imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import glob  # Import glob to list CSV files
import pandas as pd
from tabulate import tabulate
from scipy.stats import norm

from google.colab import drive

drive.mount('/content/drive')
```

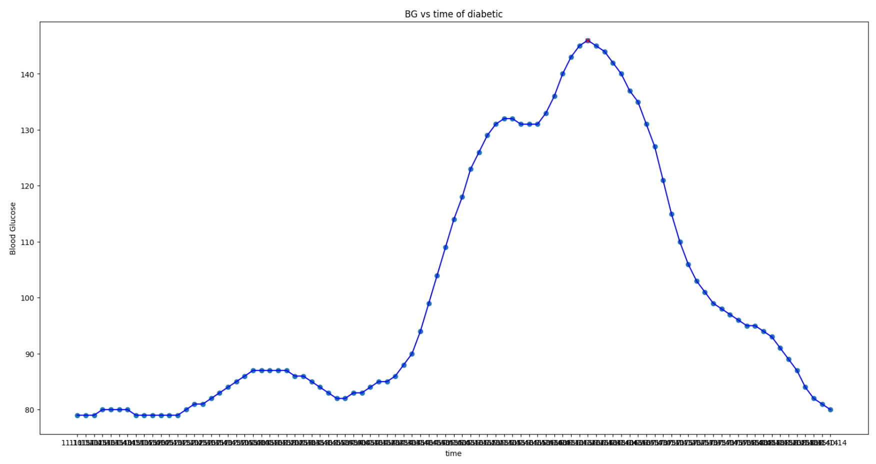
```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

import the datas

```
folder_path = 'data_for_diabetes/' # set path to 'data_for_diabetes/'
diabetic = glob.glob(folder_path + 'case *.csv') # make diabetic = case 1 - case 208

for file in diabetic:

    df = pd.read_csv(file) #make df = all the files that were selected in diabetic

nondiabetic = pd.read_csv('data for no diabetes.csv') #make nondiabetic = 'data for no diabetes.csv'
```

Blood glucose level of person with Diabetic vs time

```
plt.figure(figsize=(20, 10))
x = df.iloc[134:225 , 1] #choose data from row 134 to 225 in column 1
y = df.iloc[134:225 , 2]#choose data from row 134 to 225 in column 2

plt.scatter(x, y)

plt.title('BG vs time of diabetic')
plt.xlabel('time')
plt.ylabel('Blood Glucose')

plt.plot(x, y, color='blue')

max_y = y.max()  #finding y max
max_x = x[y.idxmax()]  #finding x max according to y max

# Highlighting the highest point
plt.plot(max_x, max_y, "x", color='red')

plt.show()
```



person with diabetic analysis

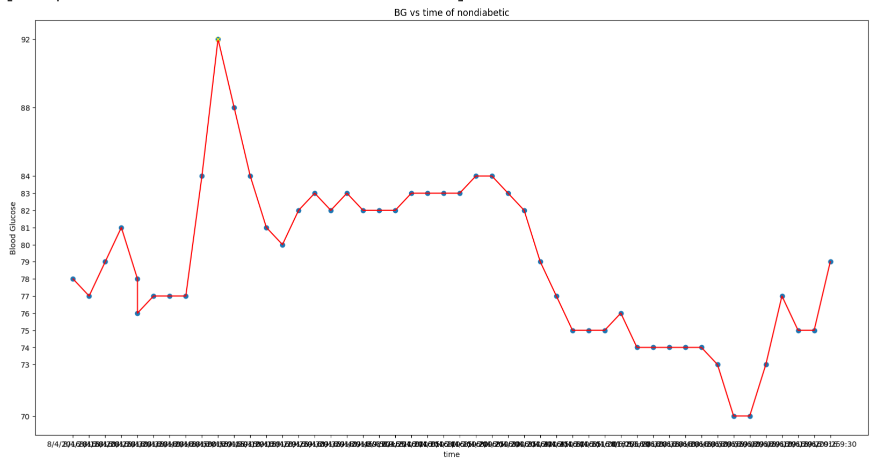
```
df_diabetic = df.iloc[:, 2]  #choose data from all row in column 2

mean_diabetic = df_diabetic.mean() #find mean
median_diabetic = df_diabetic.median()#find median
range_diabetic = df_diabetic.max() - df_diabetic.min()#find mode
std_diabetic = df_diabetic.std()#find sd


stats_df = pd.DataFrame({
    'Statistic Diabetic': ['Mean', 'Median', 'Range', 'Standard Deviation'],
    'Value': [mean_diabetic, median_diabetic, range_diabetic, std_diabetic]
}) #put datas in the table

print(tabulate(stats_df, headers='keys', tablefmt='fancy_grid'))
```

```
╒════╤══════════════════════╤═════════╕
│    │ Statistic Diabetic   │   Value │
╞════╪══════════════════════╪═════════╡
│  0 │ Mean                 │ 98.6332 │
├────┼──────────────────────┼─────────┤
│  1 │ Median               │ 94      │
├────┼──────────────────────┼─────────┤
│  2 │ Range                │ 86      │
├────┼──────────────────────┼─────────┤
│  3 │ Standard Deviation   │ 19.712  │
╘════╧══════════════════════╧═════════╛
```

