LAB 4: WORKING WITH REAL WORLD STRUCTURED DATA

University of Washington

ECE 241

Winter 2022

Author: Jimin Kim (jk55@uw.edu)
Version: v1.5.0

OUTLINE

Part 1: Data Formats

- Data types
- Structured, Semi-structured, Unstructured data
- Reading in CSV data with Pandas package

Part 2: Data Structures in Python

- Arrays
- Tuples
- Dictionaries

Part 3: Visualizing Data

- Timeseries plots
- Bar graphs
- Scatter plots
- Histograms
- Colormaps

Part 4: Processing and Analyzing Data

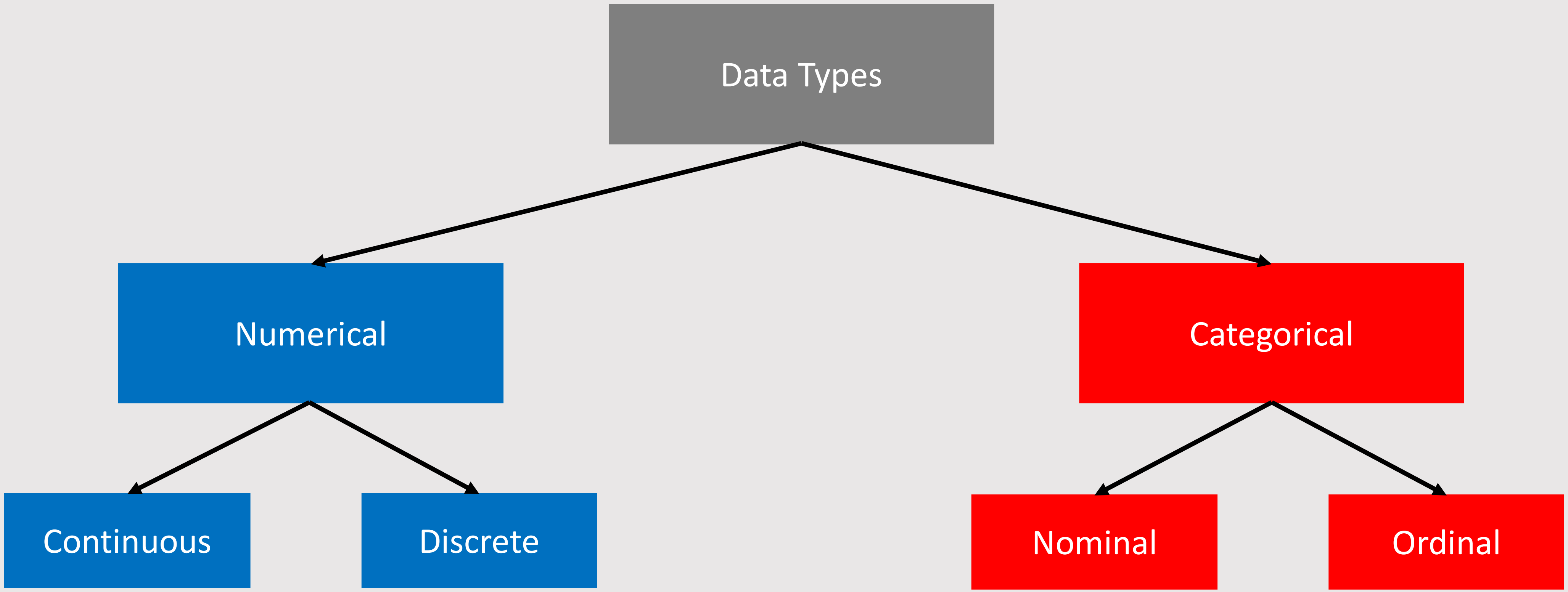
- Basic math operations
- Data smoothing
- Statistical analysis

Part 5: Lab Assignments

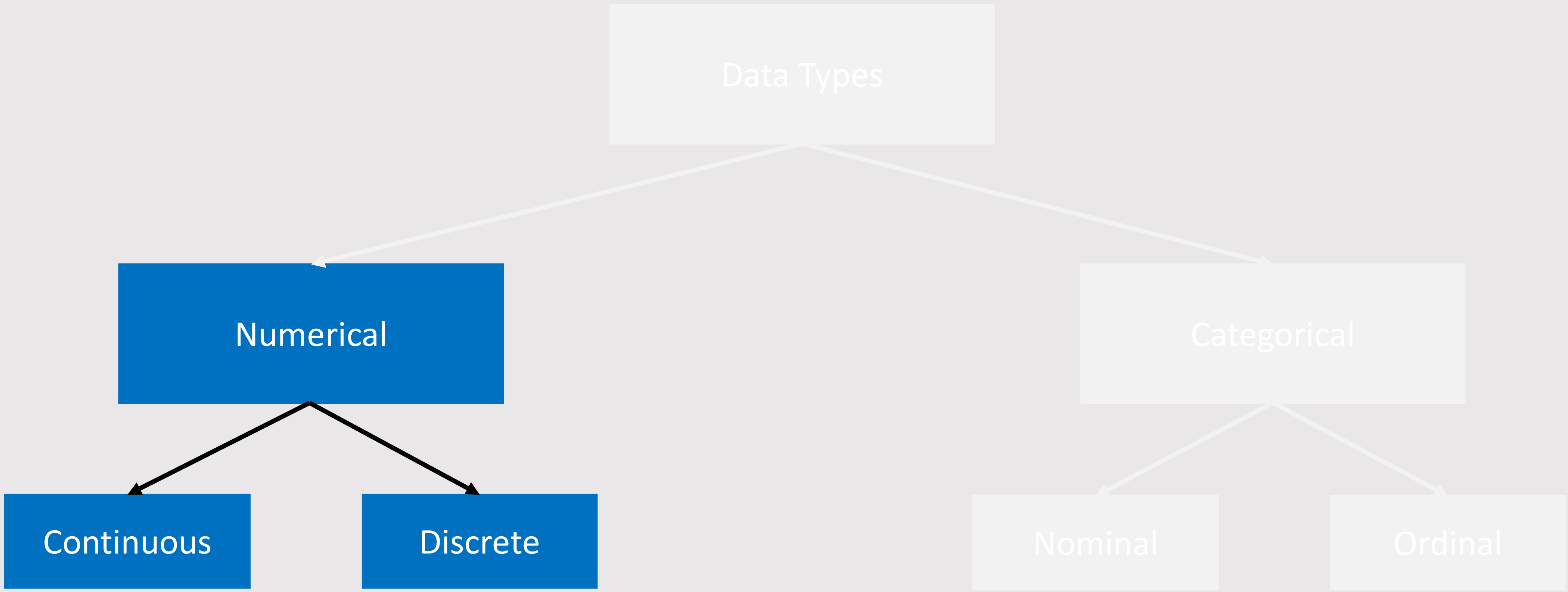
- Exercise 1 – 5

PART 1: DATA FORMATS

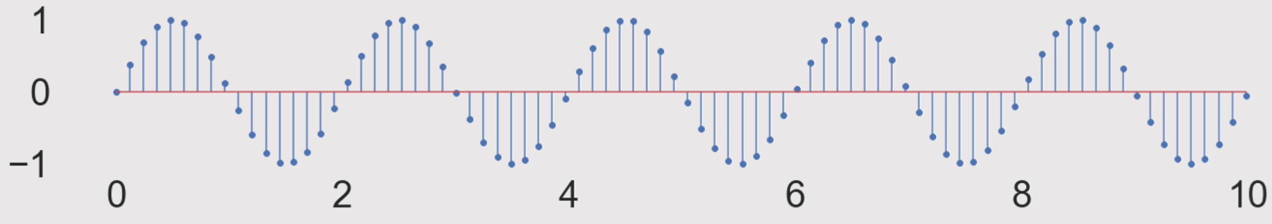
DATA TYPES


Data Types
Numerical
Categorical
Continuous
Discrete
Nominal
Ordinal

DATA TYPES: NUMERICAL



DATA TYPES: NUMERICAL



	Weekly work hours	Weekly coffee consumption (cups)
Student 1	40	7
Student 2	55	8
Student 3	33	5
Student 4	70	19

