

# Car Accidents

LIOR TURGEMAN 304941412

DAN AVERIN 204358394



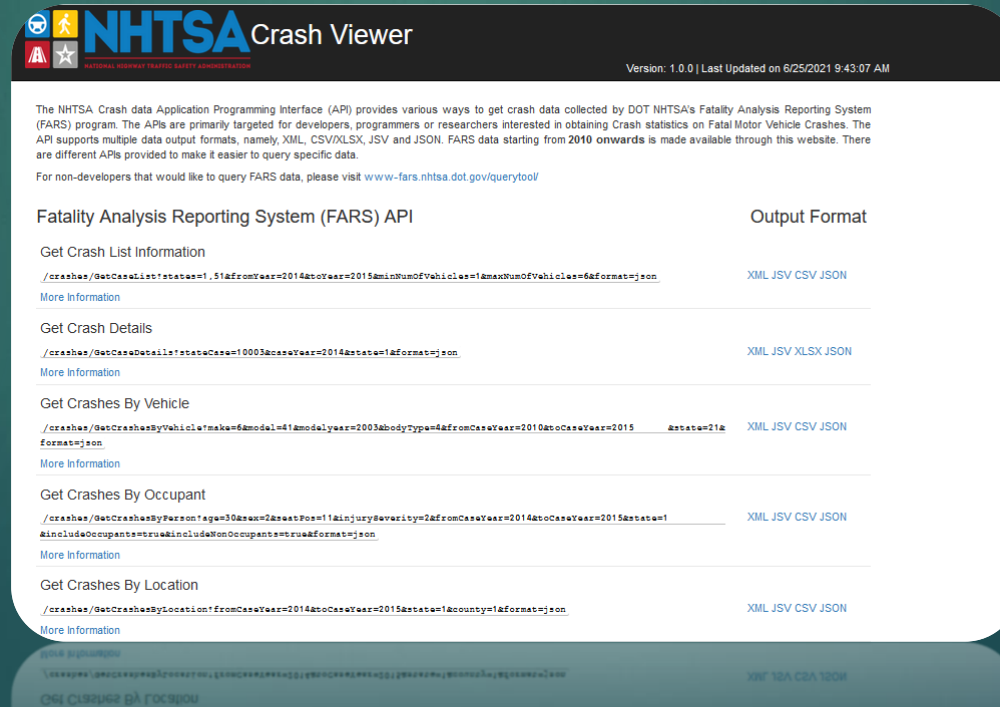
# Project Overview and Research Question

- Is there a connection between the amount of car accidents to the weather conditions occurred at the time of the accident?
- Is there a connection between the weather and the type of accidents happened (In intersection, what type of Objects were harmful in the collision...)
- Is there a connection between car accidents and the time/day/night conditions?
- In which states more car accidents are likely to occur?



# API & Sources

- We chose to use information from NHTSA(National Highway Traffic Safety Administration) - agency of the U.S. federal government, part of the Department of Transportation.  
<https://crashviewer.nhtsa.dot.gov/CrashAPI>
- We used the API to retrieve CSV data files from years 2010 up to 2019.













The screenshot displays the NHTSA Crash Viewer API documentation. At the top, the NHTSA logo and 'Crash Viewer' title are visible, along with the version '1.0.0' and a last update timestamp of '6/25/2021 9:43:07 AM'. The main content area explains that the API provides various ways to get crash data collected by DOT NHTSA's Fatality Analysis Reporting System (FARS) program. It lists supported output formats (XML, CSV/XLSX, JSV, and JSON) and mentions that FARS data starting from 2010 onwards is available. A link is provided for non-developers to query FARS data: [www-fars.nhtsa.dot.gov/querytool/](http://www-fars.nhtsa.dot.gov/querytool/).

Fatality Analysis Reporting System (FARS) API	Output Format
<b>Get Crash List Information</b> URL: <a href="#">/crashes/getCrashList?state=1,51&amp;fromYear=2014&amp;toYear=2015&amp;minNumOfVehicles=1&amp;maxNumOfVehicles=6&amp;format=json</a> <a href="#">More Information</a>	XML JSV CSV JSON
<b>Get Crash Details</b> URL: <a href="#">/crashes/getCrashDetails?stateCase=10003&amp;caseYear=2014&amp;state=1&amp;format=json</a> <a href="#">More Information</a>	XML JSV XLSX JSON
<b>Get Crashes By Vehicle</b> URL: <a href="#">/crashes/getCrashesByVehicle?make=6&amp;model=41&amp;modelYear=2003&amp;bodyType=4&amp;fromCaseYear=2010&amp;toCaseYear=2015&amp;state=21&amp;format=json</a> <a href="#">More Information</a>	XML JSV CSV JSON
<b>Get Crashes By Occupant</b> URL: <a href="#">/crashes/getCrashesByPerson?age=30&amp;sex=2&amp;seatPos=1&amp;injurySeverity=2&amp;fromCaseYear=2014&amp;toCaseYear=2015&amp;state=1&amp;includeOccupants=true&amp;includeNonOccupants=true&amp;format=json</a> <a href="#">More Information</a>	XML JSV CSV JSON
<b>Get Crashes By Location</b> URL: <a href="#">/crashes/getCrashesByLocation?fromCaseYear=2014&amp;toCaseYear=2015&amp;state=1&amp;county=1&amp;format=json</a> <a href="#">More Information</a>	XML JSV CSV JSON

# API & Sources

- The following NHTSA API - <https://crashviewer.nhtsa.dot.gov/CrashAPI>
- There is a lot of API options, and we chose to Get FARS Data By Year with the richest variety of information.
- The data pulled using the following API requests:
- <https://crashviewer.nhtsa.dot.gov/CrashAPI/FARSDData/GetFARSDData?dataset=Accident&caseYear=X&format=csv>
  - X – represents every year from 2010 up to 2019
- We combined and created new big CSV File with all the data combined.

```
def download_api_files():  
    """  
    This function downloads the data files in csv format from NHTSA API, from years 2010 to 2019  
    """  
    try:  
        os.mkdir('csv_files/')  
    except Exception as e:  
        print("Folder already exists.")  
  
    for year in range(10):  
        url = 'https://crashviewer.nhtsa.dot.gov/CrashAPI/FARSDData/GetFARSDData?dataset=Accident&caseYear=201{}&format=csv'.format(year)  
        r = requests.get(url, allow_redirects=True)  
        file_name = 'csv_files/Accidents_201{}.csv'.format(year)  
        open(file_name, 'wb').write(r.content)
```

Accidents\_2010.csv   
Accidents\_2011.csv   
Accidents\_2012.csv   
Accidents\_2013.csv   
Accidents\_2014.csv   
Accidents\_2015.csv   
Accidents\_2016.csv   
Accidents\_2017.csv   
Accidents\_2018.csv   
Accidents\_2019.csv 

# Data Handling and Cleaning

- In our project we used data from the years 2010 up to 2019.
- Each year had close to 35,000 records, eventually we got 323,480 records as result of 10 years of car accidents data from the USA.
- We omitted irrelevant data from the original csv files, combined all data to one file which will be used to research the question we presented.

```
def clear_undefined_values(df):  
    """  
    :param: df - Dataframe  
    This function clears rows from data frame according to consts.clean_dictionary  
    """  
    for key in consts.clean_dictionary:  
        for clean_value in consts.clean_dictionary[key]:  
            df.drop(df.index[df[key] == clean_value], inplace=True)
```

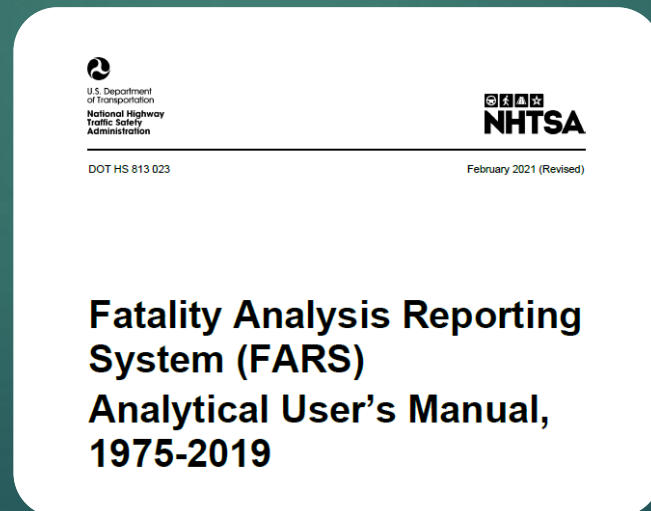
```
clean_dictionary = {"latitude": [77.7777, 88.8888, 99.9999],  
                    "harm_ev": [91, 93, 98, 99],  
                    "man_coll": [98, 99],  
                    "reljct1": [8, 9],  
                    "reljct2": [98, 99],  
                    "typ_int": [98, 99],  
                    "weather": [98, 99],  
                    "lgt_cond": [8, 9],  
                    "hour": [99]} # dictionary of column names with undefined values to be cleared from the data
```

	caseyear	state	peds	persons	...	lgt_cond	weather	fatals	drunk_dr
0	2010	1	0	2	...	3	1	1	1
1	2010	1	0	1	...	3	1	1	0
2	2010	1	0	3	...	1	1	1	0
3	2010	1	0	2	...	3	2	1	0
4	2010	1	0	2	...	4	1	1	0
...	...	...	...	...	...	...	...	...	...
289955	2018	56	0	1	...	2	1	1	1
289956	2018	56	0	2	...	1	3	1	0
289957	2018	56	0	1	...	2	11	1	0
289958	2018	56	0	2	...	1	1	1	1
289959	2018	56	0	2	...	5	1	1	1



# EDA

- As we checked our data, we cleared irrelevant information regarding our research questions.
- FARS provides PDF File with information regarding each column, under the filename: “Fatality Analysis Reporting System (FARS) Analytical User’s Manual, 1975-2019.PDF”  
The file is attached in the project under documents folder.
- We defined a List with all irrelevant columns to be removed from The CSV data files, According to the PDF file.



# EDA

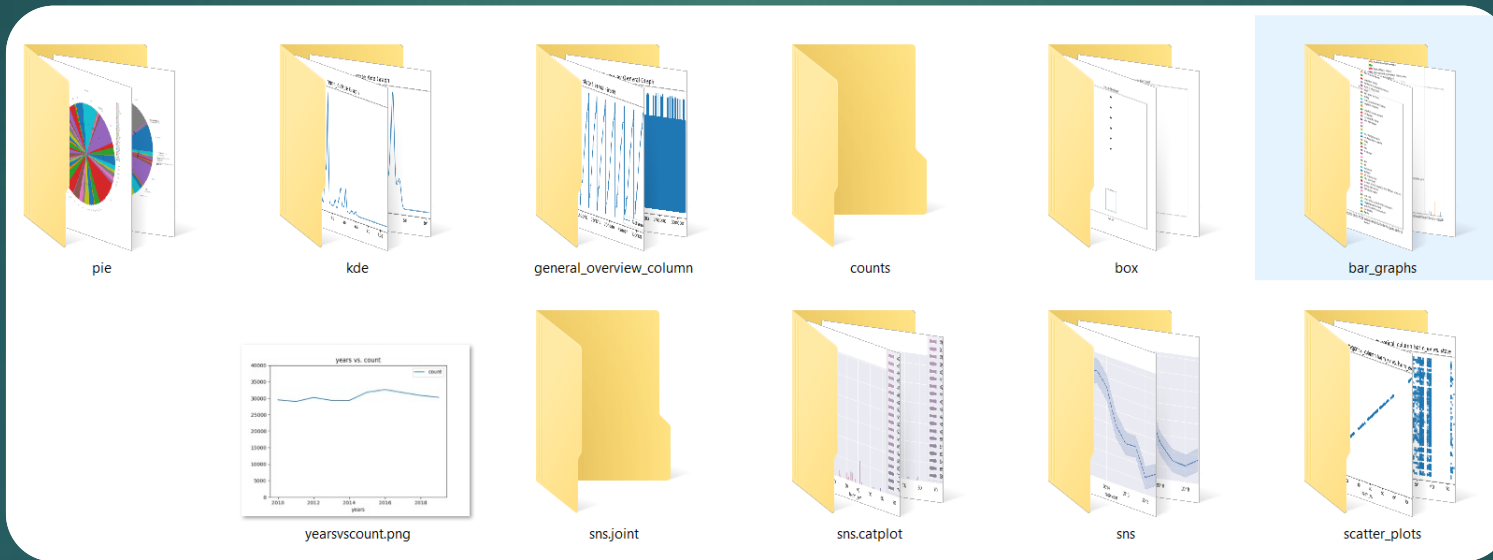
- Almost any column appears twice, first column as text, and secondly as Nominal number.
- For getting more reliable data we cleared from every column unknown values, and columns that irrelevant data for our research question
- For example:

PDF Column explanation			Reason omitted
Consecutive Number	ST_CASE	33	File consecutive number
Number of Vehicle Forms Submitted- ALL	VE_TOTAL	39	Our research question does not refers to forms.

```
drop_columns = ['st_case',  
                'statename',  
                've_total',  
                've_forms',  
                'pvh_invl',  
                'pernotmvit',  
                'permvit',  
                'county',  
                'countynome',  
                'city',  
                'cityname',  
                'monthname',  
                'day_week',  
                'day_weekname',  
                'hour',  
                'hourname',  
                'minute',
```

# Graphs

- We Used Pandas, Seaborn, Matplotlib to generate different variety of graphs according to our data.

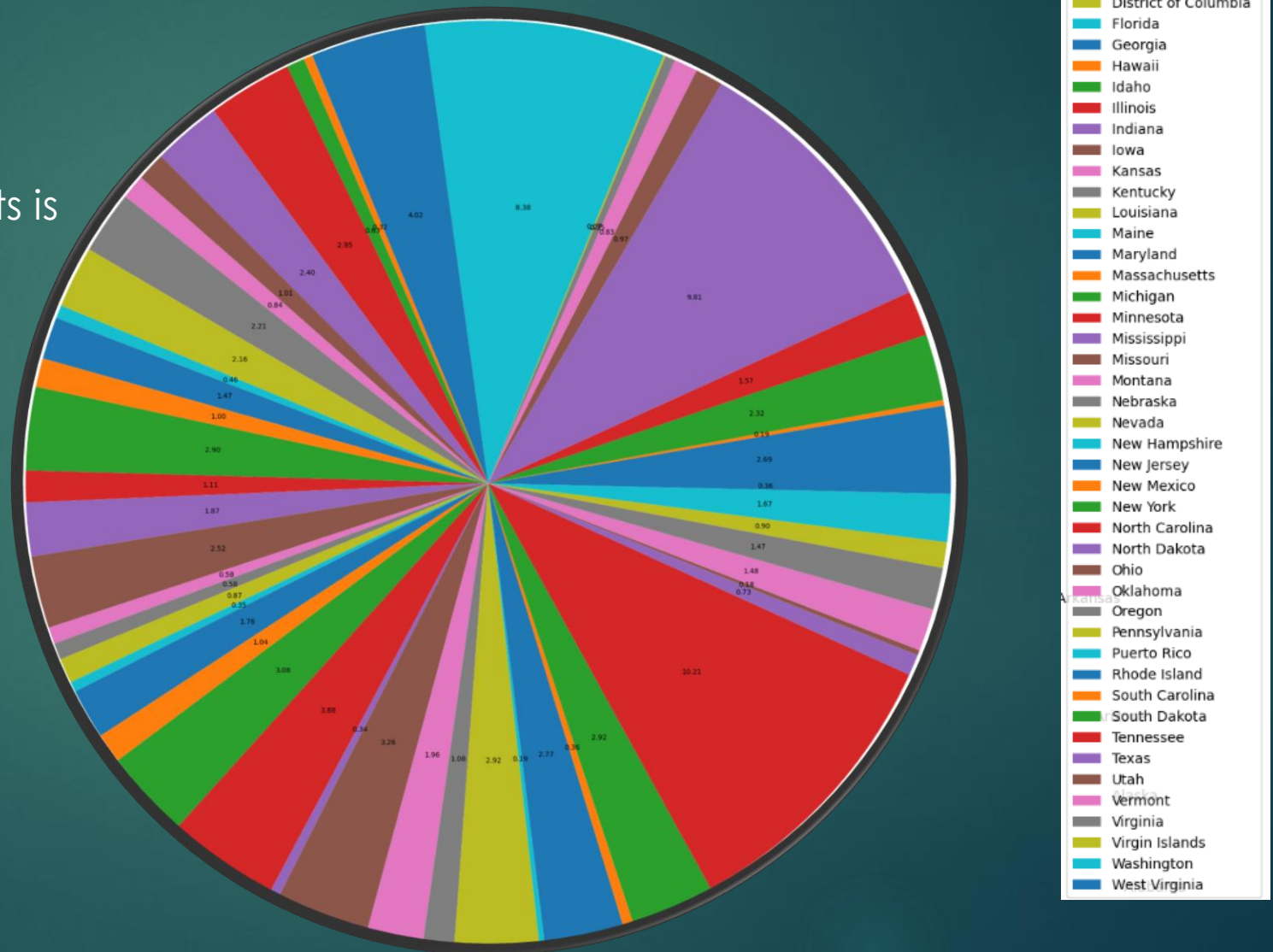




# Graphs

## Amount of Accidents Per State:

- We can see that the most common states with accidents is Tennessee and the Florida.

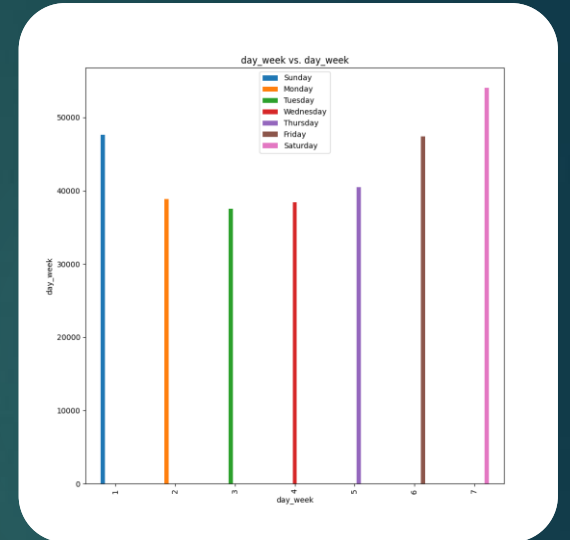


# Graphs



## Amount of Accidents Per Day:

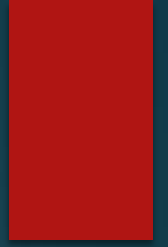
- Most accidents occur during the weekend (Friday Saturday and Sunday) - 48.98% of all accidents



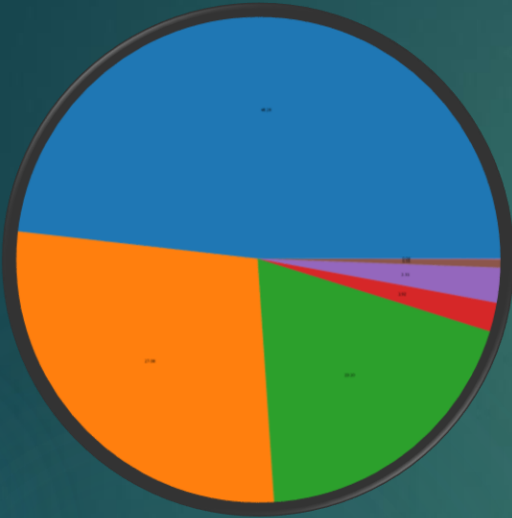
## Amount of Accidents Per Hour:

- Most accidents occur between 17:00 to 19:00.

# Graphs

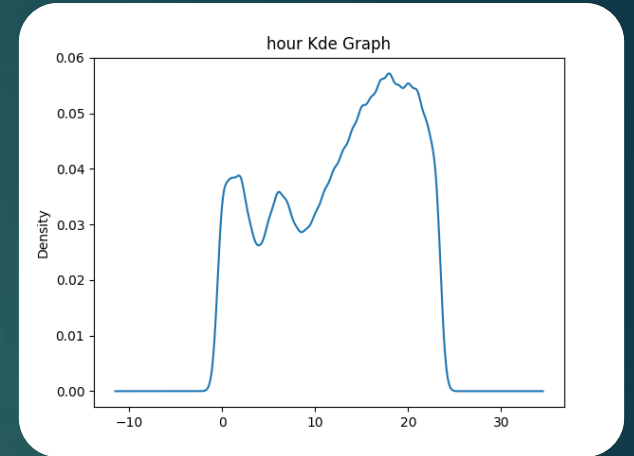


- daylight
- dark-not lighted
- dark-lighted
- dawn
- dust
- dark - unknown
- other



## Amount of Accidents – Light Conditions:

- Most accidents occur during daylight - 48%



- 0
- 1
- 2
- 3
- 4

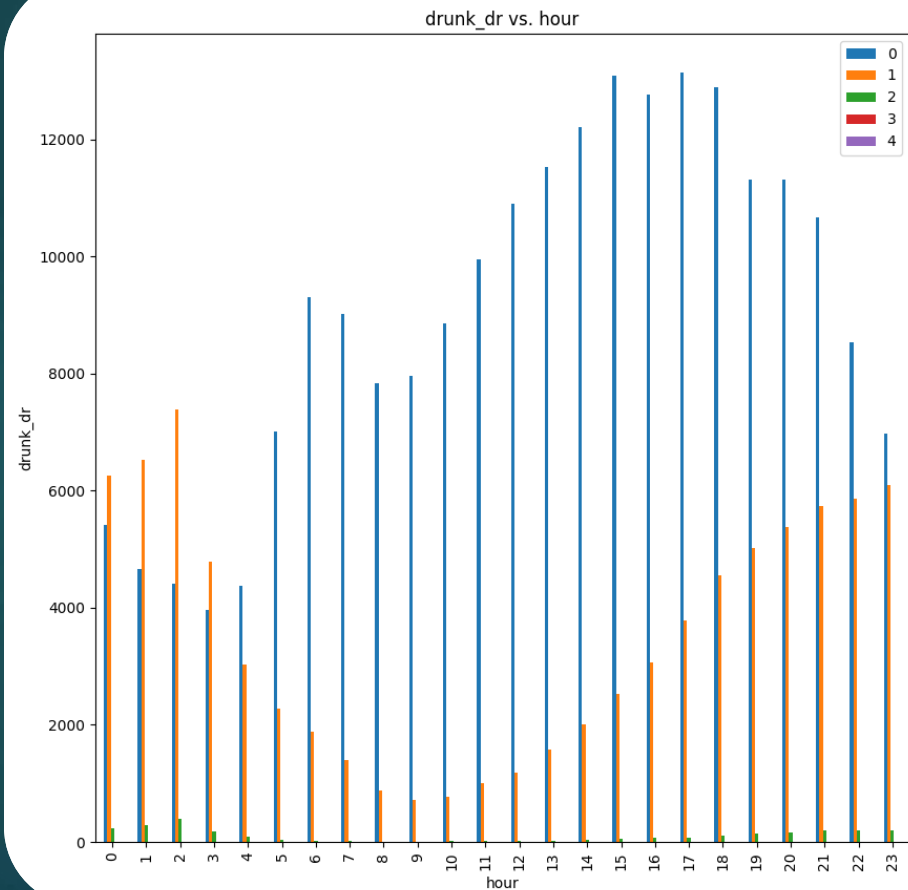


## Amount of Accidents – Drunk Drivers Involved:

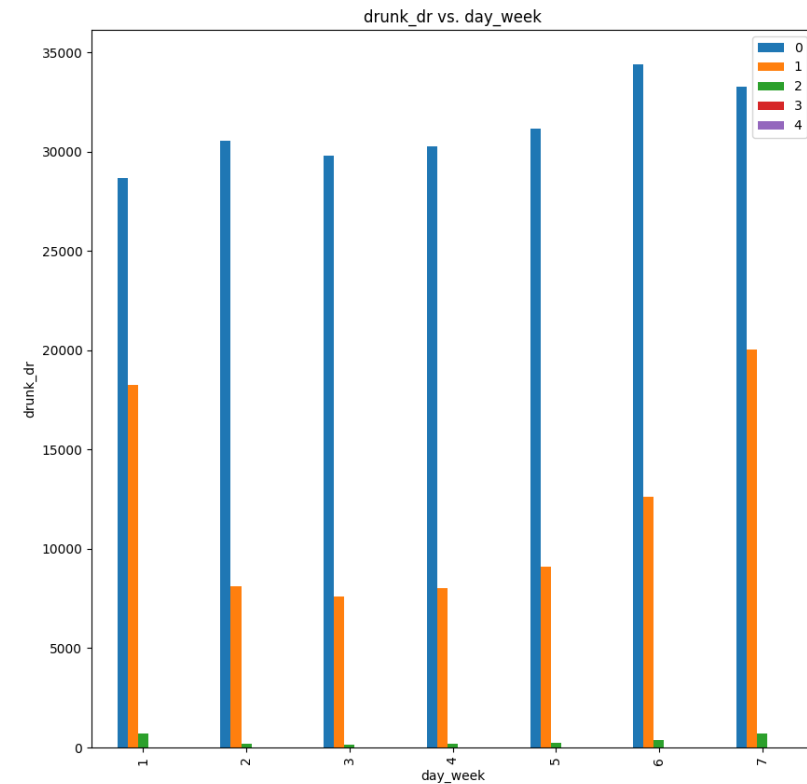
- 71% of Accidents – 0 Drunk drivers.
- 29% of Accidents involved at least 1 drunk driver.

# Graphs

Accidents with drunk drivers mostly occur at night, when the peak hour is 2 am.

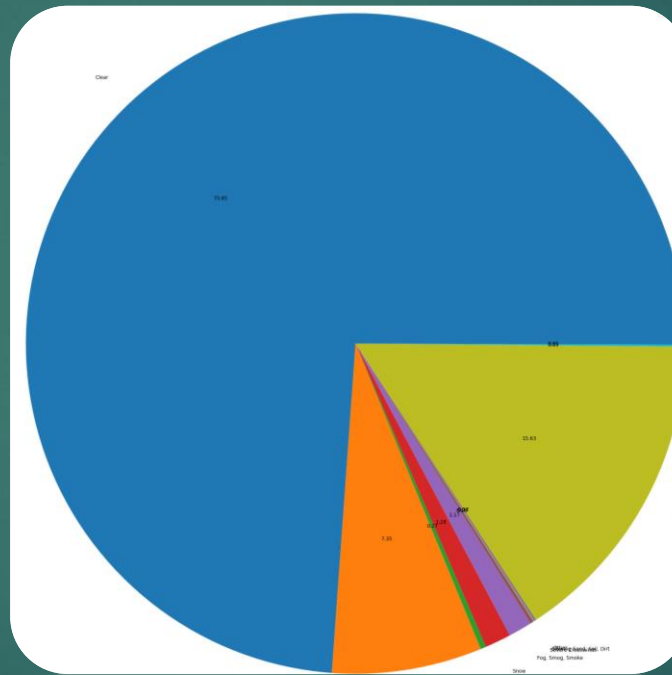
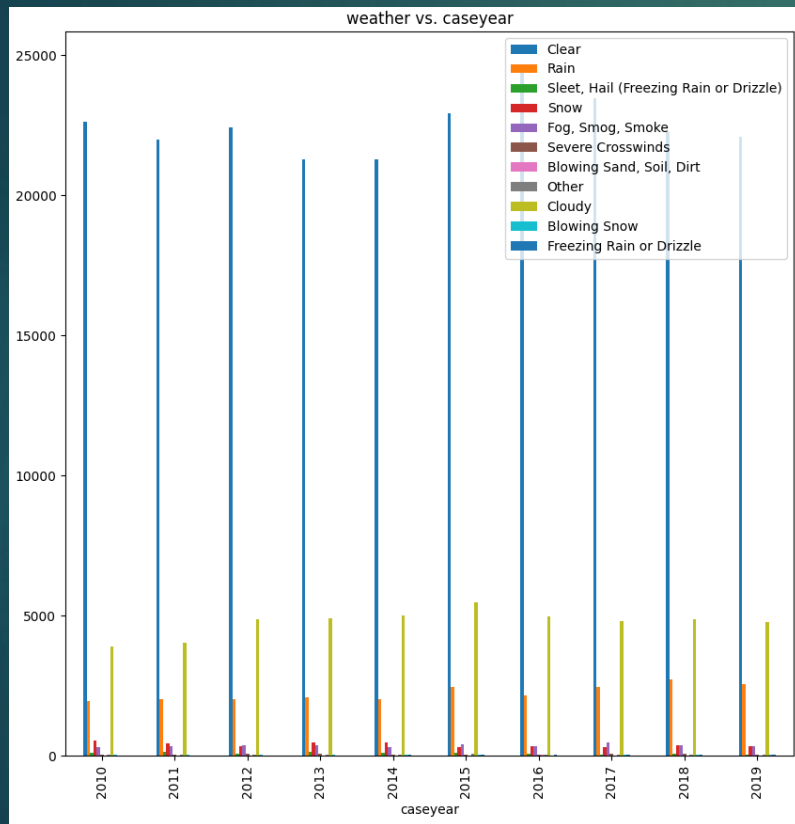


At weekends, accidents involve more drunk drivers(Friday, Saturday and Sunday).



# Graphs

- Is there a connection between the amount of car accidents to the weather conditions occurred at the time of the accident?

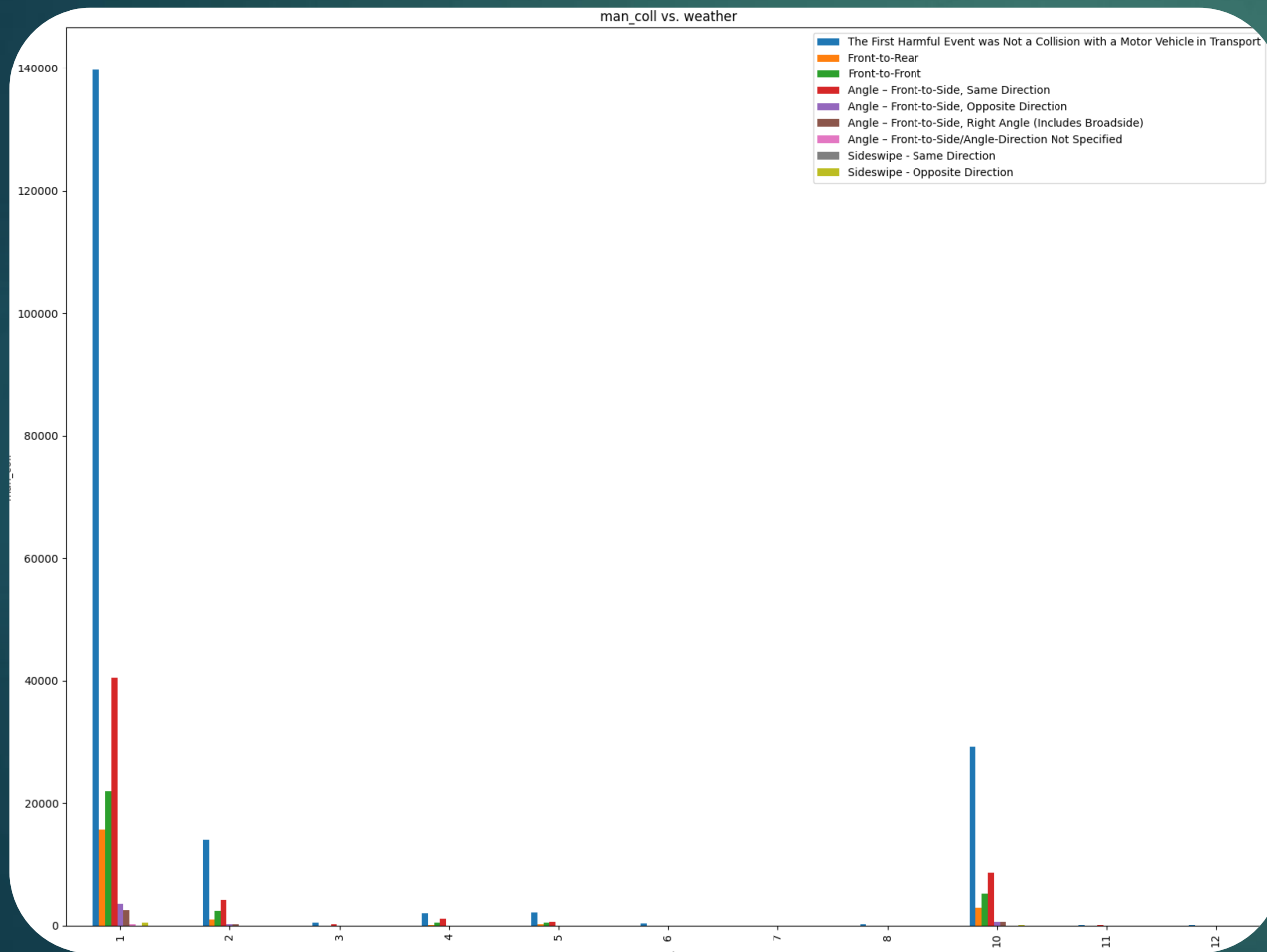


As you can see from the data, most of the accidents occur during clear weather.



# Graphs

- Is there a connection between the weather and the type of accidents happened ?



- About 50% of all accidents that occur during clear weather, not include collisions between another vehicle.
- The second common type of accidents are angle collisions.

- Clear
- Rain
- Sleet, Hail (Freezing Rain or Drizzle)
- Snow
- Fog, Smog, Smoke
- Severe Crosswinds
- Blowing Sand, Soil, Dirt
- Other
- Cloudy
- Blowing Snow
- Freezing Rain or Drizzle

# Machine Learning

- As our data is mainly categorical, we decided to use and implement Decision Tree model as our Machine Learning and prediction.
- Our code generates the decision tree and tests according to the following steps:
  1. We split the data:
    - x will hold the columns which are parameters used for the prediction.
    - y holds the prediction we wanted to know.

Our data is being sent train\_test\_split function

```
###Splitting train/test data
x=data[columns1]
y=data[column2]
X_tr, X_ts, y_tr, y_ts = tts(x, y, test_size=20/100, random_state=None)
```

- X\_tr and Y\_tr are used by the model for training.
- X\_ts and Y\_ts are used for testing.
- We chose to set the test size to 20%, which means 80% will be used for training, and 20% for testing.

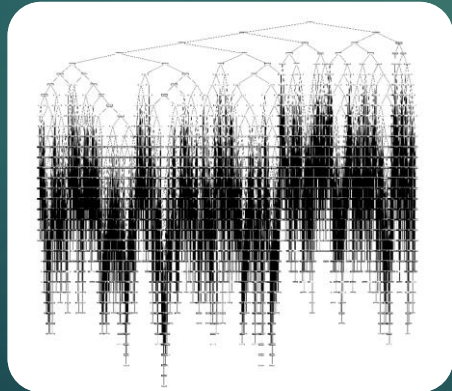
# Machine Learning

- Our code generates the Decision Tree as an output:

```
tree.plot_tree(decisionTree)
fig.savefig('machine_learning_trees/decision_tree_{}_to_{}.png'.format(columns1, column2))
```

- Here is an example of the decision tree our code generates for the following columns slicing:
- Can your predict the fatality rate according to the following data: Day, Month, State, Weather and Light Conditions?

```
machine_learning.create_prediction(data, ['day','month', 'state', 'weather','lgt_cond'], 'fatals')
```



Click To Open File:



decision\_tree\_['day', 'month', 'state', 'weather', 'lgt\_cond']\_to\_fatals.png

# Machine Learning

- In the following steps we use fit() to train our model, with X\_tr and Y\_tr.

```
###Training the Model
decisionTree.fit(X_tr,y_tr)
```

- We use predict() with 20% of the data (X\_ts and Y\_ts) to test the data:

```
###Making Predictions
y_pr=decisionTree.predict(X_ts)
###Evaluating Prediction Accuracy
print("Your Model Accuracy is %:", metrics.accuracy_score(y_ts, y_pr)*100)
```

- As a result of our requested prediction: [Day, Month, State, Weather and Light Conditions] against [Fatality], we got the following results:

```
Your Model Accuracy is %: 90.8464001313521
```

# Machine Learning

- We added auto prediction option to test different inputs of data:

```
###Making Prediction with Foreign Data
# while loop to make predictions:
while True:
    print("Do you want to predict? (Y/N)")
    answer = input()
    if answer == 'Y':
        answer_vector = []
        print("Please enter your data for predition:\n "
              "you should enter in the following format: {}".format(columns1))
        for i in range(len(columns1)):
            a = input()
            answer_vector.append(a)

        predictions = decisionTree.predict([answer_vector]) # send predictions

        print("-----\n Your prediction is: {}\n-----".format(predictions))
    else:
        break # Kill loop.
```

```
Please enter your data for predition:
you should enter in the following format: ['day', 'month', 'state', 'weather', 'lgt_cond']
15
12
40
1
1
-----
Your prediction is: [1]
-----
```



# Code Explanations and Project Structure

- main.py

Imports:

```
import process_data_helper as helper
import graph_generator
import pandas as pd
import const_variables as consts
import machine_learning
```

Getting, Handling and Cleaning the Data:

```
# 1. Handle Data:
helper.download_api_files() # download the files from the API
helper.create_merged_file() # creates clean csv file
data = pd.read_csv(consts.data_file_path) # open Dataframe with combined data
helper.clear_undefined_values(data) # clean empty or unnecessary values
data.to_csv("csv_files/result_after_cleaning.csv") # save the cleaned data to csv
```

Executing AI Machine Learning Process:

```
### 3. Machine Learning:
# Can your predict the fatality rate according to the following data: Day, Month, State, Weather and Light Conditions?
machine_learning.create_prediction(data, ['day', 'month', 'state', 'weather', 'lgt_cond'], 'fatals')

# Can your predict the First Harmful event to happend,
# according to the following data: Month, State, Weather and Light Conditions?
machine_learning.create_prediction(data, ['month', 'state', 'weather', 'lgt_cond'], 'harm_ev')
```

Generating Graphs According to Data:

```
# 2. Graph Generator:

for key in consts.columns_description.keys(): # this code generates all pie graphs
    graph_generator.pie_graph(data, key)

for key in consts.columns_description.keys(): # this code generates all pie graphs
    graph_generator.general_overview_column(data, key)

for key in consts.columns_description.keys(): # this code generates all pie graphs
    graph_generator.box_graph_column(data, key)

for key in consts.columns_description.keys():...

for key in consts.columns_description.keys():...
```

# Code Explanations and Project Structure

- The project uses Following files:
  - main.py – Hold the main code that runs and uses the below python files.
  - const\_variables.py – Holds constant variables, lists and dictionary's for various parameters in our data, we defined the variables using “Fatality Analysis Reporting System (FARS) Analytical User's Manual, 1975-2019.PDF”.
  - graph\_generator.py – Handles all function for different graph generation
  - machine\_learning.py – Holds Code for Machine Learning AI Section
  - process\_data\_helper.py – Holds Function for data cleaning and Data Merging.

