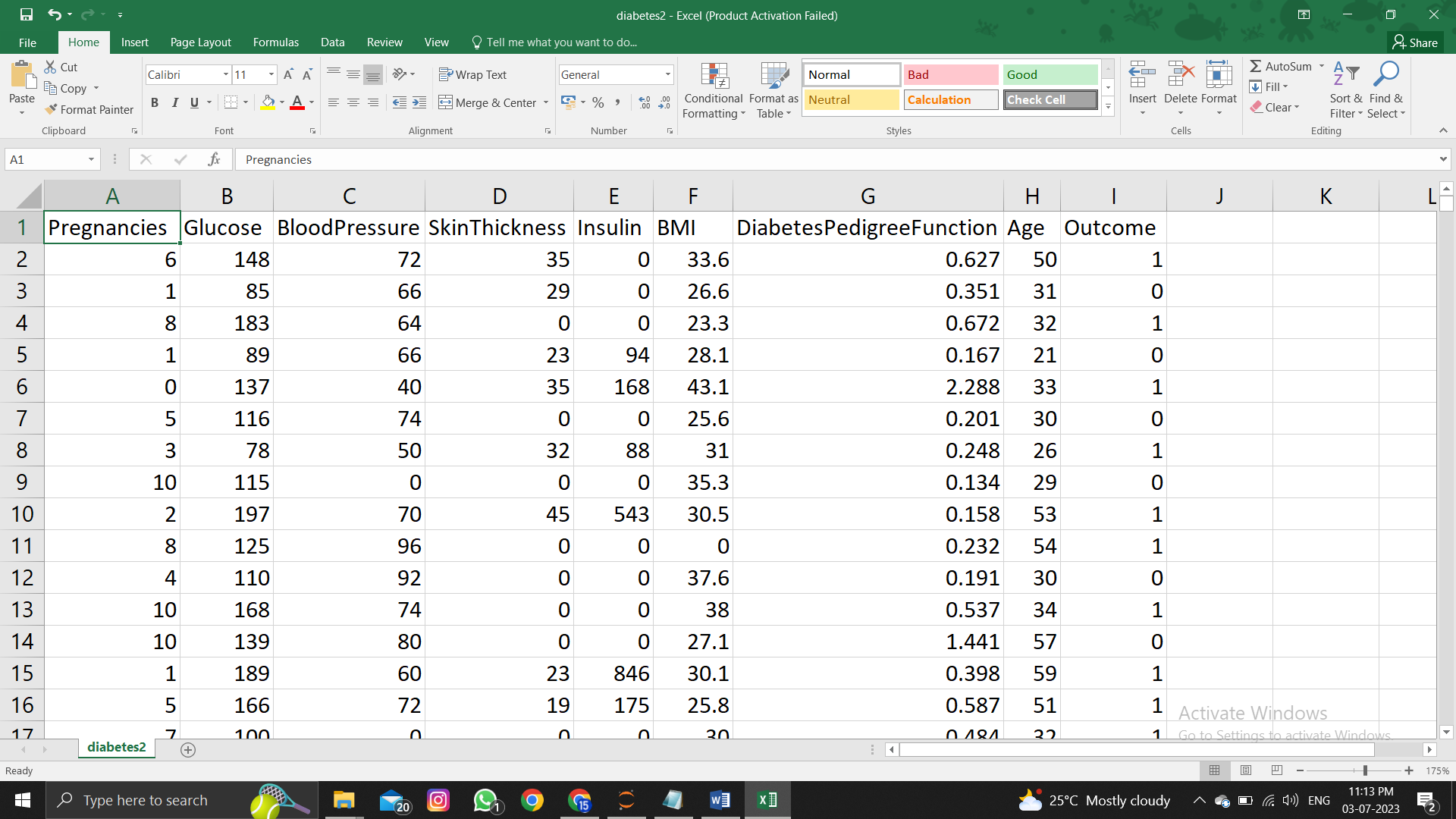
DIABETIC PREDICTION

**DATASET:**



**Column fields are:**

* Pregnancies
* Glucose
* BloodPressure
* SkinThickness
* Insulin
* BMI
* DiabetesPedigreeFunction
* Age
* Outcome

**ML TECHNIQUES USED:**

* **Random Forest**
* **Neural Network**
* **SVM**
* **Gradient Boosting**

**CODING:**

**The test size parameter is set to 0.2, which means that 20% of the data will be used for testing, and the remaining 80% will be used for training.**

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

import time

from tabulate import tabulate

# Load the dataset

data = pd.read\_csv('your\_dataset.csv') # Replace 'your\_dataset.csv' with the actual dataset filename