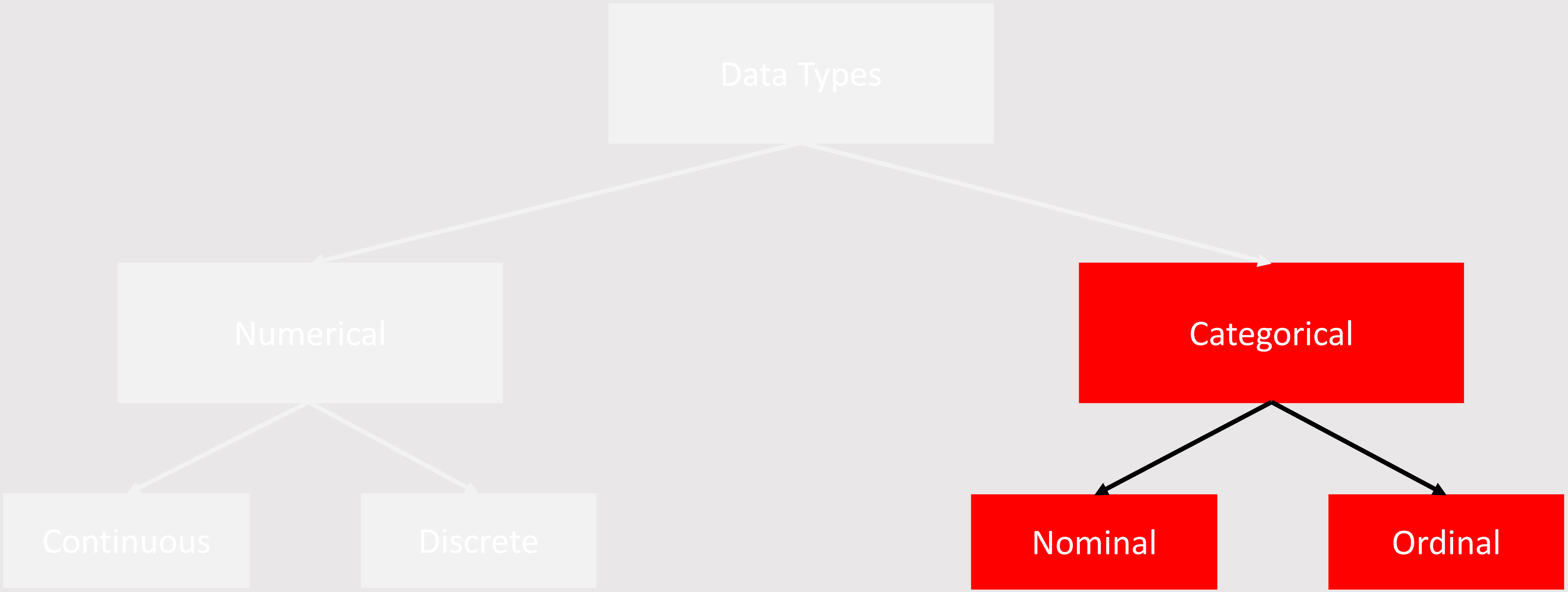
Continuous

e.g. amplitudes: 0.5, 0.76, -0.2

Discrete

e.g. work hours: 40, 55, 33

DATA TYPES: CATEGORICAL



DATA TYPES: CATEGORICAL

Favorite Dessert

	Ice cream	Fruits	Chocolate	Smoothie
Student 1			●	
Student 2	●			
Student 3		●		
Student 4				●

Nominal

(Named attributes)

Instructor’s enthusiasm was

	Excellent	Very Good	Good	Fair	Poor	Very Poor
Student 1		●				
Student 2			●			
Student 3	●					
Student 4					●	

Ordinal

(Named + ordered attributes)

STRUCTURED, SEMI-STRUCTURED, UNSTRUCTURED DATA

Structured

Semi-Structured

Unstructured

Less common

Easier to analyze

More common

Harder to analyze

STRUCTURED, SEMI-STRUCTURED, UNSTRUCTURED DATA

Structured

ID	Nation	G	S	B
1	USA	1027	800	704
2	USSR	395	319	296
3	UK	263	295	293
4	China	224	167	155
5	France	212	241	263

Summer Olympics Medal Counts by Nation

Semi-Structured

```json
{
  "All time medal counts": [
    {
      "ID": 1,
      "Nation": "USA",
      "Gold": 1027,
      "Silver": 800,
      "Bronze": 704
    },
    {
      "ID": 2,
      "Nation": "USSR",
      "Gold": 395,
      "Silver": 319,
      "Bronze": 296
    },
```

Unstructured

```
Top 5 nations for all time olympic medal counts are as follows:

USA is ID 1. USA earned 1027 Gold, 800 Silver, and 704 Bronze,
coming first in the rank.

USSR is ID 2. USSR earned 395 Gold, 316 Silver, and 296 Bronze,
coming second in the rank.

UK is ID 3. UK earned 263 Gold, 295 Silver, and 293 Bronze,
coming third in the rank.

....
```

STRUCTURED, SEMI-STRUCTURED, UNSTRUCTURED DATA

Structured

ID	Nation	G	S	B
1	USA	1027	800	704
2	USSR	395	319	296
3	UK	263	295	293
4	China	224	167	155
5	France	212	241	263

CSV, XLS…

Semi-Structured

```
{
  "All time medal counts": [
    {
      "ID": 1,
      "Nation": "USA",
      "Gold": 1027,
      "Silver": 800,
      "Bronze": 704
    },
    {
      "ID": 2,
      "Nation": "USSR",
      "Gold": 395,
      "Silver": 319,
      "Bronze": 296
    },
```

JSON, HTML…

Unstructured

```
Top 5 nations for all time olympic medal counts are as follows:

USA is ID 1. USA earned 1027 Gold, 800 Silver, and 704 Bronze,
coming first in the rank.

USSR is ID 2. USSR earned 395 Gold, 316 Silver, and 296 Bronze,
coming second in the rank.

UK is ID 3. UK earned 263 Gold, 295 Silver, and 293 Bronze,
coming third in the rank.

....
```

DOC, TXT, PDF…

DIFFERENT TYPES OF DATA STRUCTURES: FILE FORMATS

Structured

ID	Nation	G	S	B
1	USA	1027	800	704
2	USSR	395	319	296
3	UK	263	295	293
4	China	224	167	155
5	France	212	241	263

$\boxed{\text{CSV,}}$ XLS…

Semi-Structured

```
{
  "All time medal counts": [
    {
      "ID": 1,
      "Nation": "USA",
      "Gold": 1027,
      "Silver": 800,
      "Bronze": 704
    },
    {
      "ID": 2,
      "Nation": "USSR",
      "Gold": 395,
      "Silver": 319,
      "Bronze": 296
    },
```

JSON, HTML…

Unstructured

```
Top 5 nations for all time olympic medal counts are as follows:

USA is ID 1. USA earned 1027 Gold, 800 Silver, and 704 Bronze,
coming first in the rank.

USSR is ID 2. USSR earned 395 Gold, 316 Silver, and 296 Bronze,
coming second in the rank.

UK is ID 3. UK earned 263 Gold, 295 Silver, and 293 Bronze,
coming third in the rank.

....
```

DOC, TXT, PDF…

We will work with CSV data formats in this lab

EXAMPLE DATA 1: STOCK TIMESERIES DATA

	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

- TSLA.csv
- 2227 days
- 7 attributes

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	50.050049	52.082081	48.028027	50.220219	50.220219	44659000
1	2004-08-20	50.555557	54.594593	50.300301	54.209209	54.209209	22834300
2	2004-08-23	55.430431	56.796795	54.579578	54.754753	54.754753	18256100
3	2004-08-24	55.675674	55.855854	51.836838	52.487488	52.487488	15247300
4	2004-08-25	52.532532	54.054054	51.991993	53.053055	53.053055	9188600

- GOOGL.csv
- 3702 days
- 7 attributes

EXAMPLE DATA 2: DIABETES DATA

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	0	0	0	30.0	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	0	0	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1

- **diabetes.csv**

- 768 individuals
- 9 health metrics
- Outcome column indicates diabetes diagnosis (1: True, 0: False)

LOADING CSV DATA WITH PYTHON: PANDAS PACKAGE

What is Pandas Package?

- Python package for data manipulation and analysis
- Designed to work with structured datasets – e.g. relational, labeled data sets
- Provides integrated data structures – e.g. 1D series, 2D data frames
- Seamless conversions into Numpy arrays and vice versa

LOADING CSV DATA WITH PANDAS: DIABETES DATA

```
import pandas as pd

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

diabetes.head(n = 5)
```

Import Pandas package

Load csv file using read_csv()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Preview first few rows with head()

```
type(diabetes)
```

```
pandas.core.frame.DataFrame
```

Loaded csv file is a pandas DataFrame object

```
diabetes_np = diabetes.to_numpy()
print(diabetes_np)
```

```
[[  6.    148.     72.    ...   0.627  50.       1.    ]
 [  1.     85.     66.    ...   0.351  31.       0.    ]
 [  8.    183.     64.    ...   0.672  32.       1.    ]
 ...
 [  5.    121.     72.    ...   0.245  30.       0.    ]
 [  1.    126.     60.    ...   0.349  47.       1.    ]
 [  1.     93.     70.    ...   0.315  23.       0.    ]]
```

Convert to Numpy array with .to_numpy()

