Author: Jimin Kim (jk55@uw.edu)

Version 1.5.0

Lab 4 Report

Group Members: Kiyoshi Oshiro, Khoi Phan

Group Name for Leaderboard:

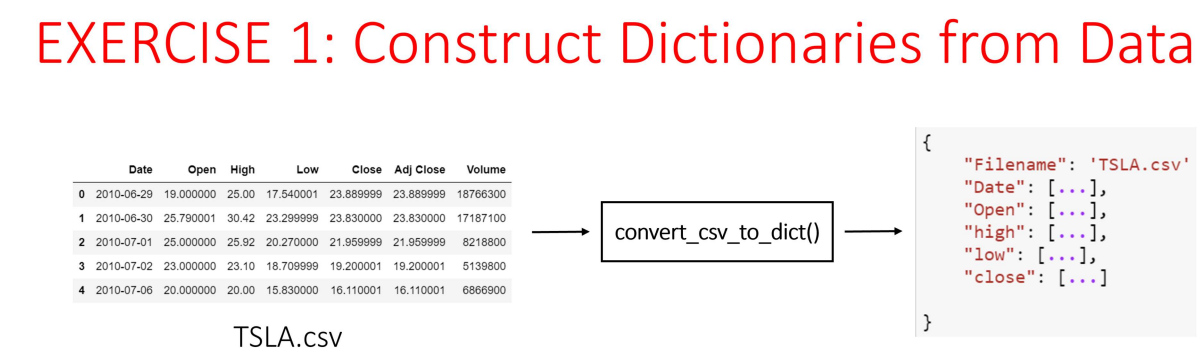
In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as st
%matplotlib inline
```

In [ ]:
```
from IPython.display import Image as ipyimage #For displaying images in colab jupyter
```

Exercise 1: Construct Dictionaries from Data

In [ ]:
```
ipyimage('lab4_exercise1.PNG', width = 1000)
```

Out[ ]:



In [ ]:
```
## method that converts a csv file to dictionary
## takes in file path and returns a dictionary with column names as keys and a list of
```

```
def convert_csv_to_dict(file_path):

  file = pd.read_csv(file_path)
  dictionary = {'Filename':file_path}
  # list of column names
  columns = list(file.columns)
  # convert to numpy array
  file_np = file.to_numpy()

  # creating the list of values the adding them to the dictionary
  for j in range(len(columns)):
    value_array = np.array([])
    for i in range(len(file_np)):
      value_array = np.append(value_array, file_np[i][j])
    dictionary[columns[j]] = value_array

  return dictionary
```

In [ ]:
```
TSLA_dict = convert_csv_to_dict('TSLA.csv')
```

In [ ]:
```
# Navigate to keys corresponding to 2nd and 4th columns (Open and Low prices) of TSLA.
# Print first 10 elements of each key.
TSLA_column2 = "Open"
TSLA_column4 = "Low"
print(TSLA_dict[TSLA_column2][:10])
print(TSLA_dict[TSLA_column4][:10])
```

```
dict_keys(['Filename', 'Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volum
e'])
[19.       25.790001 25.       23.       20.       16.4      16.139999
 17.58     17.950001 17.389999]
[17.540001 23.299999 20.27     18.709999 15.83     14.98     15.57
 16.549999 17.       16.9     ]
```

In [ ]:
```
diabetes_dict = convert_csv_to_dict('diabetes.csv')
```

In [ ]:
```
# Navigate to keys corresponding to 2nd and 4th columns (Glucose and SkinThickness) of
# Print first 10 elements of each key.

diabetes_column2 = "Glucose"
diabetes_column4 = "SkinThickness"
print(diabetes_dict[diabetes_column2][:10])
print(diabetes_dict[diabetes_column4][:10])
```

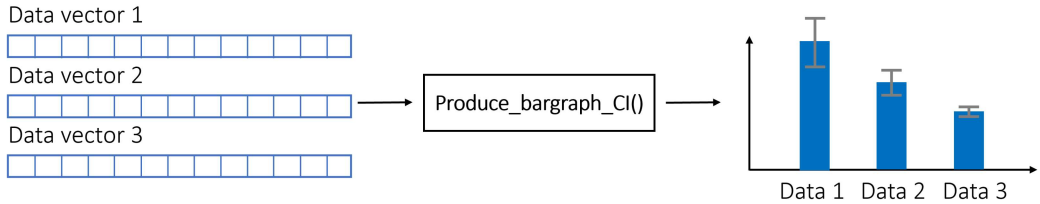
```
[148.  85. 183.  89. 137. 116.  78. 115. 197. 125.]
[35. 29.  0. 23. 35.  0. 32.  0. 45.  0.]
```

Exercise 2: Bar graph with confidence intervals

In [ ]:
```
ipyimage('lab4_exercise2.PNG', width = 1000)
```

Out[ ]:

EXERCISE 2: Bar graph with confidence intervals



- Create a function **produce_bargraph_CI()** which takes list of three 1D arrays and confidence level as inputs and output a bar graph with confidence intervals.
- The function should accept following parameters
  - data_vec_list = list of three 1D arrays each corresponding to a series of data
  - conf_level = Confidence level to be used for confidence interval – Takes one of three values - 0.9, 0.95, 0.99.
  - bar_labels = list of strings corresponding to labels for each bar
- The function should output a bar graph with 3-bars. Each bar should include a confidence interval corresponding to specified confidence level.
- Test your function against Glucose, Blood pressure and BMI columns of non-diabetics and diabetics with specified confidence intervals.
- Make sure you properly format your plot so that bars and error bars are visible. Add appropriate title and bar labels.

50

In [ ]:

```
# Load diabetes.csv
# Split the data into diabetic and non-diabetic
# Read the diabetes dataset from CSV
diabetes = pd.read_csv('/content/diabetes.csv')
wDiabetes = diabetes.query("Outcome==1")
woDiabetes = diabetes.query("Outcome==0")
withDiabetes = wDiabetes.to_numpy()
withoutDiabetes = woDiabetes.to_numpy()
```

In [ ]:

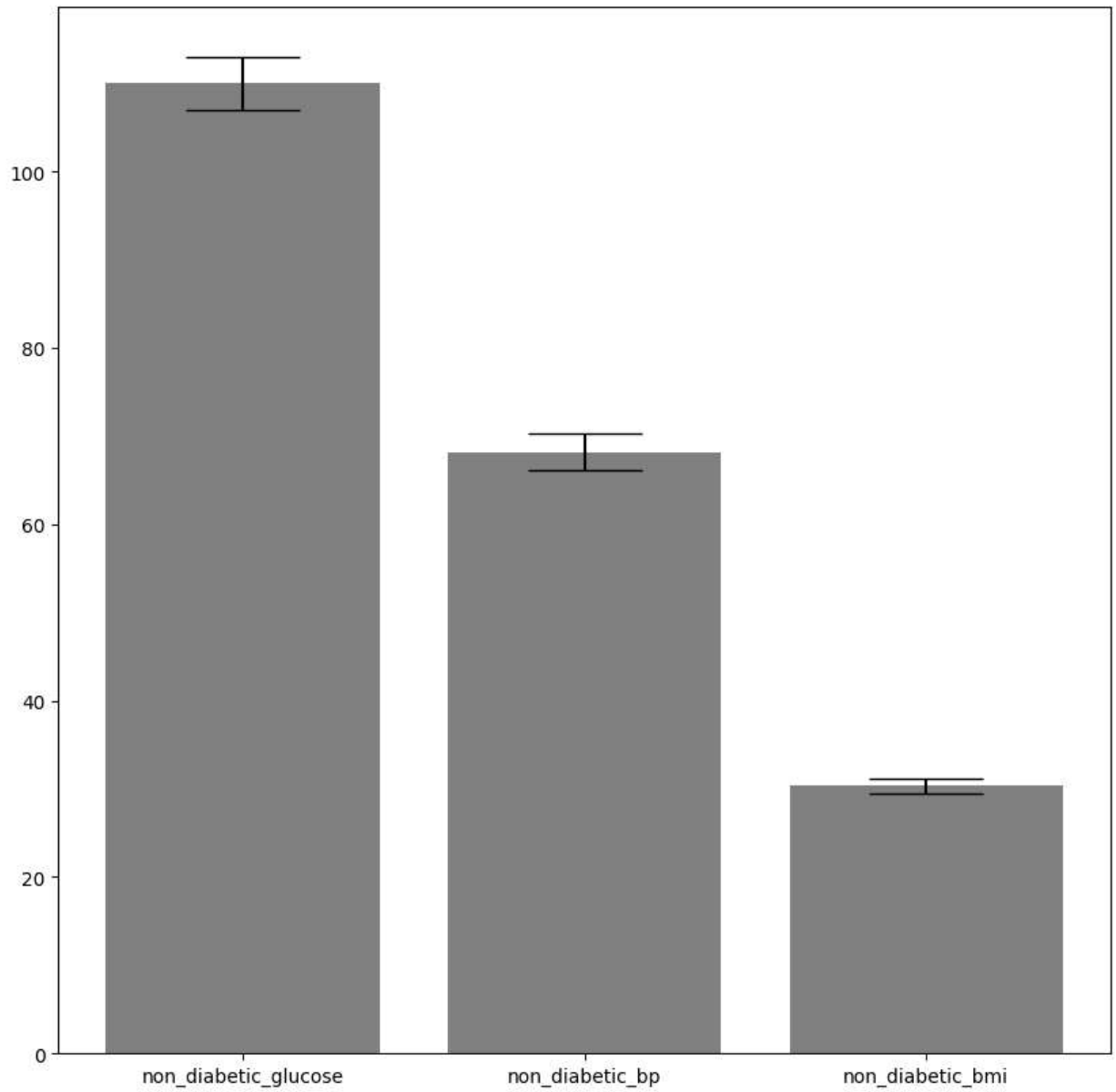
```
# Extract glucose, blood pressure, and BMI metrics from diabetic and non-diabetic
# non-diabetic metrics
non_diabetic_glucose = withoutDiabetes[:,1]
non_diabetic_bp = withoutDiabetes[:,2]
non_diabetic_bmi = withoutDiabetes[:,5]

# diabetic metrics
diabetic_glucose = withDiabetes[:,1]
diabetic_bp = withDiabetes[:,2]
diabetic_bmi = withDiabetes[:,5]
non_diabetic_list = [non_diabetic_glucose, non_diabetic_bp, non_diabetic_bmi]
diabetic_list = [diabetic_glucose, diabetic_bp, diabetic_bmi]
non_diabetic_bar_labels = ['non_diabetic_glucose', 'non_diabetic_bp', 'non_diabetic_bm
diabetic_bar_labels = ['diabetic_glucose', 'diabetic_bp', 'diabetic_bmi']
```

In [32]:

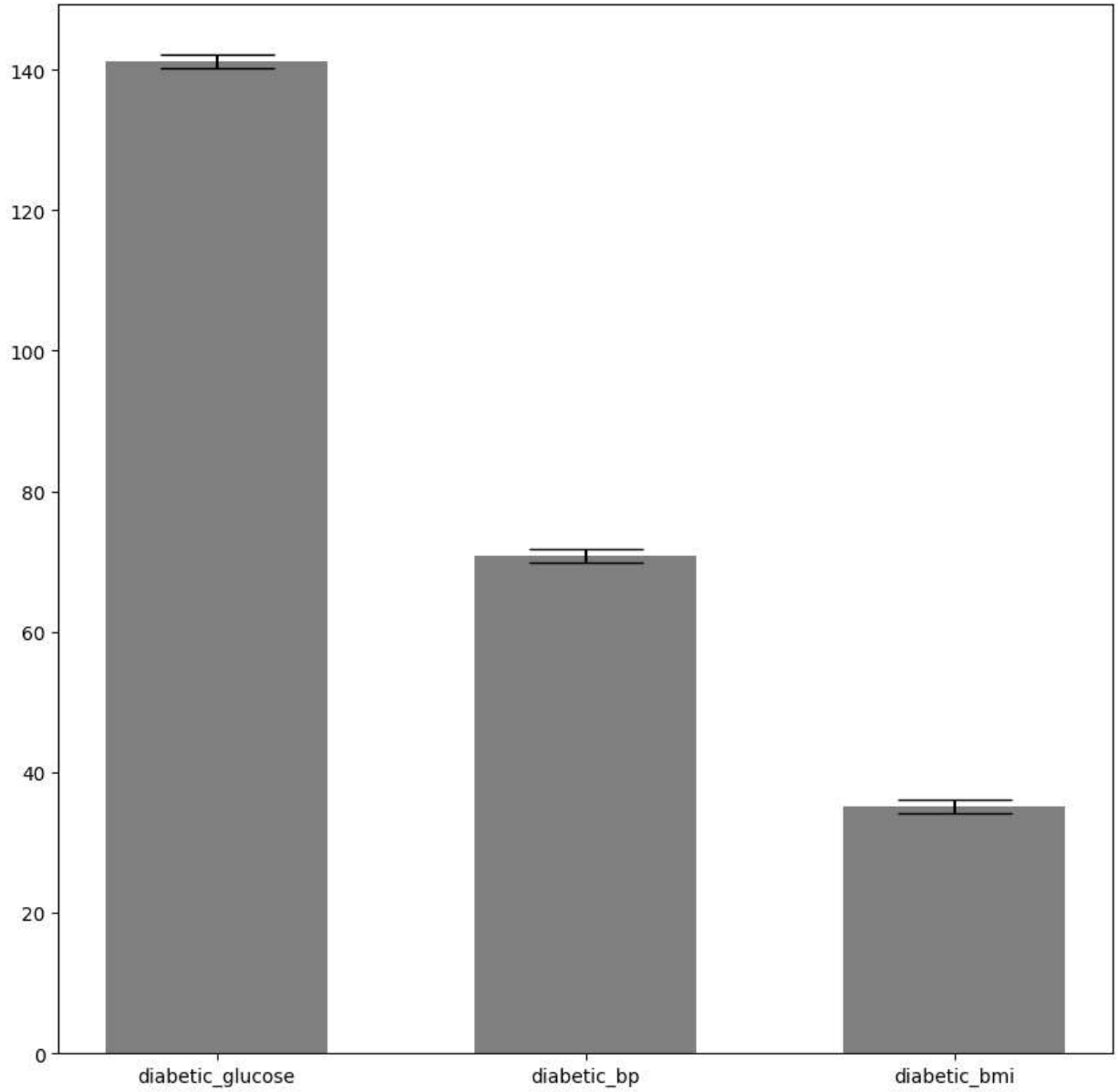
```
def produce_bargraph_CI(data_vec_list, conf_level, bar_labels):
    # Calculate the mean and standard error of each array in data_vec_list
    data_mean = np.mean(data_vec_list, axis=1)
    sample_size = len(data_vec_list[0])  # Assuming all arrays in data_vec_list have t

    # Calculate the standard error
    std_error = st.sem(data_vec_list, axis=1)

    # Calculate the confidence interval
    CI_99_lower, CI_99_upper = st.t.interval(conf_level, df=sample_size - 1, loc=data_

    conf_interval = CI_99_upper - data_mean
```

```
    # Set x-axis labels
    x_labels = bar_labels

    # Create a figure with a specified size
    fig = plt.figure(figsize=(10, 10))

    # Plot the bar graph with error bars representing confidence intervals
    plt.bar(x_labels, data_mean, color='grey', yerr=conf_interval, capsize=30)

    # Display the plot
    plt.show()
```

In [33]:
```
produce_bargraph_CI(data_vec_list = non_diabetic_list, conf_level = 0.99, bar_labels =
```



In [ ]:
```
produce_bargraph_CI(data_vec_list = diabetic_list, conf_level = 0.95, bar_labels = dia
```



In [ ]:

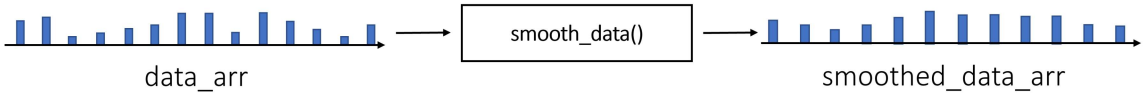
In [ ]:

Exercise 3: Rolling Mean/Median Function from Scratch

In [ ]:
```
ipyimage('lab4_exercise3.PNG', width = 1000)
```

Out[ ]:

EXERCISE 3: Rolling Mean/Median Function from Scratch



- Using illustrations from slides 39 – 42, create a function **smooth_data() from scratch** which takes a 1D array as an input and output a new 1D array with rolling mean or median applied.
- The function should accept following parameters
  - data_arr – A 1D array corresponding to a series of data
  - smooth_type – Type of smoothing method. Takes either 'mean' or 'median'.
  - window_size – Window size for the smoothing operation.
- Test your function against provided dataset in lab template. For each smoothed data, plot on top of the original data for comparison. Use **dotted line** for smoothed data and **solid line** for original data.
- NOTE: DO NOT USE PRE-BUILT SMOOTHING FUNCTIONS

51

In [ ]:

```
# Load stock datasets

tesla = pd.read_csv('/content/TSLA.csv')
tesla_np = tesla.to_numpy()

google = pd.read_csv('/content/GOOGL.csv')
google_np = google.to_numpy()

dji = pd.read_csv('/content/DJI.csv')
dji_np = dji.to_numpy()
```

In [ ]:

```
# Extract closing price for each stock data

tesla_np_closing = tesla_np[:, 4]
google_np_closing = google_np[:, 4]
DJI_np_closing = dji_np[:, 4]
```

In [ ]:

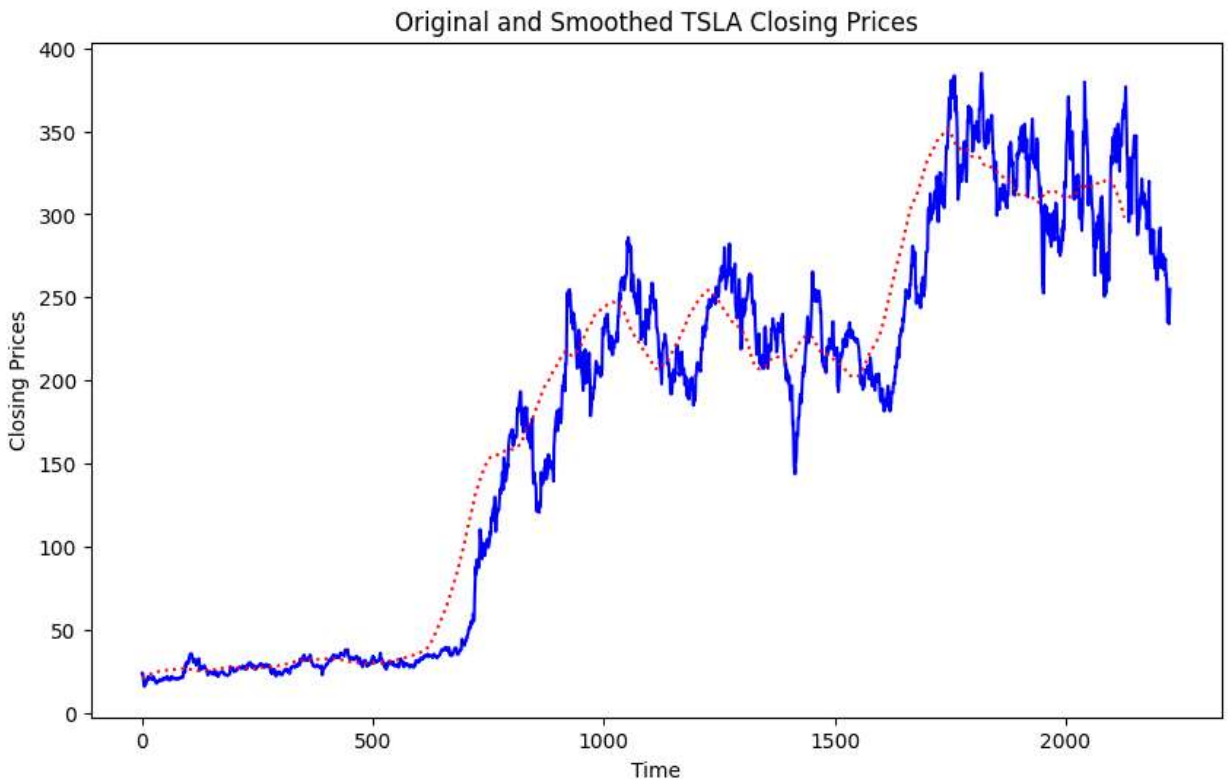
```
def smooth_data(data_arr, smooth_type, window_size):
    smoothed_data_arr = np.zeros(len(data_arr) - window_size + 1) # array size when sm
    for x in range(len(data_arr) - window_size + 1): #iterate through the array
        if smooth_type == 'mean':
            mean = 0
            for y in range(window_size):
                mean += data_arr[x + y] / window_size # add to the mean window_size ti
            smoothed_data_arr[x] = mean # add it to our smoothed data array
        elif smooth_type == 'median':
            median_arr = np.zeros(window_size)
            for y in range(window_size):
                median_arr[y] = data_arr[x + y] # add to our holder array
            smoothed_data_arr[x] = np.median(median_arr) # take median and add to our
        else:
            return smoothed_data_arr

    return smoothed_data_arr
```

In [ ]:

```
# Tesla closing prices, smooth_type = 'mean', window_size = 100
# Note your smoothed data will be shorter than the original
```

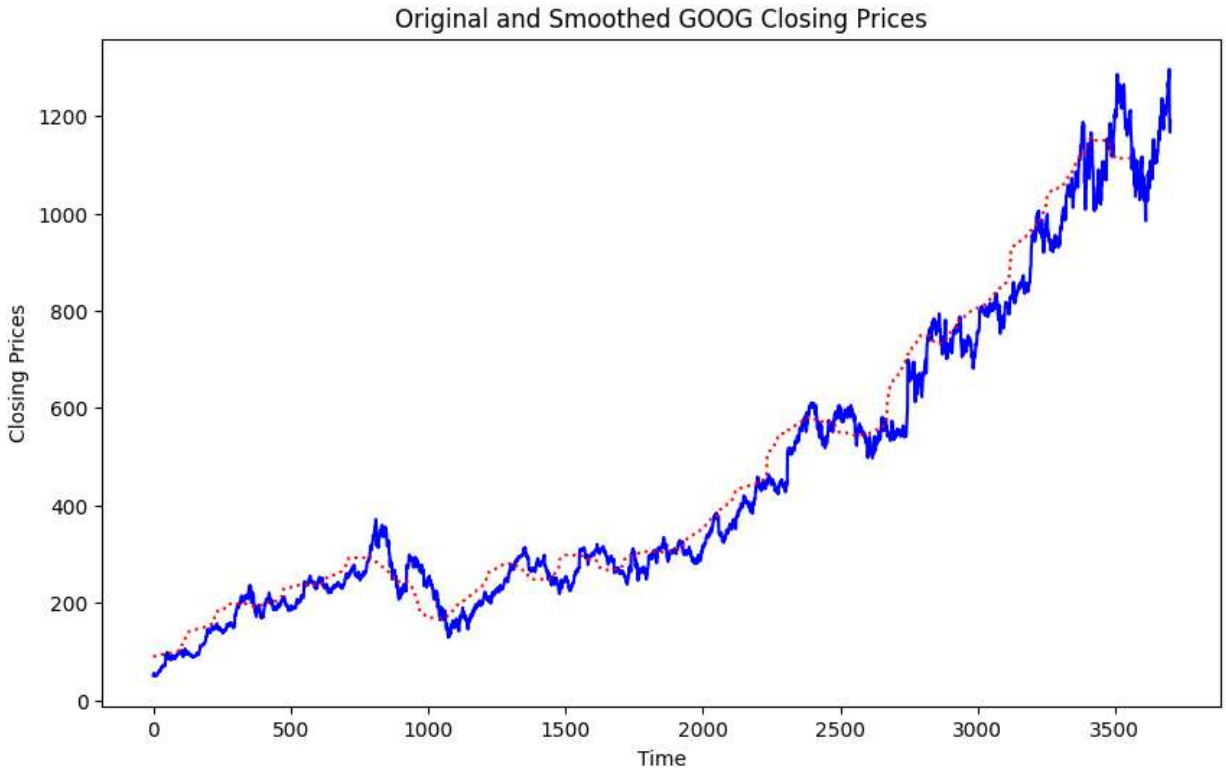
```
smoothed_tsla_closing  = smooth_data(tesla_np_closing, smooth_type = 'mean', window_si
```

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(tesla_np_closing, label='Original TSLA Closing Prices', color='blue')
plt.plot(smoothed_tsla_closing, label='Smoothed TSLA Closing Prices', linestyle='dotte

plt.xlabel('Time')
plt.ylabel('Closing Prices')
plt.title('Original and Smoothed TSLA Closing Prices')

plt.show()
```



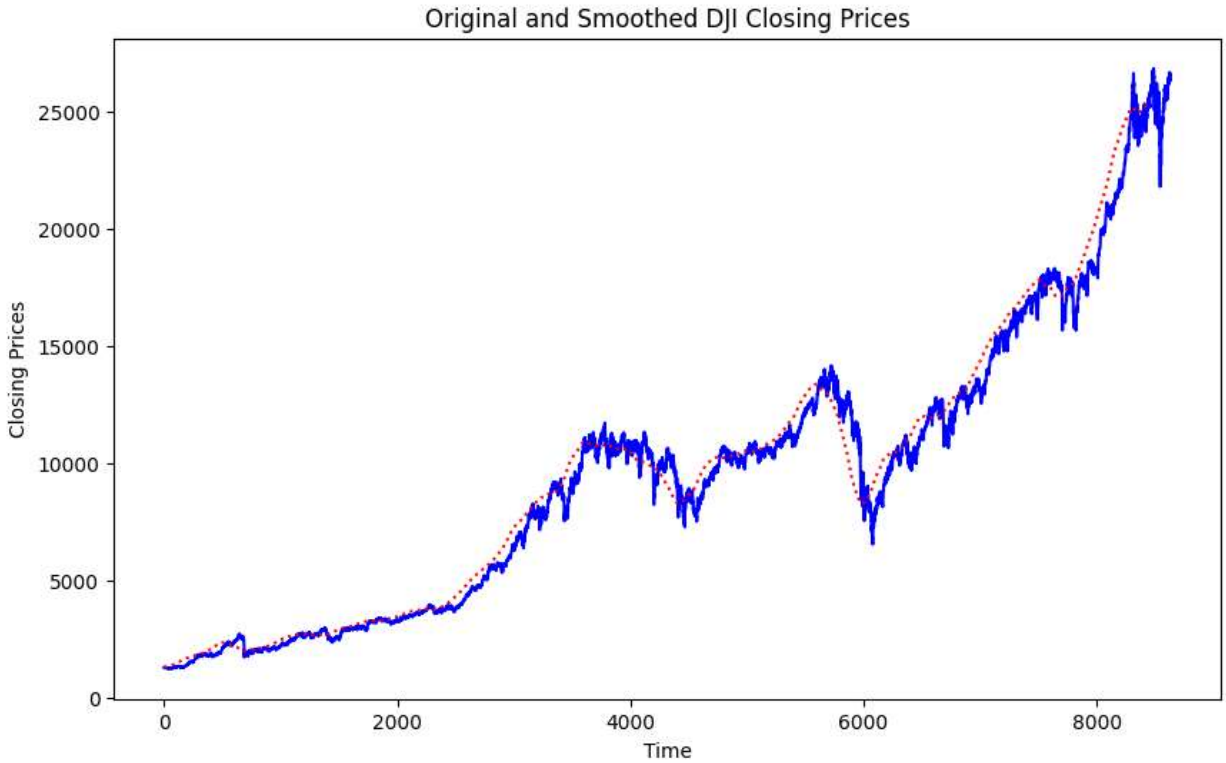
In [ ]:
```
# Google closing prices, smooth_type = 'median', window_size = 150
# Note your smoothed data will be shorter than the original

smoothed_google_closing  = smooth_data(google_np_closing, smooth_type = 'median', winc
```

In [ ]:
```
# plot smoothed_google_closing on top of google_np_closing
plt.figure(figsize=(10, 6))
plt.plot(google_np_closing, label='Original GOOG Closing Prices', color='blue')
plt.plot(smoothed_google_closing, label='Smoothed GOOG Closing Prices', linestyle='dot

plt.xlabel('Time')
plt.ylabel('Closing Prices')
plt.title('Original and Smoothed GOOG Closing Prices')

plt.show()
```



In [ ]:
```
# Dow Jones Index closing prices, smooth_type = 'mean', window_size = 200
# Note your smoothed data will be shorter than the original

smoothed_dji_closing  = smooth_data(DJI_np_closing, smooth_type = 'mean', window_size
```

In [ ]:
```
# plot smoothed_dji_closing on top of dji_np_closing
# plot smoothed_google_closing on top of google_np_closing
plt.figure(figsize=(10, 6))
plt.plot(DJI_np_closing, label='Original DJI Closing Prices', color='blue')
plt.plot(smoothed_dji_closing, label='Smoothed DJI Closing Prices', linestyle='dotted'

plt.xlabel('Time')
plt.ylabel('Closing Prices')
plt.title('Original and Smoothed DJI Closing Prices')

plt.show()
```



Extra credit: Code efficiency

Achieve a runtime speed of < 50ms

In [ ]:
```
timeit -n 1 -r 7 smoothed_google_closing  = smooth_data(google_np_closing, smooth_type
```

Exercise 4: Ranking Daily Stock Surges/Crashes

In [ ]:
```
ipyimage('lab4_exercise4.PNG', width = 1000)
```

Out[ ]:

EXERCISE 4: Ranking Daily Stock Surges/Crashes

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	2010-07-01	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	2010-07-02	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	2010-07-06	20.000000	20.00	15.830000	16.110001	16.110001	6866900

→ detect_surge_crash() →

Dates	Price changes
YYYY-MM-DD	65
YYYY-MM-DD	48
.	.
.	.
.	.

- Create a function **detect_surge_crash()** which takes a .csv file path as an input and output two lists - **dates and price changes (Close - Open)** associated with top surges (increase in stock) or crashes (decrease in stock).
- Use **Open** and **Close** columns of the stock data csv file to identify dates and price changes associated with highest surges or crashes - i.e. surge or crash within a **day's price movement (Close - Open)**. The output lists should list the items s.t. the date and price change associated with the largest surge or crash takes the 1st index, second largest takes the 2nd index and so on.
- The function should accept following parameters
  - filepath – Path to the .csv file you want analyze. Assume the file is of same structure as the lecture stock datasets
  - detect_type – Type of event to capture. Takes one of two values - 'surge', 'crash'.
  - num_output– Number of dates and price changes to output in each list. e.g. 5 - 5 dates and price changes corresponding to top 5 surges or crashes.
- Test your function against provided stock dataset in lab template.

52

In [ ]:

```
## method that reports the max crash or surge within n number of days back
# takes in a file path as well as what type 'surge' or 'crash'. A how many days back a
def detect_surge_crash(filepath, detect_type, num_output_dates):
    s_c_dict = convert_csv_to_dict(filepath)
    open = s_c_dict['Open']
    close = s_c_dict['Close']
    dates = s_c_dict['Date']
    # getting change between closing and opening
    movement = close - open
    #getting indices sort from smallest to largest
    sorted = np.argsort(movement)
    if detect_type.lower() == 'surge':
      ## getting max indices n days back
      indices = sorted[-num_output_dates:]
      # making sure to display largest numbers first
      max_indices = indices[::-1]
      price_change_list = movement[max_indices]
      date_list = dates[max_indices]
    else:
      # 'crash'
      indices = sorted[:num_output_dates]
      price_change_list = movement[indices]
      date_list = dates[indices]


    return date_list, price_change_list
```

In [ ]:

```
date_list_t, price_change_list_t = detect_surge_crash(filepath = 'TSLA.csv', detect_ty
```

In [ ]:

```
print(date_list_t, price_change_list_t)
```

```
['2018-08-07' '2018-04-04' '2018-10-23' '2018-12-26' '2018-10-26'] [35.730011 34.1600
03 30.27002  26.089996 22.649994]
```

In [ ]:

```
date_list_g, price_change_list_g = detect_surge_crash(filepath = 'GOOGL.csv', detect_t
```

In [ ]:
```
print(date_list_g, price_change_list_g)
```

```
['2018-10-29' '2018-10-24' '2018-03-27' '2018-02-08' '2018-12-04'
 '2018-10-10' '2018-12-21'] [-61.810059 -57.880005 -56.960022 -52.159973 -50.520019 -
44.23999
 -40.790039]
```

Extra credit: Code efficiency

Achieve a runtime speed of < 10ms

In [ ]:
```
timeit -n 1 -r 7 date_list_t, price_change_list_t = detect_surge_crash(filepath = 'TSL
```

```
102 ms ± 5.76 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Exercise 5: Human Debugger

In [ ]:
```
ipyimage('lab4_exercise5.PNG', width = 1000)
```

Out[ ]:

Faulty function #1

In [ ]:
```
def average_data_per_col(arr_2d):

    # NOTE FROM YOUR FRIEND PREPARING FOR STARBUCKS SOFTWARE ENGINEER TECH INTERVIEW
    """  The function takes numpy 2d array as an input, computes mean for each column
         with the length equal to the # of columns.

         For some reason I keep getting errors.... I need your help to debug the code.
         I need this position so I can get free ice lattes... :(

    """

    # placeholder for averaged values
```

```
    averaged_data = ()

    # Loop through each column data to compute mean and append to averaged_data
    for k in range(len(arr_2d[:, 0])):

        averaged_column_data = np.mean(arr_2d[:, k])
        averaged_data.append(averaged_column_data)

    # Return numpy array form of the averaged_data
    return np.array(averaged_data)
```

In [ ]:
```
# Load diabetes.csv and convert to numpy array

diabetes = pd.read_csv('/content/diabetes.csv')
diabetes_np = diabetes.to_numpy()
```

In [ ]:
```
# Run faulty function 1

averaged_diabetic_attributes =  average_data_per_col(diabetes_np)
```

In [ ]:
```
import numpy as np

def average_data_per_col_fixed(arr_2d):

    # tuples are immutable can't append
    averaged_data = []

    # was looping improperly so use shape
    for k in range(arr_2d.shape[1]):  # Use shape[1] to get the number of columns

        averaged_column_data = np.mean(arr_2d[:, k])
        averaged_data.append(averaged_column_data)

    # Return numpy array form of the averaged_data
    return np.array(averaged_data)
```

In [ ]:
```
# Test your fixed function

averaged_diabetic_attributes =  average_data_per_col_fixed(diabetes_np)
```

In [ ]:
```
# Compare with correct results

correct_result_func1 = np.load('/content/E5_correct_output_1.npy')

# Should return True if the result is correct
np.sum(np.round(correct_result_func1, 3) == np.round(averaged_diabetic_attributes, 3))
```

Out[ ]:
```
True
```

Faulty function #2

In [ ]:
```
def daily_stock_change_2_normalized_percentage(opening_price_arr, closing_price_arr):

    # NOTE FROM YOUR FRIEND WHO INVESTED IN TESLA
    """  I want to write a function which takes 2 1D numpy arrays of each correspondin
         and output 1D array of daily stock change in percentages.
```

```
        I want the percentages scale to be in a scale such that 1 = 100%, -0.5 = -50%
        For example, day 1 opening: $10, day 1 closing: $15, change in scaled percecn

        I am not really getting errors but the numbers don't look right... Can you he

    """

    # placeholder for change percentage values
    change_percentages = np.zeros(len(opening_price_arr), dtype = 'int')

    # Loop through each opening/closing price to compute the change percentage
    for date_num in range(len(opening_price_arr)):

        change_percentages[date_num] = opening_price_arr[date_num] - closing_price_arr

    return change_percentages
```

In [ ]:
```
# Load tsla.csv and convert to numpy array

tesla = pd.read_csv('TSLA.csv')
tesla_np = tesla.to_numpy()
```

In [ ]:
```
# Run faulty function 2
change_percentages = daily_stock_change_2_normalized_percentage(tesla_np[:, 1], tesla_
print(change_percentages)
```

```
[ 17  24  24 ... 237 244 242]
```

In [ ]:
```
# YOUR FIXED FUNCTION HERE

def daily_stock_change_2_normalized_percentage_fixed(opening_price_arr, closing_price_

   # placeholder for change percentage values
   ## changed to 'float' instead of 'int' because you need decimal points for percenta
    change_percentages = np.zeros(len(opening_price_arr), dtype = 'float')

    # Loop through each opening/closing price to compute the change percentage
    for date_num in range(len(opening_price_arr)):

        ## Switched opening and closing arrays so its close - open. Also put parenthes
        change_percentages[date_num] = (closing_price_arr[date_num] - opening_price_ar

    return change_percentages
```

In [ ]:
```
# Test your fixed function

change_percentages = daily_stock_change_2_normalized_percentage_fixed(tesla_np[:, 1],
print(change_percentages)
```

```
[ 0.25736837 -0.07599848 -0.12160004 ... -0.02026381 -0.00578363
  0.04580496]
```

In [ ]:
```
# Compare with correct results

correct_result_func2 = np.load('E5_correct_output_2.npy')

# Should return True if the result is correct
np.sum(np.round(correct_result_func2, 3) == np.round(change_percentages, 3)) == len(cc
```

Out[ ]: True

Faulty function #3

In [ ]:
```
def subset_diabetes_by_age(diabetes_data):

    # NOTE FROM YOUR FRIEND WHO WORKS AT UW IHME
    """ The function takes diabetes pandas data frame as an input and outputs a subplo

        Specifically, I want to divide the diabetes data into three age groups -
            1. 20 to 40
            2. 40 to 60
            3. 60 to 80

        and plot 3 histograms of glucose distribution (50 bins per histogram) in 3 x 1

        I seem to be getting error from very beginning even before I divide the datase
        My coworkers at IHME prefer R rather than Python... so you are my only hope!

    """

    # Extract the age column of the diabetes_data
    age_column = diabetes_data[:, 8]

    # Construct boolean mask for each age group
    age_20_40_bool_mask = age_column > 20 + age_column < 40
    age_40_60_bool_mask = age_column > 40 + age_column < 60
    age_60_80_bool_mask = age_column > 60 + age_column < 80

    # Get glucose data for each age group via applying the boolean mask for each age g
    age_20_40_glucose = diabetes_data[age_20_40_bool_mask, 2]
    age_40_60_glucose = diabetes_data[age_40_60_bool_mask, 2]
    age_60_80_glucose = diabetes_data[age_60_80_bool_mask, 2]

    # Plot the histogram for each age group in 3 x 1 subplot
    fig = plt.figure(figsize=(15,7))

    plt.subplot(3,1,1)

    plt.hist(age_20_40_glucose, bins = 50)
    plt.title('Age 20 to 40', fontsize = 15)

    plt.subplot(3,1,2)

    plt.hist(age_40_60_glucose, bins = 50)
    plt.title('Age 40 to 60', fontsize = 15)

    plt.subplot(3,1,3)

    plt.hist(age_60_80_glucose, bins = 50)
    plt.title('Age 60 to 80', fontsize = 15)

    plt.tight_layout()

    plt.show()
```

In [ ]:
```
# Load diabetes.csv as pandas dataframe
```

```
diabetes = pd.read_csv('diabetes.csv')
```

In [ ]:

```
# Run faulty function 3

subset_diabetes_by_age(diabetes_data = diabetes)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

TypeError: '(slice(None, None, None), 8)' is an invalid key

During handling of the above exception, another exception occurred:

InvalidIndexError                         Traceback (most recent call last)
<ipython-input-113-a4feead92bde> in <cell line: 3>()
      1 # Run faulty function 3
      2
----> 3 subset_diabetes_by_age(diabetes_data = diabetes)

<ipython-input-111-514c1248fe57> in subset_diabetes_by_age(diabetes_data)
     17
     18     # Extract the age column of the diabetes_data
---> 19     age_column = diabetes_data[:, 8]
     20
     21     # Construct boolean mask for each age group

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in __getitem__(self, key)
   3805             if self.columns.nlevels > 1:
   3806                 return self._getitem_multilevel(key)
-> 3807             indexer = self.columns.get_loc(key)
   3808             if is_integer(indexer):
   3809                 indexer = [indexer]

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3807                 #  InvalidIndexError. Otherwise we fall through and re-raise
   3808                 #  the TypeError.
-> 3809                 self._check_indexing_error(key)
   3810                 raise
   3811

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in _check_indexing_error(self, key)
   5923             # if key is not a scalar, directly raise an error (the code below
   5924             # would convert to numpy arrays and raise later any way) - GH29926
-> 5925             raise InvalidIndexError(key)
   5926
   5927     @cache_readonly

InvalidIndexError: (slice(None, None, None), 8)
```

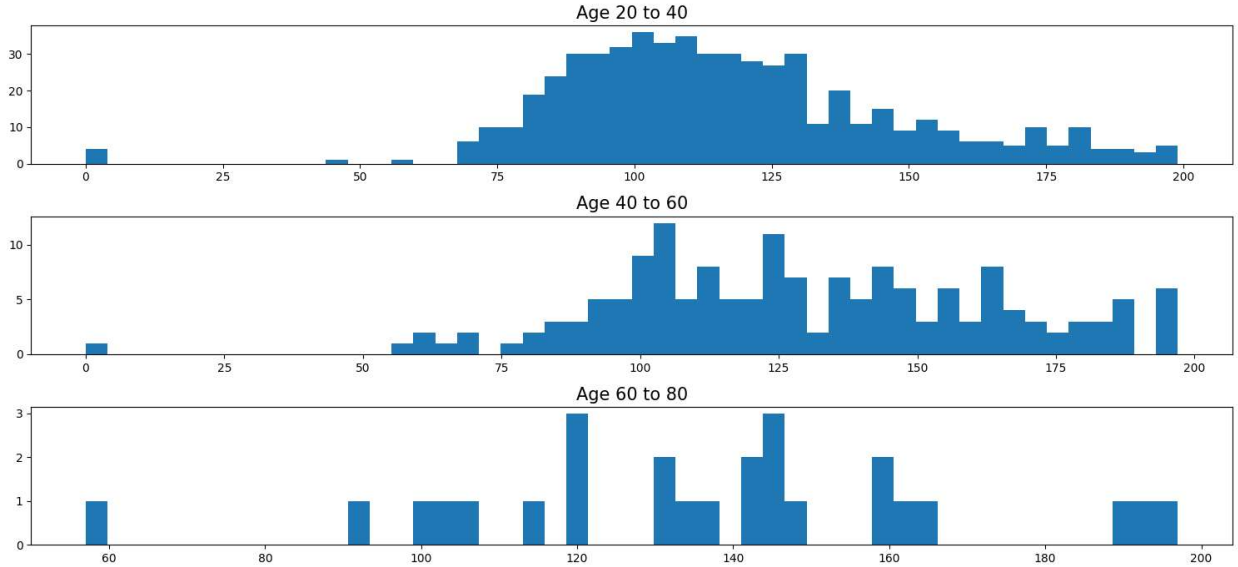
In [ ]:

```
# YOUR FIXED FUNCTION HERE
# Make sure you comment on your fixes
```

```
def subset_diabetes_by_age_fixed(diabetes_data):


    ## turned pandas data form to numpy array
    diabetes_np = diabetes_data.to_numpy()
    # Extract the age column of the diabetes_data

   ## wrong index for age it is 7
    age_column = diabetes_np[:, 7]
    # Construct boolean mask for each age group

    ## had to use np logic function
    age_20_40_bool_mask = np.logical_and(age_column > 20, age_column < 40)
    age_40_60_bool_mask = np.logical_and(age_column > 40, age_column < 60)
    age_60_80_bool_mask = np.logical_and(age_column > 60, age_column < 80)

    # Get glucose data for each age group via applying the boolean mask for each age g
    ## wrong index for glucose
    age_20_40_glucose = diabetes_np[age_20_40_bool_mask, 1]
    age_40_60_glucose = diabetes_np[age_40_60_bool_mask, 1]
    age_60_80_glucose = diabetes_np[age_60_80_bool_mask, 1]

    # Plot the histogram for each age group in 3 x 1 subplot
    fig = plt.figure(figsize=(15,7))

    plt.subplot(3,1,1)

    plt.hist(age_20_40_glucose, bins = 50)
    plt.title('Age 20 to 40', fontsize = 15)

    plt.subplot(3,1,2)

    plt.hist(age_40_60_glucose, bins = 50)
    plt.title('Age 40 to 60', fontsize = 15)

    plt.subplot(3,1,3)

    plt.hist(age_60_80_glucose, bins = 50)
    plt.title('Age 60 to 80', fontsize = 15)

    plt.tight_layout()

    plt.show()
```

In [ ]:

```
# Test your fixed function
# Compare your plot with the correct plot provided in template folder

subset_diabetes_by_age_fixed(diabetes_data = diabetes)
```



In [ ]: