MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY , BHOPAL

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

MACHINE LEARNING LAB (CS 5508)

PROF. RAJESH PATERIYA

SUBMITED BY:

NAME : Ankit Kumar

SCHOLAR NO. : 201112471

SECTION : CSE 3

DATE - 30-09-2022

ASSIGNMENT - 06

QUES - Develop a machine learning model for Pima-Indians-diabetes data set using SVM , Naive Bayes, Linear Regression, and random forest .Tune the hyper parameter to get the higher accuracy and draw graph for different accuracy .Take 5-6 variation of hyper parameter and plot the graph of confusion matrices like accuracy, recall , precision,F1 score etc.

DATASET DESCRIPTION

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables	Description
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration in an oral glucose tolerance test
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	Triceps skinfold thickness (mm)
Insulin	Two hour serum insulin
BMI	Body Mass Index
DiabetesPedigreeFunction	Diabetes pedigree function
Age	Age in years
Outcome	Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

Observations & Exploratory Data Analysis:

1. Age, Insulin, DiabetesPedigreeFunction and Pregnancies are right skewed.
2. .Zero values in blood pressure, BMI, Insulin and Glocuse clearly stands out in the plot
3. After removing zeros for non-zero expected columns, we see that except Insulin which is highly right skewed, all other are near to gaussian distribution.
4. Except for Insulin, for rest of other non-zero columns, we can take mean value.For Insulin, we took median value to fill
5. In Data type count plot, we can see that there are 2 int type columns and 7 float type

CODE SECTION & OUTPUTS :

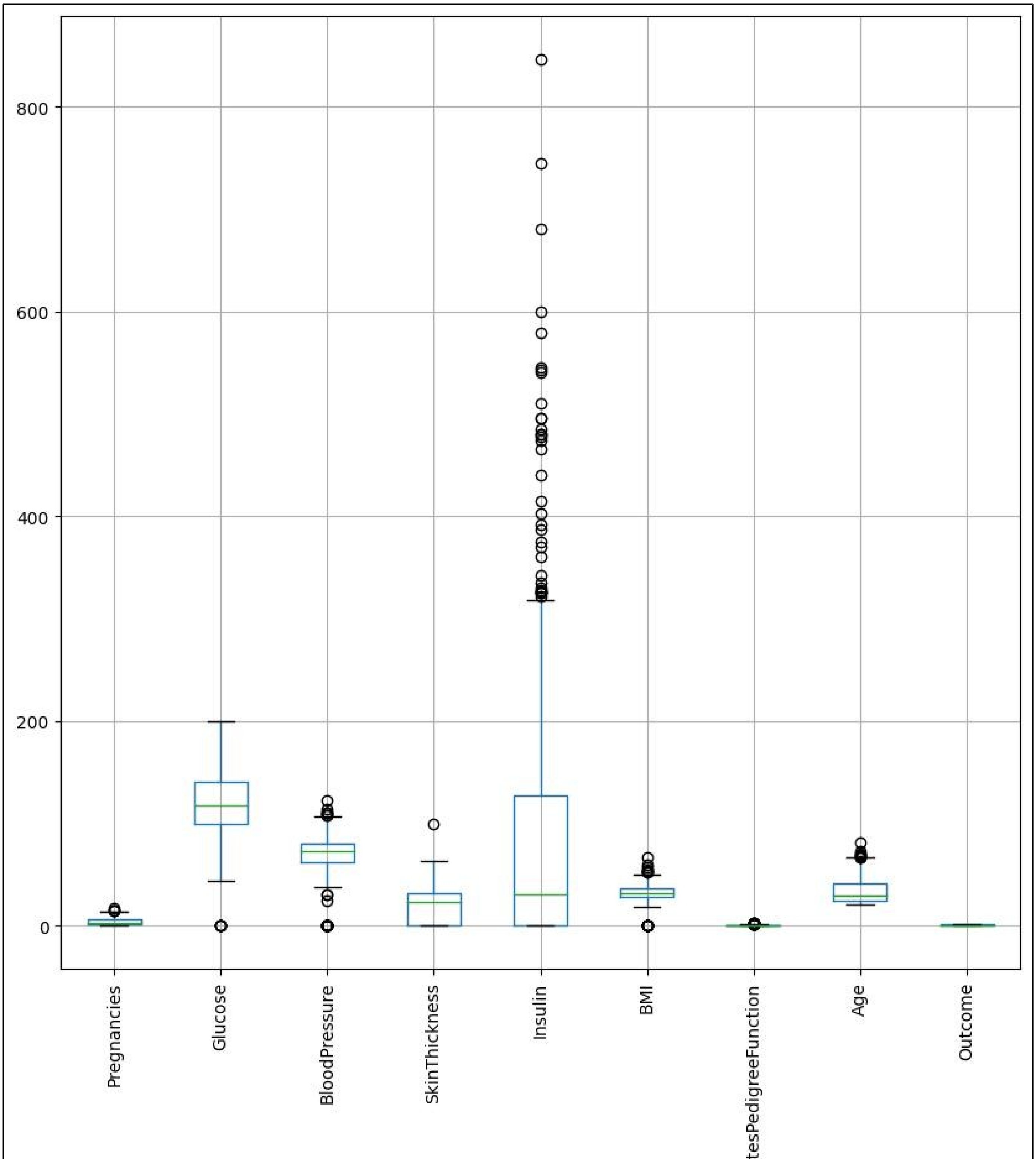
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import copy
data_raw = pd.read_csv('../input/pima-indians-diabetes-database/diabetes.csv')
data_raw.dtypes
data_raw.shape
data_raw.sample(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

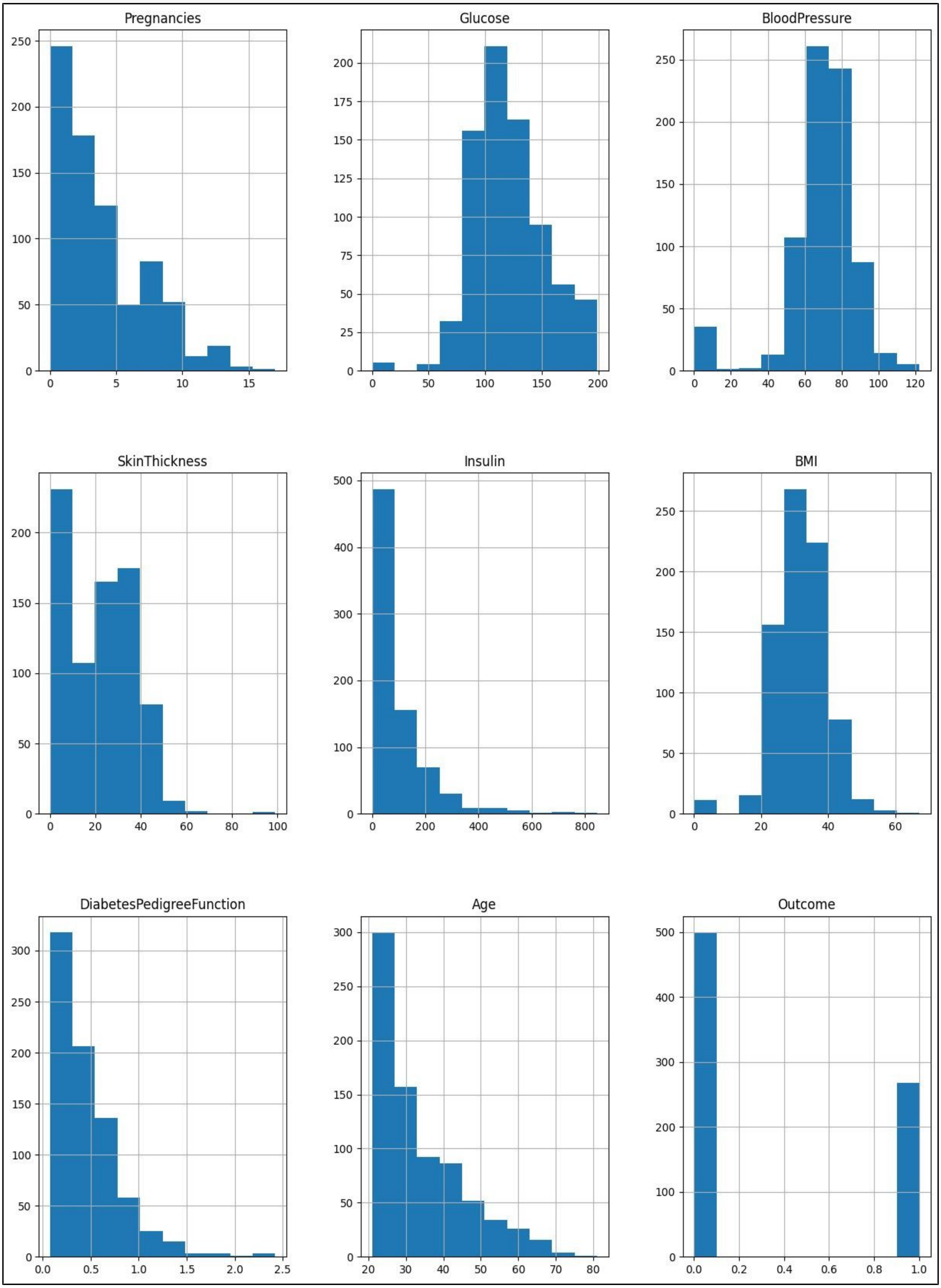
```
data_raw.describe()
```

	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

```
data_raw.boxplot(figsize=(10,10), rot=90)
```



```
data_raw.hist(figsize=(15,20), )
```

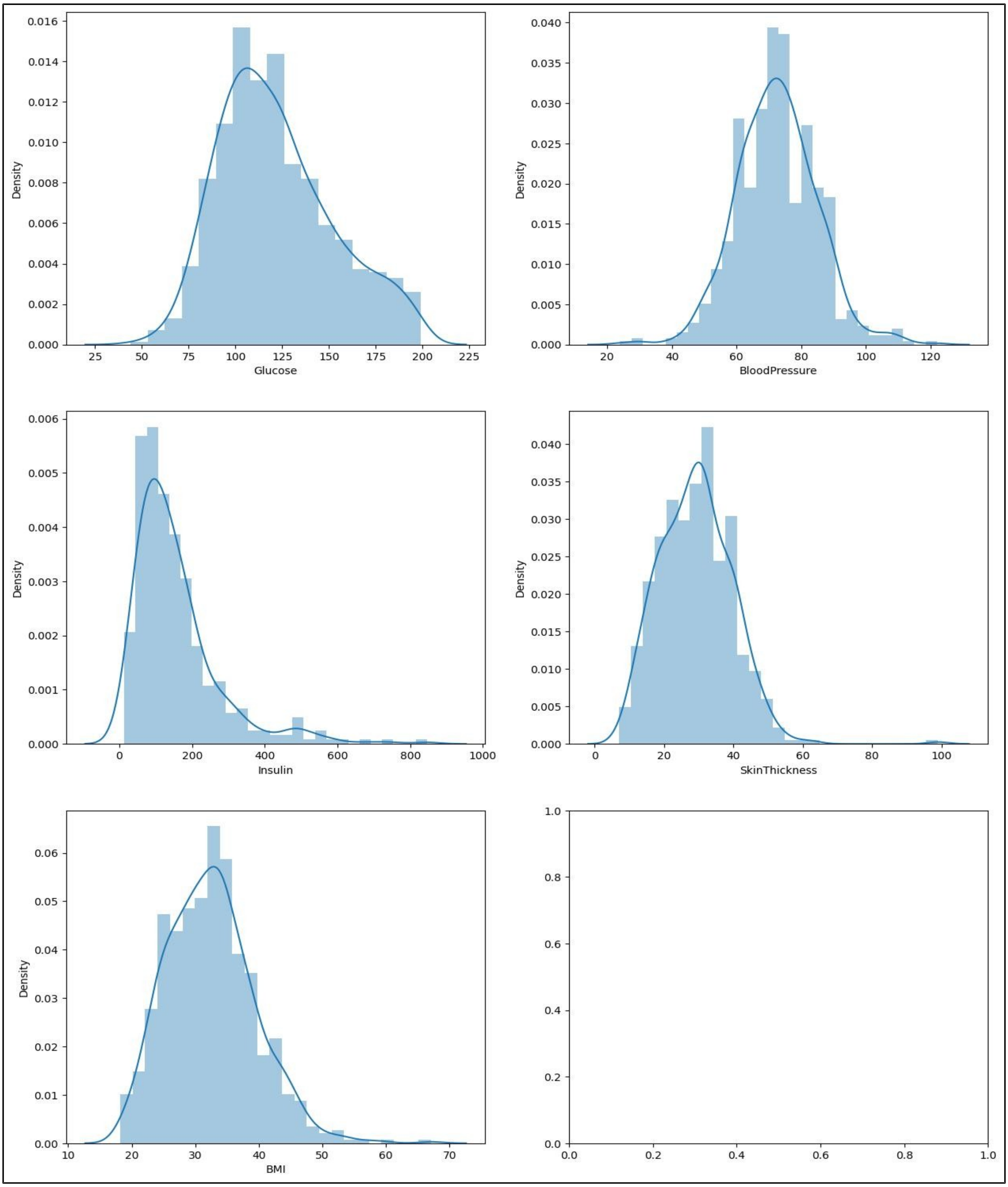


Treating Zero valued columns

```
not_allowed_zero_cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
data = copy.deepcopy(data_raw)

data[not_allowed_zero_cols] = data[not_allowed_zero_cols].replace(0, np.NaN)

data.isnull().sum()
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

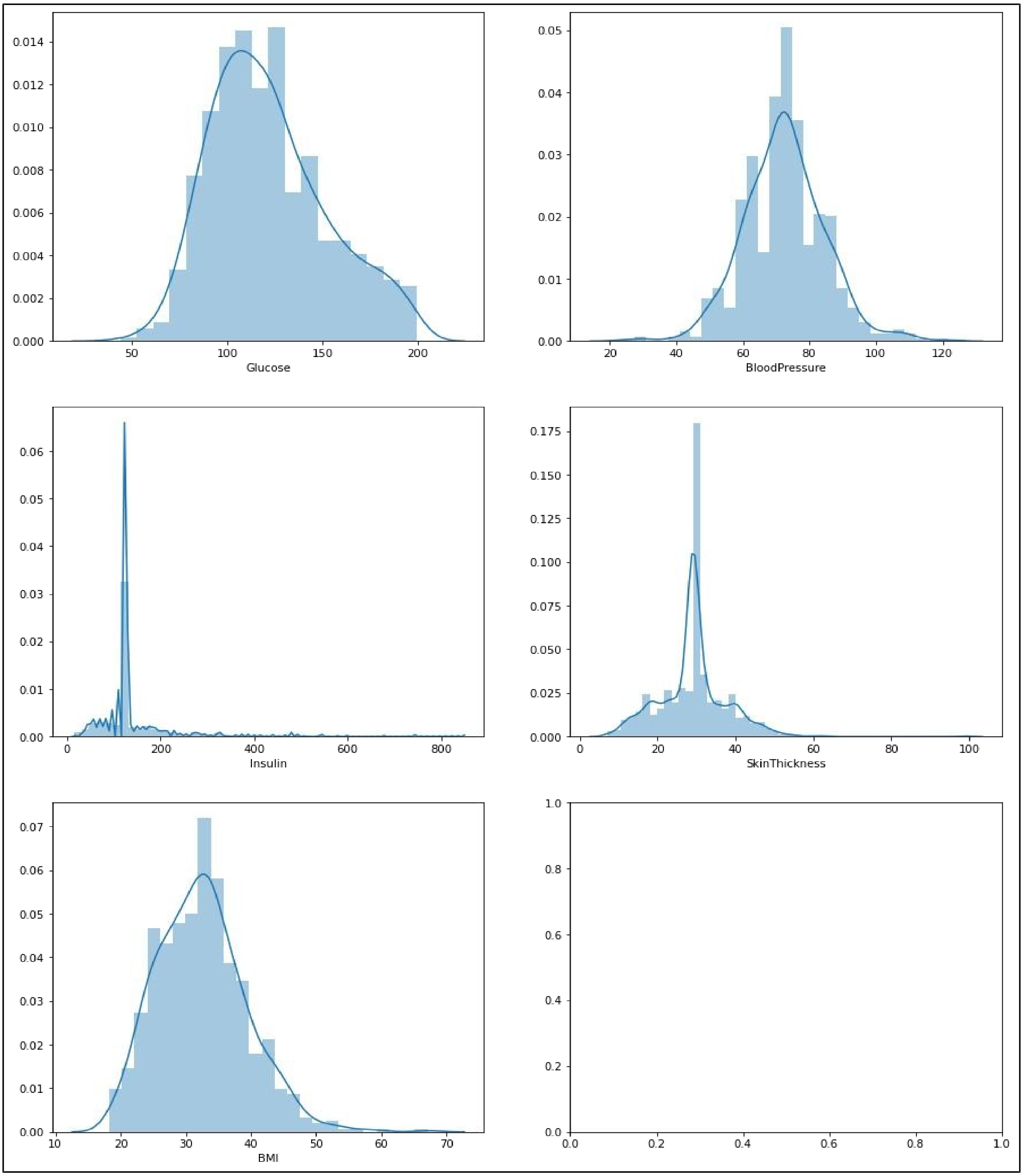
```
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(15,20))

