Referred websites

1. Data visualization - #https://towardsdatascience.com/machine-learning-workflow-on-diabetes-data-part-01-573864fcc6b8

undefined. Insights on attributes and their importance - #https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3576830/#:~:text=For%20example%2C%20a%20correlation%20coefficient,use%20the%20most%20appropriate%20one.

undefined. Tree visualization - https://mljar.com/blog/visualize-decision-tree/

undefined. Grid Search Code- https://vitalflux.com/decision-tree-hyperparameter-tuning-grid-search-example/

Diabetes Risk Prediction

```
# import libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

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

```
# load dataset

df=pd.read_csv(r'/work/1-At_risk- prediction.csv')
df.columns=[x.lower() for x in df.columns]      # lowercase for all columnn names
df.columns=df.columns.str.replace(' ','_')
```

```
df
```

	age	bp	sugarlevel	skinthickness	insulin	bmi	function	tests	result
0	50	148	72	35	0	33.6	0.627	6	1
1	31	85	66	29	0	26.6	0.351	1	0
2	32	183	64	0	0	23.3	0.672	8	1
3	21	89	66	23	94	28.1	0.167	1	0
4	33	137	40	35	168	43.1	2.288	0	1
...	...	...	...	...	...	...	...	...	...
763	63	101	76	48	180	32.9	0.171	10	0
764	27	122	70	27	0	36.8	0.340	2	0
765	30	121	72	23	112	26.2	0.245	5	0
766	47	126	60	0	0	30.1	0.349	1	1
767	23	93	70	31	0	30.4	0.315	1	0

768 rows × 9 columns

```
df.columns
```

```
Index(['age', 'bp', 'sugarlevel', 'skinthickness', 'insulin', 'bmi',
       'function', 'tests', 'result'],
      dtype='object')
```

```
# Check for null cells

nullCheck=pd.DataFrame()
```

```
nullCheck['Number of null values']=df.isnull().sum()
nullCheck['Percentage of Null Values']=(df.isnull().sum() / df.shape[0]) * 100
nullCheck=nullCheck.sort_values('Percentage of Null Values',ascending=False)
nullCheck
```

	Number of null values	Percentage of Null Values
age	0	0.0
bp	0	0.0
sugarlevel	0	0.0
skinthickness	0	0.0
insulin	0	0.0
bmi	0	0.0
function	0	0.0
tests	0	0.0
result	0	0.0

```
df[df['skinthickness']==0]
```

	age	bp	sugarlevel	skinthickness	insulin	bmi	function	tests	result
2	32	183	64	0	0	23.3	0.672	8	1
5	30	116	74	0	0	25.6	0.201	5	0
7	29	115	0	0	0	35.3	0.134	10	0
9	54	125	96	0	0	0.0	0.232	8	1
10	30	110	92	0	0	37.6	0.191	4	0
...	...	...	...	...	...	...	...	...	...
757	52	123	72	0	0	36.3	0.258	0	1
758	26	106	76	0	0	37.5	0.197	1	0
759	66	190	92	0	0	35.5	0.278	6	1
762	33	89	62	0	0	22.5	0.142	9	0
766	47	126	60	0	0	30.1	0.349	1	1

227 rows × 9 columns

skinthickness=0 --> insulin=0 (227 rows)

```
df[df['bmi']==0]
```

	age	bp	sugarlevel	skinthickness	insulin	bmi	function	tests	result
9	54	125	96	0	0	0.0	0.232	8	1
49	24	105	0	0	0	0.0	0.305	7	0
60	21	84	0	0	0	0.0	0.304	2	0
81	22	74	0	0	0	0.0	0.102	2	0
145	21	102	75	23	0	0.0	0.572	0	0
371	21	118	64	23	89	0.0	1.731	0	0
426	25	94	0	0	0	0.0	0.256	0	0
494	22	80	0	0	0	0.0	0.174	3	0
522	26	114	0	0	0	0.0	0.189	6	0
684	69	136	82	0	0	0.0	0.640	5	0
706	30	115	0	0	0	0.0	0.261	10	1

bmi=0 (11 rows) - consider as null cells

```
# converting all zeros to np.nan

dictionary1={0:np.nan}
df=df.replace(dictionary1)
```

```
df
```

	age	bp	sugarlevel	skinthickness	insulin	bmi	function	tests	result
0	50.0	148.0	72.0	35.0	0.0	33.6	0.627	6.0	1.0
1	31.0	85.0	66.0	29.0	0.0	26.6	0.351	1.0	0.0
2	32.0	183.0	64.0	0.0	0.0	23.3	0.672	8.0	1.0
3	21.0	89.0	66.0	23.0	94.0	28.1	0.167	1.0	0.0
4	33.0	137.0	40.0	35.0	168.0	43.1	2.288	0.0	1.0
...	...	...	...	...	...	...	...	...	...
763	63.0	101.0	76.0	48.0	180.0	32.9	0.171	10.0	0.0
764	27.0	122.0	70.0	27.0	0.0	36.8	0.340	2.0	0.0
765	30.0	121.0	72.0	23.0	112.0	26.2	0.245	5.0	0.0
766	47.0	126.0	60.0	0.0	0.0	30.1	0.349	1.0	1.0
767	23.0	93.0	70.0	31.0	0.0	30.4	0.315	1.0	0.0

768 rows × 9 columns

```
# checking null values
nullCheck=pd.DataFrame()
nullCheck['Number of null values']=df.isnull().sum()
nullCheck['Percentage of Null Values']=(df.isnull().sum() / df.shape[0]) * 100
nullCheck=nullCheck.sort_values('Percentage of Null Values',ascending=False)
nullCheck
```

	Number of null values	Percentage of Null Values
result	500	65.104167
insulin	374	48.697917
skinthickness	227	29.557292
tests	111	14.453125
sugarlevel	35	4.557292
bmi	11	1.432292
bp	5	0.651042
age	0	0.000000
function	0	0.000000

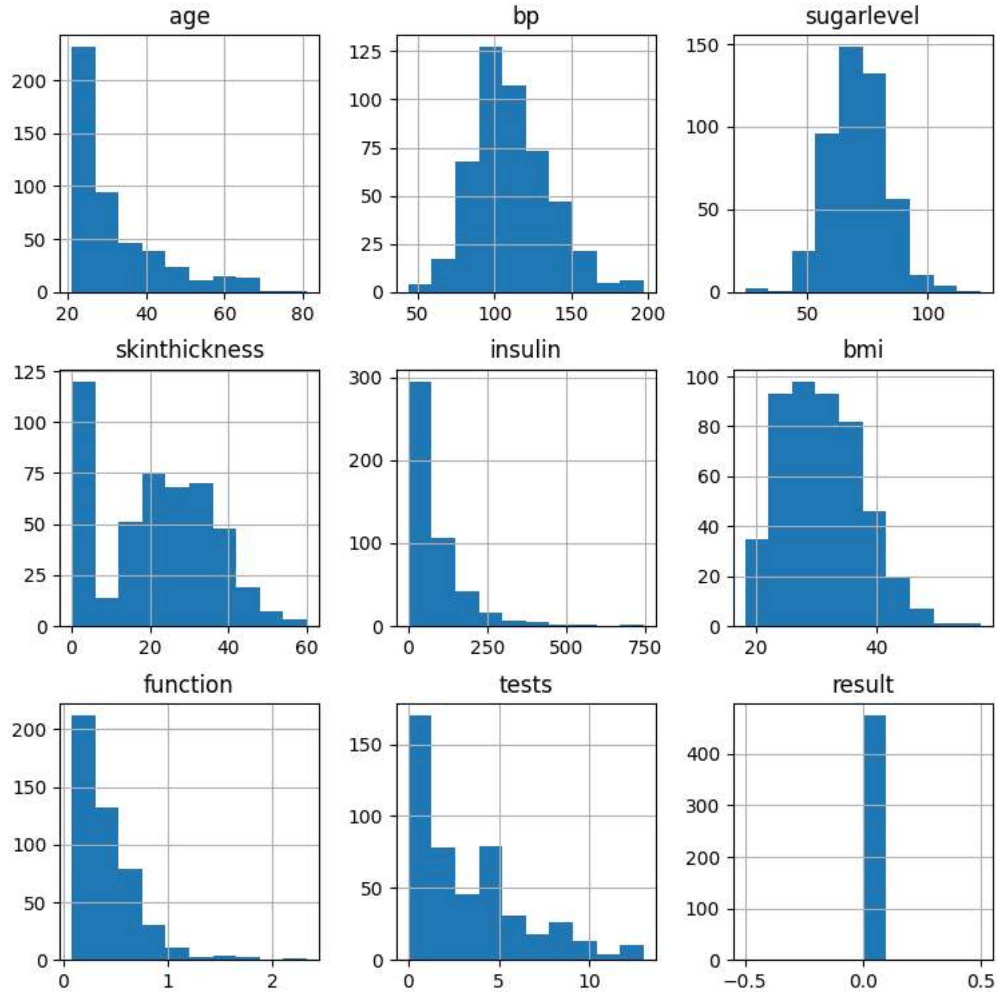
```
# change np.nan back to 0
dictionary2={np.nan:0}
df=df.replace(dictionary2)
```

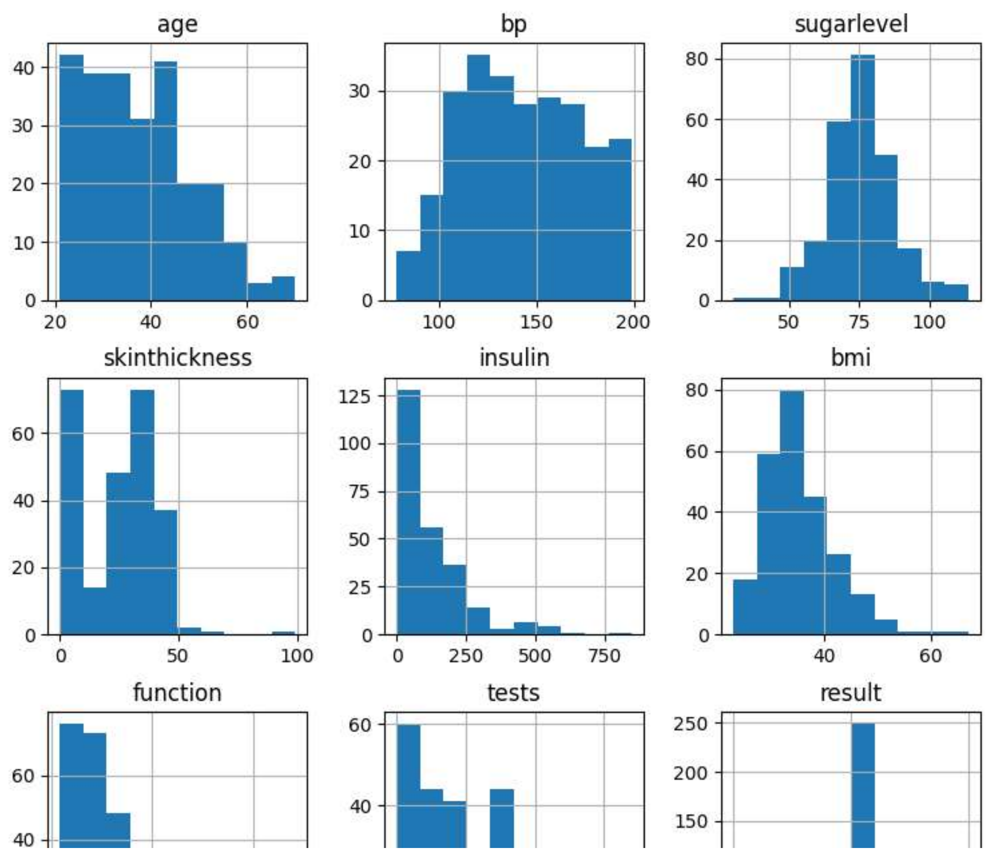
DATA VISUALISATION

HISTOGRAM

```
df.groupby('result').hist(figsize=(9, 9))
```

```
result
0.0    [[AxesSubplot(0.125,0.666111;0.215278x0.213889...
1.0    [[AxesSubplot(0.125,0.666111;0.215278x0.213889...
dtype: object
```

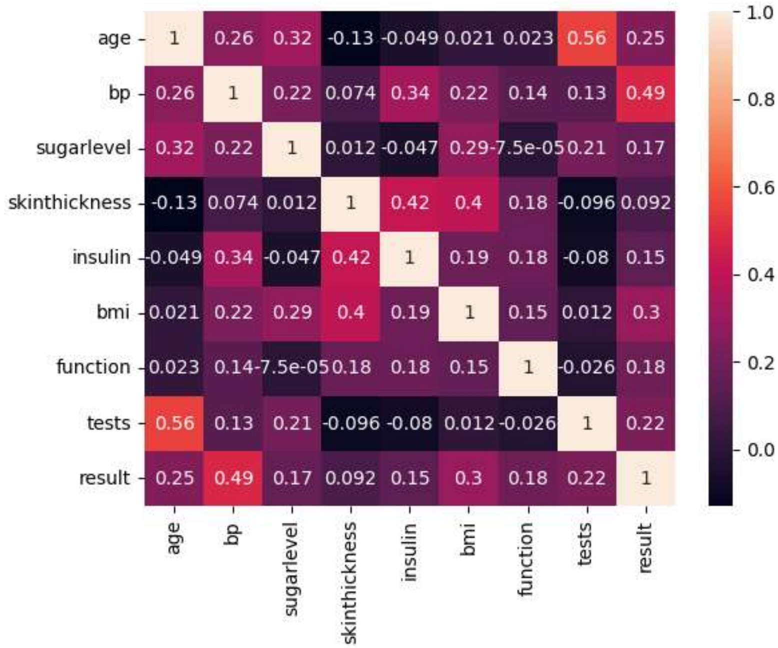




CORRELATION MATRIX

```
corr = df.corr()
sns.heatmap(corr, annot=True)
```

```
<AxesSubplot: >
```



DATA PREPROCESSING

DROP COLUMNS

```
df.drop(['tests', 'skinthickness'], axis=1, inplace=True)

# tests - because of common understanding
# sckinthickness - insignificant correlation = 0.092
```

HANDLE ZERO / NULL VALUES

drop rows with 0 in bmi, bp and sugarlevel

```
bmi_rows=df[df["bmi"]==0].index
df.drop(bmi_rows, axis=0, inplace=True)
```

```
bp_rows=df[df["bp"]==0].index
df.drop(bp_rows, axis=0, inplace=True)
```

```
sl_rows=df[df["sugarlevel"]==0].index
df.drop(sl_rows, axis=0, inplace=True)
```

drop skinthickness, tests

```
df
```

	age	bp	sugarlevel	insulin	bmi	function	result
0	50.0	148.0	72.0	0.0	33.6	0.627	1.0
1	31.0	85.0	66.0	0.0	26.6	0.351	0.0
2	32.0	183.0	64.0	0.0	23.3	0.672	1.0
3	21.0	89.0	66.0	94.0	28.1	0.167	0.0
4	33.0	137.0	40.0	168.0	43.1	2.288	1.0
...	...	...	...	...	...	...	...
763	63.0	101.0	76.0	180.0	32.9	0.171	0.0
764	27.0	122.0	70.0	0.0	36.8	0.340	0.0
765	30.0	121.0	72.0	112.0	26.2	0.245	0.0
766	47.0	126.0	60.0	0.0	30.1	0.349	1.0

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
X
```

	age	bp	sugarlevel	insulin	bmi	function
0	50.0	148.0	72.0	0.0	33.6	0.627
1	31.0	85.0	66.0	0.0	26.6	0.351
2	32.0	183.0	64.0	0.0	23.3	0.672
3	21.0	89.0	66.0	94.0	28.1	0.167
4	33.0	137.0	40.0	168.0	43.1	2.288
...	...	...	...	...	...	...
763	63.0	101.0	76.0	180.0	32.9	0.171
764	27.0	122.0	70.0	0.0	36.8	0.340
765	30.0	121.0	72.0	112.0	26.2	0.245
766	47.0	126.0	60.0	0.0	30.1	0.349
767	23.0	93.0	70.0	0.0	30.4	0.315

724 rows × 6 columns

Fill insulin with mode

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = 0, strategy='most_frequent')
col_to_be_filled=X.iloc[:,3].values
col_to_be_filled=col_to_be_filled.reshape(-1,1)
imputer.fit(col_to_be_filled)
X.iloc[:,3]=imputer.transform(col_to_be_filled)
```

```
X
```

	age	bp	sugarlevel	insulin	bmi	function
0	50.0	148.0	72.0	105.0	33.6	0.627
1	31.0	85.0	66.0	105.0	26.6	0.351
763	63.0	101.0	76.0	180.0	32.9	0.171
765	30.0	121.0	72.0	112.0	26.2	0.245

TRAIN-TEST-SPLIT

```
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=40)
```

FEATURE SCALING

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

TRAINING AND EVALUATION

1. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
```

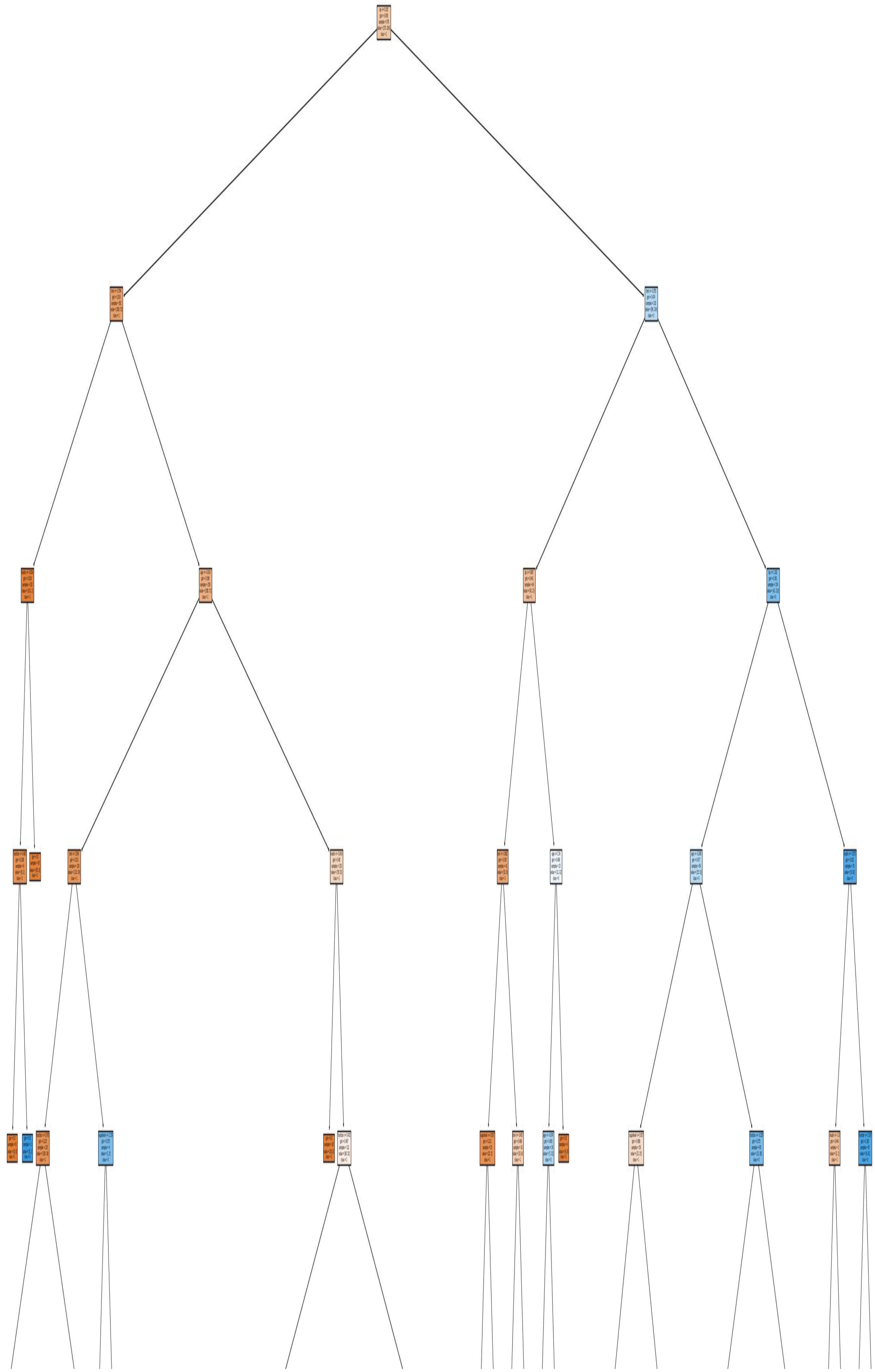
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
Y.unique()
```

```
array([1., 0.])
```

Double-click (or enter) to edit

```
from sklearn import tree
fig = plt.figure(figsize=(50,50))
graph = tree.plot_tree(dtc,
                   feature_names=X.columns,
                   class_names=['1','0'],
                   filled=True)
```



```
y_pred_reg = dtc.predict(X_test)
```

EVALUATION ON REGULAR MODEL

```
# CROSS VALIDATION

from sklearn.model_selection import cross_val_score
score=cross_val_score(dtc,X_train,y_train,cv=5,scoring='accuracy')
score
```

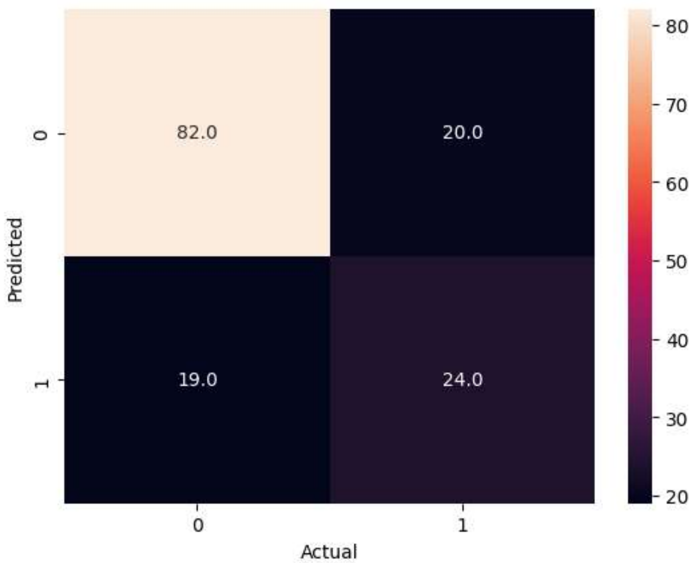
```
array([0.57758621, 0.64655172, 0.68103448, 0.70689655, 0.67826087])
```

```
dtc.score(X_test,y_test)
```

```
0.7310344827586207
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred_reg)
sns.heatmap(cm, annot=True,fmt=".1f")
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred_reg))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.80      0.81       102
         1.0       0.55      0.56      0.55        43

    accuracy                           0.73       145
   macro avg       0.68      0.68      0.68       145
weighted avg       0.73      0.73      0.73       145
```

OPTIMIZATION AND FINE TUNING

```
from sklearn.model_selection import GridSearchCV
params =  {
    'min_samples_leaf': [3, 4, 5],
    'max_depth': [3, 4, 5,6]
}
#
# Create gridsearch instance
```

```
#
grid = GridSearchCV(estimator=dtc,
                    param_grid=params,
                    cv=10,
                    n_jobs=1,
                    verbose=2)
#
# Fit the model
#
grid.fit(X_train, y_train)
#
# Assess the score
#
grid.best_score_, grid.best_params_
```

```
Fitting 10 folds for each of 12 candidates, totalling 120 fits
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=3, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.1s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=3; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=4; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=5; total time=   0.0s
[CV] END ....................max_depth=4, min_samples_leaf=5; total time=   0.0s
```

```
dtc1 = DecisionTreeClassifier(max_depth=5,min_samples_leaf=4)
dtc1.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_leaf=4)
```

```
y_pred_opt=dtc1.predict(X_test)
```

```
print(classification_report(y_test,y_pred_opt))
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.81      0.82       102
         1.0       0.57      0.58      0.57        43

    accuracy                           0.74       145
   macro avg       0.69      0.70      0.70       145
weighted avg       0.75      0.74      0.75       145
```

ALTERNATIVE MODEL

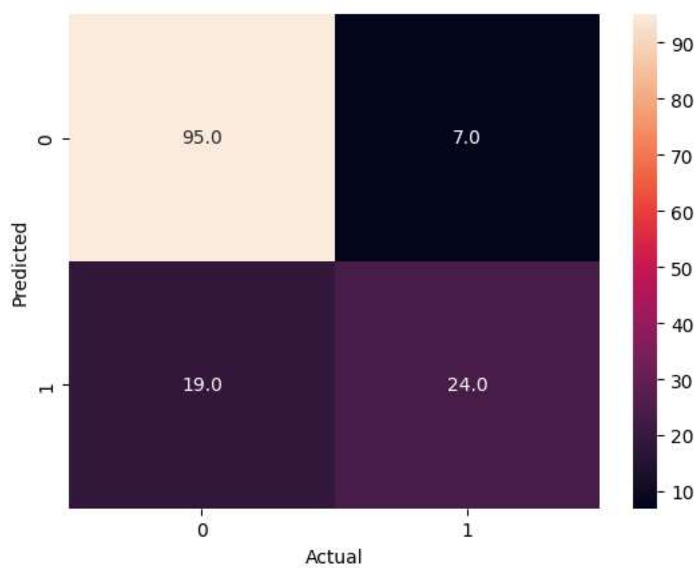
LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()

log_reg.fit(X_train, y_train)
y_pred_lr = log_reg.predict(X_test)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred_lr)
sns.heatmap(cm, annot=True,fmt=".1f")
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()
```




```
print(classification_report(y_test,y_pred_lr))
```

```
              precision    recall  f1-score   support

         0.0       0.83      0.93      0.88       102
         1.0       0.77      0.56      0.65        43

    accuracy                           0.82       145
   macro avg       0.80      0.74      0.76       145
weighted avg       0.82      0.82      0.81       145
```

Created in Deepnote