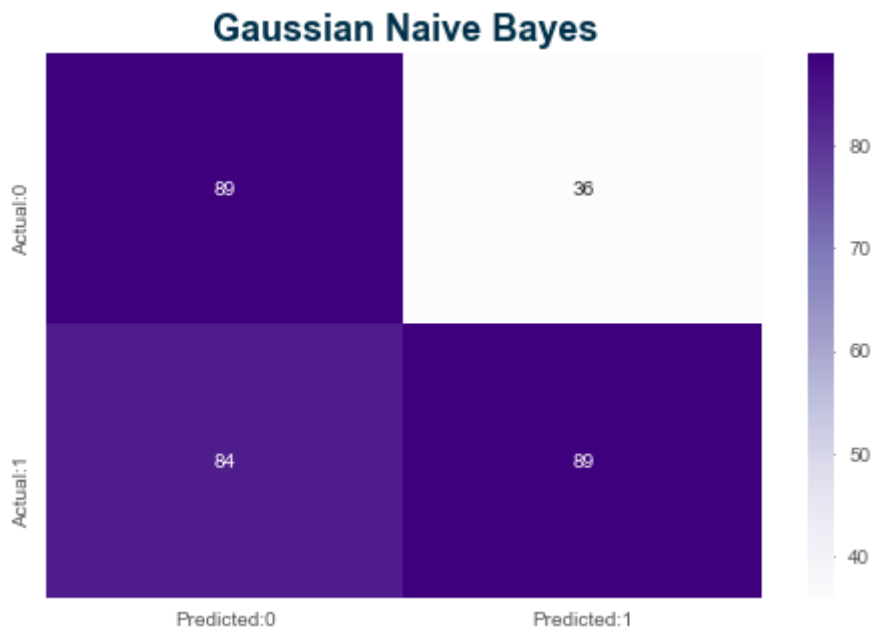
```
The accuracy of the Gaussian Naive Bayes model is:  59.73
The accuracy of the Decision Tree model is:  58.05
The accuracy of the Random Forest model is:  64.43
The accuracy of the Logistic Regression model is:  68.46
```

In [130]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,d_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap='Oranges')
fm={'size':18,'color':'#f4a261','weight':'bold'}
plt.title('Decision Tree',**fm)
plt.show()
```

In [131]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 72
True Positive 101
False Negative 72
False Positive 53
Sensitivity 0.5838150289017341
Specificity 0.576
```

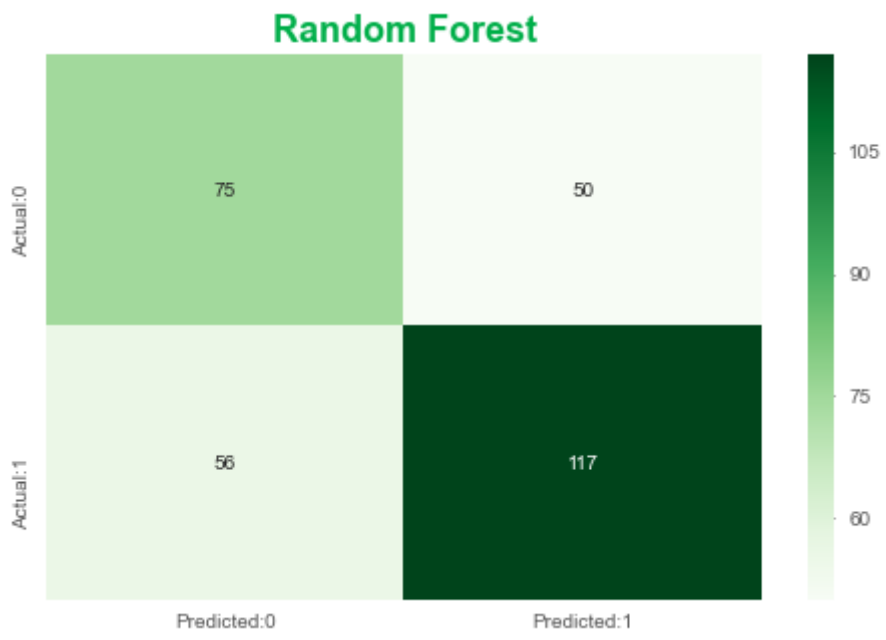
In [132]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,g_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap='Purples')
fm={'size':18,'color':'#003049','weight':'bold'}
plt.title('Gaussian Naive Bayes',**fm)
plt.show()
```



In [133]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 89
True Positive 89
False Negative 84
False Positive 36
Sensitivity 0.5144508670520231
Specificity 0.712
```

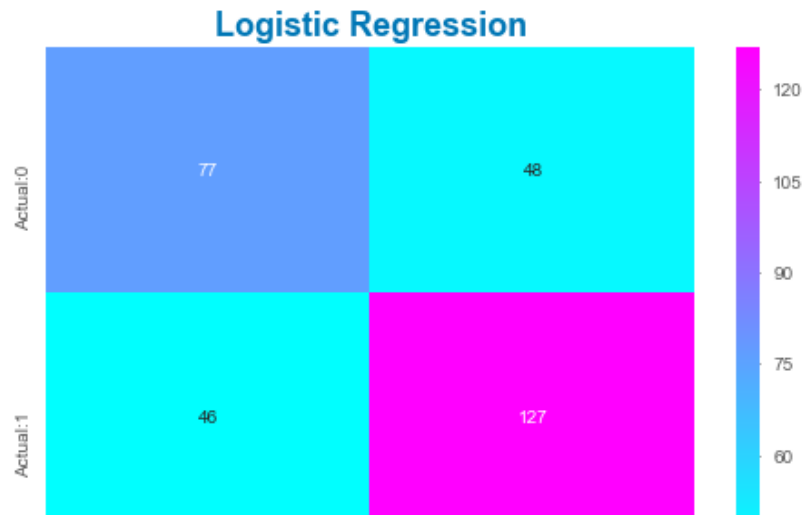
In [134]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,r_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap='Greens')
fm={'size':18,'color':'#00af49','weight':'bold'}
plt.title('Random Forest',**fm)
plt.show()
```



In [135]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 75
True Positive 117
False Negative 56
False Positive 50
Sensitivity 0.6763005780346821
Specificity 0.6
```

In [136]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,l_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap='cool')
fm={'size':18,'color':'#0077B6','weight':'bold'}
plt.title('Logistic Regression',**fm)
plt.show()
```



In [137]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Positive", TP)
print("False Negative", FN)
print("False Positive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 77
True Positive 127
False Negative 46
False Positive 48
Sensitivity 0.7341040462427746
Specificity 0.616
```

In [138]:
```
from sklearn.metrics import precision_score, classification_report,f1_score,recall_score
```

In [139]:
```
print('\t\tClassification Report for Gaussian Naive Bayes Model')
print(classification_report(y_test,g_pred))
```

```
                Classification Report for Gaussian Naive Bayes Model
              precision    recall  f1-score   support

         0.0       0.51      0.71      0.60       125
         1.0       0.71      0.51      0.60       173

    accuracy                           0.60       298
   macro avg       0.61      0.61      0.60       298
weighted avg       0.63      0.60      0.60       298
```

In [140]:
```
print('\t\tClassification Report for Decision Tree Model')
print(classification_report(y_test,d_pred))
```

```
                Classification Report for Decision Tree Model
              precision    recall  f1-score   support

         0.0       0.50      0.58      0.54       125
         1.0       0.66      0.58      0.62       173

    accuracy                           0.58       298
   macro avg       0.58      0.58      0.58       298
weighted avg       0.59      0.58      0.58       298
```

In [141]:
```
print('\t\tClassification Report for RandomForest Model')
print(classification_report(y_test,r_pred))
```

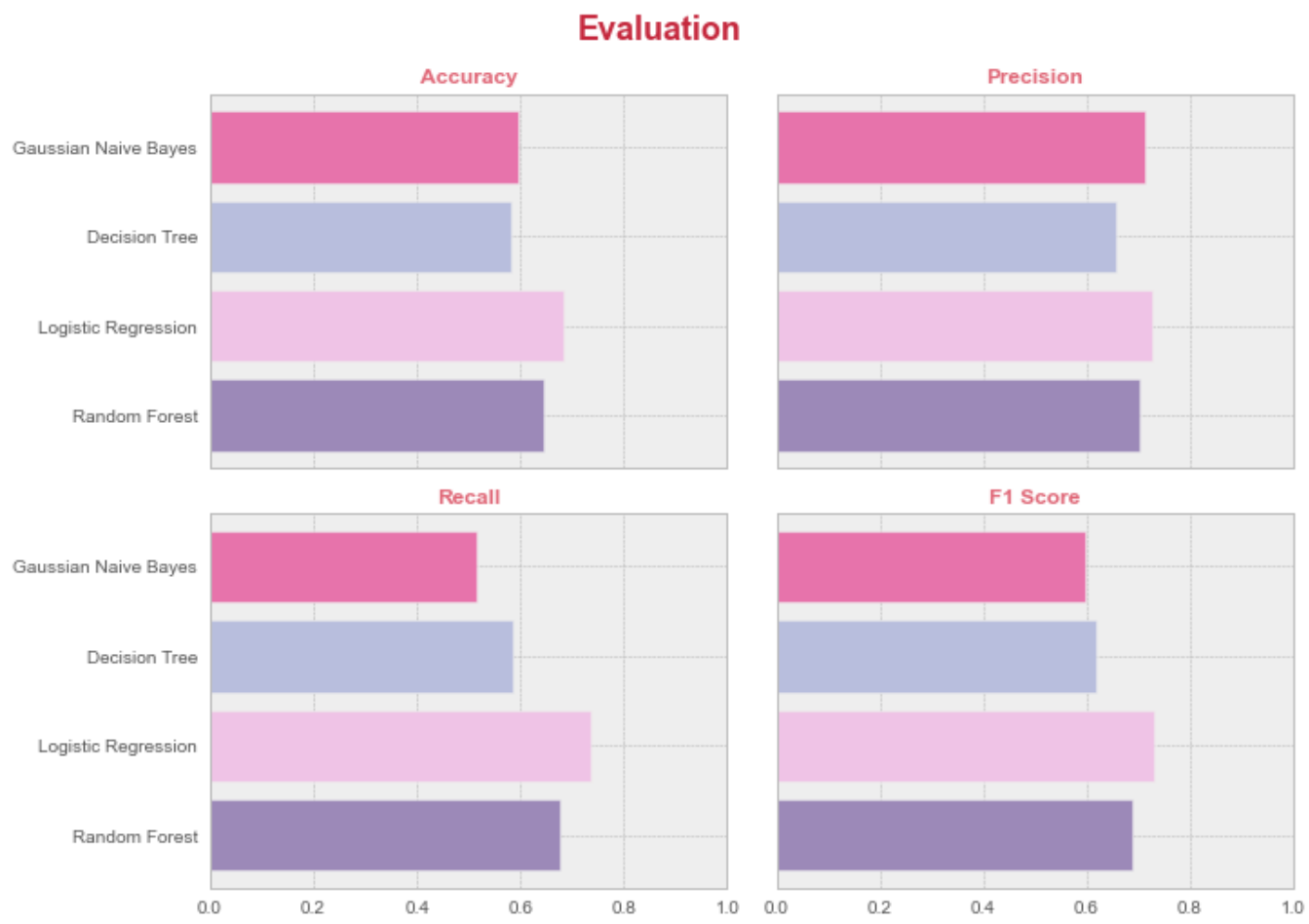
```
                Classification Report for RandomForest Model
              precision    recall  f1-score   support

         0.0       0.57      0.60      0.59       125
         1.0       0.70      0.68      0.69       173

    accuracy                           0.64       298
   macro avg       0.64      0.64      0.64       298
weighted avg       0.65      0.64      0.65       298
```

In [142]:
```
print('\t\tClassification Report for Logistic Model')
print(classification_report(y_test,l_pred))
```

```
                Classification Report for Logistic Model
              precision    recall  f1-score   support

         0.0       0.63      0.62      0.62       125
         1.0       0.73      0.73      0.73       173

    accuracy                           0.68       298
   macro avg       0.68      0.68      0.68       298
weighted avg       0.68      0.68      0.68       298
```

In [143]:
```
evl_l=insert_data(y_test,l_pred,'Logistic Regression')
evl_r=insert_data(y_test,r_pred,'Random Forest')
evl_all=append_data(evl_l,evl_r)
evl_all=append_data(insert_data(y_test,d_pred,'Decision Tree'),evl_all)
evl_all=append_data(insert_data(y_test,g_pred,'Gaussian Naive Bayes'),evl_all)
```

In [144]:
```
plot_models(evl_all)
```



Conclusion: From the above graph we can see that logistic regression is performing fairly well enough.

Question-4: Apply XGBoost Classifier over the data and evaluate the model.

In [145]:
```
from xgboost import XGBClassifier
```

In [146]:
```
xb_clf=XGBClassifier(learning_rate=0.65,n_estimators=1000)
xb_clf.fit(X_train,y_train)
xb_pred=xb_clf.predict(X_test)
```

In [147]:
```
print('The accuracy of the XGBoost model is: ',round(accuracy_score(y_test,xb_pred)*100,2))
```

```
The accuracy of the XGBoost model is:  62.42
```

In [148]:
```
print('\t\tClassification Report for XGBoost Model')
print(classification_report(y_test,xb_pred))
```

```
                Classification Report for XGBoost Model
              precision    recall  f1-score   support

         0.0       0.56      0.52      0.54       125
         1.0       0.67      0.70      0.68       173

    accuracy                           0.62       298
   macro avg       0.61      0.61      0.61       298
weighted avg       0.62      0.62      0.62       298
```

Question-5: Apply GridSearchCV over XGboost Classifier.

In [149]:
```
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

In [150]:
```
params={'learning_rate':[0.65,0.70,0.75],'n_estimators':[100,1000]}
rf_gsv=GridSearchCV(estimator=XGBClassifier(),param_grid=params,cv=3,scoring='accuracy')
rf_gsv.fit(X,y)
```

Out[150]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estim...
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=None, verbosity=None),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.65, 0.7, 0.75],
                         'n_estimators': [100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [151]:
```
pd.DataFrame(rf_gsv.cv_results_).sort_values('rank_test_score')
```

Out[151]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_n_estimators	params	split0_test_score	split1
0	0.078039	0.000119	0.005208	0.007366	0.65	100	{'learning_rate': 0.65, 'n_estimators': 100}	0.613293	
1	0.558632	0.052515	0.010560	0.007469	0.65	1000	{'learning_rate': 0.65, 'n_estimators': 1000}	0.631420	
3	0.550474	0.045305	0.000000	0.000000	0.7	1000	{'learning_rate': 0.7, 'n_estimators': 1000}	0.625378	
5	0.540443	0.046753	0.007543	0.006392	0.75	1000	{'learning_rate': 0.75, 'n_estimators': 1000}	0.619335	
4	0.080072	0.002762	0.000000	0.000000	0.75	100	{'learning_rate': 0.75, 'n_estimators': 100}	0.619335	
2	0.081598	0.008853	0.000000	0.000000	0.7	100	{'learning_rate': 0.7, 'n_estimators': 100}	0.592145	

In [152]:
```
cvr=pd.DataFrame(rf_gsv.cv_results_)
```

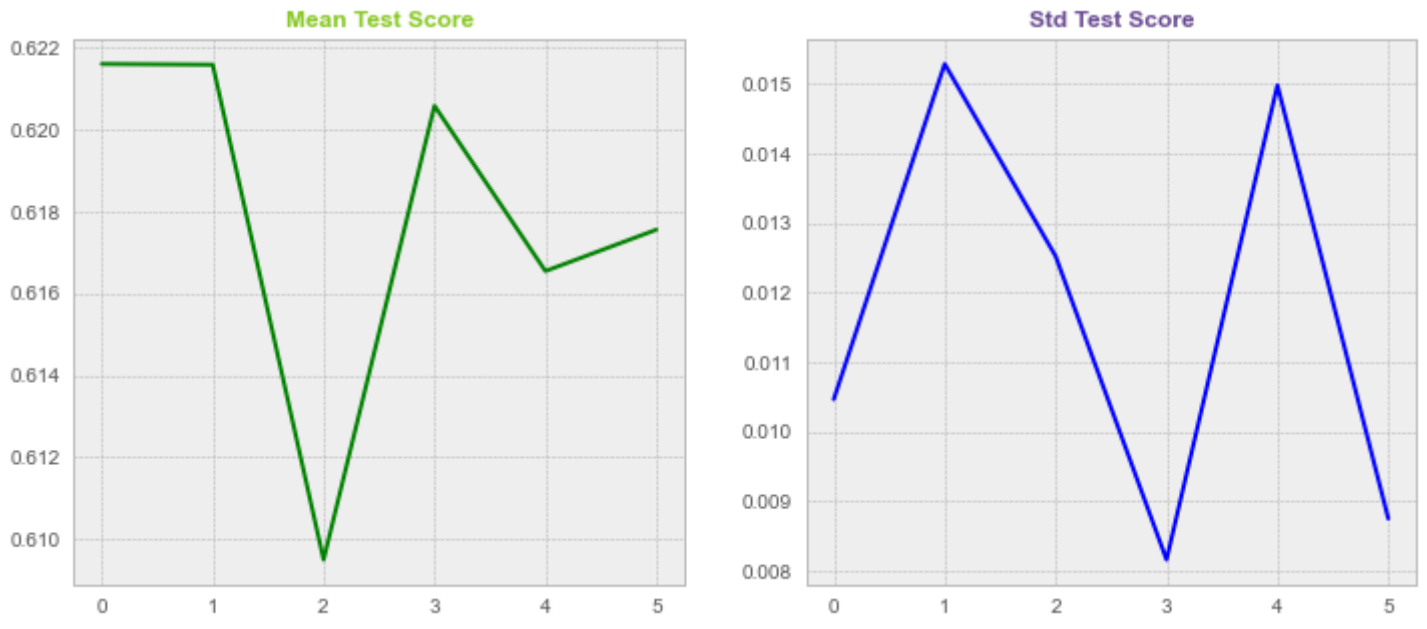
In [153]:
```
rf_gsv.best_params_
```

Out[153]:
```
{'learning_rate': 0.65, 'n_estimators': 100}
```

In [154]:
```
rf_gsv.best_estimator_
```

Out[154]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.65, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [155]:
```
from matplotlib.gridspec import GridSpec
fig=plt.figure(figsize=(12,5))
gs=GridSpec(1,2,fig)
ax1=fig.add_subplot(gs[0,0])
ax1.plot(cvr.mean_test_score,c='g')
ax1.set_title('Mean Test Score',**format_text(12,'8AC926'))
ax2=fig.add_subplot(gs[0,1])
ax2.plot(cvr.std_test_score,c='b')
ax2.set_title('Std Test Score',**format_text(12,'6A4C93'))
plt.show()
```



Model_0 has lost deviation as well as high accuracy.

Scenario-4: Shurima Surveys

Shurima Surveys is a non-government organization with the mission to empower people and communities in state of poverty, illiteracy, disease and social injustice. But the organization specializes in suicide preventions and mental healthcare. Surveys were conducted on people living in different rural zones. They record on the basis of there lifetyle, assets, income and so on.

Problem Statement:

The aim is to classify either the individual is depressed or not based on the data recorded.

Dataset Description:

The dataset contains various details of individuals along with there emotional status i.e. either they are depressed or not. The details recorded during the survey were:

Attributes:

- **Survey_id**: ID of the survey conducted
- **Ville_id**: Village ID where the survey was conducted(1-292)
- **sex**: Male or Female
- **Age**: Age of the individual(17-91)
- **Married**: Either the individual is married or not
- **Number_children**: Number of children the individual has(0-11)
- **education_level**: Education level(1-19)
- **total_members**: Total number of members in the family(1-12)
- **gained_asset**: Total assets gained(325K-99.1M)
- **durable_asset**: Total durable assets gained(163K-99.6M)
- **save_asset**: Assets saved(173K-99.9M)
- **living_expenses**: Expense of living(263K-99.3M)
- **other_expenses**: Other expenses(173K-99.8M)
- **Salaried**: Salaried or not
- **incoming_own_farm**: Does individual have a farm or not
- **incoming_business**: Any Business
- **incoming_agricultural**: Agricultural income(325K-99.8M)
- **farm_expenses**: Farming expenses(272K-99.7K)
- **lasting_investment**: Lasting investments(74.3K-99.4M)
- **no_lasting_investmen**: Non-Lasting investments(126K-99.6M)

Target Variable:

- **depressed**: Either the individual is depressed or not

Tasks to be Perfromed:

Inorder to attain the above goal below tasks must be performed:

- Read the data and remove all the missing values. - **Beginner**
- Pre-process the data using LabelEncoder and split into training and testing set. - **Beginner**
- Apply GradientBoosting Classifier and evaluate the model. - **Intermediate**
- Apply AdaBoost Classifier and evaluate the model. - **Intermediate**
- Find the ROC-AUC Score of the models and plot on ROC-AUC Curve. - **Advanced**

Topics Covered:

- ROC-AUC
- AdaBoost
- GradientBoost

Question -1: Read the data and remove all the missing values.

In [156]:
```
import pandas as pd
import numpy as np
```

In [157]:
```
data=pd.read_csv('Depressed.csv')
data.head()
```

Out[157]:

	Survey_id	Ville_id	sex	Age	Married	Number_children	education_level	total_members	gained_asset	durable_asset	...	living_expenses
0	926	91	Male	28	Married	4	10	5	28912201	22861940	...	26692283
1	747	57	Male	23	Married	3	8	5	28912201	22861940	...	26692283
2	1190	115	Male	22	Married	3	9	5	28912201	22861940	...	26692283
3	1065	97	Male	27	Married	2	10	4	52667108	19698904	...	397715
4	806	42	Female	59	Not Married	4	10	6	82606287	17352654	...	80877619

5 rows × 21 columns

In [158]:
```
data.shape
```

Out[158]: (1429, 21)

In [159]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1429 entries, 0 to 1428
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Survey_id              1429 non-null   int64
 1   Ville_id               1429 non-null   int64
 2   sex                    1429 non-null   object
 3   Age                    1429 non-null   int64
 4   Married                1429 non-null   object
 5   Number_children        1429 non-null   int64
 6   education_level        1429 non-null   int64
 7   total_members          1429 non-null   int64
 8   gained_asset           1429 non-null   int64
 9   durable_asset          1429 non-null   int64
 10  save_asset             1429 non-null   int64
 11  living_expenses        1429 non-null   int64
 12  other_expenses         1429 non-null   int64
 13  Salaried               1429 non-null   object
 14  incoming_own_farm      1429 non-null   object
 15  incoming_business      1429 non-null   object
 16  incoming_agricultural  1429 non-null   int64
 17  farm_expenses          1429 non-null   int64
 18  lasting_investment     1429 non-null   int64
 19  no_lasting_investmen   1409 non-null   float64
 20  depressed              1429 non-null   object
dtypes: float64(1), int64(14), object(6)
memory usage: 234.6+ KB
```

In [160]:
```
data.describe(include='all')
```

Out[160]:

	Survey_id	Ville_id	sex	Age	Married	Number_children	education_level	total_members	gained_asset	durable_asset	...
count	1429.00000	1429.000000	1429	1429.000000	1429	1429.000000	1429.000000	1429.000000	1.429000e+03	1.429000e+03	...
unique	NaN	NaN	2	NaN	2	NaN	NaN	NaN	NaN	NaN	...
top	NaN	NaN	Male	NaN	Married	NaN	NaN	NaN	NaN	NaN	...
freq	NaN	NaN	1312	NaN	1104	NaN	NaN	NaN	NaN	NaN	...
mean	715.00000	76.286214	NaN	34.777467	NaN	2.883135	8.687194	4.969209	3.363448e+07	2.717296e+07	...
std	412.66108	66.444012	NaN	13.986219	NaN	1.874472	2.923532	1.786317	2.003854e+07	1.815672e+07	...
min	1.00000	1.000000	NaN	17.000000	NaN	0.000000	1.000000	1.000000	3.251120e+05	1.625560e+05	...
25%	358.00000	24.000000	NaN	25.000000	NaN	2.000000	8.000000	4.000000	2.326982e+07	1.929852e+07	...
50%	715.00000	57.000000	NaN	30.000000	NaN	3.000000	9.000000	5.000000	2.891220e+07	2.286194e+07	...
75%	1072.00000	105.000000	NaN	42.000000	NaN	4.000000	10.000000	6.000000	3.717283e+07	2.656950e+07	...
max	1429.00000	292.000000	NaN	91.000000	NaN	11.000000	19.000000	12.000000	9.912755e+07	9.961560e+07	...

11 rows × 21 columns

In [161]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[161]:

	Col_name	Missing value?	Count_	Percentage
19	no_lasting_investmen	True	20	0.013996
0	Survey_id	False	0	0.000000
11	living_expenses	False	0	0.000000
18	lasting_investment	False	0	0.000000
17	farm_expenses	False	0	0.000000
16	incoming_agricultural	False	0	0.000000
15	incoming_business	False	0	0.000000
14	incoming_own_farm	False	0	0.000000
13	Salaried	False	0	0.000000
12	other_expenses	False	0	0.000000
10	save_asset	False	0	0.000000
1	Ville_id	False	0	0.000000
9	durable_asset	False	0	0.000000
8	gained_asset	False	0	0.000000
7	total_members	False	0	0.000000
6	education_level	False	0	0.000000
5	Number_children	False	0	0.000000
4	Married	False	0	0.000000
3	Age	False	0	0.000000
2	sex	False	0	0.000000
20	depressed	False	0	0.000000

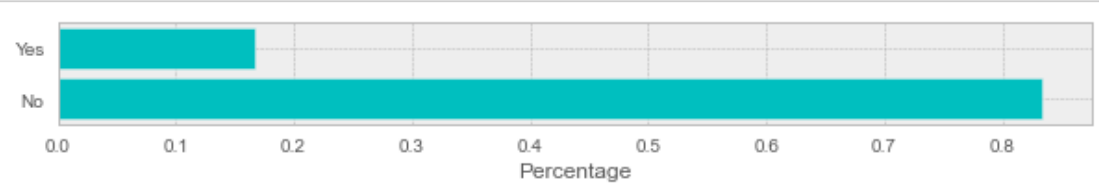
In [162]:
```
data.no_lasting_investmen.fillna(data.no_lasting_investmen.mean(),inplace=True)
```

In [163]:
```
miss=pd.DataFrame({'Col_name':data.columns,'Missing value?':
                           [any(data[x].isnull()) for x in data.columns],
                   'Count_':[sum(data[y].isnull()) for y in data.columns],
                           'Percentage':[sum(data[y].isnull())/data.shape[0] for y in data.columns]})
miss.sort_values(by='Count_',ascending=False)
```

Out[163]:

	Col_name	Missing value?	Count_	Percentage
0	Survey_id	False	0	0.0
11	living_expenses	False	0	0.0
19	no_lasting_investmen	False	0	0.0
18	lasting_investment	False	0	0.0
17	farm_expenses	False	0	0.0
16	incoming_agricultural	False	0	0.0
15	incoming_business	False	0	0.0
14	incoming_own_farm	False	0	0.0
13	Salaried	False	0	0.0
12	other_expenses	False	0	0.0
10	save_asset	False	0	0.0
1	Ville_id	False	0	0.0
9	durable_asset	False	0	0.0
8	gained_asset	False	0	0.0
7	total_members	False	0	0.0
6	education_level	False	0	0.0
5	Number_children	False	0	0.0
4	Married	False	0	0.0
3	Age	False	0	0.0
2	sex	False	0	0.0
20	depressed	False	0	0.0

In [164]:
```
#Check distribution of target variable
target_ratio=pd.DataFrame({'Counts':data.depressed.value_counts(),'Percentage':data.depressed.value_counts()/len(data
)})
target_ratio

plt.figure(figsize = (10,1))
plt.barh(target_ratio.index, target_ratio.Percentage,color='c')
plt.xlabel('Percentage')
plt.show()
```



Question-2: Pre-process the data using LabelEncoder and split into training and testing set.

In [165]:
```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
```

In [166]:
```
lb=LabelEncoder()
data.depressed=lb.fit_transform(data.depressed)
```

In [167]:
```
lb.classes_
```

Out[167]:
```
array(['No', 'Yes'], dtype=object)
```

In [168]:
```
scale_cols=['Age','gained_asset', 'durable_asset', 'save_asset',
       'living_expenses', 'other_expenses', 'incoming_agricultural',
       'farm_expenses', 'lasting_investment', 'no_lasting_investmen',]
data[scale_cols]=MinMaxScaler().fit_transform(data[scale_cols])
```

In [169]:
```
cols=['sex','Married','Salaried','incoming_own_farm','incoming_business']
data[cols]=data[cols].apply(LabelEncoder().fit_transform)
```

In [170]:
```
X=data.drop(['depressed','Survey_id','Ville_id'],axis=1)
y=data.depressed
```

In [171]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=2,test_size=0.3)
```

Question-3: Apply GradientBoosting Classifier and evaluate the model.

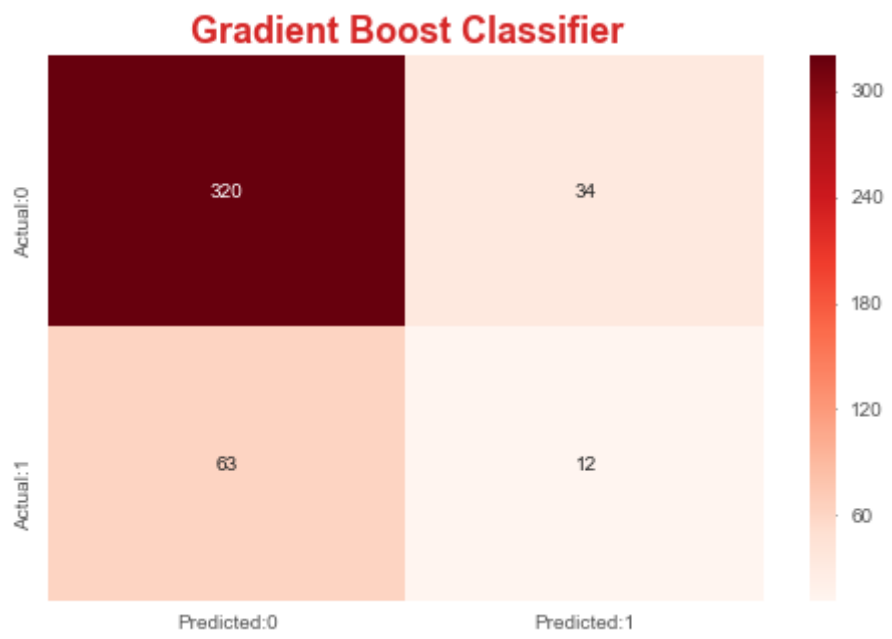
In [172]:
```
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
```

In [173]:
```
gb_clf=GradientBoostingClassifier(learning_rate=0.5,n_estimators=1000)
gb_clf.fit(X_train,y_train)
gb_pred=gb_clf.predict(X_test)
```

In [174]:
```
print('The accuracy of the GradientBoostingClassifier model is: ',round(accuracy_score(y_test,gb_pred)*100,2))
```

```
The accuracy of the GradientBoostingClassifier model is:  77.39
```

In [175]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,gb_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap='Reds')
fm={'size':18,'color':'#d62828','weight':'bold'}
plt.title('Gradient Boost Classifier',**fm)
plt.show()
```



In [176]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Posetive", TP)
print("False Negative", FN)
print("False Posetive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

```
True Negative 320
True Posetive 12
False Negative 63
False Posetive 34
Sensitivity 0.16
Specificity 0.903954802259887
```

The model is highly specific.