sns.distplot(data.Glucose, ax=ax[0][0])

sns.distplot(data.BloodPressure, ax=ax[0][1])

sns.distplot(data.Insulin, ax=ax[1][0])

sns.distplot(data.SkinThickness, ax=ax[1][1])

sns.distplot(data.BMI, ax=ax[2][0])
```




```
data['Glucose'].fillna(data.Glucose.mean(), inplace=True)
data['BloodPressure'].fillna(data.BloodPressure.mean(), inplace=True)
data['BMI'].fillna(data.BMI.mean(), inplace=True)
data['SkinThickness'].fillna(data.SkinThickness.mean(), inplace=True)
data['Insulin'].fillna(data.Insulin.median(), inplace=True)
```

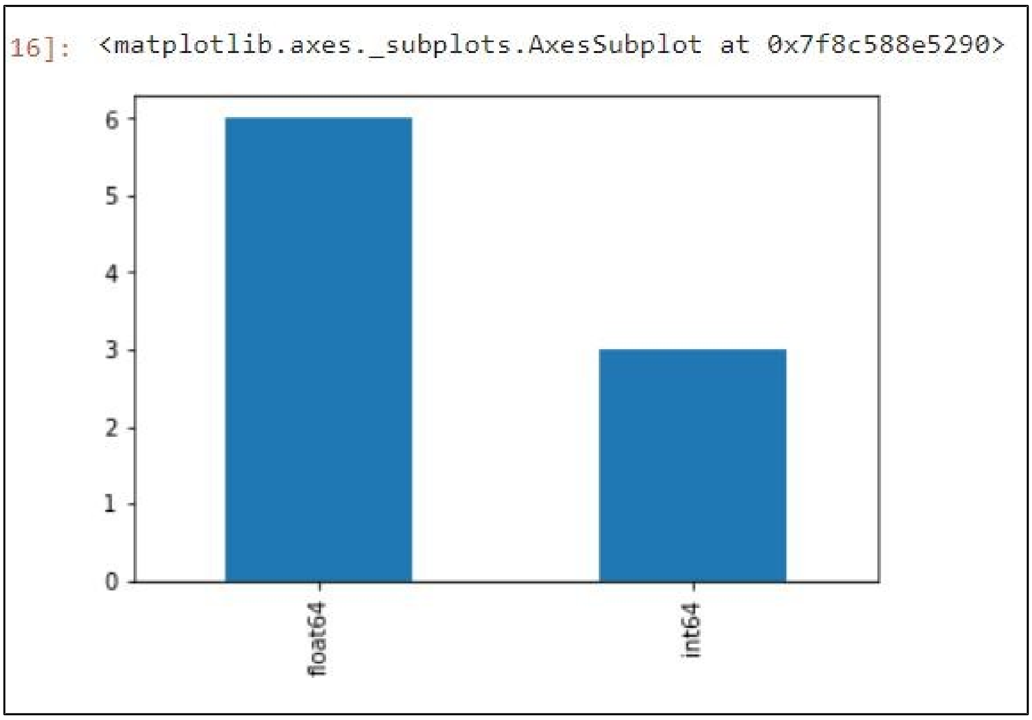
Plots after filling the NaN values.

```
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(15,20))
sns.distplot(data.Glucose, ax=ax[0][0])
sns.distplot(data.BloodPressure, ax=ax[0][1])
sns.distplot(data.Insulin, ax=ax[1][0])
sns.distplot(data.SkinThickness, ax=ax[1][1])
sns.distplot(data.BMI, ax=ax[2][0])
```



Plot data types

```
data.dtypes.value_counts().plot(kind='bar')
```



Observations

It is an imbalanced dataset where positive outcomes are almost half of the negative outcomes. While creating model, we need to balance the outcomes either by oversampling the minority class or undersampling of majority class. Other workaround could be to do a weighted computation while training the model.

Pair plot analysis

- BMI and Skin thickness have a positive correlation
- Insulin and Glucose have a positive correlation.
- Rest other fields are uncorrelated or very weakly correlated.

Correlation Analysis

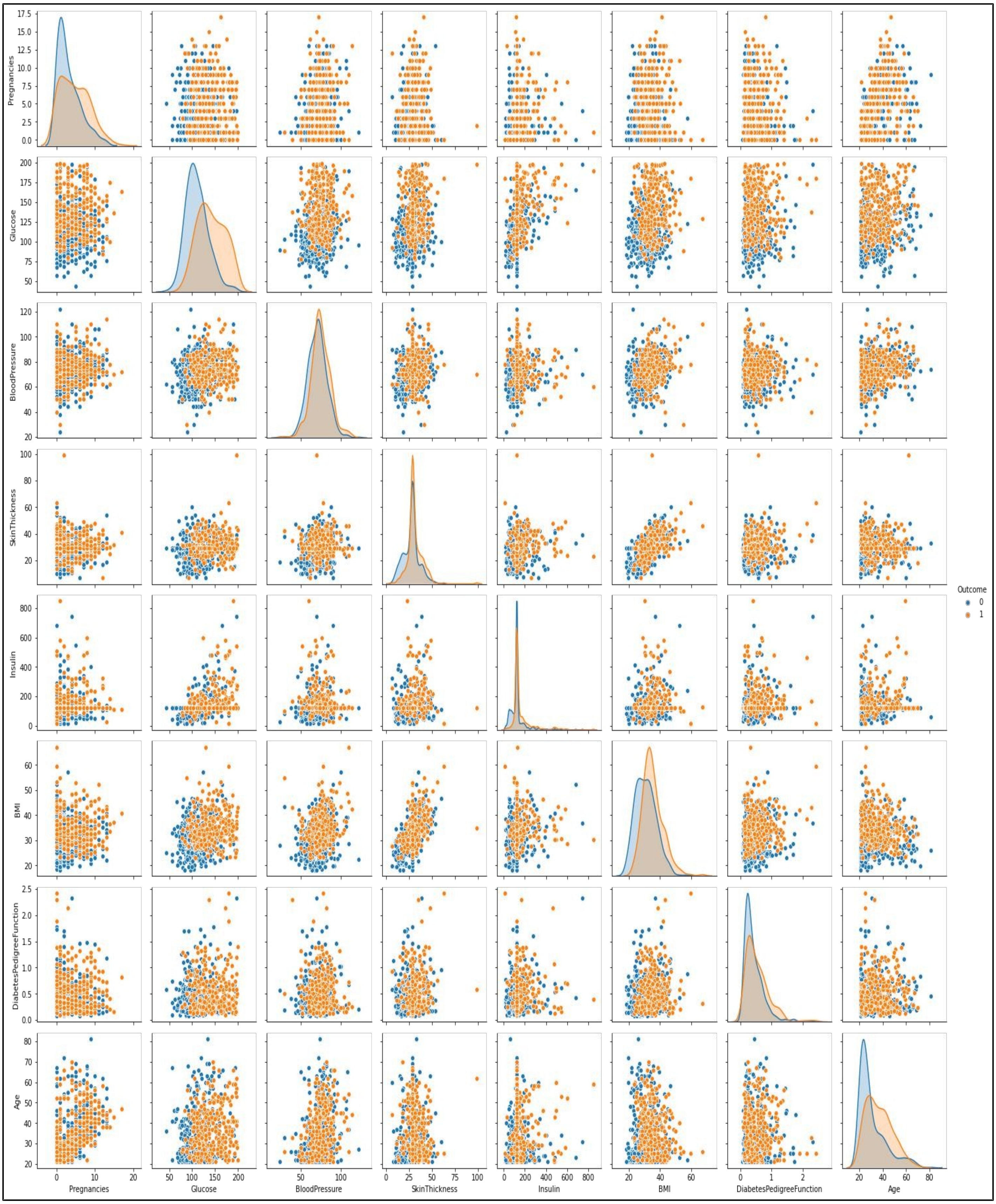
- There is no strong correlation between any two fields
- The BMI-Skinthickness and Insulin-Glucose are the highest correlated in the set but they are moderately correlated
- Outcome is moderately correlated to Glucose

Checking Data balance

