# Diabetes Analysis

SANJEEVI MR
21EUIT511

## Introduction

Diabetes is a health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar (also called glucose) and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin.

Without ongoing, careful management, diabetes can lead to a buildup of sugars in the blood, which can increase the risk of dangerous complications, including stroke and heart disease. So that i decide to predict using Machine Learning in Python.

## Objectives

1. Predict if person is diabetes patient or not
2. Find most indicative features of diabetes
3. Try different classification methods to find highest accuracy

## Installing Libraries

In this first step I have imported most common libraries used in python for machine learning such as

Pandas, Seaborn, Matplitlib etc.

I am using Python because if very flexible and effective programming language i ever used. I used

Python in software development field too.

```
# Import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
import os

# Modeling Libraries
from sklearn.preprocessing import QuantileTransformer
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier,
VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve,
train_test_split
```

The sklearn library is very versatile and handy and serves real-world purposes. It provides wide range of ML algorithms and Models

## Importing Data

# Import dataset

df = pd.read_csv("../input/diabetes/diabetes.csv")

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 769 entries, 0 to 768
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               769 non-null    int64
 1   Glucose                   769 non-null    int64
 2   BloodPressure             769 non-null    int64
 3   SkinThickness             769 non-null    int64
 4   Insulin                   769 non-null    int64
 5   BMI                       769 non-null    float64
 6   DiabetesPedigreeFunction  769 non-null    float64
 7   Age                       769 non-null    int64
 8   Outcome                   769 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.2 KB
```

Excepting BMI and DiabetesPedigreeFunction all the columns are integer. Outcome is the label containing 1 and 0 values. 1 means person has diabetes and 0 mean person is not diabetic.

*# Show top 5 rows*

df.head()

| | 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 |

## Missing Value Analysis

Next, i will cleanup the dataset which is the important part of data science. Missing data can lead to wrong statistics during modeling and predictions.

*# Explore missing values*df.isnull().sum()

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Missing Value Analysis

I observed that there is no missing values in dataset however the features like Glucose, BloodPressure, Insulin, SkinThickness has 0 values which is not possible. We have to replace 0 values with either mean or median values of specific column.

df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())*# Correcting missing values in blood pressure*

df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].mean()) # There are 35 records with 0 BloodPressure in dataset*# Correcting missing values in BMI*

df['BMI'] = df['BMI'].replace(0, df['BMI'].median())*# Correct missing values in Insulin and SkinThickness*

df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())

df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())

Now, lets review the dataset statistics

*# Review dataset statistics*

df.describe()

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| count | 769.000000 | 769.000000 | 769.000000 | 769.000000 | 769.000000 | 769.000000 | 769.000000 | 769.000000 | 769.000000 |
| mean | 3.840052 | 121.683337 | 72.261444 | 27.328999 | 93.801040 | 32.455917 | 0.471590 | 33.269181 | 0.349805 |
| std | 3.370237 | 30.416231 | 12.109128 | 9.224328 | 105.977841 | 6.872291 | 0.331208 | 11.778737 | 0.477219 |
| min | 0.000000 | 44.000000 | 24.000000 | 7.000000 | 1.000000 | 18.200000 | 0.078000 | 21.000000 | 0.000000 |
| 25% | 1.000000 | 100.000000 | 64.000000 | 23.000000 | 29.000000 | 27.500000 | 0.244000 | 24.000000 | 0.000000 |
| 50% | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 29.000000 | 32.000000 | 0.371000 | 29.000000 | 0.000000 |
| 75% | 6.000000 | 140.000000 | 80.000000 | 32.000000 | 127.000000 | 36.600000 | 0.626000 | 41.000000 | 1.000000 |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

Now i have clean dataset without missing values in features which is good.

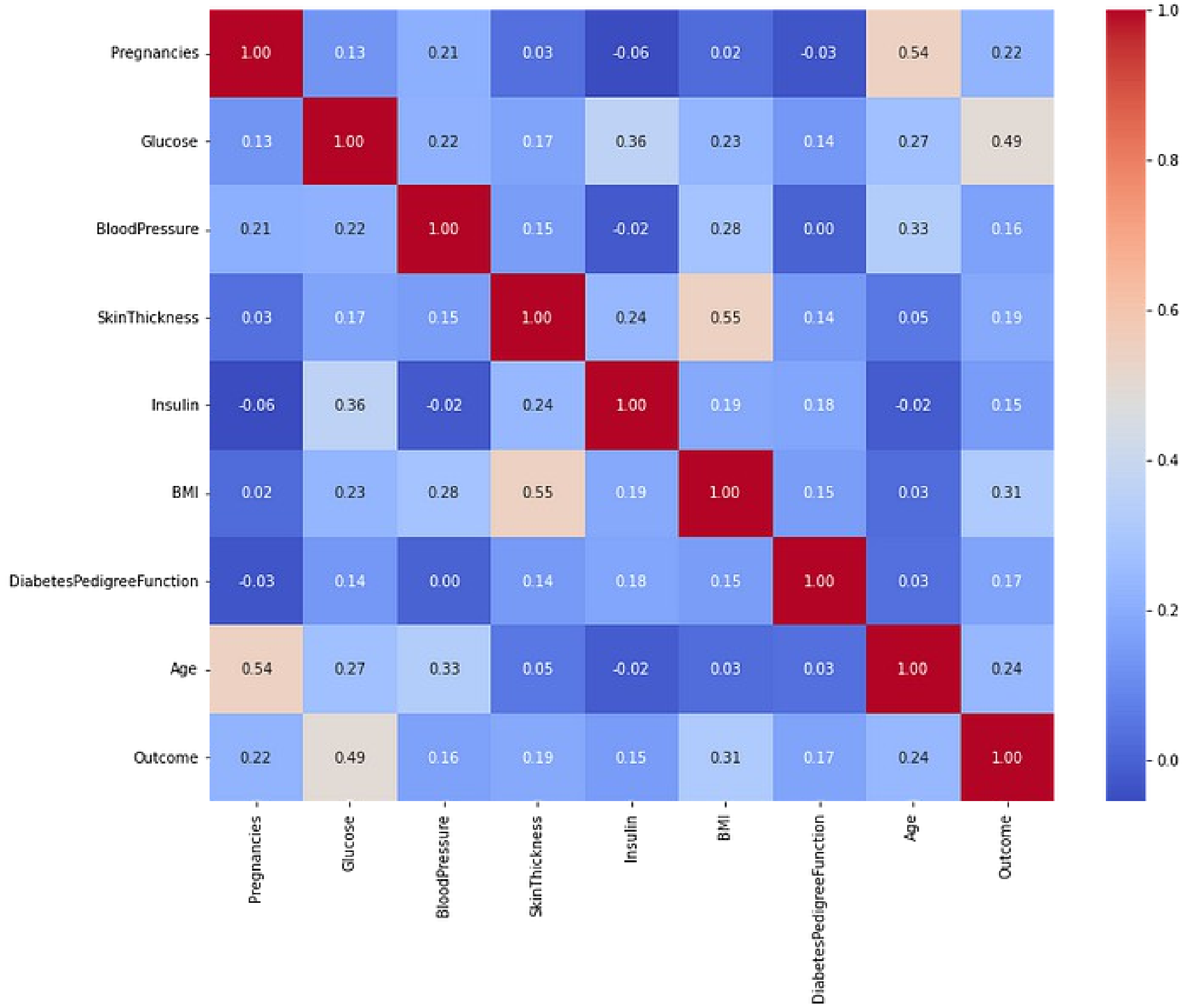
## Exploratory Data Analysis

In this step, i showcased anlytics using GUI using Seaborn.

## Correlation

Correlation is **one or more variables are related** to each other. It also helps to find the feature importance and clean the dataset before i start Modeling

