AI BASED DIABETES PREDICTION SYSTEM

TEAM NAME : Proj_224748_Team1

TOPIC :PREDICTING DIABETES USING MACHINE LEARNING

BY :

B.HEMALATHA

R.SATHYABAMA

G.GURU PRIYA

C.ROOPAVATHY

J.ABINAYA

DIABETES PREDICTION

- Diabetes prediction is vital in the modern world due to its potential to provide early intervention and prevention, improve health outcomes, reduce healthcare costs, and personalize care. Predictive models empower individuals to make lifestyle changes, optimize resource allocation, inform public health policies, support research, and enhance the quality of life. With the rising prevalence of diabetes and its associated burdens, accurate prediction models are crucial in healthcare and public health efforts. In this comprehensive guide, we will continue to delve deeper into the construction of a diabetes prediction system by focusing on three fundamental components: feature selection, model training, and evaluation.

- Feature selection is the process of identifying and selecting the most relevant features from a dataset to improve the performance of a machine learning model. Feature selection in diabetes prediction involves choosing the most relevant patient characteristics from dataset to build an accurate model for predicting diabetes. This process helps improve model performance, interpretability, and resource efficiency in healthcare by focusing on the most important data points.

- Model training is the process of feeding the selected features to a machine learning algorithm and allowing it to learn the relationship between the features and the target variable (i.e. diabetes prediction). Once the model is trained, it can be used to predict the diabetes of new people, given their features.

- Model evaluation is the process of assessing the performance of a machine learning or statistical model by comparing its predictions to actual outcomes. It involves using various metrics, data splitting techniques, and visualizations to determine how well the model is working. Model evaluation in diabetes prediction involves assessing the performance of a machine learning or statistical model specifically designed to predict diabetes. This process includes using various metrics to measure the model's accuracy, precision, recall, and other relevant criteria. The goal is to determine how well the model can predict diabetes in real-world scenarios. Model evaluation helps ensure that the model is effective, reliable, and suitable for its intended purpose, such as early diagnosis or risk assessment for diabetes.

GIVEN DATA SET :

SL.NO	PREGNANCIES	GLUCOSE	BLOOD PRESSURE	SKIN THICKNESS	INSULIN	BMI	DIABETES PEDIGREE FUNCTION	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

OVERVIEW OF THE DATA :

The following is an overview of the process of predicting diabetes system by feature selection, model training, and evaluation:

1. Prepare the data

2. Perform feature selection
3. Train the model
4. Evaluate the model

FEATURE SELECTION :

1.Identify the target variable :

This is the variable that we want to predict, such as diabetes prediction.

2. Explore the data:

This will help to understand the relationships between the different features and the target variable.

3.Feature Engineering:

Creating new variables is important for models. But we need to create a logical new variable. For this data set, some new variables were created according to BMI, Insulin and glucose variables.

```
NewBMI = pd.Series(["Underweight", "Normal", "Overweight", "Obesit
y 1", "Obesity 2", "Obesity 3"], dtype = "category")
df["NewBMI"] = NewBMI
df.loc[df["BMI"] < 18.5, "NewBMI"] = NewBMI[0]
df.loc[(df["BMI"] > 18.5) & (df["BMI"] <= 24.9), "NewBMI"] = NewBMI[
1]
df.loc[(df["BMI"] > 24.9) & (df["BMI"] <= 29.9), "NewBMI"] = NewBMI[
2]
df.loc[(df["BMI"] > 29.9) & (df["BMI"] <= 34.9), "NewBMI"] = NewBMI[
3]
```

```
df.loc[(df["BMI"] > 34.9) & (df["BMI"] <= 39.9), "NewBMI"] = NewBMI[
4]
df.loc[df["BMI"] > 39.9 ,"NewBMI"] = NewBMI[5]
df.head()
```

