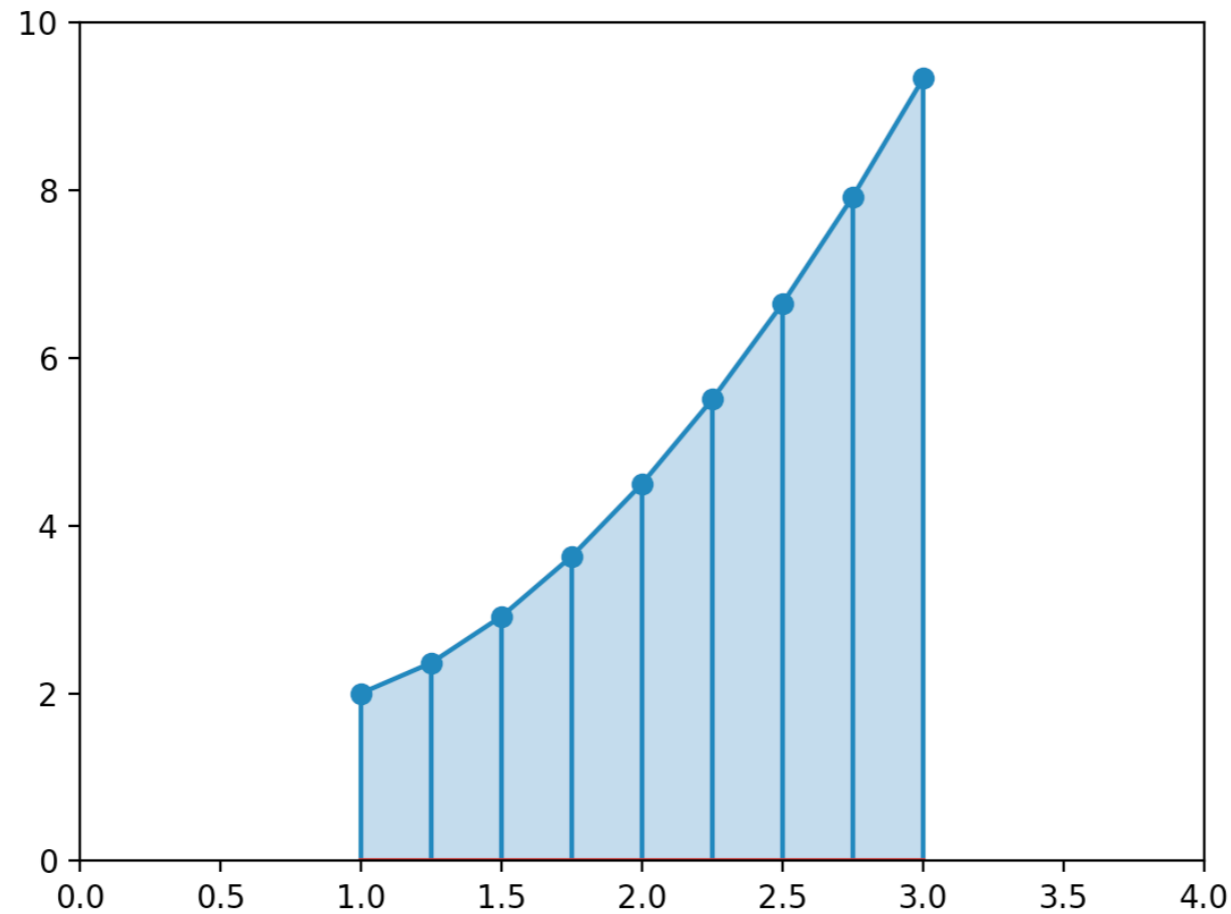


# **Week 4 Practice**

# Numerical Integration

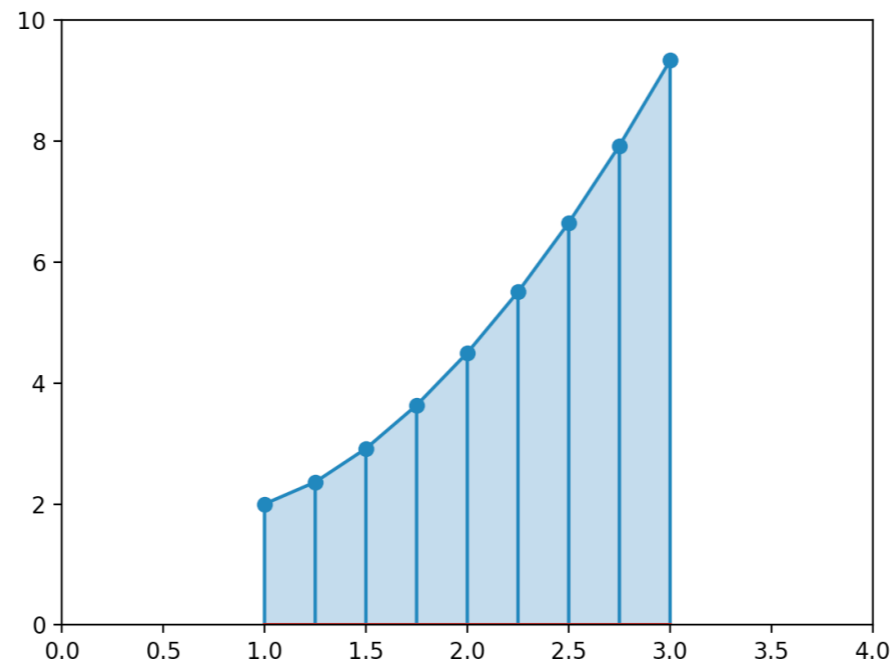
- How to calculate a one-dimensional integral?



