2-notebookfile-gmeo8hx8 02070241

2-notebook2tbsg-ernuefdm 02070274

```
import numpy as np
import pandas as pd
import os,sys
import matplotlib
matplotlib.rcParams['figure.dpi'] = 100
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import gridspec
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import *
from sklearn.metrics.cluster import *
from sklearn.metrics.pairwise import *
from sklearn.model_selection import *
from sklearn.feature_selection import *
from sklearn.pipeline import Pipeline
from sklearn.decomposition import *
from sklearn.manifold import *
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import *
from sklearn.neighbors import KDTree
from sklearn.tree import *
from sklearn.ensemble import *
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.cluster import *
from sklearn.metrics.cluster import adjusted_mutual_info_score
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics.cluster import homogeneity_completeness_v_measure
from sklearn.metrics.cluster import fowlkes_mallows_score
from sklearn.datasets import *
from sklearn.cluster import DBSCAN
from scipy.spatial import *
from scipy.interpolate import interp1d,interp2d,griddata
from scipy.stats import norm, ttest_ttest_ind
from scipy.stats import *
from scipy.optimize import *
from scipy.special import *
from statsmodels.stats.multicomp import *
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.gofplots import qqplot
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf,seasonal_decompose
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.arima.model import ARIMA,ARIMAResults
from statsmodels.tsa.ar_model import AutoReg,AR
from statsmodels.tsa.stattools import acf,pacf,adfuller,kpss,ccf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.base.tsa_model import TimeSeriesModel
from statsmodels import api as sm
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import os,kagglehub,shutil
path = kagglehub.dataset_download("uciml/pima-indians-diabetes-database")
print("Path to dataset files:", path)
os.chdir('/root/.cache/kagglehub/datasets/uciml/pima-indians-diabetes-database/versions/1')
os.listdir('.')
```

```
Path to dataset files: /root/.cache/kagglehub/datasets/uciml/pima-indians-diabetes-database/versions/1
['diabetes.csv']
```

Import Data

```
!apt-get install -y -q tree
!tree -L 2 /root/.cache/kagglehub/datasets/uciml/pima-indians-diabetes-database/versions/1
```

```
/root/.cache/kagglehub/datasets/uciml/pima-indians-diabetes-database/versions/1
└── diabetes.csv

1 directory, 1 file
```

```
!cp /root/.cache/kagglehub/datasets/uciml/pima-indians-diabetes-database/versions/1/diabetes.csv .
!ls -lrt
```

```
total 24
drwxr-xr-x 2 root root 4096 Sep 17 02:02 sample_data
-rw-r-----1 root root 23878 Sep 17 02:02 diabetes.csv
```

```
df = pd.read_csv('/root/.cache/kagglehub/datasets/uciml/pima-indians-diabetes-database/versions/1/diabetes.csv')
df.shape,df.dtypes,df.nunique(),df.isnull().sum(),df.nunique()/df.shape[0]
```

```
(Pregnancies                   int64
 Glucose                       int64
 BloodPressure                 int64
 SkinThickness                 int64
 Insulin                       int64
 BMI                         float64
 DiabetesPedigreeFunction    float64
 Age                           int64
 Outcome                       int64
 dtype: object,
 Pregnancies                   17
 Glucose                      136
 BloodPressure                 47
 SkinThickness                 51
 Insulin                      186
 BMI                          248
 DiabetesPedigreeFunction     517
 Age                           52
 Outcome                        2
 dtype: int64,
 Pregnancies                   0
 Glucose                       0
 BloodPressure                 0
 SkinThickness                 0
 Insulin                       0
 BMI                           0
 DiabetesPedigreeFunction      0
 Age                           0
 Outcome                       0
 dtype: int64,
 Pregnancies                   0.070773
 Glucose                       0.565885
 BloodPressure                 0.195513
 SkinThickness                 0.212370
 Insulin                       0.773779
 BMI                           1.032052
 DiabetesPedigreeFunction      2.150081
 Age                           0.216338
 Outcome                       0.008319
 dtype: float64)
```

```
df.head(5),df.tail(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
df.info()
```

```
<class pandas.core.frame.DataFrame>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 Pregnancies                 768 non-null   int64
 Glucose                     768 non-null   int64
 BloodPressure               768 non-null   int64
 SkinThickness               768 non-null   int64
 Insulin                     768 non-null   int64
 BMI                         768 non-null   float64
 DiabetesPedigreeFunction    768 non-null   float64
 Age                         768 non-null   int64
 Outcome                     768 non-null   int64
dtype: float64(2), int64(7)
memory usage: 54.0 KB
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
df.nunique()
```

```
Pregnancies                 17
Glucose                    136
BloodPressure               47
SkinThickness               51
Insulin                    186
BMI                        248
DiabetesPedigreeFunction   517
Age                         52
Outcome                      2
dtype: int64
```

```
df.isnull().sum(axis=1).sort_values(ascending=False)
```

```
0    0
1    0
2    0
3    0
4    0
    ..
763  0
764  0
765  0
766  0
767  0
dtype: int64
```