```
sns.countplot(data.Outcome, ).set(title="Data Imbalance Check")
```

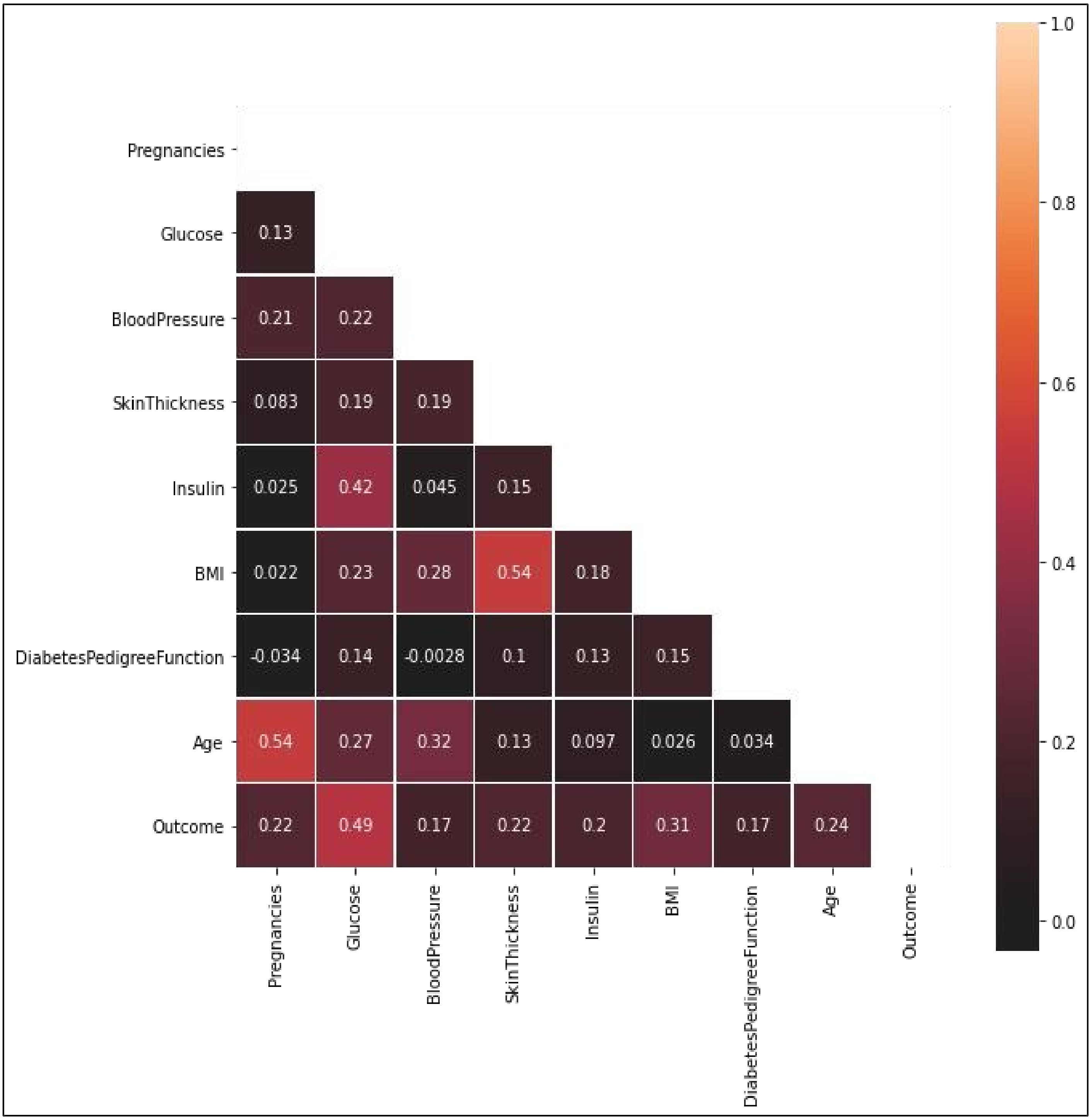
Pair plot analysis

```
sns.pairplot(data, hue='Outcome')
```



Correlation analysis

```
cor = data.corr()
mask = np.triu(np.ones_like(cor, dtype=np.bool))
plt.figure(figsize=(10,10))
sns.heatmap(cor, mask=mask, center=0,
          square=True, linewidths=.5, annot=True)
```



OBSERVATIONS FROM DATA ANALYSIS

This is a binary data classification problem where depending on the all the features, the model has to predict whether a person have diabetes or not. We have several ways to build the model for binary/multi-class classification. Few of them are listed below:

1. Logistics Regression
2. Naive Bayes classification
3. Stochastic Gradient Descent
4. K-Nearest Neighbours
5. Decision Tree
6. Random Forest
7. SVM

We are going to build four models and compare their performance on test and train dataset. We will tune the models if there is need to tune. We will use K-Fold Cross Validation to validate the models. We will plot all the models stats together and compare their performance. Of all the tuned models, we will pick up the best model. The step by step procedure can be followed below:

A. Common Terminalogy
B. Data Scaling & Splitting
C. Logistic Regression
D. Naive Bayes Classification
E. Random Forest
F. K Nearest Neighbours
G. Putting it all together
    - ROC AUC Curves
    - Model Comparison

From Model Comparison, we find that KNN is the most stable classifier. All the parameters are quite good. It has best accuracy, auc, precision and f1_score of all the models.

If we are looking for a highly sensitive model, we can take logistic regression model, which has the highest recall.

Scaling and splitting the data

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report, f1_score, accuracy_score, mean_squared_error, roc_auc_score, confusion_matrix, roc_curve, recall_score, precision_score, f1_score

from sklearn.preprocessing import StandardScaler

X_scaled = StandardScaler().fit_transform(data.drop(['Outcome'], axis='columns'))

X_train, X_test, y_train, y_test = train_test_split(X_scaled, data.Outcome, random_state=123, test_size=.2)
```

1- LINEAR REGRESSION

```
from sklearn.linear_model import LogisticRegression
lr_clf= LogisticRegression(class_weight='balanced', random_state=123, max_iter=500)
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
lr_model_vals = dict(accuracy=accuracy_score(y_test, lr_pred),
                auc=roc_auc_score(y_test, lr_pred),
                recall=recall_score(y_test, lr_pred),
                precision=precision_score(y_test, lr_pred),
                f1_score = f1_score(y_test, lr_pred),
                )
```

AUC ROC Curve parmas computation for Linear Regression

```
y_pred_prob_lr = lr_clf.predict_proba(X_test)[:, 1]
fpr_lr, tpr_lr , th_lr = roc_curve(y_test, y_pred_prob_lr)
gmean_lr = np.sqrt(tpr_lr * (1-fpr_lr))
ix_lr = np.argmax(gmean_lr)
```

Tuning by AUC_ROC Threshold

The geometric mean between TPR and FPR is an optimum value which is max for any give tpr, fpr. If our focus is to create a model that predicts both sides, then this threshold value could be choosen to be optimum threshold. The optimam threshold to classify True or False, we get at .364

```
th_lr[np.argmax(gmean_lr)]
```

```
Out[30]:
        0.36454228797001864
```

```
y_roc_pred_lr = [0 if pred < th_lr[ix_lr] else 1 for pred in y_pred_prob_lr ]
print("Test classification Report With tuned threshold")
print(classification_report(y_test, y_roc_pred_lr)  )
print("Test classification Report Without  tuned threshold")
print(classification_report(y_test, lr_pred) )
```

```
Test classification Report With  tuned threshold
              precision    recall  f1-score   support

           0       0.94      0.66      0.77        96
           1       0.62      0.93      0.74        58

    accuracy                           0.76       154
   macro avg       0.78      0.79      0.76       154
weighted avg       0.82      0.76      0.76       154

Test classification Report Without  tuned threshold
              precision    recall  f1-score   support

           0       0.83      0.79      0.81        96
           1       0.68      0.72      0.70        58

    accuracy                           0.77       154
   macro avg       0.75      0.76      0.75       154
weighted avg       0.77      0.77      0.77       154
```

```
fpr_tlr, tpr_tlr , th_tlr = roc_curve(y_test, y_roc_pred_lr)
gmean_tlr = np.sqrt(tpr_tlr * (1-fpr_tlr))
ix_tlr = np.argmax(gmean_tlr)
tlr_model_vals = dict(accuracy=accuracy_score(y_test, y_roc_pred_lr),
                auc=roc_auc_score(y_test, y_roc_pred_lr),
                recall=recall_score(y_test, y_roc_pred_lr),
                precision=precision_score(y_test, y_roc_pred_lr),
                f1_score = f1_score(y_test, y_roc_pred_lr),
                )
```

2 - NAIVE BAYES CLASSIFICATION

```
from sklearn.naive_bayes import GaussianNB
gnb_clf = GaussianNB()
gnb_clf.fit(X_train, y_train)
gnb_pred = gnb_clf.predict(X_test)
gnb_model_vals = dict(accuracy=accuracy_score(y_test, gnb_pred),
                auc=roc_auc_score(y_test, gnb_pred),
                recall=recall_score(y_test, gnb_pred),
                precision=precision_score(y_test, gnb_pred),
                f1_score = f1_score(y_test, gnb_pred),
                )
```

AUC ROC for Naive Bayes classifier

```
y_pred_prob_gnb = gnb_clf.predict_proba(X_test)[:, 1]
fpr_nb, tpr_nb , th_nb = roc_curve(y_test, y_pred_prob_gnb)
gmean_nb = np.sqrt(tpr_nb * (1-fpr_nb))ix_nb = np.argmax(gmean_nb)
print("Train Classification Report")
print(classification_report(y_train, gnb_clf.predict(X_train))  )
```

```
Train Classification Report
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       404
           1       0.64      0.59      0.61       210

    accuracy                           0.74       614
   macro avg       0.72      0.71      0.71       614
weighted avg       0.74      0.74      0.74       614
```

```
print("Test Classification Report")
print(classification_report(y_test, gnb_clf.predict(X_test))  )
```

```
Test Classification Report
              precision    recall  f1-score   support

           0       0.79      0.83      0.81        96
           1       0.70      0.64      0.67        58

    accuracy                           0.76       154
   macro avg       0.75      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154
```

3 - RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
rf_model_vals = dict(accuracy=accuracy_score(y_test, rf_pred),
                     auc=roc_auc_score(y_test, rf_pred),
                     recall=recall_score(y_test, rf_pred),
                     precision=precision_score(y_test, rf_pred),
                     f1_score = f1_score(y_test, rf_pred),
                     )
y_pred_prob_rf = rf_clf.predict_proba(X_test)[:, 1]
fpr_rf, tpr_rf , th_rf = roc_curve(y_test, y_pred_prob_rf)
gmean_rf = np.sqrt(tpr_rf * (1-fpr_rf))ix_rf = np.argmax(gmean_rf)
print("\t\tTrain Classification Report\n")
print(classification_report(y_train, rf_clf.predict(X_train))  )
```

```
                Train Classification Report

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       404
           1       1.00      1.00      1.00       210

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614
```

```
print("\t\tTest Classification Report\n")
print(classification_report(y_test, rf_clf.predict(X_test))  )
```

```
                Test Classification Report

              precision    recall  f1-score   support

           0       0.81      0.82      0.82        96
           1       0.70      0.69      0.70        58

    accuracy                           0.77       154
   macro avg       0.76      0.76      0.76       154
weighted avg       0.77      0.77      0.77       154
```

Observation from Random Forest

From the looks of training data, we can say that Random forest has overfitted. Due to overfitting, it may show very good responses but ultimately it is not a good model. We will tune the parmas for this. We will tune on Cost parameter and see what cost function makes the training and test data accuracy comparable.

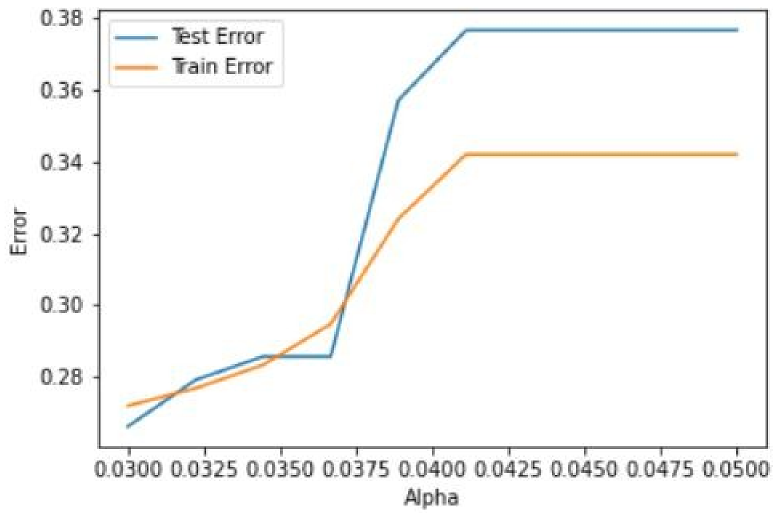
In below code, we see only one iteration, but before coming to below values, I have done several iterations and compared trin and test errors to arrive at optimum cost value. The below iteration is to arrive at more precise cost value.

```
alphas=[]test=[]train=[]for alpha in np.linspace(.03, .05, 10):
    rf = RandomForestClassifier(ccp_alpha=alpha, random_state=123)
    rf.fit(X_train, y_train)
    y_train_predicted = rf.predict(X_train)
    y_test_predicted = rf.predict(X_test)
    mse_train = mean_squared_error(y_train, y_train_predicted)
    mse_test = mean_squared_error(y_test, y_test_predicted)
    alphas.append(alpha)
    test.append(mse_test)
    train.append(mse_train)
    print("Alpha: {} Train mse: {} Test mse: {}".format(alpha, mse_train, mse_test))
     score=pd.DataFrame({'alpha': alphas, 'test':test, 'train': train})
```

```
Alpha: 0.03 Train mse: 0.2719869706840391 Test mse: 0.2662337662337662
Alpha: 0.03222222222222222 Train mse: 0.2768729641693811 Test mse: 0.2792207792207792
Alpha: 0.034444444444444444 Train mse: 0.28338762214983715 Test mse: 0.2857142857142857
Alpha: 0.03666666666666667 Train mse: 0.2947882736156352 Test mse: 0.2857142857142857
Alpha: 0.03888888888888889 Train mse: 0.3241042345276873 Test mse: 0.35714285714285715
Alpha: 0.04111111111111111 Train mse: 0.34201954397394135 Test mse: 0.37662337662337664
Alpha: 0.043333333333333335 Train mse: 0.34201954397394135 Test mse: 0.37662337662337664
Alpha: 0.04555555555555556 Train mse: 0.34201954397394135 Test mse: 0.37662337662337664
Alpha: 0.04777777777777778 Train mse: 0.34201954397394135 Test mse: 0.37662337662337664
Alpha: 0.05 Train mse: 0.34201954397394135 Test mse: 0.37662337662337664
```

```
plt.plot(score.alpha, score.test)

plt.plot(score.alpha, score.train)

plt.legend(['Test Error', 'Train Error'])

plt.xlabel('Alpha')

plt.ylabel('Error')
```

```
Text(0, 0.5, 'Error')
```




```
from sklearn.model_selection import RandomizedSearchCV
# Create the random gridrandom_grid = { 'ccp_alpha': np.linspace(.03, .05, 10),
               'n_estimators': [int(x) for x in np.linspace(start = 200, stop = 1000, num = 5)],
              'max_features': ['auto', 'sqrt'],
              'max_depth': [int(x) for x in np.linspace(5, 55, num = 10)],
              'min_samples_split': [5, 10, 12],
              'min_samples_leaf': [3,5,7,10],
              }



rf_clf_cv = RandomForestClassifier(class_weight="balanced", random_state=123)rscv = RandomizedSearc
hCV(estimator=rf_clf_cv, param_distributions=random_grid, cv=3, scoring='f1_weighted')
rscv.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=3,
                   estimator=RandomForestClassifier(class_weight='balanced',
                                                    random_state=123),
                   param_distributions={'ccp_alpha': array([0.03      , 0.03222222, 0.03444444, 0.03666667, 0.03888889,
       0.04111111, 0.04333333, 0.04555556, 0.04777778, 0.05      ]),
                                        'max_depth': [5, 10, 16, 21, 27, 32, 38,
                                                      43, 49, 55],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [3, 5, 7, 10],
                                        'min_samples_split': [5, 10, 12],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000]},
                   scoring='f1_weighted')
```

```
rscv.best_estimator_
```

```
RandomForestClassifier(ccp_alpha=0.03888888888888889, class_weight='balanced',
                       max_depth=49, max_features='sqrt', min_samples_leaf=3,
                       min_samples_split=10, n_estimators=1000,
                       random_state=123)
```

```
print("\t\tTest Classification Report\n")
print(classification_report(y_test, rscv.predict(X_test))  )
```

```
                Test Classification Report

              precision    recall  f1-score   support

           0       0.86      0.78      0.82        96
           1       0.69      0.79      0.74        58

    accuracy                           0.79       154
   macro avg       0.77      0.79      0.78       154
weighted avg       0.80      0.79      0.79       154
```

```
print("\t\tTrain Classification Report\n")
print(classification_report(y_train, rscv.predict(X_train))  )
```

```
                Train Classification Report

              precision    recall  f1-score   support

           0       0.86      0.77      0.81       404
           1       0.63      0.77      0.69       210

    accuracy                           0.77       614
   macro avg       0.75      0.77      0.75       614
weighted avg       0.78      0.77      0.77       614
```

```
tuned_rf_model_vals = dict(accuracy=accuracy_score(y_test, rscv.predict(X_test)),
                    auc=roc_auc_score(y_test, rscv.predict(X_test)),
                    recall=recall_score(y_test, rscv.predict(X_test)),
                    precision=precision_score(y_test, rscv.predict(X_test)),
                    f1_score = f1_score(y_test, rscv.predict(X_test)),
                    )
y_pred_prob_trf = rscv.predict_proba(X_test)[:, 1]
fpr_trf, tpr_trf , th_trf = roc_curve(y_test, y_pred_prob_trf)
gmean_trf = np.sqrt(tpr_trf * (1-fpr_trf))
ix_trf = np.argmax(gmean_trf)
```

```python
import sklearn.metrics
sorted(sklearn.metrics.SCORERS.keys())
```

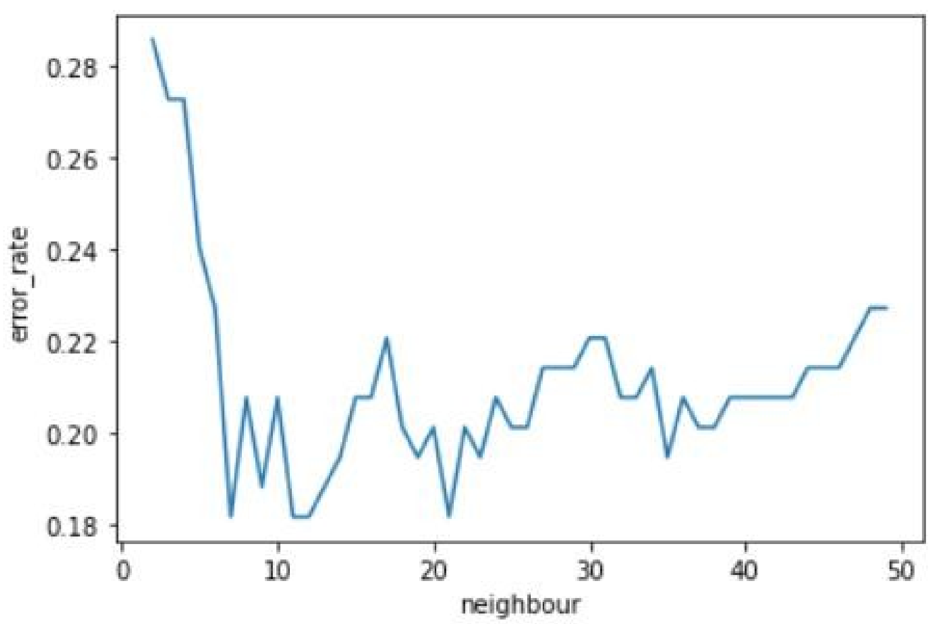
```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'max_error',
 'mutual_info_score',
 'neg_brier_score',
 'neg_log_loss',
 'neg_mean_absolute_error',
 'neg_mean_gamma_deviance',
 'neg_mean_poisson_deviance',
 'neg_mean_squared_error',
 'neg_mean_squared_log_error',
 'neg_median_absolute_error',
 'neg_root_mean_squared_error',
 'normalized_mutual_info_score',
 'precision',
 'precision_macro',
 'precision_micro',
 'precision_samples',
 'precision_weighted',
 'r2',
 'recall',
 'recall_macro',
 'recall_micro',
 'recall_samples',
 'recall_weighted',
 'roc_auc',
 'roc_auc_ovo',
 'roc_auc_ovo_weighted',
 'roc_auc_ovr',
 'roc_auc_ovr_weighted',
 'v_measure_score']
```

4 - K-Nearest Neighbour

We are capturing rmse, error_rate and accuracy for a range of nearest neighbours. We will plot all of them to observer nearest neighbours. We observe that error_rate and rmse give same plot while accuracy gives a mirror image of other two.

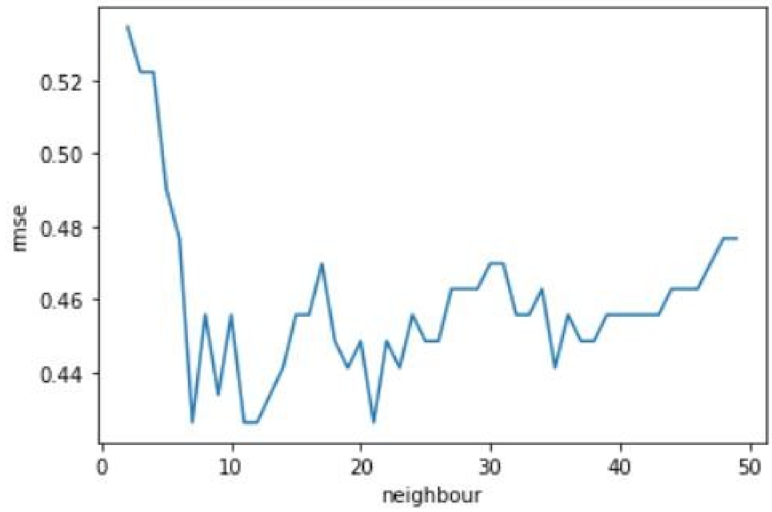
```
from sklearn.neighbors import KNeighborsClassifier
nbr = []error_rmse = []error_rate = []accuracy = []for n in range(2, 50):
    knn_clf = KNeighborsClassifier(n_neighbors=n, weights='distance')
    knn_clf.fit(X_train, y_train)
    pred = knn_clf.predict(X_test)

    nbr.append(n)
    error_rmse.append(mean_squared_error(y_test, pred, squared=False))
    error_rate.append(np.mean(y_test != pred))
    accuracy.append(accuracy_score(y_test, pred))
    knn_stats = pd.DataFrame({'neighbour': nbr, 'rmse': error_rmse, 'error_rate': error_rate, 'accuracy': accuracy})
sns.lineplot(x='neighbour', y='error_rate', data=knn_stats)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f8c4482be90>



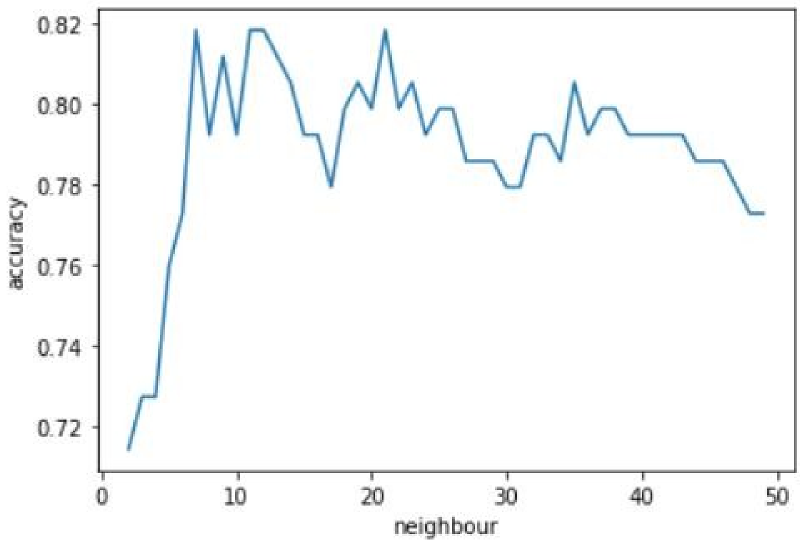
```
sns.lineplot(x='neighbour', y='rmse', data=knn_stats)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9fe2ea3110>
```




```
sns.lineplot(x='neighbour', y='accuracy', data=knn_stats)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9fe2e86e90>
```



```
knn_stats.neighbour[knn_stats.rmse.argmin()]
knn_stats.neighbour[[5,19,9,10]]
```

```
5      7
19     21
9      11
10     12
Name: neighbour, dtype: int64
```

```
knn_clf = KNeighborsClassifier(n_neighbors=7, weights='distance')
knn_clf.fit(X_train, y_train)
knn_pred = knn_clf.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.91      0.83        96
           1       0.78      0.55      0.65        58

    accuracy                           0.77       154
   macro avg       0.78      0.73      0.74       154
weighted avg       0.77      0.77      0.76       154
```

```
knn_model_vals = dict(accuracy=accuracy_score(y_test, knn_pred),
                auc=roc_auc_score(y_test, knn_pred),
                recall=recall_score(y_test, knn_pred),
                precision=precision_score(y_test, knn_pred),
                f1_score = f1_score(y_test, knn_pred),
                )



y_pred_prob_knn = knn_clf.predict_proba(X_test)[:, 1]
fpr_knn, tpr_knn , th_knn = roc_curve(y_test, y_pred_prob_knn)
gmean_knn = np.sqrt(tpr_knn * (1-fpr_knn))
ix_knn = np.argmax(gmean_knn)
```

Tuning KNN

We have seen that 4 values of Nearest Neighbours yeilded the same error in our earlier plot. We will tune the model with all those given values and pickup the best.

```
knn_param_grid = {'n_neighbors' : [7, 11, 12, 21],
                  'weights': ['distance', 'uniform'],
                  'algorithm' : ['ball_tree', 'kd_tree'],
                 'leaf_size' :[30,40,50],
              }
knn_rscv = RandomizedSearchCV(estimator=KNeighborsClassifier(), param_distributions=knn_param_grid,
 cv=3, scoring='f1_weighted')


knn_rscv.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=3, estimator=KNeighborsClassifier(),
                   param_distributions={'algorithm': ['ball_tree', 'kd_tree'],
                                        'leaf_size': [30, 40, 50],
                                        'n_neighbors': [7, 11, 12, 21],
                                        'weights': ['distance', 'uniform']},
                   scoring='f1_weighted')
```

```
knn_rscv.best_params_
```

```
{'weights': 'uniform',
 'n_neighbors': 11,
 'leaf_size': 50,
 'algorithm': 'kd_tree'}
```

```
tknn_pred = knn_rscv.predict(X_test)
y_pred_prob_knn_cv = knn_rscv.predict_proba(X_test)[:, 1]
fpr_tknn, tpr_tknn , th_tknn = roc_curve(y_test, y_pred_prob_knn_cv)
gmean_tknn = np.sqrt(tpr_tknn * (1-fpr_tknn))
ix_tknn = np.argmax(gmean_tknn)

accuracy_score(y_test, tknn_pred)
```

```
0.8311688311688312
```

```
tknn_model_vals = dict(accuracy=accuracy_score(y_test, tknn_pred),
                 auc=roc_auc_score(y_test, tknn_pred),
                 recall=recall_score(y_test, tknn_pred),
                 precision=precision_score(y_test, tknn_pred),
                 f1_score = f1_score(y_test, tknn_pred),
                 )
```

Model Comparison

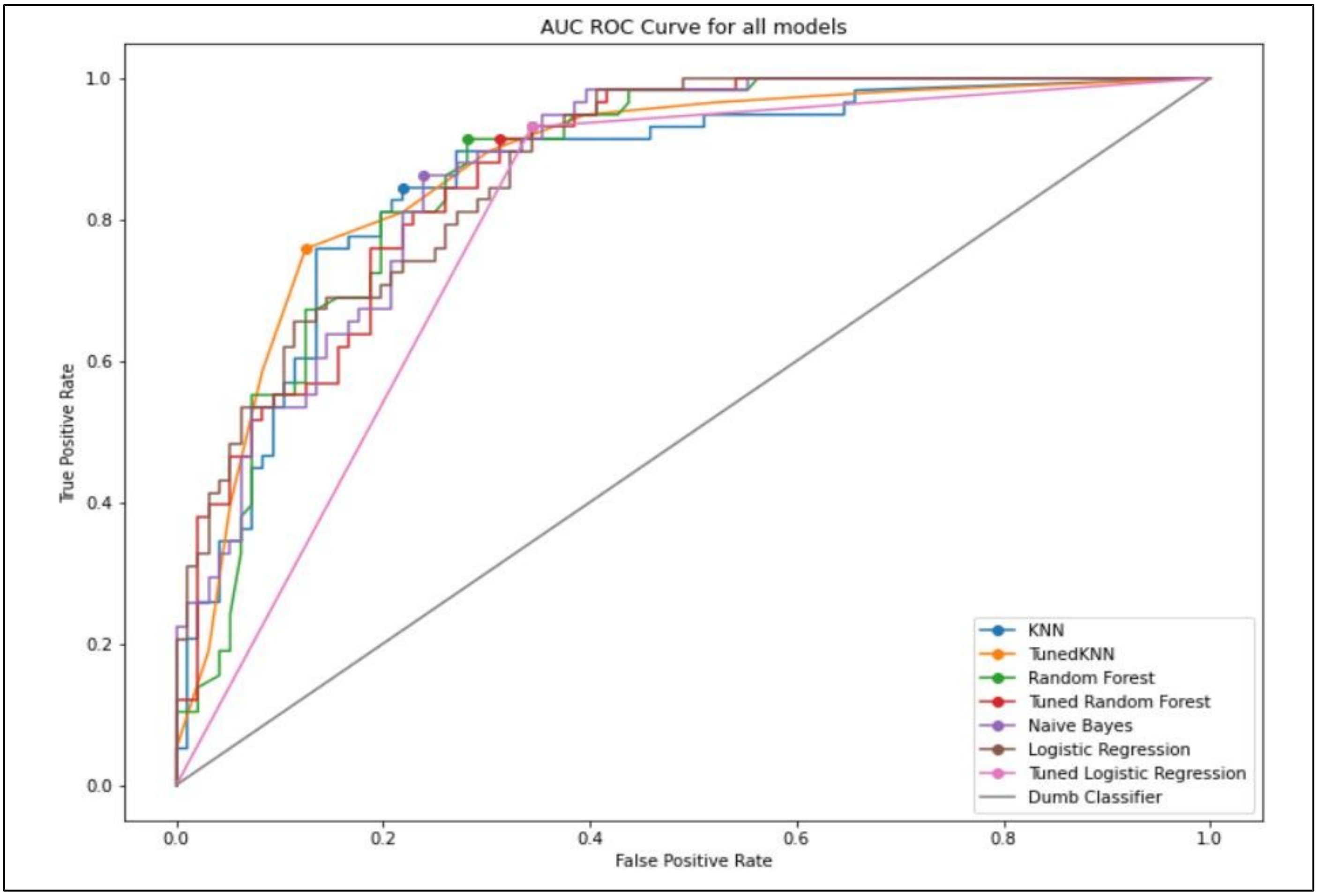
Data Modeling and Observations

I have plotted auc_roc curve of all the models together for comparison purpose. Also, I have plotted all the model stats together, tuned and non-tuned version. We can compare the performance by looking at the plots

.