# Numerical Integration

- How to calculate a one-dimensional integral?

- $\int_a^b f(x)dx$



- Approximate with trapezoids

# Numerical Integration

- We calculate the weighted average (first and last point get half weight) of  $n + 1$  points at distance  $\frac{b - a}{n}$
- And multiply by  $(b - a)$

# Numerical Integration

```
def integral(function, low, up, nr_pts):  
    delta = (up-low)/(nr_pts-1)  
    func_vals = [function(low + i*delta) for i in range(nr_pts)]  
    return (sum(func_vals)+sum(func_vals[1:-1]))/2*delta
```

# Scrabble Values

# Scrabble Values

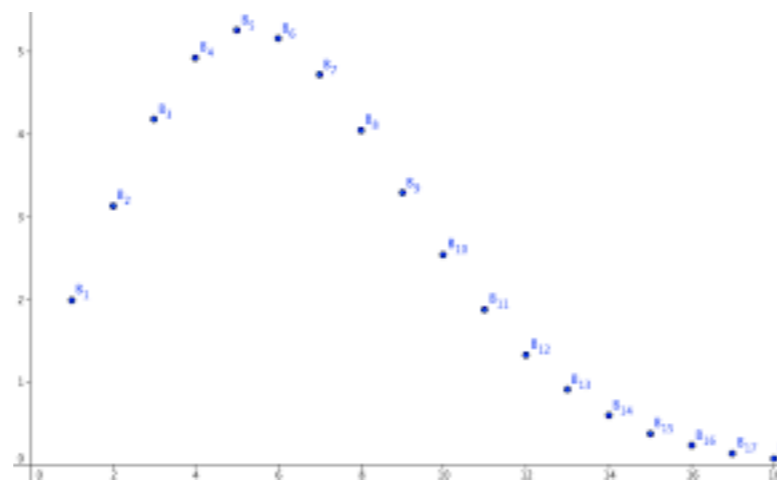
- Players place tiles with letters on a board to form a word
- The word is scored by letter value
  - 1 Point - A, E, I, L, N, O, R, S, T and U.
  - 2 Points - D and G.
  - 3 Points - B, C, M and P.
  - 4 Points - F, H, V, W and Y.
  - 5 Points - K.
  - 8 Points - J and X.
  - 10 Points - Q and Z.

# Curse of Dimensionality



# Curse of Dimensionality

- As the dimensions increase, our intuition will leave us
  - As the dimension increase, the volume of a unit-sphere goes down



# Simple Case Study

Thomas Schwarz, SJ

# Data Analytics Process

- Discovery phase: what data do we need
- Data preparation phase: Cleaning
- Modeling planning phase: how to go about dealing with the data
- Model building phase: build the models
- Communication phase
- Operationalization: build process into production

# Data Gathering

- Use Kaggle as a source of data
  - <https://www.kaggle.com/paultimothymooney/denver-crime-data/>
  - This is a huge file.

# Understanding the Data

- Read supporting information
- Look at the first lines of the file
  - There is a heading and examples

```
def head():  
    with open('crime.csv') as infile:  
        for _ in range(10):  
            print(infile.readline())
```

# Understanding the Data

```
>>> head()
"INCIDENT_ID", "OFFENSE_ID", "OFFENSE_CODE", "OFFENSE_CODE_EXTENSION", "OFFENSE_TYPE_ID", "OFFENSE_CATEGORY_ID", "FIRST_OCCURRENCE_DATE", "LAST_OCCURRENCE_DATE", "REPORTED_DATE", "INCIDENT_ADDRESS", "GEO_X", "GEO_Y", "GEO_LON", "GEO_LAT", "DISTRICT_ID", "PRECINCT_ID", "NEIGHBORHOOD_ID", "IS_CRIME", "IS_TRAFFIC"

"2018869789", "2018869789239900", "2399", "0", "theft-other", "larceny", "12/27/2018 3:58:00 PM", "", "12/27/2018 4:51:00 PM", "2681 N HANOVER CT", "3178210", "1700715", "-104.86615590", "39.75556140", "5", "512", "stapleton", "1", "0"

"2015664356", "2015664356544100", "5441", "0", "traffic-accident", "traffic-accident", "11/13/2015 7:45:00 AM", "", "11/13/2015 8:38:00 AM", "4100 BLOCK W COLFAX AVE", "3129148", "1694748", "-105.04075970", "39.73999120", "1", "122", "west-colfax", "0", "1"

"20176005213", "20176005213239901", "2399", "1", "theft-bicycle", "larceny", "6/8/2017 1:15:00 PM", "6/8/2017 5:15:00 PM", "6/12/2017 8:44:00 AM", "1705 17TH ST", "3140790", "1699792", "-104.99926360", "39.75366910", "6", "612", "union-station", "1", "0"

"20196012240", "20196012240230800", "2308", "0", "theft-from-bldg", "larceny", "12/7/2019 1:07:00 PM", "12/7/2019 6:30:00 PM", "12/9/2019 1:35:00 PM", "1350 N IRVING ST", "3132400", "1694088", "-105.02920820", "39.73813370", "1", "122", "west-colfax", "1", "0"
```

# Transforming the data

- We can look at the headers
  - "INCIDENT\_ID", "OFFENSE\_ID", "OFFENSE\_CODE", "OFFENSE\_CODE\_EXTENSION", "OFFENSE\_TYPE\_ID", "OFFENSE\_CATEGORY\_ID", "FIRST\_OCCURRENCE\_DATE", "LAST\_OCCURRENCE\_DATE", "REPORTED\_DATE", "INCIDENT\_ADDRESS", "GEO\_X", "GEO\_Y", "GEO\_LON", "GEO\_LAT", "DISTRICT\_ID", "PRECINCT\_ID", "NEIGHBORHOOD\_ID", "IS\_CRIME", "IS\_TRAFFIC"

# Transforming the data

- Notice what we could do:
  - The incidents are geo-coded
    - We can use this to overlay data points on a map of Denver
    - We can determine influence of neighborhood on crimes
    - We will wait with this until we understand matplotlib better
  - The incidents are time-stamped
    - We can look at the relationship between time and incidents



# Transforming the data

- Dealing with time stamps
  - Core Python has a datetime module that allows us to create and calculate with dates and times
  - Comes with a `strptime()` methods that needs a description of the format
    - This is because there are lot of different time formats
      - The am/pm format is one of the worst

# Transforming the data

- In our case:
  - The relevant time is the first time stamp in column 6
  - This took quite a while to get right

```
dati = datetime.strptime(contents[6], "%m/%d/%Y %I:%M:%S %p")
```

# Transforming the data

- We can also do so 'manually':

```
def translate(hour, ampm):  
    hour = int(hour)  
    if hour == 12 and ampm == 'AM':  
        return 0  
    if hour == 12 and ampm == 'PM':  
        return 12  
    if ampm == 'PM':  
        return hour  
    else:  
        return hour+12
```

# Transforming the data

```
def getdate(astring):
    astring = astring.strip('"')
    date, time, ap = astring.split()
    month, day, year = date.split('/')
    hour, minute, second = time.split(':')
    #print(year, month, day, hour, minute, second)

    return datetime(int(year),
                    int(month),
                    int(day),
                    translate(hour, ap),
```

# Selecting the Data

- Assume we only want to look at traffic problems in Denver
  - First question: When do traffic accidents happen?
    - Notice: Presumably winter weather (snow & ice) are important, so we should distinguish between winter month and the rest of the year
  - We look up the offense code and find that all traffic related incidents have codes between 15

# Selecting the Data

- Write down all time stamps of traffic related incidents

```
def select_vehicular():
    traffic = []
    with open('crime.csv') as infile:
        infile.readline()
        for line in infile:
            contents = line.split(',')
            dati = getdate(contents[6])
            code = int(contents[2].strip('"'))
            if 5400 <= code <= 5499:
                traffic.append(dati)
    return traffic
```

# Selecting the Data

- Determine number of accidents during a given hour at a given week-day
- A datetime object allows us access to the week-day and the hour

```
item.weekday()  
item.hour
```

- One is a method, the other a field
- Can use the Counter object in collections
  - Because there is no need to initialize a dictionary

# Selecting the Data

- Create seven counters
  - For each item (a datetime) in the traffic incident list:
    - Update the counter

```
def process_traffic(traffic):  
    weekdays = [Counter() for i in range(7)]  
    for item in traffic:  
        weekdays[item.weekday()][item.hour]+=1  
    return weekdays
```



# Displaying the Data

- From within Python, use matplotlib
  - Developed from matlab interface with many add-ons
  - `import matplotlib.pyplot as plt`
  - If you use IDLE, need to say `plt.show()` at the end
    - This will show all plot elements that you created

# Displaying the Data

- Pyplot supports many different types of graphs
  - We use mostly scatter and plot
  - There is even support for three-dimensions

# Displaying the Data

- Create a figure with `plt.figure( )`
- Create several line-plot elements using
  - `plt.plot(x-data, y-data)`
  - Can add a legend as a named parameter
  - But we need to place the legend then
    - `plt.legend(loc='upper left')`

# Displaying the Data

```
def show(counters):
    weekday = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
               'Friday', 'Saturday', 'Sunday']
    f = plt.figure()
    for i, counter in enumerate(counters):
        hours = [i for i in range(24)]
        numbers = [counter[i] for i in range(24)]
        s = plt.plot(hours, numbers, label=weekday[i])
    plt.legend(loc='upper left')
    plt.show()
```