Python Code

```
import os
os.chdir(r"D:\HarshDeepProject12024")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec #subplots
from matplotlib.ticker import FormatStrFormatter

#Import models from scikit learn module:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
#For K-fold cross validation
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics
import statsmodels.api as sm
import scipy.stats as st
```

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

diabetes=pd.read_csv("diabetes.csv")
diabetes
diabetes["P_I"]=np.where(diabetes.Pregnancies>=7,1,0)
diabetes
tab=pd.crosstab(diabetes.Outcome, diabetes.P_I, margins=True)

## same code used for other variables for contingency table construction


# Code for stacked bar diagram construction for other variables
and also testing using chi square
tab
tab1=tab.iloc[:-1,:-1]
tab1
st.chi2_contingency(tab1)
Non_diabetic=[426,74]
Diabetic=[173,95]
l=["<7␣pregnancies",">7␣pregnancies"]
plt.bar(l,Non_diabetic,0.4,label="Non_diabetic")
plt.bar(l,Diabetic,0.4,bottom=Non_diabetic,label="Diabetic")

plt.legend()

#classification
#Logit model:-
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import matplotlib.pyplot as plt

#define the predictor variables and the response variable
X = diabetes.iloc[ :,: 8]
y = diabetes.iloc[:, 8]
#split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3)

#instantiate the model
log_regression = LogisticRegression()

#fit the model using the training data
log_regression.fit(X_train,y_train)
(log_regression.fit(X_train,y_train)).summary()


y_pred_proba = log_regression.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True␣Positive␣Rate')
plt.xlabel('False␣Positive␣Rate')
```

```python
plt.legend(loc=4)
plt.show()

y_pred=log_regression.predict(X_test)

#confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)



#Probit Model:-
probit_model=sm.Probit(y,X)
result=probit_model.fit()
print(result.summary())

from sklearn import metrics
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
probit=sm.Probit(y_train,X_train)
probit.fit()
print(probit.fit().summary())



result1 = X_test
result1['y_pred'] = result1['Pregnancies'] * 0.0837 + result1['Glucose'] *0.0128
result1



import scipy.stats as si
def normsdist(z):
```

```python
    z = si.norm.cdf(z,0.0,1.0)
    return (z)
normsdist(1.96)
result1['y_pred_Probit'] = normsdist(result1['y_pred'])
result1

d = {'y_pred_proba': result1['y_pred_Probit']}
df23 = pd.DataFrame(data=d)
df23 = df23.reset_index()
df23.drop(['index'], axis=1, inplace=True)
df23['y_pred'] = 0.000
for i in range(0,len(df23['y_pred_proba'])):
    if df23['y_pred_proba'][i] > 0.500:
        df23['y_pred'][i] = 1.000
    else:
        df23['y_pred'][i] = 0.000
y_pred = np.array(df23['y_pred'])
y_pred = y_pred.astype('int64')
y_pred

from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
y_pred_proba = np.array(df23['y_pred_proba'])
y_pred_proba

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
probit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

```
plt.figure()
plt.plot(fpr, tpr, label='Probit␣Model␣(area␣=␣%0.2f)' % probit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False␣Positive␣Rate')
plt.ylabel('True␣Positive␣Rate')
plt.title('Receiver␣operating␣characteristic')
plt.legend(loc="lower␣right")
plt.savefig('Probit_ROC')
plt.show()


# KNN Classification Model:-
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
rom sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy␣:␣", accuracy_score(y_test, y_pred))
cm
```

```
roc_auc=auc(fpr,tpr)

# Plot of a ROC curve for a specific class
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```