	PREGNANCIES	GLUCOSE	BLOOD PRESSURE	SKIN THICKNESS	INSULIN	BMI	DIABETES PEDIGREE FUNCTION	AGE	OUTCOME	NEW BMI
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	OBESITY 1
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	OVERWEIGHT
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	NORMAL
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	OVERWEIGHT
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	OBESITY 3

```
def set_insulin(row):
    if row["Insulin"] >= 16 and row["Insulin"] <= 166:
        return "Normal"
    else:
        return "Abnormal"


df = df.assign(NewInsulinScore=df.apply(set_insulin, axis=1))

df.head()
```

	PREGNANCIES	GLUCOSE	BLOOD PRESSURE	SKIN THICKNESS	INSULIN	BMI	DIABETES PEDIGREE FUNCTION	AGE	OUTCOME	NEW BMI	NEW INSULIN SCORE
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	OBESITY 1	ABNORMAL
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	OVER WEIGHT	NORMAL
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	NORMAL	ABNORMAL
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	OVER WEIGHT	NORMAL
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	OBESITY 3	ABNORMAL

```
NewGlucose = pd.Series(["Low", "Normal", "Overweight", "Secret", "High"], dtype = "category")
df["NewGlucose"] = NewGlucose
df.loc[df["Glucose"] <= 70, "NewGlucose"] = NewGlucose[0]
df.loc[(df["Glucose"] > 70) & (df["Glucose"] <= 99), "NewGlucose"] = NewGlucose[1]
df.loc[(df["Glucose"] > 99) & (df["Glucose"] <= 126), "NewGlucose"] = NewGlucose[2]
df.loc[df["Glucose"] > 126 ,"NewGlucose"] = NewGlucose[3]
df.head()
```

	PREGNANCIES	GLUCOSE	BLOOD PRESSURE	SKIN THICKNESS	INSULIN	BMI	DIABETES PEDIGREE FUNCTION	AGE	OUTCOME	NEW BMI	NEW INSULIN SCORE	NEW GLUCOSE
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	OBESITY 1	ABNORMAL	SECRET
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	OVER WEIGHT	NORMAL	NORMAL
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1	NORMAL	ABNORMAL	SECRET
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	OVER WEIGHT	NORMAL	NORMAL
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	OBESITY 3	ABNORMAL	SECRET

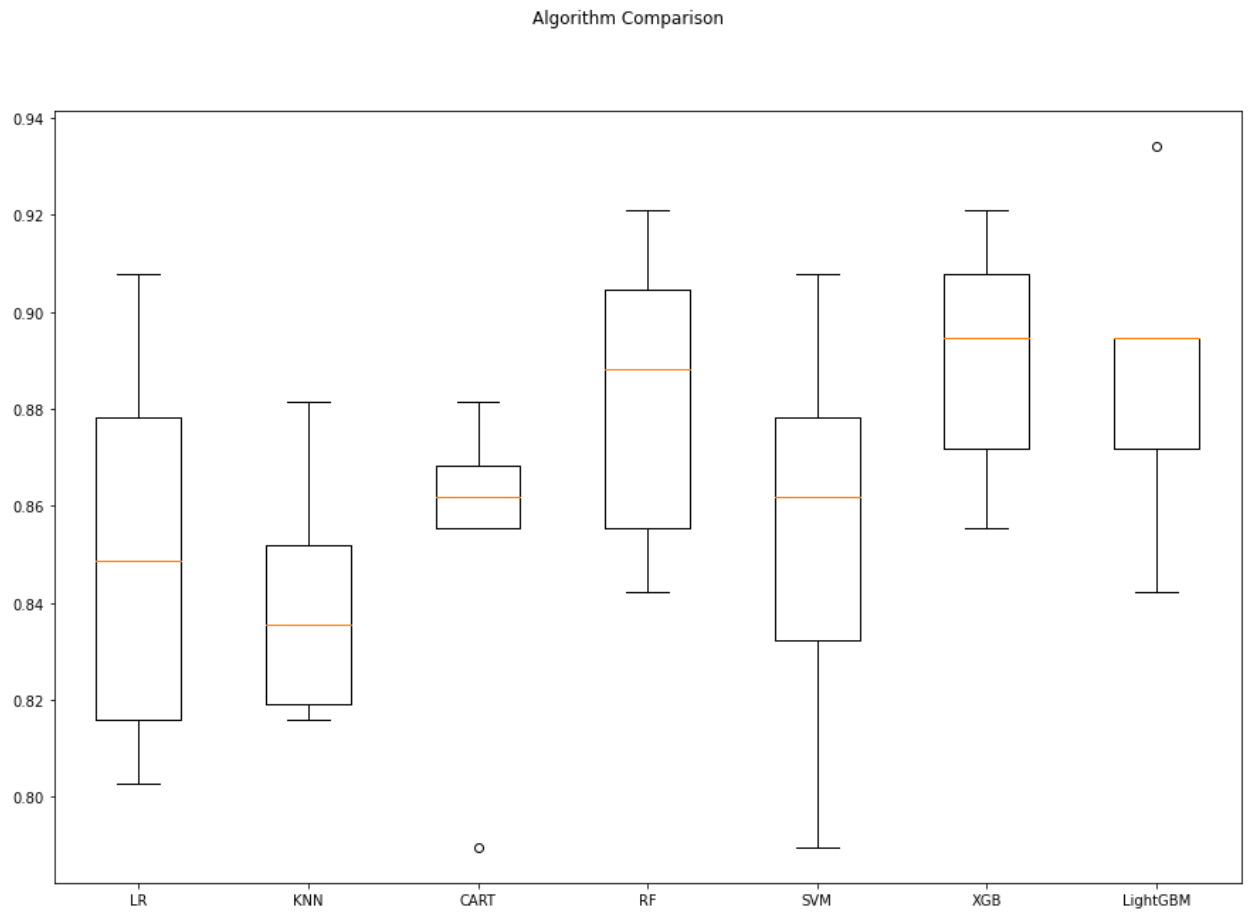
MODEL TRAINING :

