

# Collisions and Traffic Workshop



Hello and welcome!

# What we'll be doing today

- Giving a brief overview of the datasets
- Doing some beginner data science things on the dataset
- Answering any questions/concerns

# San Diego Collisions and Traffic

What can we learn about San Diego  
transportation?



# The Two Datasets

1. Traffic Collisions within the city of San Diego
2. Traffic Volumes within the city of San Diego

# Load data into DataFrame

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

## Loading Data

```
collisions = pd.read_csv("pd_collisions_data.csv")
counts = pd.read_csv("traffic_counts_data.csv")
```

# Previewing the Data

```
collisions.head()
```

	report_id	date_time	police_beat	street_no	street_dir	street_name	street_type	cross_st_dir	cross_st_name	cross_st_type	violation_section	violation
0	170166	2017-01-01 00:01:00	124	8300		CAM DEL ORO					MISC-HAZ	
1	170218	2017-01-01 00:01:00	325	8100		ROYAL GORGE	DRIVE				22107	
2	170101	2017-01-01 00:01:00	322	6400		CRAWFORD	STREET				MISC-HAZ	

Information about these column names can be found at the same link where the datasets are found, entitled 'traffic collisions dictionary'

# What information do we have?

The collisions dataset comes with information about when the collision occurred, what police beat the collision occurred in, street number, name, and type, violation section, type, and charge description.

report_id	date_time	police_beat	street_no	street_dir	street_name	street_type	cross_st_dir	cross_st_name	cross_st_type	violation_section
0	170166 2017-01-01 00:01:00	124	8300		CAM DEL ORO					MISC-HAZ

violation_type	charge_desc	injured	killed	hit_run_lvl
VC	MISCELLANEOUS HAZARDOUS VIOLATIONS OF THE VEHI...	0	0	MISDEMEANOR

# What information do we have?

The volumes dataset contains info on the street name, number of cars going north, south, east, and west, the total count, and when the count of cars occurred.

	id	street_name	limits	all_count	northbound_count	southbound_count	eastbound_count	westbound_count	total_count	file_no	count_date
0	01AV018207	01 AV	A ST - ASH ST		18010	NaN	NaN	NaN	18010	0182- 07	2007-03-13 00:00:00



# First steps

The first step when working is to take a step back and think about the data. What information might you want to pull out of the provided data? What do you want to prove? What impacts could it have?

# Percentage of collisions resulting in injury

```
injured = sum(collisions['injured'] > 1)
```

```
number_of_collisions = len(collisions)
```

```
(injured/(1.0*number_of_collisions))*100
```

```
10.103800062369137
```

# Most Frequent Collision Causes

```
pd.value_counts(collisions['charge_desc'])
```

TURNING MOVEMENTS AND REQUIRED SIGNALS	5887
MISCELLANEOUS HAZARDOUS VIOLATIONS OF THE VEHICLE CODE	5136
VIOLATION OF BASIC SPEED LAW SPEED UNSAFE FOR CONDITIONS	2723
STARTING PARKED VEHICLES OR BACKING	1504
RED OR STOPVEHICLES STOP AT LIMIT LINE	1220
FOLLOWING TOO CLOSELY	845
LEFT TURN YIELD UNTIL SAFE OR U-TURN	648
YIELD RIGHT OF WAY TO PEDESTRIANS	470
ENTERING HWY FROM PRIVATE ROAD OR DRIVEWAY	428
ENTRANCE FROM STOP THROUGH HIGHWAYYIELD UNTIL REASONABLY SAFE	401
STOP REQUIREMENTS STATE STOP LINE	218
URNS:UNSAFE TURN AND/OR NO TURN SIGNAL (I)	176
PEDESTRIANS OUTSIDE CROSSWALKS	152
HIT AND RUN	148
LANED ROADWAYS - STRADDLING OR UNSAFE CHANGING	136
PEDESTRIAN NOT TO SUDDENLY ENTER PATH ETC	135
DUI ALCOHOL	112
RED ARROWDO NOT ENTER INTERSECTION	105
UNSAFE SPEED (BASIC SPEED LAW) (I)	90
LEFT TURN AT INTERSECTIONS	84

# Which time of day has most collisions?

1. Parse the date\_time format to only utilize time
2. Count the number of collisions for each hour of the day
3. Graph results in bar chart

