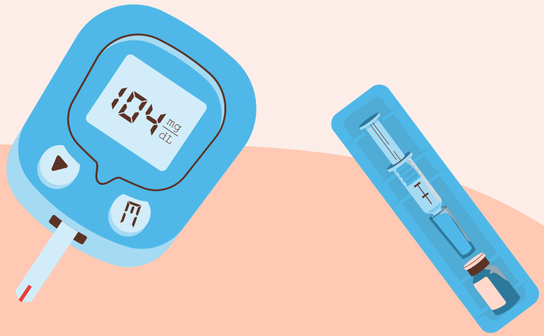
PREDICTING DIABETES: A MACHINE LEARNING APPROACH



AGENDA:

- Introduction
- Data Collection
- DATA EXPLORING & CLEANING
- Features extraction
- Data Preprocessing
- Data splitting into training and testing sets
- Model selection
- GUI
- Data visualization

INTRODUCTION:

Diabetes is a chronic disease that affects millions of people worldwide. Early detection and management of diabetes are crucial for preventing complications such as heart disease, kidney failure, and blindness. Machine learning has emerged as a powerful tool for predicting and managing diabetes. By analyzing large amounts of data from electronic health records, wearable devices, and other sources, machine learning algorithms can identify patterns and predict the likelihood of developing diabetes.

DATA COLLECTING:

To train our machine learning model, we collected data . The data included gender, age, hypertension , heart disease , smoking history , bmi , HbA1c level and blood glucose level

DATA EXPLORING & CLEANING

Exploring and cleaning data are important steps in the data analysis process. In order to get accurate insights from data, it is essential to understand the data and ensure that it is clean and reliable.

FEATURES EXTRACTION:

There are several features that can be used to predict whether a patient is likely to have diabetes or not.

extract features

```
target = dataset.iloc[:,-1]
target.head()
```

[7]

```
0    0
1    0
2    0
3    0
4    0
Name: diabetes, dtype: int64
```

```
data = dataset.iloc[:,:-1]
data.head()
```

[8]

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level
0	Female	80.0	0	1	never	25.19	6.6	140
1	Female	54.0	0	0	No Info	27.32	6.6	80
2	Male	28.0	0	0	never	27.32	5.7	158
3	Female	36.0	0	0	current	23.45	5.0	155
4	Male	76.0	1	1	current	20.14	4.8	155

DATA PREPROCESSING:

Before training our model, we needed to preprocess the data to ensure its quality and consistency. We removed any missing or irrelevant data, standardized the remaining data, and split it into training and testing sets

preprocessing the data

+ Code + Markdown

```
from sklearn.preprocessing import OneHotEncoder , LabelEncoder , MinMaxScaler
```

```
data['gender'] = LabelEncoder().fit_transform(data['gender'])
data.head()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level
0	0	80.0	0	1	never	25.19	6.6	140
1	0	54.0	0	0	No Info	27.32	6.6	80
2	1	28.0	0	0	never	27.32	5.7	158
3	0	36.0	0	0	current	23.45	5.0	155
4	1	76.0	1	1	current	20.14	4.8	155

```
data['smoking_history'] = LabelEncoder().fit_transform(data['smoking_history'])
data.head()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level
0	0	80.0	0	1	4	25.19	6.6	140
1	0	54.0	0	0	0	27.32	6.6	80
2	1	28.0	0	0	4	27.32	5.7	158
3	0	36.0	0	0	1	23.45	5.0	155
4	1	76.0	1	1	1	20.14	4.8	155

```
from sklearn.preprocessing import StandardScaler
data = StandardScaler().fit_transform(data)
```

```
data.shape
```

```
(99982, 8)
```

DATA SPLITTING INTO TRAINING AND TESTING SETS:

When building a machine learning model for diabetes prediction, it is important to split the data into training and testing sets. The purpose of this is to evaluate the performance of the model on new, unseen data

split the data into train and test

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size = 0.25, random_state = 0)
```

MODEL SELECTION:

We experimented with several machine learning algorithms including KNN classifier, decision tree, and random forests , Naive bayes. After evaluating their performance on our testing set, we selected the decision tree algorithm as it had the highest accuracy.

KNN CLASSIFIER:

knn classefier

```
from sklearn.neighbors import KNeighborsClassifier
```

```
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
#The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric.
classifier.fit(x_train, y_train)
```

```
KNeighborsClassifier()
```

```
y_pred = classifier.predict(x_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

+ Code + Markdown

THE ACCURACY : 0.9613138102096336

NAIVE BAYES CLASSIFIER AND THE ACCURACY:

naive bayes Classifier

```
from sklearn.naive_bayes import BernoulliNB
```

```
BernoulliNBModel = BernoulliNB(alpha=1.0,binarize=1)
BernoulliNBModel.fit(x_train, y_train)
```

```
BernoulliNB(binarize=1)
```

```
y_pred = BernoulliNBModel.predict(x_test)
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[22380   474]
 [ 1234   908]]

0.9316690670507282
```

DECISION TREE CLASSIFIER AND THE ACCURACY:

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
DecisionTreeClassifierModel = DecisionTreeClassifier(criterion='gini',max_depth=3,random_state=33) #criterion can be entropy
DecisionTreeClassifierModel.fit(x_train, y_train)
```

```
DecisionTreeClassifier(max_depth=3, random_state=33)
```

+ Code + Markdown

```
y_pred = DecisionTreeClassifierModel.predict(x_test)
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[22854     0]
 [  708  1434]]

0.971675468074892
```

RANDOM FORESTS CLASSIFIER AND THE ACCURACY:

Random forest

```
from sklearn.ensemble import RandomForestClassifier
#from sklearn.metrics import classification_report
randomforest = RandomForestClassifier(criterion='entropy',n_estimators=12) #criterion can be gini  for classification
```

```
randomforest.fit(x_train,y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=12)
```

```
rf_prediction = randomforest.predict(x_test)
```

```
#print(classification_report(y_test,rf_prediction))
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[22677   177]
 [  790  1352]]

0.9613138102096336
```

GRIDSEARCHCV KNN:

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid={'n_neighbors':[8,9],
            'metric':['minkowski','Euclidean',],
            'p':[6,8]}
```

```
grid = GridSearchCV(KNeighborsClassifier(), param_grid, refit = True, verbose = 3,n_jobs=-1)
```

```
grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
print(grid.best_params_)
grid_predictions = grid.predict(x_test)
```

```
pd.DataFrame(grid.cv_results_)[['mean_test_score','std_test_score','params']]
```

```
accuracy=grid.best_score_
print(accuracy)
```

```
0.9625530200052547
```

THE ACCURACY:

0.9625530200052547

GRIDSEARCHCV NAIVE BAYES:

```
param={'alpha':[1.0,0.9,0.1,0.5],
       'binarize':[6,2,3]}
```

```
grid = GridSearchCV(BernoulliNB(), param, refit = True, verbose = 3,n_jobs=-1)
```

```
grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

GridSearchCV(estimator=BernoulliNB(), n_jobs=-1,
             param_grid={'alpha': [1.0, 0.9, 0.1, 0.5], 'binarize': [6, 2, 3]},
             verbose=3)
```

```
print(grid.best_params_)
grid_predictions = grid.predict(x_test)
```

```
{'alpha': 1.0, 'binarize': 2}
```

```
pd.DataFrame(grid.cv_results_)[['mean_test_score','std_test_score','params']]
```

```
accuracy=grid.best_score_
print(accuracy)
```

```
0.9600325399124363
```

THE ACCURACY:

0.9600325399124363

GRIDSEARCHCV DECISION TREE :

```
param_grid={'criterion':['entropy','gini'],
            'max_depth':[3,4,6],
           'random_state':[5,200,100]}
```

```
grid = GridSearchCV(DecisionTreeClassifier(), param_grid, refit = True, verbose = 3,n_jobs=-1)
```

```
grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits

GridSearchCV(estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': [3, 4, 6],
                         'random_state': [5, 200, 100]},
             verbose=3)
```

```
print(grid.best_params_)
grid_predictions = grid.predict(x_test)
```

```
{'criterion': 'entropy', 'max_depth': 3, 'random_state': 5}
```

```
pd.DataFrame(grid.cv_results_)[['mean_test_score','std_test_score','params']]
```

```
accuracy=grid.best_score_
print(accuracy)
```

```
0.9719280941132886
```

THE ACCURACY:

0.9719280941132886

GRIDSEARCHCV RANDOM FOREST :

```
param_grid={'criterion':['entropy','gini'],
            'n_estimators':[13,8,15]
           }
```

```
grid = GridSearchCV(RandomForestClassifier(), param_grid, refit = True, verbose = 3,n_jobs=-1)
```

```
grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

GridSearchCV(estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['entropy', 'gini'],
                         'n_estimators': [13, 8, 15]},
             verbose=3)
```

```
print(grid.best_params_)
grid_predictions = grid.predict(x_test)
```

```
{'criterion': 'entropy', 'n_estimators': 8}
```

```
accuracy=grid.best_score_
print(accuracy)
```

```
0.9698610513763863
```

THE ACCURACY:

0.9698610513763863

THE "GUI" OF OUR PROJECT


WELCOME TO DIABETES PREDICTION SYSTEM
Get Started

Please enter the following information

Gender Female

Age write...

hypertension 0

heart_disease 0

smoking_history never

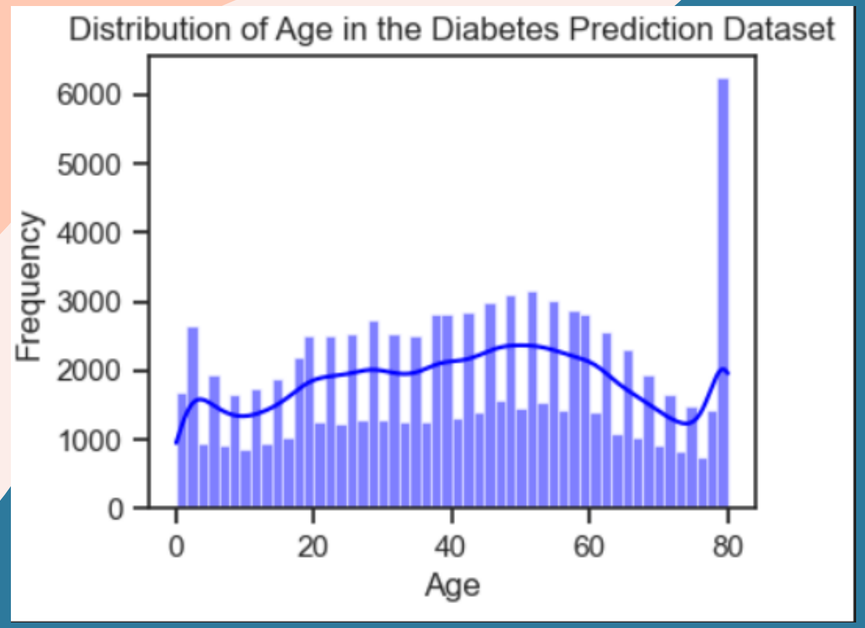
bmi write...