The process aims to create a model that can predict diabetes status based on input data, such as a patient's characteristics, and it often involves fine-tuning and testing to improve accuracy.

1.Base Model :

```
models = []
models.append(('LR', LogisticRegression(random_state = 12345)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state = 12345)
))
models.append(('RF', RandomForestClassifier(random_state = 12345)))
models.append(('SVM', SVC(gamma='auto', random_state = 12345)))
models.append(('XGB', GradientBoostingClassifier(random_state = 123
45)))
models.append(("LightGBM", LGBMClassifier(random_state = 12345)))

results = []
```

```
names = []
for name, model in models:


 kfold = KFold(n_splits = 10, random_state = 12345)
       cv_results = cross_val_score(model, X, y, cv = 10,
 scoring= "accuracy")
       results.append(cv_results)
       names.append(name)
       msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
       print(msg)

fig = plt.figure(figsize=(15,10))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
LR: 0.848684 (0.036866)
KNN: 0.840789 (0.023866)
CART: 0.857895 (0.024826)
RF: 0.881579 (0.026316)
SVM: 0.853947 (0.036488)
XGB: 0.890789 (0.020427)
LightGBM: 0.885526 (0.024298)
```



2. Random Forests Tuning:

When using Random Forest for diabetes prediction, it's essential to tune the model's hyperparameters to achieve the best possible performance. To tune the hyperparameters for diabetes prediction, we can use techniques like grid search or random search. These methods systematically explore different hyperparameter combinations and evaluate the model's performance using cross-validation. The goal is to identify the hyperparameter settings that produce the best accuracy and reliability in predicting diabetes status based on the given features.