non-diabetic vs time graph

```
plt.figure(figsize=(20, 10))
x = nondiabetic.iloc[:, 2]#choose data from all row in column 2
y = data = pd.to_numeric(nondiabetic.iloc[:, 3], errors='coerce') #from stackoverflow, this ignores the non-numeric number
plt.scatter(x, y)

plt.title('BG vs time of nondiabetic')
plt.xlabel('time')
plt.ylabel('Blood Glucose')



sorted_y = np.sort(y.unique())
plt.yticks(sorted_y)

plt.plot(x, y, color='red')

max_y = y.max()  #find y max
max_x = x[y.idxmax()]  #finding x max according to y max


plt.plot(max_x, max_y, "x", color='yellow')#mark the highest peak
```

```
[<matplotlib.lines.Line2D at 0x7d0cdc67ce80>]
```



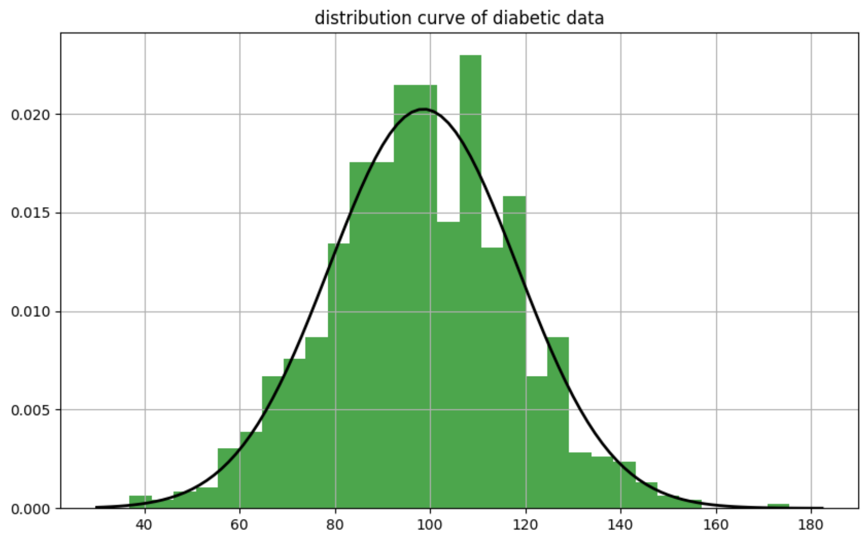
non-diabetic analysis

```
df_nondiabetic = pd.to_numeric(nondiabetic.iloc[:, 3], errors='coerce') #from stackoverflow, this ignores the non-numeric number
#1:50
mean_nondiabetic = df_nondiabetic.mean()#find mean
median_nondiabetic = df_nondiabetic.median()#find median
range_nondiabetic = df_nondiabetic.max() - df_nondiabetic.min()#find range
std_nondiabetic = df_nondiabetic.std()#find


stats_df = pd.DataFrame({
    'Statistic nondiabetic': ['Mean', 'Median', 'Range', 'Standard Deviation'],
    'Value': [mean_nondiabetic, median_nondiabetic, range_nondiabetic, std_nondiabetic]
})
print(tabulate(stats_df, headers='keys', tablefmt='fancy_grid'))#putting data in the table
```

```
╒════╤═════════════════════════╤═════════╕
│    │ Statistic nondiabetic   │   Value │
╞════╪═════════════════════════╪═════════╡
│  0 │ Mean                    │ 82.9286 │
├────┼─────────────────────────┼─────────┤
│  1 │ Median                  │ 82      │
├────┼─────────────────────────┼─────────┤
│  2 │ Range                   │ 43      │
├────┼─────────────────────────┼─────────┤
│  3 │ Standard Deviation      │  8.53051 │
╘════╧═════════════════════════╧═════════╛
```

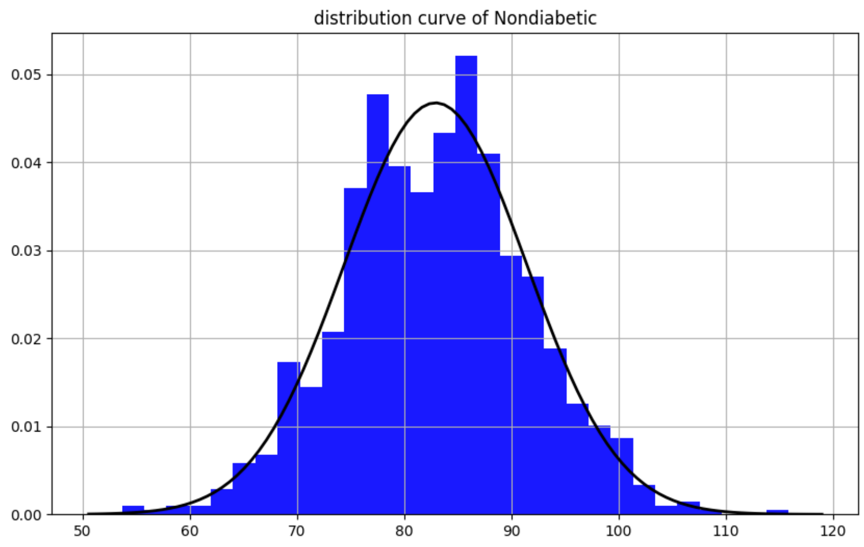
distribution graph of diabetic data

```
data = np.random.normal(mean_diabetic , std_diabetic, 1000)#randomly generate 1000 values with the same mean and std as diabetic data


plt.figure(figsize=(10, 6))
plt.hist(data, bins=30, density=True, alpha=0.7, color='g')#create histogram of the random data
xmin, xmax = plt.xlim() #finding the domain of x
x = np.linspace(xmin, xmax, 100)#This creates an array of 100 points along the x-axis
pdf_values = norm.pdf(x, mean_diabetic , std_diabetic) #sd plot

plt.plot(x, pdf_values, 'k', linewidth=2) #plot normal dist. curve on top of where the diabetic distribution curve is


plt.title("distribution curve of diabetic data" )
plt.grid(True)
plt.show()
```



distribution for nondiabetic

```
data = np.random.normal(mean_nondiabetic , std_nondiabetic , 1000) #randomly generate 1000 values with the same mean and std as non-diabe


plt.figure(figsize=(10, 6))

plt.hist(data, bins=30, density=True, alpha=0.9, color='b')#create histogram of the random data
xmin, xmax = plt.xlim()#finding the domain of x
x = np.linspace(xmin, xmax, 100)#This creates an array of 100 points along the x-axis

pdf_values = norm.pdf(x, mean_nondiabetic , std_nondiabetic )

plt.plot(x, pdf_values, 'k', linewidth=2) #the normal dist. curve

plt.title("distribution curve of Nondiabetic")

plt.grid(True)
plt.show()
```



finding confidence interval

```
import scipy.stats as stats

x1_bar = mean_diabetic
x2_bar = mean_nondiabetic
n1 = 208
n2 = 57

# Calculate the standard error (SE)
SE = ((std_diabetic**2 / n1) + (std_nondiabetic**2 / n2)) ** 0.5
SE
# Degrees of freedom
df = n2 - 1

# Critical value from the t-distribution for a 95% confidence interval
t_critical = stats.t.ppf(0.95, df)  # Two-tailed test using scipy.stat

# Margin of error (ME)
ME = t_critical * SE

# Confidence interval (CI)
CI_lower = (x1_bar - x2_bar) - ME
CI_upper = (x1_bar - x2_bar) + ME


confidence_interval = pd.DataFrame({
    'confidence interval' : ['lower bound', 'upper bound'],
    'Value': [CI_lower, CI_upper]
})
print(tabulate(confidence_interval, headers='keys', tablefmt='fancy_grid'))#plot the data in table
```

```
╒════╤═══════════════════════╤═════════╕
│    │ confidence interval   │   Value │
╞════╪═══════════════════════╪═════════╡
│  0 │ lower bound           │ 12.7386 │
├────┼───────────────────────┼─────────┤
│  1 │ upper bound           │ 18.6705 │
╘════╧═══════════════════════╧═════════╛
```

calculating P value

Double-click (or enter) to edit

```
import scipy.stats as stats
#find t-value
T_value = (x1_bar - x2_bar)/SE
T_value

#find p-value
p_value = 2 * (1 - stats.t.cdf(abs(T_value), df))


p = pd.DataFrame({
    'p value of two tailed test' : ['P value', 'conclusion:'],
    'Value': [p_value, 'p value < 0.05 thus rejecting the null hypothesis, the datas are significantly different']
})
```

```
print(tabulate(p, headers='keys', tablefmt='fancy_grid'))#plot data in table
```

	p value of two tailed test	Value
0	P value	3.08575387464316e-12
1	conclusion:	p value < 0.05 thus rejecting the null hypothesis, the datas are significantly different

cohen's d test

```
#finding SD pool
SDp = np.sqrt(((n1 - 1) * std_diabetic**2 + (n2 - 1) * std_nondiabetic**2) / (n1 + n2 - 2))

#doing cohen's d test
d = (x1_bar - x2_bar) / SDp

p = pd.DataFrame({
    'value of cohens d' : ['d', 'conclusion'],
    'Value': [d, 'the value of d is 1, indicating that the difference between the groups are 1 standard deviation away, meaning the datas
})
print(tabulate(p, headers='keys', tablefmt='fancy_grid'))#plot data in table
```

	value of cohens d	Value
0	d	0.8761034499057128
1	conclusion	the value of d is 0.876, indicating that the difference between the groups are around 0.9 standard devi