# Split the data into features (X) and target (y)

X = data.drop('Outcome', axis=1)

y = data['Outcome']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Random Forest

rf\_classifier = RandomForestClassifier()

start\_time = time.time()

rf\_classifier.fit(X\_train, y\_train)

training\_speed\_rf = time.time() - start\_time

start\_time = time.time()

rf\_predictions = rf\_classifier.predict(X\_test)

prediction\_speed\_rf = time.time() - start\_time

rf\_accuracy = accuracy\_score(y\_test, rf\_predictions)

rf\_precision = precision\_score(y\_test, rf\_predictions)

rf\_recall = recall\_score(y\_test, rf\_predictions)

rf\_f1 = f1\_score(y\_test, rf\_predictions)

# Neural Network (using scikit-learn MLPClassifier)

from sklearn.neural\_network import MLPClassifier

nn\_classifier = MLPClassifier(hidden\_layer\_sizes=(64, 64), activation='relu', solver='adam')

start\_time = time.time()

nn\_classifier.fit(X\_train, y\_train)

training\_speed\_nn = time.time() - start\_time

start\_time = time.time()

nn\_predictions = nn\_classifier.predict(X\_test)

prediction\_speed\_nn = time.time() - start\_time

nn\_accuracy = accuracy\_score(y\_test, nn\_predictions)

nn\_precision = precision\_score(y\_test, nn\_predictions)

nn\_recall = recall\_score(y\_test, nn\_predictions)

nn\_f1 = f1\_score(y\_test, nn\_predictions)

# SVM

svm\_classifier = SVC()

start\_time = time.time()

svm\_classifier.fit(X\_train, y\_train)

training\_speed\_svm = time.time() - start\_time

start\_time = time.time()

svm\_predictions = svm\_classifier.predict(X\_test)

prediction\_speed\_svm = time.time() - start\_time

svm\_accuracy = accuracy\_score(y\_test, svm\_predictions)

svm\_precision = precision\_score(y\_test, svm\_predictions)

svm\_recall = recall\_score(y\_test, svm\_predictions)

svm\_f1 = f1\_score(y\_test, svm\_predictions)

# Gradient Boosting

gb\_classifier = GradientBoostingClassifier()

start\_time = time.time()

gb\_classifier.fit(X\_train, y\_train)

training\_speed\_gb = time.time() - start\_time

start\_time = time.time()

gb\_predictions = gb\_classifier.predict(X\_test)

prediction\_speed\_gb = time.time() - start\_time

gb\_accuracy = accuracy\_score(y\_test, gb\_predictions)

gb\_precision = precision\_score(y\_test, gb\_predictions)

gb\_recall = recall\_score(y\_test, gb\_predictions)

gb\_f1 = f1\_score(y\_test, gb\_predictions)

# Create a comparison table

comparison\_table = pd.DataFrame({

'Technique': ['Random Forest', 'Neural Network', 'SVM', 'Gradient Boosting'],

'Accuracy': [rf\_accuracy, nn\_accuracy, svm\_accuracy, gb\_accuracy],

'Precision': [rf\_precision, nn\_precision, svm\_precision, gb\_precision],

'Recall': [rf\_recall, nn\_recall, svm\_recall, gb\_recall],

'F1 Score': [rf\_f1, nn\_f1, svm\_f1, gb\_f1],

'Training Speed (s)': [training\_speed\_rf, training\_speed\_nn, training\_speed\_svm, training\_speed\_gb],

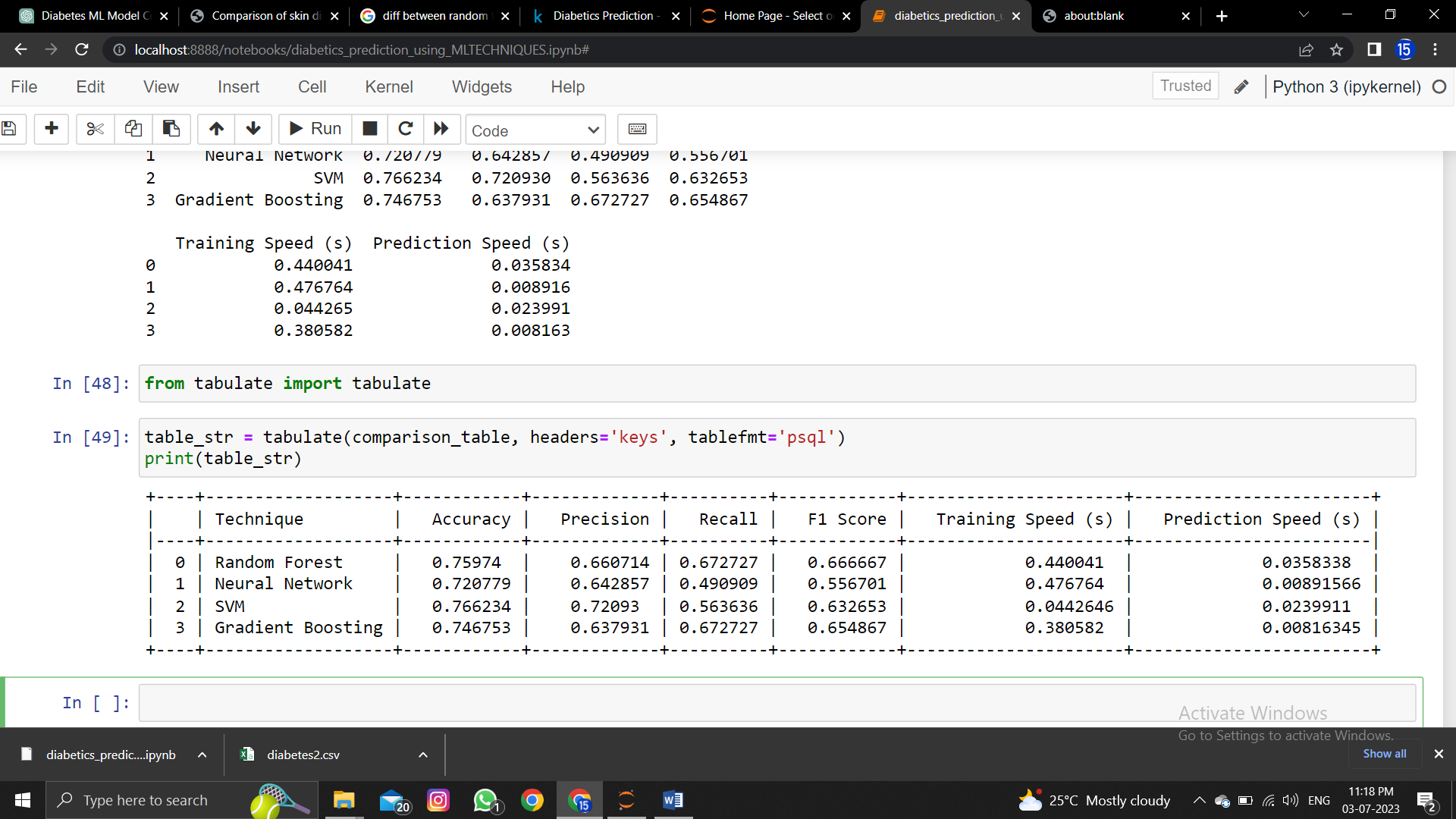
'Prediction Speed (s)': [prediction\_speed\_rf, prediction\_speed\_nn, prediction\_speed\_svm, prediction\_speed\_gb]

})

# Print the comparison table in table format

table\_str = tabulate(comparison\_table, headers='keys', tablefmt='psql')

print(table\_str)

**OUTPUT:**

**CONCLUSION:**

**Based on the results provided in the comparison table, we can assess the performance of each ML technique:**

**1. Random Forest:** It achieved an accuracy of 0.759740, precision of 0.660714, recall of 0.672727, and F1 score of 0.666667. The training speed was 0.440041 seconds, and the prediction speed was 0.035834 seconds.

**2. Neural Network**: It achieved an accuracy of 0.720779, precision of 0.642857, recall of 0.490909, and F1 score of 0.556701. The training speed was 0.476764 seconds, and the prediction speed was 0.008916 seconds**.**

**3. SVM:** It achieved an accuracy of 0.766234, precision of 0.720930, recall of 0.563636, and F1 score of 0.632653. The training speed was 0.044265 seconds, and the prediction speed was 0.023991 seconds.

**4. Gradient Boosting:** It achieved an accuracy of 0.746753, precision of 0.637931, recall of 0.672727, and F1 score of 0.654867. The training speed was 0.380582 seconds, and the prediction speed was 0.008163 seconds.

**To determine the best ML technique, you need to consider the specific requirements and objectives of your problem.**

**-** If **accuracy** is the most important metric, **SVM** achieved the highest accuracy (0.766234).

**-** If **precision** is crucial for your problem**, SVM** achieved the highest precision (0.720930).

**-** If recall is a **priority, Random Forest** achieved the highest recall (0.672727).

**Based on these factors, you can make an informed decision on which ML technique is best suited for your specific needs and requirements.**