Data Preparation

```
df.columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
df['Pregnancies'] = df['Pregnancies'].astype('int')
df['Glucose'] = df['Glucose'].astype('int')
df['BloodPressure'] = df['BloodPressure'].astype('int')
df['SkinThickness'] = df['SkinThickness'].astype('int')
df['Insulin'] = df['Insulin'].astype('int')
df['BMI'] = df['BMI'].astype('float')
df['DiabetesPedigreeFunction'] = df['DiabetesPedigreeFunction'].astype('float')
df['Age'] = df['Age'].astype('int')
df['Outcome'] = df['Outcome'].astype('int')
df.shape,df.dtypes,df.nunique(),df.isnull().sum(),df.nunique()/df.shape[0]
```

```
(Pregnancies                   int64
 Glucose                       int64
 BloodPressure                 int64
 SkinThickness                 int64
 Insulin                       int64
 BMI                         float64
 DiabetesPedigreeFunction    float64
 Age                           int64
 Outcome                       int64
 dtype: object,
 Pregnancies                   17
 Glucose                      136
 BloodPressure                 47
 SkinThickness                 51
 Insulin                      186
 BMI                          248
 DiabetesPedigreeFunction     517
 Age                           52
 Outcome                        2
 dtype: int64,
 Pregnancies                   0
 Glucose                       0
 BloodPressure                 0
 SkinThickness                 0
 Insulin                       0
 BMI                           0
 DiabetesPedigreeFunction      0
 Age                           0
 Outcome                       0
 dtype: int64,
 Pregnancies                   0.070773
 Glucose                       0.565885
 BloodPressure                 0.195513
 SkinThickness                 0.212370
 Insulin                       0.773779
 BMI                           1.032052
 DiabetesPedigreeFunction      2.150081
 Age                           0.216338
 Outcome                       0.008319
 dtype: float64)
```

```
df.head(5),df.tail(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
df.info()
```

```
<class pandas.core.frame.DataFrame>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 Pregnancies                 768 non-null   int64
 Glucose                     768 non-null   int64
 BloodPressure               768 non-null   int64
 SkinThickness               768 non-null   int64
 Insulin                     768 non-null   int64
 BMI                         768 non-null   float64
 DiabetesPedigreeFunction    768 non-null   float64
 Age                         768 non-null   int64
 Outcome                     768 non-null   int64
dtype: float64(2), int64(7)
memory usage: 54.0 KB
```

Feature Extraction

```
df.pivot_table(index=['Pregnancies'],aggfunc=[np.mean,np.sum])
df.groupby(['Pregnancies']).agg({'Glucose':[np.mean,np.sum],'BloodPressure':[np.mean,np.sum]})
```

	mean																	
	Age	BMI	DiabetesPedigreeFunction	Glucose	Insulin	Outcome	Pregnancies	SkinThickness	sum									
Pregnancies									Age	BMI	DiabetesPedigreeFunction	Glucose	Insulin	Outcome	Pregnancies	SkinThickness		
0	31	30.4	0.470667	118	84	0.4	0	31.6	93	91.2	1.412000	354	252	1.2	0	94.8		
1	29	28.6	0.351000	85	66	0.0	1	29.0	29	28.6	0.351000	85	66	0.0	1	29.0		
2	32	23.3	0.672000	183	64	1.0	2	0.0	32	23.3	0.672000	183	64	1.0	2	0.0		
3	21	28.1	0.167000	89	66	0.0	3	23.0	21	28.1	0.167000	89	66	0.0	3	23.0		
4	33	43.1	2.288000	137	40	1.0	4	35.0	33	43.1	2.288000	137	40	1.0	4	35.0		
5	44	19.9	0.971000	116	74	0.0	5	29.0	44	19.9	0.971000	116	74	0.0	5	29.0		
6	51	33.6	0.587000	78	50	1.0	6	0.0	51	33.6	0.587000	78	50	1.0	6	0.0		
7	54	34.6	0.756167	197	70	0.666667	7	15.0	162	103.8	2.268500	591	210	2.0	7	45.0		
8	58	35.9	0.884000	171	72	1.0	8	0.0	58	35.9	0.884000	171	72	1.0	8	0.0		
9	57	34.9	0.918000	171	74	1.000000	9	0.0	171	104.7	2.754000	513	222	3.0	9	0.0		
10	63	30.8	0.429500	139	68	0.500000	10	46.0	126	61.6	0.859000	278	136	1.0	10	92.0		
11	46	32.4	0.519000	127	62	0.000000	11	0.0	46	32.4	0.519000	127	62	0.0	11	0.0		
12	60	32.9	0.171000	101	76	0.000000	12	48.0	60	32.9	0.171000	101	76	0.0	12	48.0		
13	43	36.8	0.340000	122	70	0.000000	13	27.0	43	36.8	0.340000	122	70	0.0	13	27.0		
14	62	26.2	0.245000	121	72	0.000000	14	23.0	62	26.2	0.245000	121	72	0.0	14	23.0		
15	30	30.1	0.349000	126	60	1.000000	15	0.0	30	30.1	0.349000	126	60	1.0	15	0.0		
16	23	30.4	0.315000	93	70	0.000000	16	31.0	23	30.4	0.315000	93	70	0.0	16	31.0		
17	81	67.1	2.420000	199	122	1.000000	17	99.0	81	67.1	2.420000	199	122	1.0	17	99.0		