In [177]:
```
from sklearn.metrics import precision_score,recall_score, f1_score
```

In [178]:
```
print('The precision of the GradientBoostingClassifier model is: ',round(precision_score(y_test,gb_pred)*100,2))
print('The recall of the GradientBoostingClassifier model is: ',round(recall_score(y_test,gb_pred)*100,2))
print('The F1_Score of the GradientBoostingClassifier model is: ',round(f1_score(y_test,gb_pred)*100,2))
print('\n\n\t\tGradient Boosting Classifier Classification Report')
print(classification_report(y_test,gb_pred))
```

```
The precision of the GradientBoostingClassifier model is:  26.09
The recall of the GradientBoostingClassifier model is:  16.0
The F1_Score of the GradientBoostingClassifier model is:  19.83


                Gradient Boosting Classifier Classification Report
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       354
           1       0.26      0.16      0.20        75

    accuracy                           0.77       429
   macro avg       0.55      0.53      0.53       429
weighted avg       0.74      0.77      0.75       429
```

In [179]:
```
evl_gb=insert_data(y_test,gb_pred,'Gradient Boosting Model')
```

Question-4: Apply AdaBoost Classifier and evaluate the model.

In [180]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [181]:
```
ada=AdaBoostClassifier(n_estimators=100,learning_rate=0.6)
ada.fit(X_train,y_train)
ada_pred=ada.predict(X_test)
```

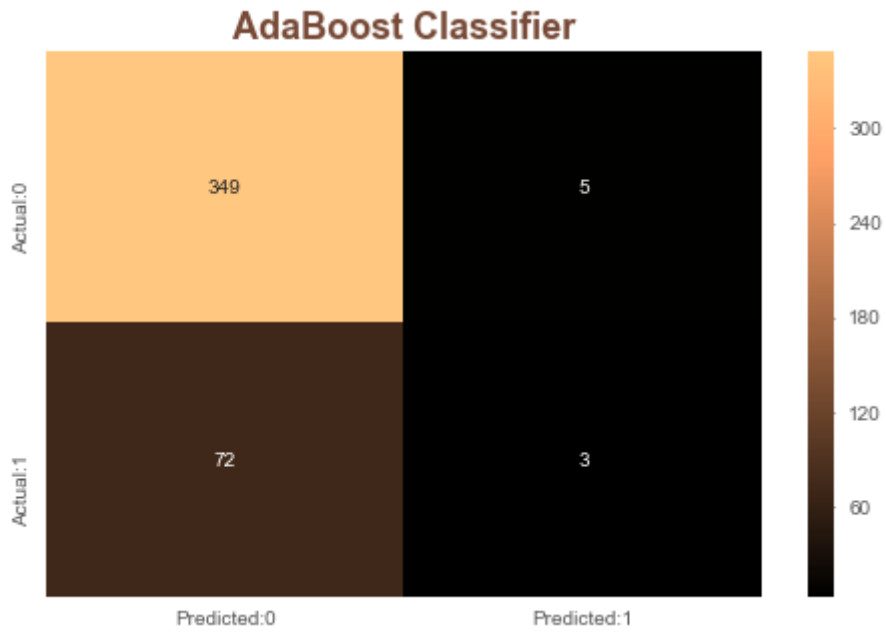
In [182]:
```
print('The accuracy of the AdaBoost model is: ',round(accuracy_score(y_test,ada_pred)*100,2))
```

```
The accuracy of the AdaBoost model is:  82.05
```

In [183]:
```
print('The precision of the GradientBoostingClassifier model is: ',round(precision_score(y_test,gb_pred)*100,2))
print('The recall of the GradientBoostingClassifier model is: ',round(recall_score(y_test,gb_pred)*100,2))
print('The F1_Score of the GradientBoostingClassifier model is: ',round(f1_score(y_test,gb_pred)*100,2))
print('\n\n\t\tGradient Boosting Classifier Classification Report')
print(classification_report(y_test,gb_pred))
```

```
The precision of the GradientBoostingClassifier model is:  26.09
The recall of the GradientBoostingClassifier model is:  16.0
The F1_Score of the GradientBoostingClassifier model is:  19.83


                Gradient Boosting Classifier Classification Report
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       354
           1       0.26      0.16      0.20        75

    accuracy                           0.77       429
   macro avg       0.55      0.53      0.53       429
weighted avg       0.74      0.77      0.75       429
```

In [184]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,ada_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap='copper')
fm={'size':18,'color':'#774936','weight':'bold'}
plt.title('AdaBoost Classifier',**fm)
plt.show()
```



In [185]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print("True Negative", TN)
print("True Posetive", TP)
print("False Negative", FN)
print("False Posetive", FP)
print("Sensitivity", sensitivity)
print("Specificity", specificity)
```

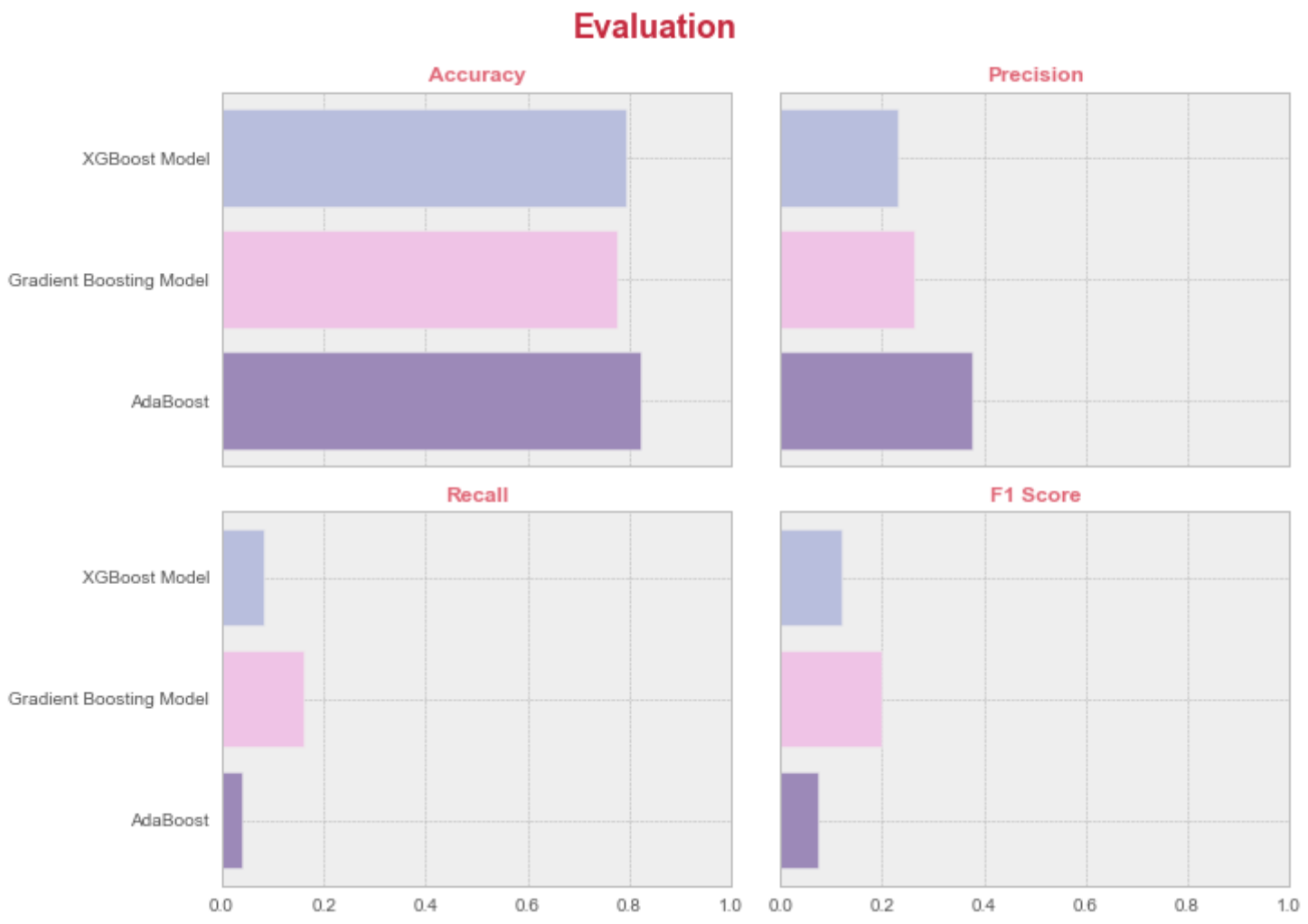
```
True Negative 349
True Posetive 3
False Negative 72
False Posetive 5
Sensitivity 0.04
Specificity 0.9858757062146892
```

Both the models are highly specific. This is due to the number of positive observations are very less.

In [186]:
```
evl_ada=insert_data(y_test,ada_pred,'AdaBoost')
```

In [187]:
```
# Let's try using XGBoost
xm=XGBClassifier()
xm.fit(X_train,y_train)
x_pred=xm.predict(X_test)
evl=append_data(evl_gb,evl_ada)
evl=append_data(insert_data(y_test,x_pred,'XGBoost Model'),evl)
```

In [188]:
```
plot_models(evl)
```

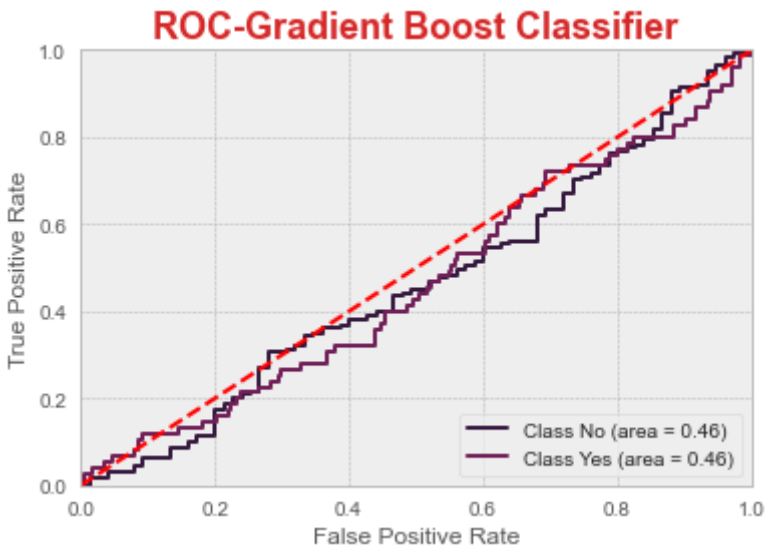


Even though Gradient Boosting model has lowest accuracy it has good precision and f1 score to show that model is better than other two models. The reason for low precision and recall is due highly imbalanced target class.

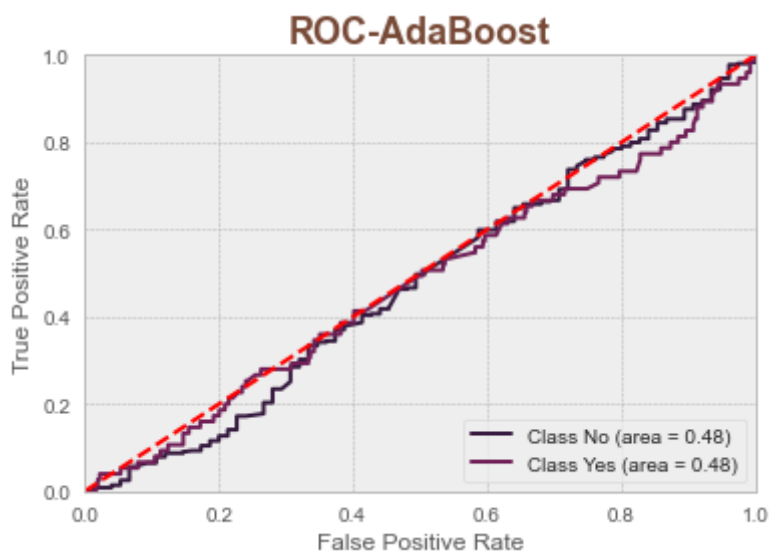
Question-5: Find the ROC-AUC Score of the models and plot on ROC-AUC Curve.

In [189]:
```
from sklearn.metrics import roc_auc_score,roc_curve
```

In [190]:
```
adamodel_prob=gb_clf.predict_proba(X_test).T


dummy_y_test=pd.get_dummies(y_test)

roc_auc=dict()
lfpr4=dict()
ltpr6=dict()
lthresholds4=dict()
for i in dummy_y_test.columns:
    roc_auc[i]=roc_auc_score(dummy_y_test[i],adamodel_prob[i-1])
    lfpr4[i], ltpr6[i], lthresholds4[i] = roc_curve(dummy_y_test[i], adamodel_prob[i-1])


for i in dummy_y_test.columns:
    cls=lb.classes_
    plt.plot(lfpr4[i], ltpr6[i], label='Class '+str(cls[i])+ ' (area = %0.2f)' % roc_auc[i])



plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
fm={'size':18,'color':'#d62828','weight':'bold'}
plt.title('ROC-Gradient Boost Classifier',**fm)
plt.legend(loc="lower right")
plt.show()
```



In [191]:
```
adamodel_prob=ada.predict_proba(X_test).T


dummy_y_test=pd.get_dummies(y_test)

roc_auc=dict()
lfpr4=dict()
ltpr6=dict()
lthresholds4=dict()
for i in dummy_y_test.columns:
    roc_auc[i]=roc_auc_score(dummy_y_test[i],adamodel_prob[i-1])
    lfpr4[i], ltpr6[i], lthresholds4[i] = roc_curve(dummy_y_test[i], adamodel_prob[i-1])


for i in dummy_y_test.columns:
    cls=lb.classes_
    plt.plot(lfpr4[i], ltpr6[i], label='Class '+str(cls[i])+ ' (area = %0.2f)' % roc_auc[i])



plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
fm={'size':18,'color':'#774936','weight':'bold'}
plt.title('ROC-AdaBoost',**fm)
plt.legend(loc="lower right")
plt.show()
```



Unsupervised Learning

Kmeans Clustering

K-means Clustering Algorithm is simple and easy to implement **unsupervised** machine learning algorithm. The main objective of K-means clustering is to group similar data points into a cluster. It is used when we have unlabelled data. This algorithm aims to divide 'n' number of observations into 'k' number of clusters.

A **cluster** refers to a collection of data points clubbed together because of existing similarities. The task of identifying and assigning similar data points to a group is known as **Clustering**.

Hierarchical Clustering

Hierarchical Clustering is an unsupervised learning algorithm that is used to group similar data-points in a cluster. It creates clusters that have a pre-determined order from top to bottom—for example, files and folders organized in a hierarchy on a hard-disk.

There are two types of Hierarchical Clustering:

- **Agglomerative Hierarchical Clustering** :

It is most commonly used. It works in a **bottom-up manner**. Here, we assign each data-point to an individual cluster and then calculate the similarity between the groups using either **Euclidean Distance** or **Manhattan Distance** and club the most similar clusters. It merges the same points of the cluster and stops when all the data-points are combined into a single cluster.

- **Divisive Hierarchical Clustering** :

It is not used much. It is the inverse of Agglomerative Hierarchical Clustering. It works in a top-down manner. Here, we assign all the data points to a single cluster after each iteration. We remove the data points from the cluster, which are not similar, and each data-point that we remove is treated as an individual cluster. Here, we are dividing the cluster in every step. This is why it is known as Divisive Hierarchical Clustering.

Scenario-1: Bigmart

Bigmart is a supermarket mall, with some primary data about customers like customer ID, age, gender, annual income, and spending Score. Spending Score is something they assign to the customer based on their defined parameters like customer behavior and purchasing data.

Customer segmentation is one of the major applications of K-means Clustering. It is the practice of dividing a customer base into groups of individuals having similar features. It is a useful method because Bigmart can approach these different customer segments and distribute marketing resources effectively.

Problem Statement:

Bigmart CEO wants to understand the customers who can converge easily [Target Customers] so that the marketing team can have a sense and plan the strategy accordingly

So, being an Analyst, you must

1. Achieve customer segmentation using the machine learning algorithm (KMeans Clustering) in Python
2. Identify target customers with whom you can start marketing strategy

Tasks to be performed:

To attain the above goal below, tasks must be completed:

Importing and preprocessing the data is priority

- Segregate the columns to be clustered (**Age** and **Spending Score (1-100)**)- Beginner
- Identify the value of K using Elbow curve - Beginner
- Create, fit and predict K-means model with the help of clusters found by Elbow method - Intermediate
- Calculate the centroids of each cluster - Intermediate
- Visualize customer based on their ***Age*** and ***Spending Score (1-100)*** - Advance

Dataset Description:

The Bigmart Dataset contains 200 rows and 5 columns

- **Customer Id**- Identity number of a Customer
- **Gender**- Gender of the Customer
- **Age**- Age of the Customer
- **Annual Income (k$)**- Annual income of the Customer in Dollars
- **Spending Score (1-100)**- Spending score of customer

Topics Covered:

- Elbow Method
- K-means Clustering

Importing and Preprocessing the Data

In [ ]:
```
# importing the required library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
!wget https://www.dropbox.com/s/no4nw204xhep7ov/Bigmart.csv?dl=0
```

```
--2020-07-13 05:42:16--  https://www.dropbox.com/s/no4nw204xhep7ov/Bigmart.csv?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.65.1, 2620:100:6021:1::a27d:4101
Connecting to www.dropbox.com (www.dropbox.com)|162.125.65.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/no4nw204xhep7ov/Bigmart.csv [following]
--2020-07-13 05:42:16--  https://www.dropbox.com/s/raw/no4nw204xhep7ov/Bigmart.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc84194ca987e075f244b237fd11.dl.dropboxusercontent.com/cd/0/inline/A7as238vH4cb5YuuXV7JPvzyTAJv0NGL
WC2bWGNhJkg_7F6Hsb6ymii_-8P_98qLtpZzloXQYNm6c5oEm79cm6GITmL5a67RTypjJ4n3GXhPCi3OKUXGADkQIUcDzRB3PL4/file# [following]
--2020-07-13 05:42:16--  https://uc84194ca987e075f244b237fd11.dl.dropboxusercontent.com/cd/0/inline/A7as238vH4cb5YuuX
V7JPvzyTAJv0NGLWC2bWGNhJkg_7F6Hsb6ymii_-8P_98qLtpZzloXQYNm6c5oEm79cm6GITmL5a67RTypjJ4n3GXhPCi3OKUXGADkQIUcDzRB3PL4/fi
le
Resolving uc84194ca987e075f244b237fd11.dl.dropboxusercontent.com (uc84194ca987e075f244b237fd11.dl.dropboxusercontent.
com)... 162.125.65.15, 2620:100:6021:15::a27d:410f
Connecting to uc84194ca987e075f244b237fd11.dl.dropboxusercontent.com (uc84194ca987e075f244b237fd11.dl.dropboxusercont
ent.com)|162.125.65.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3981 (3.9K) [text/plain]
Saving to: ‘Bigmart.csv?dl=0.1’

Bigmart.csv?dl=0.1  100%[===================>]   3.89K  --.-KB/s    in 0s

2020-07-13 05:42:17 (634 MB/s) - ‘Bigmart.csv?dl=0.1’ saved [3981/3981]
```

In [ ]:
```
customer=pd.read_csv('/content/Bigmart.csv?dl=0')
customer.head()
```

Out[ ]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [ ]:
```
# check the shape of the data
print(f'This dataset has {customer.shape[0]} rows and {customer.shape[1]} columns.')
```

```
This dataset has 200 rows and 5 columns.
```

In [ ]:
```
# check for null data
customer.isnull().sum()
```

Out[ ]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

Dataset does not have any null values

In [ ]:
```
# Dataset info
customer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

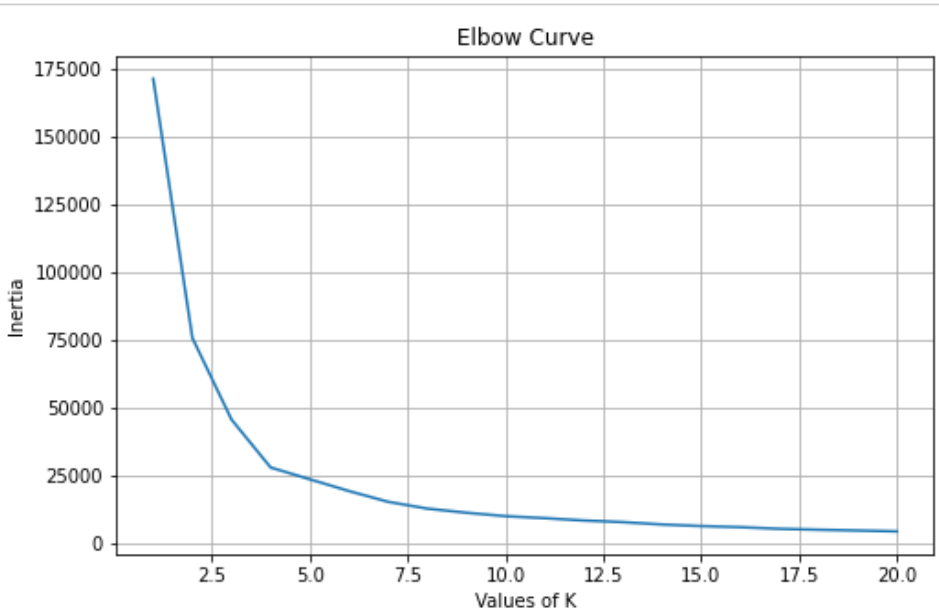
Question-1: Segregate the columns to be clustered (Age and Spending Score (1-100))

In [ ]:
```
#considering the columns we have to cluster
Y= customer.iloc[:, [2,4]].values
```

Question-2: Identify the value of K using Elbow curve

In [ ]:
```
inertia_list = []
for num_clusters in np.arange(1,21):
    kmeans =KMeans(n_clusters=num_clusters)
    kmeans.fit(Y)
    inertia_list.append(kmeans.inertia_)
```

In [ ]:
```
#Plotting the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(np.arange(1, 21), inertia_list)
plt.grid(True)
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('Elbow Curve')
plt.show()
```



From above, we select the optimum value of K by determining the Elbow Point - a point after which the inertia starts decreasing linearly. In this case, we can select the value of K as 4

Question-3: Create, fit and predict K-means model with the help of clusters found by Elbow method

In [ ]:
```
kmeansmodel = KMeans(n_clusters= 4, init='k-means++', random_state=0)
y_kmeans= kmeansmodel.fit_predict(Y)
```

K-means starts with allocating cluster centers randomly and then looks for "better" solutions. K-means++ starts with allocation one cluster center randomly and then searches for other centers given the first one

Hence we use K-means++

Question-4: Calculate the centroids of each clusters

A centroid is a data point (imaginary or real) at the center of a cluster.

In [ ]:
```
#printing the centroids of each cluster
centroids = kmeans.cluster_centers_
centroids
```

Out[ ]:
```
array([[57.23076923, 11.53846154],
       [29.125     , 75.34375   ],
       [47.        , 35.58823529],
       [24.65384615, 55.34615385],
       [40.04761905, 13.04761905],
       [64.85      , 49.85      ],
       [20.6       ,  7.6       ],
       [29.84210526, 37.68421053],
       [48.09090909, 51.22727273],
       [31.52      , 91.32      ]])
```

Question-5: Visulalize customer based on their *Age* and *Spending Score (1-100)*

In [ ]:
```
plt.scatter(Y[y_kmeans == 0, 0], Y[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(Y[y_kmeans == 1, 0], Y[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(Y[y_kmeans == 2, 0], Y[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(Y[y_kmeans == 3, 0], Y[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.title('Clusters of customers')
plt.xlabel('Age)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

We have 4 clusters of customer based on Age and Spending Score (1-100):

- **Cluster-1 (Green)**: Customers of this group have a low score no matter the age. Perhaps the marketing team should develop a different strategy and a different approach to giving these costumers a new perspective which will increase their scores
- **Cluster-2 (Blue)**: Customers of this group are young and adults age < 40 with medium scores between 35 to 75. To increase this group's count, maybe the marketing team can create new actions for young public like games, customized products, and others
- **Cluster-3 (cyan)**: Customers of this group are 40 and above whose scores are medium. To increase this group count, the marketing team can come up with calm places inside the mall, like new restaurants, typical food courts, and others
- **Cluster-4 (Red)**: These customers are with highest spending score. The marketing team should pay close attention to retain these customers

Scenario-2: Caltech Bank

Sydney based Caltech bank plans a new loan scheme for its customers and wants to analyze its customer data to find out how the customer's earning is associated with their credit score.

A credit score is a number ranging from 300-850, which shows the creditworthiness of a customer. That is the number of open accounts, total levels of debt, and repayment history. Higher the credit score, the more trustworthy is the customer.

Problem Statement:

Our objective here is to use clustering method to find the high credit score clusters of customers. It will summarize the existing loan scheme and help Caltech bank to make decision about the new loan scheme

Tasks to be performed:

In order to attain the above goal below, tasks must be performed:

Importing and preprocessing the data is priority

- Segregate the columns to be clustered (**Earning** and **Credit Score**)- Beginner
- Identify the number of clusters using Dendrogram- Intermediate
- Apply a threshold value to the above visualization (dendrogram) to find the clusters- Intermediate
- Create, fit and predict hierarchical model with the help of clusters found by Dendrogram-Advance
- Visualize customer based on their ***Earning*** and ***Credit Score***- Advance

Dataset Description:

The dataset consist of 4 columns and 200 rows,

- **Cust_id**- Id of the Customer
- **Age**- Age of the Customer
- **Earning**- Earning of the Customer
- **Credit Score**- Credit score of the Customer

Topics Covered:

- Dendrograms
- Hierarchical Clustering

Import and preprocess the Data

In [ ]:
```
#importing the libtrary
import numpy as np # Linear Algebra
import pandas as pd # Data Processing
import matplotlib.pyplot as plt # Matplotlib for Plotting
import seaborn as sns
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
!wget https://www.dropbox.com/s/uysjhtgf8mdj1tv/Caltech_Bank-Customers.csv?dl=0
```

```
--2020-06-12 06:41:41--  https://www.dropbox.com/s/uysjhtgf8mdj1tv/Caltech_Bank-Customers.csv?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.3.1, 2620:100:6018:1::a27d:301
Connecting to www.dropbox.com (www.dropbox.com)|162.125.3.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/uysjhtgf8mdj1tv/Caltech_Bank-Customers.csv [following]
--2020-06-12 06:41:41--  https://www.dropbox.com/s/raw/uysjhtgf8mdj1tv/Caltech_Bank-Customers.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucc65f93b6f1976e3dbccbe6cd4f.dl.dropboxusercontent.com/cd/0/inline/A5fhUATSP9DZTs6hBhizHmneMlzmEyZN
6QL-Cw6-xk7lA-BBFUkJseaftFYpef15XKfcZJ1EHY_opTnm_FqqlMWCMghDi8Eq_ReWqSQFYQ86RmEMff1tzTKbmuUfZkz6-us/file# [following]
--2020-06-12 06:41:42--  https://ucc65f93b6f1976e3dbccbe6cd4f.dl.dropboxusercontent.com/cd/0/inline/A5fhUATSP9DZTs6hB
hizHmneMlzmEyZN6QL-Cw6-xk7lA-BBFUkJseaftFYpef15XKfcZJ1EHY_opTnm_FqqlMWCMghDi8Eq_ReWqSQFYQ86RmEMff1tzTKbmuUfZkz6-us/fi
le
Resolving ucc65f93b6f1976e3dbccbe6cd4f.dl.dropboxusercontent.com (ucc65f93b6f1976e3dbccbe6cd4f.dl.dropboxusercontent.
com)... 162.125.3.15, 2620:100:6018:15::a27d:30f
Connecting to ucc65f93b6f1976e3dbccbe6cd4f.dl.dropboxusercontent.com (ucc65f93b6f1976e3dbccbe6cd4f.dl.dropboxusercont
ent.com)|162.125.3.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2726 (2.7K) [text/plain]
Saving to: ‘Caltech_Bank-Customers.csv?dl=0.1’

Caltech_Bank-Custom 100%[===================>]   2.66K  --.-KB/s    in 0s

2020-06-12 06:41:42 (338 MB/s) - ‘Caltech_Bank-Customers.csv?dl=0.1’ saved [2726/2726]
```

In [ ]:
```
loan=pd.read_csv('/content/Caltech_Bank-Customers.csv?dl=0')
loan.head()
```

Out[ ]:

	Cust_id	Age	Earning	Credit Score
0	1	19	15	39
1	2	21	15	81
2	3	20	16	6
3	4	23	16	77
4	5	31	17	40

In [ ]:
```
#check for null values
loan.isnull().sum()
```

Out[ ]:
```
Cust_id         0
Age             0
Earning         0
Credit Score    0
dtype: int64
```

There is no null values in the dataset

In [ ]:
```
# check the shape of the data
print(f'This dataset has {loan.shape[0]} rows and {loan.shape[1]} columns.')
```

```
This dataset has 200 rows and 4 columns.
```

In [ ]:
```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Cust_id       200 non-null    int64
 1   Age           200 non-null    int64
 2   Earning       200 non-null    int64
 3   Credit Score  200 non-null    int64
dtypes: int64(4)
memory usage: 6.4 KB
```