AUC ROC Curve of all the models put together

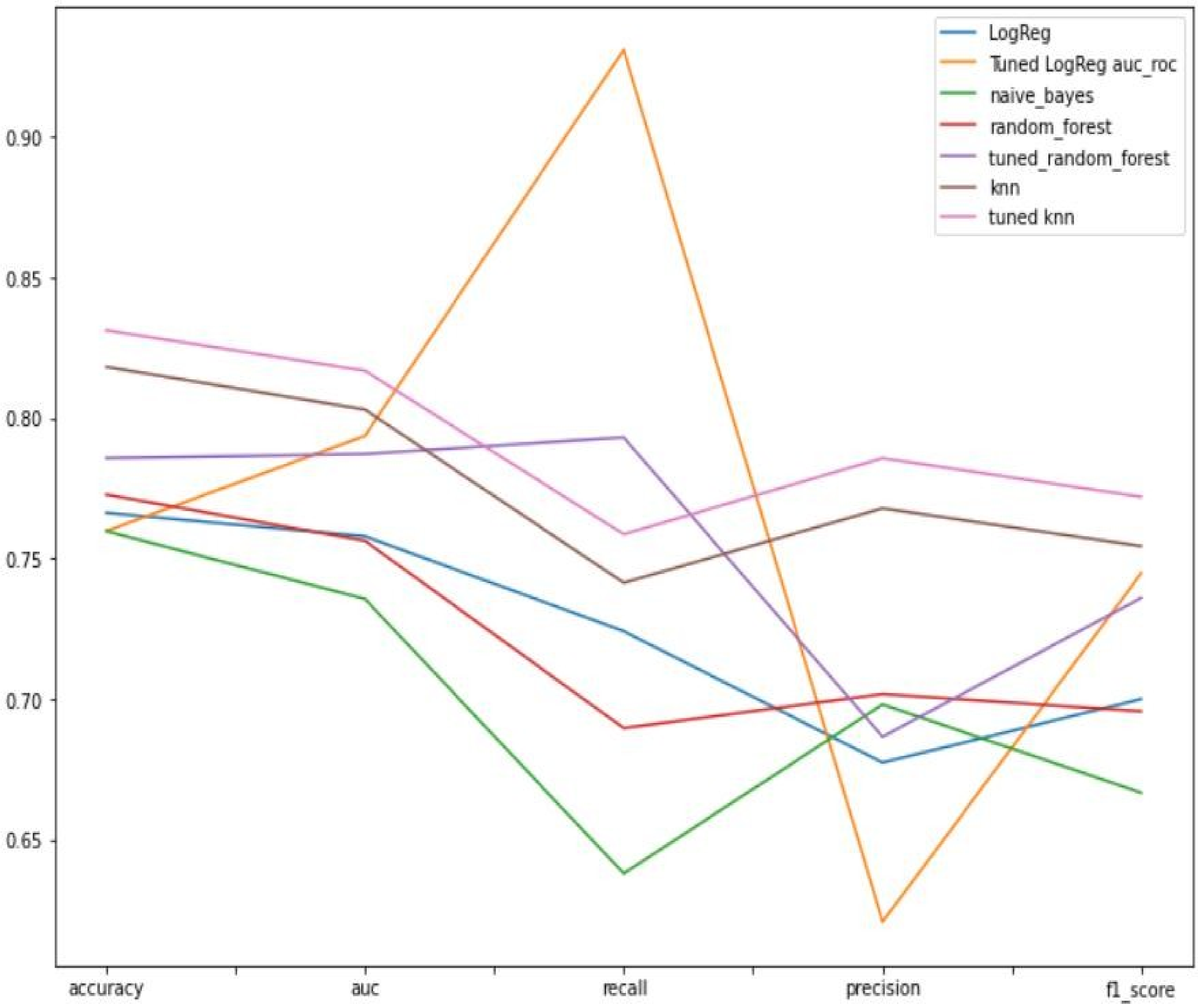
```
plt.subplots(figsize=(12,9))
plt.plot(fpr_knn, tpr_knn, marker='o', markevery=[ix_knn])
plt.plot(fpr_tknn, tpr_tknn, marker='o', markevery=[ix_tknn])
plt.plot(fpr_rf, tpr_rf, marker='o', markevery=[ix_rf])
plt.plot(fpr_trf, tpr_trf, marker='o', markevery=[ix_trf])
plt.plot(fpr_nb, tpr_nb, marker='o', markevery=[ix_nb])
plt.plot(fpr_lr, tpr_lr, marker='o', markevery=[ix_lr])
plt.plot(fpr_tlr, tpr_tlr, marker='o', markevery=[ix_tlr])
plt.plot([0,1], [0,1])plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC ROC Curve for all models')
plt.legend(['KNN', 'TunedKNN', 'Random Forest', 'Tuned Random Forest', 'Naive Bayes', 'Logistic Regressi
on',  'Tuned Logistic Regression','Dumb Classifier'])
plt.show()
```



Comparing model parameters

```
model_stats = pd.DataFrame(data=[lr_model_vals, tlr_model_vals, gnb_model_vals, rf_model_vals,
                                 tuned_rf_model_vals, knn_model_vals, tknn_model_vals ],
                           index=['LogReg', 'Tuned LogReg auc_roc', 'naive_bayes', 'random_forest',
                                  'tuned_random_forest ', 'knn', 'tuned knn'])


model_stats.T.plot(kind='line', figsize=(12,9))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c4468bdd0>
```



FINAL OBSERVATIONS :

I have plotted different parameters of all the models above. By a simple look, we know that recall and precision are in opposite direction for all the models. The overfitted Random Forest had similar charaterstics as KNN but once that was tuned, its recall has gone up and precision came down.

Considering overall parameter values, KNN is best predictor of all the models. I have tuned KNN as well. With tuning the model performance has increased on all the parameters.

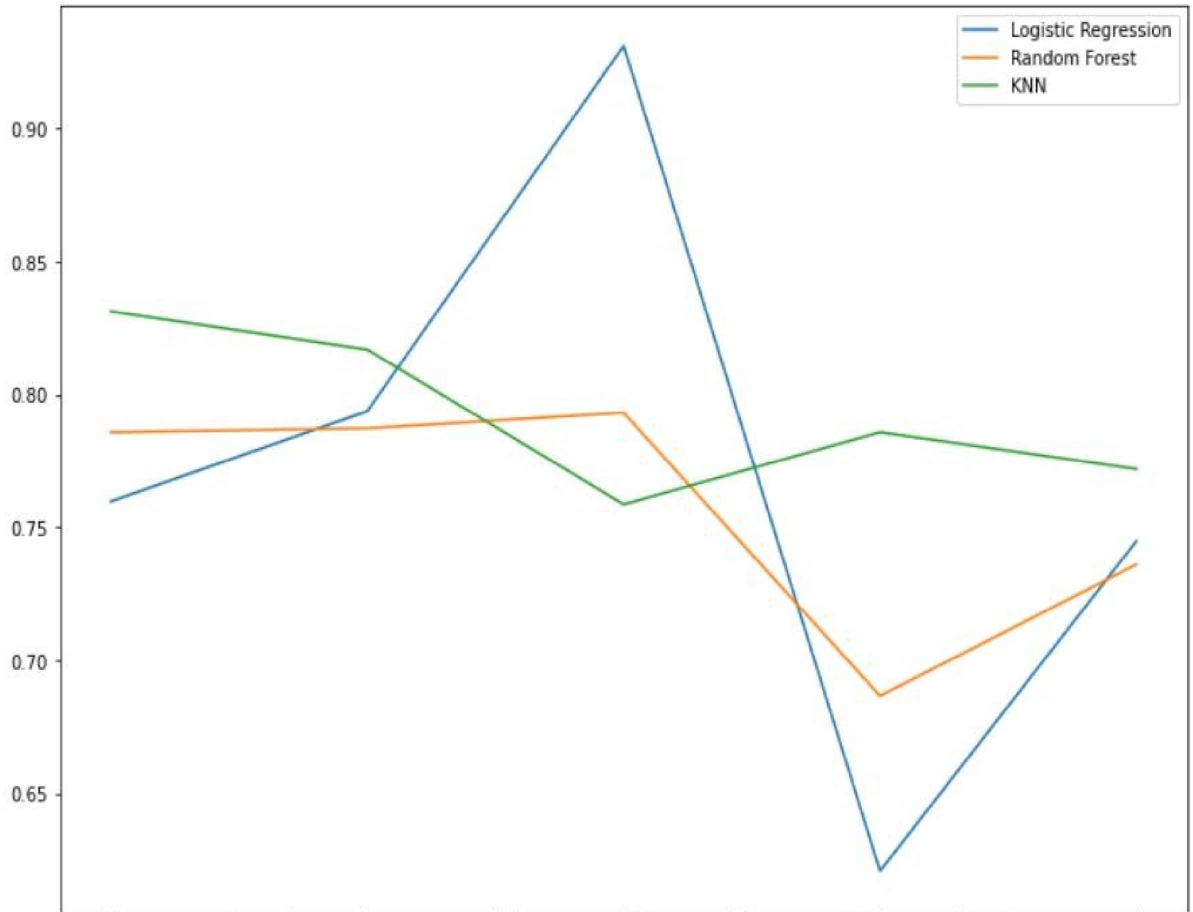
I have tuned Logistic regression as well. After tuning, TPR has gone up while precision has gone down.

If we are looking for a model with high Sensitivity, we can pick up Logistic Regression model. For over-all better performance, we can choose KNN

FINAL MODEL :

```
final_models = pd.DataFrame(data=[ tlr_model_vals, tuned_rf_model_vals,  tknn_model_vals ],
                      index=['Logistic Regression',  'Random Forest', 'KNN'])
final_models.T.plot(kind='line', figsize=(12,9), table=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c72132150>
```



	accuracy	auc	recall	precision	f1_score
Logistic Regression	0.7597402597402597	0.7936422413793103	0.9310344827586207	0.6206896551724138	0.7448275862068966
Random Forest	0.7857142857142857	0.787176724137931	0.7931034482758621	0.6865671641791045	0.736
KNN	0.8311688311688312	0.8168103448275862	0.7586206896551724	0.7857142857142857	0.7719298245614034