# Parsing Date Time

date\_time

---

2017-01-01  
00:01:00

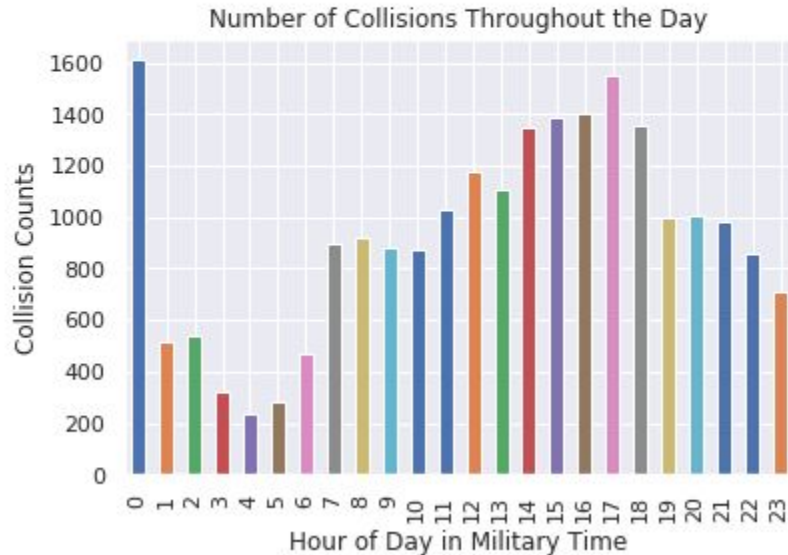
Need a way to parse date\_time to only get time

```
def grabtime(datetime):  
    return int(datetime[11:13])
```

```
crash_time=collisions['date_time'].apply(grabtime)
```

# Counting collisions and graphing

```
crash_time.value_counts().sort_index().plot(kind='bar')  
plt.title("Number of Collisions Throughout the Day")  
plt.xlabel("Hour of Day in Military Time")  
plt.ylabel("Collision Counts")  
plt.show()
```




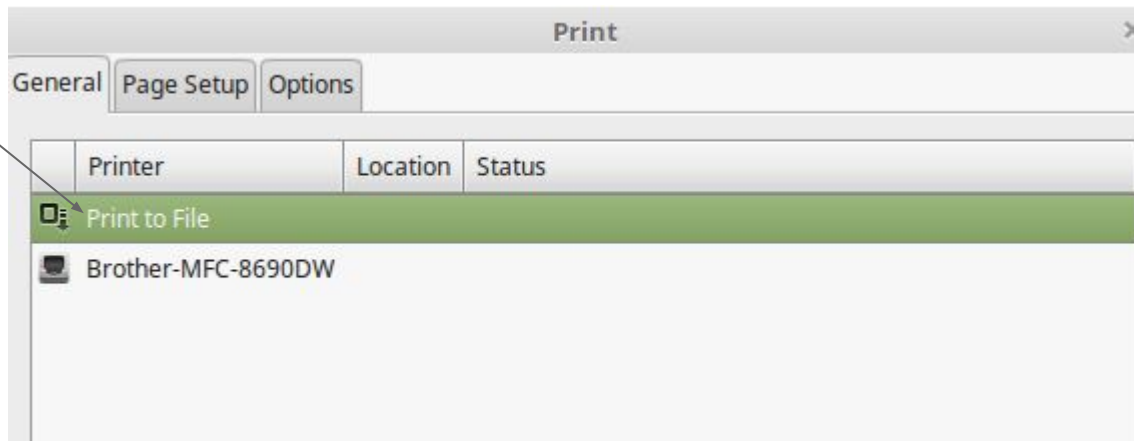
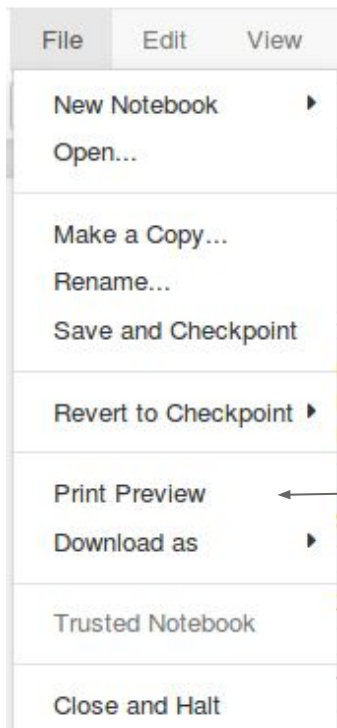
# Formatting submissions

- Make a Google Doc, Latex, Markdown, Jupyter Notebook and convert to pdf
- Jupyter Notebook lets you write markdown in cells



# Jupyter Notebook to pdf (cheating way)

 jupyter Sd La





# Working with Data

- Make sure you fully understand the data before you start working with it!
- If you want some extra supplemental data, like police beats, you are welcome to pull anything from <https://data.sandiego.gov/>.