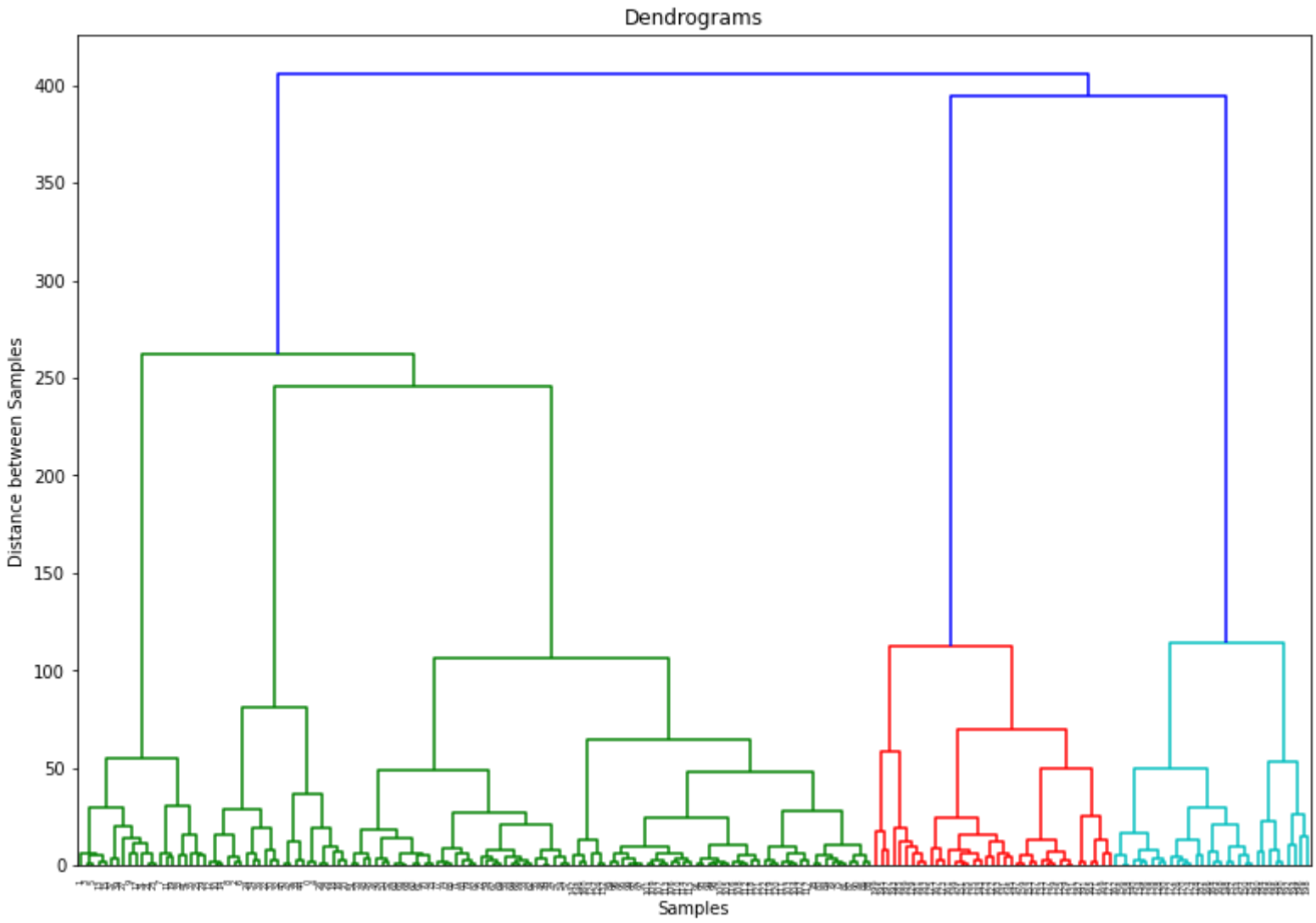
Question-1: Segregate the columns to be clustered (Earning and Credit Score)

In [ ]:
```
#Selecting Earning and Credit Score column as we want to do prediction on those columns and assigning to variable Y
Y=loan.iloc[:,[2,3]].values
```

Question-2: Identify the number of clusters using Dendrogram

In [ ]:
```
plt.figure(figsize=(13, 9))
plt.title("Dendrograms")
dendrogram = shc.dendrogram(shc.linkage(Y, method='ward'))#ward is one of the methods that is used to calculate distan
ce between newly formed clusters
plt.xlabel('Samples')
plt.ylabel('Distance between Samples')
```

Out[ ]: Text(0, 0.5, 'Distance between Samples')

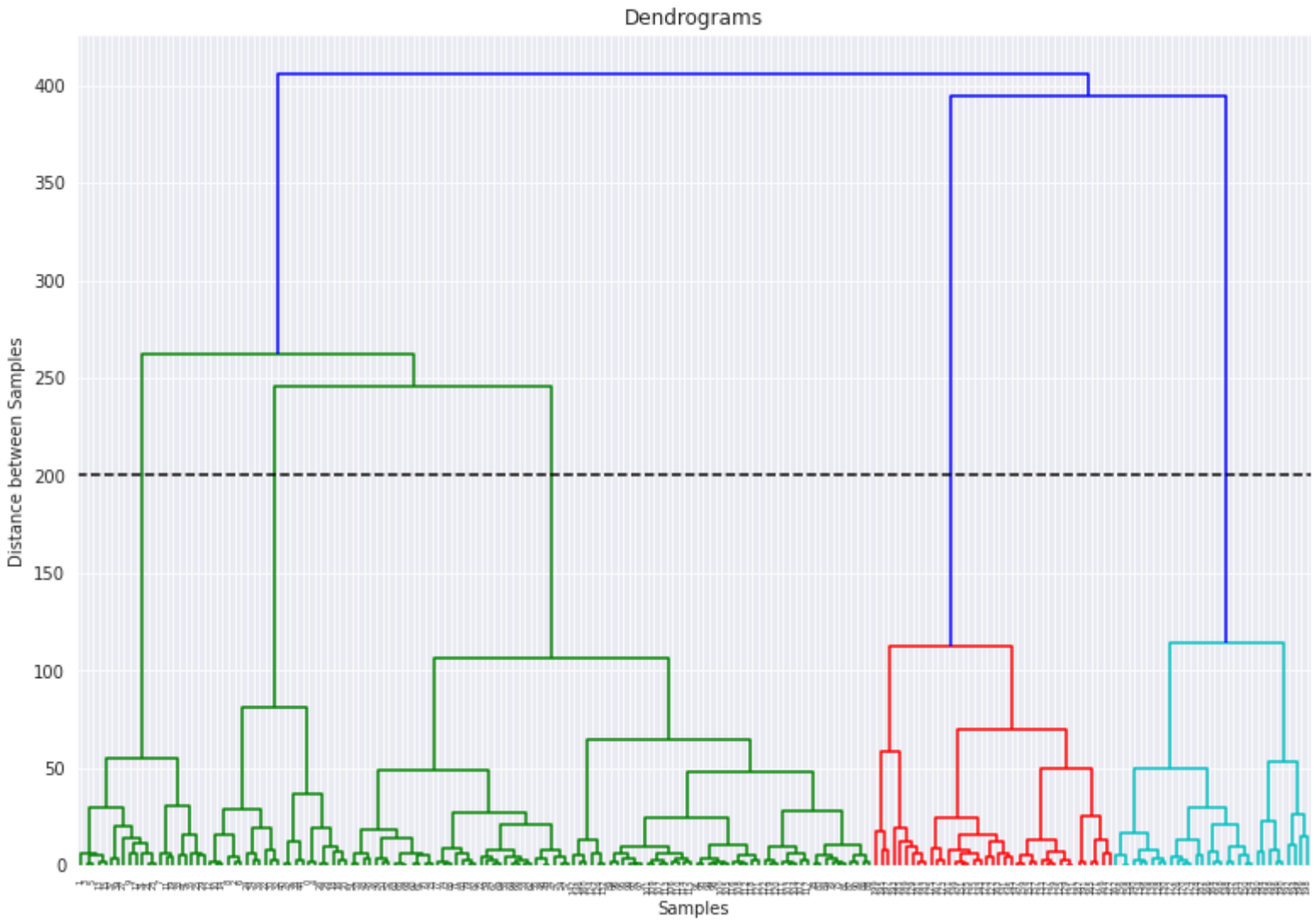


To interpret a Dendrogram one must focus on the height at which any two objects are joined together.

From above, you can see that blue and green line has the maximum distance. We can select a threshold of 200 and the cut the dendrogram.

Question-3: Apply a threshold value to the above visualization (Dendrogram) to find the clusters

In [ ]:
```
plt.figure(figsize=(13, 9))
plt.title("Dendrograms")
dendrogram = shc.dendrogram(shc.linkage(Y, method='ward'))#ward is used to calculate distance between newly formed clu
sters and can only be used with Euclidean Distance
plt.axhline(y=200, color='k', linestyle='--')
plt.xlabel('Samples')
plt.ylabel('Distance between Samples')
plt.show()
```



From above, you can see that the line cuts the Dendrogram at five points. That means we are going to apply hierarchial clustering for five clusters

Question-4:

Create, fit and predict hierarchical model with five clusters found by Dendrogram

In [ ]:
```
h_clustering=AgglomerativeClustering(n_clusters=5,affinity='euclidean',linkage='ward')
h_clustering1=h_clustering.fit_predict(Y)
```

Question-5:

Visualize customer based on their ***Earning*** and ***Credit Score***

In [ ]:
```
#Visualizing the clusters using scatter plots
plt.scatter(Y[h_clustering1==0,0],Y[h_clustering1==0,1],color='blue',label='more earning and low credit score')
plt.scatter(Y[h_clustering1==1,0],Y[h_clustering1==1,1],color='green',label='average earning and average credit score'
)
plt.scatter(Y[h_clustering1==2,0],Y[h_clustering1==2,1],color='red',label='more earning and high credit score')
plt.scatter(Y[h_clustering1==3,0],Y[h_clustering1==3,1],color='orange',label='low earning and high credit score')
plt.scatter(Y[h_clustering1==4,0],Y[h_clustering1==4,1],color='cyan',label='low earning and low credit score')
plt.title('Bank Customers_Earning VS Credit Score')
fig=plt.gcf()
fig.set_size_inches(13,9)
sns.set_style('darkgrid')
plt.xlabel('Earning')
plt.ylabel('Credit Score')
plt.legend()
plt.show()
```

We have 5 clusters of customer based on Earning and Credit Score:

- **Cluster-1 (Blue)**: Customers of this group earn more but maintain a very low credit score.
- **Cluster-2 (Green)**: Customers of this group are earn average and maintain a decent credit score. Bank may consider them for the new loan scheme
- **Cluster-3 (Red)**: Customers of this group earn more and have a high credit score. Bank will surely consider them for the new loan scheme
- **Cluster-4 (Orange)**: These customers earn less but have a high credit score, seems like they repay their loan on time
- **Cluster-5 (Cyan)**: These customers earn less and have a low credit score, they might not be eligible for the new loan scheme

In [ ]:

Dimensionality Reduction

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique in which we extract a new set of variables from the dataset. It is one of the widely used unsupervised algorithms. This technique can also be used for visualization, noise filtering, feature extraction or engineering, and much more.

Linear Discriminant Analysis (LDA) a dimensionality reduction technique which is used to reduce the number of dimensions or features or variables in a dataset while retaining as much information as possible.

LDA is a **supervised** learning algorithm which is used for classification. Here, in LDA, we are interested in **maximizing the separability** between the two groups to make the best decision. LDA is like PCA but focuses on maximizing the separability among known categories.

Scenario-1: U.S Nutrient Database ¶

The dataset, U.S. Nutrient Database, includes the nutrient data from the national survey What We Eat In America, National Health and Nutrition Review Survey (WWEIA, NHANES), for determining dietary intakes. U.S. Nutrient data have historically been used for national nutrition monitoring

Earlier databases were composed mainly of commodity-type items such as wheat flour, sugar, milk, etc. However, with increased consumption of commercial processed and restaurant foods and changes in how national nutrition monitoring data are used, many commercial processed and restaurant items have been added to the database.

Problem Statement:

The increase of commercial commodities and fancy foods may create an imbalance in the nutrition of Americans. Hence the Department of Agriculture decides to maintain balance by finding out the right nutrition from the dataset. As the features are more, they decide to hire an Analyst who can help them by applying some the Machine Learning techniques such as PCA and Kmeans clustering to reduce the components and visually analyze the same

Tasks to be performed:

In order to help out, the Department of Agriculture below tasks should be performed,

- Import and preprocess the data- Beginner
- Fit and transform the data on the PCA model- Beginner
- Calculate the variance ratio and plot the same to find the principal components- Intermediate
- Fit and transform the PCA model for the principal components found- Beginner
- Create a Kmeans model for the PCA components- Intermediate
- Add the components and the clusters found by PCA and K-means to the scaled data- Advance
- Visualize the components (1 and 2) based on its first five k-means cluster- Intermediate
- Find out the rich nutrients in each component- Advance

Dataset Description:

The Dataset consist of 8618 rows and 45 columns, some of the major column descriptions are given below,

- **FoodGroup**- Category the food products belongs to
- **ShortDescrip**- Brief Description about the food group
- **CommonName**- Common name of the food
- **ScientificName**- Scientific name of the food
- **Energy_kcal**- Energy produced by food in calories
- **Protein_g**- Protein intake in grams
- **Fat_g**- Fat intake in grams
- **Carb_g**- Carbohydrates intake in grams
- **Sugar_g**- Sugar intake in grams
- **Fiber_g**- Fiber intake in grams
- **VitA_mcg VitB6_mg VitB12_mcg VitC_mg VitE_mg**- Different vitamins intake in milligrams
- **Calcium_mg**- Calcium intake in milligrmas
- **Iron_mg**- Iron intake in milligrams

Topics Covered:

- PCA
- K-means

In [ ]:
```
# importing the libraries
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

In [ ]:
```
!wget https://www.dropbox.com/s/s1n56r5siezy9zs/USA_Nutrition_Data.csv?dl=0
```

```
--2020-07-13 05:47:50--  https://www.dropbox.com/s/s1n56r5siezy9zs/USA_Nutrition_Data.csv?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:100:6032:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/s1n56r5siezy9zs/USA_Nutrition_Data.csv [following]
--2020-07-13 05:47:50--  https://www.dropbox.com/s/raw/s1n56r5siezy9zs/USA_Nutrition_Data.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucc96641a994b833f1da00d4e1cc.dl.dropboxusercontent.com/cd/0/inline/A7ZXein_wABQlUXmlM5OYQX622CB96fA
skoZqVKIQ_ASP97c5P98adtNcdQsxAf70k6DpOCx2t2zK5q-uX2qA8OuXAWXOuJuA2sL9dJM4_GYGX3BZHAMUm1WPMsHPjcFnfw/file# [following]
--2020-07-13 05:47:50--  https://ucc96641a994b833f1da00d4e1cc.dl.dropboxusercontent.com/cd/0/inline/A7ZXein_wABQlUXml
M5OYQX622CB96fAskoZqVKIQ_ASP97c5P98adtNcdQsxAf70k6DpOCx2t2zK5q-uX2qA8OuXAWXOuJuA2sL9dJM4_GYGX3BZHAMUm1WPMsHPjcFnfw/fi
le
Resolving ucc96641a994b833f1da00d4e1cc.dl.dropboxusercontent.com (ucc96641a994b833f1da00d4e1cc.dl.dropboxusercontent.
com)... 162.125.82.15, 2620:100:6032:15::a27d:520f
Connecting to ucc96641a994b833f1da00d4e1cc.dl.dropboxusercontent.com (ucc96641a994b833f1da00d4e1cc.dl.dropboxusercont
ent.com)|162.125.82.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3065736 (2.9M) [text/plain]
Saving to: ‘USA_Nutrition_Data.csv?dl=0’

USA_Nutrition_Data. 100%[===================>]   2.92M  --.-KB/s    in 0.1s

2020-07-13 05:47:51 (26.4 MB/s) - ‘USA_Nutrition_Data.csv?dl=0’ saved [3065736/3065736]
```

Question-1: Importing and preprocessing the data

In [ ]:
```
nutrients=pd.read_csv('/content/USA_Nutrition_Data.csv?dl=0')
nutrients.head()
```

Out[ ]:

	ID	FoodGroup	ShortDescrip	Descrip	CommonName	MfgName	ScientificName	Energy_kcal	Protein_g	Fat_g	Carb_g	Sugar
0	1001	Dairy and Egg Products	BUTTER,WITH SALT	Butter, salted	NaN	NaN	NaN	717	0.85	81.11	0.06	0
1	1002	Dairy and Egg Products	BUTTER,WHIPPED,WITH SALT	Butter, whipped, with salt	NaN	NaN	NaN	717	0.85	81.11	0.06	0
2	1003	Dairy and Egg Products	BUTTER OIL,ANHYDROUS	Butter oil, anhydrous	NaN	NaN	NaN	876	0.28	99.48	0.00	0
3	1004	Dairy and Egg Products	CHEESE,BLUE	Cheese, blue	NaN	NaN	NaN	353	21.40	28.74	2.34	0
4	1005	Dairy and Egg Products	CHEESE,BRICK	Cheese, brick	NaN	NaN	NaN	371	23.24	29.68	2.79	0

In [ ]:
```
nutrients.shape
```

Out[ ]: (8618, 45)

In [ ]:
```
nutrients.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8618 entries, 0 to 8617
Data columns (total 45 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                8618 non-null   int64  
 1   FoodGroup         8618 non-null   object 
 2   ShortDescrip      8618 non-null   object 
 3   Descrip           8618 non-null   object 
 4   CommonName        1063 non-null   object 
 5   MfgName           1560 non-null   object 
 6   ScientificName    732 non-null    object 
 7   Energy_kcal       8618 non-null   int64  
 8   Protein_g         8618 non-null   float64
 9   Fat_g             8618 non-null   float64
 10  Carb_g            8618 non-null   float64
 11  Sugar_g           8618 non-null   float64
 12  Fiber_g           8618 non-null   float64
 13  VitA_mcg          8618 non-null   int64  
 14  VitB6_mg          8618 non-null   float64
 15  VitB12_mcg        8618 non-null   float64
 16  VitC_mg           8618 non-null   float64
 17  VitE_mg           8618 non-null   float64
 18  Folate_mcg        8618 non-null   int64  
 19  Niacin_mg         8618 non-null   float64
 20  Riboflavin_mg     8618 non-null   float64
 21  Thiamin_mg        8618 non-null   float64
 22  Calcium_mg        8618 non-null   int64  
 23  Copper_mcg        8618 non-null   float64
 24  Iron_mg           8618 non-null   float64
 25  Magnesium_mg      8618 non-null   int64  
 26  Manganese_mg      8618 non-null   float64
 27  Phosphorus_mg     8618 non-null   int64  
 28  Selenium_mcg      8618 non-null   float64
 29  Zinc_mg           8618 non-null   float64
 30  VitA_USRDA        8618 non-null   float64
 31  VitB6_USRDA       8618 non-null   float64
 32  VitB12_USRDA      8618 non-null   float64
 33  VitC_USRDA        8618 non-null   float64
 34  VitE_USRDA        8618 non-null   float64
 35  Folate_USRDA      8618 non-null   float64
 36  Niacin_USRDA      8618 non-null   float64
 37  Riboflavin_USRDA  8618 non-null   float64
 38  Thiamin_USRDA     8618 non-null   float64
 39  Calcium_USRDA     8618 non-null   float64
 40  Copper_USRDA      8618 non-null   float64
 41  Magnesium_USRDA   8618 non-null   float64
 42  Phosphorus_USRDA  8618 non-null   float64
 43  Selenium_USRDA    8618 non-null   float64
 44  Zinc_USRDA        8618 non-null   float64
dtypes: float64(32), int64(7), object(6)
memory usage: 3.0+ MB
```

In [ ]:
```
used = []
corrs = []
# The enumerate() function assigns an index to each item in an iterable object that can be used to reference the item
 later
for i, j in enumerate(nutrients.corr().columns):
    for k in range(len(nutrients.corr())):
        if ((nutrients.corr().iloc[k, i] > 0.9) &
            (j not in used) &
            (j != nutrients.corr().index[k])):

            used.append(j)
            corrs.append((j, nutrients.corr().index[k],
                          np.round(nutrients.corr().iloc[k, i], 2)))


corrs_nutrients = pd.DataFrame([[i[0] for i in corrs],
                        [i[1] for i in corrs],
                        [i[2] for i in corrs]])

corrs_nutrients = corrs_nutrients.T.rename(columns = {0:'column',1:'row',2:'corr'})
corrs_nutrients[:15]
```

Out[ ]:

	column	row	corr
0	VitA_mcg	VitA_USRDA	1
1	VitB6_mg	VitB6_USRDA	1
2	VitB12_mcg	VitB12_USRDA	1
3	VitC_mg	VitC_USRDA	1
4	VitE_mg	VitE_USRDA	1
5	Folate_mcg	Folate_USRDA	1
6	Niacin_mg	Niacin_USRDA	1
7	Riboflavin_mg	Riboflavin_USRDA	1
8	Thiamin_mg	Thiamin_USRDA	1
9	Calcium_mg	Calcium_USRDA	1
10	Copper_mcg	Copper_USRDA	1
11	Magnesium_mg	Magnesium_USRDA	1
12	Phosphorus_mg	Phosphorus_USRDA	1
13	Selenium_mcg	Selenium_USRDA	1
14	Zinc_mg	Zinc_USRDA	1

We can see that the "_USRDA" features are redundant. They should be removed

In [ ]:
```
# dropping _USRDA features
nutrients.drop(nutrients.columns[nutrients.columns.str.contains('_USRDA')].values,
        inplace=True, axis=1)
```

Dropping first six categorical features as PCA and k-means clustering does not affect them much

In [ ]:
```
# dropping the categorical features as well
nutrients.set_index('ID', inplace=True)
nutrients_desc = nutrients.iloc[:, :6]
nutrients.drop(nutrients.columns[:6].values, axis=1, inplace=True)
```

We need to perform Scaling before applying PCA since PCA creates the Principal Component 1 in the direction of the maximum variance. But, if we do not scale, some features in our dataset might show high variance because of their larger values.

This is why it is strongly advisable to scale the data before applying the PCA technique.

StandardScaler() will normalize each column of the dataset INDIVIDUALLY so that each column or feature or variable will have mean = 0 and standard deviation = 1.

In [ ]:
```
# standardizing the data
nutrients_TF = StandardScaler().fit_transform(nutrients)
```

In [ ]:
```
# printing the shape of the data
nutrients_TF.shape
```

Out[ ]: (8618, 23)

Question-2: Fit and transform the data on the PCA model

In [ ]:
```
# from sklearn.decomposition import PCA
pca = PCA()
pca.fit_transform(nutrients_TF)
```

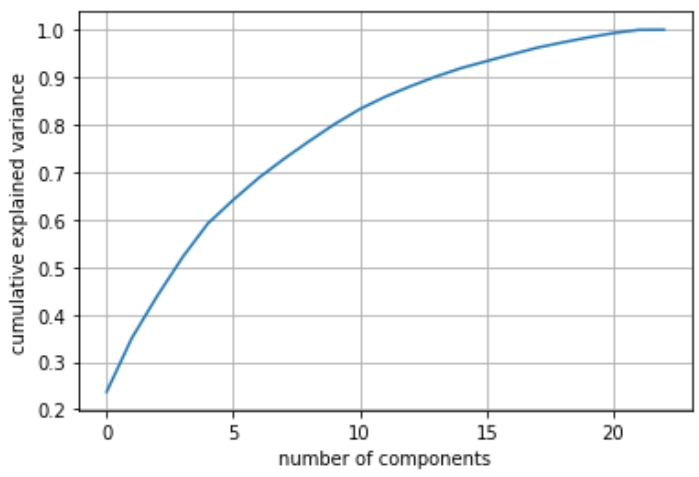
Out[ ]:
```
array([[-1.12177585e+00, -1.18225141e+00, -3.66193973e+00, ...,
         2.08470136e-01, -3.98100905e-02,  3.70138174e-02],
       [-1.11468691e+00, -1.18417302e+00, -3.66232928e+00, ...,
         2.22450074e-01, -3.76157809e-02,  3.68193054e-02],
       [-9.94919411e-01, -1.57357953e+00, -4.69772411e+00, ...,
         2.38667900e-01, -7.22173197e-02,  4.31958958e-02],
       ...,
       [-7.67670698e-01, -3.26765632e+00,  9.85205562e-01, ...,
         3.64559090e-01, -1.48468056e-01,  8.65889021e-02],
       [ 3.55897094e-01,  6.78435359e-01, -1.00293556e+00, ...,
        -8.24537105e-01,  1.12523136e-01, -4.86965696e-02],
       [-8.66889802e-01,  1.19845904e+00,  1.93486895e-01, ...,
        -1.53381540e-01,  2.30499506e-01, -1.91554534e-03]])
```

Question-3: Calculate the variance ratio and plot the same to find the principal components

In [ ]:
```
pca.explained_variance_ratio_
```

Out[ ]:
```
array([2.36925468e-01, 1.13846015e-01, 8.83433734e-02, 8.17013669e-02,
       7.11161480e-02, 4.95813332e-02, 4.61246638e-02, 4.02719041e-02,
       3.74790508e-02, 3.58559322e-02, 3.17927198e-02, 2.59519549e-02,
       2.21153503e-02, 2.04113766e-02, 1.77385355e-02, 1.46927827e-02,
       1.43290735e-02, 1.39668837e-02, 1.11392349e-02, 1.03538827e-02,
       9.18082227e-03, 6.91711649e-03, 1.65010749e-04])
```

In [ ]:
```
pca = PCA().fit(nutrients_TF)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.grid(True)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



From above, we can observe that the explained variance ratio after 10 is negligible

Question-4: Fit and transform the PCA model for the principal components found

In [ ]:
```
pca=PCA(n_components=10)
pca.fit(nutrients_TF)
```

Out[ ]:
```
PCA(copy=True, iterated_power='auto', n_components=10, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [ ]:
```
scores_pca=pca.transform(nutrients_TF)
```

In [ ]:
```
print(pca.explained_variance_ratio_[:10].sum())
```

```
0.8012449032271972
```

We can observe that first 10 eigenvectors account for almost 80% of the variance

Question-5: Create a K-means model for the PCA components

In [ ]:
```
#Set a 5 KMeans clustering
from sklearn.cluster import KMeans
kmeans_pca=KMeans(n_clusters=10, init='k-means++', random_state=42)
kmeans_pca.fit(nutrients_TF)
```

Out[ ]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=10, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
```

K-means starts with allocating cluster centers randomly and then looks for "better" solutions. K-means++ starts with allocation one cluster center randomly and then searches for other centers given the first one

Hence we use K-means++

Question-6: Add the components and the clusters found by PCA and K-means to the scaled data

In [ ]:
```
nutrients_seg_pca_kmeans=pd.concat([nutrients.reset_index(drop=True),pd.DataFrame(scores_pca)],axis=1)
nutrients_seg_pca_kmeans.columns.values[-10: ]=['Component1','Component2','Component3','Component4','Component5','Component6','Component7','Component8','Component9','Component10']
nutrients_seg_pca_kmeans['segment_kmeans_pca']=kmeans_pca.labels_
nutrients_seg_pca_kmeans.head()
```

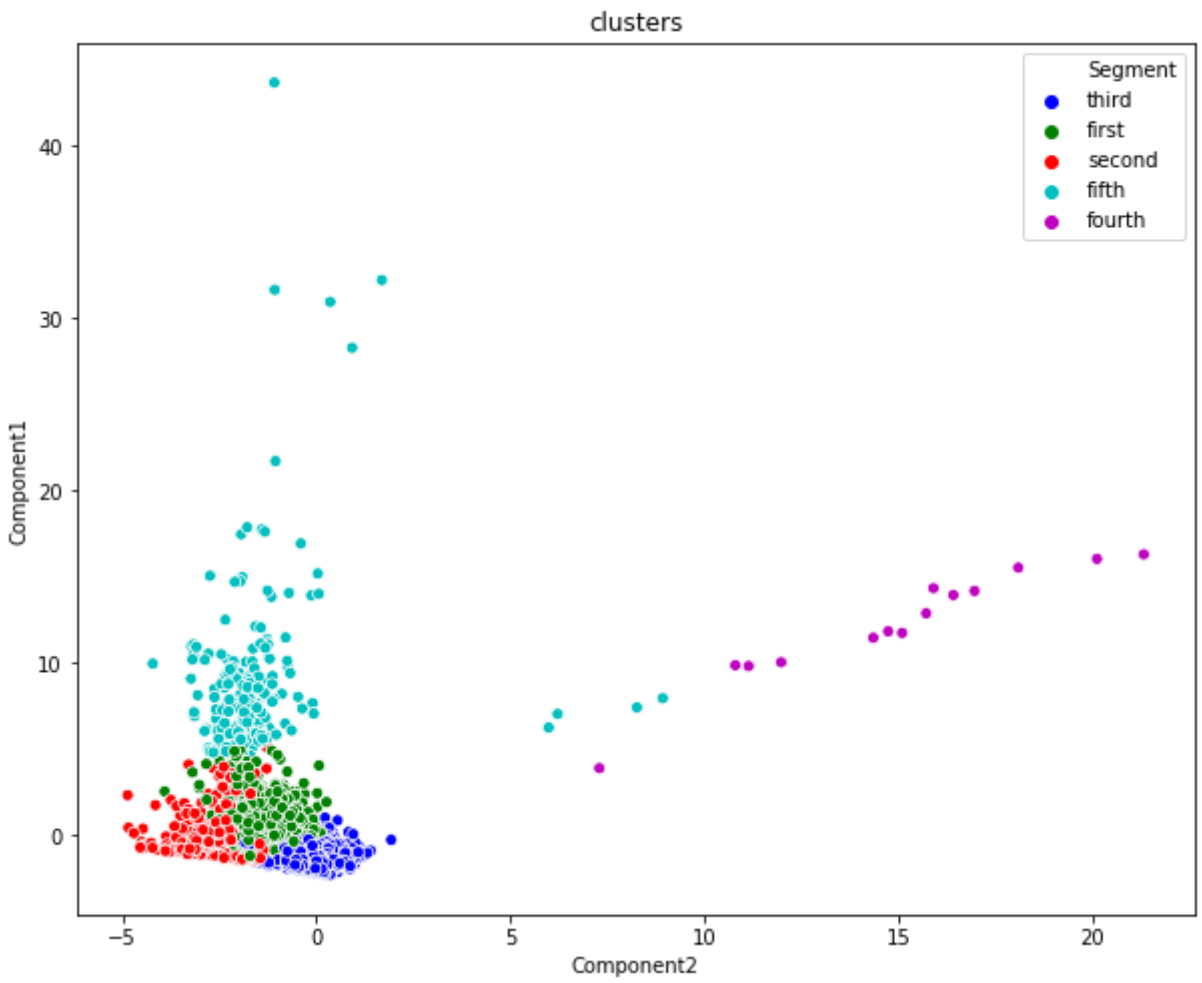
Out[ ]:

	Energy_kcal	Protein_g	Fat_g	Carb_g	Sugar_g	Fiber_g	VitA_mcg	VitB6_mg	VitB12_mcg	VitC_mg	VitE_mg	Folate_mcg	Niacin_mg	Ribof
0	717	0.85	81.11	0.06	0.06	0.0	684	0.003	0.17	0.0	2.32	3	0.042	
1	717	0.85	81.11	0.06	0.06	0.0	684	0.003	0.13	0.0	2.32	3	0.042	
2	876	0.28	99.48	0.00	0.00	0.0	840	0.001	0.01	0.0	2.80	0	0.003	
3	353	21.40	28.74	2.34	0.50	0.0	198	0.166	1.22	0.0	0.25	36	1.016	
4	371	23.24	29.68	2.79	0.51	0.0	292	0.065	1.26	0.0	0.26	20	0.118	

Question-7: Visualize the components (1 and 2) based on its first five k-means cluster

In [ ]:
```
nutrients_seg_pca_kmeans['Segment']=nutrients_seg_pca_kmeans['segment_kmeans_pca'].map({0:'first',
                                                                                        1:'second',
                                                                                        2:'third',
                                                                                        3:'fourth',
                                                                                        4:'fifth'})
                                                                                        #5:'sixth',
                                                                                        #6:'seventh',
                                                                                        #7:'eighth',
                                                                                        #8:'ninth'})
```

In [ ]:
```
import seaborn as sns
x_axis= nutrients_seg_pca_kmeans['Component2']
y_axis= nutrients_seg_pca_kmeans['Component1']
plt.figure(figsize=(10,8))
sns.scatterplot(x_axis,y_axis,hue=nutrients_seg_pca_kmeans['Segment'],palette=['b','g','r','c','m'])
plt.title('clusters')
plt.show()
```



As we can see we have five different clusters based on components 1 and 2.

