PROJECT DVELOPMENT PHASE

SPRINT-2

BUILDING MODELS

SPLIT THE DATAS

```
x=df.drop(['Serial No.','Chance of Admit '],axis=1)
y=df['Chance of Admit ']
print(x.shape,y.shape)
```

TRAIN AND TEST THE DATA

```
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)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

SCALING:

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x_train[x_train.columns]=mms.fit_transform(x_train[x_train.columns].values)
x_test[x_test.columns]=mms.transform(x_test[x_test.columns].values)
```

BUILDING MODELS

MODEL 1: RANDOM FOREST REGRESSOR

```
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor()
```

```
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(y_pred)
print(y_test)
```

EVALUATION

```
from sklearn.metrics import mean_squared_error,
r2_score,mean_absolute_error,roc_auc_score,recall_score
print('model score:',model.score(x_test,y_test))
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
print('roc score:',roc_auc_score(y_test>0.5, y_pred>0.5))
print('recall score:',recall_score(y_test>0.5, y_pred>0.5))
```

MODEL 2: LINEAR REGRESSION

```
x1=df.drop(['Serial No.','Chance of Admit '],axis=1)
y1=df['Chance of Admit ']
x1_train,x1_test,y1_train,y1_test=train_test_split(x1,y1,test_size=0.2)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x1_train=sc.fit_transform(x1_train)
x1_test=sc.fit_transform(x1_test)
from sklearn.linear_model import LinearRegression
model1=LinearRegression()
model1.fit(x1_train,y1_train)
```

```
y1_pred=model1.predict(x1_test)
```

EVALUATION

```
print('model score:',model1.score(x1_test,y1_test))
print('Mean Absolute Error:', mean_absolute_error(y1_test, y1_pred))
print('Mean Squared Error:', mean_squared_error(y1_test, y1_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y1_test,
y1_pred)))
print('roc score:',roc_auc_score(y1_test>0.5, y1_pred>0.5))
print('recall score:',recall_score(y1_test>0.5, y1_pred>0.5))
```

MODEL 3: LOGISTIC REGRESSION

```
x2=df.iloc[:,1:8].values
y2=df.iloc[:,-1:].values
x2_train,x2_test,y2_train,y2_test=train_test_split(x1,y1,test_size=0.2)
y2_train=y2_train>0.5
y2_test=y2_test>0.5
from sklearn.linear_model import LogisticRegression
model2=LogisticRegression()
model2.fit(x2_train,y2_train)
```

EVALUATION

```
from sklearn.metrics import accuracy_score,roc_auc_score,recall_score
y2_pred=model2.predict(x2_test)
print('model score:',model2.score(x2_test,y2_test))
```

```
print('roc score:',roc_auc_score(y2_test, y2_pred))
print('recall score:',recall_score(y2_test, y2_pred))
print(type(y2_test),type(y2_pred))
```

XGBREGRESSOR

```
import xgboost as xgb
xg_reg=xgb.XGBRegressor(objective='reg:logistic',colsample_bytree=0.3,learning_rate=0.5,max_depth=5,n_estimators=100)
x3=df.iloc[ :,1:8].values
y3=df.iloc[ :,-1: ].values
x3_train,x3_test,y3_train,y3_test=train_test_split(x3,y3,test_size=0.2)
xg_reg.fit(x3_test,y3_test)
xg_reg.score(x3_test,y3_test)
y3_pred=xg_reg.predict(x3_test)
np.sqrt(mean_squared_error(y3_test,y3_pred))
```

SAVING THE MODEL:

```
 import joblib
joblib.dump(model,'model.pkl')
```

SPLIT THE DATA INTO DEPENDENT AND INDEPENDENT VARIABLES

In [18]:
```
x=df.drop(['Serial No.','Chance of Admit '],axis=1)
y=df['Chance of Admit ']
print(x.shape,y.shape)
```

```
(400, 7) (400,)
```

TRAIN AND TEST THE DATA

In [19]:
```
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)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(320, 7)
(80, 7)
(320,)
(80,)
```

In [20]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x_train[x_train.columns]=mms.fit_transform(x_train[x_train.columns].values)
x_test[x_test.columns]=mms.transform(x_test[x_test.columns].values)
```

BUILDING MODEL(RANDOM FOREST REGRESSOR)

In [21]:
```
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(y_pred)
print(y_test)
```

```
[0.7122 0.871  0.4652 0.735  0.7675 0.5536 0.7058 0.7311 0.9252 0.8572
 0.667  0.6446 0.6406 0.7494 0.7868 0.4745 0.624  0.6708 0.6034 0.8027
 0.6961 0.8995 0.5204 0.6888 0.7303 0.7711 0.7774 0.6876 0.4735 0.6785
 0.7882 0.5214 0.6294 0.7082 0.6321 0.9324 0.6365 0.7152 0.5684 0.6115
 0.5309 0.68   0.5397 0.8431 0.6541 0.7117 0.65   0.9083 0.723  0.5496
 0.7322 0.7185 0.8958 0.9507 0.6091 0.8255 0.6528 0.9276 0.9386 0.8172
 0.6118 0.7808 0.6626 0.671  0.6695 0.9369 0.7554 0.9571 0.6104 0.6372
 0.5865 0.7475 0.644  0.7155 0.7164 0.9538 0.7338 0.7827 0.6155 0.6633]
250    0.74
260    0.87
78     0.44
40     0.78
192    0.86
       ...
130    0.96
10     0.52
249    0.77
331    0.73
123    0.61
Name: Chance of Admit , Length: 80, dtype: float64
```

EVALUATION

In [22]:
```
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error,roc_auc_score,recall_score
print('model score:',model.score(x_test,y_test))
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_test, y_pred)))
print('roc score:',roc_auc_score(y_test>0.5, y_pred>0.5))
print('recall score:',recall_score(y_test>0.5, y_pred>0.5))
```

```
model score: 0.8855653618704457
Mean Absolute Error: 0.045142499999999974
Mean Squared Error: 0.004044682000000002
Root Mean Squared Error: 0.06359718547231474
roc score: 0.6875
recall score: 1.0
```

In [23]:
```
x1=df.drop(['Serial No.','Chance of Admit '],axis=1)
y1=df['Chance of Admit ']
x1_train,x1_test,y1_train,y1_test=train_test_split(x1,y1,test_size=0.2)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x1_train=sc.fit_transform(x1_train)
x1_test=sc.fit_transform(x1_test)
```

In [24]:
```
#BUILDING MODEL(LINEAR REGRESSION)
```

In [25]:
```
from sklearn.linear_model import LinearRegression
model1=LinearRegression()
model1.fit(x1_train,y1_train)
y1_pred=model1.predict(x1_test)
print('model score:',model1.score(x1_test,y1_test))
print('Mean Absolute Error:', mean_absolute_error(y1_test, y1_pred))
print('Mean Squared Error:', mean_squared_error(y1_test, y1_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y1_test, y1_pred)))
print('roc score:',roc_auc_score(y1_test>0.5, y1_pred>0.5))
print('recall score:',recall_score(y1_test>0.5, y1_pred>0.5))
```

```
model score: 0.731897247854991
Mean Absolute Error: 0.05047269380860454
Mean Squared Error: 0.004217926548121353
Root Mean Squared Error: 0.06494556603896338
roc score: 0.6601731601731602
recall score: 0.987012987012987
```

In [26]:
```
x2=df.iloc[:,1:8].values
y2=df.iloc[:,-1:].values
x2_train,x2_test,y2_train,y2_test=train_test_split(x1,y1,test_size=0.2)
y2_train=y2_train>0.5
y2_test=y2_test>0.5
```

In [27]:
```

from sklearn.linear_model import LogisticRegression
model2=LogisticRegression()
model2.fit(x2_train,y2_train)
```

```
C:\Users\mithr\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

In [27]:

```
from sklearn.linear_model import LogisticRegression
model2=LogisticRegression()
model2.fit(x2_train,y2_train)
```

```
C:\Users\mithr\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[27]:

```
LogisticRegression()
```

In [28]:

```
from sklearn.metrics import accuracy_score,roc_auc_score,recall_score
y2_pred=model2.predict(x2_test)
print('model score:',model2.score(x2_test,y2_test))
print('roc score:',roc_auc_score(y2_test, y2_pred))
print('recall score:',recall_score(y2_test, y2_pred))
print(type(y2_test),type(y2_pred))
```

```
model score: 0.95
roc score: 0.6
recall score: 1.0
<class 'pandas.core.series.Series'> <class 'numpy.ndarray'>
```

In [29]:

```
import xgboost as xgb
```

```
C:\Users\mithr\anaconda3\lib\site-packages\xgboost\compat.py:36: FutureWarning: pandas.Int64Index is deprecated and will be rem
oved from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

In [30]:

```
xg_reg=xgb.XGBRegressor(objective='reg:logistic',colsample_bytree=0.3,learning_rate=0.5,max_depth=5,n_estimators=100)
```

In [31]:

```
x3=df.iloc[ :,1:8].values
y3=df.iloc[ :,-1: ].values
```

In [32]:

```
x3_train,x3_test,y3_train,y3_test=train_test_split(x3,y3,test_size=0.2)
```

In [33]:

```
xg_reg.fit(x3_test,y3_test)
xg_reg.score(x3_test,y3_test)
```

Out[33]:

```
0.9936079320168772
```

In [34]:

```
y3_pred=xg_reg.predict(x3_test)
np.sqrt(mean_squared_error(y3_test,y3_pred))
```

Out[34]:

```
0.811759434546517107
```

SAVING THE MODEL

#Though the accuracy of Logistic regression model is more we prefer Random forest regressor if we also want the percentage of chance or else we can use Logistic regression model.