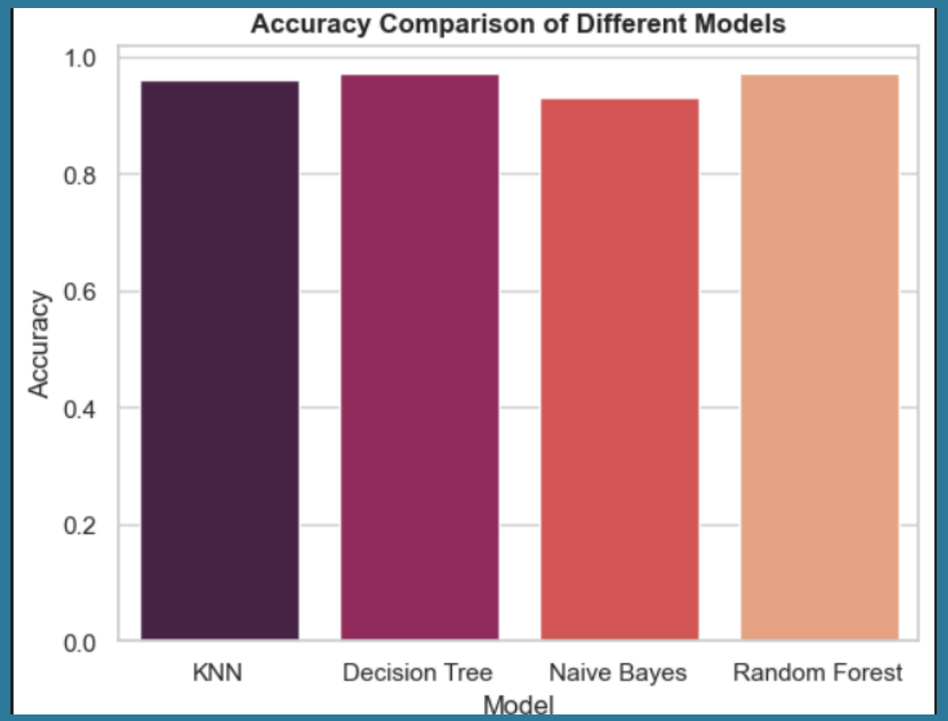
HbA1c_level write...

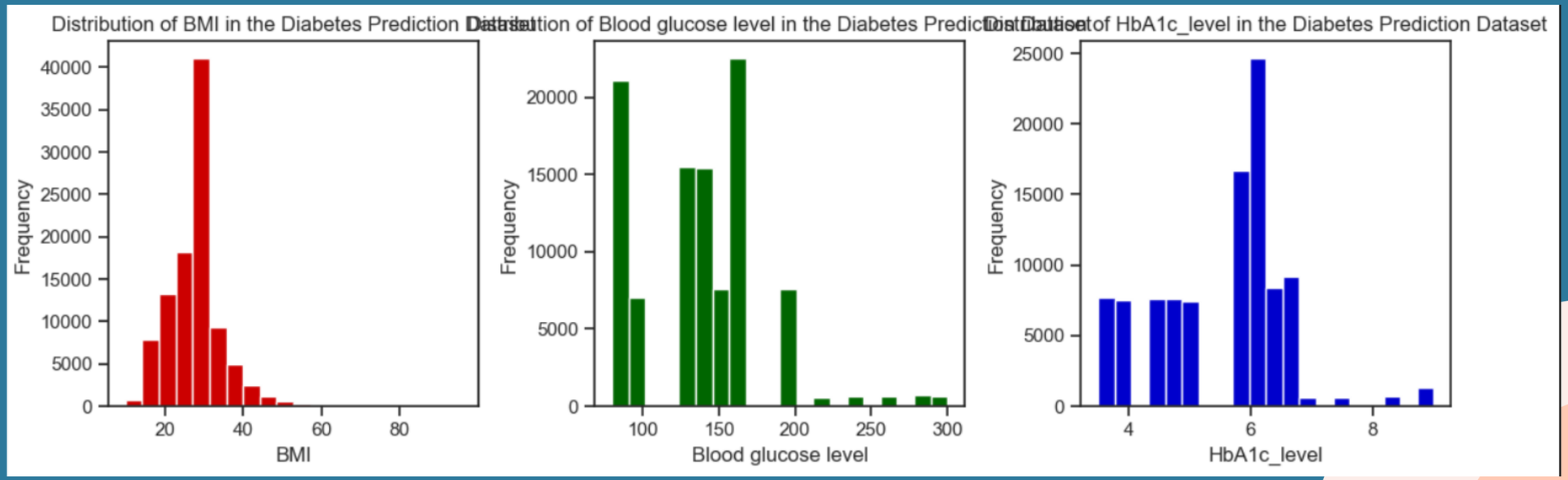
blood_glucose_level write...