# Good Luck!

- If you feel confident that you can begin working with the dataset, feel free to go begin working!
- There are also other workshops happening soon, like Omnisci's, so if you are interested in using their platforms make sure to check them out!
- If you still want some tutorials on Pandas or Python, feel free to stick around, the rest of this presentation will be giving some more in depth examples on how to use them.

# More Pandas!

- We're going to quickly cover some basics to Pandas in the remaining time we have.
- Tutorials will also be on the Github repo!
- Will cover things like filtering, transformations, selecting, etc.
- Lastly, Google is your friend! Could never survive Pandas without it.

# Selecting Data by column

```
collisions['street_name']  
collisions.street_name
```

These access information the exact same way, returns information in a Series.

0	CAM DEL ORO
1	CRAWFORD
2	ROYAL GORGE
3	VALERIO
4	11TH
5	A
6	RAMFOS
7	DE LA VALLE
8	SYDNEY
9	HORNBLEND
10	EMERALD
11	IMPERIAL
12	LAUREL
13	BLACK MTN
14	UNIVERSITY
15	MIRAMAR
16	INGRAHAM
17	MISSION BAY
18	EL CAJON
19	ASHFORD
20	GARNET
21	EL CAJON
22	IMPERIAL
23	MISSION CENTER
24	SAN DIEGO MISSION
25	SPORTS ARENA
26	WASHINGTON
27	EL CAJON
28	08TH
29	COLLEGE

...

# Selecting Multiple Columns

```
collisions[['report_id', 'street_name']]
```

You can select many columns to be output in a dataframe by passing in a list of column names to the selection brackets.

Also works with single names, i.e.:

```
[['street_name']]
```

	report_id	street_name
0	170166	CAM DEL ORO
1	170101	CRAWFORD
2	170218	ROYAL GORGE
3	170082	VALERIO
4	170097	11TH
5	170220	A
6	170153	RAMFOS
7	170035	DE LA VALLE
8	170044	SYDNEY
9	170042	HORNBLEND
10	170041	EMERALD
11	170139	IMPERIAL
12	170090	LAUREL
13	170073	BLACK MTN
14	170217	UNIVERSITY
15	170074	MIRAMAR
16	170118	INGRAHAM

# Filter Selections

```
collisions[collisions['injured'] > 1].head(3)
```

street_name	street_type	cross_st_dir	cross_st_name	cross_st_type	violation_section	violation_type	charge_desc	injured	killed	hit_run_lvl
SPORTS ARENA	BOULEVARD				22107	VC	TURNING MOVEMENTS AND REQUIRED SIGNALS	4	0	NaN
EL CAJON	BOULEVARD				21801A	VC	LEFT TURN YIELD UNTIL SAFE OR U- TURN	2	0	NaN
COLLEGE	AVENUE				21804	VC	ENTERING HWY FROM PRIVATE ROAD OR DRIVEWAY	3	0	NaN

You can filter out data by including a conditional in your selection. In this case, we only want to only include columns where the injured value is more than 1. You can have multiple conditions by surrounding each conditional with ( ) and separating them with either and & or or |.

# How does this filtering work?

```
collisions[ 'injured' ] > 1
```

The code shown generates a boolean array that can be used to filter out any rows where “False” appears.

0	False
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False
10	False
11	False
12	False
13	False
14	False
15	False
16	False
17	False
18	False
--	--

# Using .iloc

```
collisions.iloc[1:3, 2:5]
```

	police_beat	street_no	street_dir
1	322	6400	
2	325	8100	

.iloc is used for selecting rows and columns based on index. The first parameter is which rows you want, followed by the column indices.



```
collisions.loc[1:3, "police_beat":"street_dir"]
```

## Using .loc

	police_beat	street_no	street_dir
1	322	6400	
2	325	8100	
3	935	5500	

.loc is used for selecting rows and columns based on index or column name. It's similar to .iloc but you can use column names or indices.

# Dropping columns

```
collisions.drop('police_beat', axis=1)
```

You can drop columns using the drop command. Make sure to specify axis=1. Axis can either be 0 or 1, or along columns or along rows.

# Find unique values in a column

```
collisions['street_type'].unique()
```

```
array([' ', 'STREET', 'DRIVE', 'TRAIL', 'AVENUE', 'PLACE', 'COURT',  
      'ROAD', 'BOULEVARD', 'PARKWAY', 'WAY', 'COVE', 'CIRCLE', 'LANE',  
      'HIGHWAY', 'TERRACE', 'PLAZA', 'MALL', 'ROW', 'POINT', 'WALK',  
      'PASEO', 'SQUARE', 'RAMP', 'DRIVEWAY', 'KNOLLS', nan, 'LIGHTS'],  
      dtype=object)
```

You can use `.unique` on a column or series to see the unique values.

# Grouping

Grouping is a powerful tool, and can give you insights into how different values for columns could affect an outcome for another.

You can pick or design a function that aggregates data when grouping.

```
collisions.groupby('street_type').mean()
```

	police_beat	street_no	injured	killed
street_type				
	490.713604	5227.621321	0.640414	0.007955
<b>AVENUE</b>	519.518253	3354.103436	0.595025	0.004474
<b>BOULEVARD</b>	513.738517	4440.148282	0.732369	0.007233
<b>CIRCLE</b>	578.274725	2264.835165	0.428571	0.000000
<b>COURT</b>	415.598802	6344.311377	0.239521	0.000000

# Another grouping example

You can see that group only applies the aggregation function to columns that will work with it.

You can also see that it sometimes gets confused and produces results that don't make too much sense, like summing the street numbers.

```
grouped = collisions.groupby('violation_type').sum()  
grouped
```

	police_beat	street_no	injured	killed
violation_type				
CO	2107	16000	5	0
HS	468	9900	1	0
MC	7103	26700	10	0
PC	1304	12300	1	0
VC	10758172	96644170	13580	114
ZZ	5504	68500	4	1

# Grouping continued

You can use `reset_index` to put the grouped values back into the dataset.

```
grouped.reset_index()
```

	violation_type	police_beat	street_no	injured	killed
0	CO	2107	16000	5	0
1	HS	468	9900	1	0
2	MC	7103	26700	10	0
3	PC	1304	12300	1	0
4	VC	10758172	96644170	13580	114
5	ZZ	5504	68500	4	1

## Note about making these changes.

Pandas methods are not usually in place, meaning that when you call a method the changes are not saved unless you explicitly tell Python that you'd like them to be.

This is done something along the lines as seen previously, like:

```
grouped = collisions.groupby("{column name}").
```

Just make sure to save changes you make to avoid confusion.

# Apply

One of the next best tools in your arsenal is `apply`. This can be used to apply a specific method over each row or column. This is handy as you can have control over what you do with each column.



# Using apply

```
def edit_row(row):  
    if row['killed'] > 0:  
        return 'fatal'  
    return 'not fatal'  
  
collisions['fatal'] = collisions.apply(edit_row, axis=1)  
collisions.head()
```

You can apply a method to either rows or columns with axis. When applying, you will return a series of new information based on your return values. This series can be used to create a new column.

	killed	hit_run_lvl	fatal
0	0	MISDEMEANOR	not fatal
0	0	MISDEMEANOR	not fatal
0	0	MISDEMEANOR	not fatal
0	0	MISDEMEANOR	not fatal
0	0	MISDEMEANOR	not fatal

# More in depth with apply

What is happening with apply when we call it? Let's have a look. We'll start by just getting one row of data to work with so we can see what happens during our function calls.

```
mini_row = collisions.head(1)
mini_row
```

	report_id	date_time	police_beat	street_no	street_dir	street_name	street_type	cross_st_dir	cross_st_name	cross_st_type	violation_section	violation_type
0	170166	2017-01-01 00:01:00	124	8300		CAM DEL ORO					MISC-HAZ	VC

# Apply, axis=1

We can see here that axis=1 works with the row, so we have one print statement that contains all the information in the row.

This also means you can index the input of the lambda function, x, with the column names.

```
mini_row.apply(lambda x: print(x), axis=1)
```

```
report_id                170166
date_time                2017-01-01 00:01:00
police_beat              124
street_no                8300
street_dir
street_name              CAM DEL ORO
street_type
cross_st_dir
cross_st_name
cross_st_type
violation_section        MISC-HAZ
violation_type            VC
charge_desc              MISCELLANEOUS HAZARDOUS VIOLATIONS OF THE VEHI...
injured                  0
killed                   0
hit_run_lvl              MISDEMEANOR
fatal                    not fatal
Name: 0, dtype: object
```

# Apply, axis=0

Alternatively, for axis=0, you can see we go column by column. We get a print statement for each, so anything you do to the input x in this case would apply to the entire column you are currently working with.

```
: mini_row.apply(lambda x: print(x), axis=0)
0    170166
Name: report_id, dtype: object
0    2017-01-01 00:01:00
Name: date_time, dtype: object
0    124
Name: police_beat, dtype: object
0    8300
Name: street_no, dtype: object
0
Name: street_dir, dtype: object
0    CAM DEL ORO
Name: street_name, dtype: object
0
Name: street_type, dtype: object
0
Name: cross_st_dir, dtype: object
0
Name: cross_st_name, dtype: object
0
Name: cross_st_type, dtype: object
0    MISC-HAZ
Name: violation_section, dtype: object
0    VC
Name: violation_type, dtype: object
0    MISCELLANEOUS HAZARDOUS VIOLATIONS OF THE VEHI...
Name: charge_desc, dtype: object
0    0
Name: injured, dtype: object
0    0
Name: killed, dtype: object
0    MISDEMEANOR
Name: hit_run_lvl, dtype: object
0    not fatal
Name: fatal, dtype: object
```

# For loops vs Apply

You may be wondering, why should I use apply when I could just use a for loop? You should avoid using for loops whenever possible when working with data like this. Methods like apply have been optimized, whereas for may take longer, so try to use built in methods whenever possible.

# Graphing

There are many ways you can represent your data. Graphs and plots can be the most clear way of presenting your points with visual aids, so making sure to include some is very important.

# Graphing

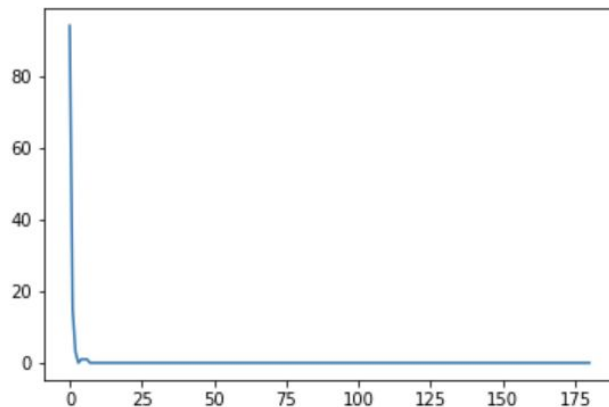
In this situation, I've grouped by the injured column and summed the results. We can plot the total number of people killed in an accident by the number of people injured.

```
grouped = collisions.groupby('injured').sum()  
grouped.head()
```

	police_beat	street_no	killed
<b>injured</b>			
0	6052975	53616090	94
1	3674374	32125320	15
2	741354	7841250	3
3	212218	2234400	0
4	66336	679110	1

```
plt.plot(grouped.index, grouped.killed)
```

```
[<matplotlib.lines.Line2D at 0x12cb71048>]
```



# Graphing continued

There are many kinds of graphs to work with. Bar charts, horizontal and vertical, line plots, scatter plots, and more. Do some research to decide which may be the best to work with depending on what you may be trying to argue.



# Datetime

```
collisions['new_time'] = collisions['date_time'].apply(pd.to_datetime)  
collisions['new_time']
```

0	2017-01-01	00:01:00
1	2017-01-01	00:01:00
2	2017-01-01	00:01:00
3	2017-01-01	00:01:00
4	2017-01-01	01:00:00
5	2017-01-01	01:00:00
6	2017-01-01	01:18:00
7	2017-01-01	01:53:00
8	2017-01-01	01:58:00
9	2017-01-01	02:00:00
10	2017-01-01	02:00:00
11	2017-01-01	02:03:00
12	2017-01-01	04:10:00
13	2017-01-01	04:43:00
14	2017-01-01	05:26:00
--	--	--

Datetime is another important tool. You can use this to conduct time series analysis. Your information starts off as a string, but you can convert it to a datetime object which allows you to properly compare times, as well as more easily pull things like hours and seconds out of it.

# Wrapping up

- Any questions?
- Anything you're wondering if Pandas has the capability to do?
- Still wondering where to start?

Talk to us and let us know! Communicate with your team and decide which direction you may want to go in! There will be mentors about and Piazza available for any questions you may have.

**Good luck today!**