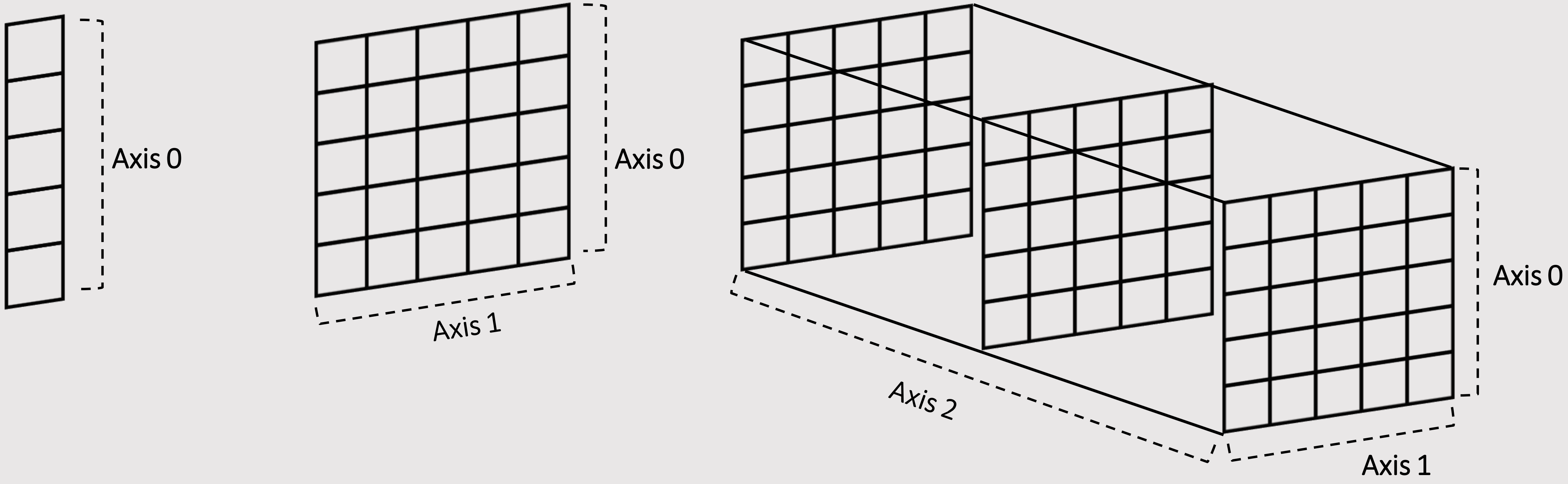
```
diabetes_np.shape
```

```
(768, 9)
```

Converted Numpy array has shape (768, 9)

PART 2: DATA STRUCTURES IN PYTHON

DATA STRUCTURES: NUMPY ARRAYS []



1-D

Shape = (i,)

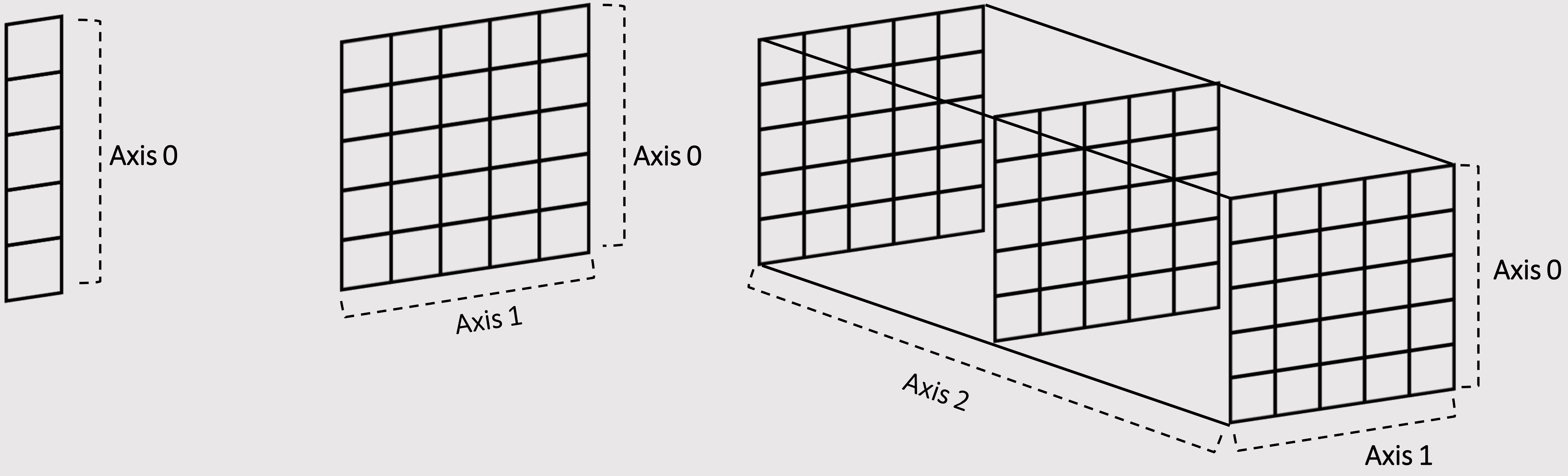
2-D

Shape = (i,j)

3-D

Shape = (i,j,k)

DATA STRUCTURES: NUMPY ARRAYS []



1-D

Shape = (i,)

e.g. mono sound data

2-D

Shape = (i,j)

e.g. data frame, table, greyscale image

3-D

Shape = (i,j,k)

e.g. RGB color image, stacked images

DATA STRUCTURES: NUMPY ARRAYS []

```
import numpy as np
```

```
# 1D

array_1d = diabetes_np[:, 1] # Glucose column
print(array_1d.shape)
```

```
(768,)
```

1D array example

```
# 2D

array_2d = diabetes_np[:, 1:4] # Glucose column - skin thickness
print(array_2d.shape)
```

```
(768, 3)
```

2D array example

```
# 3D

diabetes_np_first100 = diabetes_np[:100, :]           # First 100 rows (row 0 - row 100)
diabetes_np_100_to_200 = diabetes_np[100:200, :]      # Row 100 - 200

print(diabetes_np_first100.shape, diabetes_np_100_to_200.shape)  # Each sub-data is 2D array

array_3d = np.stack([diabetes_np_first100, diabetes_np_100_to_200]) # Using np.stack() to combine 2D arrays -> 3D
print(array_3d.shape)
```

```
(100, 9) (100, 9)
(2, 100, 9)
```

3D array example

DATA STRUCTURES: TUPLES ()

T = (20, ‘Python’, 36.5, [1, 5, 10])

T[0] T[1] T[2] T[3]

DATA STRUCTURES: TUPLES ()

```
tuple_1 = (1,2,3,4,5)

print(tuple_1)
```

```
(1, 2, 3, 4, 5)
```

Tuples are defined by casting items in ()

```
tuple_2 = (1,2,3, 'banana', 'apple', 'orange')

print(tuple_2)
```

```
(1, 2, 3, 'banana', 'apple', 'orange')
```

Tuple can store different data types (e.g. integer, string) like list

Tuples vs Lists – Tuples are **immutable**

```
tuple_1 = (1,2,3,4,5)
list_1 = [1,2,3,4,5]
```

1,2,3,4,5 sequence as both list and tuple

```
list_1[0] = 10
print(list_1)
```

```
[10, 2, 3, 4, 5]
```

Changing first element of the list to 10

```
tuple_1[0] = 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-b72e5a26927e> in <module>
----> 1 tuple_1[0] = 10

TypeError: 'tuple' object does not support item assignment
```

Doing the same results in an error with tuple

DATA STRUCTURES: TUPLES () vs LISTS []

	Tuples	Lists
Mutability	Immutable	Mutable
Can change order?	No	Yes
Stored data types	Usually heterogeneous e.g. (Banana, 5)	Usually homogeneous e.g. [1,2,3,4]
Memory allocation	Smaller	Larger

DATA STRUCTURES: DICTIONARIES {}

ece241_dict = {

Key 1 “Department” : “UW ECE”,
Key 2 “Instructor”: “Jimin Kim”,
Key 3 “Number of students”: 100,
Key 4 “Number of students per lab”: np.array([20, 24, 24, 24, 8])
Key 5 “Topics covered”: [‘Python’, ‘Signal processing’, ‘Data Types’]

}

DATA STRUCTURES: DICTIONARIES {}

Dictionaries are Mapping style data structure

Data are stored in 'keys' – "Department", "instructor" …

Dict.keys() displays all the keys within the dictionary

```
ece241_dict = {

    "Department": 'UW ECE',
    "Instructor": 'Jimin Kim',
    "Number of students": 100,
    "Number of students per lab": np.array([20, 24, 24, 24, 8]),
    "Topics covered": ['Python', 'Signal processing', 'Data Types']
}
```

```
ece241_dict.keys()
```

```
dict_keys(['Department', 'Instructor', 'Number of students', 'Number of students per lab', 'Topics covered'])
```

```
ece241_dict['Department']
```

```
'UW ECE'
```

Data are accessed via referring to keys

```
ece241_dict['Number of students per lab']
```

```
array([20, 24, 24, 24,  8])
```

```
ece241_dict['Topics covered']
```

```
['Python', 'Signal processing', 'Data Types']
```

DATA STRUCTURES: DICTIONARIES {}

Adding a key to dictionary

```
ece241_dict['Meeting times'] = ['M', 'T', 'W', 'Th', 'F']
```

```
ece241_dict['Meeting times']
```

```
['M', 'T', 'W', 'Th', 'F']
```

Deleting a key from dictionary

```
del ece241_dict['Topics covered']
```

```
ece241_dict.keys()
```

```
dict_keys(['Department', 'Instructor', 'Number of students', 'Number of students per lab', 'Meeting times'])
```