predict Reset

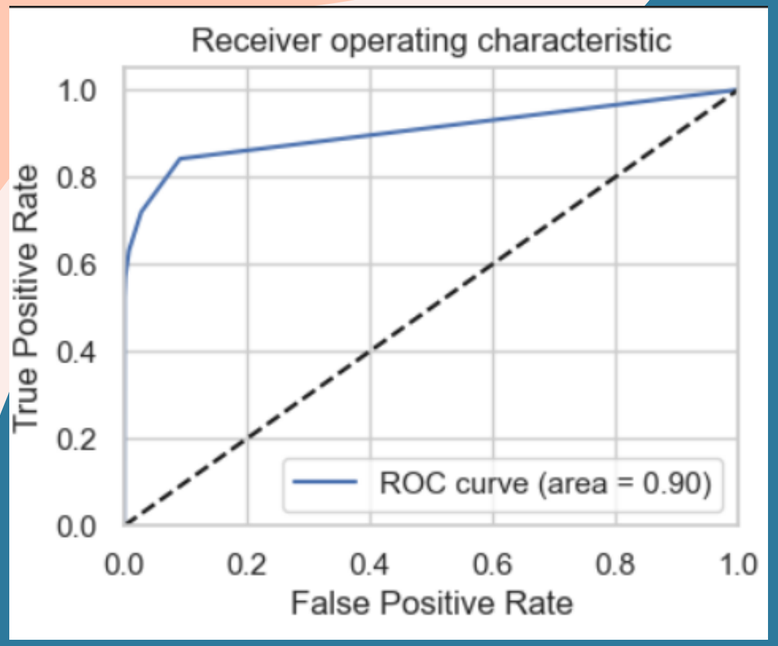
DATA VISUALIZATION:

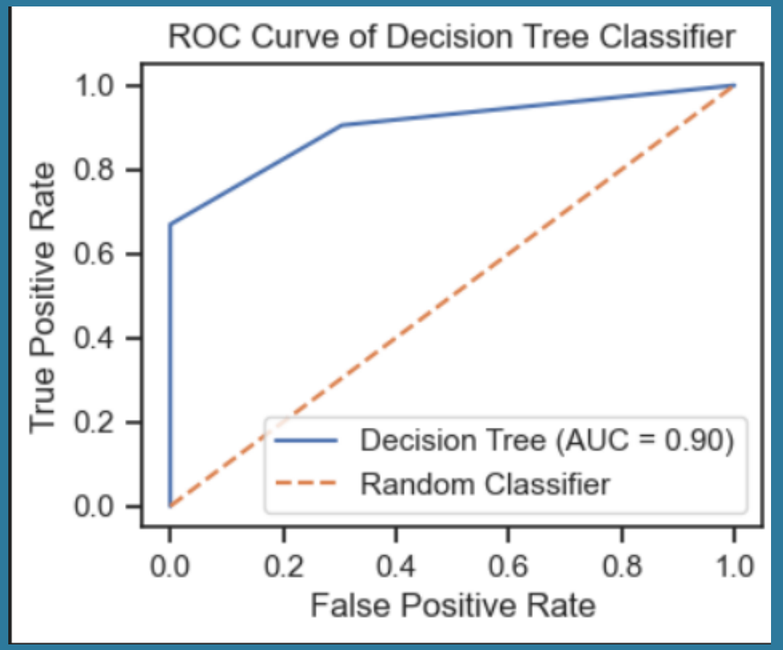


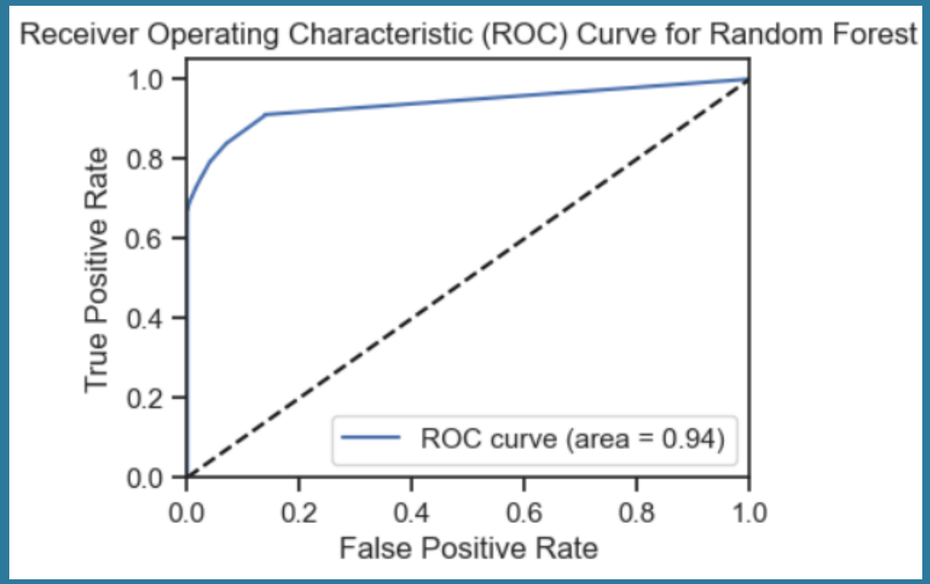


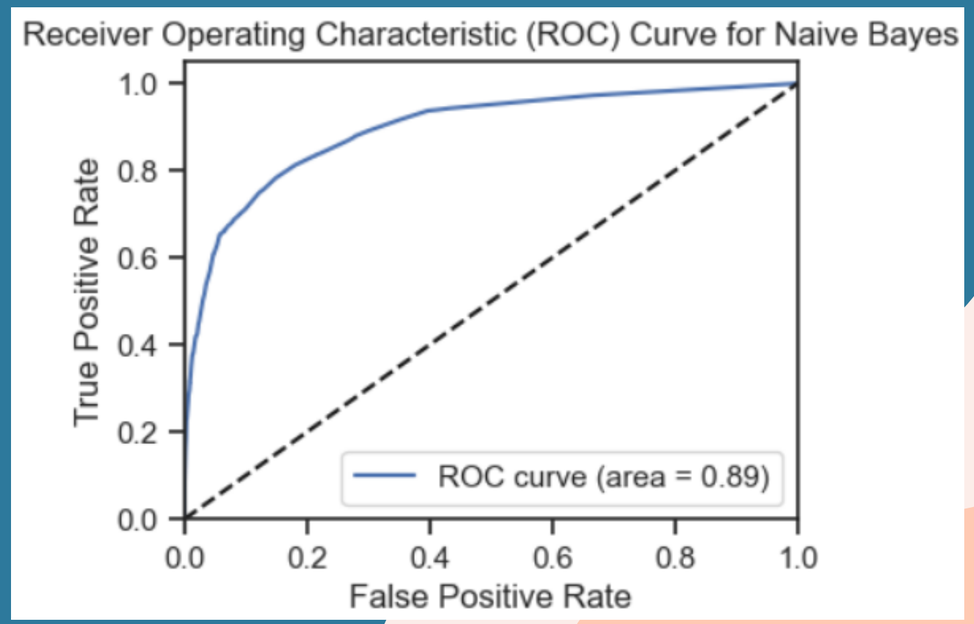


DATA VISUALIZATION OF 4 MODELS









THANK YOU!

- عبدالرحمن محمد احمد
- اميرة علاء الدين امين علي
- نانسي عصام عبدالنبي
- ندى خالد احمد

- يوسف محمد عبدالراضي
- ندى بدري غريب
- ندى احمد شحات