But we can further analyze for rest of the components by sorting them to find what nutrients each component is rich in.

Question-8: Find out the the rich nutrients in each component

In [ ]:
```
vects = pca.components_[:10]

first = pd.Series(vects[0], index=nutrients.columns)
first.sort_values(ascending=False)
```

Out[ ]:
```
Riboflavin_mg    0.341325
Niacin_mg        0.337779
VitB6_mg         0.315663
Iron_mg          0.299857
Folate_mcg       0.284102
Thiamin_mg       0.272453
Zinc_mg          0.243551
Magnesium_mg     0.241348
Phosphorus_mg    0.199403
Fiber_g          0.181570
Copper_mcg       0.180806
VitB12_mcg       0.177985
Carb_g           0.169685
Calcium_mg       0.168112
Energy_kcal      0.157814
Protein_g        0.140620
VitE_mg          0.137122
VitA_mcg         0.133519
Manganese_mg     0.093567
Selenium_mcg     0.092319
VitC_mg          0.087639
Sugar_g          0.076323
Fat_g            0.033008
dtype: float64
```

First component is rich in Riboflavin_mg, Niacin_mg and VitB6

In [ ]:
```
second = pd.Series(vects[1], index=nutrients.columns)
second.sort_values(ascending=False)
```

Out[ ]:
```
VitB12_mcg       0.355044
Protein_g        0.343396
Selenium_mcg     0.239322
VitA_mcg         0.236470
Copper_mcg       0.212669
Zinc_mg          0.177798
Manganese_mg     0.088783
Phosphorus_mg    0.087448
Niacin_mg        0.084800
Riboflavin_mg    0.073473
VitB6_mg         0.021129
VitC_mg         -0.038525
Thiamin_mg      -0.075151
Iron_mg         -0.093812
Folate_mcg      -0.097093
Magnesium_mg    -0.103362
Calcium_mg      -0.105173
VitE_mg         -0.106372
Fat_g           -0.111670
Fiber_g         -0.257733
Energy_kcal     -0.273449
Sugar_g         -0.358769
Carb_g          -0.443417
dtype: float64
```

Second component is rich in VitaminB12 and Protein

In [ ]:
```
third = pd.Series(vects[2], index=nutrients.columns)
third.sort_values(ascending=False)
```

Out[ ]:
```
Folate_mcg       0.230985
Riboflavin_mg    0.192097
Thiamin_mg       0.184352
VitB6_mg         0.174648
Niacin_mg        0.164885
VitC_mg          0.162303
Iron_mg          0.087109
Sugar_g          0.055247
Carb_g           0.049822
VitB12_mcg      -0.012759
VitA_mcg        -0.021929
Zinc_mg         -0.038639
Fiber_g         -0.040398
Manganese_mg    -0.072630
Calcium_mg      -0.128138
Copper_mcg      -0.152263
Selenium_mcg    -0.163361
Magnesium_mg    -0.201225
VitE_mg         -0.207331
Protein_g       -0.213567
Phosphorus_mg   -0.274815
Energy_kcal     -0.462006
Fat_g           -0.534051
dtype: float64
```

Third component is rich in Folate, Riboflavin and Thiamin

In [ ]:
```
fourth = pd.Series(vects[3], index=nutrients.columns)
fourth.sort_values(ascending=False)
```

Out[ ]:
```
VitA_mcg         0.530395
Copper_mcg       0.389930
VitB12_mcg       0.346546
Manganese_mg     0.311369
Sugar_g          0.217375
Carb_g           0.174106
Energy_kcal      0.052279
VitC_mg          0.047583
Riboflavin_mg    0.047000
Fiber_g          0.042132
Fat_g            0.026522
VitE_mg          0.026173
Folate_mcg      -0.032482
Iron_mg         -0.059442
Magnesium_mg    -0.071807
Calcium_mg      -0.099343
Thiamin_mg      -0.103522
VitB6_mg        -0.114375
Niacin_mg       -0.156390
Selenium_mcg    -0.161623
Zinc_mg         -0.166321
Phosphorus_mg   -0.207867
Protein_g       -0.311115
dtype: float64
```

Fourth component is rich in Vitamin-A, copper and Vitamin-B12

In [ ]:
```
fifth = pd.Series(vects[4], index=nutrients.columns)
fifth.sort_values(ascending=False)
```

Out[ ]:
```
Calcium_mg       0.388687
Magnesium_mg     0.352739
Phosphorus_mg    0.344935
Fiber_g          0.332164
Copper_mcg       0.161243
Manganese_mg     0.125602
Iron_mg          0.097111
Carb_g           0.082720
VitC_mg          0.024712
Protein_g        0.013172
VitA_mcg         0.008137
Selenium_mcg    -0.005049
Sugar_g         -0.048419
VitB12_mcg      -0.058864
Zinc_mg         -0.064317
VitB6_mg        -0.133750
Folate_mcg      -0.137698
Riboflavin_mg   -0.153451
Thiamin_mg      -0.161526
Niacin_mg       -0.203431
VitE_mg         -0.238019
Energy_kcal     -0.293598
Fat_g           -0.394449
dtype: float64
```

Fifth Component is rich in Calcium, Magnesium and Phosphorus

In [ ]:
```
sixth = pd.Series(vects[5], index=nutrients.columns)
sixth.sort_values(ascending=False)
```

Out[ ]:
```
VitC_mg          0.545230
VitE_mg          0.475303
VitB6_mg         0.200559
Manganese_mg     0.182514
Magnesium_mg     0.140152
Fiber_g          0.134845
Fat_g            0.116474
Calcium_mg       0.070576
VitA_mcg         0.039029
Niacin_mg        0.026240
Iron_mg         -0.030552
Riboflavin_mg   -0.050837
Zinc_mg         -0.054555
Folate_mcg      -0.054681
Phosphorus_mg   -0.083852
Copper_mcg      -0.086236
VitB12_mcg      -0.123084
Thiamin_mg      -0.129940
Energy_kcal     -0.132847
Protein_g       -0.141427
Selenium_mcg    -0.243776
Carb_g          -0.279338
Sugar_g         -0.337044
dtype: float64
```

Sixth component is rich in VitC_mg, VitE_mg, VitB6_mg

In [ ]:
```
seventh = pd.Series(vects[6], index=nutrients.columns)
seventh.sort_values(ascending=False)
```

Out[ ]:
```
Calcium_mg       0.462919
VitC_mg          0.451307
Phosphorus_mg    0.323384
Sugar_g          0.298561
Riboflavin_mg    0.141626
VitA_mcg         0.101345
Energy_kcal      0.066867
VitB12_mcg       0.056224
Niacin_mg        0.045594
Fat_g            0.045275
VitB6_mg         0.024780
Carb_g           0.022154
Protein_g        0.006402
Selenium_mcg    -0.021900
Thiamin_mg      -0.071704
Zinc_mg         -0.105966
Copper_mcg      -0.119264
VitE_mg         -0.125430
Folate_mcg      -0.130927
Iron_mg         -0.133354
Manganese_mg    -0.136536
Magnesium_mg    -0.269531
Fiber_g         -0.414795
dtype: float64
```

Seventh component is rich in Calcium_mg, VitC_mg and Phosphorus_mg

In [ ]:
```
eighth = pd.Series(vects[7], index=nutrients.columns)
eighth.sort_values(ascending=False)
```

Out[ ]:
```
VitC_mg          0.515879
Selenium_mcg     0.410381
Magnesium_mg     0.191403
Copper_mcg       0.188064
Fiber_g          0.187961
Carb_g           0.154911
Sugar_g          0.143632
Protein_g        0.116072
VitB6_mg         0.083053
Energy_kcal      0.074304
Zinc_mg          0.034069
Niacin_mg        0.010761
VitB12_mg        0.004152
Riboflavin_mg   -0.044293
VitA_mcg        -0.046241
Fat_g           -0.049490
Iron_mg         -0.107517
VitE_mg         -0.148575
Folate_mcg      -0.162232
Phosphorus_mg   -0.169948
Thiamin_mg      -0.190809
Manganese_mg    -0.345888
Calcium_mg      -0.367258
dtype: float64
```

Eighth component is rich in VitC_mg, Selenium_mcg and Magnesium_mg

In [ ]:
```
ninth = pd.Series(vects[8], index=nutrients.columns)
ninth.sort_values(ascending=False)
```

Out[ ]:
```
Manganese_mg     0.757073
Selenium_mcg     0.358361
Sugar_g          0.151174
Protein_g        0.148392
Niacin_mg        0.121018
Carb_g           0.117768
VitC_mg          0.085523
Energy_kcal      0.041513
Magnesium_mg     0.031464
Thiamin_mg       0.026593
VitB6_mg         0.022902
Phosphorus_mg   -0.012463
VitA_mcg        -0.021988
Riboflavin_mg   -0.044818
VitE_mg         -0.061850
Folate_mcg      -0.064159
Fiber_g         -0.065706
Fat_g           -0.091573
Iron_mg         -0.108780
Zinc_mg         -0.108849
VitB12_mcg      -0.132239
Calcium_mg      -0.135987
Copper_mcg      -0.361687
dtype: float64
```

Ninth component is rich in Manganese_mg, Selenium_mcg and Sugar_g

In [ ]:
```
tenth = pd.Series(vects[9], index=nutrients.columns)
tenth.sort_values(ascending=False)
```

Out[ ]:
```
Thiamin_mg       0.437952
Fat_g            0.200828
Riboflavin_mg    0.195897
VitC_mg          0.155441
Energy_kcal      0.140967
VitA_mcg         0.118084
Magnesium_mg     0.110490
Niacin_mg        0.088437
Fiber_g          0.086771
Phosphorus_mg    0.062097
Copper_mcg       0.047236
Protein_g        0.018057
Folate_mcg       0.017480
Manganese_mg    -0.018068
Carb_g          -0.044196
Selenium_mcg    -0.075541
Calcium_mg      -0.080641
Iron_mg         -0.083104
VitB12_mcg      -0.133779
VitB6_mg        -0.198338
Sugar_g         -0.387868
VitE_mg         -0.424275
Zinc_mg         -0.477655
dtype: float64
```

Tenth component is rich in Thiamin_mg, Fat_g and Riboflavin_mg

Inference:

Now instead of going through all the features in the dataset. Nutritionist from the Department of Agriculture can go through these five components and create a new nutrition plan for Americans

Scenario-2: Velcro Winery

Velcro Winery is one of the finest Wineries in Italy. They prepare three different types of wine, which are represented using 178 samples.

They have performed 13 different chemical analyses on each of their sample and have recorded them in the form of a dataset.

Problem Statement:

The CEO of the winery is interested in classifying the wines into its respective classes based on its chemical analyses. Hence, he decides to hire you as an Analyst who can apply the Dimensionality reduction technique (LDA) to classify the wines and also build a model to check its accuracy.

Tasks to be performed:

- Importing and preprocessing the data is a priority- Beginner
- Build an LDA model using sklearn- Beginner
- Visualize the LDA components with the help of model- Intermediate
- Split the data into Training and Testing. (keep the test as 30% and the random state as 2)- Beginner
- Build a Random forest classifier model (keep random state as 2)- Intermediate
- Calculate the accuracy of the model- Beginner
- Create a confusion matrix to verify the prediction-Intermediate
- Visualize the confusion matrix using heatmap- Advance
- Create a classification report using sklearn- Advance

Dataset Description

The dataset contains 178 rows and 14 columns which is described as,

- **alcohol**- Alcohol content in wine
- **malic_acid**- Amount of malic acid present in wine
- **ash**- Amount of ash present in wine
- **alcalinity_of_ash**- Amount of alcalinity of ash present in wine
- **magnesium**- Amount of magnesium present in wine
- **total_phenols**- Amount of total phenols present in wine
- **flavanoids**- Amount of total flavored phenols present in wine
- **nonflavanoid_phenols**- Amount of total non-flavored phenols present in wine
- **proanthocyanins**- Amount of proanthocyanins present in wine
- **color_intensity**- color intensity of the wine
- **hue**- pH range of wine which inturn decides the acidity of wine
- **od280/od315_of_diluted_wines**- Diluted wines.
- **proline**- Amino acid present in grapes or wines
- **class**- class to which wines belong

Topic Covered

LDA

Question-1: Importing and preprocessing the data

In [ ]:
```
# importing the libraries
from sklearn.datasets import load_wine
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [ ]:
```
w = load_wine()
x = pd.DataFrame(w.data, columns=w.feature_names)
y = pd.Categorical.from_codes(w.target, w.target_names)
x.head()
```

Out[ ]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04

In [ ]:
```
#joining both train and the target variable
w1 = x.join(pd.Series(y, name='class'))
w1.head()
```

Out[ ]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04

Question-2: Build an LDA model using sklearn

In [ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=3)
X_lda = lda.fit_transform(x, y)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/discriminant_analysis.py:463: ChangedBehaviorWarning: n_components can
not be larger than min(n_features, n_classes - 1). Using min(n_features, n_classes - 1) = min(13, 3 - 1) = 2 componen
ts.
  ChangedBehaviorWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/discriminant_analysis.py:469: FutureWarning: In version 0.23, setting
n_components > min(n_features, n_classes - 1) will raise a ValueError. You should set n_components to None (default),
or a value smaller or equal to min(n_features, n_classes - 1).
  warnings.warn(future_msg, FutureWarning)
```

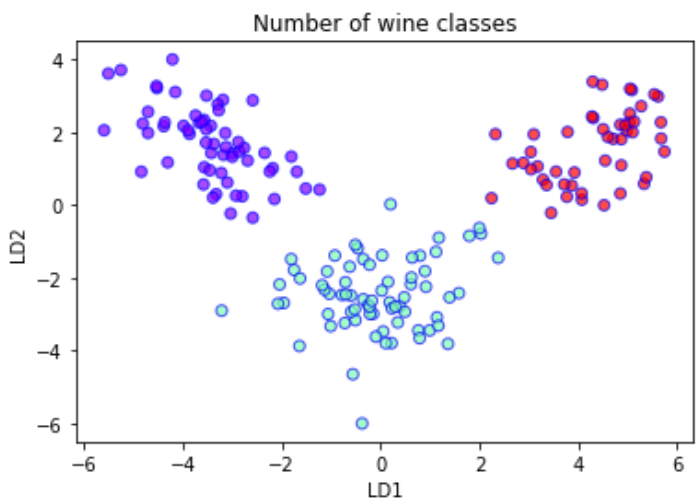
Question-3: Visualize the LDA components with the help of model

As matplotlib cannot handle categorical values, you must convert it into numerical values

This can be done using label encoder

In [ ]:
```
l_encoder = LabelEncoder()
y = l_encoder.fit_transform(w1['class']) #transforming the categorical value to numerical value
```

In [ ]:
```
#classifying the classes using LDA
plt.scatter(X_lda[:,0],X_lda[:,1],c=y,cmap='rainbow',alpha=0.7,edgecolors='b')
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.title('Number of wine classes')
plt.show()
```



Question-4: Split the data into Training and Testing. (keep the test as 30% and random state as 2)

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(w1.drop(['class'],axis = 'columns'),w.target,test_size = 0.3,rando
m_state=2)
```

Question-5: Build a Randomforest classifier model (keep random state as 2)

In [ ]:
```
model = RandomForestClassifier(random_state=0)
model.fit(X_train,y_train)
```

Out[ ]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, 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=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [ ]:
```
y_predicted = model.predict(X_test)
```

Question-6: Calculate the accuracy of the model

In [ ]:
```
print('Accuracy of the model is: ' + str(accuracy_score(y_test, y_predicted)))
```

```
Accuracy of the model is: 0.9629629629629629
```

Question-7: Create a confusion matrix to verify the prediction

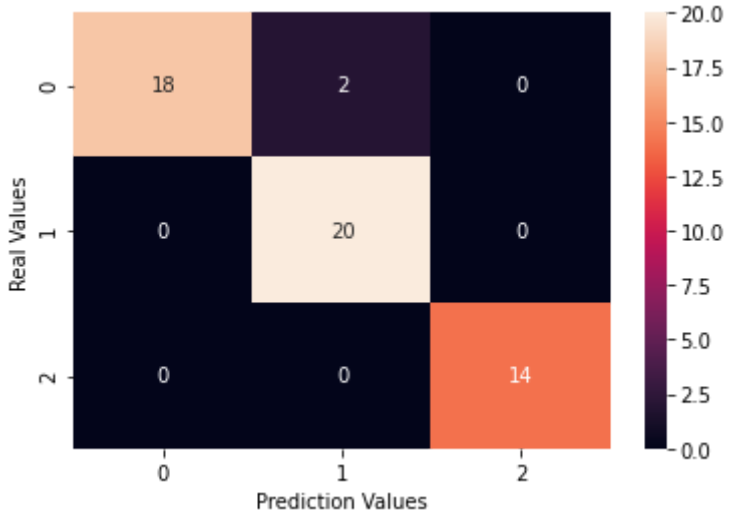
In [ ]:
```
c_matrix = confusion_matrix(y_test,y_predicted)
c_matrix
```

Out[ ]:
```
array([[18,  2,  0],
       [ 0, 20,  0],
       [ 0,  0, 14]])
```

Question-8: Visualize the confusion matrix using heatmap

In [ ]:
```
sns.heatmap(c_matrix,annot = True)
plt.xlabel('Prediction Values')
plt.ylabel('Real Values')
```

Out[ ]:
```
Text(33.0, 0.5, 'Real Values')
```



We can infer that class 1 and class 2 are prefectly predicted where in class 3 predicted one wrong value

Question-9: Create a classification report using sklearn

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predicted, digits=3))
```

```
              precision    recall  f1-score   support

           0      1.000     0.900     0.947        20
           1      0.909     1.000     0.952        20
           2      1.000     1.000     1.000        14

    accuracy                          0.963        54
   macro avg      0.970     0.967     0.967        54
weighted avg      0.966     0.963     0.963        54
```

Scenario-3: FIFA

FIFA Dataset is a football simulation video game developed by EA Vancouver as part of Electronic Arts' FIFA series.It has plenty of attributes covering all aspects of a real-life footballer in an attempt to imitate him as much as possible in the virtual world.

Problem Statement:

As EA Vancouver CEO decides to add a premier league in his next update of the game. He wishes to know how players are targeted through Transfer Window (A gap after a season), he wants to analyze the players through their attributes rather than their reputation to make the game as real as possible.

You being an Analyst, are expected to perform PCA to reduce the dimensions along with k means to find out the hidden patterns in the data.

Tasks to be performed:

In order to attain the above objective below, tasks should be performed:

- Importing and preprocessing the data is a priority- Beginner
- Identify the appropriate number of components by building a PCA model and explained_variance_ratio plot- Intermediate
- Fit and transform the PCA model again for the principal components found- Intermediate
- Build a Kmeans model using the transformed vector and write your inference- Advance
- Create a Kmeans model with the help Elbow method- Advance
- Analyze the above model and find the pattern in the data (Infer the same)- Advance

Dataset Description:

The dataset contains 89 columns and 18207 rows of FIFA Datasets by EA Vancouver. Here's a brief description of the 10 columns in the dataset:

- **ID** - Id of Each player
- **Name** - Name of each player
- **Age** - Age of each player
- **Nationality** - Nationality of each player
- **Overall** - Overall performance of each player
- **Potential** - Potential of each player
- **Club** - Name of the Club each player belong to
- **Value** - Net value of each player
- **Wage** - Amount each player earns per match
- **Preferred Foot** - Preferred foot of the player's (Right or Left)

Topics Covered:

- PCA
- Kmeans Clustering

Question-1: Importing and preprocessing the data is priority

In [ ]:
```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprec
ated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [ ]:
```
!wget https://www.dropbox.com/s/2m892ugv2m8mmnu/Fifa.csv?dl=0
```

```
--2020-07-14 05:40:25--  https://www.dropbox.com/s/2m892ugv2m8mmnu/Fifa.csv?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.65.1, 2620:100:6021:1::a27d:4101
Connecting to www.dropbox.com (www.dropbox.com)|162.125.65.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/2m892ugv2m8mmnu/Fifa.csv [following]
--2020-07-14 05:40:25--  https://www.dropbox.com/s/raw/2m892ugv2m8mmnu/Fifa.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc211848b481c240b7ca041d371b.dl.dropboxusercontent.com/cd/0/inline/A7cMS0IqNRPeNJvb7ZSpXv3hUqhL9Iw6
TXMv3rU1jC2Wh1pugHZNCgTMQUd_WG40RxRM7jFGyTwiJV6knRM8H_cswNsj-VENfoogd8vHs4hPhDZxwDKoO5wV25pTLrxyiug/file# [following]
--2020-07-14 05:40:26--  https://uc211848b481c240b7ca041d371b.dl.dropboxusercontent.com/cd/0/inline/A7cMS0IqNRPeNJvb7
ZSpXv3hUqhL9Iw6TXMv3rU1jC2Wh1pugHZNCgTMQUd_WG40RxRM7jFGyTwiJV6knRM8H_cswNsj-VENfoogd8vHs4hPhDZxwDKoO5wV25pTLrxyiug/fi
le
Resolving uc211848b481c240b7ca041d371b.dl.dropboxusercontent.com (uc211848b481c240b7ca041d371b.dl.dropboxusercontent.
com)... 162.125.65.15, 2620:100:6021:15::a27d:410f
Connecting to uc211848b481c240b7ca041d371b.dl.dropboxusercontent.com (uc211848b481c240b7ca041d371b.dl.dropboxusercont
ent.com)|162.125.65.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9140113 (8.7M) [text/plain]
Saving to: ‘Fifa.csv?dl=0’

Fifa.csv?dl=0       100%[===================>]   8.72M  28.6MB/s    in 0.3s

2020-07-14 05:40:27 (28.6 MB/s) - ‘Fifa.csv?dl=0’ saved [9140113/9140113]
```

In [ ]:
```
fifa= pd.read_csv("/content/Fifa.csv?dl=0")
fifa.head()
```

Out[ ]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	
0	0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	94	Barc
1	1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	94	Juv
2	2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	93	Paris Ge
3	3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	93	Mancl t
4	4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	92	Mancl

5 rows × 89 columns

In [ ]:
```
#Dropping the unnecessary columns
fifa.drop(columns=['Unnamed: 0'], inplace=True)
```

In [ ]:
```
fifa.columns
```

Out[ ]:
```
Index(['ID', 'Name', 'Age', 'Photo', 'Nationality', 'Flag', 'Overall',
       'Potential', 'Club', 'Club Logo', 'Value', 'Wage', 'Special',
       'Preferred Foot', 'International Reputation', 'Weak Foot',
       'Skill Moves', 'Work Rate', 'Body Type', 'Real Face', 'Position',
       'Jersey Number', 'Joined', 'Loaned From', 'Contract Valid Until',
       'Height', 'Weight', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW',
       'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB', 'LDM',
       'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB', 'Crossing',
       'Finishing', 'HeadingAccuracy', 'ShortPassing', 'Volleys', 'Dribbling',
       'Curve', 'FKAccuracy', 'LongPassing', 'BallControl', 'Acceleration',
       'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower',
       'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression',
       'Interceptions', 'Positioning', 'Vision', 'Penalties', 'Composure',
       'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving', 'GKHandling',
       'GKKicking', 'GKPositioning', 'GKReflexes', 'Release Clause'],
      dtype='object')
```

We are only interested in those attributes that constitute the skillset of a football player.

In [ ]:
```
#creating a dataframe fifa_opa by selecting only the required columns
fifa_opa = fifa[['Name', 'Age', 'Overall', 'Potential', 'Value', 'International Reputation', 'Height', 'Weight', 'Cros
sing', 'Wage','Club','Nationality','Finishing','HeadingAccuracy','ShortPassing','Volleys','Dribbling','Curve','FKAccur
acy','LongPassing','BallControl','Acceleration','SprintSpeed','Agility','Reactions','Balance','ShotPower','Jumping','S
tamina','Strength','LongShots','Aggression','Interceptions','Positioning','Vision','Penalties','Composure','Marking',
'StandingTackle','SlidingTackle']]
fifa_opa.head()
```

Out[ ]:

	Name	Age	Overall	Potential	Value	International Reputation	Height	Weight	Crossing	Wage	Club	Nationality	Finishing	HeadingAccuracy
0	L. Messi	31	94	94	€110.5M	5.0	5'7	159lbs	84.0	€565K	FC Barcelona	Argentina	95.0	70.0
1	Cristiano Ronaldo	33	94	94	€77M	5.0	6'2	183lbs	84.0	€405K	Juventus	Portugal	94.0	89.0
2	Neymar Jr	26	92	93	€118.5M	5.0	5'9	150lbs	79.0	€290K	Paris Saint-Germain	Brazil	87.0	62.0
3	De Gea	27	91	93	€72M	4.0	6'4	168lbs	17.0	€260K	Manchester United	Spain	13.0	21.0
4	K. De Bruyne	27	91	92	€102M	4.0	5'11	154lbs	93.0	€355K	Manchester City	Belgium	82.0	55.0

All the players in the dataset are sorted by their **'Overall'** value. We can reorder the samples in the dataset as 'Clustering' algorithms get biased by order

In [ ]:
```
# Considering 100 samples from data
fifa_opa = fifa_opa.sample(100)
fifa_opa.head()
```

Out[ ]:

	Name	Age	Overall	Potential	Value	International Reputation	Height	Weight	Crossing	Wage	Club	Nationality	Finishing	HeadingAccu
3324	T. Lees	27	72	73	€2.9M	1.0	6'1	161lbs	53.0	€19K	Sheffield Wednesday	England	36.0	
16531	N. Kariri	20	57	70	€180K	1.0	5'5	128lbs	54.0	€3K	Al Fayha	Saudi Arabia	29.0	
12694	L. John-Lewis	29	63	63	€375K	1.0	5'10	190lbs	54.0	€2K	Shrewsbury	England	58.0	
13444	M. Piątkowski	33	62	62	€190K	1.0	6'1	165lbs	37.0	€1K	Miedź Legnica	Poland	67.0	
13924	M. Frankoch	22	62	72	€425K	1.0	5'11	165lbs	44.0	€2K	Vendsyssel FF	Denmark	31.0	

We need to focus only on the attributes of a footballer like **Passing, Shooting, Dribbling etc** and identify those footballers who are similar

In [ ]:
```
# Focussing only on the attributes
fifa_pep = fifa_opa.drop([ 'Name', 'Club','Nationality','Overall','Potential','Height','Weight','Wage','Value'], axis=
1)
fifa_pep.head()
```

Out[ ]:

	Age	International Reputation	Crossing	Finishing	HeadingAccuracy	ShortPassing	Volleys	Dribbling	Curve	FKAccuracy	LongPassing	BallControl
3324	27	1.0	53.0	36.0	71.0	64.0	20.0	59.0	33.0	20.0	60.0	63.0
16531	20	1.0	54.0	29.0	39.0	55.0	27.0	56.0	49.0	43.0	50.0	54.0
12694	29	1.0	54.0	58.0	55.0	58.0	47.0	65.0	24.0	21.0	44.0	62.0
13444	33	1.0	37.0	67.0	67.0	53.0	55.0	64.0	44.0	33.0	40.0	61.0
13924	22	1.0	44.0	31.0	50.0	64.0	45.0	53.0	53.0	56.0	59.0	59.0

We need to scale the features as some of the features like `Age` are in the range 16-40 whereas `Crossing` is in the range 0-100. We will use **MinMax scaling** since we want discover clusters with reference to the best and worst values of players across different attributes.

In [ ]:
```
#scaling the data
from sklearn.preprocessing import MinMaxScaler
X_min = MinMaxScaler().fit_transform(fifa_pep)
X_min
```

Out[ ]:
```
array([[0.5       , 0.        , 0.59459459, ..., 0.94202899, 0.91304348,
        0.86956522],
       [0.15      , 0.        , 0.60810811, ..., 0.62318841, 0.66666667,
        0.63768116],
       [0.6       , 0.        , 0.60810811, ..., 0.1884058 , 0.23188406,
        0.11594203],
       ...,
       [0.5       , 0.        , 0.04054054, ..., 0.07246377, 0.07246377,
        0.04347826],
       [0.65      , 0.        , 0.40540541, ..., 0.86956522, 0.89855072,
        0.85507246],
       [0.45      , 0.        , 0.58108108, ..., 0.76811594, 0.8115942 ,
        0.65217391]])
```

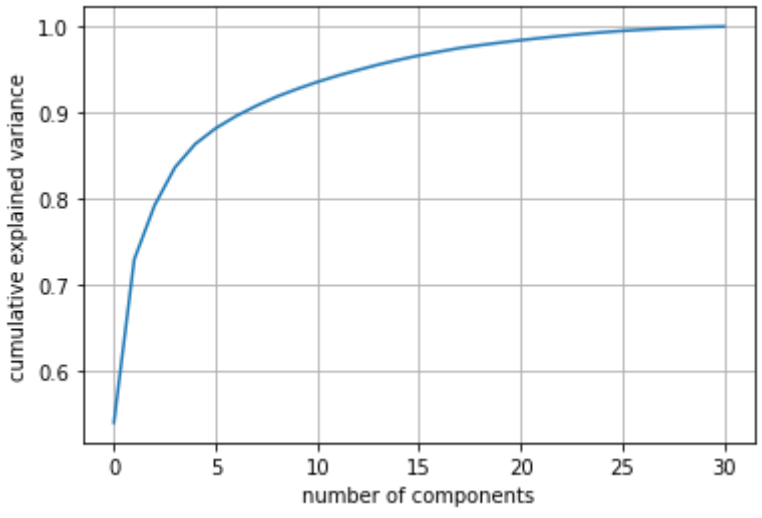
Question-2: Identify the appropriate number of components by building a PCA model and explained_variance_ratio plot

Since we are dealing with a huge number of attributes - 31, our analysis will be paralyzed by the **'Curse of Dimensionality'**. So, let us perform **Principal Component Analysis** to identify those principal components that explain most of the variance in the dataset.

In [ ]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=31)
pca.fit(X_min)
print(pca.explained_variance_ratio_)
```

```
[5.39404792e-01 1.89769949e-01 6.29099044e-02 4.40980825e-02
 2.72658913e-02 1.81826296e-02 1.41664106e-02 1.20479584e-02
 1.05983973e-02 8.94053386e-03 8.17484170e-03 7.31600904e-03
 6.54208315e-03 6.32998521e-03 5.53221450e-03 5.10363596e-03
 4.41213516e-03 4.26012445e-03 3.28137795e-03 3.02699854e-03
 2.78493667e-03 2.54445307e-03 2.46074206e-03 2.20895172e-03
 1.88085709e-03 1.74780854e-03 1.40600133e-03 1.22486405e-03
 1.06608760e-03 8.20131916e-04 4.91211373e-04]
```

In [ ]:
```
pca = PCA().fit(X_min)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.grid(True)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



After applying PCA, we can see that the dimensionality has been reduced from 31 to just 10

Question-3: Fit and transform the PCA model again for the principal components found

In [ ]:
```
pca=PCA(n_components=10)
pca.fit(X_min)
scores_pca=pca.transform(X_min)
```

In [ ]:
```
print(pca.explained_variance_ratio_[:10].sum())
```

```
0.9273845489385982
```

From the above plot, we can notice that 10 principal components are explaining 93% of the variance in the dataset

Question-4: Build a Kmeans model using the transformed vector and write your inference

In [ ]:
```
#let's build clusters using the Transformed_vector
from sklearn.cluster import KMeans
num_of_clusters = range(2,25)
error = []

for num_clusters in num_of_clusters:
    clusters = KMeans(num_clusters)
    clusters.fit(scores_pca)
    error.append(clusters.inertia_/100)

temp=pd.DataFrame({"Cluster_Numbers":num_of_clusters, "Error_Term":error})
temp
```

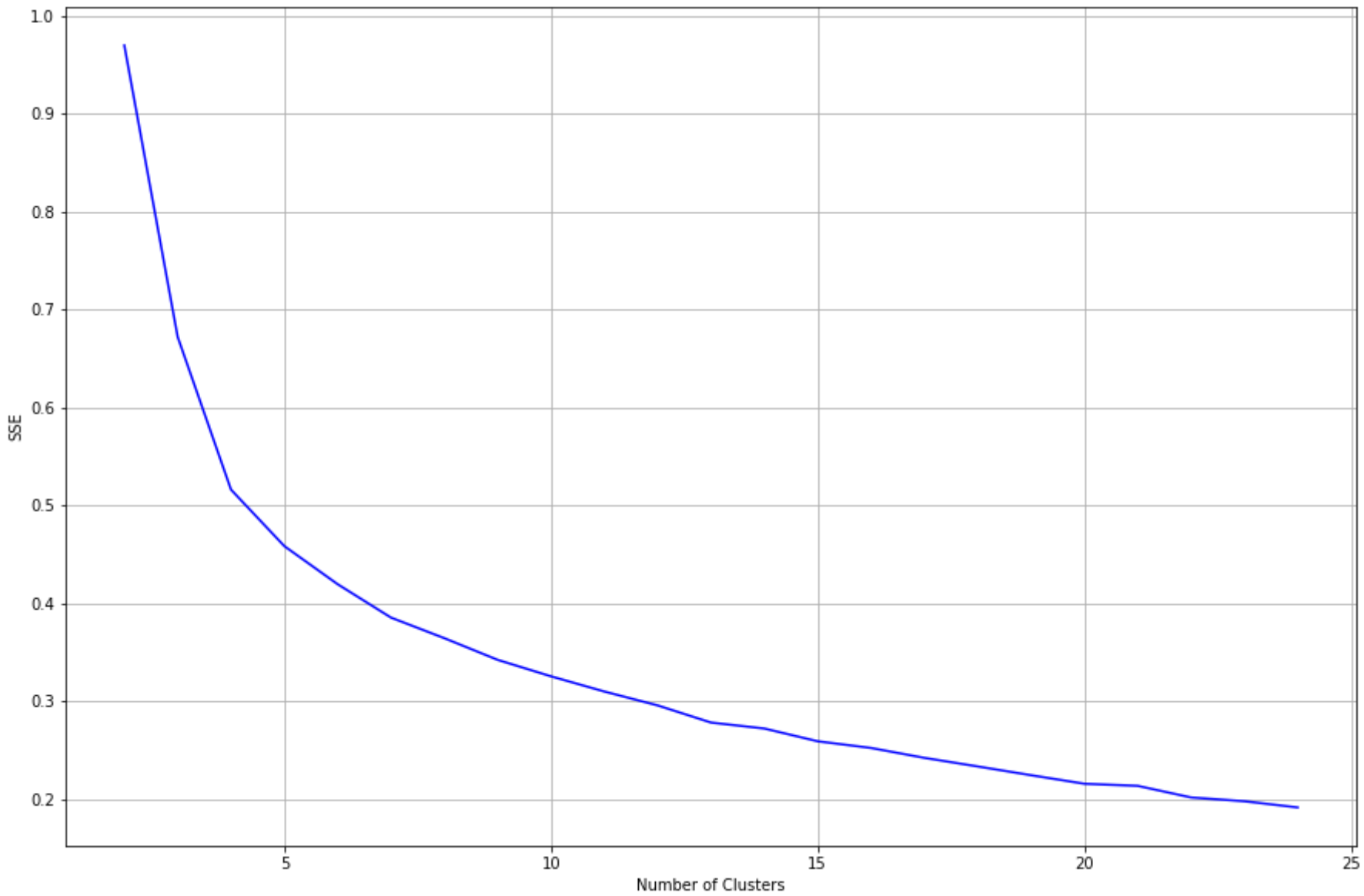
Out[ ]:

	Cluster_Numbers	Error_Term
0	2	0.970250
1	3	0.672435
2	4	0.516374
3	5	0.458662
4	6	0.419714
5	7	0.385618
6	8	0.364589
7	9	0.342439
8	10	0.325618
9	11	0.310089
10	12	0.295890
11	13	0.278364
12	14	0.272285
13	15	0.259240
14	16	0.252419
15	17	0.242303
16	18	0.233540
17	19	0.224585
18	20	0.215855
19	21	0.213704
20	22	0.201856
21	23	0.197974
22	24	0.191652

We can observe that there are so many error term hence, we can apply elbow method to find the right number of clusters

Question-5: Create a Kmeans model with the help Elbow method

In [ ]:
```
#Find the right number of clusters
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(figsize=(15,10))
plt.plot(temp.Cluster_Numbers, temp.Error_Term,  color='blue')
plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
plt.grid(True)
plt.show()
```



From the above elbow plot, it seems convergence started happening when the no.of clusters is **'5'**

In [ ]:
```
#Build 5 clusters
clusters1 = KMeans(5)
clusters1.fit(scores_pca)
clusters1.labels_
```

Out[ ]:
```
array([4, 0, 3, 3, 0, 3, 2, 1, 2, 1, 2, 0, 4, 0, 4, 3, 3, 3, 1, 3, 0, 1,
       0, 2, 3, 4, 4, 2, 0, 1, 1, 3, 1, 0, 3, 0, 3, 3, 0, 1, 3, 3, 0, 3,
       2, 1, 4, 0, 3, 4, 4, 0, 1, 1, 3, 1, 1, 1, 0, 3, 3, 1, 2, 1, 3, 4,
       1, 2, 0, 2, 4, 0, 0, 2, 4, 3, 0, 3, 1, 1, 4, 2, 0, 1, 0, 2, 3, 2,
       2, 3, 3, 3, 1, 0, 1, 1, 0, 2, 4, 1], dtype=int32)
```

Question-6: Analyze the above model and find the pattern in the data (Infer the same)

We have to make some Changes,

we have to convert the string type currency to integer inorder to apply some matematical operations

In [ ]:
```
#converting the string currency to integer.

def currencyconversion(amount):
    n_amount = []
    for s in amount:
        list(s)
        abbr = s[-1]
        if abbr is 'M':
            s = s[1:-1]
            s = float(''.join(s))
            s *= 1000000
        elif abbr is 'K':
            s = s[1:-1]
            s = float(''.join(s))
            s *= 1000
        else:
            s = 0
        n_amount.append(s)
    return n_amount
```

In [ ]:
```
#Applying the above function to convert string to integer
fifa_opa['Value'] = currencyconversion(list(fifa_opa['Value']))
fifa_opa['Wage'] = currencyconversion(list(fifa_opa['Wage']))
```

In [ ]:
```
# Apply some matematical operations which makes the analysis easier
players = fifa_opa['Name']
wage = fifa_opa['Wage']/1000
overall = fifa_opa['Overall']
age = fifa_opa['Age']
value = fifa_opa['Value']/100000
```

Create a Dataframe consisting of clusters, players, Wages in thousands, overall, age, and Transfer value in millions

In [ ]:
```
dff = pd.DataFrame({'clusters': clusters1.labels_, 'players': players, 'Wages in thousands':wage, 'overall':overall,
'age':age, 'Transfer value in millions':value})
dff.sort_values('clusters')
```

Out[ ]:

	clusters	players	Wages in thousands	overall	age	Transfer value in millions
14859	0	R. Garay	1.0	60	21	4.50
14696	0	I. Wilson	1.0	60	19	4.00
13296	0	M. Juel Andersen	1.0	62	20	4.50
8263	0	M. Ullmann	3.0	67	22	10.00
11679	0	H. Morita	2.0	64	23	5.25
...	...	...	...	...	...	...
4628	4	Defendi	8.0	71	32	14.00
12014	4	G. Arokoyo	1.0	64	25	5.00
7345	4	P. Hanlon	3.0	68	28	8.50
1267	4	I. Ordets	1.0	76	25	85.00
2702	4	J. Basanta	31.0	73	34	12.00

100 rows × 6 columns

Group those clusters and describe them statistically

In [ ]:
```
dff.groupby('clusters').describe().transpose()
```

Out[ ]:

	clusters	0	1	2	3	4
Wages in thousands	**count**	22.000000	24.000000	15.000000	26.000000	13.000000
	mean	2.181818	19.000000	4.533333	5.000000	8.384615
	std	2.422835	32.811716	5.125102	5.699123	8.597853
	min	1.000000	0.000000	1.000000	0.000000	1.000000
	25%	1.000000	3.750000	1.000000	1.250000	3.000000
	50%	1.000000	8.500000	2.000000	3.000000	6.000000
	75%	2.000000	19.000000	5.000000	5.750000	11.000000
	max	12.000000	160.000000	18.000000	26.000000	31.000000
overall	**count**	22.000000	24.000000	15.000000	26.000000	13.000000
	mean	59.636364	72.125000	65.400000	66.500000	69.615385
	std	4.665430	5.160995	8.007140	4.777028	3.617975
	min	50.000000	63.000000	51.000000	55.000000	63.000000
	25%	57.250000	69.000000	61.000000	63.250000	68.000000
	50%	60.000000	71.500000	65.000000	66.500000	71.000000
	75%	62.000000	75.250000	73.500000	69.750000	72.000000
	max	67.000000	83.000000	75.000000	75.000000	76.000000
age	**count**	22.000000	24.000000	15.000000	26.000000	13.000000
	mean	21.318182	26.500000	26.933333	26.653846	26.846154
	std	3.682473	4.211785	5.993647	4.049121	3.954874
	min	17.000000	19.000000	18.000000	19.000000	19.000000
	25%	19.000000	24.250000	22.000000	24.250000	25.000000
	50%	20.000000	26.500000	27.000000	26.000000	26.000000
	75%	22.000000	30.000000	30.000000	29.000000	30.000000
	max	34.000000	33.000000	36.000000	37.000000	34.000000
Transfer value in millions	**count**	22.000000	24.000000	15.000000	26.000000	13.000000
	mean	3.709091	57.916667	16.916667	16.744231	20.442308
	std	2.906955	69.991679	22.617247	23.042623	21.630215
	min	0.200000	0.000000	0.400000	0.000000	5.000000
	25%	1.875000	7.062500	0.850000	5.312500	7.750000
	50%	3.125000	33.500000	3.500000	8.375000	12.000000
	75%	4.500000	70.000000	34.500000	16.750000	27.000000
	max	11.000000	245.000000	60.000000	90.000000	85.000000

Inferences,

- If you are looking at controlling the **wages and transfer fee** and still get the brightest talent in the Dutch league, then you should focus on players in **Cluster 3**. The overall players in this cluster are at par with those in other clusters, and they have lower wages and transfer value than players in other clusters.
- If you have **unlimited budget to spend** and are focused only on getting the **top players** in the league, then you should focus on buying players in Clusters 0,1 and 4.
- You need to be a bit careful with players in **Cluster 2** as there seem to be a lot of **outliers having high wages**.

In [ ]:
```
```

Association Rule Mining

- An Association Rule **A** -> **B**, where **A** & **B** are two disjoint sets of items.
  - **A** -> **B** means that if a customer buys item **A**, he/she is also likely to buy item **B**.
- For Example, Customers who purchase a **pencil** have certain likelihood of purchasing a **sharepner** for their pencil as well.

Understanding Apriori Algorithm

- A Classical Algorithm used in Data Mining
- Used for finding frequent itemsets and mining association rules that may exist between different itemsets
- Operates very well on a dataset containing a large number of transactions
- Easy to understand and implement

Recommendation Engine

- Recommendation Engine (or Recommender System) predicts what a user may or may not like among a list of given items
- Helps the user to discover products or content that may not have come across otherwise

Scenario 1 Market Basket Analysis on Bakery Dataset

Dataset Description

This data belongs to a bakery called "The Bread Basket," located in Edinburgh, Scotland. This bakery presents a refreshing offer of Argentine and Spanish products.

It contains over 6,000 transactions and over 15,000 observations. The following are the features of this dataset and their definitions.

- **Date** : Categorical variable that tells us the date of the transactions (YYYY-MM-DD format). The column includes dates from 30/10/2016 to 09/04/2017.
- **Time** : Categorical variable that tells us the time of the transactions (HH:MM:SS format).
- **Transactions** : Quantitative variable that allows us to differentiate the transactions. The rows that share the same value in this field belong to the same transaction, that's why the data set has less transactions than observations.
- **Item** : Categorical variable with the products.

Let us say that the Bakery Owner hired you and wants you to use this data set to help them increase their sales by extracting hidden insights from the dataset.

Tasks to be Performed

- Import the libraries for exploratory analysis of the data, and load and analyze the dataset - Beginner
- Data Pre-processing - Beginner

```
Check for Null Values
Check for missing values and zeros and "None" values
```

- Visualizing and Understanding the Data - Intermediate

```
Which items do customers purchase most?
Which months were more successful?
```

- Perform Market Basket Analysis on the pre-processed dataset - Advance

```
Implement the Apriori Algorithm on the pre-processed dataset
Mine the "Association Rules" from the "frequent_itemsets"
```

Question 1 :

Import the libraries for exploratory analysis of the data, and load and analyze the dataset.

In [ ]:

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

# Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: F
utureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [ ]:

```
!wget https://www.dropbox.com/s/c30ckgtaay6ams5/BreadBasket.csv
```

```
--2020-07-15 11:05:54--  https://www.dropbox.com/s/c30ckgtaay6ams5/BreadBa
sket.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.1, 2620:100:601d:
1::a27d:501
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.1|:443... connec
ted.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/c30ckgtaay6ams5/BreadBasket.csv [following]
--2020-07-15 11:05:54--  https://www.dropbox.com/s/raw/c30ckgtaay6ams5/Bre
adBasket.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc2536d9fa5b58cf77fbd60e2f51.dl.dropboxusercontent.com/c
d/0/inline/A7kco3CoiCQM_Sa-ig5i5KOSLyqVILXqIxVGd74mCrQo0FPFwD03D_Hfk8tBZtM
N5-khFVy2mxeWlIR9nxxBfDZ7aoKZuXSipNt4PZsJVAyMcBlLlHq5AWdBTbGfdrrMap8/file#
[following]
--2020-07-15 11:05:55--  https://uc2536d9fa5b58cf77fbd60e2f51.dl.dropboxus
ercontent.com/cd/0/inline/A7kco3CoiCQM_Sa-ig5i5KOSLyqVILXqIxVGd74mCrQo0FPF
wD03D_Hfk8tBZtMN5-khFVy2mxeWlIR9nxxBfDZ7aoKZuXSipNt4PZsJVAyMcBlLlHq5AWdBTb
GfdrrMap8/file
Resolving uc2536d9fa5b58cf77fbd60e2f51.dl.dropboxusercontent.com (uc2536d9
fa5b58cf77fbd60e2f51.dl.dropboxusercontent.com)... 162.125.5.15, 2620:100:
601d:15::a27d:50f
Connecting to uc2536d9fa5b58cf77fbd60e2f51.dl.dropboxusercontent.com (uc25
36d9fa5b58cf77fbd60e2f51.dl.dropboxusercontent.com)|162.125.5.15|:443... c
onnected.
HTTP request sent, awaiting response... 200 OK
Length: 710518 (694K) [text/plain]
Saving to: ‘BreadBasket.csv’

BreadBasket.csv     100%[===================>] 693.87K  --.-KB/s    in 0.1
s

2020-07-15 11:05:55 (5.29 MB/s) - ‘BreadBasket.csv’ saved [710518/710518]
```

In [ ]:

```
df = pd.read_csv('BreadBasket.csv')
df.head()
```

Out[ ]:

	Date	Time	Transaction	Item
0	2016-10-30	09:58:11	1	Bread
1	2016-10-30	10:05:34	2	Scandinavian
2	2016-10-30	10:05:34	2	Scandinavian
3	2016-10-30	10:07:57	3	Hot chocolate
4	2016-10-30	10:07:57	3	Jam

In [ ]:

```
print('Dataset Information: \n')
print(df.info())
```

```
Dataset Information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21293 entries, 0 to 21292
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Date         21293 non-null  object
 1   Time         21293 non-null  object
 2   Transaction  21293 non-null  int64 
 3   Item         21293 non-null  object
dtypes: int64(1), object(3)
memory usage: 665.5+ KB
None
```

In [ ]:

```
print('Unique Items: ', df['Item'].nunique())
print( '\n', df['Item'].unique())
```

```
Unique Items:  95

 ['Bread' 'Scandinavian' 'Hot chocolate' 'Jam' 'Cookies' 'Muffin' 'Coffee'
 'Pastry' 'Medialuna' 'Tea' 'NONE' 'Tartine' 'Basket' 'Mineral water'
 'Farm House' 'Fudge' 'Juice' "Ella's Kitchen Pouches" 'Victorian Sponge'
 'Frittata' 'Hearty & Seasonal' 'Soup' 'Pick and Mix Bowls' 'Smoothies'
 'Cake' 'Mighty Protein' 'Chicken sand' 'Coke' 'My-5 Fruit Shoot'
 'Focaccia' 'Sandwich' 'Alfajores' 'Eggs' 'Brownie' 'Dulce de Leche'
 'Honey' 'The BART' 'Granola' 'Fairy Doors' 'Empanadas' 'Keeping It Local'
 'Art Tray' 'Bowl Nic Pitt' 'Bread Pudding' 'Adjustment' 'Truffles'
 'Chimichurri Oil' 'Bacon' 'Spread' 'Kids biscuit' 'Siblings'
 'Caramel bites' 'Jammie Dodgers' 'Tiffin' 'Olum & polenta' 'Polenta'
 'The Nomad' 'Hack the stack' 'Bakewell' 'Lemon and coconut' 'Toast'
 'Scone' 'Crepes' 'Vegan mincepie' 'Bare Popcorn' 'Muesli' 'Crisps'
 'Pintxos' 'Gingerbread syrup' 'Panatone' 'Brioche and salami'
 'Afternoon with the baker' 'Salad' 'Chicken Stew' 'Spanish Brunch'
 'Raspberry shortbread sandwich' 'Extra Salami or Feta' 'Duck egg'
 'Baguette' "Valentine's card" 'Tshirt' 'Vegan Feast' 'Postcard'
 'Nomad bag' 'Chocolates' 'Coffee granules ' 'Drinking chocolate spoons '
 'Christmas common' 'Argentina Night' 'Half slice Monster ' 'Gift voucher'
 'Cherry me Dried fruit' 'Mortimer' 'Raw bars' 'Tacos/Fajita']
```

Question 2 :

Data Pre-processing

- Check for Null Values

Check for missing values and zeros and "None" values

In [ ]:

```
# List how many null values for each feature:

print(df.isnull().sum().sort_values(ascending=False))
```

```
Item           0
Transaction    0
Time           0
Date           0
dtype: int64
```

It seems that our dataset is not missing any values. Let's now check for **None** values within the Item feature.

In [ ]:

```
print(df[df['Item']=='NONE'])
```

```
             Date      Time  Transaction  Item
26     2016-10-30  10:27:21           11  NONE
38     2016-10-30  10:34:36           15  NONE
39     2016-10-30  10:34:36           15  NONE
66     2016-10-30  11:05:30           29  NONE
80     2016-10-30  11:37:10           37  NONE
...           ...       ...          ...   ...
21108  2017-04-08  11:54:22         9590  NONE
21122  2017-04-08  12:58:25         9599  NONE
21254  2017-04-09  12:01:07         9666  NONE
21255  2017-04-09  12:04:13         9667  NONE
21266  2017-04-09  12:31:28         9672  NONE

[786 rows x 4 columns]
```

From above, you can see that we have **NONE** values in our dataset.

It means that no item was purchased, or the name of the item was not recorded. Either way, this is of no use to us so we drop these rows.

In [ ]:

```
df.drop(df[df['Item']=='NONE'].index, inplace=True)
```

In [ ]:

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20507 entries, 0 to 21292
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Date         20507 non-null  object
 1   Time         20507 non-null  object
 2   Transaction  20507 non-null  int64 
 3   Item         20507 non-null  object
dtypes: int64(1), object(3)
memory usage: 801.1+ KB
None
```

As we can see above, the Date and Time features are not numerical types. For better future visualization and understanding of the data, we add a few more features to this DataFrame based on the information from these two features.

In [ ]:

```
# Year
df['Year'] = df['Date'].apply(lambda x: x.split("-")[0])
# Month
df['Month'] = df['Date'].apply(lambda x: x.split("-")[1])
# Day
df['Day'] = df['Date'].apply(lambda x: x.split("-")[2])
```

In [ ]:

```
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20507 entries, 0 to 21292
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Date         20507 non-null  object
 1   Time         20507 non-null  object
 2   Transaction  20507 non-null  int64 
 3   Item         20507 non-null  object
 4   Year         20507 non-null  object
 5   Month        20507 non-null  object
 6   Day          20507 non-null  object
dtypes: int64(1), object(6)
memory usage: 1.3+ MB
None
         Date      Time  Transaction           Item  Year Month Day
0  2016-10-30  09:58:11            1          Bread  2016    10  30
1  2016-10-30  10:05:34            2   Scandinavian  2016    10  30
2  2016-10-30  10:05:34            2   Scandinavian  2016    10  30
3  2016-10-30  10:07:57            3  Hot chocolate  2016    10  30
4  2016-10-30  10:07:57            3            Jam  2016    10  30
```

Question 3 :

Visualizing and Understanding the Data

- Which items do customers purchase most?
- Which months were more successful?

Let's check the most sold items from the bakery:

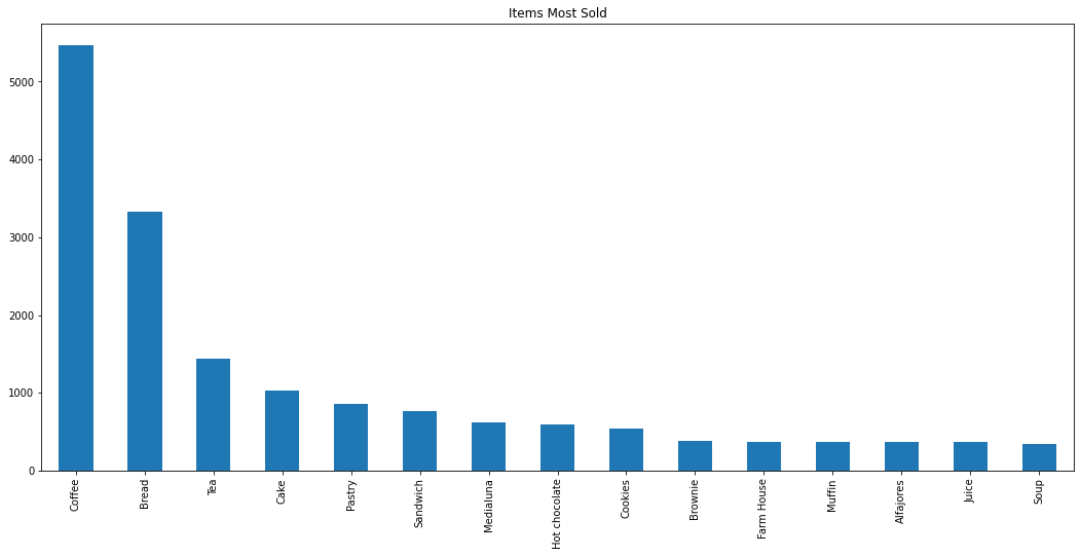
In [ ]:

```
most_sold = df['Item'].value_counts().head(15)

print('Most Sold Items: \n')
print(most_sold)
```

```
Most Sold Items:

Coffee            5471
Bread             3325
Tea               1435
Cake              1025
Pastry             856
Sandwich           771
Medialuna          616
Hot chocolate      590
Cookies            540
Brownie            379
Farm House         374
Muffin             370
Alfajores          369
Juice              369
Soup               342
Name: Item, dtype: int64
```

In [ ]:

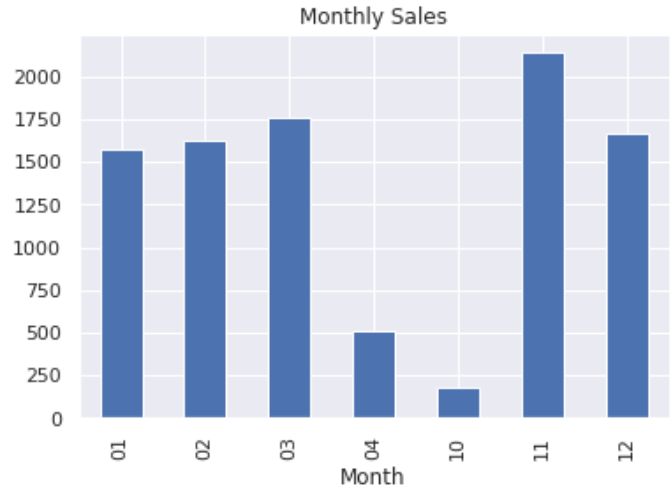
```
plt.figure(figsize=(18,8))

most_sold.plot(kind='bar')
plt.title('Items Most Sold')
plt.show()
```



From above, **Coffee** is the most sold item, followed by **bread**, **tea**, **cake**, and **pastry**, respectively

In [ ]:

```
# let's check out which months bring in the most sales

df.groupby('Month')['Transaction'].nunique().plot(kind='bar', title='Monthly Sales')
plt.show()
```



Is this drastic difference in sales is due to the dataset having less data for April and October ? This would make sense, as they are the outlier months in the dataset. Let's check to see if there are less daily transactions recorded for these months in comparison to the others.

In [ ]:

```
print(df.groupby('Month')['Day'].nunique())
```

```
Month
01    30
02    28
03    31
04     9
10     2
11    30
12    29
Name: Day, dtype: int64
```

From above, you can see that Only 9 days' worth of transactions were recorded for April, and 2 days for October.

Question 4 :

Perform Market Basket Analysis on the pre-processed dataset

In [ ]:

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules, apriori
```

Let's now create a list of the unique transactions so that we can transform our data into the correct format using TransactionEncoder.

In [ ]:

```
transaction_list = []

# For loop to create a list of the unique transactions throughout the dataset:
for i in df['Transaction'].unique():
    tlist = list(set(df[df['Transaction']==i]['Item']))
    if len(tlist)>0:
        transaction_list.append(tlist)
print(len(transaction_list))
```

9465

In [ ]:

```
te = TransactionEncoder()
te_ary = te.fit(transaction_list).transform(transaction_list)
df2 = pd.DataFrame(te_ary, columns=te.columns_)
```

The three major components of the Apriori Algorithm are:

- **Support :** It can be defined as the popularity of a particular item.It can be calculated as the number of transactions involving that particular item divided by the total number of transactions.
- **Confidence :**It is the likelihood of an item **B** being purchased when **A** was purchased.
- **Lift :**It is the likelihood of an item **B** being purchased when item **A** is purchased while considering the popularity of **B**.

Now let's apply the Apriori Algorithm.

- Use the min_threshold parameter in the association rules for the lift metric to be 1.0 because if it is less than one, then the two items are not likely to be bought together
- Sort the values by confidence to see the likelihood that an item is bought if its antecedent is bought

In [ ]:

```
frequent_itemsets = apriori(df2, min_support=0.01, use_colnames=True)
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1.0)
rules.sort_values('confidence', ascending=False)
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leve
30	(Toast)	(Coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	0.0(
28	(Spanish Brunch)	(Coffee)	0.018172	0.478394	0.010882	0.598837	1.251766	0.0(
18	(Medialuna)	(Coffee)	0.061807	0.478394	0.035182	0.569231	1.189878	0.0(
22	(Pastry)	(Coffee)	0.086107	0.478394	0.047544	0.552147	1.154168	0.0(
1	(Alfajores)	(Coffee)	0.036344	0.478394	0.019651	0.540698	1.130235	0.0(
16	(Juice)	(Coffee)	0.038563	0.478394	0.020602	0.534247	1.116750	0.0(
24	(Sandwich)	(Coffee)	0.071844	0.478394	0.038246	0.532353	1.112792	0.0(
6	(Cake)	(Coffee)	0.103856	0.478394	0.054728	0.526958	1.101515	0.0(
27	(Scone)	(Coffee)	0.034548	0.478394	0.018067	0.522936	1.093107	0.0(
12	(Cookies)	(Coffee)	0.054411	0.478394	0.028209	0.518447	1.083723	0.0(
15	(Hot chocolate)	(Coffee)	0.058320	0.478394	0.029583	0.507246	1.060311	0.0(
4	(Brownie)	(Coffee)	0.040042	0.478394	0.019651	0.490765	1.025860	0.0(
20	(Muffin)	(Coffee)	0.038457	0.478394	0.018806	0.489011	1.022193	0.0(
2	(Pastry)	(Bread)	0.086107	0.327205	0.029160	0.338650	1.034977	0.0(
10	(Cake)	(Tea)	0.103856	0.142631	0.023772	0.228891	1.604781	0.0(
39	(Coffee, Tea)	(Cake)	0.049868	0.103856	0.010037	0.201271	1.937977	0.0(
32	(Sandwich)	(Tea)	0.071844	0.142631	0.014369	0.200000	1.402222	0.0(
9	(Hot chocolate)	(Cake)	0.058320	0.103856	0.011410	0.195652	1.883874	0.0(
38	(Cake, Coffee)	(Tea)	0.054728	0.142631	0.010037	0.183398	1.285822	0.0(
11	(Tea)	(Cake)	0.142631	0.103856	0.023772	0.166667	1.604781	0.0(
37	(Pastry)	(Bread, Coffee)	0.086107	0.090016	0.011199	0.130061	1.444872	0.0(
36	(Bread, Coffee)	(Pastry)	0.090016	0.086107	0.011199	0.124413	1.444872	0.0(
7	(Coffee)	(Cake)	0.478394	0.103856	0.054728	0.114399	1.101515	0.0(
34	(Bread, Coffee)	(Cake)	0.090016	0.103856	0.010037	0.111502	1.073621	0.0(
8	(Cake)	(Hot chocolate)	0.103856	0.058320	0.011410	0.109868	1.883874	0.0(
33	(Tea)	(Sandwich)	0.142631	0.071844	0.014369	0.100741	1.402222	0.0(
23	(Coffee)	(Pastry)	0.478394	0.086107	0.047544	0.099382	1.154168	0.0(
40	(Cake)	(Coffee, Tea)	0.103856	0.049868	0.010037	0.096643	1.937977	0.0(
35	(Cake)	(Bread, Coffee)	0.103856	0.090016	0.010037	0.096643	1.073621	0.0(
3	(Bread)	(Pastry)	0.327205	0.086107	0.029160	0.089119	1.034977	0.0(
25	(Coffee)	(Sandwich)	0.478394	0.071844	0.038246	0.079947	1.112792	0.0(
19	(Coffee)	(Medialuna)	0.478394	0.061807	0.035182	0.073542	1.189878	0.0(

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leve
41	(Tea)	(Cake, Coffee)	0.142631	0.054728	0.010037	0.070370	1.285822	0.00
14	(Coffee)	(Hot chocolate)	0.478394	0.058320	0.029583	0.061837	1.060311	0.00
13	(Coffee)	(Cookies)	0.478394	0.054411	0.028209	0.058966	1.083723	0.00
31	(Coffee)	(Toast)	0.478394	0.033597	0.023666	0.049470	1.472431	0.00
17	(Coffee)	(Juice)	0.478394	0.038563	0.020602	0.043065	1.116750	0.00
0	(Coffee)	(Alfajores)	0.478394	0.036344	0.019651	0.041078	1.130235	0.00
5	(Coffee)	(Brownie)	0.478394	0.040042	0.019651	0.041078	1.025860	0.00
21	(Coffee)	(Muffin)	0.478394	0.038457	0.018806	0.039311	1.022193	0.00
26	(Coffee)	(Scone)	0.478394	0.034548	0.018067	0.037765	1.093107	0.00
29	(Coffee)	(Spanish Brunch)	0.478394	0.018172	0.010882	0.022747	1.251766	0.00

Inferences

The data clearly shows that coffee is a popular consequent, which makes sense because it is a bakery. Besides coffee, let's look at the more interesting item correlations (antecedant(s) -> consequent):

- Pastry -> Bread
- Cake -> Tea
- (Coffee + Tea) -> Cake
- Sandwhich -> Tea
- Hot Chocolate -> Cake

The bakery owner can make decisions based on these findings. For example, he might want to place their freshly baked **bread** near their **pastries**, since customers who purchase pastries are likely to buy bread

Scenario 2 Analysing Groceries Dataset

Dataset Description

The dataset contains 9835 transactions with 32 different items being sold at the store over the course of a week.

The Manager of a retail store, located in Chennai is trying to find out an association rule between 32 different items, to find out which items are frequently bought together so that he can keep the items together to boost the sales.

You as a Data Scientist is required to perform Market Basket Analysis using Apriori Algorithm on a dataset containing 9835 transactions of a retail store.

Tasks to be Performed

- Load, analyze,and pre-process the dataset and convert the dataframe into a list of lists - Beginner
- Implement the Apriori Algorithm from the Apyori package - Intermediate
- Analyse the first relation obtained from the rules obtained & derive useful insgihts - Advanced

Topics Covered

- Association Rule Mining
- Apriori Algorithm

In [ ]:

```
!wget https://www.dropbox.com/s/6gnowmpqro12b3o/Grocery.csv
```

```
--2020-07-13 03:58:48--  https://www.dropbox.com/s/6gnowmpqro12b3o/Grocer
y.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.1, 2620:100:601d:
1::a27d:501
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.1|:443... connec
ted.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/6gnowmpqro12b3o/Grocery.csv [following]
--2020-07-13 03:58:48--  https://www.dropbox.com/s/raw/6gnowmpqro12b3o/Gro
cery.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc8b2737d212a52907626e7d4bc9.dl.dropboxusercontent.com/c
d/0/inline/A7a6VHRRW4SmXmVovL_4Z64pWXynh_vBVVZfY2TpA4YuS25QDHTT4EsOU2HBnvA
Ppdty1YZ06uWD18I8nHTqhxRC08ToDPXl22CFBmzJS6pRCe815VtOVWS5sOtdcDT93AE/file#
[following]
--2020-07-13 03:58:49--  https://uc8b2737d212a52907626e7d4bc9.dl.dropboxus
ercontent.com/cd/0/inline/A7a6VHRRW4SmXmVovL_4Z64pWXynh_vBVVZfY2TpA4YuS25Q
DHTT4EsOU2HBnvAPpdty1YZ06uWD18I8nHTqhxRC08ToDPXl22CFBmzJS6pRCe815VtOVWS5sO
tdcDT93AE/file
Resolving uc8b2737d212a52907626e7d4bc9.dl.dropboxusercontent.com (uc8b2737
d212a52907626e7d4bc9.dl.dropboxusercontent.com)... 162.125.5.15, 2620:100:
601d:15::a27d:50f
Connecting to uc8b2737d212a52907626e7d4bc9.dl.dropboxusercontent.com (uc8b
2737d212a52907626e7d4bc9.dl.dropboxusercontent.com)|162.125.5.15|:443... c
onnected.
HTTP request sent, awaiting response... 200 OK
Length: 782050 (764K) [text/plain]
Saving to: ‘Grocery.csv’

Grocery.csv         100%[===================>] 763.72K  --.-KB/s    in 0.1
s

2020-07-13 03:58:50 (5.45 MB/s) - ‘Grocery.csv’ saved [782050/782050]
```

Question 1:

Load, analyze,and pre-process the dataset and convert the dataframe into a list of lists - Beginner

In [ ]:

```
import pandas as pd
df = pd.read_csv('/content/Grocery.csv',header=None)
df.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
3	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

From above, you can see that the dataset contains NaN values. That means that item was not purchased in that transaction.

Analyzing the Dataset

In [ ]:

```
df.describe()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	
count	9835	7676	6033	4734	3729	2874	2229	1684	1246	896	
unique	158	151	155	153	150	137	138	140	128	120	
top	sausage	whole milk	whole milk	whole milk	rolls/buns	soda	soda	shopping bags	soda	shopping bags	shopp ba
freq	825	654	506	315	176	150	120	76	61	49	

In [ ]:

```
df.shape
```

Out[ ]:

```
(9835, 32)
```

Converting the dataframe into a list of lists.

Apriori library requires the dataset to be in the form of a list of lists, where the entire dataset is a big list and each transaction in the dataset is stored as a list inside the bigger list.

In [ ]:

```
dataset=[]
for i in range(0, df.shape[0]):
    dataset.append([str(df.values[i,j]) for j in range(0, df.shape[1])])
```

In [2]:

```
#dataset
```

Question 2:

Implement the Apriori Algorithm from the Apyori package - Intermediate

In [ ]:

```
!pip install apyori
```

```
Collecting apyori
  Downloading https://files.pythonhosted.org/packages/5e/62/5ffde5c473ea4b
033490617ec5caa80d59804875ad3c3c57c0976533a21a/apyori-1.1.2.tar.gz
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-cp36-none-any.whl size=5
975 sha256=1e45f4e797824961fed8dbe7b42550d6d3831ad8ed1297787a48b5986fe079b
3
  Stored in directory: /root/.cache/pip/wheels/5d/92/bb/474bbadbc8c0062b9e
b168f69982a0443263f8ab1711a8cad0
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

In [ ]:

```
import pandas as pd
import numpy as np
from apyori import apriori
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
rules = apriori(dataset, min_support=0.002,min_confidence=0.7,min_lift=3,min_length=1,
use_colnames=True)
result = list(rules)
```

In [1]:

```
#result
```

Question 3:

Analyze the first relation obtained from the rules obtained previously & derive useful insights - Advanced

In [ ]:

```
print(result[0])
```

```
RelationRecord(items=frozenset({'other vegetables', 'baking powder', 'root
vegetables'}), support=0.002541942043721403, ordered_statistics=[OrderedSt
atistic(items_base=frozenset({'baking powder', 'root vegetables'}), items_
add=frozenset({'other vegetables'}), confidence=0.7142857142857143, lift=
3.691539674198634)])
```

From above, you can observe the following for the first rule

- **Support =** 0.0025 implies the frequnecy of a transaction containing **root vegetables**, **other vegetables** and **baking powder**. Therefore, we can say that in every 10,000 transactions 25 transaction contains **root vegetables**, **other vegetables** and **baking powder**
- **Confidence =** 0.71 implies that if a customer buys both **root vegetables** and **baking powder**, we can say with 71% confidence that he/she will also buy **other vegetables**
- **Lift=** 3.69 implies that if a customer buys both **root vegetables** and **baking powder** he/she is 3.69 times more likely to buy **other vegetables**

Scenario 3 Movie Recommender System

Dataset Description

The dataset (**movies.csv**) contains 3 attributes:

> **1. movieId**: Represents ID of the Movie
>
> **2. title**: Title of the Movie
>
> **1. genres**: Genres of the Movie

The dataset (**ratings.csv**) contains 4 attributes:

> **1. userId**: Represents ID of the User
>
> **2. movieId**: Represents ID of the Movie
>
> **3. rating**: Represents rating of the movie
>
> **4. timestamp**: Represents seconds since midnight UTC of January 1,1970

Implement a Content-Based Movie Recommender System in Python

Tasks to be Performed

- Load, analyze, and pre-process the data set - Beginner
- Initialize the **userInput** and take title of movies and ratings accordingly in a list of dictionaries - Beginner
- Add movieId to input user. Extract the input movie's ID's from the movies dataframe and add them into it - Intermediate
- Find the movies that the input user has watched from the original data frame that contains the genres defined with binary values
- Convert each of the genres into weights. Multiply the input's reviews with the input's genre and then, sum up the final table

Topics Covered

- Recommendation System

In [ ]:

```
!wget https://www.dropbox.com/s/e15ytdn1w0c6onz/movies.csv
!wget https://www.dropbox.com/s/43i2lln73t5qkh9/ratings.csv
```

Question 1:

Load, analyse, and pre-process the data set

In [ ]:

```
#Dataframe manipulation library
import pandas as pd
#Math functions, we'll only need the sqrt function so let's import only that
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Now let's read each file into their Dataframes:

In [ ]:

```
movies_df = pd.read_csv('movies.csv')
#Loading the user information using the pandas read_csv function
ratings_df = pd.read_csv('ratings.csv')

movies_df.head()
```

Out[ ]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [ ]:

```
movies_df.shape
```

Out[ ]:

```
(34208, 3)
```

In [ ]:

```
ratings_df.shape
```

Out[ ]:

```
(22884377, 4)
```

Here, we are removing the year from the **title** column using the pandas' **replace** function and storing it in a new **year** column

In [ ]:

```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have years in their t
itles
movies_df['year'] = movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
#Applying the strip function to get rid of any ending whitespace characters that may ha
ve appeared
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

Out[ ]:

	movieId	title	genres	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995
1	2	Jumanji	Adventure|Children|Fantasy	1995
2	3	Grumpier Old Men	Comedy|Romance	1995
3	4	Waiting to Exhale	Comedy|Drama|Romance	1995
4	5	Father of the Bride Part II	Comedy	1995

With that, let's also split the values in the **Genres** column into a **list of Genres** by applying Python's **split** string function on the correct column

In [ ]:

```
#Every genre is separated by a | so we simply have to call the split function on |
movies_df['genres'] = movies_df.genres.str.split('|')
movies_df.head()
```

Out[ ]:

	movieId	title	genres	year
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995
1	2	Jumanji	[Adventure, Children, Fantasy]	1995
2	3	Grumpier Old Men	[Comedy, Romance]	1995
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995
4	5	Father of the Bride Part II	[Comedy]	1995

We will use the One Hot Encoding technique to convert the list of genres to a vector where each column corresponds to one possible value of the feature.

In [ ]:

```
#Copying the movie dataframe into a new one since we won't need to use the genre inform
ation in our first case.
moviesWithGenres_df = movies_df.copy()

#For every row in the dataframe, iterate through the list of genres and place a 1 into
 the corresponding column
for index, row in movies_df.iterrows():
    for genre in row['genres']:
        moviesWithGenres_df.at[index, genre] = 1
#Filling in the NaN values with 0 to show that a movie doesn't have that column's genre
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

Out[ ]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0
2	3	Grumpier Old Men	[Comedy, Romance]	1995	0.0	0.0	0.0	1.0	0.0
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995	0.0	0.0	0.0	1.0	0.0
4	5	Father of the Bride Part II	[Comedy]	1995	0.0	0.0	0.0	1.0	0.0

Next, let's look at the ratings dataframe.

In [ ]:

```
ratings_df.head()
```

Out[ ]:

	userId	movieId	rating	timestamp
0	1	169	2.5	1204927694
1	1	2471	3.0	1204927438
2	1	48516	5.0	1204927435
3	2	2571	3.5	1436165433
4	2	109487	4.0	1436165496

Every row in the ratings dataframe has a **userId** associated with at least one movie, a rating and a timestamp.

In [ ]:

```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
ratings_df.head()
```

Out[ ]:

	userId	movieId	rating
0	1	169	2.5
1	1	2471	3.0
2	1	48516	5.0
3	2	2571	3.5
4	2	109487	4.0

Question 2:

Implement the Content-Based recommendation system using Python

Initialize the **userInput** and take title of movies and ratings accordingly in a list of dictionaries

In [ ]:

```
userInput = [
            {'title':'Breakfast Club, The', 'rating':5},
            {'title':'Toy Story', 'rating':3.5},
            {'title':'Jumanji', 'rating':2},
            {'title':"Pulp Fiction", 'rating':5},
            {'title':'Akira', 'rating':4.5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Out[ ]:

	title	rating
0	Breakfast Club, The	5.0
1	Toy Story	3.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	4.5

Question 3:

Add movieId to input user. Extract the input movie's ID's from the movies dataframe and add them into it

In [ ]:

```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

Out[ ]:

	movieId	title	rating
0	1	Toy Story	3.5
1	2	Jumanji	2.0
2	296	Pulp Fiction	5.0
3	1274	Akira	4.5
4	1968	Breakfast Club, The	5.0

Question 4:

Find the movies that the input user has watched from the original data frame that contains the genres defined with binary values

In [ ]:

```
#Filtering out the movies from the input
userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['movieId'].tolist())]
userMovies
```

Out[ ]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantas
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.
293	296	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	1994	0.0	0.0	0.0	1.0	0.
1246	1274	Akira	[Action, Adventure, Animation, Sci-Fi]	1988	1.0	1.0	0.0	0.0	0.
1885	1968	Breakfast Club, The	[Comedy, Drama]	1985	0.0	0.0	0.0	1.0	0.

We only need the actual genre table, so drop the **movieId**, **title**, **genres** and **year** columns

In [ ]:

```
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Dropping unnecessary issues due to save memory and to avoid issues
userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop(
'year', 1)
userGenreTable
```

Out[ ]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thr
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	

Question 5:

Convert each of the genre into weights. Multiply the input's reviews with the input's genre and then, sum up the final table

In [ ]:

```
inputMovies['rating']
```

Out[ ]:

```
0    3.5
1    2.0
2    5.0
3    4.5
4    5.0
Name: rating, dtype: float64
```

In [ ]:

```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

Out[ ]:

```
Adventure            10.0
Animation             8.0
Children              5.5
Comedy               13.5
Fantasy               5.5
Romance               0.0
Drama                10.0
Action                4.5
Crime                 5.0
Thriller              5.0
Horror                0.0
Mystery               0.0
Sci-Fi                4.5
IMAX                  0.0
Documentary           0.0
War                   0.0
Musical               0.0
Western               0.0
Film-Noir             0.0
(no genres listed)    0.0
dtype: float64
```

Let's start by extracting the genre table from the original dataframe:

In [ ]:

```
#Now let's get the genres of every movie in our original dataframe
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('yea
r', 1)
genreTable.head()
```

Out[ ]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crim
movieId									
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.

In [ ]:

```
genreTable.shape
```

Out[ ]:

```
(34208, 20)
```

In [ ]:

```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df.head()
```

Out[ ]:

```
movieId
1    0.594406
2    0.293706
3    0.188811
4    0.328671
5    0.188811
dtype: float64
```

In [ ]:

```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df.head()
```

Out[ ]:

```
movieId
5018      0.748252
26093     0.734266
27344     0.720280
148775    0.685315
6902      0.678322
dtype: float64
```

In [ ]:

```
#The final recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

Out[ ]:

	movieId	title	genres	year
664	673	Space Jam	[Adventure, Animation, Children, Comedy, Fanta...	1996
1824	1907	Mulan	[Adventure, Animation, Children, Comedy, Drama...	1998
2902	2987	Who Framed Roger Rabbit?	[Adventure, Animation, Children, Comedy, Crime...	1988
4923	5018	Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mys...	1991
6793	6902	Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, S...	2002
8605	26093	Wonderful World of the Brothers Grimm, The	[Adventure, Animation, Children, Comedy, Drama...	1962
8783	26340	Twelve Tasks of Asterix, The (Les douze travau...	[Action, Adventure, Animation, Children, Comed...	1976
9296	27344	Revolutionary Girl Utena: Adolescence of Utena...	[Action, Adventure, Animation, Comedy, Drama, ...	1999
9825	32031	Robots	[Adventure, Animation, Children, Comedy, Fanta...	2005
11716	51632	Atlantis: Milo's Return	[Action, Adventure, Animation, Children, Comed...	2003
11751	51939	TMNT (Teenage Mutant Ninja Turtles)	[Action, Adventure, Animation, Children, Comed...	2007
13250	64645	The Wrecking Crew	[Action, Adventure, Comedy, Crime, Drama, Thri...	1968
16055	81132	Rubber	[Action, Adventure, Comedy, Crime, Drama, Film...	2010
18312	91335	Gruffalo, The	[Adventure, Animation, Children, Comedy, Drama]	2009
22778	108540	Ernest & Célestine (Ernest et Célestine)	[Adventure, Animation, Children, Comedy, Drama...	2012
22881	108932	The Lego Movie	[Action, Adventure, Animation, Children, Comed...	2014
25218	117646	Dragonheart 2: A New Beginning	[Action, Adventure, Comedy, Drama, Fantasy, Th...	2000
26442	122787	The 39 Steps	[Action, Adventure, Comedy, Crime, Drama, Thri...	1959
32854	146305	Princes and Princesses	[Animation, Children, Comedy, Drama, Fantasy, ...	2000
33509	148775	Wizards of Waverly Place: The Movie	[Adventure, Children, Comedy, Drama, Fantasy, ...	2009

Scenario 4 Mining Association Rules on Wine Dataset

Dataset Description

The dataset contains 20 transactions with different kinds of alcohol being sold at the Liquor Shop over the course of a week.

Let us say that you are a new wine taster and want to use this data set on Alcohol to perform Market Basket Analysis using the Apriori Algorithm on a dataset containing 20 transactions over a period of one week of a retail store to increase the Sales.

Tasks to be Performed

- Load, analyze,and pre-process the dataset and convert the dataframe into a list of lists - Beginner
- Implement the Apriori Algorithm from the Apyori package - Intermediate
- Analyse the first relation obtained from the rules obtained & derive useful insights - Advanced

Topics Covered

- Association Rule Mining
- Apriori Algorithm

In [ ]:

```
!wget https://www.dropbox.com/s/4108bbm2mn536w3/Wine.xlsx
```

```
--2020-07-13 14:49:32--  https://www.dropbox.com/s/4108bbm2mn536w3/Wine.xl
sx
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:100:603
2:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443... conne
cted.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/4108bbm2mn536w3/Wine.xlsx [following]
--2020-07-13 14:49:33--  https://www.dropbox.com/s/raw/4108bbm2mn536w3/Win
e.xlsx
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc851c6d51978ea67bc1eb2fdcb4.dl.dropboxusercontent.com/c
d/0/inline/A7fSFL8jfekkscM5dCbwdBtcGCfz_AGsvZ7ozI-gfK6pSe2bU9SrYM3-b91kCiV
u4Russ40nO5OKPWLQo1BKlSau3ClacUHFs-czUwCZObx2-0WfAd8FYO_Z2WF69TTFLeM/file#
[following]
--2020-07-13 14:49:33--  https://uc851c6d51978ea67bc1eb2fdcb4.dl.dropboxus
ercontent.com/cd/0/inline/A7fSFL8jfekkscM5dCbwdBtcGCfz_AGsvZ7ozI-gfK6pSe2b
U9SrYM3-b91kCiVu4Russ40nO5OKPWLQo1BKlSau3ClacUHFs-czUwCZObx2-0WfAd8FYO_Z2W
F69TTFLeM/file
Resolving uc851c6d51978ea67bc1eb2fdcb4.dl.dropboxusercontent.com (uc851c6d
51978ea67bc1eb2fdcb4.dl.dropboxusercontent.com)... 162.125.82.15, 2620:10
0:6032:15::a27d:520f
Connecting to uc851c6d51978ea67bc1eb2fdcb4.dl.dropboxusercontent.com (uc85
1c6d51978ea67bc1eb2fdcb4.dl.dropboxusercontent.com)|162.125.82.15|:443...
connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/A7fo9cLZgNXN7ypgHcI_sbWfHRjY2wg-Vu55fYjhwAXvgJVFuM
oQyBFnit-pft_Q4HYQJ2uGR3y2N9FiaGe3RN6bwFggAEXqtBt-aDzMK-B3C8q60-FUDmgyMM3Y
twx8a_X6DjvXjMoxW27j5p79d0lJM4sGErl85Nr9nEWbZ7cm74JgYeaZKvK91Uv5vmclFKpuO6
PlkktJ_z8c9fn0E4zHaCXeA8DQc5FOd2MuRccWtJ1elXtD3XvGrmiqJgXz-e7Thweo75o2-QwX
j0u7xibswyM0gmXSqvS7_BjP68ul4ndzgNJoF5JofZaZFzKik9Li9fd0F_w9tKeTY1WYOxSxYZ
K-IfkD2G163DdWnW2tQg/file [following]
--2020-07-13 14:49:34--  https://uc851c6d51978ea67bc1eb2fdcb4.dl.dropboxus
ercontent.com/cd/0/inline2/A7fo9cLZgNXN7ypgHcI_sbWfHRjY2wg-Vu55fYjhwAXvgJV
FuMoQyBFnit-pft_Q4HYQJ2uGR3y2N9FiaGe3RN6bwFggAEXqtBt-aDzMK-B3C8q60-FUDmgyM
M3Ytwx8a_X6DjvXjMoxW27j5p79d0lJM4sGErl85Nr9nEWbZ7cm74JgYeaZKvK91Uv5vmclFKp
uO6PlkktJ_z8c9fn0E4zHaCXeA8DQc5FOd2MuRccWtJ1elXtD3XvGrmiqJgXz-e7Thweo75o2-
QwXj0u7xibswyM0gmXSqvS7_BjP68ul4ndzgNJoF5JofZaZFzKik9Li9fd0F_w9tKeTY1WYOxS
xYZK-IfkD2G163DdWnW2tQg/file
Reusing existing connection to uc851c6d51978ea67bc1eb2fdcb4.dl.dropboxuser
content.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 9035 (8.8K) [application/vnd.openxmlformats-officedocument.spreads
heetml.sheet]
Saving to: ‘Wine.xlsx’

Wine.xlsx           100%[===================>]   8.82K  --.-KB/s    in 0s

2020-07-13 14:49:35 (179 MB/s) - ‘Wine.xlsx’ saved [9035/9035]
```

Question 1:

Load, analyze,and pre-process the dataset and convert the dataframe into a list of lists - Beginner

In [ ]:

```
import pandas as pd

df = pd.read_excel('/content/Wine.xlsx',header=None)
df.head()
```

Out[ ]:

	0	1	2	3	4
0	Royal Stag	Old Monk	Black Dog	NaN	Kingfisher
1	Old Monk	Royal Stag	NaN	Teachers	NaN
2	NaN	NaN	Kingfisher	NaN	NaN
3	Teachers	NaN	Royal Stag	Jack Daniels	NaN
4	Jack Daniels	Old Monk	NaN	NaN	Royal Stag

From above, you can see that the dataset contains Nan values. That means that item was not purchased in that transaction.

In [ ]:

```
df.describe()
```

Out[ ]:

	0	1	2	3	4
count	13	11	11	11	7
unique	6	6	6	5	4
top	Royal Stag	Old Monk	Old Monk	Old Monk	Kingfisher
freq	6	4	3	3	3

In [ ]:

```
df.shape
```

Out[ ]:

```
(20, 5)
```

Converting the dataframe into a list of lists.

Apriori library requires the dataset to be in the form of a list of lists, where the entire dataset is a big list and each transaction in the dataset is stored as a list inside the bigger list.

In [ ]:

```
dataset=[]
for i in range(0, df.shape[0]):
    dataset.append([str(df.values[i,j]) for j in range(0, df.shape[1])])
```

In [ ]:

```
dataset
```

Out[ ]:

```
[['Royal Stag', 'Old Monk', 'Black Dog', 'nan', 'Kingfisher'],
 ['Old Monk', 'Royal Stag', 'nan', 'Teachers', 'nan'],
 ['nan', 'nan', 'Kingfisher', 'nan', 'nan'],
 ['Teachers', 'nan', 'Royal Stag', 'Jack Daniels', 'nan'],
 ['Jack Daniels', 'Old Monk', 'nan', 'nan', 'Royal Stag'],
 ['nan', 'Royal Stag', 'nan', 'Jack Daniels', 'nan'],
 ['nan', 'nan', 'Old Monk', 'nan', 'nan'],
 ['Royal Stag', 'nan', 'Black Dog', 'nan', 'Johnny Walker'],
 ['nan', 'Jack Daniels', 'nan', 'Royal Stag', 'nan'],
 ['Royal Stag', 'nan', 'nan', 'Old Monk', 'nan'],
 ['nan', 'nan', 'Teachers', 'nan', 'Black Dog'],
 ['Black Dog', 'Teachers', 'nan', 'Jack Daniels', 'nan'],
 ['Royal Stag', 'nan', 'Teachers', 'Black Dog', 'Kingfisher'],
 ['Kingfisher', 'nan', 'Old Monk', 'nan', 'nan'],
 ['nan', 'Jack Daniels', 'nan', 'Teachers', 'Black Dog'],
 ['Teachers', 'Black Dog', 'Kingfisher', 'Old Monk', 'nan'],
 ['Black Dog', 'Old Monk', 'nan', 'nan', 'Kingfisher'],
 ['Royal Stag', 'nan', 'Jack Daniels', 'Old Monk', 'nan'],
 ['nan', 'Kingfisher', 'Old Monk', 'nan', 'nan'],
 ['Royal Stag', 'Old Monk', 'nan', 'Teachers', 'nan']]
```

Question 2:

Implement the Apriori Algorithm from the Apyori package - Intermediate

In [ ]:

```
!pip install apyori
```

```
Requirement already satisfied: apyori in /usr/local/lib/python3.6/dist-pac
kages (1.1.2)
```

In [ ]:

```
import pandas as pd
import numpy as np
from apyori import apriori
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
rules = apriori(dataset, min_support=0.002,min_confidence=0.7,min_lift=3,min_length=1,
use_colnames=True)
result = list(rules)
```

In [ ]:

```
result
```

Out[ ]:

```
[RelationRecord(items=frozenset({'Royal Stag', 'Johnny Walker', 'Black Do
g'}), support=0.05, ordered_statistics=[OrderedStatistic(items_base=frozen
set({'Johnny Walker'}), items_add=frozenset({'Royal Stag', 'Black Dog'}),
confidence=1.0, lift=6.666666666666667)]),
 RelationRecord(items=frozenset({'nan', 'Royal Stag', 'Johnny Walker', 'Bl
ack Dog'}), support=0.05, ordered_statistics=[OrderedStatistic(items_base=
frozenset({'Johnny Walker'}), items_add=frozenset({'Royal Stag', 'nan', 'B
lack Dog'}), confidence=1.0, lift=6.666666666666667), OrderedStatistic(ite
ms_base=frozenset({'nan', 'Johnny Walker'}), items_add=frozenset({'Royal S
tag', 'Black Dog'}), confidence=1.0, lift=6.666666666666667)])]
```

Question 3:

Analyze the first relation obtained from the rules above & derive useful insights - Advanced

In [ ]:

```
result[0]
```

Out[ ]:

```
RelationRecord(items=frozenset({'Royal Stag', 'Johnny Walker', 'Black Do
g'}), support=0.05, ordered_statistics=[OrderedStatistic(items_base=frozen
set({'Johnny Walker'}), items_add=frozenset({'Royal Stag', 'Black Dog'}),
confidence=1.0, lift=6.666666666666667)])
```

From above, you can observe the following for the first rule

- **Confidence =** 1.0 implies that if a customer buys both **Johnny Walker**, we can say with 100% confidence that he/she will also buy **Royal Stag** and **Black Dog**
- **Lift=** 6.67 implies that if a customer buys both **Johnny Walker** he/she is 6.67 times more likely to buy **Royal Stag** and **Black Dog**

Time Series Analysis

- Time Series is a set of observation taken at specified times usually at equal intervals.
- It is used to predict future values based on the previous observed values

Components of Time Series

- **Trend** - Variation in data over a long period of time
- **Seasonality** - Variation in data occurring at regular intervals of time
- **Cyclical Patterns** - Data pattern with incertain timing and fluctuations
- **Irregularity** - Data with short-periodic, non-repeated and unpredictable events

Scenario 1:Shampoo Sales Analysis

Garnier is a mass market cosmetics brand of French cosmetics company L'Oréal which is a French personal care company headquartered in Clichy, Hauts-de-Seine with a registered office in Paris. It is the world's largest cosmetics company. They have collected sales data of their shampoo for a period of 3 years.

