Phase-5: Project Documentation and Submission

Title: AI Based Diabetes Prediction System

Dataset link: https://www.kaggle.com/datasets/mathchi/diabetes-data-set

Software used: google colab

Introduction:

Healthcare sectors have large volume databases. Such databases may contain structured, semi-structured or unstructured data. Big data analytics is the process which analyses huge data sets and reveals hidden information, hidden patterns to discover knowledge from the given data. Diabetes mellitus (DM) is classified as-

Type 1:

Type-1 known as Insulin-Dependent Diabetes Mellitus (IDDM). Inability of human's body to generate sufficient insulin is the reason behind this type of DM and hence it is required to inject insulin to a patient.

Type 2:

Type-2 also known as Non-Insulin-Dependent Diabetes Mellitus (NIDDM). This type of Diabetes is seen when body cells are not able to use insulin properly.

Type 3:

Type-3 Gestational Diabetes, increase in blood sugar level in pregnant woman where diabetes is not detected earlier results in this type of diabetes. DM has long term complications associated with it.

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.

Importing Data:

The diabetes data set was originated from **https://www.kaggle.com/datasets/mathchi/diabetes-data-set.** Diabetes dataset containing 768 instances with 9 features.The objective is to predict if the patient is diabetic or not. The “Outcome” is the feature we are going to predict ,0 means No diabetes, 1 means diabetes.

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.

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.

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.

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 Splitting:

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

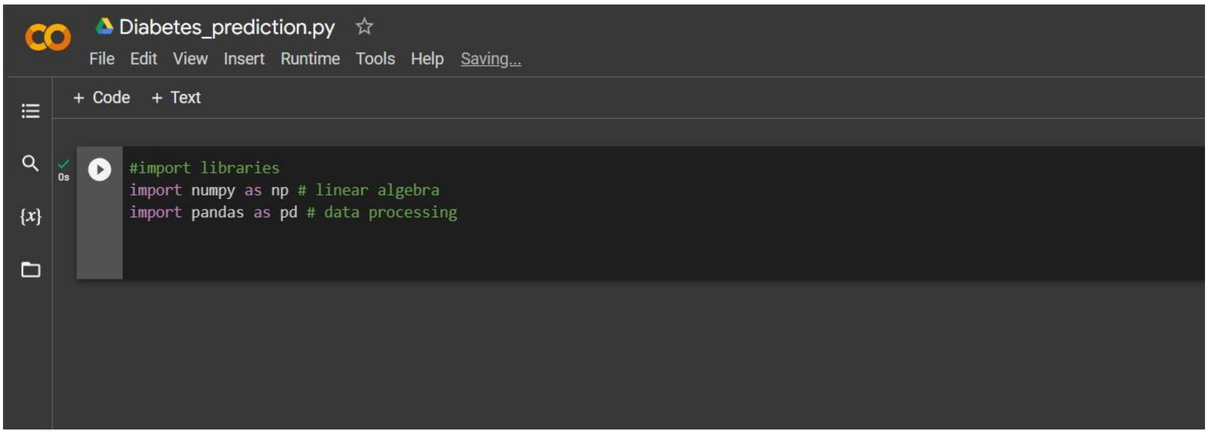
Prediction:

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.

Process:

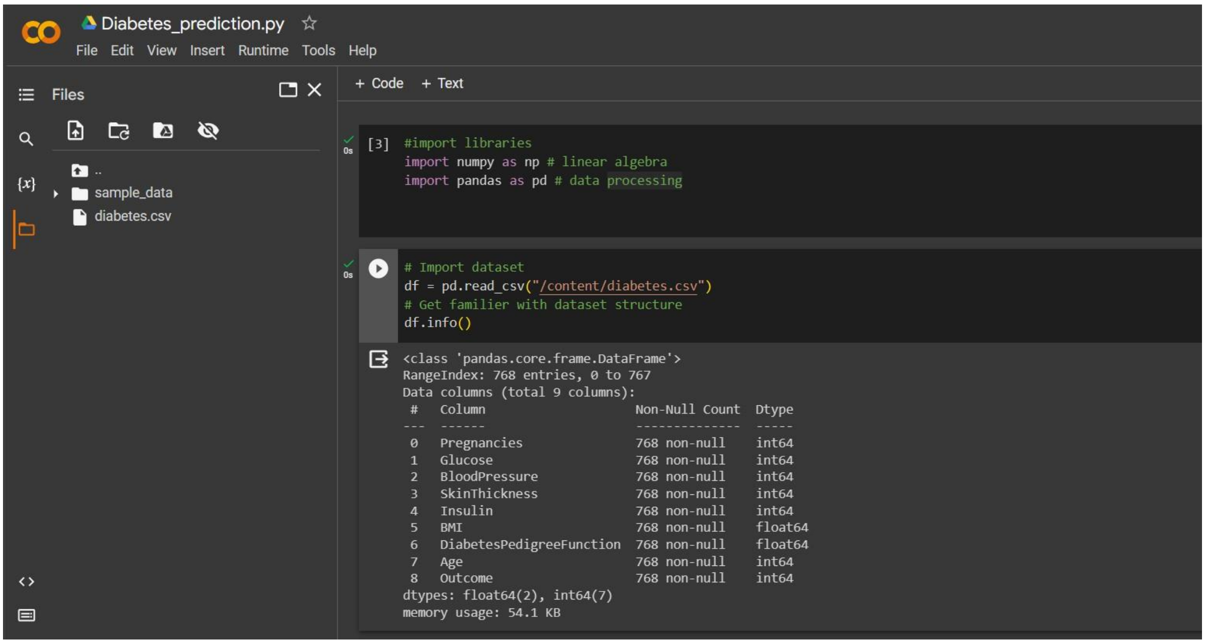
1.Installing Libraries:

In this first step I have imported most common libraries used in python for machine learning such as Pandas, Numpy etc.

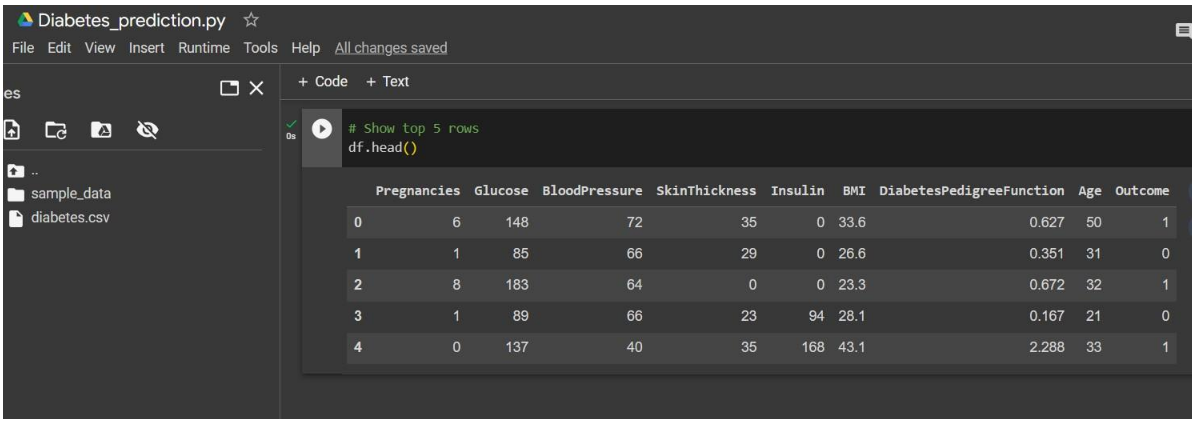


2.Importing Data:

The diabetes data set was originated from **https://www.kaggle.com/datasets/mathchi/diabetes-data-set**. Diabetes dataset containing 768 instances with 9 features.The objective is to predict if the patient is diabetic or not.

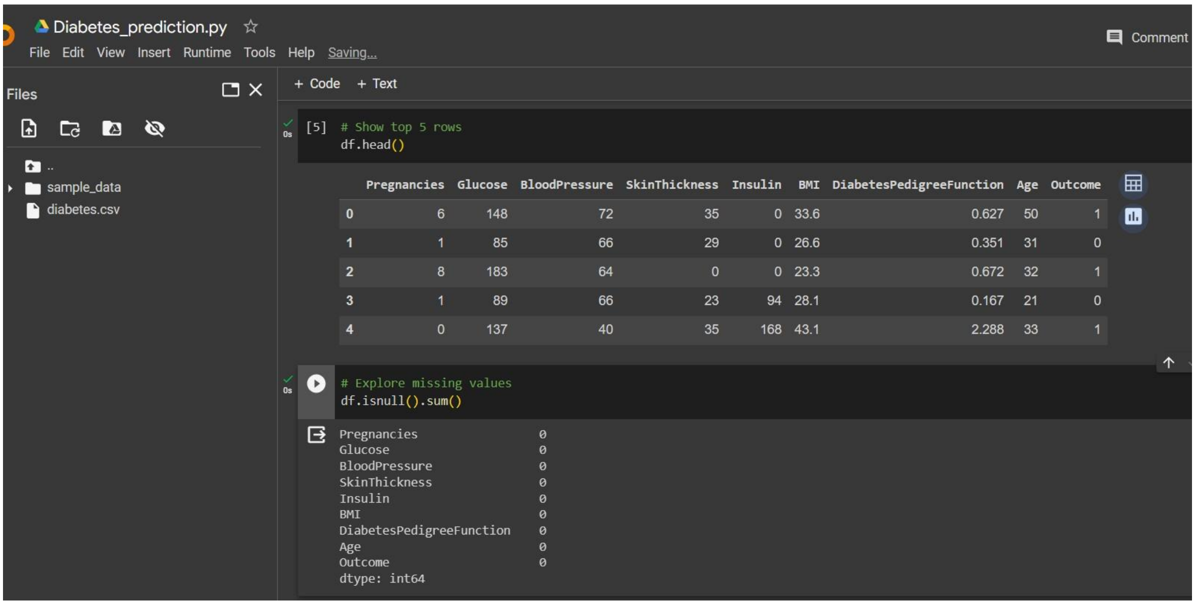


Displaying Data:



Data Processing:

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.



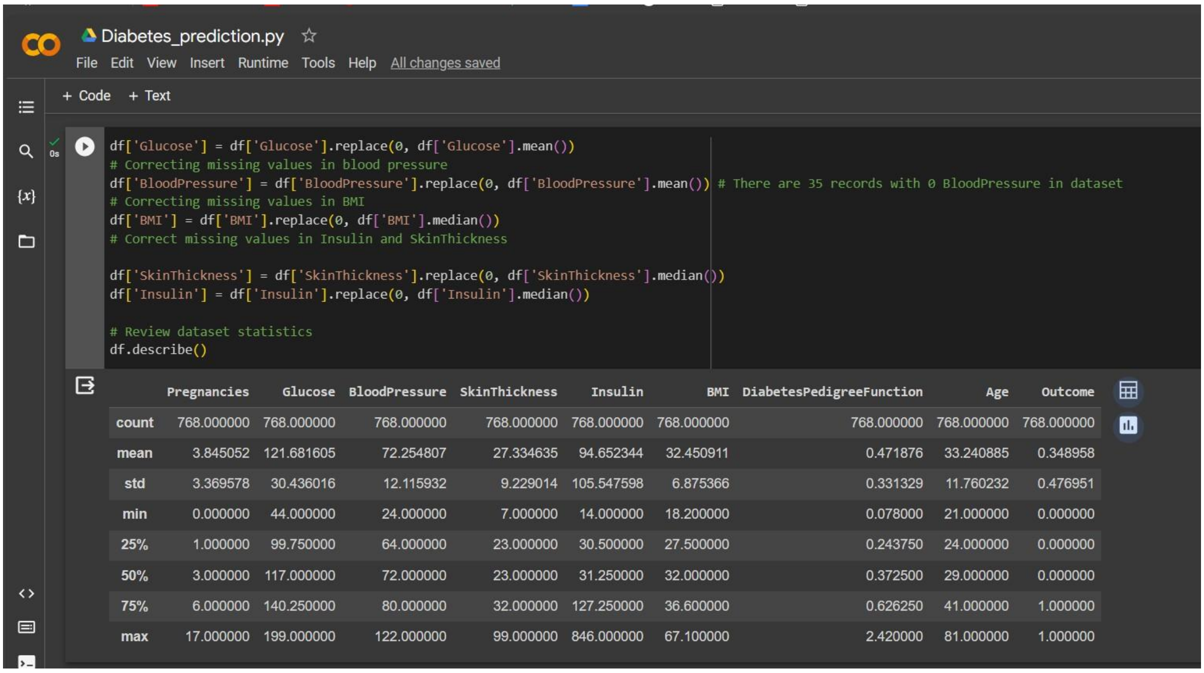

```
# Show top 5 rows
df.head()
```

	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

```
# 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:




```
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())

# Review dataset statistics
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	121.681605	72.254807	27.334635	94.652344	32.450911	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.229014	105.547598	6.875366	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	23.000000	30.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	31.250000	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

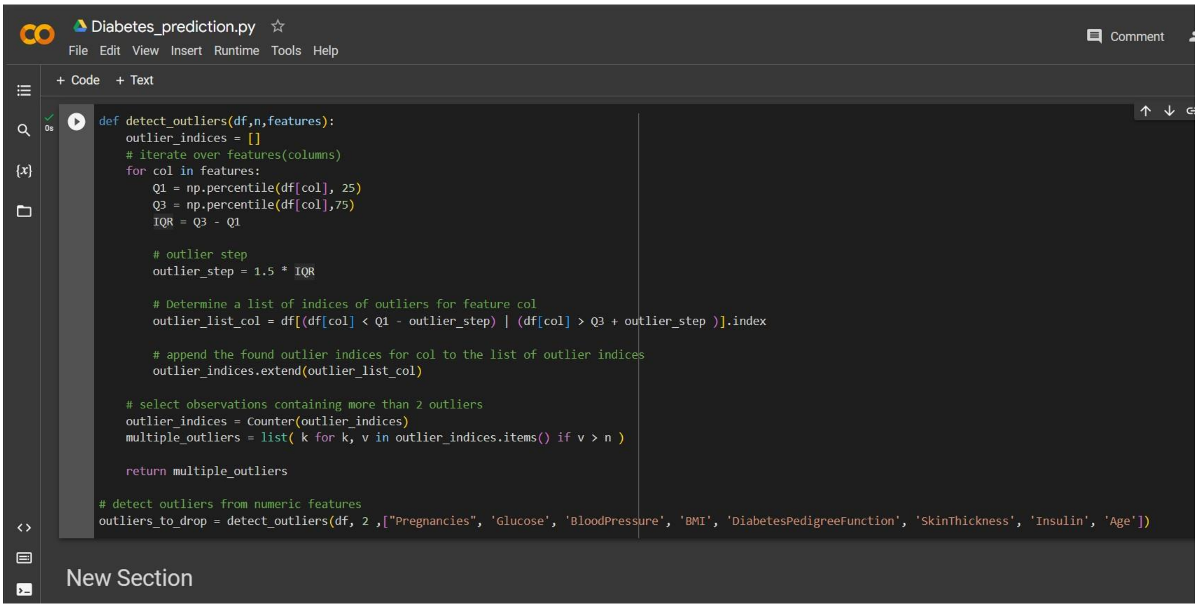
Fearure Engineering:

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.



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)
print(df)
```

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()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but QuantileTransformer was fitted with fe
  warnings.warn(
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.747718	0.810300	0.516949	0.801825	0.000000	0.591265	0.750978	0.889831	1.0
1	0.232725	0.097784	0.336375	0.644720	0.000000	0.227510	0.475880	0.558670	0.0
2	0.863755	0.956975	0.279009	0.000000	0.000000	0.091917	0.782269	0.585398	1.0
3	0.232725	0.131030	0.336375	0.505867	0.662973	0.298566	0.106258	0.000000	0.0
4	0.000000	0.721643	0.050847	0.801825	0.834420	0.926988	0.997392	0.606258	1.0

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))
```

```
    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
```

As per above observation, I found that Logistic Regression model has more accuracy.next, I will do hyperparameter tuning on model.

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.

```
# Import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
def 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': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Accuracy : 0.774909090909091
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.241) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.775 (+/-0.226) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.240) for {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.224) for {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.242) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.720 (+/-0.225) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.764 (+/-0.245) for {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.687 (+/-0.256) for {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.74      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.77      0.73      0.74       231
weighted avg       0.77      0.77      0.77       231
```

As per my observation, in LogisticRegression it returned best score 0.78 with `{'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}` parameters.

Prediction:

Till now, i worked on 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))
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.74      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.77      0.73      0.74       231
weighted avg       0.77      0.77      0.77       231
```

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

Final output:

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

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
353            1       90             62             12       43  27.2
236            7      181             84             21      192  35.9
323           13      152             90             33       29  26.8
98             6       93             50             30       64  28.7
701            6      125             78             31        0  27.6
..           ...      ...            ...            ...      ...   ...
188            8      109             76             39      114  27.9
351            4      137             84              0        0  31.2
120            0      162             76             56      100  53.2
108            3       83             58             31       18  34.3
616            6      117             96              0        0  28.7

     DiabetesPedigreeFunction  Age  pred
353                     0.580   24     0
236                     0.586   51     1
323                     0.731   43     1
98                      0.356   23     0
701                     0.565   49     0
..                        ...  ...   ...
188                     0.640   31     0
351                     0.252   30     0
120                     0.759   25     1
108                     0.336   25     0
616                     0.157   30     0

[231 rows x 9 columns]
```

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.