Rows: 17 Columns: 10

Feature Selection

```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
feature_scores = pd.Series(model.feature_importances_, index=X.columns)
print(feature_scores.sort_values(ascending=False))
```

```
[0.09922889 0.24755853 0.08893338 0.07910166 0.07772008 0.14636698
 0.12092715 0.14216323]
Glucose                     0.247559
BMI                         0.146367
Age                         0.142163
DiabetesPedigreeFunction    0.120927
Pregnancies                 0.099229
BloodPressure               0.088933
SkinThickness               0.079102
Insulin                     0.077720
dtype: float64
```

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

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

Modeling

```
# pip install pycaret
!apt-get -y -q install -qq -y libsndfile1 libsox-fmt-all
!pip install -q -U -q pycaret[full] numba==0.55.1 llvmlite==0.39.1 "numpy<1.23" "pandas<1.5" "scikit-learn~=1.0.1" "pyyaml>=5.4" "markupsafe==2.0.1" "nltk==3.6.7" "shap==0.42.1" "yellowbrick>=1.4" "plotly>=5.10.0" "kaleido>=0.2.1" "schemdraw>=0.14" "plotly-resampler>=0.8.3.1" "scipy<1.11.0" "tsfresh>=0.20.0" "sklearn-rvs>=0.1.1" "sklearn-extra>=1.0.1" "imbalanced-learn>=0.12.0" "imblearn-extensions>=0.1.1" "matplotlib>=3.5.3" "scikit-plot>=0.3.7" "yellowbrick>=1.4" "plotly>=5.10.0" "kaleido>=0.2.1" "schemdraw>=0.14" "plotly-resampler>=0.8.3.1" "scipy<1.11.0" "tsfresh>=0.20.0" "sklearn-rvs>=0.1.1" "sklearn-extra>=1.0.1" "imbalanced-learn>=0.12.0" "imblearn-extensions>=0.1.1" "matplotlib>=3.5.3" "scikit-plot>=0.3.7"
```

```
Reading package lists...
Building dependency tree...
Reading state information...
libsndfile1-dev is set to manual mode
libsox-fmt-all is set to manual mode
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
ERROR: Invalid requirement: 'numpy<1.23,pandas<1.5,scikit-learn~=1.0.1,pyyaml>=5.4,markupsafe==2.0.1,nltk==3.6.7,shap==0.42.1,yellowbrick>=1.4,plotly>=5.10.0,kaleido>=0.2.1,schemdraw>=0.14,plotly-resampler>=0.8.3.1,scipy<1.11.0,tsfresh>=0.20.0,sklearn-rvs>=0.1.1,sklearn-extra>=1.0.1,imbalanced-learn>=0.12.0,imblearn-extensions>=0.1.1,matplotlib>=3.5.3,scikit-plot>=0.3.7,yellowbrick>=1.4,plotly>=5.10.0,kaleido>=0.2.1,schemdraw>=0.14,plotly-resampler>=0.8.3.1,scipy<1.11.0,tsfresh>=0.20.0,sklearn-rvs>=0.1.1,sklearn-extra>=1.0.1,imbalanced-learn>=0.12.0,imblearn-extensions>=0.1.1,matplotlib>=3.5.3,scikit-plot>=0.3.7" (from "numpy<1.23,pandas<1.5,scikit-learn~=1.0.1,pyyaml>=5.4,markupsafe==2.0.1,nltk==3.6.7,shap==0.42.1,yellowbrick>=1.4,plotly>=5.10.0,kaleido>=0.2.1,schemdraw>=0.14,plotly-resampler>=0.8.3.1,scipy<1.11.0,tsfresh>=0.20.0,sklearn-rvs>=0.1.1,sklearn-extra>=1.0.1,imbalanced-learn>=0.12.0,imblearn-extensions>=0.1.1,matplotlib>=3.5.3,scikit-plot>=0.3.7")
```

Decision Tree

```
# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
print("Training score: %f" %dt.score(X_train,Y_train))
print("Testing score: %f" %dt.score(X_test,Y_test))
Y_pred = dt.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 1.000000
Testing score: 0.727273
Accuracy score: 0.727273
Recall score: 0.604938
Precision score: 0.571429
F1 score: 0.588000
```

```
# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
print("Training score: %f" %dt.score(X_train,Y_train))
print("Testing score: %f" %dt.score(X_test,Y_test))
Y_pred = dt.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 1.000000
Testing score: 0.727273
Accuracy score: 0.727273
Recall score: 0.604938
Precision score: 0.571429
F1 score: 0.588000
```

```
# Decision Tree
dt = DecisionTreeClassifier(criterion='entropy',max_depth=3,min_samples_leaf=5,min_samples_split=2)
dt.fit(X_train,Y_train)
print("Training score: %f" %dt.score(X_train,Y_train))
print("Testing score: %f" %dt.score(X_test,Y_test))
Y_pred = dt.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 0.777778
Testing score: 0.753247
Accuracy score: 0.753247
Recall score: 0.666667
Precision score: 0.611111
F1 score: 0.637681
```

```
# Decision Tree
dt = DecisionTreeClassifier(criterion='entropy',max_depth=3,min_samples_leaf=5,min_samples_split=2)
dt.fit(X_train,Y_train)
print("Training score: %f" %dt.score(X_train,Y_train))
print("Testing score: %f" %dt.score(X_test,Y_test))
Y_pred = dt.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 0.777778
Testing score: 0.753247
Accuracy score: 0.753247
Recall score: 0.666667
Precision score: 0.611111
F1 score: 0.637681
```

```
# Decision Tree
dt = DecisionTreeClassifier(criterion='entropy',max_depth=3,min_samples_leaf=5,min_samples_split=2)
dt.fit(X_train,Y_train)
print("Training score: %f" %dt.score(X_train,Y_train))
print("Testing score: %f" %dt.score(X_test,Y_test))
Y_pred = dt.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 0.777778
Testing score: 0.753247
Accuracy score: 0.753247
Recall score: 0.666667
Precision score: 0.611111
F1 score: 0.637681
```

Random Forest

```
# random forest(n_trees, n_trees,n_test,n_test)
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
print("Training score: %f" %rf.score(X_train,Y_train))
print("Testing score: %f" %rf.score(X_test,Y_test))
Y_pred = rf.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 1.000000
Testing score: 0.727273
Accuracy score: 0.727273
Recall score: 0.604938
Precision score: 0.571429
F1 score: 0.588000
```

```
# random forest(n_trees, n_trees,n_test,n_test)
rf = RandomForestClassifier(criterion='entropy',max_depth=3,min_samples_leaf=5,min_samples_split=2)
rf.fit(X_train,Y_train)
print("Training score: %f" %rf.score(X_train,Y_train))
print("Testing score: %f" %rf.score(X_test,Y_test))
Y_pred = rf.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 0.777778
Testing score: 0.753247
Accuracy score: 0.753247
Recall score: 0.666667
Precision score: 0.611111
F1 score: 0.637681
```

K-Nearest Neighbors

```
# k-nearest neighbors(n_trees,n_trees,n_test,n_test)
knn = KNeighborsClassifier()
knn.fit(X_train,Y_train)
print("Training score: %f" %knn.score(X_train,Y_train))
print("Testing score: %f" %knn.score(X_test,Y_test))
Y_pred = knn.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 0.777778
Testing score: 0.727273
Accuracy score: 0.727273
Recall score: 0.604938
Precision score: 0.571429
F1 score: 0.588000
```

```
# k-nearest neighbors(n_trees,n_trees,n_test,n_test)
knn = KNeighborsClassifier(n_neighbors=5,weights='distance',algorithm='auto',leaf_size=30,p=2,metric='minkowski',metric_params=None,n_jobs=None)
knn.fit(X_train,Y_train)
print("Training score: %f" %knn.score(X_train,Y_train))
print("Testing score: %f" %knn.score(X_test,Y_test))
Y_pred = knn.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 1.000000
Testing score: 0.727273
Accuracy score: 0.727273
Recall score: 0.604938
Precision score: 0.571429
F1 score: 0.588000
```



```
# k-nearest neighbors(n_trees,n_trees,n_test,n_test)
knn = KNeighborsClassifier(n_neighbors=5,weights='distance',algorithm='auto',leaf_size=30,p=2,metric='minkowski',metric_params=None,n_jobs=None)
knn.fit(X_train,Y_train)
print("Training score: %f" %knn.score(X_train,Y_train))
print("Testing score: %f" %knn.score(X_test,Y_test))
Y_pred = knn.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 1.000000
Testing score: 0.727273
Accuracy score: 0.727273
Recall score: 0.604938
Precision score: 0.571429
F1 score: 0.588000
```

Logistic Regression

```
# logistic regression(n_trees,n_trees,n_test,n_test)
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train,Y_train)
print("Training score: %f" %lr.score(X_train,Y_train))
print("Testing score: %f" %lr.score(X_test,Y_test))
Y_pred = lr.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 0.777778
Testing score: 0.753247
Accuracy score: 0.753247
Recall score: 0.666667
Precision score: 0.611111
F1 score: 0.637681
```

```
# logistic regression(n_trees,n_trees,n_test,n_test)
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train,Y_train)
print("Training score: %f" %lr.score(X_train,Y_train))
print("Testing score: %f" %lr.score(X_test,Y_test))
Y_pred = lr.predict(X_test)
print("Accuracy score: %f" %accuracy_score(Y_test,Y_pred))
print("Recall score: %f" %recall_score(Y_test,Y_pred))
print("Precision score: %f" %precision_score(Y_test,Y_pred))
print("F1 score: %f" %f1_score(Y_test,Y_pred))
```

```
Training score: 0.777778
Testing score: 0.753247
Accuracy score: 0.753247
Recall score: 0.666667
Precision score: 0.611111
F1 score: 0.637681
```

```
test_acc = accuracy_score(test_predictions, y_test)

fn, tn, fp, tp = confusion_matrix(y_test, test_predictions).ravel()
specificity = tn / (tn + fp) if (tn + fp) > 0 else 0.0
fp_rate = fp / (fp + tn) if (fp + tn) > 0 else 0.0
fn_rate = fn / (fn + tp) if (fn + tp) > 0 else 0.0
model_name_mapping = {'gaussian': 'Gaussian',
'multinomial': 'Multinomial',
'bernoulli': 'Bernoulli',
'complement': 'Complement'}
display_name = model_name_mapping.get(variant, variant)
print(f"\n{display_name} Results:")
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Specificity: {specificity:.4f}")
print(f"False Positive Rate: {fp_rate:.4f}")
print(f"False Negative Rate: {fn_rate:.4f}")
print(f"TP: {tp}, TN: {tn}, FP: {fp}, FN: {fn}")
```

```
Gaussian Results:
Test Accuracy: 0.7500
Specificity: 0.7500
False Positive Rate: 0.2500
False Negative Rate: 0.2500
TP: 6, TN: 6, FP: 2, FN: 2

Multinomial Results:
Test Accuracy: 0.8125
Specificity: 0.8750
False Positive Rate: 0.1250
False Negative Rate: 0.2500
TP: 6, TN: 7, FP: 1, FN: 2

Bernoulli Results:
Test Accuracy: 0.8125
Specificity: 0.8750
False Positive Rate: 0.1250
False Negative Rate: 0.2500
TP: 6, TN: 7, FP: 1, FN: 2

Complement Results:
Test Accuracy: 0.7500
Specificity: 0.7500
False Positive Rate: 0.2500
False Negative Rate: 0.2500
TP: 6, TN: 6, FP: 2, FN: 2
```

```
# Logistic Regression Results
print("\n" + "="*50)
print("Logistic Regression Results")
print("="*50)

logit = LogisticRegression(max_iter=1000, random_state=42)
logit.fit(X_train, y_train)
logit_predictions = logit.predict(X_test)
logit_acc = accuracy_score(y_test, logit_predictions)

print(f"Test Accuracy: {logit_acc:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, logit_predictions))

logit_cm = confusion_matrix(y_test, logit_predictions)
print("\nConfusion Matrix:")
print(logit_cm)

plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'DejaVu Sans']
plt.rcParams['axis.unicode_minus'] = False

plt.figure(figure_size=(6, 5))
sns.heatmap(logit_cm, annot=True, fmt='d',
cmap='Blue',
cbar_kws={'shrink': 0.8},
square=True,
linewidths=2,
annot_kws={'size': 20, 'weight': 'bold'})
plt.title('Logistic Regression\n', fontsize=14, fontweight='bold')
plt.ylabel('True Label', fontsize=12)
plt.xlabel('Predicted Label', fontsize=12)
plt.tick_params(axis='both', which='major', labelsize=11)
plt.tight_layout()
plt.show()
```

```
==================================================
Logistic Regression Results
==================================================
Test Accuracy: 0.7500

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.75      0.75         8
           1       0.75      0.75      0.75         8

    accuracy                           0.75        16
   macr o avg       0.75      0.75      0.75        16
weighted avg       0.75      0.75      0.75        16

Confusion Matrix:
[[6 2]
 [2 6]]
```

```
# Create comparison table
comparison_df = pd.DataFrame(results)
comparison_df = comparison_df.sort_values('Test Accuracy', ascending=False)

print("\n" + "="*80")
print("COMPLETE MODEL COMPARISON")
print("="*80")
print(comparison_df.round(4).to_string(index=False))
```

Data visualization

- Bar plot Acc,Precision,Recall,F1

```
# Create visualization
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'DejaVu Sans']
plt.rcParams['axis.unicode_minus'] = False

plt.figure(figure_size=(12, 6))
sns.barplot(data=comparison_df, x='Model', y='Test Accuracy', hue='Variant', palette='Blues')
plt.title('Accuracy Score\n', fontsize=14, fontweight='bold')
plt.ylabel('Test Accuracy', fontsize=12)
plt.xlabel('Model Variant', fontsize=12)
plt.legend(title='Variant', fontsize=11, title_fontsize=12)
plt.tick_params(axis='x', rotation=45, labelsize=11)
plt.tick_params(axis='y', labelsize=11)
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', labelsize=11)
plt.tight_layout()
plt.show()
```

```
# Create visualization
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'DejaVu Sans']
plt.rcParams['axis.unicode_minus'] = False

plt.figure(figure_size=(12, 6))
sns.barplot(data=comparison_df, x='Model', y='Precision', hue='Variant', palette='Greens')
plt.title('Precision Score\n', fontsize=14, fontweight='bold')
plt.ylabel('Precision', fontsize=12)
plt.xlabel('Model Variant', fontsize=12)
plt.legend(title='Variant', fontsize=11, title_fontsize=12)
plt.tick_params(axis='x', rotation=45, labelsize=11)
plt.tick_params(axis='y', labelsize=11)
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', labelsize=11)
plt.tight_layout()
plt.show()
```

```
# Create visualization
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'DejaVu Sans']
plt.rcParams['axis.unicode_minus'] = False

plt.figure(figure_size=(12, 6))
sns.barplot(data=comparison_df, x='Model', y='Recall', hue='Variant', palette='Purples')
plt.title('Recall Score\n', fontsize=14, fontweight='bold')
plt.ylabel('Recall', fontsize=12)
plt.xlabel('Model Variant', fontsize=12)
plt.legend(title='Variant', fontsize=11, title_fontsize=12)
plt.tick_params(axis='x', rotation=45, labelsize=11)
plt.tick_params(axis='y', labelsize=11)
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', labelsize=11)
plt.tight_layout()
plt.show()
```

```
# Create visualization
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'DejaVu Sans']
plt.rcParams['axis.unicode_minus'] = False

plt.figure(figure_size=(12, 6))
sns.barplot(data=comparison_df, x='Model', y='F1-Score', hue='Variant', palette='Reds')
plt.title('F1-Score\n', fontsize=14, fontweight='bold')
plt.ylabel('F1-Score', fontsize=12)
plt.xlabel('Model Variant', fontsize=12)
plt.legend(title='Variant', fontsize=11, title_fontsize=12)
plt.tick_params(axis='x', rotation=45, labelsize=11)
plt.tick_params(axis='y', labelsize=11)
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', labelsize=11)
plt.tight_layout()
plt.show()
```

