# Code for ML project

import numpy as np

import pandas as pd

df = pd.read\_csv('diabetes.csv')

df

df.shape

df.columns

df.dtypes

df.head()

df.info()

# Returns basic statistics on numeric columns

df.describe().T

# Returns true for a column having null values, else false

df.isnull().any()

df = df.rename(columns={'DiabetesPedigreeFunction':'DPF'})

df.head()

# Importing essential libraries for visualization

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

# Plotting the Outcomes based on the number of dataset entries

plt.figure(figsize=(10,7))

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

# Removing the unwanted spines

plt.gca().spines['top'].set\_visible(False)

plt.gca().spines['right'].set\_visible(False)

# Headings

plt.xlabel('Has Diabetes')

plt.ylabel('Count')

plt.show()

# Replacing the 0 values from ['Glucose','BloodPressure','SkinThickness','Insulin','BMI'] by NaN

df\_copy = df.copy(deep=True)

df\_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df\_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

df\_copy.isnull().sum()

# To fill these Nan values the data distribution needs to be understood

# Plotting histogram of dataset before replacing NaN values

p = df\_copy.hist(figsize = (15,15))

# Replacing NaN value by mean, median depending upon distribution

df\_copy['Glucose'].fillna(df\_copy['Glucose'].mean(), inplace=True)

df\_copy['BloodPressure'].fillna(df\_copy['BloodPressure'].mean(), inplace=True)

df\_copy['SkinThickness'].fillna(df\_copy['SkinThickness'].median(), inplace=True)

df\_copy['Insulin'].fillna(df\_copy['Insulin'].median(), inplace=True)

df\_copy['BMI'].fillna(df\_copy['BMI'].median(), inplace=True)

# Plotting histogram of dataset after replacing NaN values

p = df\_copy.hist(figsize=(15,15))

df\_copy.isnull().sum()

from sklearn.model\_selection import train\_test\_split

X = df.drop(columns='Outcome')

y = df['Outcome']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=0)

print('X\_train size: {}, X\_test size: {}'.format(X\_train.shape, X\_test.shape))

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Using GridSearchCV to find the best algorithm for this problem

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import ShuffleSplit

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

# Creating a function to calculate best model for this problem

def find\_best\_model(X, y):

models = {

'logistic\_regression': {

'model': LogisticRegression(solver='lbfgs', multi\_class='auto'),

'parameters': {

'C': [1,5,10]

}

},

'decision\_tree': {

'model': DecisionTreeClassifier(splitter='best'),

'parameters': {

'criterion': ['gini', 'entropy'],

'max\_depth': [5,10]

}

},

'random\_forest': {

'model': RandomForestClassifier(criterion='gini'),

'parameters': {

'n\_estimators': [10,15,20,50,100,200]

}

},

'svm': {

'model': SVC(gamma='auto'),

'parameters': {

'C': [1,10,20],

'kernel': ['rbf','linear']

}

}

}

scores = []

cv\_shuffle = ShuffleSplit(n\_splits=5, test\_size=0.20, random\_state=0)

for model\_name, model\_params in models.items():

gs = GridSearchCV(model\_params['model'], model\_params['parameters'], cv = cv\_shuffle, return\_train\_score=False)

gs.fit(X, y)

scores.append({

'model': model\_name,

'best\_parameters': gs.best\_params\_,

'score': gs.best\_score\_

})

return pd.DataFrame(scores, columns=['model','best\_parameters','score'])

find\_best\_model(X\_train, y\_train)

# Using cross\_val\_score for gaining average accuracy

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(RandomForestClassifier(n\_estimators=20, random\_state=0), X\_train, y\_train, cv=5)

print('Average Accuracy : {}%'.format(round(sum(scores)\*100/len(scores)), 3))

# Creating Random Forest Model

classifier = RandomForestClassifier(n\_estimators=20, random\_state=0)

classifier.fit(X\_train, y\_train)

# Creating a confusion matrix

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score

y\_pred = classifier.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

# Plotting the confusion matrix

plt.figure(figsize=(10,7))

p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')

plt.title('Confusion matrix for Random Forest Classifier Model - Test Set')

plt.xlabel('Predicted Values')

plt.ylabel('Actual Values')

plt.show()

# Accuracy Score

score = round(accuracy\_score(y\_test, y\_pred),4)\*100

print("Accuracy on test set: {}%".format(score))

# Classification Report

print(classification\_report(y\_test, y\_pred))S

# Creating a confusion matrix for training set

y\_train\_pred = classifier.predict(X\_train)

cm = confusion\_matrix(y\_train, y\_train\_pred)

cm

# Plotting the confusion matrix

plt.figure(figsize=(10,7))

p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')

plt.title('Confusion matrix for Random Forest Classifier Model - Train Set')

plt.xlabel('Predicted Values')

plt.ylabel('Actual Values')

plt.show()

# Accuracy Score

score = round(accuracy\_score(y\_train, y\_train\_pred),4)\*100

print("Accuracy on trainning set: {}%".format(score))

# Classification Report

print(classification\_report(y\_train, y\_train\_pred))

# Creating a function for prediction

def predict\_diabetes(Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age):

preg = int(Pregnancies)

glucose = float(Glucose)

bp = float(BloodPressure)

st = float(SkinThickness)

insulin = float(Insulin)

bmi = float(BMI)

dpf = float(DPF)

age = int(Age)

x = [[preg, glucose, bp, st, insulin, bmi, dpf, age]]

x = sc.transform(x)

return classifier.predict(x)

# Prediction 1

# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age

prediction = predict\_diabetes(2, 81, 72, 15, 76, 30.1, 0.547, 25)[0]

if prediction:

print('Oops! You have diabetes.')

else:

print("Great! You don't have diabetes.")

# Prediction 2

# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age

prediction = predict\_diabetes(1, 117, 88, 24, 145, 34.5, 0.403, 40)[0]

if prediction:

print('Oops! You have diabetes.')

else:

print("Great! You don't have diabetes.")

# Prediction 3

# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age

prediction = predict\_diabetes(5, 120, 92, 10, 81, 26.1, 0.551, 67)[0]

if prediction:

print('Oops! You have diabetes.')

else:

print("Great! You don't have diabetes.")