```
plt.figure(figsize=(13,10))
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap = "coolwarm")
```

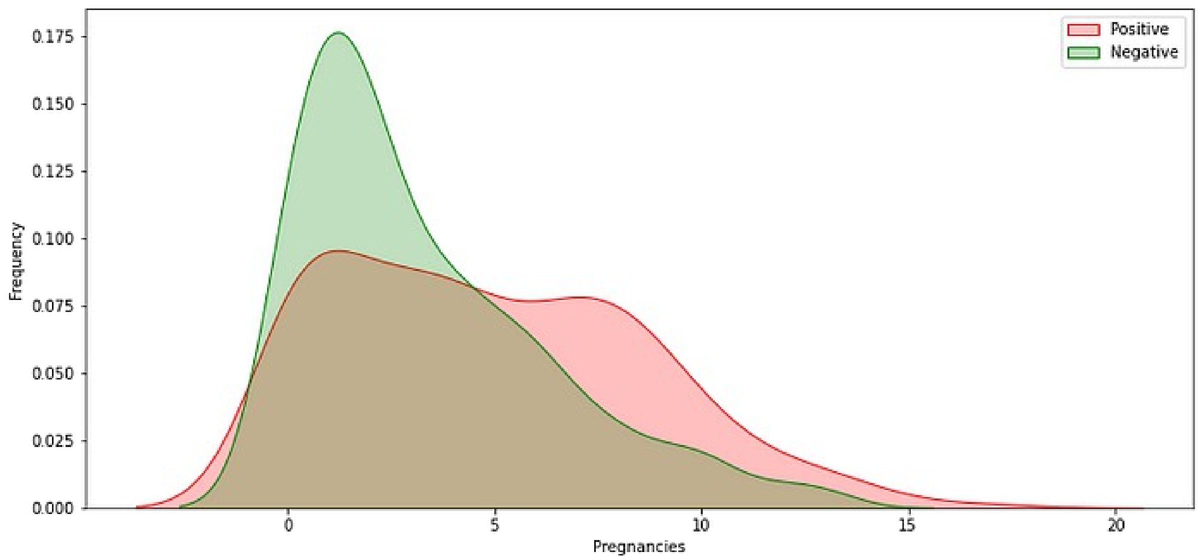


According to observation, features like Pregnancies, Gluecose, BMI, and Age is more correlated with Outcome. In next steps, i showcased details representation of these features.

## Pregnancies

Women with diabetes can and do have healthy pregnancies and healthy babies. Managing diabetes can help reduce your risk for complications. Untreated diabetes increases your risk for pregnancy complications, like high blood pressure, depression, premature birth, birth defects and pregnancy loss.

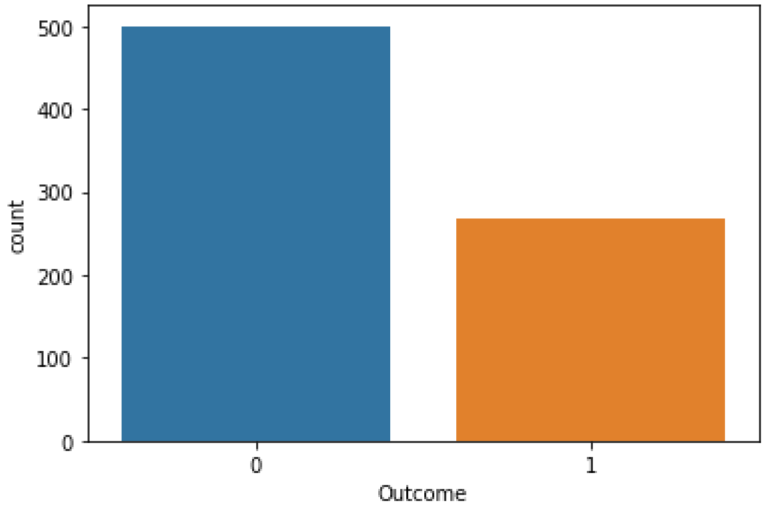
```
# Explore Pregnancies vs Outcomeplt.figure(figsize=(13,6))
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1],
    color="Red", shade = True)
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0],
    ax =g, color="Green", shade= True)g.set_xlabel("Pregnancies")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```



## Outcome

Outcome has 1 and 0 values where 1 indicates that person has diabetes and 0 shows person has no diabetes. This is my label column in dataset.

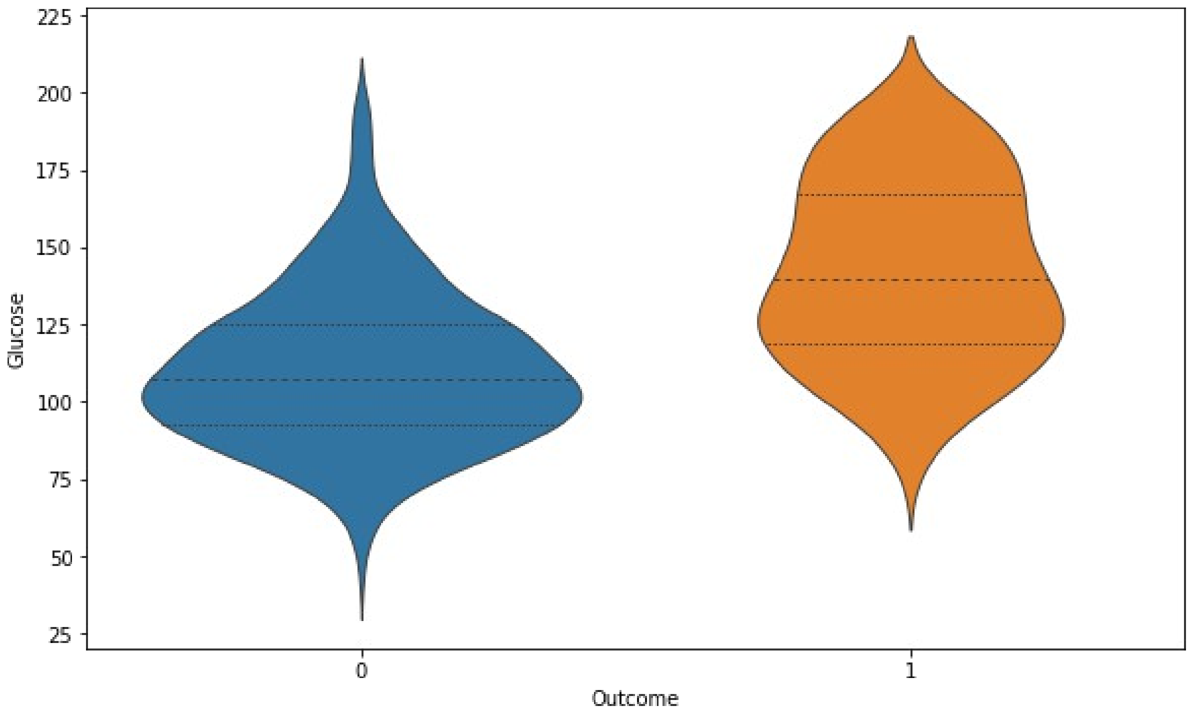
```
sns.countplot('Outcome', data = df)
```




It indicates, There are more people who do not have diabetes in dataset which is around 65% and 35% people has diabetes.