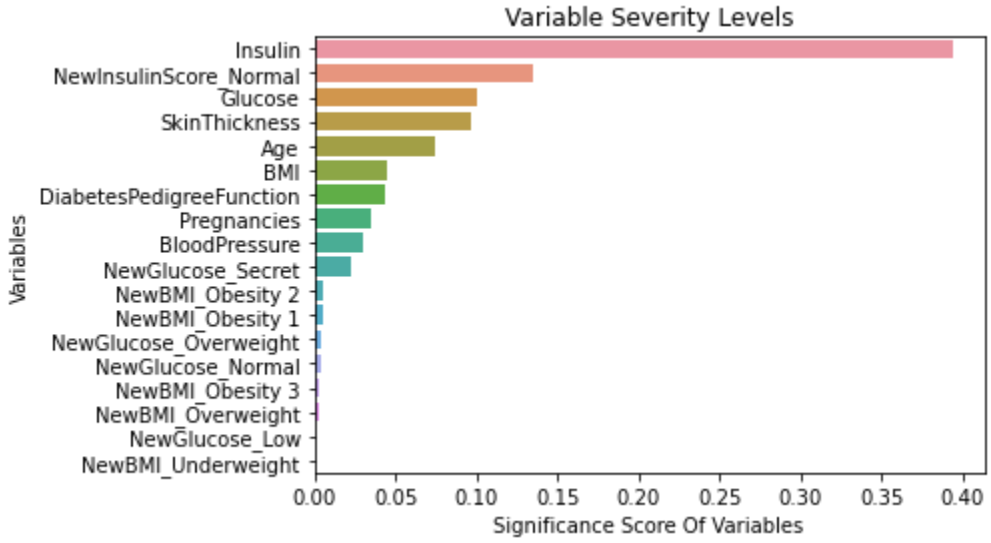
```
rf_params = {"n_estimators" :[100,200,500,1000],
           "max_features": [3,5,7],
           "min_samples_split": [2,5,10,30],
          "max_depth": [3,5,8,None]}

rf_model = RandomForestClassifier(random_state = 12345)


gs_cv = GridSearchCV(rf_model,
                     rf_params,
                     cv = 10,
                     n_jobs = -1,
                     verbose = 2).fit(X, y)
Fitting 10 folds for each of 192 candidates, totalling 1920 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   10.4s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   48.2s
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done 1450 tasks      | elapsed:  8.4min
[Parallel(n_jobs=-1)]: Done 1920 out of 1920 | elapsed: 11.4min finished

gs_cv.best_params_

{'max_depth': 8,
 'max_features': 7,
 'min_samples_split': 2,
 'n_estimators': 500}
```

21) Final Model Installation:

```
rf_tuned = RandomForestClassifier(**gs_cv.best_params_)

rf_tuned = rf_tuned.fit(X,y)

cross_val_score(rf_tuned, X, y, cv = 10).mean()
```

0.8921052631578947

```
feature_imp = pd.Series(rf_tuned.feature_importances_,
                        index=X.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Significance Score Of Variables')
plt.ylabel('Variables')
plt.title("Variable Severity Levels")
plt.show()
```



3. LightGBMTuning:

LightGBM is a gradient boosting framework that can be highly effective for diabetes prediction.

```
lgbm = LGBMClassifier(random_state = 12345)

lgbm_params = {"learning_rate": [0.01, 0.03, 0.05, 0.1, 0.5],
           "n_estimators": [500, 1000, 1500],
           "max_depth":[3,5,8]}

gs_cv = GridSearchCV(lgbm,
                     lgbm_params,
                     cv = 10,
                     n_jobs = -1,
                     verbose = 2).fit(X, y)
```

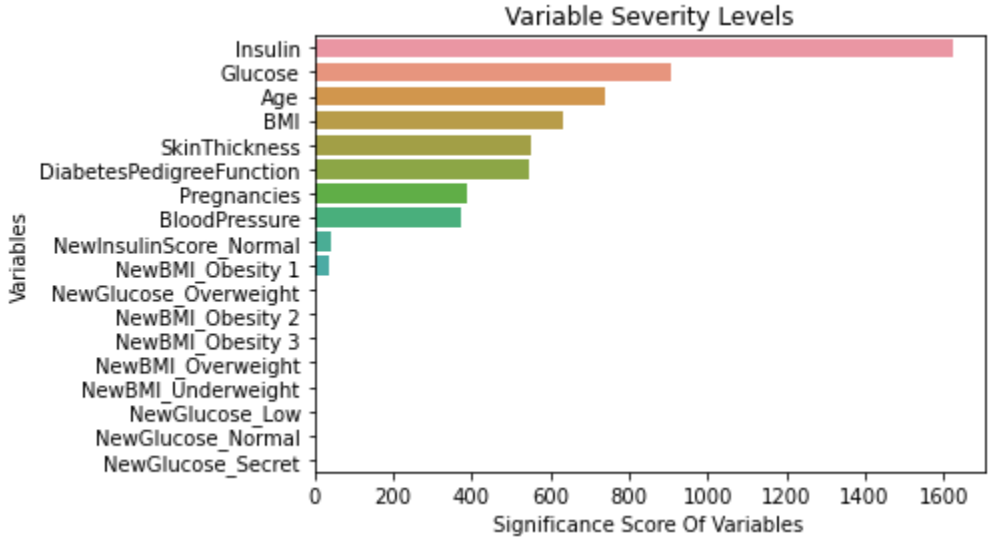
```
Fitting 10 folds for each of 45 candidates, totalling 450 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    3.7s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   25.3s
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:   57.4s
[Parallel(n_jobs=-1)]: Done 450 out of 450 | elapsed:  1.1min finished
```

```
gs_cv.best_params_
```

```
{'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 1000}
```

3.1) Final Model Installation:

```
lgbm_tuned = LGBMClassifier(**gs_cv.best_params_).fit(X,y)

cross_val_score(lgbm_tuned, X, y, cv = 10).mean()

0.8960526315789474

feature_imp = pd.Series(lgbm_tuned.feature_importances_,
                        index=X.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Significance Score Of Variables')
plt.ylabel('Variables')
plt.title("Variable Severity Levels")
plt.show()
```



4. XGBoost Tuning:

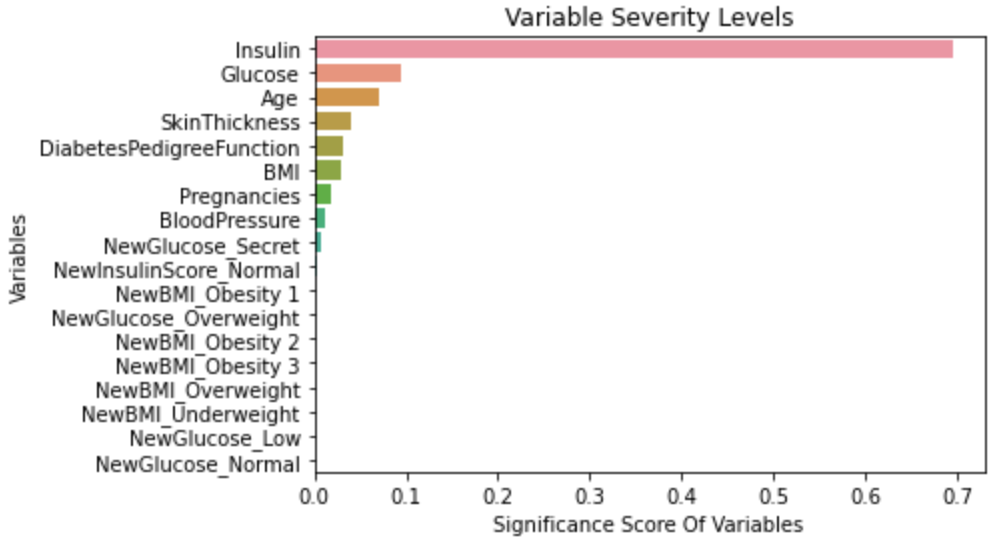
XGBoost is a powerful gradient boosting algorithm that can be effectively tuned for diabetes prediction.

```
xgb = GradientBoostingClassifier(random_state = 12345)

xgb_params = {
    "learning_rate": [0.01, 0.1, 0.2, 1],
    "min_samples_split": np.linspace(0.1, 0.5, 10),
    "max_depth":[3,5,8],
    "subsample":[0.5, 0.9, 1.0],
    "n_estimators": [100,1000]}

xgb_cv_model  = GridSearchCV(xgb,xgb_params, cv = 10, n_jobs = -1, v
erbose = 2).fit(X, y)
Fitting 10 folds for each of 720 candidates, totalling 7200 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    4.0s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   42.3s
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:  5.4min
[Parallel(n_jobs=-1)]: Done 1450 tasks      | elapsed:  8.4min
[Parallel(n_jobs=-1)]: Done 1977 tasks      | elapsed: 11.6min
[Parallel(n_jobs=-1)]: Done 2584 tasks      | elapsed: 14.8min
[Parallel(n_jobs=-1)]: Done 3273 tasks      | elapsed: 19.3min
[Parallel(n_jobs=-1)]: Done 4042 tasks      | elapsed: 23.6min
[Parallel(n_jobs=-1)]: Done 4893 tasks      | elapsed: 28.7min
[Parallel(n_jobs=-1)]: Done 5824 tasks      | elapsed: 34.5min
[Parallel(n_jobs=-1)]: Done 6837 tasks      | elapsed: 40.9min
```

```
[Parallel(n_jobs=-1)]: Done 7200 out of 7200 | elapsed: 43.2min
 finished


xgb_cv_model.best_params_

{'learning_rate': 0.1,
 'max_depth': 5,
 'min_samples_split': 0.1,
 'n_estimators': 100,
 'subsample': 1.0}
```

4.1) Final Model Installation:

```
xgb_tuned = GradientBoostingClassifier(**xgb_cv_model.
best_params_).fit(X,y)

cross_val_score(xgb_tuned, X, y, cv = 10).mean()

0.9013157894736843

feature_imp = pd.Series(xgb_tuned.feature_importances_,
                        index=X.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Significance Score Of Variables')
plt.ylabel('Variables')
plt.title("Variable Severity Levels")
plt.show()
```

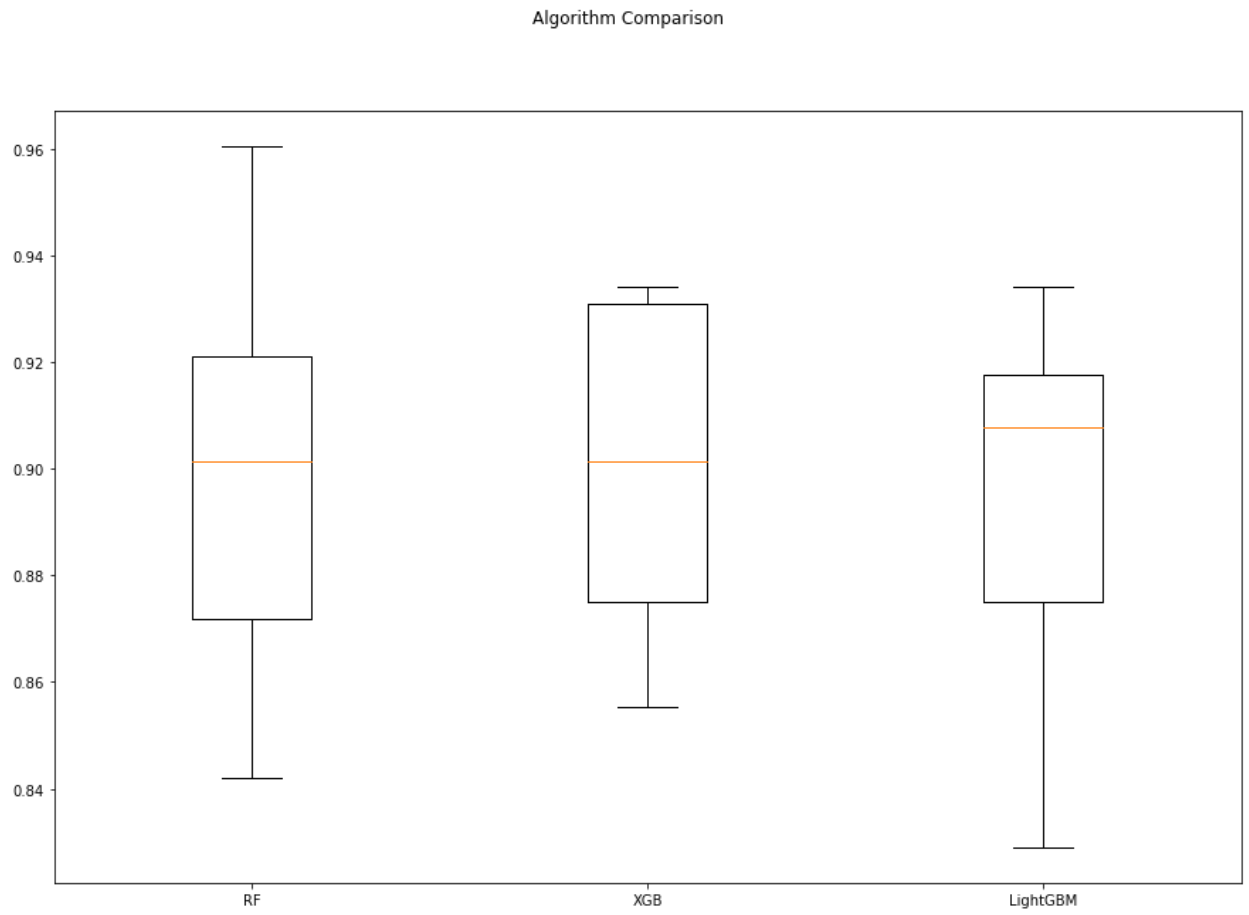


MODEL EVALUATION :

Model evaluation for diabetes prediction is a critical step in assessing how well the machine learning model performs on a given dataset. Remember that the choice of evaluation metrics depends on the context and goals of diabetes prediction system. Some metrics may be more critical than others, depending on whether false positives or fals e negatives have more significant consequences. The choice of metrics should align with the application's objectives and priorities.

```
models = []
models.append(('RF', RandomForestClassifier(random_state = 12345,
 max_depth = 8, max_features = 7, min_samples_split = 2,
n_estimators = 500)))
models.append(('XGB', GradientBoostingClassifier(random_state = 123
45, learning_rate = 0.1, max_depth = 5, min_samples_split = 0.1,
 n_estimators = 100, subsample = 1.0)))
models.append(("LightGBM", LGBMClassifier(random_state = 12345,
```

```
 learning_rate = 0.01,  max_depth = 3, n_estimators = 1000)))


results = []
names = []
for name, model in models:

        kfold = KFold(n_splits = 10, random_state = 12345)
        cv_results = cross_val_score(model, X, y, cv = 10,
scoring= "accuracy")
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

fig = plt.figure(figsize=(15,10))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
RF: 0.897368 (0.034211)
XGB: 0.901316 (0.028373)
LightGBM: 0.896053 (0.033000)
```



CONCLUSION :

The aim of this study was to create classification models for the diabetes data set and to predict whether a person is sick by establishing models and to obtain maximum validation scores in the established models. The work done is as follows:

1) Diabetes Data Set read.

2) With Exploratory Data Analysis; The data set's structural data were checked. The types of variables in the dataset were examined. Size information of the dataset was accessed. The 0 values in the data set are missing values. Primarily these 0 values were replaced with NaN values. Descriptive statistics of the data set were examined.

3) Data Preprocessing section; df for: The NaN values missing observations were filled with the median values of whether each variable was sick or not. The outliers were determined by LOF and dropped. The X variables were standardized with the rubost method..

4) During Model Building; Logistic Regression, KNN, SVM, CART, Random Forests, XGBoost, LightGBM like using machine learning models Cross Validation Score were calculated. Later Random Forests, XGBoost, LightGBM hyperparameter optimizations optimized to increase Cross Validation value.

5) Result; The model created as a result of XGBoost hyperparameter optimization became the model with the lowest Cross Validation Score value. (0.90)