- Auc Score

```
scores_df = ROC_results[['Variant', 'AUC']].drop_duplicates()
scores_df = scores_df.sort_values('AUC', ascending=False)
scores_df['AUC'] = scores_df['AUC'].round(4)

print("AUC SCORES RANKING:")
print("="*40")
for idx, row in scores_df.iterrows:
    print(f"{row['Variant']:<12} {row['AUC']:.4}")
print("="*40")
print("Best Variant: {scores_df.iloc[0]['Variant']}")
print("Best AUC: {scores_df.iloc[0]['AUC']:.4}")
print("Worst Variant: {scores_df.iloc[-1]['Variant']}")
print("Worst AUC: {scores_df.iloc[-1]['AUC']:.4}")
print("Range: {scores_df.iloc[0]['AUC'] - scores_df.iloc[-1]['AUC']:.4}")
```

```
AUC SCORES RANKING:
========================
Complement   1.0
Gaussian     1.0
Multinomial  1.0
Bernoulli    1.0
========================
Best Variant: {scores_df.iloc[0]['Variant']}
Best AUC: {scores_df.iloc[0]['AUC']:.4}
Worst Variant: {scores_df.iloc[-1]['Variant']}
Worst AUC: {scores_df.iloc[-1]['AUC']:.4}
Range: {scores_df.iloc[0]['AUC'] - scores_df.iloc[-1]['AUC']:.4}
```

```
# Create ROC curve visualization
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'DejaVu Sans']
plt.rcParams['axis.unicode_minus'] = False

plt.figure(figure_size=(8, 8))
for variant in ['Gaussian', 'Multinomial', 'Bernoulli', 'Complement']:
    variant_data = ROC_results[ROC_results['Variant'] == variant]
    if len(variant_data) > 0:
        fpr = variant_data['FPR']
        tpr = variant_data['TPR']
        auc = variant_data['AUC'].iloc[0]
        plt.plot(fpr, tpr, linewidth=2, label=f'{variant} (AUC = {auc:.4})')

plt.plot([0, 1], [0, 1], 'k--', linewidth=1, alpha=0.5, label='Random Classifier')

plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC Curves\n', fontsize=14, fontweight='bold')
plt.legend(fontsize=11, loc='lower right')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

- etc2

scores=[Best Accuracy score, Precision score, F1 score, auc-roc score, Random Forest Feature Importance, Feature Importance, Deployment]

- Deployment

```
$ ngrok http 8501
ngrok
Hello world!
Session Status                online
Account                       Free (Upgrade to ngrok.com/pricing)
Version                       3.3.0
Latency                       -
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://unfilmed-ungovernably-ilsa.ngrok-free.app -> http://localhost:8501
Traffic                       0 (0 conn)
Error                         failed to connect to http://localhost:8501: dial tcp [::1]:8501: connect: connection refused
$ pip install streamlit
Collecting streamlit
  Downloading streamlit-1.52.1-py3-none-any.whl (8.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 2.6 MB/s eta 0:00:00
Collecting altair!=5.4.0,!=5.4.1,<6,>=4.0
  Downloading altair-5.5.0-py3-none-any.whl (731 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.2/731.2 kB 1.7 MB/s eta 0:00:00
Collecting blinker<1.5.0,>=1.0.0
  Downloading blinker-1.4.0.tar.gz (111 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 111.1/111.1 kB 1.6 MB/s eta 0:00:00
Collecting cachetools<6,>=4.0
  Downloading cachetools-5.5.2-py3-none-any.whl (10.1 kB)
Collecting click<8.1.0,>=7.0
  Downloading click-8.0.4.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Collecting dataclasses-dataclass-wizard<0.22,>=0.20.1
  Downloading dataclasses_dataclass_wizard-0.21.1.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Collecting gitdb<4,>=3.0.1
  Downloading gitdb-4.0.12.tar.gz (394 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 394.1/394.1 kB 1.7 MB/s eta 0:00:00
Collecting GitPython!=3.1.19,<3,>=3.0.7
  Downloading GitPython-3.1.45.tar.gz (2.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 1.7 MB/s eta 0:00:00
Collecting numpy<2,>=1.19.3
  Downloading numpy-1.26.4.tar.gz (15.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.8/15.8 MB 1.7 MB/s eta 0:00:00
Collecting packaging<25,>=20
  Downloading packaging-24.2-py3-none-any.whl (65 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.451/65.451 kB 1.7 MB/s eta 0:00:00
Collecting pandas<3,>=1.3.0
  Downloading pandas-2.2.3.tar.gz (4.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.4/4.4 MB 1.7 MB/s eta 0:00:00
Collecting pillow<12,>=7.1.0
  Downloading pillow-11.0.0.tar.gz (47.0 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.0/47.0 kB 1.7 MB/s eta 0:00:00
Collecting protobuf<6,>=3.20
  Downloading protobuf-5.29.3.tar.gz (424 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 424.3/424.3 kB 1.7 MB/s eta 0:00:00
Collecting pyarrow<19,>=7.0
  Downloading pyarrow-18.1.0.tar.gz (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 1.7 MB/s eta 0:00:00
Collecting pydeck<0.9.1b0,>=0.8.0b4
  Downloading pydeck-0.9.1b0.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Collecting requests<2,>=2.27
  Downloading requests-2.32.3.tar.gz (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 1.7 MB/s eta 0:00:00
Collecting tenacity<8,>=8.1.0
  Downloading tenacity-8.2.3.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Collecting toml<2,>=0.10.1
  Downloading toml-0.10.2.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Collecting tornado<6.0.3,>=6.0.3
  Downloading tornado-6.0.3.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Collecting typing-extensions<4,>=4.3.0
  Downloading typing_extensions-4.12.2.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Collecting watchdog<2,>=2.1.5
  Downloading watchdog-2.1.9.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 1.7 MB/s eta 0:00:00
Requirement already satisfied: jinja2<3.1.0,>=3.0.0 in /usr/local/lib/python3.12/dist-packages (from altair!=5.4.0,!=5.4.1,<6,>=4.0) (3.1.3)
Requirement already satisfied: jsonschema<4.23.0,>=3.0.0 in /usr/local/lib/python3.12/dist-packages (from altair!=5.4.0,!=5.4.1,<6,>=4.0) (4.23.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.12/dist-packages (from altair!=5.4.0,!=5.4.1,<6,>=4.0) (2024.10.1)
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.12/dist-packages (from altair!=5.4.0,!=5.4.1,<6,>=4.0) (1.29.1)
Requirement already satisfied: smmap<5,>=3.0.1 in /usr/local/lib/python3.12/dist-packages (from gitdb<4,>=3.0.1) (3.0.5)
Requirement already satisfied: gunicorn<23,>=19.9.0 in /usr/local/lib/python3.12/dist-packages (from dataclasses-dataclass-wizard<0.22,>=0.20.1) (22.0.0)
Requirement already satisfied: python-dateutil<2.9.0,>=2.7.0 in /usr/local/lib/python3.12/dist-packages (from dataclasses-dataclass-wizard<0.22,>=0.20.1) (2.8.2)
Requirement already satisfied: pytz<2,>=2017.3 in /usr/local/lib/python3.12/dist-packages (from dataclasses-dataclass-wizard<0.22,>=0.20.1) (2025.2)
Requirement already satisfied: PyYAML<6,>=5.1.0 in /usr/local/lib/python3.12/dist-packages (from dataclasses-dataclass-wizard<0.22,>=0.20.1) (6.0.2)
Requirement already satisfied: typing-extensions<2,>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from dataclasses-dataclass-wizard<0.22,>=0.20.1) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<2,>=2.27) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<2,>=2.27) (3.10)
Requirement already satisfied: urllib3<2.2.0,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<2,>=2.27) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<2,>=2.27) (2025.1.31)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2<3.1.0,>=3.0.0) (2.1.5)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-packages (from jsonschema<4.23.0,>=3.0.0) (25.3.0)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/dist-packages (from jsonschema<4.23.0,>=3.0.0) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.12/dist-packages (from jsonschema<4.23.0,>=3.0.0) (0.22.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil<2.9.0,>=2.7.0) (1.17.0)
Requirement already satisfied: numpy>=1.22.4; python_version<"3.11" in /usr/local/lib/python3.12/dist-packages (from narwhals>=1.14.2) (1.26.4)
Requirement already satisfied: zipp>3.1.0; python_version<"3.10" in /usr/local/lib/python3.12/dist-packages (from importlib-resources>=1.4.0; python_version<"3.9"; platform_machine!="s390x"; platform_machine!="ppc64le"; platform_machine!="x86_64"; platform_machine!="aarch64"; platform_machine!="arm64"; platform_machine!="aarch32"; platform_machine!="arm"; platform_machine!="riscv64"; platform_machine!="loongarch64"; platform_machine!="mips64"; platform_machine!="ppc64"; platform_machine!="s390"; platform_machine!="sparc"; platform_machine!="wasm32"; platform_machine!="wasm64"; platform_machine!="zos"; platform_machine!="cygwin"; platform_machine!="emscripten"; platform_machine!="wasi"; platform_machine!="win32"; platform_machine!="aix"; platform_machine!="msys"; platform_machine!="haiku"; platform_machine!="sunos5"; platform_machine!="darwin"; platform_machine!="freebsd"; platform_machine!="openbsd"; platform_machine!="dragonfly"; platform_machine!="gnu0"; platform_machine!="gnukfreebsd"; platform_machine!="neuter"; platform_machine!="hurd"; platform_machine!="linux"; platform_machine!="nacl"; platform_machine!="pnacl"; platform_machine!="solaris
```