## Glucose

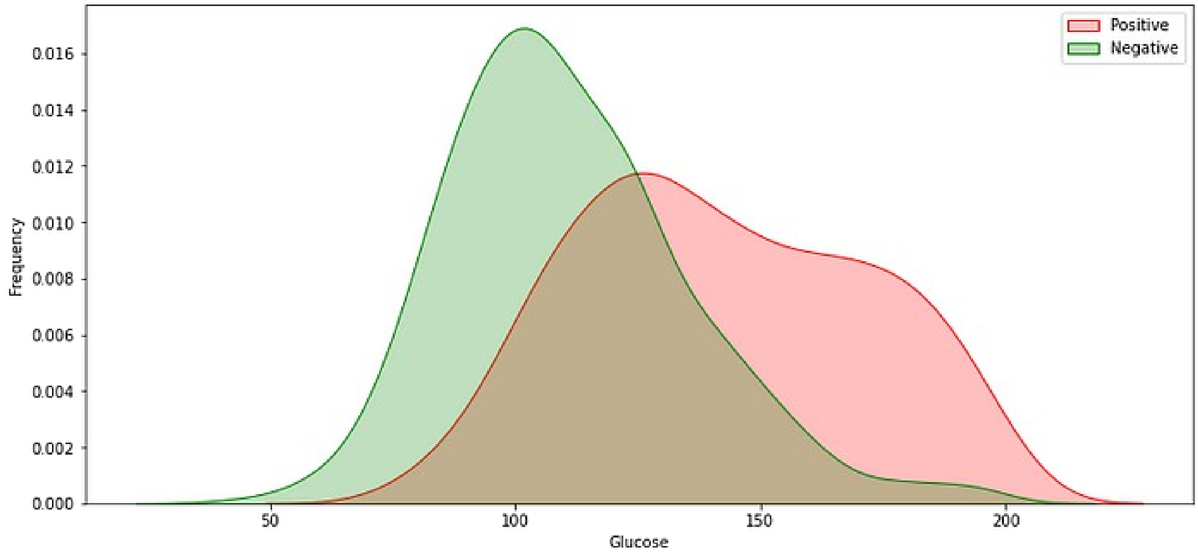
```
# Explore Gluecose vs Outcomeplt.figure(figsize=(10,6))
sns.violinplot(data=df, x="Outcome", y="Glucose",
           split=True, inner="quart", linewidth=1)
```




The chances of diabetes is gradually increasing with level of Glucose.

*# Explore Glucose vs Outcome*

```
plt.figure(figsize=(13,6))

g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = True)

g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade= True)

g.set_xlabel("Glucose")

g.set_ylabel("Frequency")

g.legend(["Positive","Negative"])
```

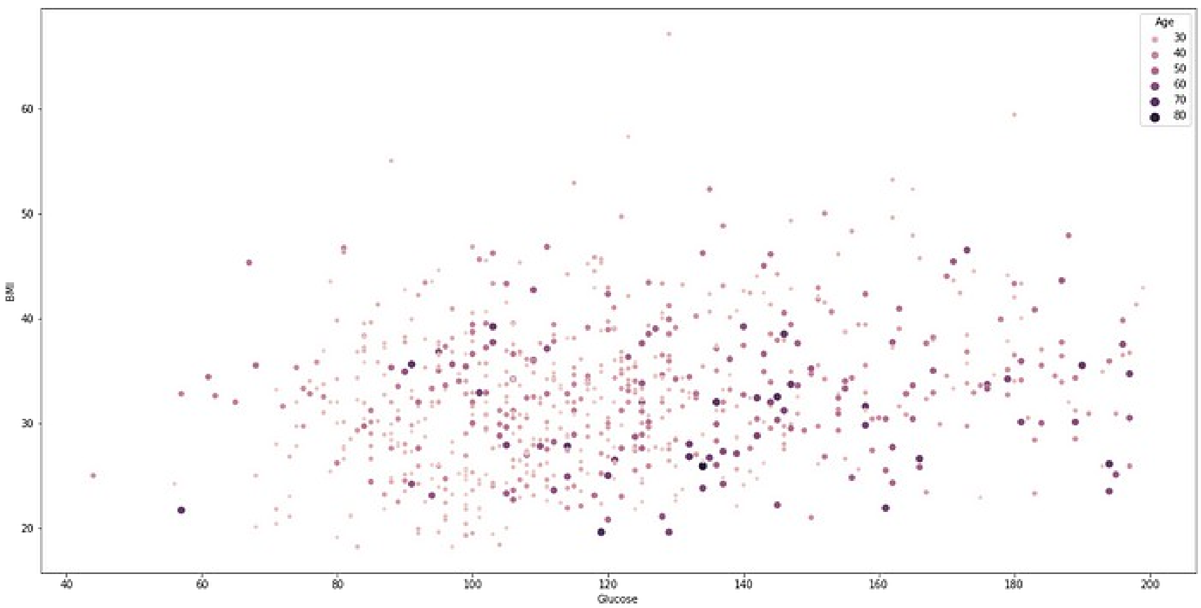



## Explore Glucose vs BMI vs Age

*# Glucose vs BMI vs Age*

plt.figure(figsize=(20,10))

sns.scatterplot(data=df, x="Glucose", y="BMI", hue="Age", size="Age")



As per observation there are some outliers in features. We need to remove outliers in feature engineering.

## Feature Engineering

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning.

Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.

## Outlier Detection

In this part i removed all the records outlined in dataset. Outliers impacts Model accuracy. I used *Tukey Method* used for outlier detection.

```
def detect_outliers(df,n,features):
    outlier_indices = []
    """
    Detect outliers from given list of features. It returns a list of the indices
    according to the observations containing more than n outliers according
    to the Tukey method
    """
    # iterate over features(columns)
    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col],75)
```

```
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers

# detect outliers from numeric features
outliers_to_drop = detect_outliers(df, 2 ,["Pregnancies", 'Glucose', 'BloodPressure', 'BMI',
'DiabetesPedigreeFunction', 'SkinThickness', 'Insulin', 'Age'])
```

Here, I find outliers from all the features such as Pregnancies, Glucose, BloodPressure, BMI, DiabetesPedigreeFunction, SkinThickness, Insulin, and Age.

```
df.drop(df.loc[outliers_to_drop].index, inplace=True)
```

I have successfully removed all outliers from dataset now. The next step is to split the dataset in train and test and proceed the modeling.

**Modeling**

In this sections, i tried different models and compare the accuracy for each. Then, i performed Hyperparameter Tuning on Models that has high accuracy.

Before i split the dataset i need to transform the data into quantile using sklearn.preprocessing .

# Data Transformation

q  = QuantileTransformer()

X = q.fit_transform(df)

transformedDF = q.transform(X)

transformedDF = pd.DataFrame(X)

transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']# Show top 5 rows

transformedDF.head()

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.748047 | 0.810547 | 0.516276 | 0.802083 | 0.000000 | 0.590495 | 0.751302 | 0.888672 | 1.0 |
| 1 | 0.233724 | 0.097656 | 0.335286 | 0.645182 | 0.000000 | 0.227865 | 0.476562 | 0.557943 | 0.0 |
| 2 | 0.863932 | 0.957682 | 0.278646 | 0.000000 | 0.000000 | 0.091797 | 0.782552 | 0.583984 | 1.0 |
| 3 | 0.233724 | 0.130859 | 0.335286 | 0.505859 | 0.663411 | 0.298177 | 0.106120 | 0.000000 | 0.0 |
| 4 | 0.000000 | 0.722656 | 0.050781 | 0.802083 | 0.833984 | 0.927083 | 0.997396 | 0.606120 | 1.0 |

Data Transformation

## Data Splitting

Next, i split data in test and train dataset. Train dataset will be used in Model training and evaluation and test dataset will be used in prediction. Before i predict the test data, i performed cross validation for various models.

features = df.drop(["Outcome"], axis=1)

labels = df["Outcome"]x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.30, random_state=7)

Above code splits dataset into train (70%) and test (30%) dataset.

**Cross Validate Models**

```
def evaluate_model(models):
    """
    Takes a list of models and returns chart of cross validation scores using mean accuracy
    """

    # Cross validate model with Kfold stratified cross val
    kfold = StratifiedKFold(n_splits = 10)

    result = []
    for model in models :
        result.append(cross_val_score(estimator = model, X = x_train, y = y_train, scoring = "accuracy", cv
= kfold, n_jobs=4))

    cv_means = []
    cv_std = []
    for cv_result in result:
        cv_means.append(cv_result.mean())
        cv_std.append(cv_result.std())

    result_df = pd.DataFrame({
```

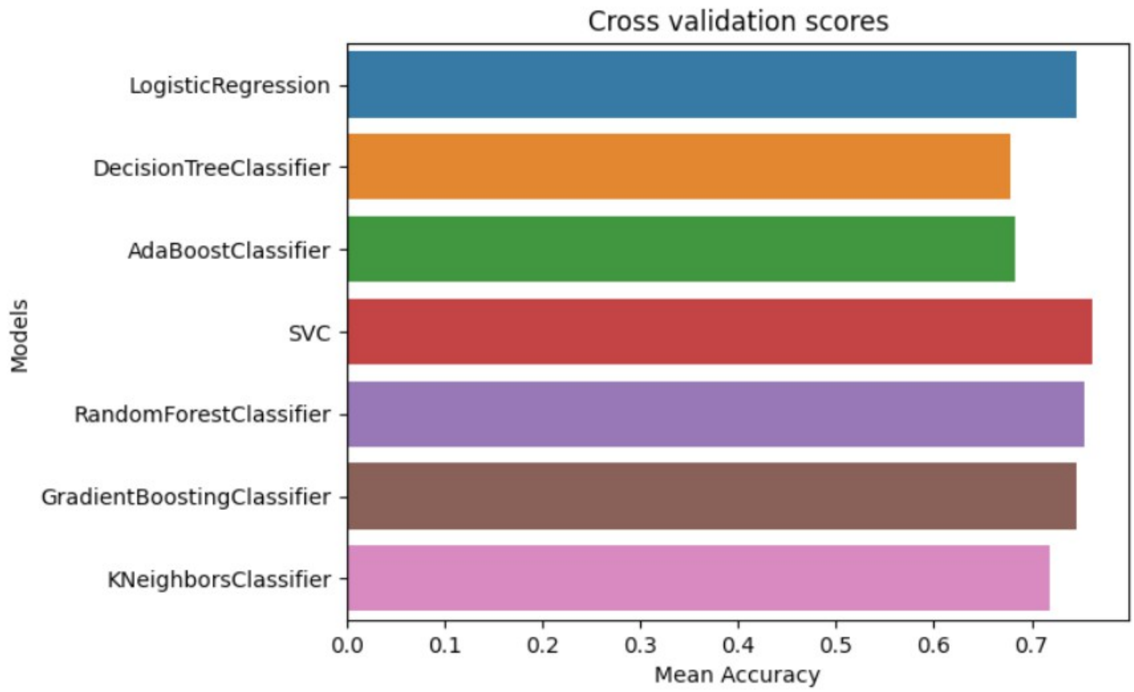
```
        "CrossValMeans":cv_means,
        "CrossValerrors": cv_std,
        "Models":[
            "LogisticRegression",
            "DecisionTreeClassifier",
            "AdaBoostClassifier",
            "SVC",
            "RandomForestClassifier",
            "GradientBoostingClassifier",
            "KNeighborsClassifier"
        ]
    })

    # Generate chart
    bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result_df, orient = "h")
    bar.set_xlabel("Mean Accuracy")
    bar.set_title("Cross validation scores")
    return result_df
```

*Method `evaluate_model` takes a list of models and returns chart of cross validation scores using mean accuracy.*

```
# Modeling step Test differents algorithms
random_state = 30
models = [
    LogisticRegression(random_state = random_state, solver='liblinear'),
    DecisionTreeClassifier(random_state = random_state),
    AdaBoostClassifier(DecisionTreeClassifier(random_state = random_state), random_state =
```

```
random_state, learning_rate = 0.2),
  SVC(random_state = random_state),
  RandomForestClassifier(random_state = random_state),
  GradientBoostingClassifier(random_state = random_state),
  KNeighborsClassifier(),
]
evaluate_model(models)
```

| | CrossValMeans | CrossValerrors | Models |
|---|---|---|---|
| 0 | 0.746436 | 0.049309 | LogisticRegression |
| 1 | 0.677463 | 0.062306 | DecisionTreeClassifier |
| 2 | 0.683159 | 0.073619 | AdaBoostClassifier |
| 3 | 0.761251 | 0.041392 | SVC |
| 4 | 0.753878 | 0.075555 | RandomForestClassifier |
| 5 | 0.746541 | 0.087685 | GradientBoostingClassifier |
| 6 | 0.718484 | 0.073027 | KNeighborsClassifier |



Cross Validate Models

As per above observation, i found that SVC, RandomForestClassifier, and LogisticRegression model has more accuracy. Next, i will do hyper parameter tuning on three models.

## Hyperparameter Tuning

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

I have done tuning process for SVC, RandomForestClassifier, and LogisticRegression models one by one.

```
# Import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_reportdef analyze_grid_result(grid_result):
    '''
    Analysis of GridCV result and predicting with test dataset
    Show classification report at last
    '''    # Best parameters and accuracy
    print("Tuned hyperparameters: (best parameters) ", grid_result.best_params_)
    print("Accuracy :", grid_result.best_score_)

    means = grid_result.cv_results_["mean_test_score"]
    stds = grid_result.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds, grid_result.cv_results_["params"]):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

```
    print()    print("Detailed classification report:")
    y_true, y_pred = y_test, grid_result.predict(x_test)
    print(classification_report(y_true, y_pred))
    print()
```

First of all i have imported GridSearchCV and classification_report from sklearn library. Then, i have defined `analyze_grid_result` method which will show prediction result. I called this method for each Model used in SearchCV. In next step, i will perform tuning for each model.

## Logistic Regression

```
# Define models and parameters for LogisticRegression
model = LogisticRegression(solver='liblinear')
solvers = ['newton-cg', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]# Define grid search
grid = dict(solver = solvers, penalty = penalty, C = c_values)
cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = cv, scoring = 'accuracy',
error_score = 0)
logi_result = grid_search.fit(x_train, y_train)# Logistic Regression Hyperparameter Result
analyze_grid_result(logi_result)
```

Output:

```
Tuned hyperparameters: (best parameters)  {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
Accuracy : 0.7641818181818182
0.764 (+/-0.254) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.764 (+/-0.254) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.764 (+/-0.254) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.764 (+/-0.254) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.759 (+/-0.257) for {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.753 (+/-0.253) for {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.751 (+/-0.251) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.707 (+/-0.217) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.751 (+/-0.239) for {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.674 (+/-0.213) for {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86       150
           1       0.78      0.62      0.69        81

    accuracy                           0.81       231
   macro avg       0.80      0.76      0.77       231
weighted avg       0.80      0.81      0.80       231
```

As per my observation, in LogisticRegression it returned best score 0.78 with `*{'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}*` parameters. Next i will perform tuning for other models.

**SVC**

# Define models and parameters for LogisticRegression

model = SVC()# Define grid search

tuned_parameters = [

{"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},

{"kernel": ["linear"], "C": [1, 10, 100, 1000]},

]

cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)

grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv = cv, scoring =

'accuracy', error_score = 0)

scv_result = grid_search.fit(x_train, y_train)# SVC Hyperparameter Result

analyze_grid_result(scv_result)

Output:

```
Tuned hyperparameters: (best parameters)  {'C': 10, 'kernel': 'linear'}
Accuracy : 0.7779850746268657
0.709 (+/-0.045) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.731 (+/-0.015) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.683 (+/-0.037) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.729 (+/-0.004) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.625 (+/-0.034) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.720 (+/-0.022) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.632 (+/-0.019) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.718 (+/-0.034) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.772 (+/-0.037) for {'C': 1, 'kernel': 'linear'}
0.778 (+/-0.019) for {'C': 10, 'kernel': 'linear'}
0.759 (+/-0.004) for {'C': 100, 'kernel': 'linear'}
0.744 (+/-0.026) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.77      0.84      0.80       144
           1       0.68      0.58      0.63        86

    accuracy                           0.74       230
   macro avg       0.73      0.71      0.72       230
weighted avg       0.74      0.74      0.74       230
```

SVC Model gave max 0.77 accuracy which is bit less than LogisticRegression. I will not use this model anymore.

## RandomForestClassifier

# Define models and parameters for LogisticRegression

model = RandomForestClassifier(random_state=42)# Define grid search

```
tuned_parameters = {
   'n_estimators': [200, 500],
   'max_features': ['auto', 'sqrt', 'log2'],
   'max_depth' : [4,5,6,7,8],
   'criterion' :['gini', 'entropy']
}
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = tuned_parameters, cv = cv, scoring =
'accuracy', error_score = 0)
grid_result = grid_search.fit(x_train, y_train)# SVC Hyperparameter Result
analyze_grid_result(grid_result)
```

Output:

```
   macro avg       0.74      0.73      0.73       230
weighted avg       0.75      0.76      0.75       230
```

Tuned hyperparameters: (best parameters)  {'criterion': 'entropy', 'max_depth': 5, 'max_features': 'log2', 'n_estimators': 200}

Accuracy : 0.7663648051875454

Detailed classification report:

Randomforest model gave max 0.76% accuracy which is not best comparing to other model. So i decided to use LogisticRegression Model for prediction.

## Prediction

Till now, i worked on EDA, Feature Engineering, Cross Validation of Models, and Hyperparameter Tuning and find the best working Model for my dataset. Next, I did prediction from my test dataset and storing the result in CSV.

```
# Test predictions
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
#output
```

```
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       144
           1       0.70      0.58      0.64        86

    accuracy                           0.75       230
   macro avg       0.74      0.72      0.72       230
weighted avg       0.75      0.75      0.75       230
```

Finally append new feature column in test dataset called Prediction and print the dataset.

```
x_test['pred'] = y_pred
print(x_test)
```

Diabetes Predictions

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
236            7    181.0           84.0             21      192  35.9
716            3    173.0           78.0             39      185  33.8
767            1     93.0           70.0             31       29  30.4
499            6    154.0           74.0             32      193  29.3
61             8    133.0           72.0             23       29  32.9
..           ...      ...            ...            ...      ...   ...
189            5    139.0           80.0             35      160  31.6
351            4    137.0           84.0             23       29  31.2
120            0    162.0           76.0             56      100  53.2
108            3     83.0           58.0             31       18  34.3
637            2     94.0           76.0             18       66  31.6

     DiabetesPedigreeFunction  Age  pred
236                     0.586   51     1
716                     0.970   31     1
767                     0.315   23     0
499                     0.839   39     1
61                      0.270   39     1
..                        ...  ...   ...
189                     0.361   25     0
351                     0.252   30     0
120                     0.759   25     1
108                     0.336   25     0
637                     0.649   23     0

[230 rows x 9 columns]
```

I will perform feature importance in separate article for more understanding the impact of feature after modeling.

```
Click here to ask Blackbox to help you code faster
# test for to prediction

new_data = [[1, 85, 66, 29, 0, 26.6, 0.351, 31],  # Example new data point 1
            [6, 148, 72, 35, 0, 33.6, 0.627, 50]]  # Example new data point 2

# Make predictions on new data
predictions = logi_result.predict(new_data)
print("Predictions:", predictions)
```
[40] ✓ 0.0s Python

```
Predictions: [0 1]
c:\Python310\lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature
  warnings.warn(
```

```
Click here to ask Blackbox to help you code faster
import joblib

# Assuming 'model' is your trained model in test.ipynb
joblib.dump(logi_result, 'trained_model.pkl')
```
[42] ✓ 0.0s Python

```
['trained_model.pkl']
```

## Conclusion

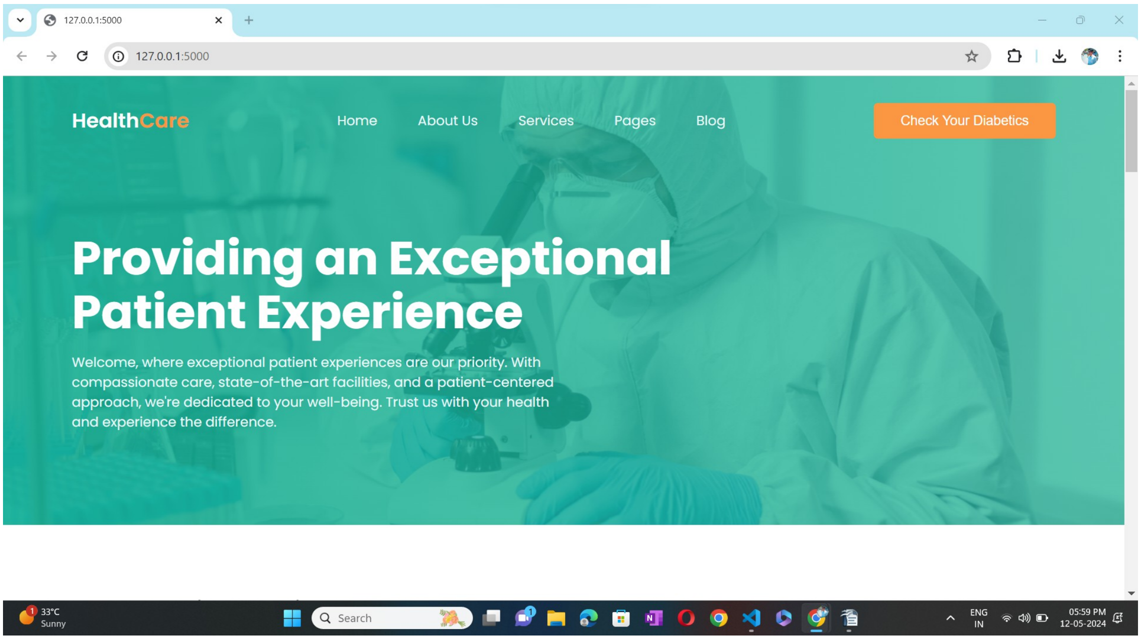
1. Diabetes is one of the ricks during Pregnancy. It has to be treat to avoid complications.

2. BMI index can help to avoid complications of diabetes a way before

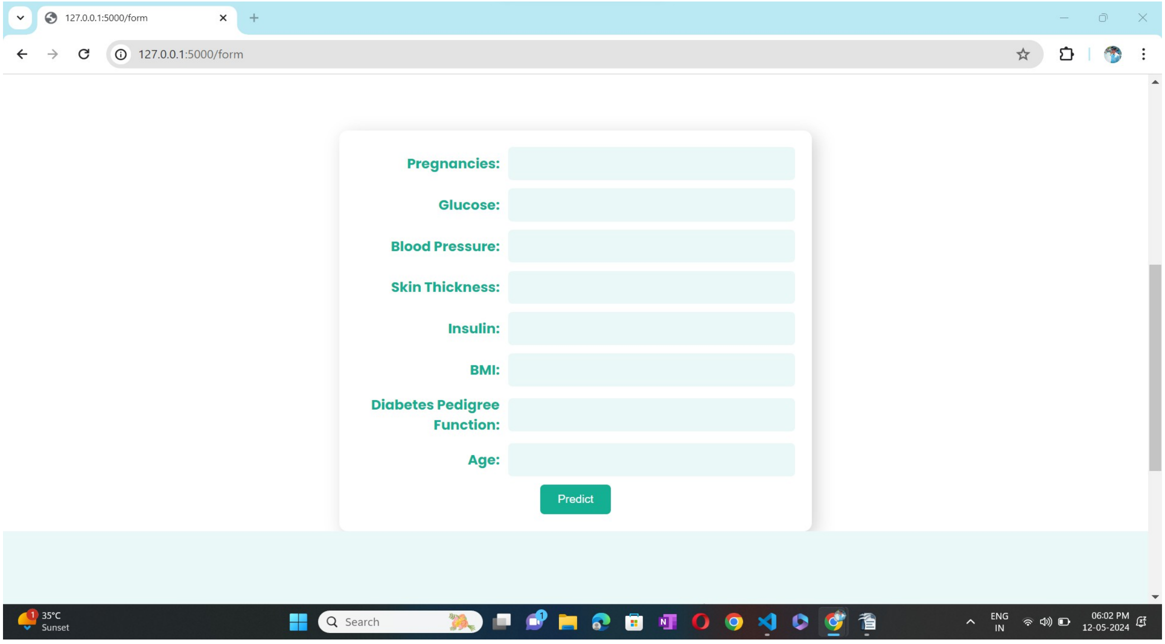
3. Diabetes start showing in age of 35 – 40 and increase with person age.

**RESULT OUTPUTS:**

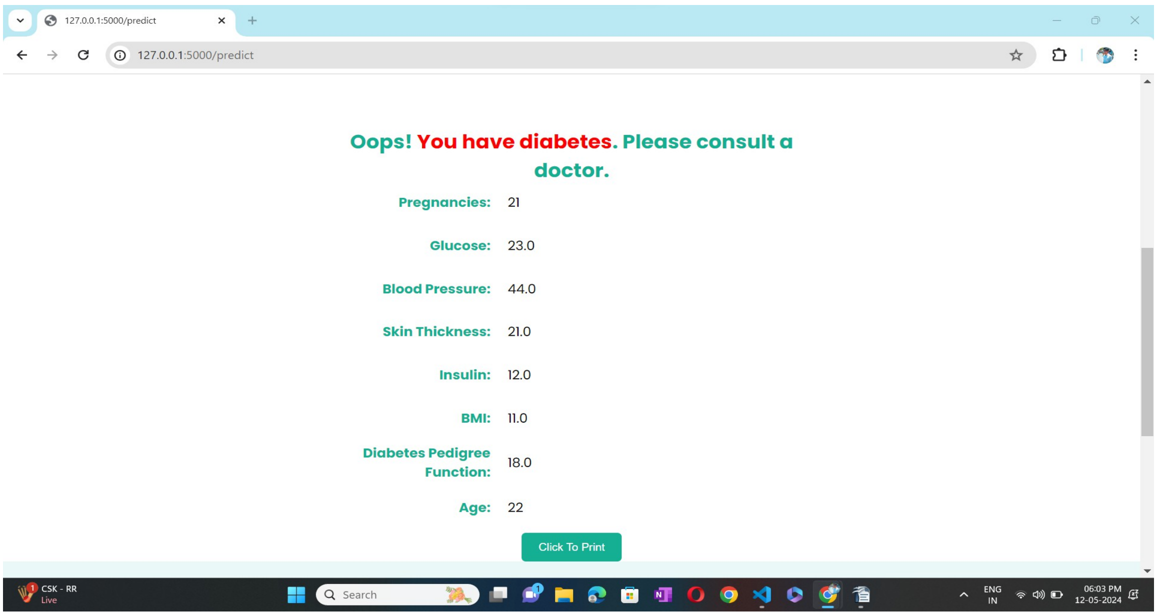
HOME PAGE



STEP 1 : CLICK CHECK YOUR DIABETICS BUTTON TO ENTER DETAILS




STEP 2 : ENTER DETAILS TO SEE RESULTS



IT WILL SHOW IF YOU HAVE DIABETES , THEN YOU HAVING RISK OR NOT

STEP 3 : CLICK THE PRINT BUTTON TO SAVE THE REPORT AND IT WILL BE DOWNLOADED SUCCESSFULLY