You can also use .pop(key) to delete a key from dictionary

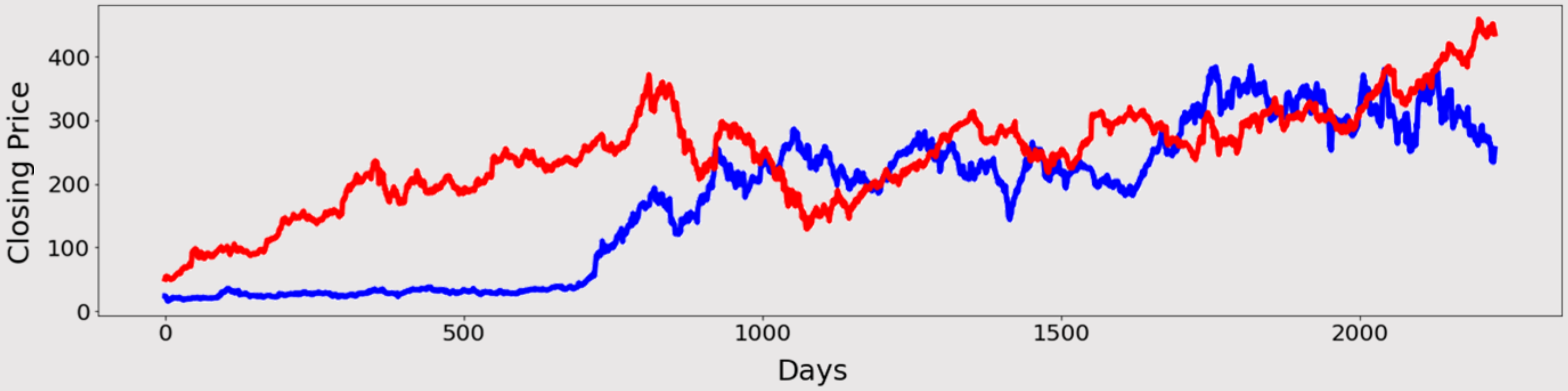
PART 3: VISUALIZING DATA

TIMESERIES PLOTS

```
fig = plt.figure(figsize=(23,5))

plt.plot(tesla_np[:len(tesla_np), 4], linewidth = 5, color = 'blue')
plt.plot(google_np[:len(tesla_np), 4], linewidth = 5, color = 'red')
plt.xlabel('Closing Price')
plt.ylabel('Days')
```

Set figure size

Plot closing price (column 5) of both tesla and google in a single plot



Tesla = Blue
Google = Red

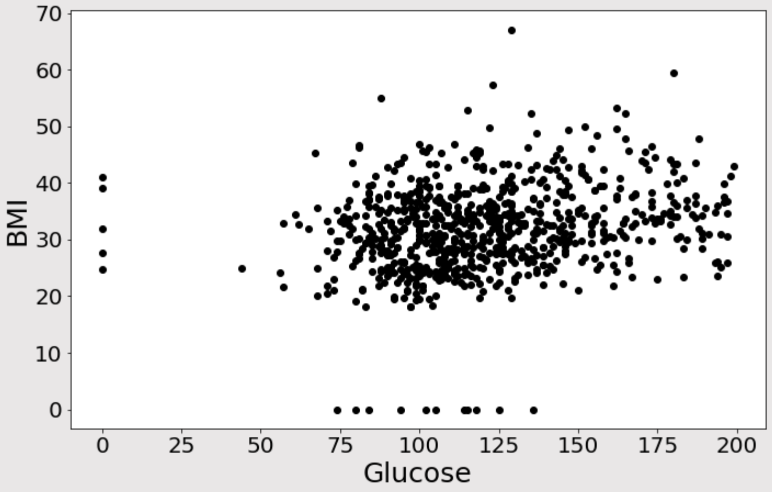
SCATTER PLOTS

```
fig = plt.figure(figsize=(20,7))

plt.subplot(1, 2, 1)
plt.scatter(diabetes_np[:, 1], diabetes_np[:, 5], color = 'black')
plt.xlabel('Glucose')
plt.ylabel('BMI')

plt.subplot(1, 2, 1)
plt.scatter(diabetes_np[:, 1], diabetes_np[:, 4], color = 'black')
plt.xlabel('Glucose')
plt.ylabel('Insulin')
```

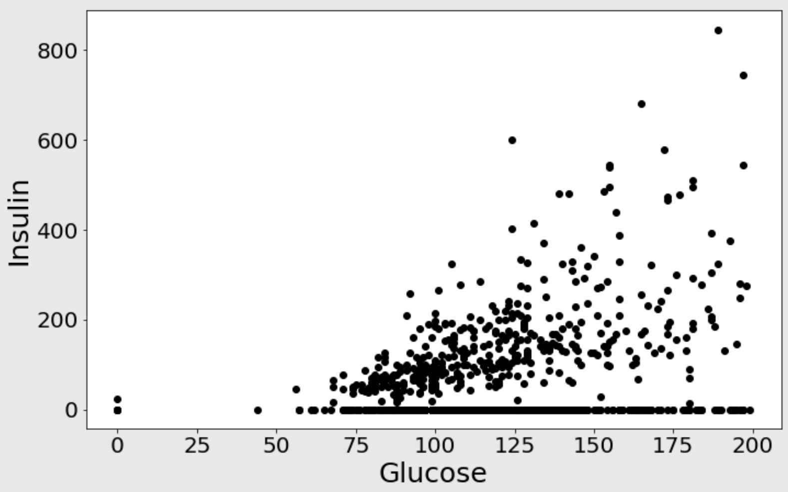
Set figure size

Compare Glucose (2nd column) vs BMI (6th column)

Compare Glucose (2nd column) vs Insulin (5th column)



Glucose vs BMI



Glucose vs Insulin

BAR GRAPHS

```
diabetes_pos_ind = diabetes_np[:, -1] == 1
diabetes_neg_ind = diabetes_np[:, -1] == 0

diabetes_np_pos = diabetes_np[diabetes_pos_ind, :]
diabetes_np_neg = diabetes_np[diabetes_neg_ind, :]

x_labels = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DPF', 'Age']

fig = plt.figure(figsize=(20,10))

plt.subplot(1, 2, 1)

plt.bar(x_labels, diabetes_np_pos.mean(axis = 0)[:-1], color = 'blue')
plt.ylim(0, 150)

plt.subplot(1, 2, 2)

plt.bar(x_labels, diabetes_np_neg.mean(axis = 0)[:-1], color = 'red')
plt.ylim(0, 150)
```

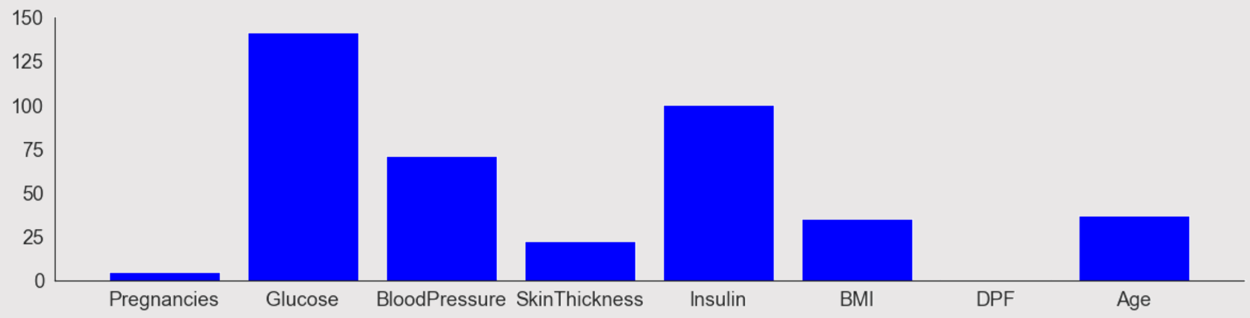
Extract rows with diabetes and no diabetes using Boolean masks

Split dataset into two

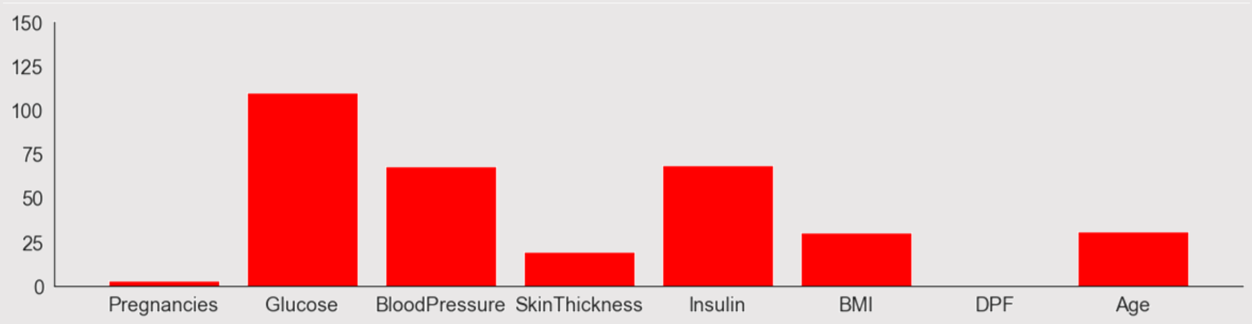
Construct x-label string list

Plot bar graphs of averaged attributes for each dataset

.mean() function is discussed in slide 32



Diabetes = 1



Diabetes = 0

COLORMAPS

	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

Select the columns to visualize (columns 1,2,3,4)