In Time Series Forecasting, we use a model to predict future values based on previously observed values. It is a useful method because L'Oréal can use it to predict/forecast future sales of their shampoo.

Problem Statement

A Cosmetic Company called **L'Oréal** has the data of its shampoo sales for a period of 3 years. The data is classified in date/time and the sales data of shampoo.

So, being a Data Scientist you must -

Perform Time Series Analysis on the given dataset

Tasks to be Performed

- Load, analyze, pre-process and visualize the dataset to determine the trend - Beginner
- Check the Stationarity of the dataset using the Rolling Statistics technique and also plot the Rolling Statistics - Intermediate
- Check the Stationarity of the dataset using the Dickey Fuller Test - Intermediate
- Convert Non-Stationary data to Stationary data using the **Log** method and then, check for Stationarity using both the Rolling Statistics and Dickey Fuller Test - Intermediate

Dataset Description

The dataset describes the monthly number of sales of a shampoo over a 3 year period.

The units are a sales count and there are 36 observations. The original dataset is credited to Makridakis, Wheelwright and Hyndman (1998).

Here's a brief description of the dataset:

- **Month** - Month
- **Sales** - Sale of Shampoo

Topics Covered

- Rolling Statistics
- Dickey Fuller Test

In [1]:

```
!wget https://www.dropbox.com/s/pljx6u7zpz9dmqp/shampoo.csv
```

```
--2020-07-17 10:44:46--  https://www.dropbox.com/s/pljx6u7zpz9dmqp/shampo
o.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:100:603
2:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443... conne
cted.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/pljx6u7zpz9dmqp/shampoo.csv [following]
--2020-07-17 10:44:47--  https://www.dropbox.com/s/raw/pljx6u7zpz9dmqp/sha
mpoo.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc303a50f4ff67f07316be7a678e.dl.dropboxusercontent.com/c
d/0/inline/A7t25YRuA3bu7wO3WOqwJJ-vTb4z9etnaC47Gd34z0bxTfmXoPyqcsEWNeOwgvw
13qM29PXfbWY7BecRp-vxCdjivY3CDz4woAVa_c2xgim7ht73Qdqpo7oDYCq8G2571Is/file#
[following]
--2020-07-17 10:44:47--  https://uc303a50f4ff67f07316be7a678e.dl.dropboxus
ercontent.com/cd/0/inline/A7t25YRuA3bu7wO3WOqwJJ-vTb4z9etnaC47Gd34z0bxTfmX
oPyqcsEWNeOwgvw13qM29PXfbWY7BecRp-vxCdjivY3CDz4woAVa_c2xgim7ht73Qdqpo7oDYC
q8G2571Is/file
Resolving uc303a50f4ff67f07316be7a678e.dl.dropboxusercontent.com (uc303a50
f4ff67f07316be7a678e.dl.dropboxusercontent.com)... 162.125.82.15, 2620:10
0:6032:15::a27d:520f
Connecting to uc303a50f4ff67f07316be7a678e.dl.dropboxusercontent.com (uc30
3a50f4ff67f07316be7a678e.dl.dropboxusercontent.com)|162.125.82.15|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 519 [text/plain]
Saving to: ‘shampoo.csv’

shampoo.csv         100%[===================>]     519  --.-KB/s    in 0s

2020-07-17 10:44:48 (46.7 MB/s) - ‘shampoo.csv’ saved [519/519]
```

Question 1:

Load, analyze, pre-process and visualize the dataset to determine the trend - Beginner

In [2]:

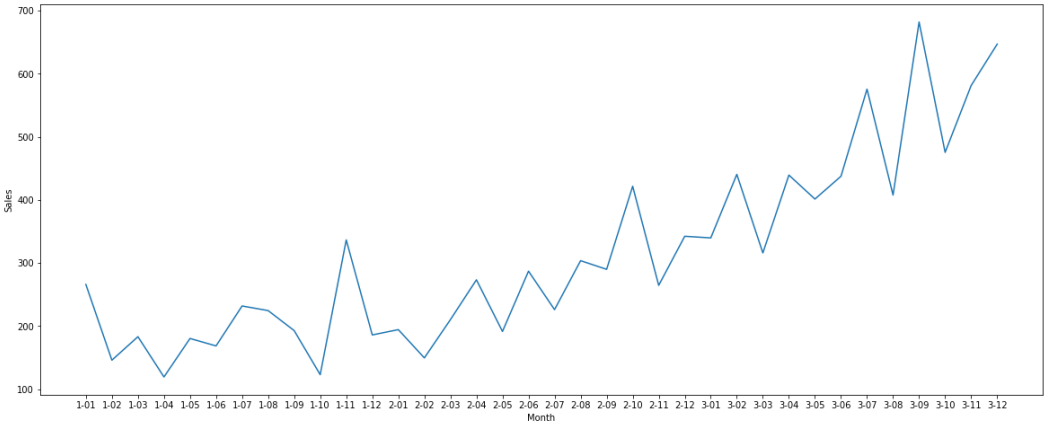
```
import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv('/content/shampoo.csv',parse_dates=True, index_col='Month')
df.head()
```

Out[2]:

Month	Sales
1-01	266.0
1-02	145.9
1-03	183.1
1-04	119.3
1-05	180.3

In [6]:

```
plt.figure(figsize=(20,8))
plt.xlabel('Month')
plt.ylabel('Sales')
plt.plot(df)
plt.show()
```



From above, you can see that the data has an overall increasing trend

Question 2:

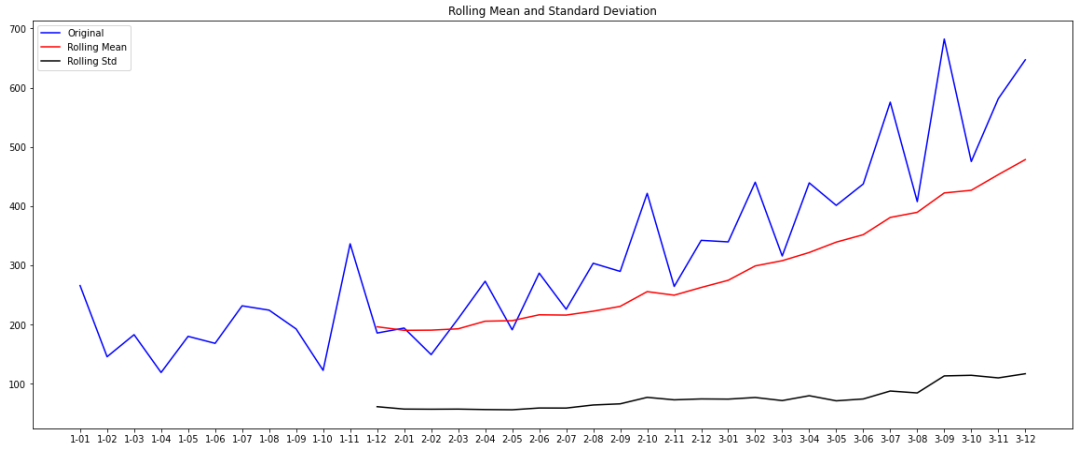
Check the Stationarity of the dataset using the Rolling Statistics technique and also plot the Rolling Statistics - Intermediate

In [9]:

```
#Determine Rolling Statistics

rollmean = df.rolling(window = 12).mean() #Window of 12 Months

rollstd = df.rolling(window = 12).std()

#print(rollmean, rollstd)
```

In [8]:

```
#Plot Rolling Statistics
plt.figure(figsize=(20,8))

orig = plt.plot(df, color = 'blue', label = 'Original')
mean = plt.plot(rollmean, color = 'red', label = 'Rolling Mean')
std = plt.plot(rollstd, color = 'black', label = 'Rolling Std')
plt.legend(loc = 'best')
plt.title("Rolling Mean and Standard Deviation")
plt.show()
```



From above, you can see that both the **Mean** and **Standard Deviation** is not constant. So, our data is not stationary.

Question 3:

Check the Stationarity of the dataset using the Dickey Fuller Test - Intermediate

Null Hypothesis - Data is Not Stationary

Alternate Hypothesis - Data is Stationary

In [10]:

```
from statsmodels.tsa.stattools import adfuller

print("Results of Dickey Fuller Test:")

dftest = adfuller(df['Sales'], autolag = 'AIC') #AIC is Akaike Information Criterion

dfoutput = pd.Series(dftest[0:4], index = ['Test Statistic', 'p-value', '#Lags Used',
'No. of Observations Used'])

for key, value in dftest[4].items():
  dfoutput['Critical Value: %s'%key] = value

print(dfoutput)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: F
utureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm

Results of Dickey Fuller Test:
Test Statistic                3.060142
p-value                       1.000000
#Lags Used                   10.000000
No. of Observations Used     25.000000
Critical Value: 1%           -3.723863
Critical Value: 5%           -2.986489
Critical Value: 10%          -2.632800
dtype: float64
```

From above, you can see that,

- p-value should be always less but, here we have a very large p-value
- Critical Value should be more than the Test Statistic.

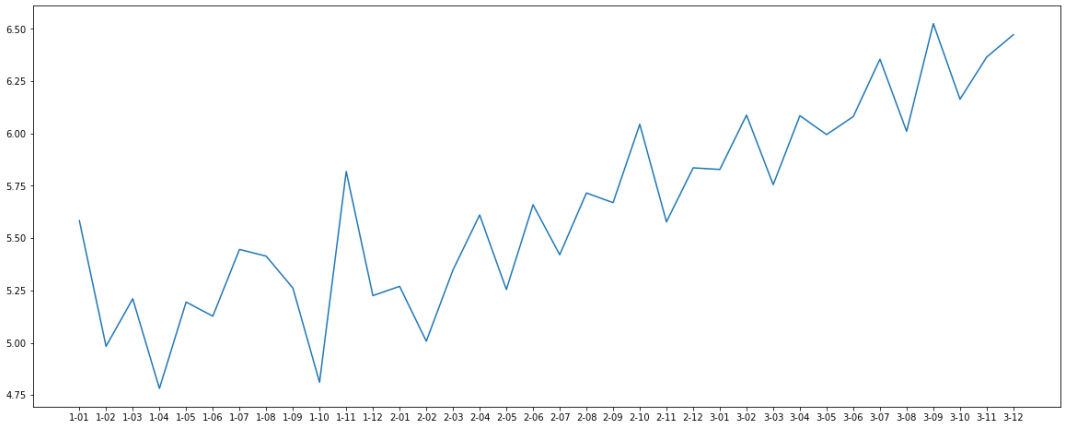
So, here we cannot reject the Null Hypothesis and we can say that the Data is not Stationary.

Question 4:

Convert Non-Stationary data to Stationary data using the **Log** method and then, check for Stationarity using both the Rolling Statistics and Dickey Fuller Test - Intermediate

In [12]:

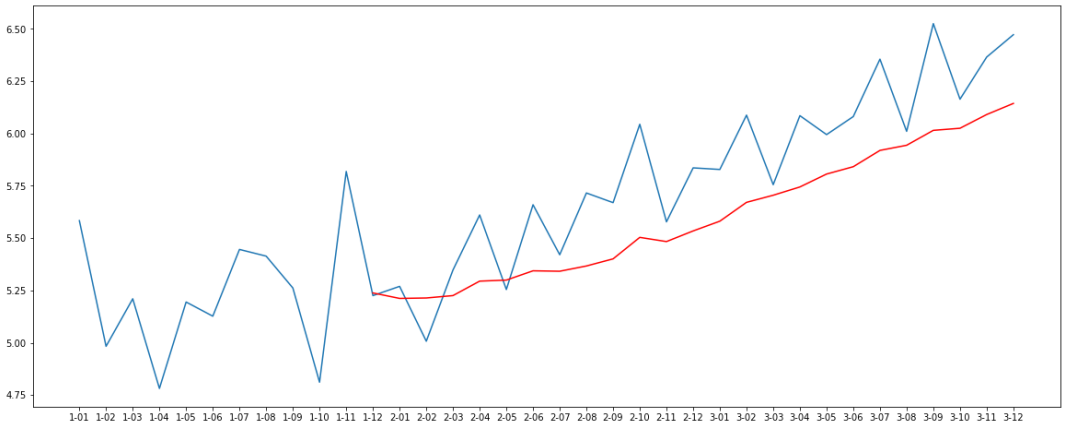
```
plt.figure(figsize=(20,8))
import numpy as np

df_log = np.log(df) #log of the dataset
plt.plot(df_log)
plt.show()
```



From above, you can see that the numbers on y-axis have changed because the scale itself has changed.

In [13]:

```
ma = df_log.rolling(window = 12).mean() #Window of 12 Months

mstd = df_log.rolling(window = 12).std()

plt.figure(figsize=(20,8))

plt.plot(df_log)
plt.plot(ma, color = 'Red')
plt.show()
```



In [14]:

```
df_diff = df_log - ma #Difference between the Moving Average and the Actual Number of P
asssengers

#df_diff.head(12)

df_diff.dropna(inplace = True)
df_diff.head(12)
```

Out[14]:

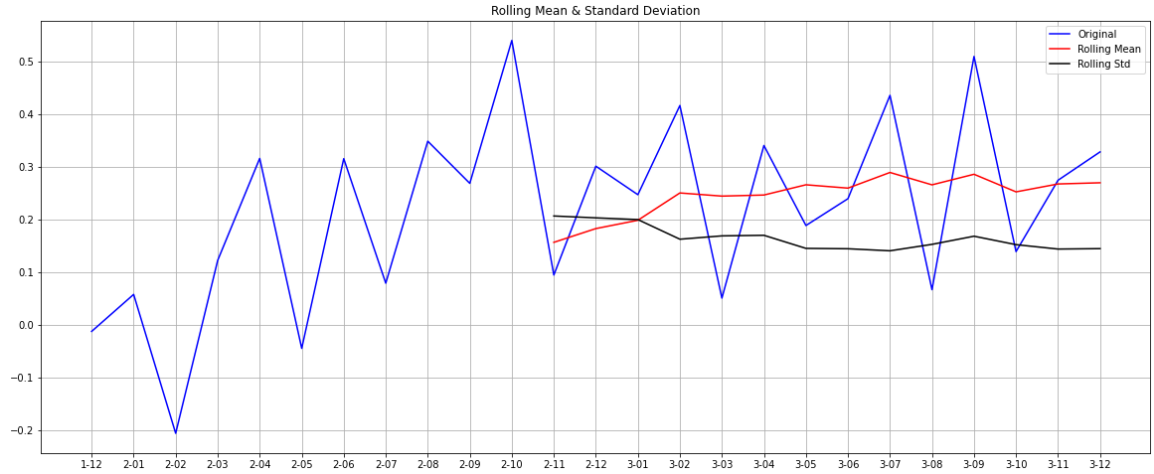
Month	Sales
1-12	-0.012810
2-01	0.057558
2-02	-0.206580
2-03	0.122245
2-04	0.316154
2-05	-0.045029
2-06	0.315709
2-07	0.078873
2-08	0.348896
2-09	0.268731
2-10	0.540528
2-11	0.094375

In [17]:

```
def test_stationarity(timeseries):

    #Determing rolling statistics
    ma = timeseries.rolling(window=12).mean()
    mstd = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    plt.figure(figsize=(20,8))

    #plt.figure(figsize=(12,3))
    plt.grid(True)
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(ma, color='red', label='Rolling Mean')
    std = plt.plot(mstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print("Results of Dickey Fuller Test:")

    dftest = adfuller(timeseries['Sales'], autolag = 'AIC') #AIC is Akaike Information
 Criterion

    dfoutput = pd.Series(dftest[0:4], index = ['Test Statistic', 'p-value', '#Lags Use
d', 'No. of Observations Used'])

    for key, value in dftest[4].items():
      dfoutput['Critical Value: %s'%key] = value

    print(dfoutput)
```

In [18]:

```
test_stationarity(df_diff)
```




```
Results of Dickey Fuller Test:
Test Statistic                -2.866638
p-value                        0.049367
#Lags Used                     8.000000
No. of Observations Used      16.000000
Critical Value: 1%            -3.924019
Critical Value: 5%            -3.068498
Critical Value: 10%           -2.673893
dtype: float64
```

Secanario 2: Forecast Star Air Passenger Traffic

Star Air is a British Airline with its head office in Crawley, England, with some basic data of its passengers across months which is classified in date and the number of passengers travelling per month.

In Time Series Forecasting, we use a model to predict future values based on previously observed values. It is a useful method because Star Air can forecast future passenger traffic and plan flight routes accordingly.

Problem Statement

An Airline called **Star Air** has the data of its passengers across months. The data is classified in date/time and the passengers travelling per month.

So, being an Analyst you must -

Build a model to forecast the demand (passenger traffic) in Airplanes in the future in Python

Tasks to be Performed

- Load, analyze, pre-process and visualize the dataset to determine the trend - Beginner
- Check the Stationarity of the dataset using the Rolling Statistics technique and also plot the Rolling Statistics - Intermediate
- Check the Stationarity of the dataset using the Dickey Fuller Test - Intermediate
- Convert Non-Stationary data to Stationary data using the **Log** method and then, check for Stationarity using both the Rolling Statistics and Dickey Fuller Test - Intermediate
- Plot the Auto-Correlation and Partial Auto-Correlation Graph and find out the **p** & **q** values - Intermediate
- Build the ARIMA Model and forecast the demand - Advanced

Dataset Description

Here's a brief description of Dataset:

- **Month** - Dates from from 1949 till 1960 monthwise
- **#Passengers** - Number of Passengers travelling per month

Topics Covered

- Rolling Statistics
- Dickey Fuller Test
- Auto-correlation Function (ACF) and Partial-autocorrelation function (PACF)
- ARIMA Model

In [19]:

```
!wget https://www.dropbox.com/s/ogl8t4g1wfn8duh/AirPassengers.csv
```

```
--2020-07-17 10:47:54--  https://www.dropbox.com/s/ogl8t4g1wfn8duh/AirPass
engers.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:100:603
2:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443... conne
cted.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/ogl8t4g1wfn8duh/AirPassengers.csv [following]
--2020-07-17 10:47:54--  https://www.dropbox.com/s/raw/ogl8t4g1wfn8duh/Air
Passengers.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucb17a896bfacdcd77e0d6fd3e0c.dl.dropboxusercontent.com/c
d/0/inline/A7tIOiuAW-k9UreZQTIi40R2O8kA_EEmXl5pqRxrwbUXVwvhqRUTu2d04ikX8qC
gTP75EonJcQF2LdcA5ZasBKrLZw9i0AEukS8ShoEUpdGe2SyOqYdHFNdzxZ-JrUeW1C0/file#
[following]
--2020-07-17 10:47:55--  https://ucb17a896bfacdcd77e0d6fd3e0c.dl.dropboxus
ercontent.com/cd/0/inline/A7tIOiuAW-k9UreZQTIi40R2O8kA_EEmXl5pqRxrwbUXVwvh
qRUTu2d04ikX8qCgTP75EonJcQF2LdcA5ZasBKrLZw9i0AEukS8ShoEUpdGe2SyOqYdHFNdzxZ
-JrUeW1C0/file
Resolving ucb17a896bfacdcd77e0d6fd3e0c.dl.dropboxusercontent.com (ucb17a89
6bfacdcd77e0d6fd3e0c.dl.dropboxusercontent.com)... 162.125.82.15, 2620:10
0:6032:15::a27d:520f
Connecting to ucb17a896bfacdcd77e0d6fd3e0c.dl.dropboxusercontent.com (ucb1
7a896bfacdcd77e0d6fd3e0c.dl.dropboxusercontent.com)|162.125.82.15|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 1746 (1.7K) [text/plain]
Saving to: ‘AirPassengers.csv’

AirPassengers.csv   100%[===================>]   1.71K  --.-KB/s    in 0s

2020-07-17 10:47:55 (198 MB/s) - ‘AirPassengers.csv’ saved [1746/1746]
```

Question 1:

Load, analyze, pre-process and visualize the dataset to determine the trend - Beginner

In [20]:

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

df=pd.read_csv('/content/AirPassengers.csv', parse_dates=True, index_col='Month')
df.head()
```

Out[20]:

Month	#Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

In [21]:

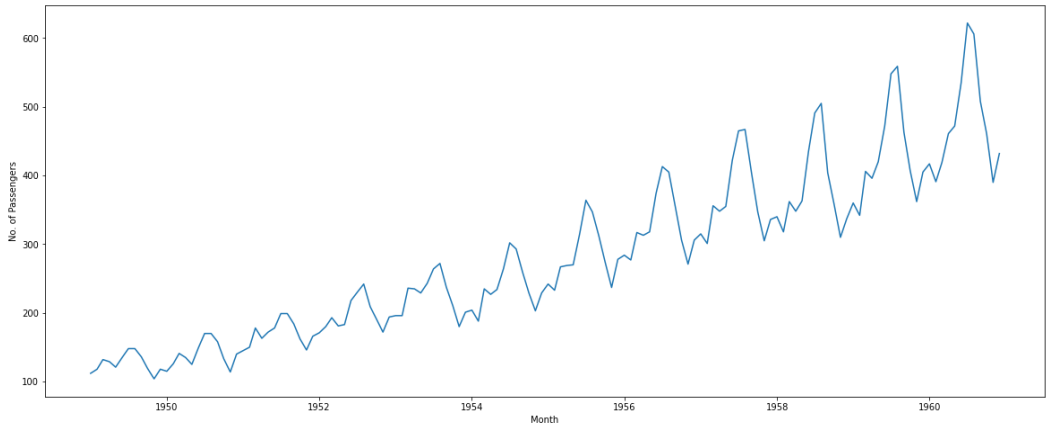
```
df.shape
```

Out[21]:

```
(144, 1)
```

In [22]:

```
plt.figure(figsize=(20,8))

plt.xlabel('Month')
plt.ylabel('No. of Passengers')
plt.plot(df)
plt.show()
```



From above, you can see that there is an **overall increasing trend** in the dataset with some variations.

Question 2:

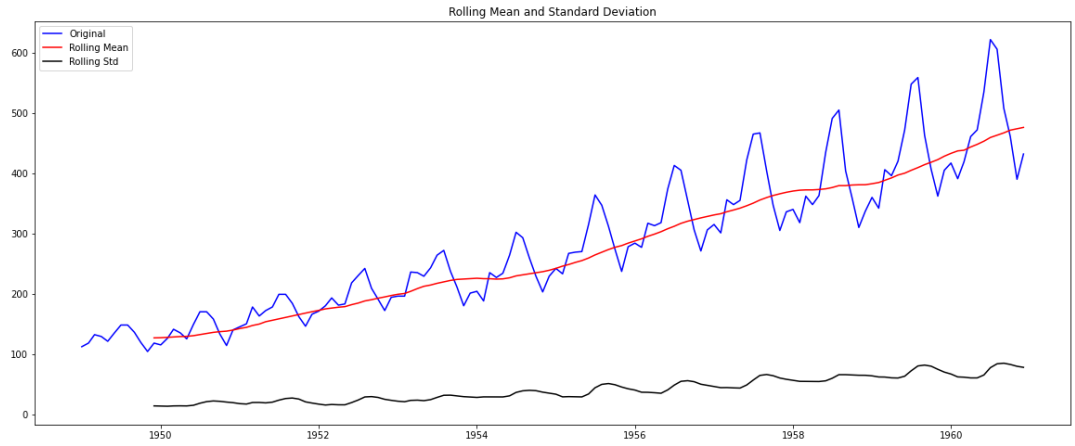
Check the Stationarity of the dataset using the Rolling Statistics technique and also plot the Rolling Statistics - Intermediate

In [24]:

```
#Determine Rolling Statistics

rollmean = df.rolling(window = 12).mean() #Window of 12 Months

rollstd = df.rolling(window = 12).std()

#print(rollmean, rollstd)
```

In [25]:

```
#Plot Rolling Statistics
plt.figure(figsize=(20,8))

orig = plt.plot(df, color = 'blue', label = 'Original')
mean = plt.plot(rollmean, color = 'red', label = 'Rolling Mean')
std = plt.plot(rollstd, color = 'black', label = 'Rolling Std')
plt.legend(loc = 'best')
plt.title("Rolling Mean and Standard Deviation")
plt.show()
```



From above, you can see that both the **Mean** and **Standard Deviation** is not constant. So, our data is not stationary.

Question 3:

Check the Stationarity of the dataset using the Dickey Fuller Test - Intermediate

Null Hypothesis - Data is Not Stationary

Alternate Hypothesis - Data is Stationary

In [26]:

```
from statsmodels.tsa.stattools import adfuller

print("Results of Dickey Fuller Test:")

dftest = adfuller(df['#Passengers'], autolag = 'AIC') #AIC is Akaike Information Criterion

dfoutput = pd.Series(dftest[0:4], index = ['Test Statistic', 'p-value', '#Lags Used', 'No. of Observations Used'])

for key, value in dftest[4].items():
  dfoutput['Critical Value: %s'%key] = value

print(dfoutput)
```

```
Results of Dickey Fuller Test:
Test Statistic                  0.815369
p-value                         0.991880
#Lags Used                     13.000000
No. of Observations Used      130.000000
Critical Value: 1%             -3.481682
Critical Value: 5%             -2.884042
Critical Value: 10%            -2.578770
dtype: float64
```

From above, you can see that,

- p-value should be always less but, here we have a very large p-value i.e., 0.9
- Critical Value should be more than the Test Statistic.

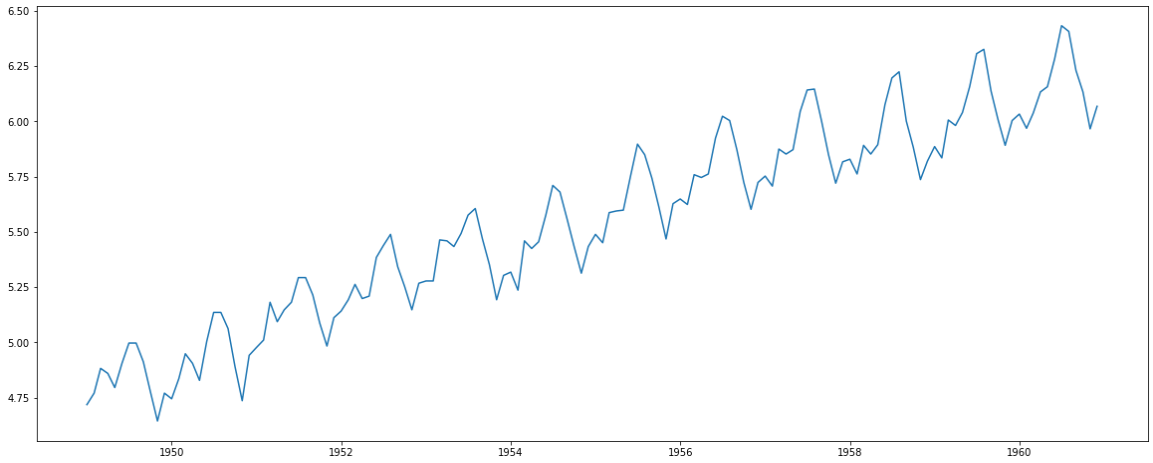
So, here we cannot reject the Null Hypothesis and we can say that the Data is not Stationary.

Question 4:

Convert Non-Stationary data to Stationary data using the **Log** method and then, check for Stationarity using both the Rolling Statistics and Dickey Fuller Test - Intermediate

In [28]:

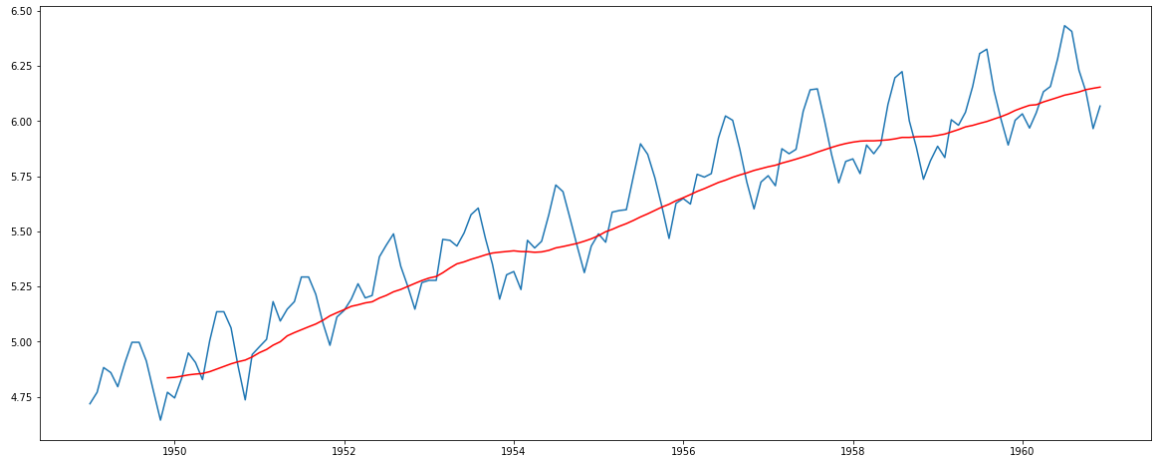
```
plt.figure(figsize=(20,8))

df_log = np.log(df) #log of the dataset
plt.plot(df_log)
plt.show()
```



From above, you can see that the numbers on y-axis have changed because the scale itself has changed.

In [29]:

```
ma = df_log.rolling(window = 12).mean() #Window of 12 Months

mstd = df_log.rolling(window = 12).std()

plt.figure(figsize=(20,8))

plt.plot(df_log)
plt.plot(ma, color = 'Red')
plt.show()
```



From above, you can see that the Mean is not Stationary but, it is quite better than the previous one

In [30]:

```
df_diff = df_log - ma #Difference between the Moving Average and the Actual Number of Passsengers

#df_diff.head(12)

df_diff.dropna(inplace = True)
df_diff.head(12)
```

Out[30]:

Month	#Passengers
1949-12-01	-0.065494
1950-01-01	-0.093449
1950-02-01	-0.007566
1950-03-01	0.099416
1950-04-01	0.052142
1950-05-01	-0.027529
1950-06-01	0.139881
1950-07-01	0.260184
1950-08-01	0.248635
1950-09-01	0.162937
1950-10-01	-0.018578
1950-11-01	-0.180379

In [31]:

```
def test_stationarity(timeseries):

    #Determing rolling statistics
    ma = timeseries.rolling(window=12).mean()
    mstd = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    plt.figure(figsize=(20,8))

    #plt.figure(figsize=(12,3))
    plt.grid(True)
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(ma, color='red', label='Rolling Mean')
    std = plt.plot(mstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print("Results of Dickey Fuller Test:")

    dftest = adfuller(timeseries['#Passengers'], autolag = 'AIC') #AIC is Akaike Inform
ation Criterion

    dfoutput = pd.Series(dftest[0:4], index = ['Test Statistic', 'p-value', '#Lags Use
d', 'No. of Observations Used'])

    for key, value in dftest[4].items():
      dfoutput['Critical Value: %s'%key] = value

    print(dfoutput)
```

In [32]:

```
test_stationarity(df_diff)
```


```
Results of Dickey Fuller Test:
Test Statistic                -3.162908
p-value                        0.022235
#Lags Used                    13.000000
No. of Observations Used     119.000000
Critical Value: 1%            -3.486535
Critical Value: 5%            -2.886151
Critical Value: 10%           -2.579896
dtype: float64
```

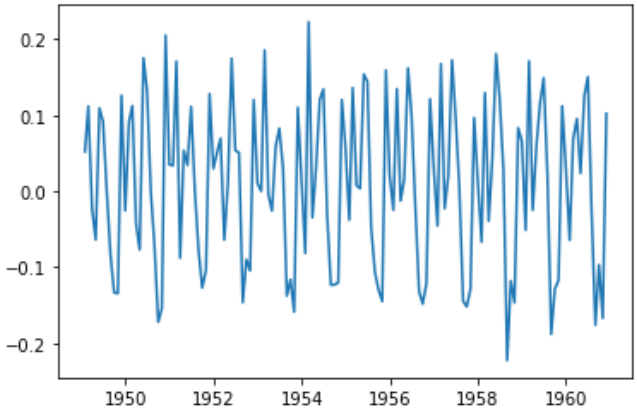
From above, you can see that the **p-value** is relatively less and also the Critical Value and Test Statistic value is almost equal which helps us to determine whether the data is Stationary or not.

Question 5:

Plot the Auto-Correlation and Partial Auto-Correlation Graph and find out the **p** & **q** values - Intermediate

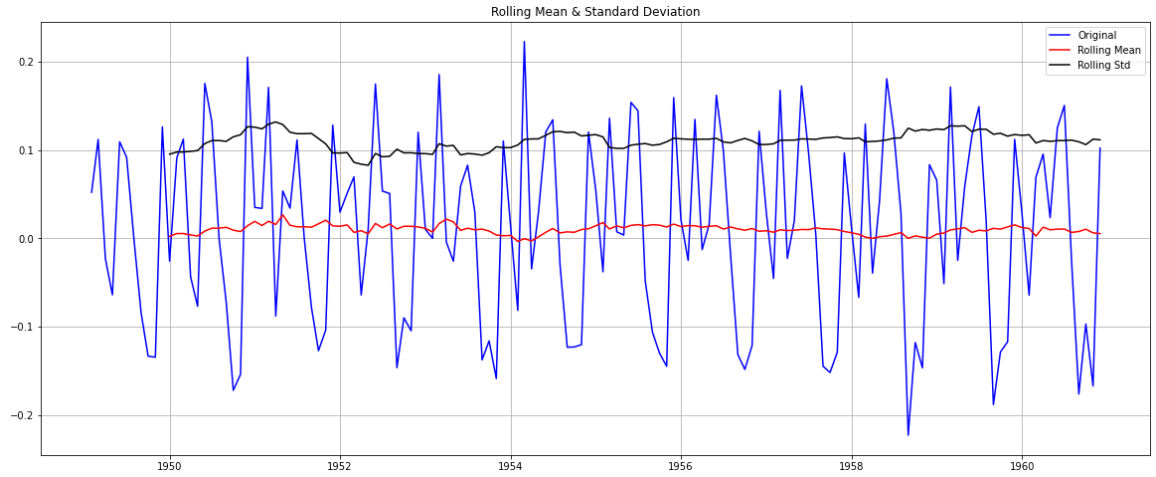
In [33]:

```
df_log_shift = df_log - df_log.shift()

plt.plot(df_log_shift)
plt.show()
```



In [34]:

```
df_log_shift.dropna(inplace = True)
test_stationarity(df_log_shift)
```



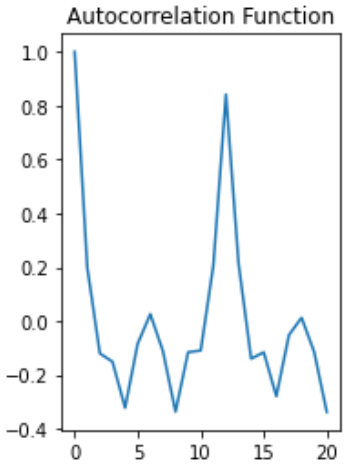

```
Results of Dickey Fuller Test:
Test Statistic                -2.717131
p-value                        0.071121
#Lags Used                    14.000000
No. of Observations Used     128.000000
Critical Value: 1%            -3.482501
Critical Value: 5%            -2.884398
Critical Value: 10%           -2.578960
dtype: float64
```

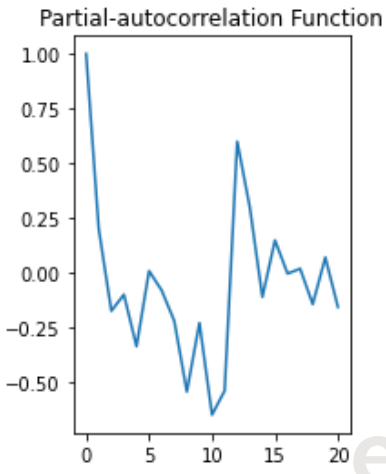
From above, you can see that the time series is Stationary

In [35]:

```
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(df_log_shift, nlags = 20)
lag_pacf = pacf(df_log_shift, nlags = 20, method = 'ols')

#Plot ACF
plt.subplot(121)
plt.plot(lag_acf)
plt.title("Autocorrelation Function")
plt.show()

#Plot PACF
plt.subplot(122)
plt.plot(lag_pacf)
plt.title("Partial-autocorrelation Function")
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/stattools.py:541: F
utureWarning: fft=True will become the default in a future version of stat
smodels. To suppress this warning, explicitly set fft=False.
  warnings.warn(msg, FutureWarning)
```





Now, in order to calculate the **p-value** and **q-value**, you need to check what is the value where the graph drops to 0 for the first time. Therefore, p = 2 & q = 2.

Question 6:

Build the ARIMA Model and forecast the demand - Advanced

In [36]:

```
from statsmodels.tsa.arima_model import ARIMA

#AR Model
model = ARIMA(df_log, order = (2, 1, 2))
results_AR = model.fit()

results_AR.summary()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:1
65: ValueWarning: No frequency information was provided, so inferred frequ
ency MS will be used.
  % freq, ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:1
65: ValueWarning: No frequency information was provided, so inferred frequ
ency MS will be used.
  % freq, ValueWarning)
```

Out[36]:

ARIMA Model Results

Dep. Variable:	D.#Passengers	**No. Observations:**	143
Model:	ARIMA(2, 1, 2)	**Log Likelihood**	149.640
Method:	css-mle	**S.D. of innovations**	0.084
Date:	Fri, 17 Jul 2020	**AIC**	-287.281
Time:	10:49:47	**BIC**	-269.504
Sample:	02-01-1949	**HQIC**	-280.057
	- 12-01-1960		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0096	0.003	3.697	0.000	0.005	0.015
ar.L1.D.#Passengers	1.6293	0.039	41.868	0.000	1.553	1.706
ar.L2.D.#Passengers	-0.8946	0.039	-23.127	0.000	-0.970	-0.819
ma.L1.D.#Passengers	-1.8270	0.036	-51.303	0.000	-1.897	-1.757
ma.L2.D.#Passengers	0.9245	0.036	25.568	0.000	0.854	0.995

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.9106	-0.5372j	1.0573	-0.0848
AR.2	0.9106	+0.5372j	1.0573	0.0848
MA.1	0.9881	-0.3245j	1.0400	-0.0505
MA.2	0.9881	+0.3245j	1.0400	0.0505

In [37]:

```
# Analyse residual errors. They should be normal
residuals = pd.DataFrame(results_AR.resid)
print(residuals.describe())
```

```
                0
count  143.000000
mean     0.001691
std      0.085116
min     -0.193387
25%     -0.063327
50%     -0.005020
75%      0.074605
max      0.210671
```

In [39]:

```
#results_AR.forecast(steps=120)
```

In [41]:

```
predictions = []
for x in np.arange(-5, 0):
    model = ARIMA(df_log.iloc[:x], order=(2, 1, 2))
    results_AR = model.fit()
    predictions.append(results_AR.forecast()[0])
predictions=np.array(predictions).ravel()
predictions
```

Out[41]:

```
array([6.39078517, 6.34745416, 6.20273937, 6.181387  , 6.00689881])
```

In [42]:

```
actual = df_log.tail()
actual
```

Out[42]:

Month	#Passengers
1960-08-01	6.406880
1960-09-01	6.230481
1960-10-01	6.133398
1960-11-01	5.966147
1960-12-01	6.068426

In [43]:

```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(actual, predictions)
print('Test MSE: %.6f' % error)
```

```
Test MSE: 0.013773
```

In [44]:

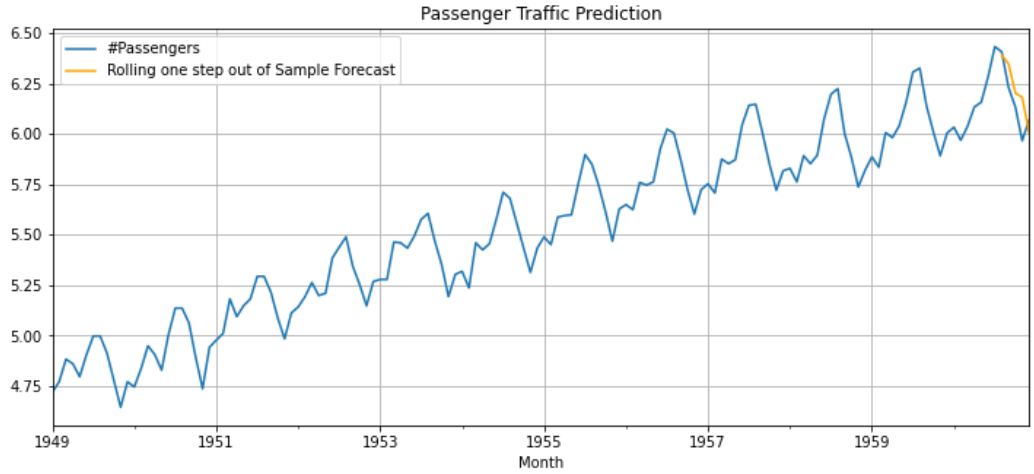
```
predictions_series = pd.Series(predictions, index = actual.index)
predictions_series
```

Out[44]:

```
Month
1960-08-01    6.390785
1960-09-01    6.347454
1960-10-01    6.202739
1960-11-01    6.181387
1960-12-01    6.006899
dtype: float64
```

In [45]:

```
#Future Forecasting Using the ARIMA Model

df_log.plot(figsize=(12, 5), grid=True, label= 'observed')
predictions_series.plot(c='orange', figsize=(12, 5), grid=True,label = 'Rolling one ste
p out of Sample Forecast')
plt.title('Passenger Traffic Prediction')
plt.legend()
plt.show()
```



Scenario 3: Minimum Daily Temperature Analysis

The **Bureau of Meteorology**(Australian Weather Department) is an Executive Agency of the **Australian Government** responsible for providing weather services to Australia and surrounding areas with some basic data of Daily Minimum Temperatures over 10 years i.e. from 1981 - 1990 in the city of Melbourne, Australia.

In Time Series Forecasting, we use a model to predict future values based on previously observed values. It is a useful method because the Australian Weather Department can use it predict/forecaset future weather conditions in the city of Melbourne.

Problem Statement

The Bureau of Metrology has the data of minimum daily temperatures of the city of Melbourne, Australia. The data is classified in two columns i.e., date and the minimum tempearture.

So, being a Data Scientist you must -

Build a model to forecast the minimum daily temperatures in the future.

Tasks to be performed

- Load, analyze, pre-process and visualize the dataset to determine the trend - Beginner
- Check the Stationarity of the dataset using the Rolling Statistics technique and also plot the Rolling Statistics - Intermediate
- Check the Stationarity of the dataset using the Dickey Fuller Test - Intermediate
- Plot the Auto-Correlation and Partial Auto-Correlation Graph - Intermediate
- Build the ARIMA Model and forecast the demand - Advanced

Dataset Description

This dataset describes the minimum daily temperatures over 10 years (1981-1990) in the city Melbourne, Australia.

The units are in degrees Celsius and there are 3650 observations. The source of the data is credited as the Australian Bureau of Meteorology.

Here's a brief description of the dataset:

- **Date** - A Date
- **Temp** - Minimum Temperature

Topics Covered

- Rolling Statistics
- Dickey Fuller Test
- Auto-correlation Function (ACF) and Partial-autocorrelation function (PACF)
- ARIMA Model

In [46]:

```
!wget https://www.dropbox.com/s/3tvhg63inpw052u/daily-min-temperatures.csv
```

```
--2020-07-17 10:51:09--  https://www.dropbox.com/s/3tvhg63inpw052u/daily-m
in-temperatures.csv
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:100:603
2:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443... conne
cted.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/3tvhg63inpw052u/daily-min-temperatures.csv [following]
--2020-07-17 10:51:09--  https://www.dropbox.com/s/raw/3tvhg63inpw052u/dai
ly-min-temperatures.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc6d24d95f5720c09dda3cfdef1e.dl.dropboxusercontent.com/c
d/0/inline/A7sXMoGUNJYC5nScN6OdKcff9fAgemylkuoAcjgTFwVxUMYrjdR6xekUoN2s_CW
avEl1kOTTvJBheZQSNW6GAudI7etsuAew0a1le7nzUYbjedPQJaD61IdURW16Em2YPf4/file#
[following]
--2020-07-17 10:51:10--  https://uc6d24d95f5720c09dda3cfdef1e.dl.dropboxus
ercontent.com/cd/0/inline/A7sXMoGUNJYC5nScN6OdKcff9fAgemylkuoAcjgTFwVxUMYr
jdR6xekUoN2s_CWavEl1kOTTvJBheZQSNW6GAudI7etsuAew0a1le7nzUYbjedPQJaD61IdURW
16Em2YPf4/file
Resolving uc6d24d95f5720c09dda3cfdef1e.dl.dropboxusercontent.com (uc6d24d9
5f5720c09dda3cfdef1e.dl.dropboxusercontent.com)... 162.125.82.15, 2620:10
0:6032:15::a27d:520f
Connecting to uc6d24d95f5720c09dda3cfdef1e.dl.dropboxusercontent.com (uc6d
24d95f5720c09dda3cfdef1e.dl.dropboxusercontent.com)|162.125.82.15|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 67921 (66K) [text/plain]
Saving to: ‘daily-min-temperatures.csv’

daily-min-temperatu 100%[===================>]  66.33K  --.-KB/s    in 0.0
3s

2020-07-17 10:51:10 (2.37 MB/s) - ‘daily-min-temperatures.csv’ saved [6792
1/67921]
```

Question 1:

Load, analyze, pre-process and visualize the dataset to determine the trend - Beginner

In [47]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 10,6
```

In [48]:

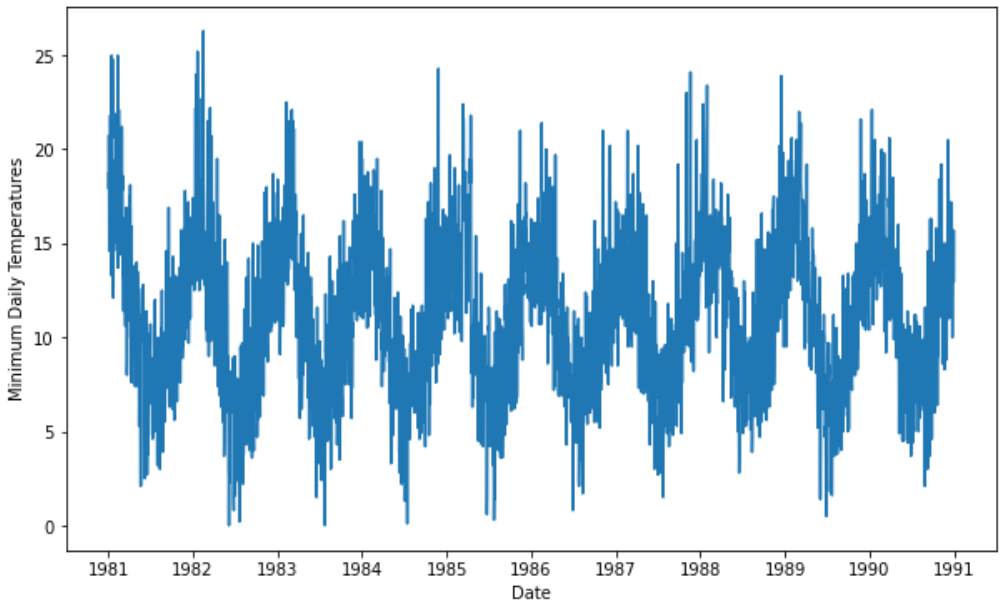
```
df=pd.read_csv('/content/daily-min-temperatures.csv', parse_dates=True, index_col='Dat
e')
df.head()
```

Out[48]:

Date	Temp
1981-01-01	20.7
1981-01-02	17.9
1981-01-03	18.8
1981-01-04	14.6
1981-01-05	15.8

In [50]:

```
plt.xlabel('Date')
plt.ylabel('Minimum Daily Temperatures')
plt.plot(df)
plt.show()
```



From above, you can see that the data has no particular trend. It is Stationary.

Question 2:

Check the Stationarity of the dataset using the Rolling Statistics technique and also plot the Rolling Statistics - Intermediate

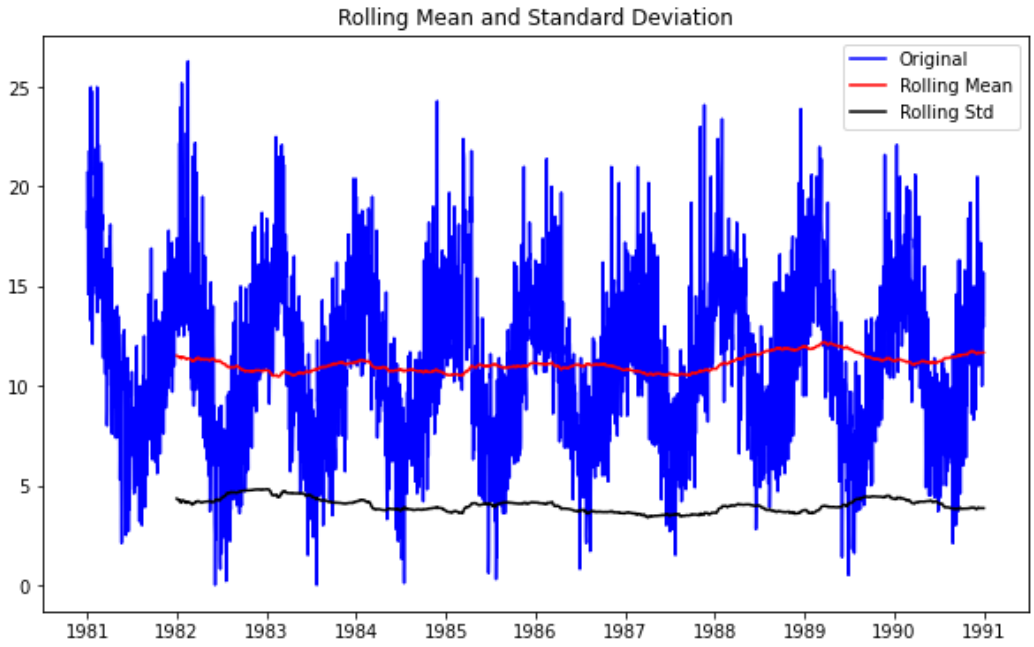
In [51]:

```
#Determine Rolling Statistics

rollmean = df.rolling(window = 365).mean() #Rolling Mean at daily level

rollstd = df.rolling(window = 365).std()

print(rollmean, rollstd)
```

```
                Temp
Date
1981-01-01       NaN
1981-01-02       NaN
1981-01-03       NaN
1981-01-04       NaN
1981-01-05       NaN
...              ...
1990-12-27  11.651507
1990-12-28  11.656712
1990-12-29  11.665205
1990-12-30  11.668767
1990-12-31  11.669589

[3650 rows x 1 columns]                Temp
Date
1981-01-01       NaN
1981-01-02       NaN
1981-01-03       NaN
1981-01-04       NaN
1981-01-05       NaN
...              ...
1990-12-27  3.856232
1990-12-28  3.857580
1990-12-29  3.858218
1990-12-30  3.861348
1990-12-31  3.861600

[3650 rows x 1 columns]
```

In [52]:

```
#Plot Rolling Statistics

orig = plt.plot(df, color = 'blue', label = 'Original')
mean = plt.plot(rollmean, color = 'red', label = 'Rolling Mean')
std = plt.plot(rollstd, color = 'black', label = 'Rolling Std')
plt.legend(loc = 'best')
plt.title("Rolling Mean and Standard Deviation")
plt.show()
```



From above, you can see that the **Rolling Mean** and **Standard Deviation** is constant. That means the data set is stationary.

Question 3:

Check the Stationarity of the dataset using the Dickey Fuller Test - Intermediate

Null Hypothesis - Minimum Daily Temperatures Dataset is Non-Stationary

Alternate Hypothesis - Minimum Daily Temperatures Dataset is Stationary

In [53]:

```
from statsmodels.tsa.stattools import adfuller

print("Results of Dickey Fuller Test:")

dftest = adfuller(df['Temp'], autolag = 'AIC')

dfoutput = pd.Series(dftest[0:4], index = ['Test Statistic', 'p-value', '#Lags Used',
'No. of Observations Used'])

for key, value in dftest[4].items():
  dfoutput['Critical Value: %s'%key] = value

print(dfoutput)
```

```
Results of Dickey Fuller Test:
Test Statistic                -4.444805
p-value                        0.000247
#Lags Used                    20.000000
No. of Observations Used    3629.000000
Critical Value: 1%            -3.432153
Critical Value: 5%            -2.862337
Critical Value: 10%           -2.567194
dtype: float64
```

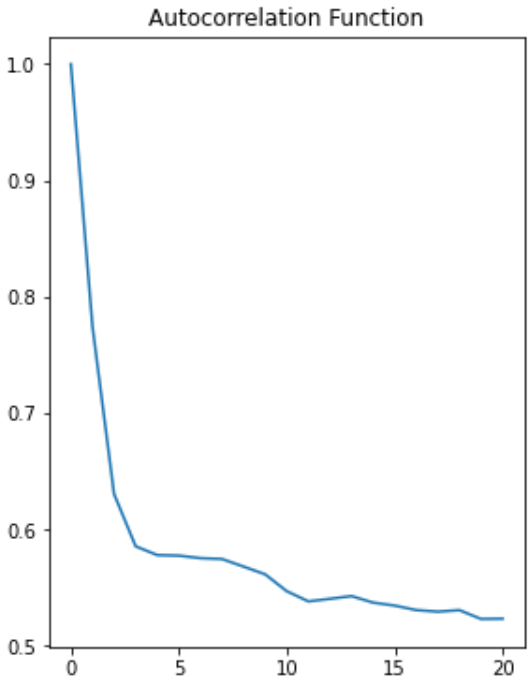
From above, you can see that the p-value is very less i.e. We reject the **Null Hypothesis** and conclude that the data is **Stationary**

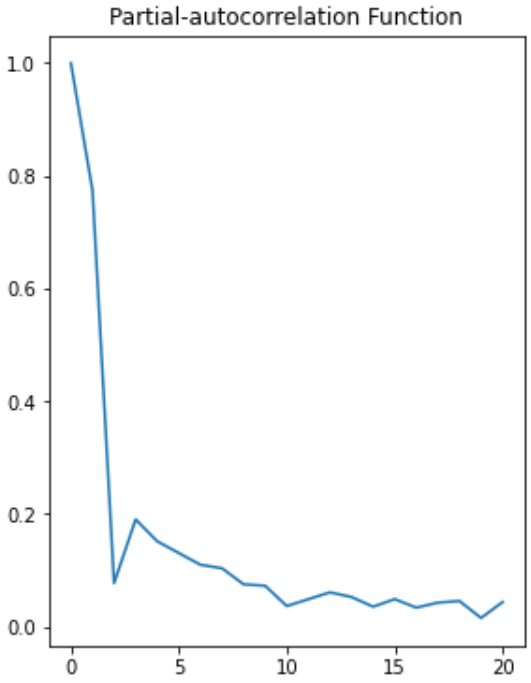
Question 4:

Plot the Auto-Correlation and Partial Auto-Correlation Graph - Intermediate

In [54]:

```
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(df, nlags = 20)
lag_pacf = pacf(df, nlags = 20, method = 'ols')

#Plot ACF
plt.subplot(121)
plt.plot(lag_acf)
plt.title("Autocorrelation Function")
plt.show()

#Plot PACF
plt.subplot(122)
plt.plot(lag_pacf)
plt.title("Partial-autocorrelation Function")
plt.show()
```





Question 5:

Build the ARIMA Model and forecast the demand - Advanced

In [55]:

```
from statsmodels.tsa.arima_model import ARIMA

#AR Model
model = ARIMA(df, order = (2, 1, 2))
results_AR = model.fit()

results_AR.summary()
```

Out[55]:

ARIMA Model Results

Dep. Variable:	D.Temp	**No. Observations:**	3649
Model:	ARIMA(2, 1, 2)	**Log Likelihood**	-8386.371
Method:	css-mle	**S.D. of innovations**	2.409
Date:	Fri, 17 Jul 2020	**AIC**	16784.742
Time:	10:51:54	**BIC**	16821.955
Sample:	1	**HQIC**	16797.995

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0012	0.006	-0.193	0.847	-0.013	0.011
ar.L1.D.Temp	0.4853	0.139	3.490	0.000	0.213	0.758
ar.L2.D.Temp	-0.1243	0.067	-1.851	0.064	-0.256	0.007
ma.L1.D.Temp	-0.8896	0.140	-6.351	0.000	-1.164	-0.615
ma.L2.D.Temp	-0.0104	0.129	-0.080	0.936	-0.264	0.243

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.9521	-2.0577j	2.8364	-0.1292
AR.2	1.9521	+2.0577j	2.8364	0.1292
MA.1	1.1097	+0.0000j	1.1097	0.0000
MA.2	-86.7975	+0.0000j	86.7975	0.5000

In [56]:

```
# Analyse residual errors. They should be normal
residuals = pd.DataFrame(results_AR.resid)
print(residuals.describe())
```

```
                0
count  3649.000000
mean     -0.001378
std       2.409638
min      -7.402870
25%      -1.595123
50%       0.045428
75%       1.536410
max      10.442989
```

In [58]:

```
#results_AR.forecast(steps = 100)
```

In [59]:

```
predictions = []
for x in np.arange(-5, 0):
    model = ARIMA(df.iloc[:x], order=(2, 1, 2))
    results_AR = model.fit()
    predictions.append(results_AR.forecast()[0])
predictions=np.array(predictions).ravel()
predictions
```

Out[59]:

```
array([14.27878895, 13.7351572 , 13.60272146, 13.59319607, 14.90716914])
```

In [60]:

```
actual = df.tail()
actual
```

Out[60]:

	Temp
Date	
1990-12-27	14.0
1990-12-28	13.6
1990-12-29	13.5
1990-12-30	15.7
1990-12-31	13.0

In [61]:

```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(actual, predictions)
print('Test MSE: %.6f' % error)
```

```
Test MSE: 1.636492
```

In [62]:

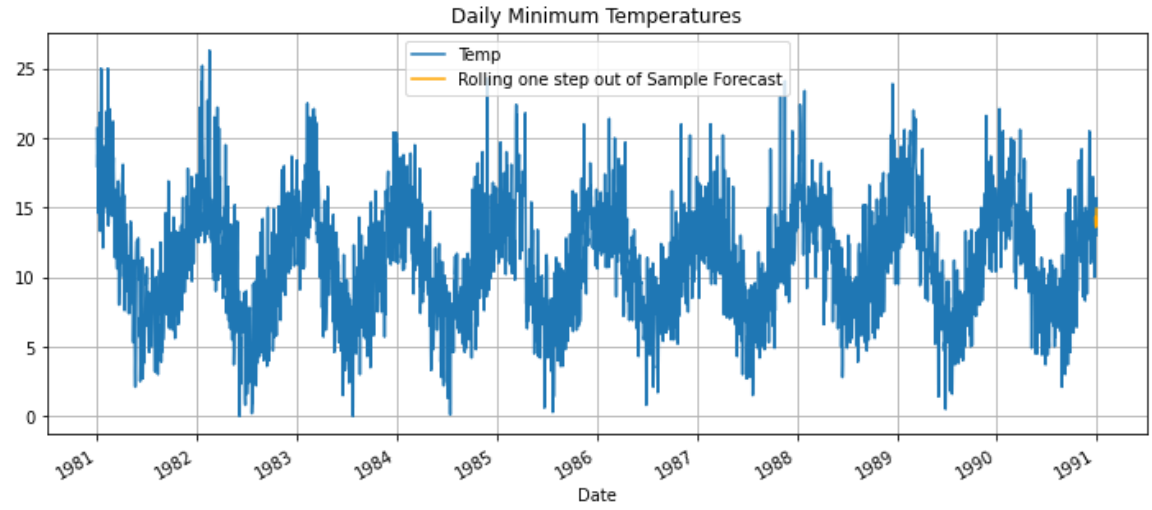
```
predictions_series = pd.Series(predictions, index = actual.index)
predictions_series
```

Out[62]:

```
Date
1990-12-27    14.278789
1990-12-28    13.735157
1990-12-29    13.602721
1990-12-30    13.593196
1990-12-31    14.907169
dtype: float64
```

In [63]:

```
#Forecasting Using the ARIMA Model

df.plot(figsize=(20, 15), grid=True, label= 'observed')
predictions_series.plot(c='orange', figsize=(12, 5), grid=True,label = 'Rolling one ste
p out of Sample Forecast')
plt.title('Daily Minimum Temperatures')
plt.legend()
plt.show()
```



In [ ]: