```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
```

```
fileName = '/content/drive/MyDrive/gsk_excel.xlsx'
```

```
df= pd.read_excel(fileName)
```

```
df.head()
```

	participant_id	age	sex	weight	height	trt_grp	BMI	protein_concentrat:
0	SUBJ_001	46	Female	84.66	1.59	DRUG	33.487599	14
1	SUBJ_002	47	Female	71.21	1.64	DRUG	26.476056	8
2	SUBJ_003	48	Female	69.85	1.73	CONTROL	23.338568	18
3	SUBJ_004	59	Female	62.94	1.50	DRUG	27.973333	8
4	SUBJ_005	59	Female	113.91	1.63	CONTROL	42.873273	13

```
df.dtypes
```

```
participant_id            object
age                        int64
sex                       object
weight                   float64
height                   float64
trt_grp                   object
BMI                      float64
protein_concentration    float64
RESPONSE                  object
dtype: object
```

```
df1=df
```

```
# data transformation to build the model
# this function takes a dataframe an loops through it to print columns of type object
def print_unique_col_values(df):
  for column in df:
    if df[column].dtypes =='object':
      print(f'{column}: {df[column].unique()}')
```

```
#Removing participant_id
df1 = df1.drop('participant_id', axis=1)
```

```
print_unique_col_values(df1)
```

```
sex: ['Female' 'Male']
trt_grp: ['DRUG' 'CONTROL']
RESPONSE: ['N' 'Y']
```

```
#Extracting record for only miraculon-B

miraculonB_df = df1[df1['trt_grp']=='DRUG']
miraculonB_df
```

	age	sex	weight	height	trt_grp	BMI	protein_concentration	RESPONSE
0	46	Female	84.66	1.59	DRUG	33.487599	148.0	N
1	47	Female	71.21	1.64	DRUG	26.476056	85.0	Y
3	59	Female	62.94	1.50	DRUG	27.973333	89.0	Y
7	57	Male	93.50	1.63	DRUG	35.191388	115.0	N
8	72	Male	85.57	1.68	DRUG	30.318169	197.0	N
...	...	...	...	...	...	...	...	...
762	70	Female	62.21	1.66	DRUG	22.575846	89.0	Y
764	65	Male	112.86	1.76	DRUG	36.434659	122.0	N
765	60	Male	81.03	1.77	DRUG	25.864215	121.0	N
766	53	Male	88.67	1.72	DRUG	29.972282	126.0	Y
767	68	Female	80.29	1.63	DRUG	30.219429	93.0	Y

383 rows × 8 columns

```
# Dropping column containing sngle value
miraculonB_df=miraculonB_df.drop('trt_grp', axis=1)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-37-80ed898e9bcd> in <cell line: 2>()
      1 # Dropping column containing sngle value
----> 2 miraculonB_df=miraculonB_df.drop('trt_grp', axis=1)
      3 
      4 miraculonB_df

5 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in drop(self,
labels, errors)
   6932         if mask.any():
   6933             if errors != "ignore":
-> 6934                 raise KeyError(f"{list(labels[mask])} not found in axis")
   6935         indexer = indexer[~mask]
   6936         return self.delete(indexer)

KeyError: "['trt_grp'] not found in axis"
```

SEARCH STACK OVERFLOW

```
miraculonB_df
```

	age	sex	weight	height	BMI	protein_concentration	RESPONSE
0	46	Female	84.66	1.59	33.487599	148.0	N
1	47	Female	71.21	1.64	26.476056	85.0	Y
3	59	Female	62.94	1.50	27.973333	89.0	Y
7	57	Male	93.50	1.63	35.191388	115.0	N
8	72	Male	85.57	1.68	30.318169	197.0	N
...	...	...	...	...	...	...	...
762	70	Female	62.21	1.66	22.575846	89.0	Y
764	65	Male	112.86	1.76	36.434659	122.0	N
765	60	Male	81.03	1.77	25.864215	121.0	N
766	53	Male	88.67	1.72	29.972282	126.0	Y
767	68	Female	80.29	1.63	30.219429	93.0	Y

383 rows × 7 columns

```
miraculonB_df.dtypes
```

```
age                        int64
sex                       object
weight                   float64
height                   float64
BMI                      float64
protein_concentration    float64
RESPONSE                  object
dtype: object
```

```
miraculonB_df['sex'].replace({'Female':1, 'Male':0}, inplace=True)
```

```
miraculonB_df
```

	age	sex	weight	height	BMI	protein_concentration	RESPONSE
0	46	1	84.66	1.59	33.487599	148.0	N
1	47	1	71.21	1.64	26.476056	85.0	Y
3	59	1	62.94	1.50	27.973333	89.0	Y
7	57	0	93.50	1.63	35.191388	115.0	N
8	72	0	85.57	1.68	30.318169	197.0	N
...	...	...	...	...	...	...	...
762	70	1	62.21	1.66	22.575846	89.0	Y
764	65	0	112.86	1.76	36.434659	122.0	N
765	60	0	81.03	1.77	25.864215	121.0	N
766	53	0	88.67	1.72	29.972282	126.0	Y
767	68	1	80.29	1.63	30.219429	93.0	Y

383 rows × 7 columns

```
miraculonB_df['RESPONSE'].replace({'Y':1, 'N':0}, inplace=True)
```

```
miraculonB_df.dtypes
```

```
age                        int64
sex                        int64
weight                   float64
height                   float64
BMI                      float64
protein_concentration    float64
RESPONSE                   int64
dtype: object
```

```
miraculonB_df.describe()
```

	age	sex	weight	height	BMI	protein_concentration
count	383.000000	383.000000	383.000000	383.000000	383.000000	383.000000
mean	61.759791	0.488251	90.844100	1.682742	31.992628	122.077669
std	7.565750	0.500516	22.465539	0.097062	7.161227	30.183344
min	37.000000	0.000000	46.170000	1.430000	17.975421	56.000000
25%	57.000000	0.000000	74.340000	1.610000	26.704177	99.500000
50%	62.000000	0.000000	89.220000	1.680000	31.678201	118.000000
75%	67.000000	1.000000	104.135000	1.760000	36.318756	141.500000
max	79.000000	1.000000	160.120000	1.940000	67.515601	199.000000

```
df3=miraculonB_df
```

```
# scaling some of the features
cols_to_scale = ['age', 'weight', 'height','BMI','protein_concentration']
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df3[cols_to_scale] = scaler.fit_transform(df3[cols_to_scale])
```

```
for col in df3:
  print(f'{col}: {df3[col].unique()}')
```

```
age: [0.21428571 0.23809524 0.52380952 0.47619048 0.83333333 0.85714286
 0.38095238 0.5        0.42857143 0.66666667 0.4047619  0.33333333
 0.45238095 0.69047619 0.76190476 0.28571429 0.61904762 0.5952381
 0.64285714 0.80952381 0.57142857 0.97619048 0.73809524 0.95238095
 0.54761905 0.35714286 0.71428571 1.         0.78571429 0.88095238
 0.16666667 0.30952381 0.9047619  0.04761905 0.92857143 0.
 0.26190476 0.19047619]
sex: [1 0]
weight: [0.33777973 0.2197455  0.14716981 0.41535761 0.34576569 0.3968195
 0.69197016 0.26450197 0.39333041 0.29872751 0.35612111 0.74997806
 0.4230803  0.34681878 0.61254936 0.46967968 0.25247916 0.44107065
 0.0817025  0.50179903 0.33058359 0.36314173 0.20991663 0.36796841
 0.58411584 0.55594559 0.46634489 0.16191312 0.71443616 0.27836771
 0.4754717  0.50864414 0.02395788 0.14918824 0.51426064 0.8896007
 0.80421237 0.44914436 0.05660377 0.91426064 0.35726196 0.28740676
 0.436595   0.69556823 0.17586661 0.42448442 0.06265906 0.40008776
 0.16129882 0.28600263 0.71232997 0.5667398  0.4388767  0.39789381
 0.28328214 0.29679684 0.23931549 0.40947784 0.49881527 0.30153576
 0.78709961 0.53777973 0.59526108 0.2771391  0.45335674 0.01728828
 0.11926283 0.57876262 0.49925406 0.31127688 0.6912681  0.04826678
 0.61720053 0.90118473 0.72496709 0.30829311 0.54637999 0.79868363
 0.31320755 0.43483984 0.38437911 0.47591049 0.3763054  0.41421676
 0.39526108 0.74251865 0.46915314 0.43835015 1.         0.37270733
 0.40149188 0.30092146 0.32487933 0.29372532 0.50013164 0.13067135
 0.43369899 0.39815709 0.34778412 0.20684511 0.53830627 0.39648969
 0.23940325 0.53514699 0.21930671 0.32408951 0.19631417 0.47950856
 0.49284774 0.50750329 0.38078104 0.59096095 0.80351031 0.23413778
 0.58464239 0.41834138 0.1228609  0.33216323 0.24344011 0.2180781
 0.48161474 0.27345327 0.30039491 0.78622203 0.44695042 0.57849934
 0.25484862 0.19648969 0.3436595  0.21825362 0.50530935 0.30557262
 0.50495832 0.7538394  0.26151821 0.24036858 0.6794208  0.3307591
 0.09109258 0.1966652  0.37823607 0.57411145 0.28749452 0.23685827
 0.17314612 0.42114963 0.65739359 0.24080737 0.44493199 0.01869241
 0.72119351 0.20315928 0.16252742 0.2935498  0.51320755 0.08038613
 0.24194822 0.18279947 0.80903905 0.45581395 0.16454585 0.69986836
 0.4596753  0.18999561 0.28319438 0.4022817  0.12172005 0.63747258
 0.49179465 0.31952611 0.44071961 0.44809127 0.58209741 0.5819219
 0.40263273 0.63115401 0.53523475 0.41562089 0.58490566 0.33795524
 0.05993857 0.19455902 0.30846863 0.13821852 0.29495393 0.53681439
 0.92242211 0.10048267 0.55822729 0.39824484 0.25748135 0.48486178
 0.26248355 0.1842036  0.83589294 0.6326459  0.12566915 0.19429574
 0.67266345 0.49802545 0.4207986  0.58323826 0.13584906 0.26766125
 0.28214129 0.1093462  0.48284335 0.15831505 0.34620448 0.45283019
 0.57182975 0.4270294  0.31250548 0.0715226  0.6230803  0.45677929
 0.33839403 0.43071523 0.49548047 0.         0.12066696 0.33391839
 0.14418605 0.23624397 0.87915753 0.59289162 0.48740676 0.28047389
 0.15743747 0.20807372 0.57946468 0.83247038 0.48723124 0.22518649
 0.54831066 0.23150505 0.24273804 0.67204914 0.45370777 0.58051777
 0.225362   0.24914436 0.14743308 0.12093023 0.12988153 0.34295744
 0.65756911 0.06099166 0.31777095 0.00359807 0.6467749  0.47020623
 0.47178587 0.33813076 0.34313295 0.50688899 0.48714348 0.4602896
 0.39710399 0.35436595 0.29091707 0.32373848 0.37779728 0.27819219
 0.14067573 0.52812637 0.17455024 0.66494076 0.29197016 0.30021939
 0.19763054 0.33339184 0.36015796 0.58139535 0.70487056 0.14243089
 0.59622642 0.16867047 0.56243967 0.5087319  0.18516893 0.25107503
 0.4355419  0.01000439 0.20386134 0.55445371 0.74418605 0.15234752
 0.22676613 0.39394471 0.32189557 0.44940763 0.1960509  0.11733216
 0.26099166 0.34418605 0.31241773 0.39780606 0.56410706 0.52303642
```

```
df4=df3
```

```
#Train and split the data
X = df3.drop('RESPONSE', axis = 'columns')
y = testLabels = df3.RESPONSE.astype(np.float32)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=15, stratify = y)
```

```
y_train.value_counts()
```

```
1.0    168
0.0    138
Name: RESPONSE, dtype: int64
```

```
y_test.value_counts()
```

```
1.0    42
0.0    35
Name: RESPONSE, dtype: int64
```

```
X_train.shape
```

```
(306, 6)
```

```
X_test.shape
```

```
(77, 6)
```

```
X_train
```

	age	sex	weight	height	BMI	protein_concentration
12	0.500000	1	0.264502	0.490196	0.182919	0.580420
312	0.285714	0	0.293550	0.607843	0.167998	0.692308
755	0.666667	0	0.622817	0.725490	0.366953	0.503497
676	0.380952	1	0.047652	0.019608	0.139459	0.699301
315	0.690476	0	0.513208	0.647059	0.319111	0.391608
...	...	...	...	...	...	...
43	0.809524	1	0.714436	0.490196	0.549599	0.804196
382	0.714286	1	0.194559	0.411765	0.150051	0.370629
507	0.333333	1	0.225362	0.294118	0.218126	0.517483
474	0.547619	1	0.280474	0.411765	0.223526	0.405594
384	0.690476	0	0.308469	0.803922	0.122001	0.482517

306 rows × 6 columns

```
#X_train[:10]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
684	1	0	0	0	0.000000	1	0
2446	1	0	0	0	0.239437	1	1
2842	1	0	0	0	0.042254	0	0

```
len(X_train.columns)
```

```
6
```

```
# Use logistic regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report


def log_reg(X_train, y_train, X_test, y_test, weights):
  if weights ==-1:
    model = LogisticRegression()
  else:
    model = LogisticRegression(class_weight={0:weights[0], 1:weights[1]})
  model.fit(X_train, y_train)
  acc = model.score(X_test, y_test)
  print('Accuracy', acc, '\n')

  y_pred = model.predict(X_test)
  print('preds', y_pred[:5], '\n')

  cl_rep = classification_report(y_test, y_pred)
  print(cl_rep)
```

```
weights = -1 # pass - 1 to use logistics regression without weights
log_reg(X_train, y_train, X_test, y_test, weights)
```

```
Accuracy 0.8181818181818182

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

              precision    recall  f1-score   support

         0.0       0.89      0.69      0.77        35
         1.0       0.78      0.93      0.85        42

    accuracy                           0.82        77
   macro avg       0.83      0.81      0.81        77
weighted avg       0.83      0.82      0.81        77
```

```
# OVERSAMPLING TECHNIQUE TO ADDRESS CLASS IMBALANCE
X = df3.drop('RESPONSE', axis='columns')
y = df3['RESPONSE']
```

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy = 'minority')
X_sm, y_sm = smote.fit_resample(X, y)
y_sm.value_counts()
```

```
0    210
1    210
Name: RESPONSE, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_sm, y_sm, test_size = 0.2, random_state=15, stratify = y_sm)
```

```
y_train.value_counts()
```

```
0    168
1    168
```

```
Name: RESPONSE, dtype: int64
```

```
# LOGISTIC REGRESSION
weights = -1 # pass - 1 to use logistics regression without weights
log_reg(X_train, y_train, X_test, y_test, weights)
```

```
Accuracy 0.8452380952380952

preds [0 1 0 1 0]

              precision    recall  f1-score   support

           0       0.91      0.76      0.83        42
           1       0.80      0.93      0.86        42

    accuracy                           0.85        84
   macro avg       0.86      0.85      0.84        84
weighted avg       0.86      0.85      0.84        84
```

```
# feature importance
# better underatsna dthe model and the data
# reducing the number of input features

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
# fit the model
model.fit(X_train, y_train)
# get importance
importance = model.coef_[0]
acc = model.score(X_test, y_test)
print('Accuracy', acc, '\n')
y_pred = model.predict(X_test)
cl_rep = classification_report(y_test,y_pred)
print(cl_rep)
```

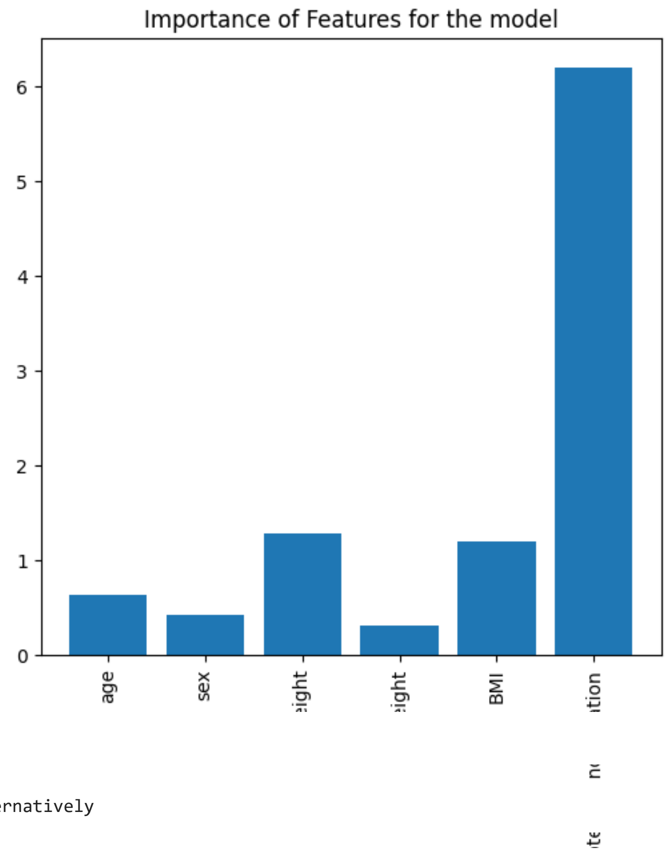
```
Accuracy 0.8452380952380952

              precision    recall  f1-score   support

           0       0.91      0.76      0.83        42
           1       0.80      0.93      0.86        42

    accuracy                           0.85        84
   macro avg       0.86      0.85      0.84        84
weighted avg       0.86      0.85      0.84        84
```

```
# the higher the coefficient of the feature the higher the importance regardless of the sign
importantFeatures = zip(X_train.columns, importance)
data =list(importantFeatures)
sorted_by_second = sorted (data, key =lambda tup: tup[1], reverse =True)
```

```
importance_abs = [abs(i) for i in importance]
```

```
def plotFeatures (X_train_columns, Fimportance):
  from matplotlib import pyplot as plt
  #plot the feature importance
  plt.figure(figsize = (6,6))
  y_pos= range(len(X_train_columns))
  plt.bar(X_train_columns, Fimportance)
  plt.title('Importance of Features for the model')
  plt.xticks(y_pos, X_train_columns, rotation=90)
  plt.show()
```

```
plotFeatures(X_train.columns, importance_abs)
```



#Alternatively

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
import tensorflow as tf
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
df4
```

	age	sex	weight	height	BMI	protein_concentration	RESPONSE
0	0.214286	1	0.337780	0.313725	0.313123	0.643357	0
1	0.238095	1	0.219746	0.411765	0.171591	0.202797	1
3	0.523810	1	0.147170	0.137255	0.201814	0.230769	1

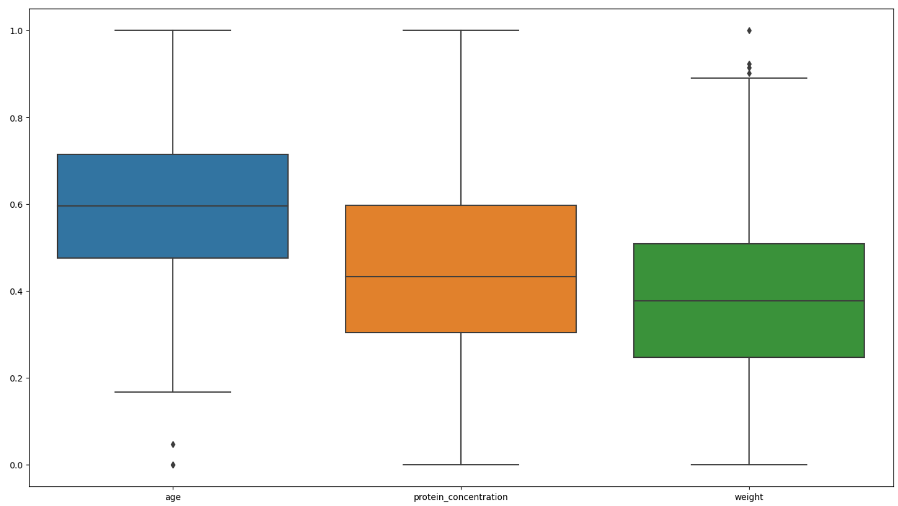
```
df4['RESPONSE'].value_counts(normalize=True)
```

```
1    0.548303
0    0.451697
Name: RESPONSE, dtype: float64
```

765	0.547619	0	0.305924	0.666667	0.159240	0.454545	0

```
plt.figure(figsize=(18,10))
sns.boxplot(data=df4[['age','protein_concentration','weight']])
```

```
<Axes: >
```



	age	sex	weight	height	BMI	protein_concentration
count	383.000000	383.000000	383.000000	383.000000	383.000000	383.000000
mean	0.589519	0.488251	0.392050	0.495572	0.282946	0.462082
std	0.180137	0.500516	0.197153	0.190317	0.144554	0.211072
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

```
# To split data into X and y variables before handling class imbalance using oversampling
X = df4.drop('RESPONSE', axis=1)
y = df4['RESPONSE']
sampler = RandomOverSampler()
X_resampled, y_resampled = sampler.fit_resample(X, y)
```

```
X_resampled.shape
```

```
(420, 6)
```

```
y_resampled.shape
```

```
(420,)
```

```
y_resampled.value_counts(normalize=True)
```

```
0    0.5
1    0.5
Name: RESPONSE, dtype: float64
```

```
plt.figure(figsize=(15,8))
plt.title('Class Balance', fontsize=15)
sns.countplot(x= y_resampled)
```

```
<Axes: title={'center': 'Class Balance'}, xlabel='RESPONSE', ylabel='count'>
```

```
X_resampled.head()
```

	age	sex	weight	height	BMI	protein_concentration
0	0.214286	1	0.337780	0.313725	0.313123	0.643357
1	0.238095	1	0.219746	0.411765	0.171591	0.202797
2	0.523810	1	0.147170	0.137255	0.201814	0.230769
3	0.476190	0	0.415358	0.392157	0.347515	0.412587
4	0.833333	0	0.345766	0.490196	0.249146	0.986014


```
y_resampled.tail()
```

```
415    0
416    0
417    0
418    0
419    0
Name: RESPONSE, dtype: int64
```

```
[0 1]
```

```
y_resampled.tail()
```

```
10321    1
10322    1
10323    1
10324    1
10325    1
Name: Churn, dtype: int64
```

```
X = X_resampled
```

```
y = y_resampled
```

```
corr = df.corr()
corr
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
SeniorCitizen	1.000000	0.015683	0.219874	0.102411
tenure	0.015683	1.000000	0.246862	0.825880
MonthlyCharges	0.219874	0.246862	1.000000	0.651065
TotalCharges	0.102411	0.825880	0.651065	1.000000

```
# Instantiate SelectKBest with f_classif as the scoring function
selector = SelectKBest(score_func=f_classif, k=5)

# Fit the selector to the data
selector.fit(X_resampled, y_resampled)

# Get the indices of the selected features
selected_features_indices = selector.get_support(indices=True)

# Get the names of the selected features
```

```
selected_features_names = X_resampled.columns[selected_features_indices]

# Print the names of the selected features
print(selected_features_names)
```

```
Index(['age', 'weight', 'height', 'BMI', 'protein_concentration'], dtype='object')
```

```
# Display the scores of the top 5 features
scores = selector.scores_
top_k_scores = sorted(scores, reverse=True)[:5]
top_k_indices = np.argsort(scores)[::-1][:5]

print("Top 5 feature scores:")
for i in range(len(top_k_scores)):
    print("Feature {}: Score = {:.2f}".format(top_k_indices[i], top_k_scores[i]))
```

```
Top 5 feature scores:
Feature 5: Score = 316.64
Feature 3: Score = 0.74
Feature 2: Score = 0.62
Feature 0: Score = 0.28
Feature 4: Score = 0.24
```

```
# Get the names and scores of the top 10 features
feature_names = df4.drop('RESPONSE', axis=1).columns
top_scores = selector.scores_.argsort()[-5:][::-1]
top_features = feature_names[top_scores]

# Print the names and scores of the top 10 features
for i, feature in enumerate(top_features):
    print("{}. {} ({:.2f})".format(i+1, feature, selector.scores_[top_scores][i]))

# Create a bar plot of the top 10 features and their scores
plt.figure(figsize=(20,8))
sns.set(font_scale=1.5)
plt.bar(range(len(top_scores)), selector.scores_[top_scores])
plt.xticks(range(len(top_scores)), top_features, rotation='horizontal')
plt.xlabel("Feature")
plt.ylabel("Score")
plt.title("Top 5 Features")
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-0c79e6c48ff7> in <cell line: 2>()
      1 # Get the names and scores of the top 10 features
----> 2 feature_names = df4.drop('RESPONSE', axis=1).columns
      3 top_scores = selector.scores_.argsort()[-5:][::-1]
      4 top_features = feature_names[top_scores]
      5 

NameError: name 'df4' is not defined
```

SEARCH STACK OVERFLOW

```
# select the top K features using f_classic
kbest = SelectKBest(score_func=f_classif, k='all')
X_resampled = kbest.fit_transform(X_resampled, y_resampled)

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3,random_state = 30)
```

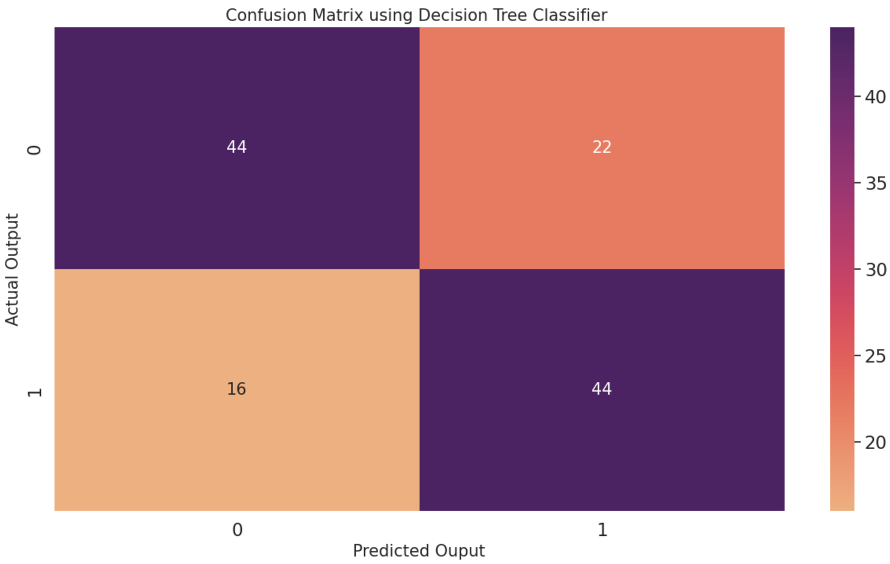
```
# train a decision tree classifier on the data
clf = DecisionTreeClassifier()

#clf = RandomForestClassifier()

clf.fit(X_train, y_train)

# test the classifier on the test set and print the classification report
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.67      0.70        66
           1       0.67      0.73      0.70        60

    accuracy                           0.70       126
   macro avg       0.70      0.70      0.70       126
weighted avg       0.70      0.70      0.70       126
```

```
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('---------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.70


The confusion matrix is:
[[44 22]
 [16 44]]


---------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.73      0.67      0.70        66
           1       0.67      0.73      0.70        60

    accuracy                           0.70       126
   macro avg       0.70      0.70      0.70       126
weighted avg       0.70      0.70      0.70       126
```

```
plt.figure(figsize=(15,8))
zx = sns. heatmap(conf_matrix, cmap ='flare', annot_kws={"size": 15}, annot= True, fmt = 'd')
plt.title('Confusion Matrix using Decision Tree Classifier ', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```




```
# Define the Decision Tree classifier
dt = DecisionTreeClassifier()

# Define the hyperparameters to tune
param_grid = {'max_depth': [12, 15, 20],
              'min_samples_split': [2,3, 4, 6, 8,],
              'min_samples_leaf': [1, 2, 3, 4, 5]}

# Perform hyperparameter tuning using GridSearchCV
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, verbose=0)
grid_search.fit(X, y)

# Print the results
print("Best accuracy score: {:.2f}".format(grid_search.best_score_))
print("Best parameters: {}".format(grid_search.best_params_))
```

```
Best accuracy score: 0.79
Best parameters: {'max_depth': 15, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

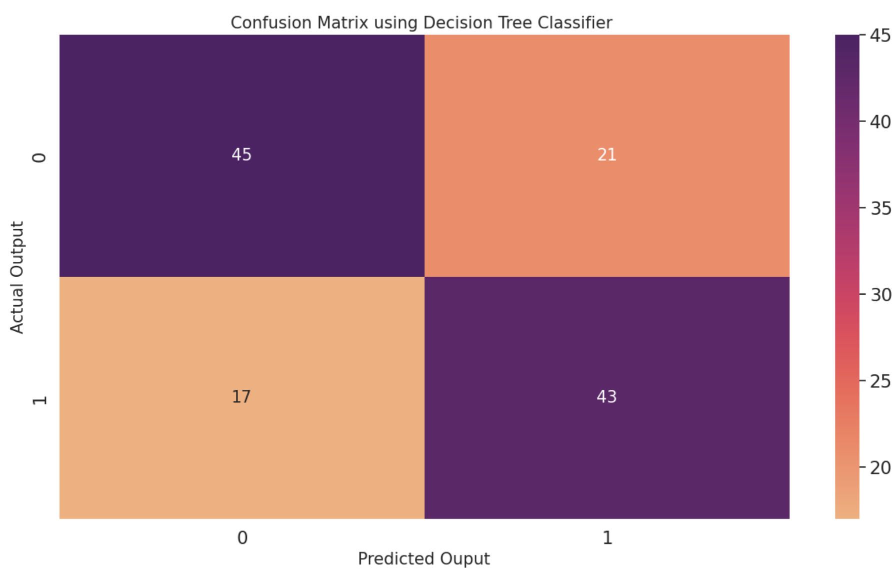
```
# Using best hypeparameter


# train a decision tree classifier on the data
clf = DecisionTreeClassifier(max_depth= 20, min_samples_leaf = 1, min_samples_split= 2)

#clf = RandomForestClassifier()

clf.fit(X_train, y_train)

# test the classifier on the test set and print the classification report
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.68      0.70        66
           1       0.67      0.72      0.69        60

    accuracy                           0.70       126
   macro avg       0.70      0.70      0.70       126
weighted avg       0.70      0.70      0.70       126
```

```
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('--------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.70


The confusion matrix is:
[[45 21]
 [17 43]]


--------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.73      0.68      0.70        66
           1       0.67      0.72      0.69        60

    accuracy                           0.70       126
   macro avg       0.70      0.70      0.70       126
weighted avg       0.70      0.70      0.70       126
```

```
plt.figure(figsize=(15,8))
zx = sns. heatmap(conf_matrix, cmap ='flare', annot_kws={"size": 15}, annot= True, fmt = 'd')
plt.title('Confusion Matrix using Decision Tree Classifier ', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```



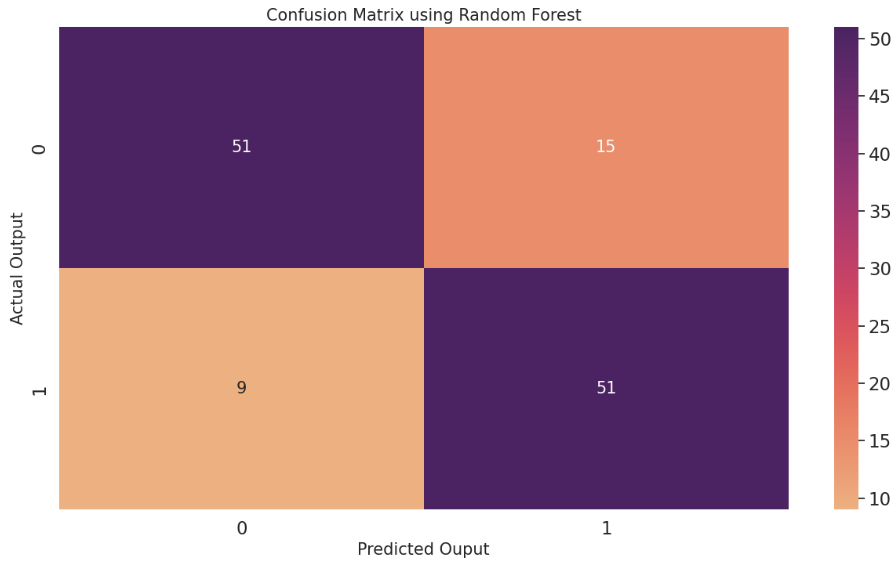
```
#Using RandomForest

# train a decision tree classifier on the data
#clf = DecisionTreeClassifier()

clf = RandomForestClassifier()

clf.fit(X_train, y_train)

# test the classifier on the test set and print the classification report
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.77      0.81        66
           1       0.77      0.85      0.81        60

    accuracy                           0.81       126
   macro avg       0.81      0.81      0.81       126
weighted avg       0.81      0.81      0.81       126
```

```
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('---------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.81


The confusion matrix is:
[[51 15]
 [ 9 51]]


---------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.85      0.77      0.81        66
           1       0.77      0.85      0.81        60

    accuracy                           0.81       126
   macro avg       0.81      0.81      0.81       126
weighted avg       0.81      0.81      0.81       126
```

```
plt.figure(figsize=(15,8))
zx = sns. heatmap(conf_matrix, cmap ='flare',annot_kws={"size": 15}, annot= True, fmt = 'd')
plt.title('Confusion Matrix using Random Forest', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```




```
# GRADIENT BOOSTED DECISION TREE
```

```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.5, max_depth=14, random_state=42)
clf.fit(X_train, y_train)

# test the classifier on the test set and print the classification report
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

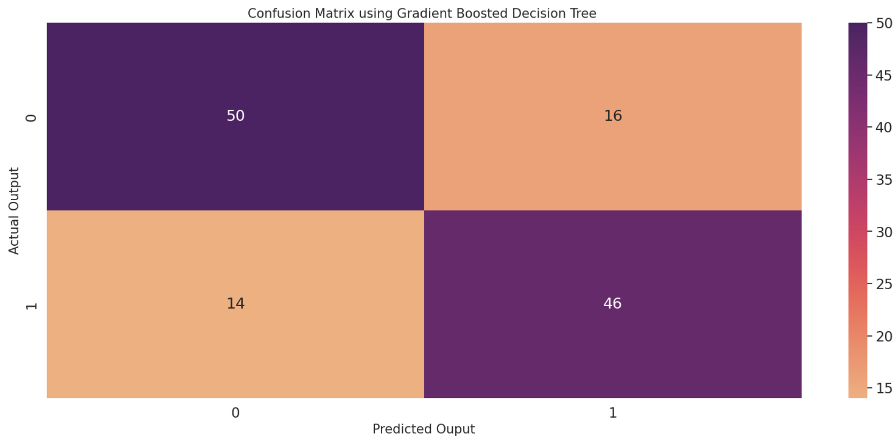
```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77        66
           1       0.74      0.77      0.75        60

    accuracy                           0.76       126
   macro avg       0.76      0.76      0.76       126
weighted avg       0.76      0.76      0.76       126
```

```
from sklearn import metrics
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('---------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.76


The confusion matrix is:
[[50 16]
 [14 46]]


---------------------
Classification Report:

              precision    recall  f1-score   support
```

```
              0       0.78      0.76      0.77        66
              1       0.74      0.77      0.75        60

       accuracy                           0.76       126
      macro avg       0.76      0.76      0.76       126
   weighted avg       0.76      0.76      0.76       126
```

```
plt.figure(figsize=(20,8))
zx = sns. heatmap(conf_matrix, cmap ='flare', annot_kws={"size": 18},annot= True, fmt = 'd')
plt.title('Confusion Matrix using Gradient Boosted Decision Tree ', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```




```
# Identifying top 10 features driving the Gradient Boosted Decision model by allowing the model to select importance features itself


# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state = 40)



scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform (X_test)


# Step 4: Model Evaluation
from sklearn.metrics import f1_score


clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.5, max_depth=18, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
score = f1_score(y_test, y_pred)
print(f"F1-score: {score:.4f}")
```

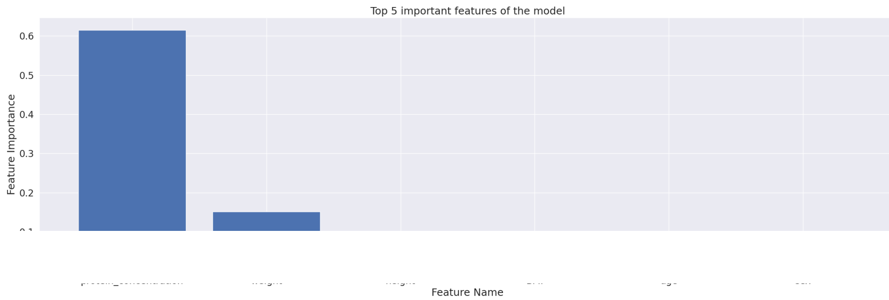
```
F1-score: 0.8130
```

```
from sklearn import metrics
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('---------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.80


The confusion matrix is:
[[42 10]
 [13 50]]


---------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.76      0.81      0.79        52
           1       0.83      0.79      0.81        63

    accuracy                           0.80       115
   macro avg       0.80      0.80      0.80       115
weighted avg       0.80      0.80      0.80       115
```

```
#Step 5: Feature Importance Analysis
feature_importances = clf.feature_importances_
feature_names = df4.drop('RESPONSE', axis=1).columns
```

```
# Step 6: Plot Feature Importance Graph


top_features = pd.Series(feature_importances, index=feature_names).sort_values(ascending=False)[:10]
plt.figure(figsize=(27,8))
plt.bar(top_features.index, top_features)
plt.title('Top 5 important features of the model')
plt.xlabel('Feature Name')
plt.ylabel('Feature Importance')
plt.show()
```




```
plt.figure(figsize=(20,8))
zx = sns. heatmap(conf_matrix, cmap ='flare', annot_kws={"size": 18},annot= True, fmt = 'd')
plt.title('Confusion Matrix using Gradient Boosted Decision Tree ', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```