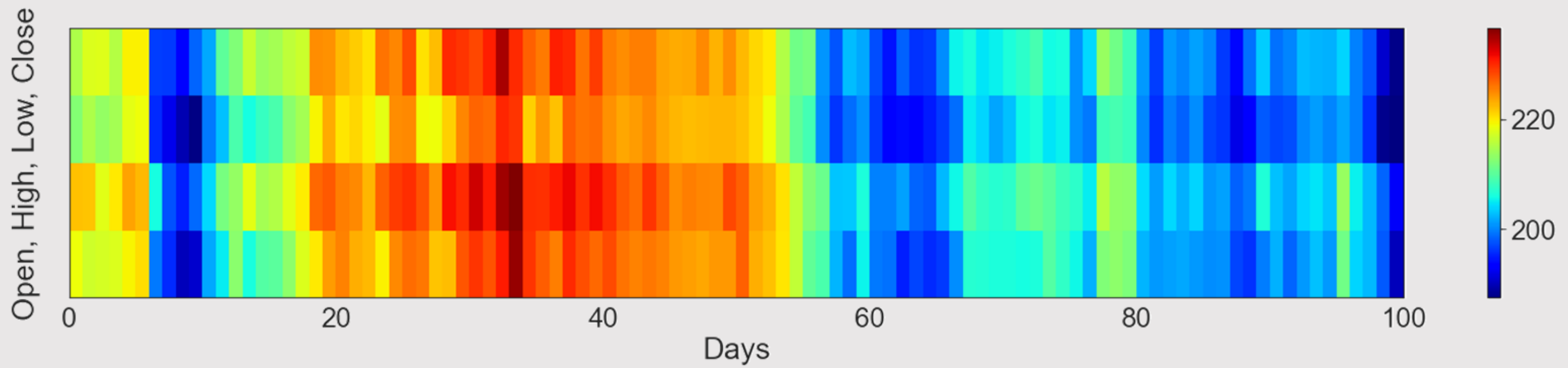
```
tesla_2_visualize = tesla_np[1500:1600, [1,2,3,4]]
tesla_2_visualize = tesla_2_visualize.T
```

Subset the rows and columns to visualize (row:1500th – 1600th days, columns: 1,2,3,4)

Apply transpose to dataset so that rows = attributes, columns = days

```
fig = plt.figure(figsize=(30,5))

plt.pcolor(tesla_2_visualize.astype('float'), cmap = 'jet')
plt.xlabel('Days')
plt.ylabel('Open, High, Low, Close')
plt.yticks(color='white')
plt.colorbar()
```

Convert the array type to 'float' to make sure the data is compatible with colormap function



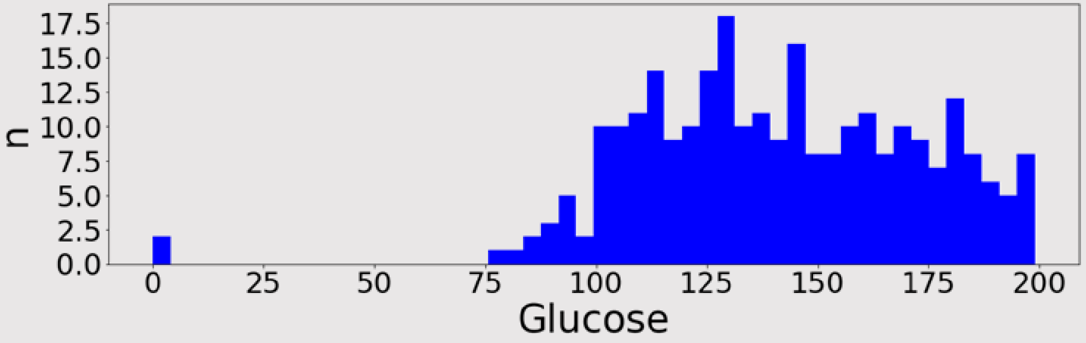
HISTOGRAMS

```
diabetes_pos_ind = diabetes_np[:, -1] == 1
diabetes_neg_ind = diabetes_np[:, -1] == 0

diabetes_np_pos = diabetes_np[diabetes_pos_ind, :]
diabetes_np_neg = diabetes_np[diabetes_neg_ind, :]

fig = plt.figure(figsize=(40,5))

plt.subplot(1, 2, 1)

plt.hist(diabetes_np_pos[:, 1], color = 'blue', bins = 50)
plt.xlabel('Glucose')
plt.ylabel('n')

plt.subplot(1, 2, 2)

plt.hist(diabetes_np_neg[:, 1], color = 'red', bins = 50)
plt.xlabel('Glucose')
plt.ylabel('n')
```

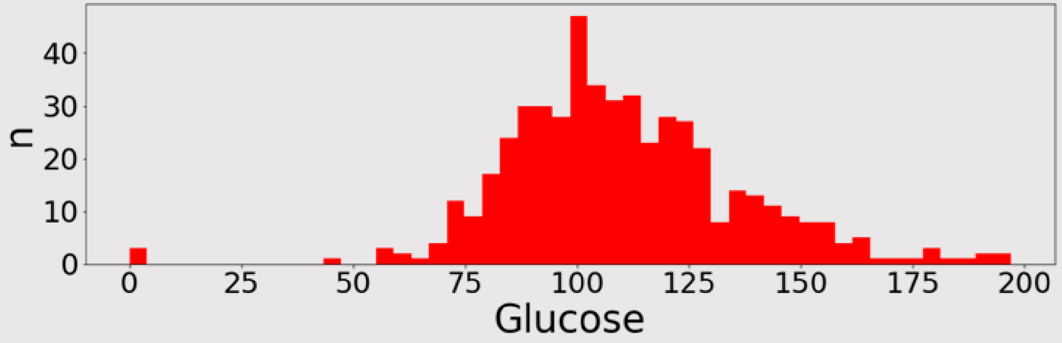
Extract rows with diabetes and no diabetes

Split dataset into two

Plot histogram of Glucose column for each dataset



Diabetes = 1



Diabetes = 0

PART 4: PROCESSING AND ANALYZING DATA

BASIC MATH OPERATIONS: SUMMATION ALONG AXIS

Axis 1

Axis 0

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

array2d

55	60	65	70	75

1+6+11+16+21

array2d.sum(axis = 0)

15	40	65	90	115

1+2+3+4+5

array2d.sum(axis = 1)

BASIC MATH OPERATIONS: AVERAGING ALONG AXIS

Axis 1

Axis 0

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

array2d

11	12	13	14	15

(1+6+11+16+21)/5

array2d.mean(axis = 0)

3	8	13	17	22

(1+2+3+4+5)/3

array2d.mean(axis = 1)

BASIC MATH OPERATIONS: MINIMUM ALONG AXIS

Axis 1

Axis 0

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

array2d

1	2	3	4	5

Min{1,6,11,16,21}

array2d.min(axis = 0)

1	6	11	16	21

Min{1,2,3,4,5}

array2d. min(axis = 1)

BASIC MATH OPERATIONS: MAXIMUM ALONG AXIS

Axis 1

Axis 0

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

array2d

21	22	23	24	25

Max{1,6,11,16,21}

array2d.max(axis = 0)

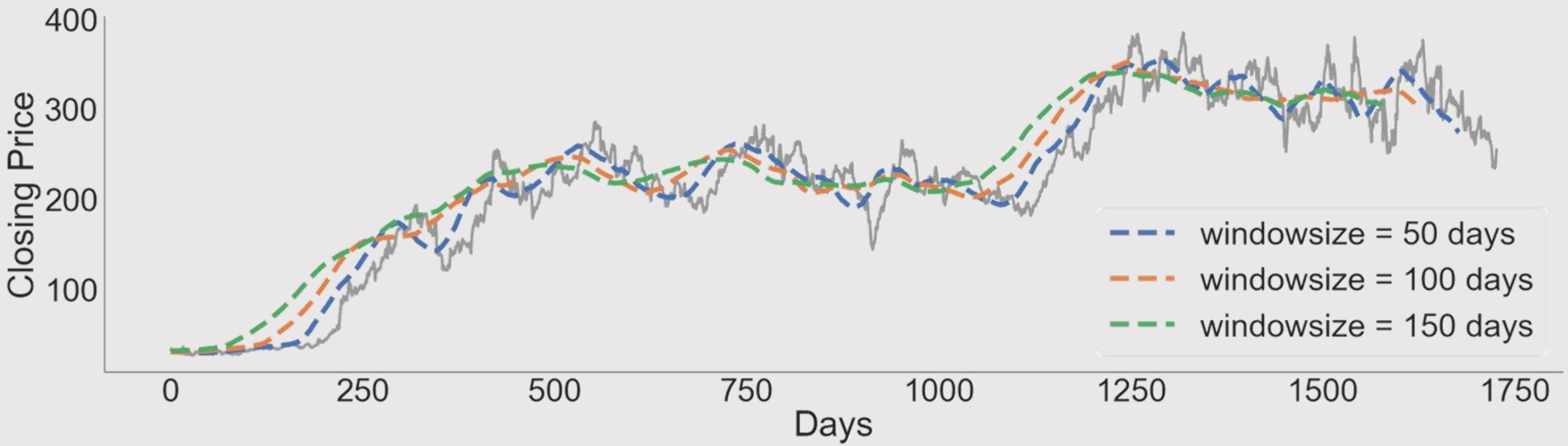
5	10	15	20	25

Max{1,2,3,4,5}

array2d.max(axis = 1)

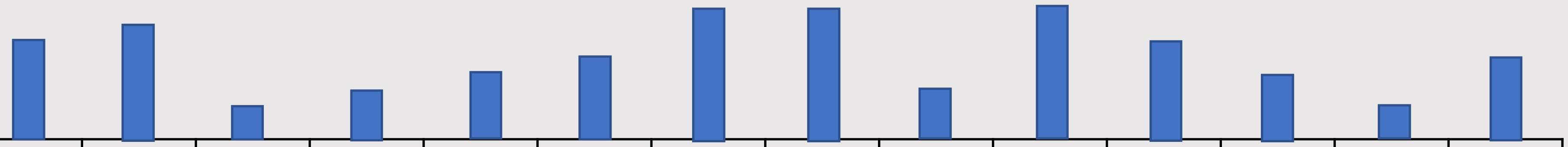
DATA SMOOTHING: ROLLING MEAN

Application to TSLA.csv (Day 500 – 2227, closing price)



Rolling mean smooths the noisy data

DATA SMOOTHING: ROLLING MEAN



6	7	2	3	4	5	8	8	3	8	6	4	2	5

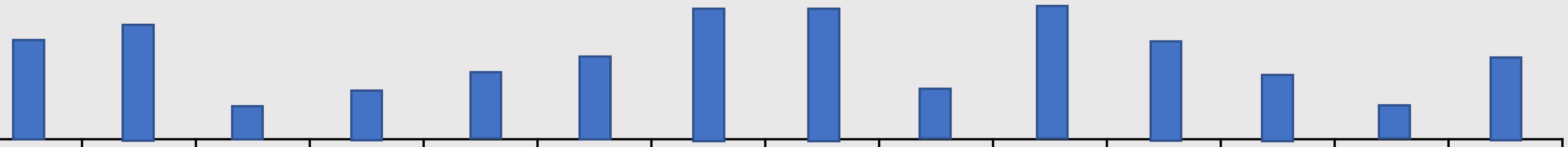
Window Size = 3

6	7	2	3	4	5	8	8	3	8	6	4	2	5

(6+7+2)/3

5													

DATA SMOOTHING: ROLLING MEAN



6	7	2	3	4	5	8	8	3	8	6	4	2	5

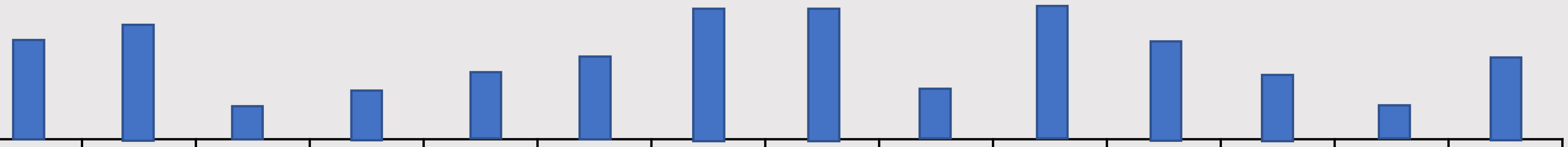
Window Size = 3

6	7	2	3	4	5	8	8	3	8	6	4	2	5

(7+2+3)/3

5	4												

DATA SMOOTHING: ROLLING MEAN



6	7	2	3	4	5	8	8	3	8	6	4	2	5

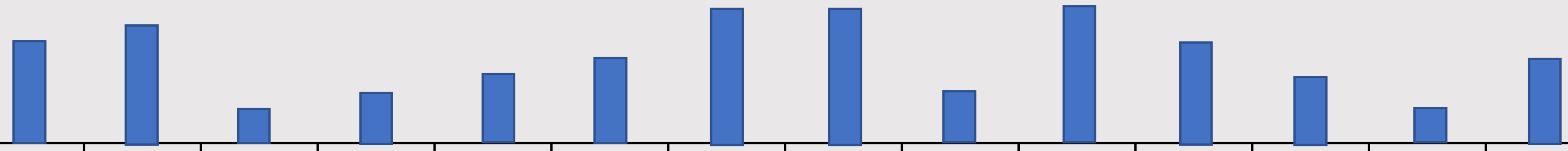
Window Size = 3

6	7	2	3	4	5	8	8	3	8	6	4	2	5

(2+3+4)/3

5	4	3											

DATA SMOOTHING: ROLLING MEAN



6	7	2	3	4	5	8	8	3	8	6	4	2	5

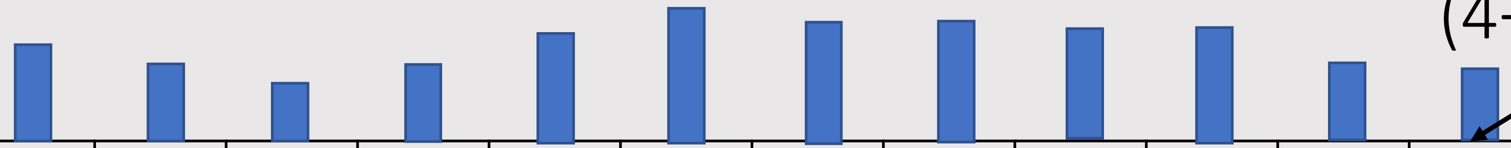
Window Size = 3

Note: Without padding, rolling operation reduces the data length by (window size - 1)

Note: Rolling median has identical principle except it uses **median value** of data window

6	7	2	3	4	5	8	8	3	8	6	4	2	5

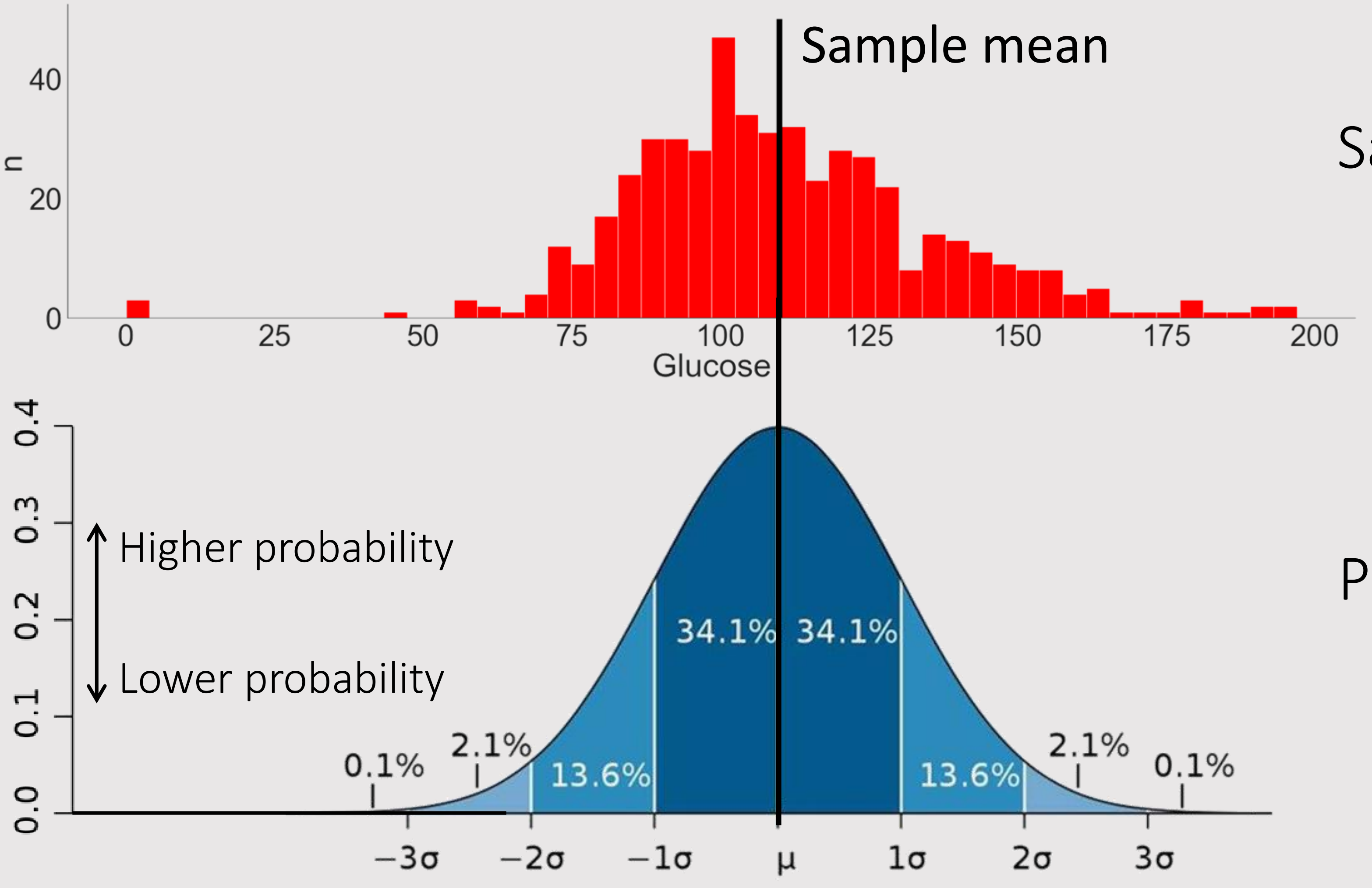
(4+2+5)/3



5	4	3	4	5.7	7	6.3	6.3	5.7	6	4	3.7

STATISTICAL ANALYSIS: CONFIDENCE INTERVALS

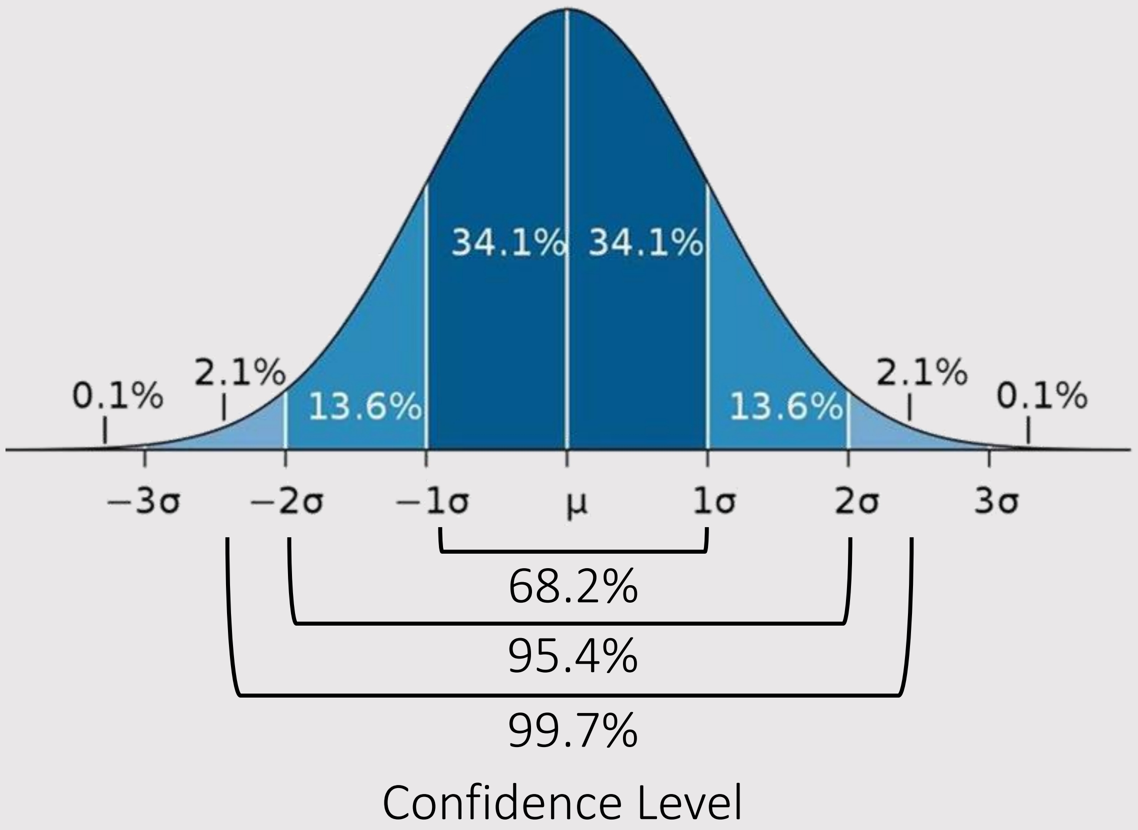
Can we estimate the population Glucose mean from our sample data?



Sample distribution
(n = 500)

Probability distribution of true population mean

STATISTICAL ANALYSIS: CONFIDENCE INTERVALS



CI = the probability that a parameter will fall between a pair of values around the mean

$$CI = \bar{x} \pm z\frac{s}{\sqrt{n}}$$

$\bar{x}$ = sample mean
z = confidence level value
s = sample standard deviation
n = sample size

Confidence Level	z
90%	1.645
95%	1.96
99%	2.576

STATISTICAL ANALYSIS: CONFIDENCE INTERVALS

```
import scipy.stats as st

glucose_control = diabetes_np_neg[:, 1]


CI_99_lower, CI_99_upper = st.t.interval(alpha=0.99, df=len(glucose_control)-1,
                                         loc=np.mean(glucose_control), scale=st.sem(glucose_control))

h = CI_99_upper - np.mean(glucose_control)
```

Import scipy.stats to use pre-built statistical functions

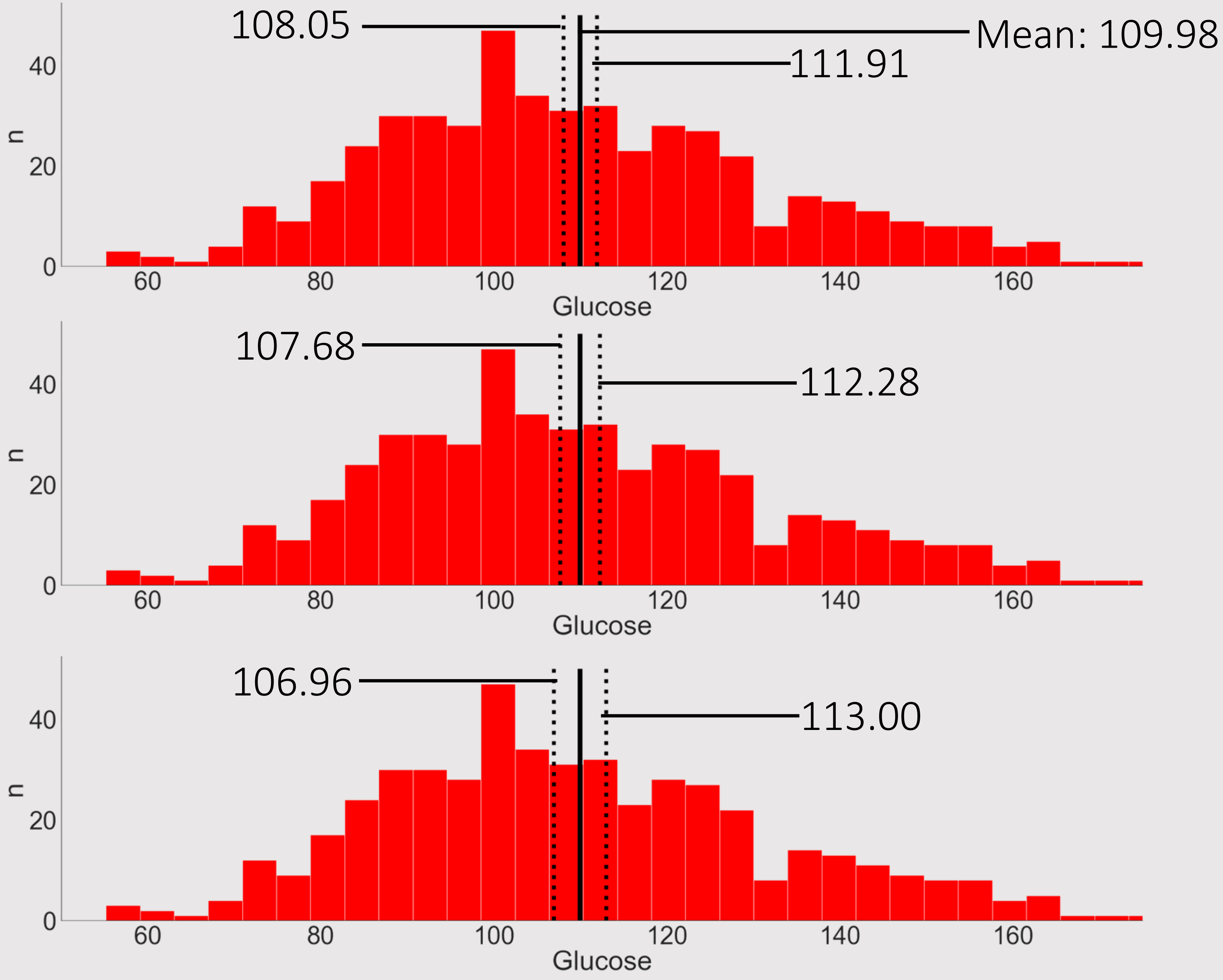
Extract glucose column from non-diabetic dataset

st.t.interval() computes lower and upper bound for provided confidence level using t-distribution

- alpha - confidence level
- df - degree of freedom (size of the data - 1)
- loc - The mean value of the data
- scale = standard error of the data

Confidence interval size

STATISTICAL ANALYSIS: CONFIDENCE INTERVALS



Confidence level = 90%

Confidence level = 95%

Confidence level = 99%

STATISTICAL ANALYSIS: CONFIDENCE INTERVALS

Including confidence intervals in bar graph

```
fig = plt.figure(figsize=(7,5))

plt.bar(['Non-diabetic'], [109.98],
        width = 0.5, color = 'black',
        yerr = [109.98 - 106.96], ecolor = 'grey',
        error_kw=dict(lw=5, capsize=50, capthick=5))
plt.xlim(-1, 1)
plt.ylabel('Glucose')
sns.despine()
```

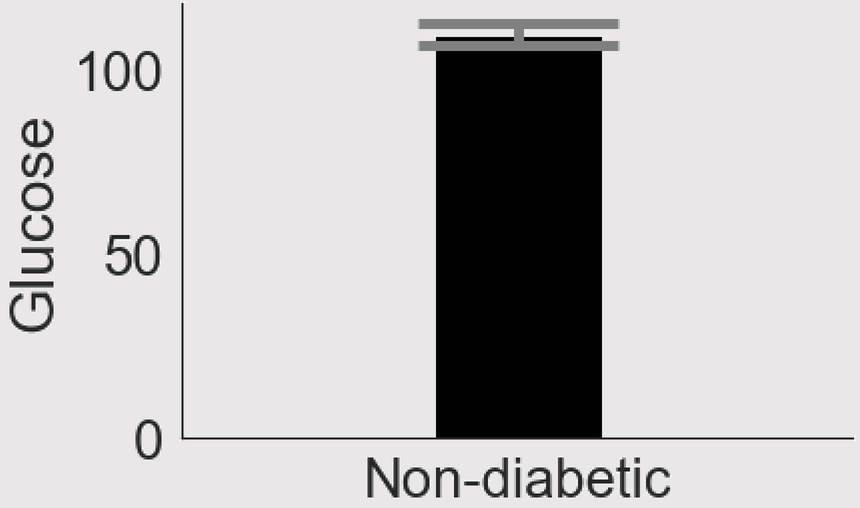
Set figure size

Define x and y-axis data for the bar

Define visual property of the main bar

Define the confidence interval size (h = $z\frac{s}{\sqrt{n}}$) and error bar color

Define visual properties of the confidence interval
- lw: Vertical linewidth
- capsize: Length of the horizontal lines
- capthink: Thickness of the horizontal lines



For more info:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html

LAB ASSIGNMENTS

Download ipynb template in Canvas page:
Assignments/Lab 4 report -> click “Lab 4 Report Templates”

EXERCISE 1: Construct Dictionaries from Data

	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

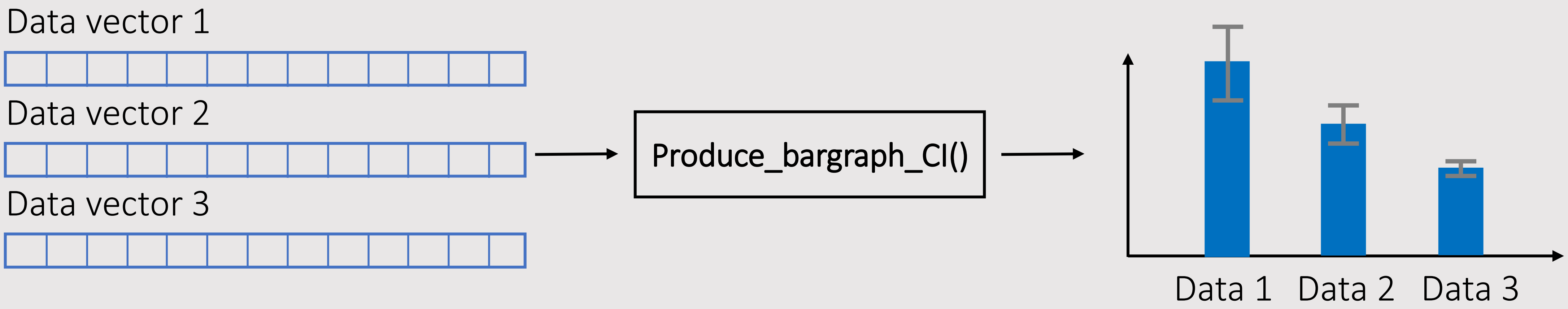
TSLA.csv

convert_csv_to_dict()

```
{
    "Filename": 'TSLA.csv'
    "Date": [...],
    "Open": [...],
    "high": [...],
    "low": [...],
    "close": [...]

}
```

- Create a function convert_csv_to_dict() which takes csv file path as input and output a python dictionary.
- The function should accept following parameters
  - file path – the path to the .csv file you want to convert
- The function should use the filename as “Filename” key and each column name as a key for the dictionary. Each key should represent a list or 1D numpy array containing strings, integers or float data corresponding to each column.
- Test your function against TSLA.csv and diabetes.csv and print the first 10 items of 2nd and 4th row of each dataset by referring to dictionary keys.

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.

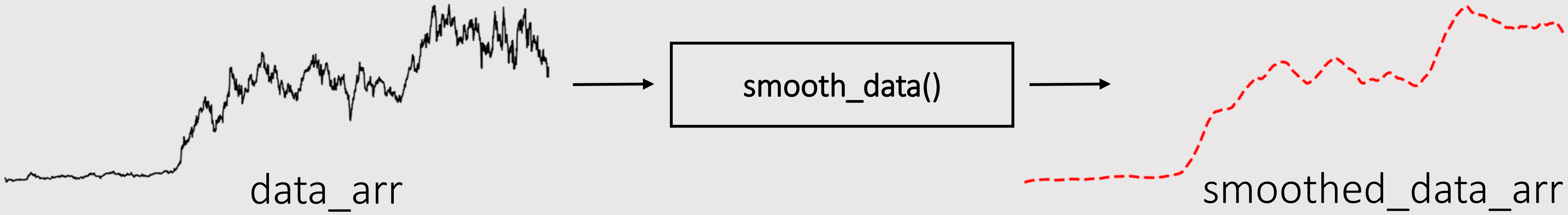
- The function should accept following parameters
  - data_vec_list = list of three 1D arrays each corresponding to a series of data
  - conf_level = Confidence level to be used for confidence interval – Takes one of three values - 0.9, 0.95, 0.99.
  - bar_labels = list of strings corresponding to labels 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.

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**

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
.	.
.	.
.	.

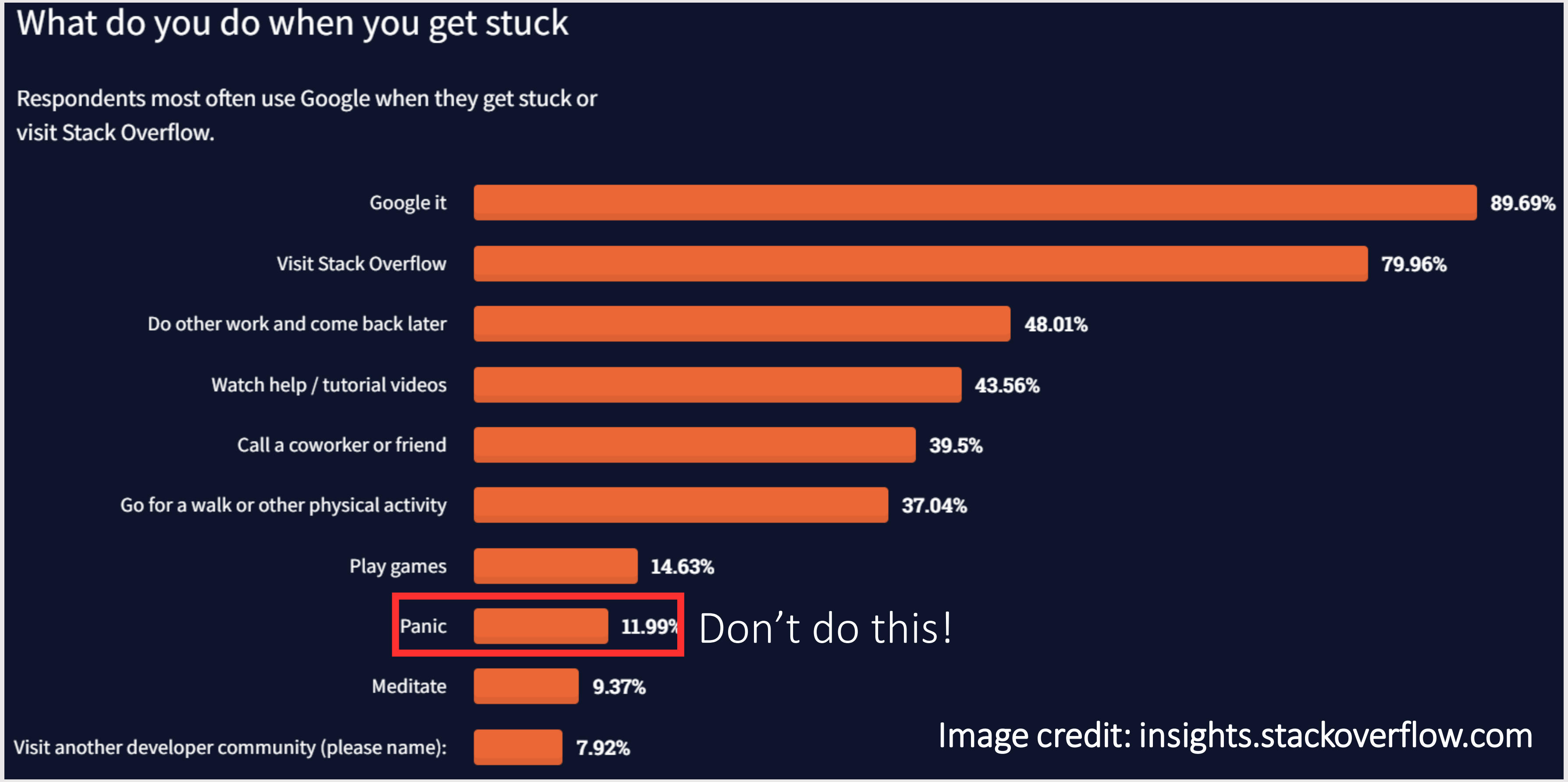
- Create a function **detect_surge_crash()** which takes a .csv file path as an input and output two lists - **dates and price changes (Close - Open)** associated with top surges (increase in stock) or crashes (decrease in stock).

- 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.

EXERCISE 5: Human Debugger



- In lab template ipynb, we included three functions each with errors preventing the function from running successfully. Your task is to identify the source of error and provide fixed functions for three examples.
- Feel free to use Google/Stack overflow to assist your debugging. **Make sure to comment how you fixed the issue.**
- Confirm that your fix has worked by comparing your outputs with the intended function outputs.