Car Accidents

LIOR TURGEMAN 304941412

DAN AVERIN 204358394

Project Overview and Research Question

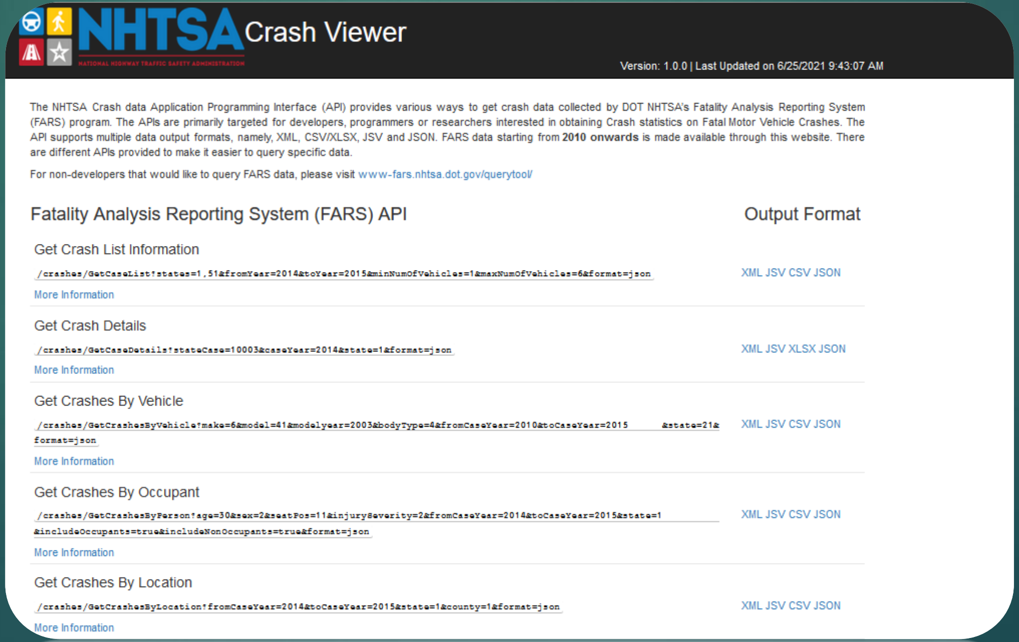
- Is there a connection between the amount of car accidents to the weather conditions occurred at the time of the accident?
- Is there a connection between the weather and the type of accidents happened (In intersection, what type of Objects were harmful in the collision…)

- 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.



NHTSA Crash Viewer

Version: 1.0.0 | Last Updated on 6/25/2021 9:43:07 AM

The NHTSA Crash data Application Programming Interface (API) provides various ways to get crash data collected by DOT NHTSA's Fatality Analysis Reporting System (FARS) program. The APIs are primarily targeted for developers, programmers or researchers interested in obtaining Crash statistics on Fatal Motor Vehicle Crashes. The API supports multiple data output formats, namely, XML, CSV/XLSX, JSV and JSON. FARS data starting from **2010 onwards** is made available through this website. There are different APIs provided to make it easier to query specific data.

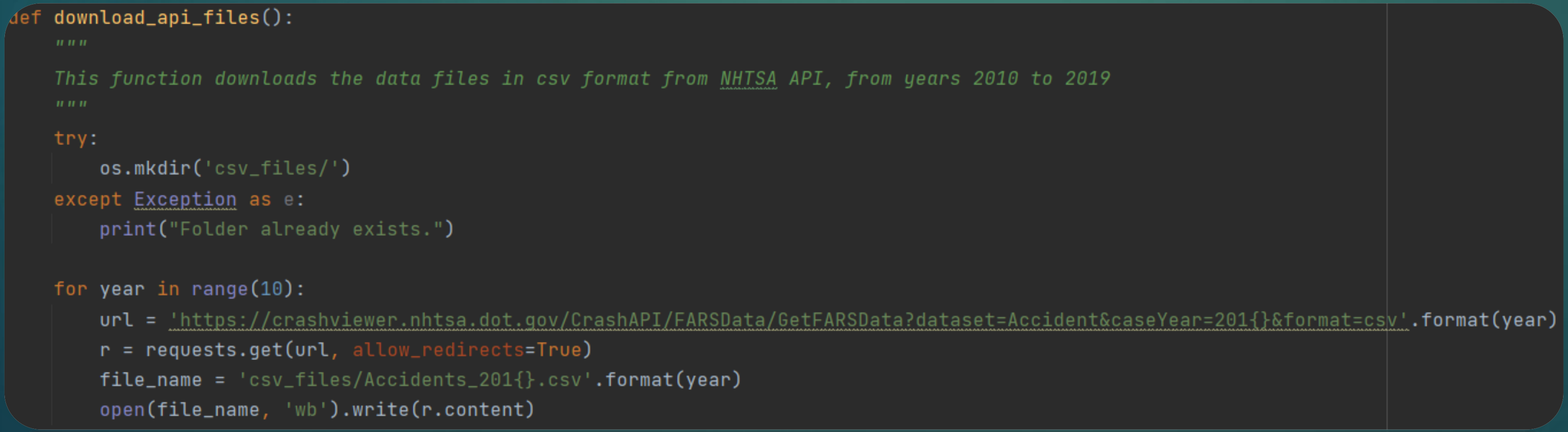
For non-developers that would like to query FARS data, please visit www-fars.nhtsa.dot.gov/querytool/

Fatality Analysis Reporting System (FARS) API	Output Format
Get Crash List Information `/crashes/GetCaseList?states=1,51&fromYear=2014&toYear=2015&minNumOfVehicles=1&maxNumOfVehicles=6&format=json` More Information	XML JSV CSV JSON
Get Crash Details `/crashes/GetCaseDetails?stateCase=10003&caseYear=2014&state=1&format=json` More Information	XML JSV XLSX JSON
Get Crashes By Vehicle `/crashes/GetCrashesByVehicle?make=6&model=41&modelyear=2003&bodyType=4&fromCaseYear=2010&toCaseYear=2015 &state=21&format=json` More Information	XML JSV CSV JSON
Get Crashes By Occupant `/crashes/GetCrashesByPerson?age=30&sex=2&seatPos=11&injurySeverity=2&fromCaseYear=2014&toCaseYear=2015&state=1&includeOccupants=true&includeNonOccupants=true&format=json` More Information	XML JSV CSV JSON
Get Crashes By Location `/crashes/GetCrashesByLocation?fromCaseYear=2014&toCaseYear=2015&state=1&county=1&format=json` More Information	XML JSV CSV JSON

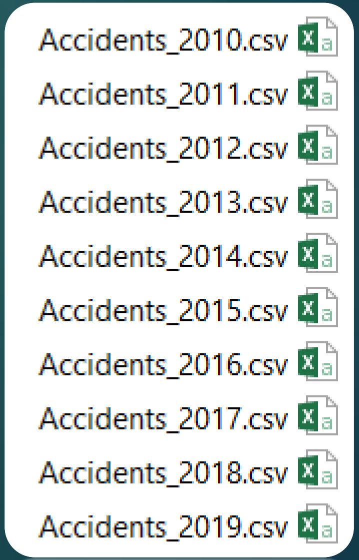
API & Sources

- 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/FARSData/GetFARSData?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.



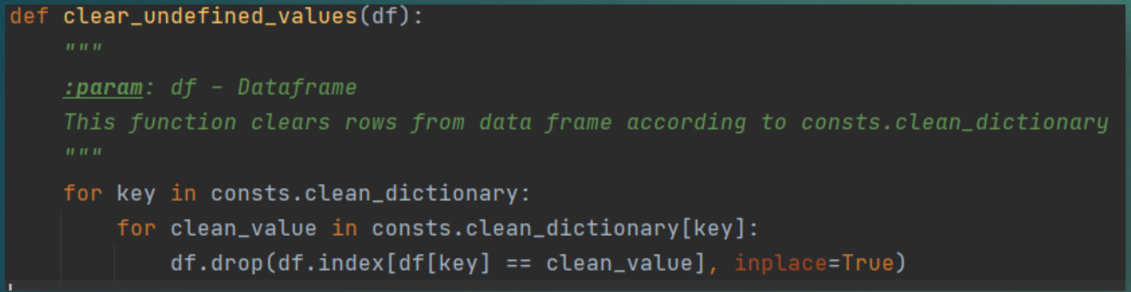
```python
def download_api_files():
    """
    This function downloads the data files in csv format from NHTSA API, from years 2010 to 2019
    """
    try:
        os.mkdir('csv_files/')
    except Exception as e:
        print("Folder already exists.")

    for year in range(10):
        url = 'https://crashviewer.nhtsa.dot.gov/CrashAPI/FARSData/GetFARSData?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)
```



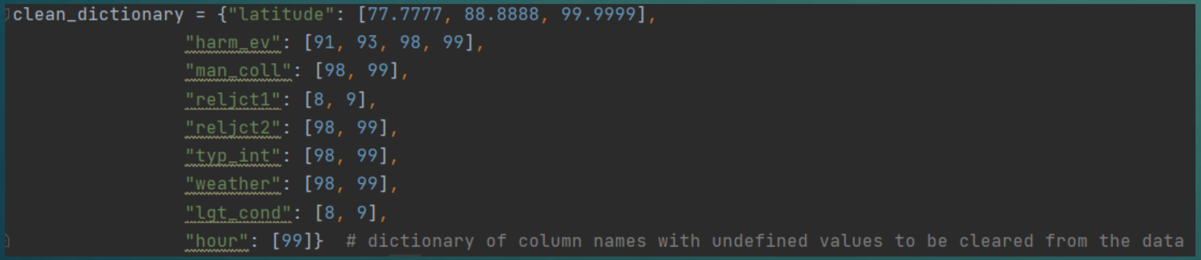
Accidents_2010.csv
Accidents_2011.csv
Accidents_2012.csv
Accidents_2013.csv
Accidents_2014.csv
Accidents_2015.csv
Accidents_2016.csv
Accidents_2017.csv
Accidents_2018.csv
Accidents_2019.csv

Data Handling and Cleaning

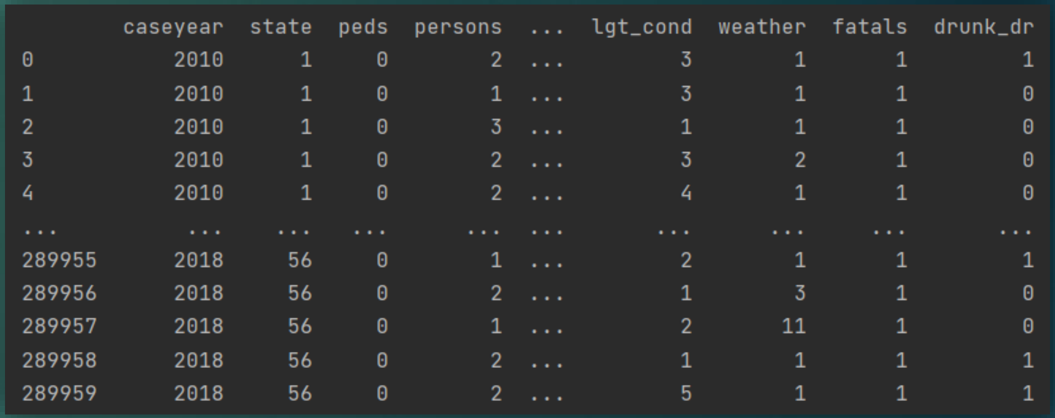
- In our project we used data from the years 2010 up to 2019.
- Each year had close to 35,000 records, eventually we got 323,480 records as result of 10 years of car accidents data from the USA.
- We omitted irrelevant data from the original csv files, combined all data to one file which will be used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

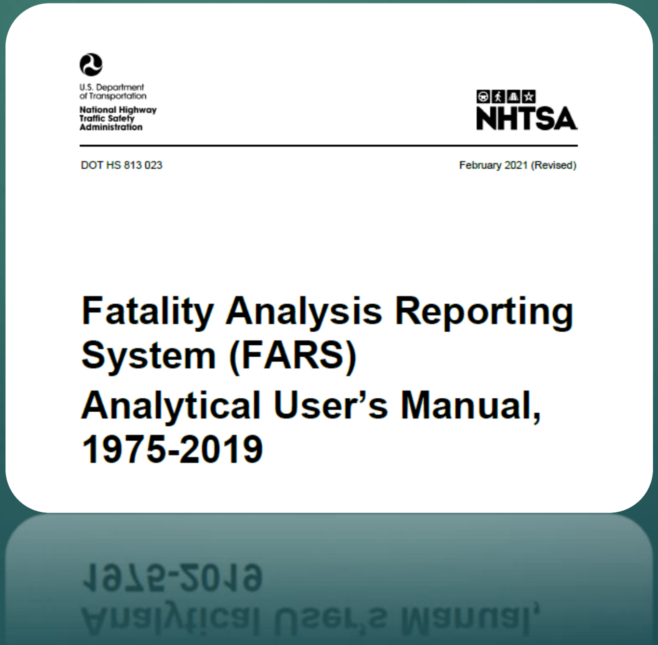


```
        caseyear  state  peds  persons  ...  lgt_cond  weather  fatals  drunk_dr
0           2010      1     0        2  ...         3        1       1         1
1           2010      1     0        1  ...         3        1       1         0
2           2010      1     0        3  ...         1        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.



U.S. Department of Transportation
National Highway Traffic Safety Administration

NHTSA

DOT HS 813 023

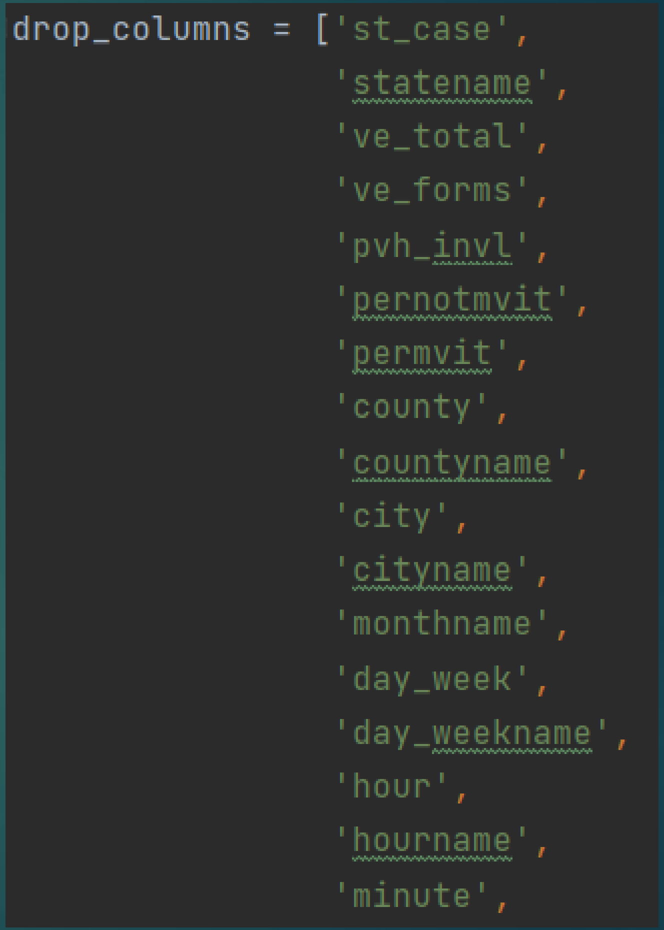
February 2021 (Revised)

Fatality Analysis Reporting System (FARS) Analytical User’s Manual, 1975-2019

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:

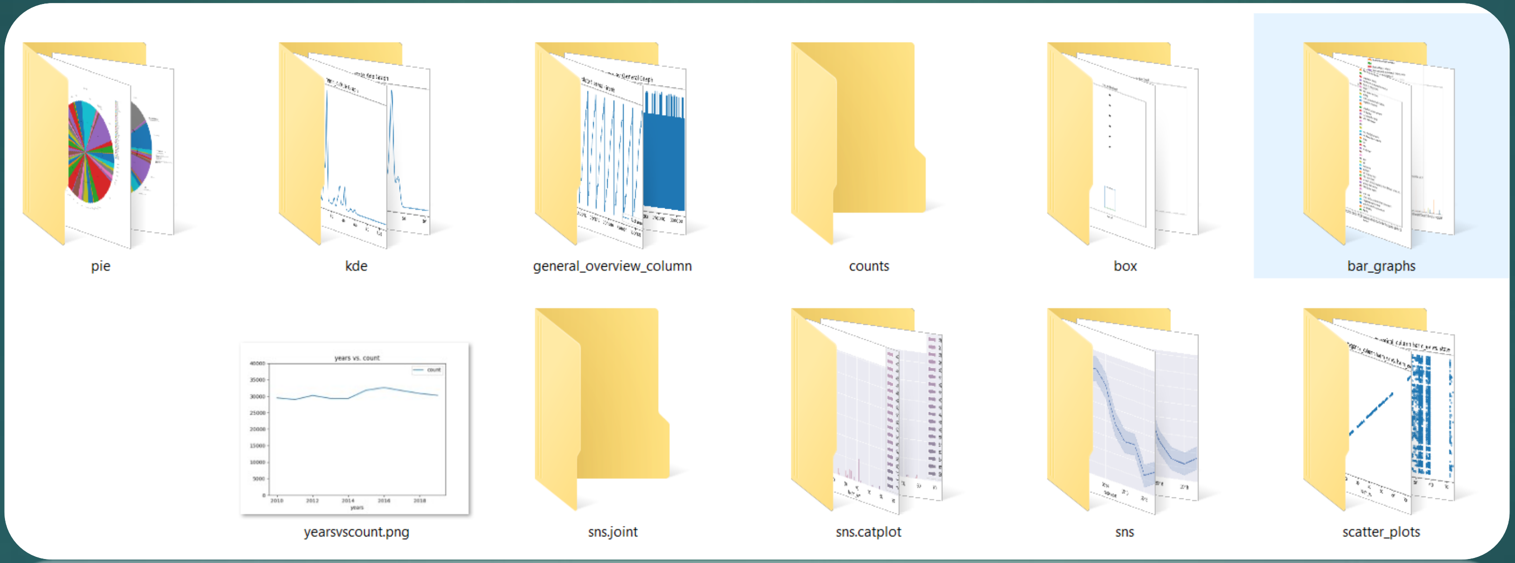
PDF Column explanation	Reason omitted
Consecutive Number ST_CASE 33	File consecutive number
Number of Vehicle Forms Submitted- ALL VE_TOTAL 39	Our research question does not refers to forms.



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

Graphs

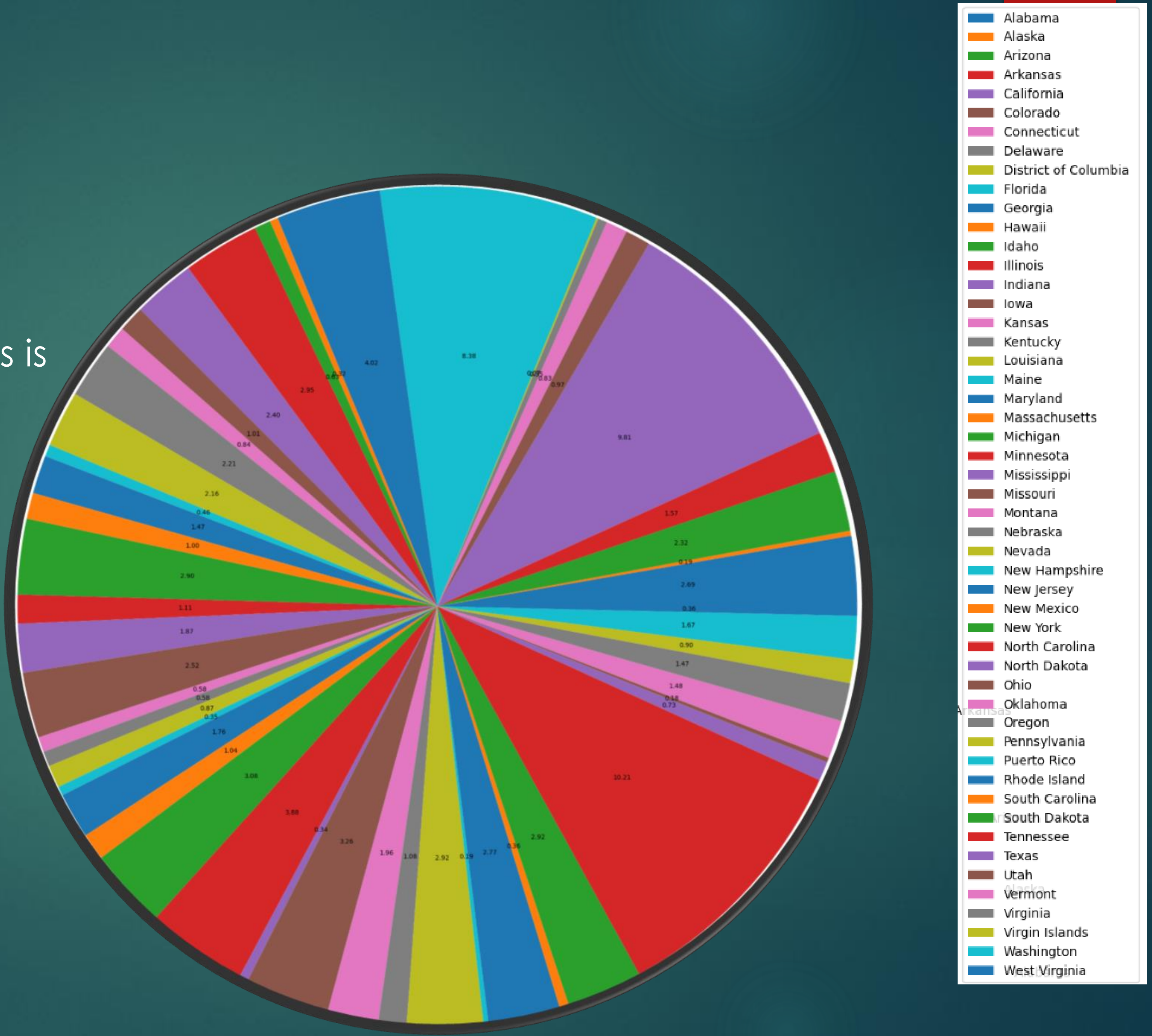
- We Used Pandas, Seaborn, Matplotlib to generate different variety of graphs according to our data.



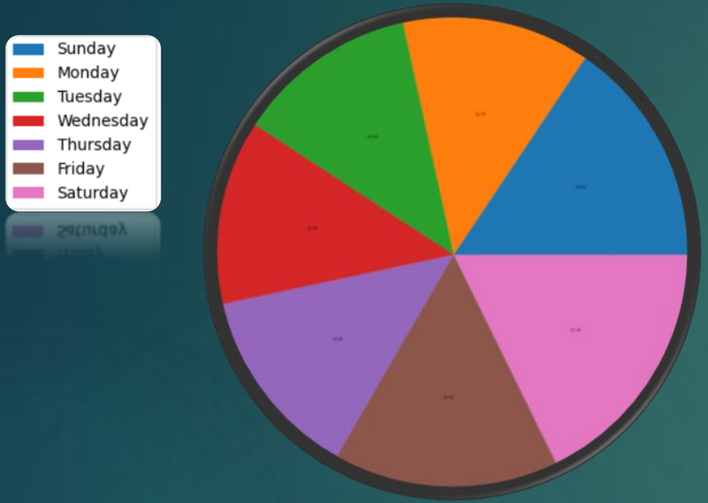
Graphs

Amount of Accidents Per State:

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

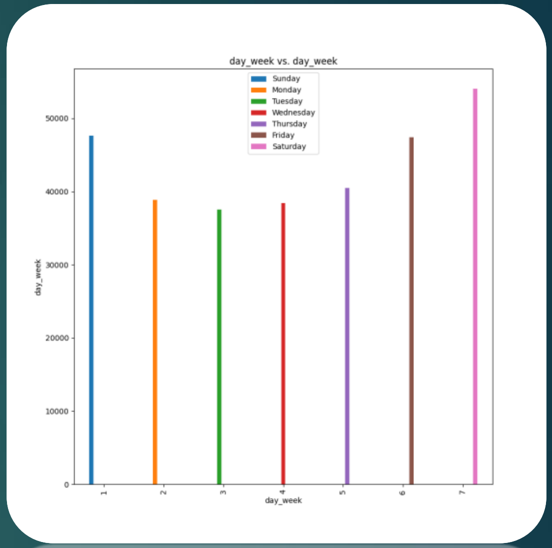


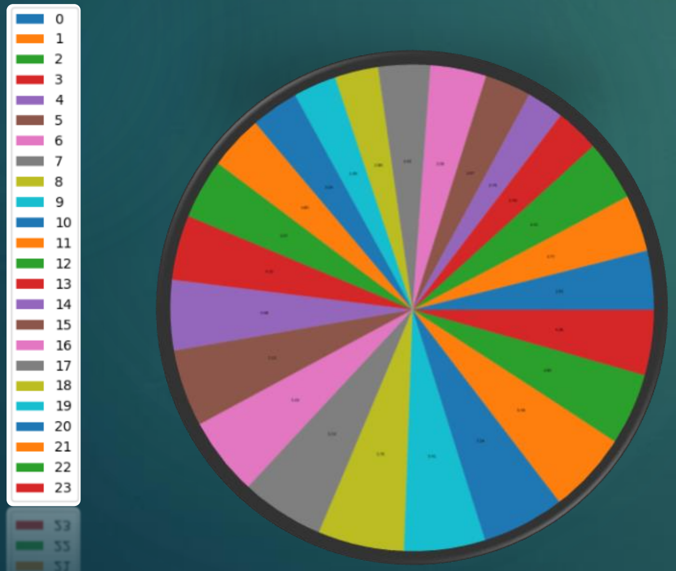
Graphs



Amount of Accidents Per Day:

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

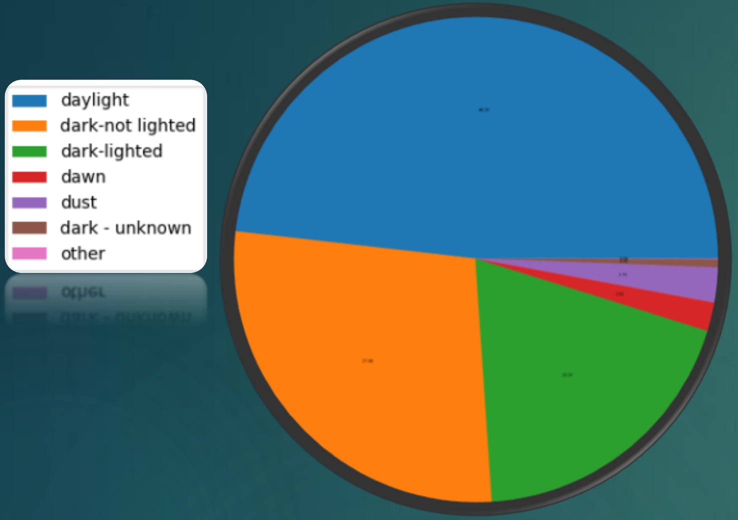




Amount of Accidents Per Hour:

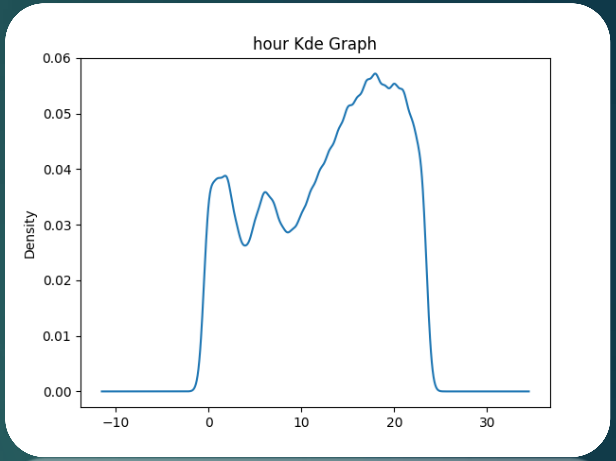
- Most accidents occur between 17:00 to 19:00.

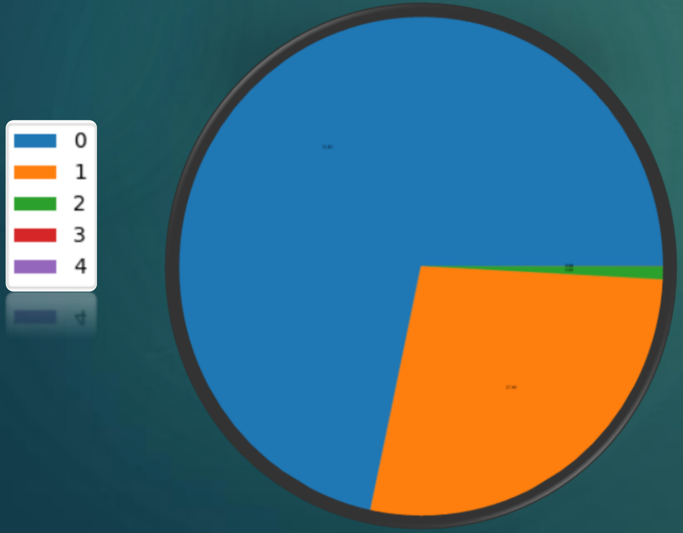
Graphs



Amount of Accidents – Light Conditions:

- Most accidents occur during daylight - 48%



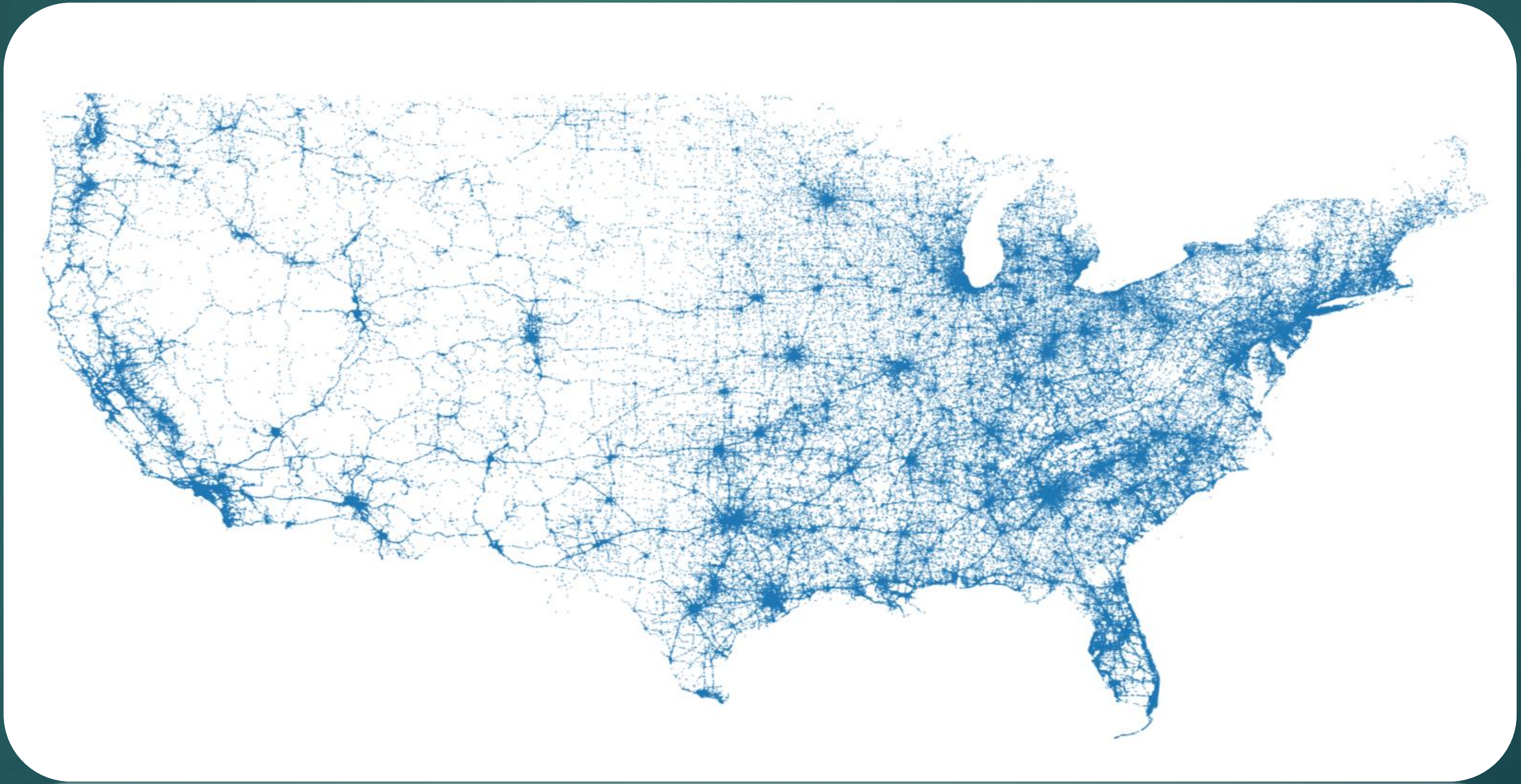


Amount of Accidents – Drunk Drivers Involved:

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

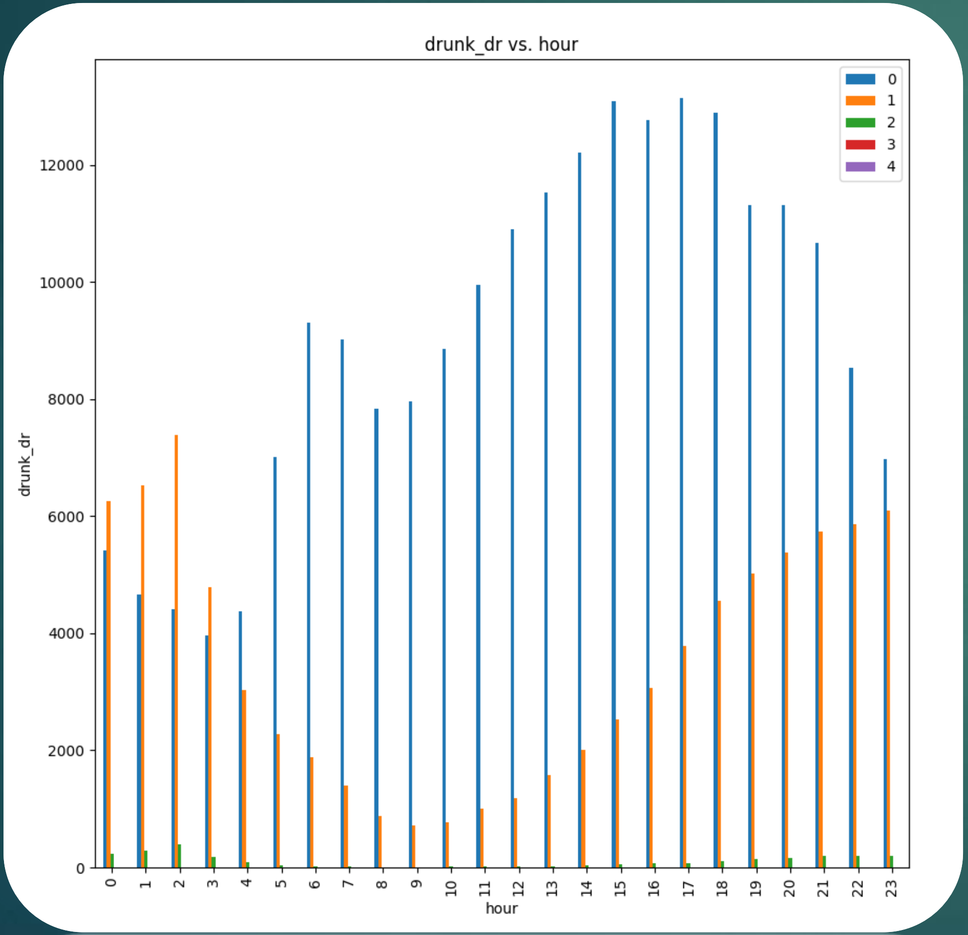
Graphs

In the following scatter plot, we can see the USA map, generated according to the accidents longitude and latitude locations, from the database.

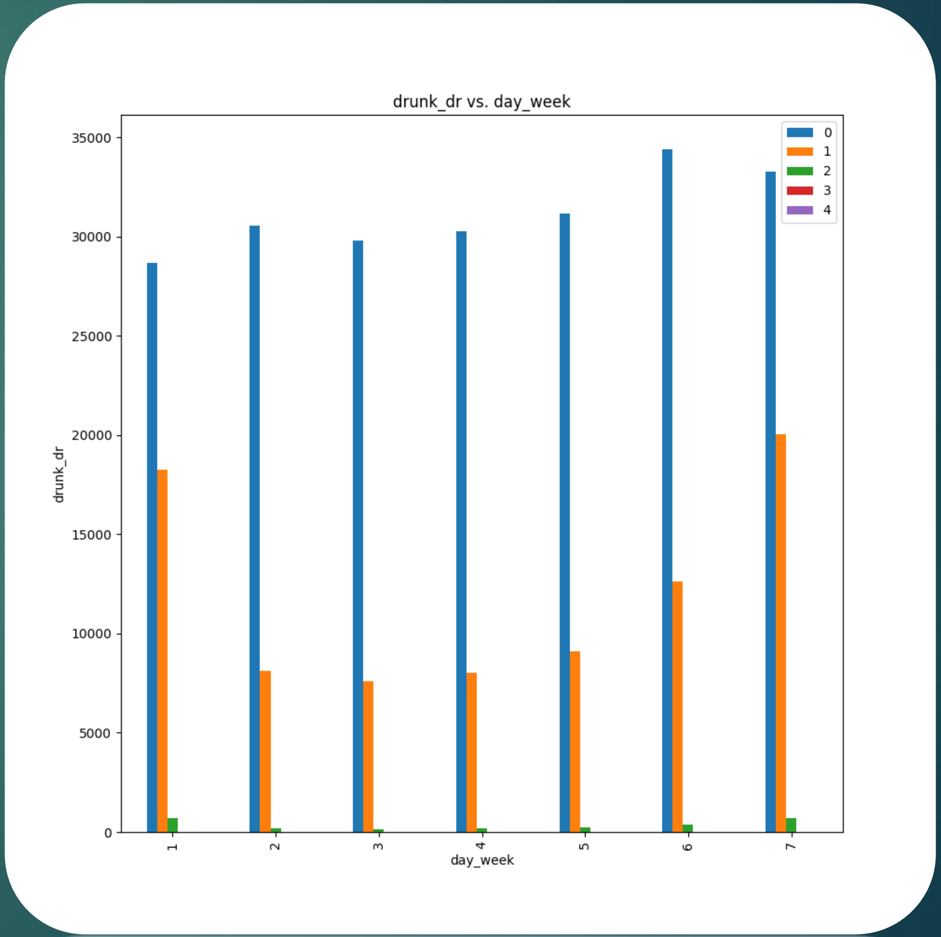


Graphs

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

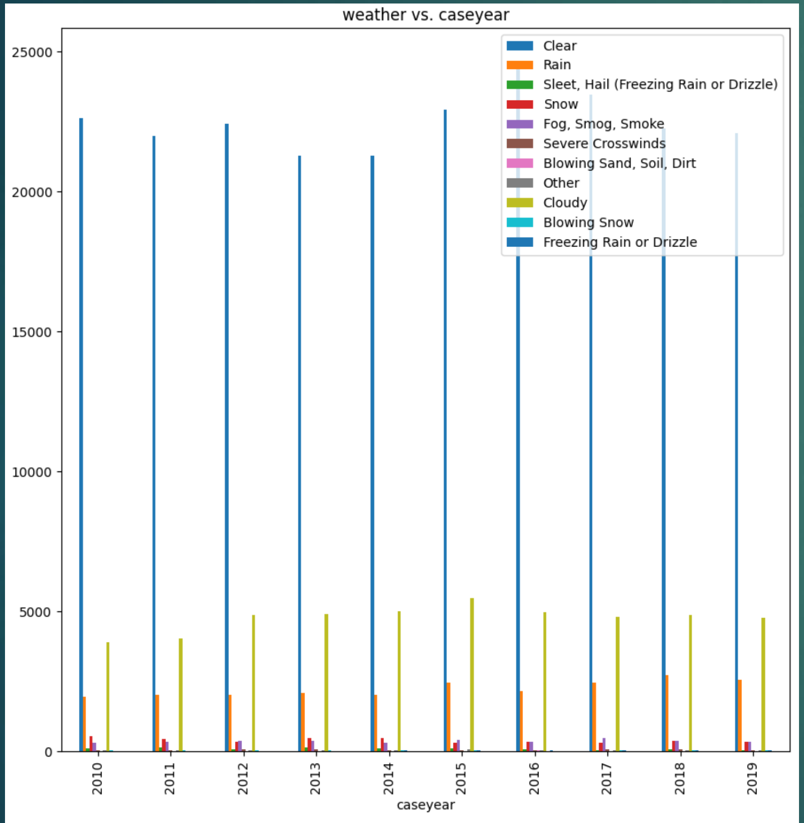


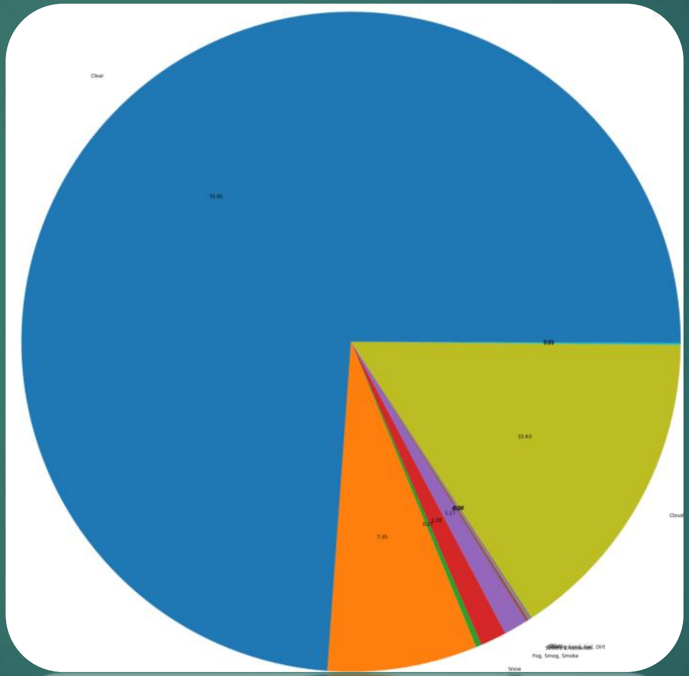
At weekends, accidents involve more drunk drivers(Friday, Saturday and Sunday).



Graphs

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

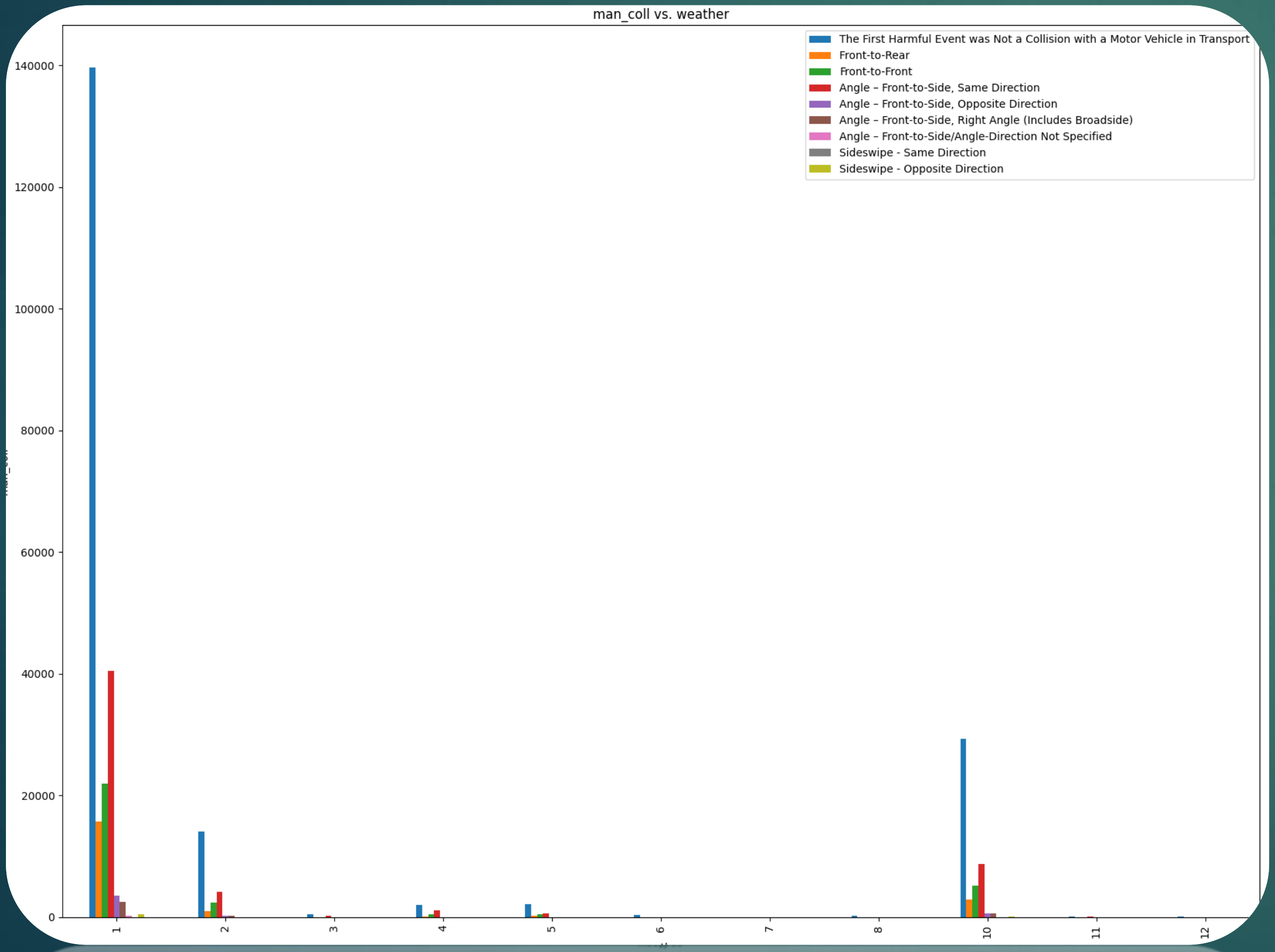




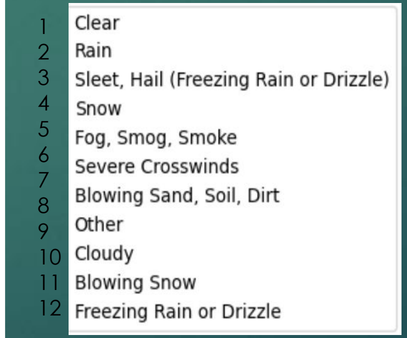
As you can see from the data, most of the accidents occur during clear weather.

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.

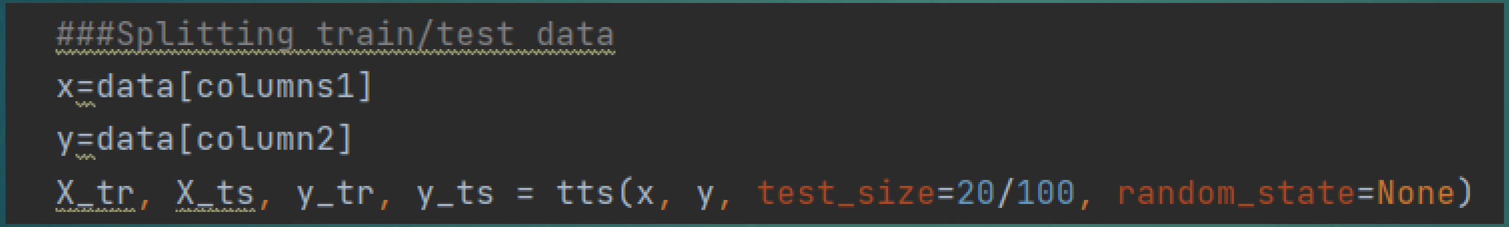


1	Clear
2	Rain
3	Sleet, Hail (Freezing Rain or Drizzle)
4	Snow
5	Fog, Smog, Smoke
6	Severe Crosswinds
7	Blowing Sand, Soil, Dirt
8	Other
9	Other
10	Cloudy
11	Blowing Snow
12	Freezing Rain or Drizzle

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

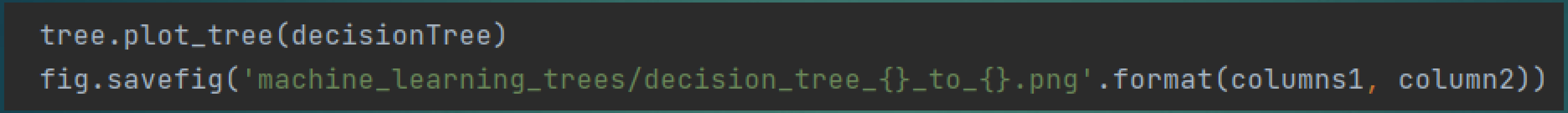


```
###Splitting train/test data
x=data[columns1]
y=data[column2]
X_tr, X_ts, y_tr, y_ts = tts(x, y, test_size=20/100, random_state=None)
```

- 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

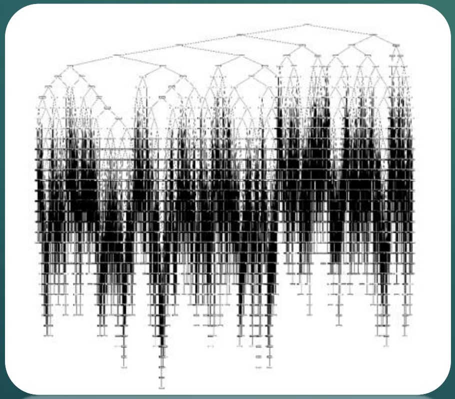
- Our code generates the Decision Tree as an output:



```
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')
```

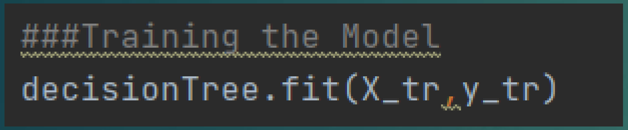


Click To Open File:

decision_tree_['day', 'month', 'state', 'weather', 'lgt_cond']_to_fatals.png

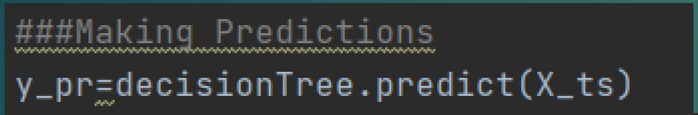
Machine Learning

- In the following steps we use fit() to train our model, with X_tr and Y_tr.

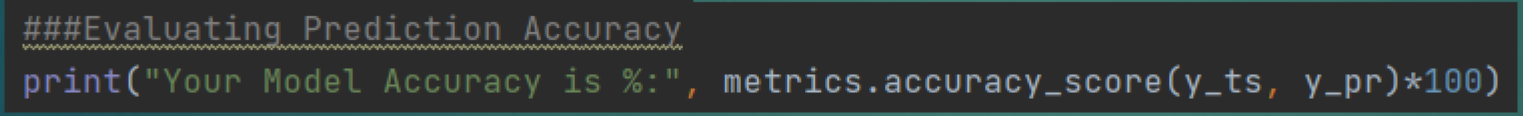


```
###Training the Model
decisionTree.fit(X_tr,y_tr)
```

- We use predict() with 20% of the data (X_ts and Y_ts) to test the data:

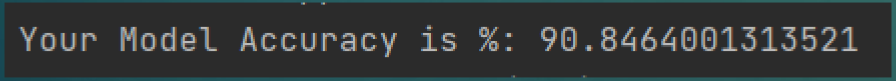


```
###Making Predictions
y_pr=decisionTree.predict(X_ts)
```



```
###Evaluating Prediction Accuracy
print("Your Model Accuracy is %:", metrics.accuracy_score(y_ts, y_pr)*100)
```

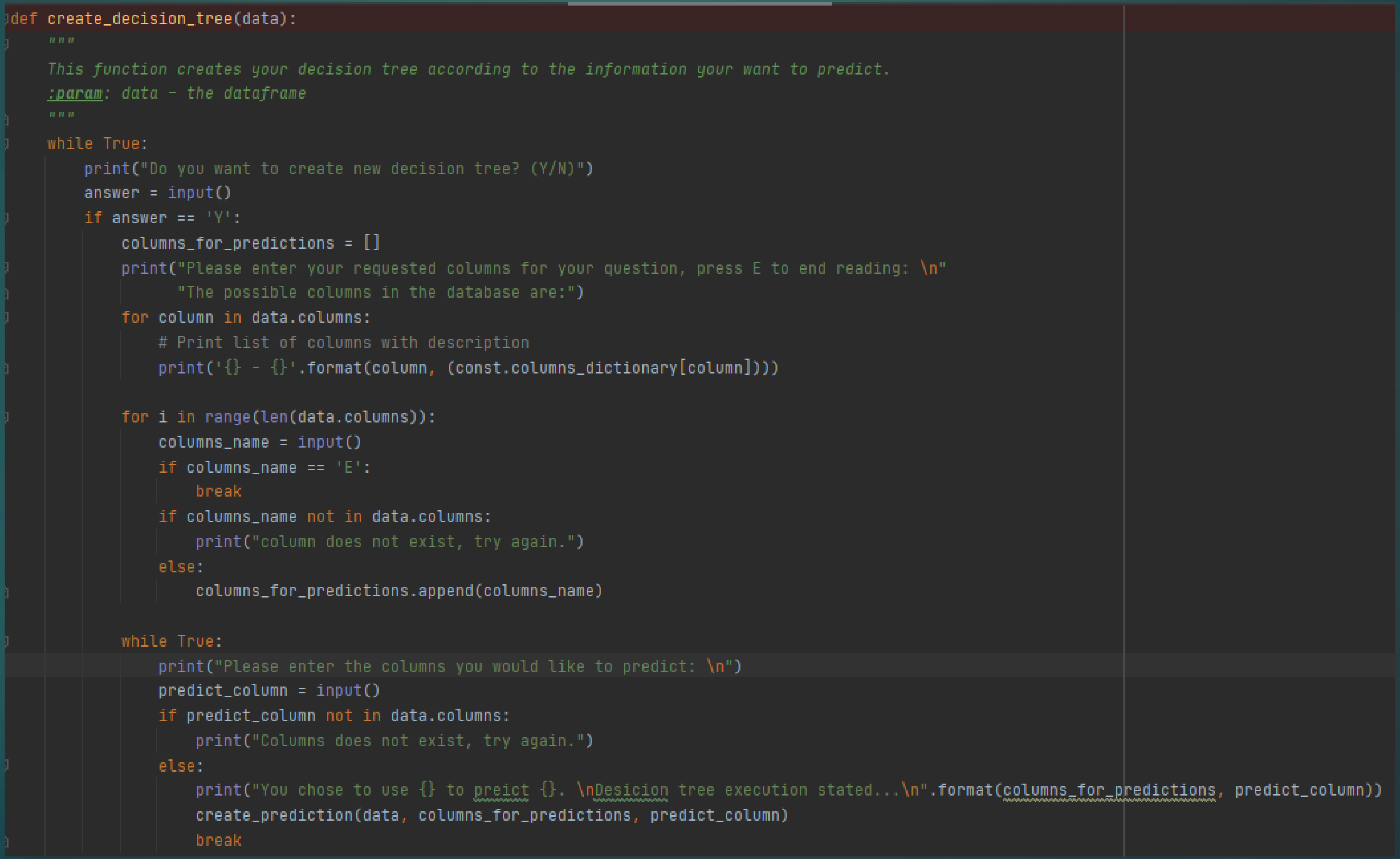
- As a result of our requested prediction: [Day, Month, State, Weather and Light Conditions] against [Fatality], we got the following results:



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

Machine Learning

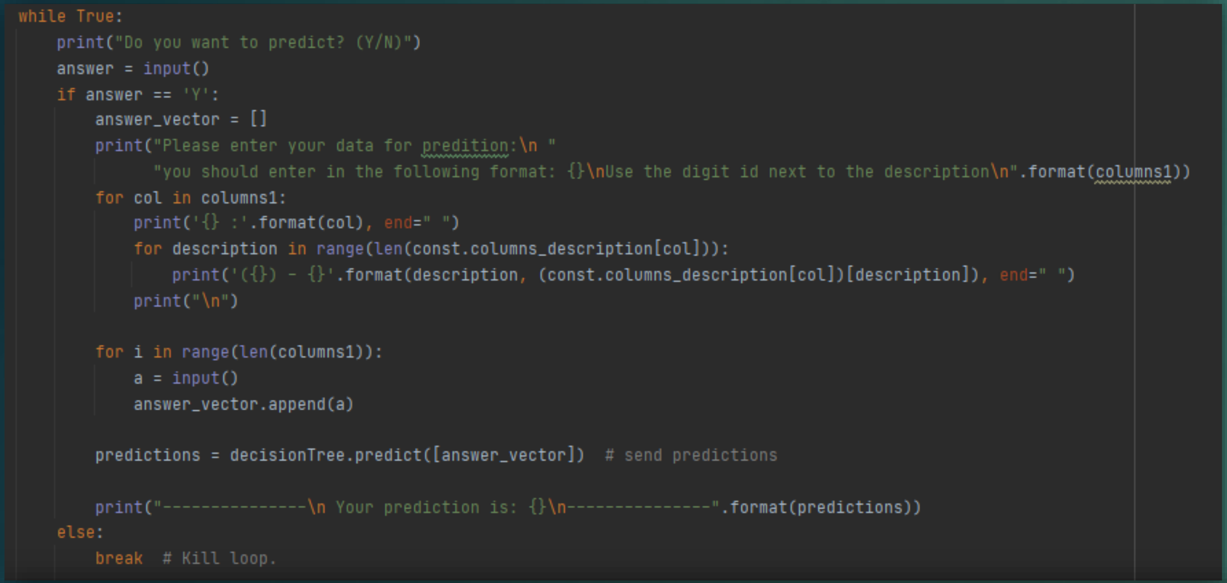
- We added auto prediction function to create and predict different inputs of columns:



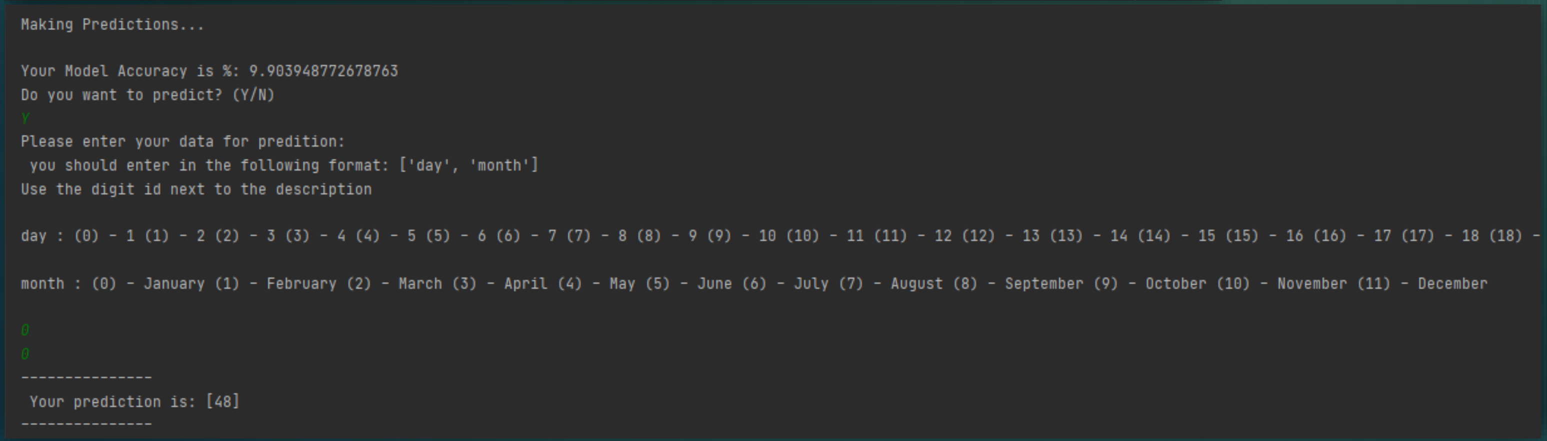
```python
def create_decision_tree(data):
    """
    This function creates your decision tree according to the information your want to predict.
    :param: data - the dataframe
    """
    while True:
        print("Do you want to create new decision tree? (Y/N)")
        answer = input()
        if answer == 'Y':
            columns_for_predictions = []
            print("Please enter your requested columns for your question, press E to end reading: \n"
                  "The possible columns in the database are:")
            for column in data.columns:
                # Print list of columns with description
                print('{} - {}'.format(column, (const.columns_dictionary[column])))

            for i in range(len(data.columns)):
                columns_name = input()
                if columns_name == 'E':
                    break
                if columns_name not in data.columns:
                    print("column does not exist, try again.")
                else:
                    columns_for_predictions.append(columns_name)

            while True:
                print("Please enter the columns you would like to predict: \n")
                predict_column = input()
                if predict_column not in data.columns:
                    print("Columns does not exist, try again.")
                else:
                    print("You chose to use {} to preict {}. \nDesicion tree execution stated...\n".format(columns_for_predictions, predict_column))
                    create_prediction(data, columns_for_predictions, predict_column)
                    break
```

Machine Learning

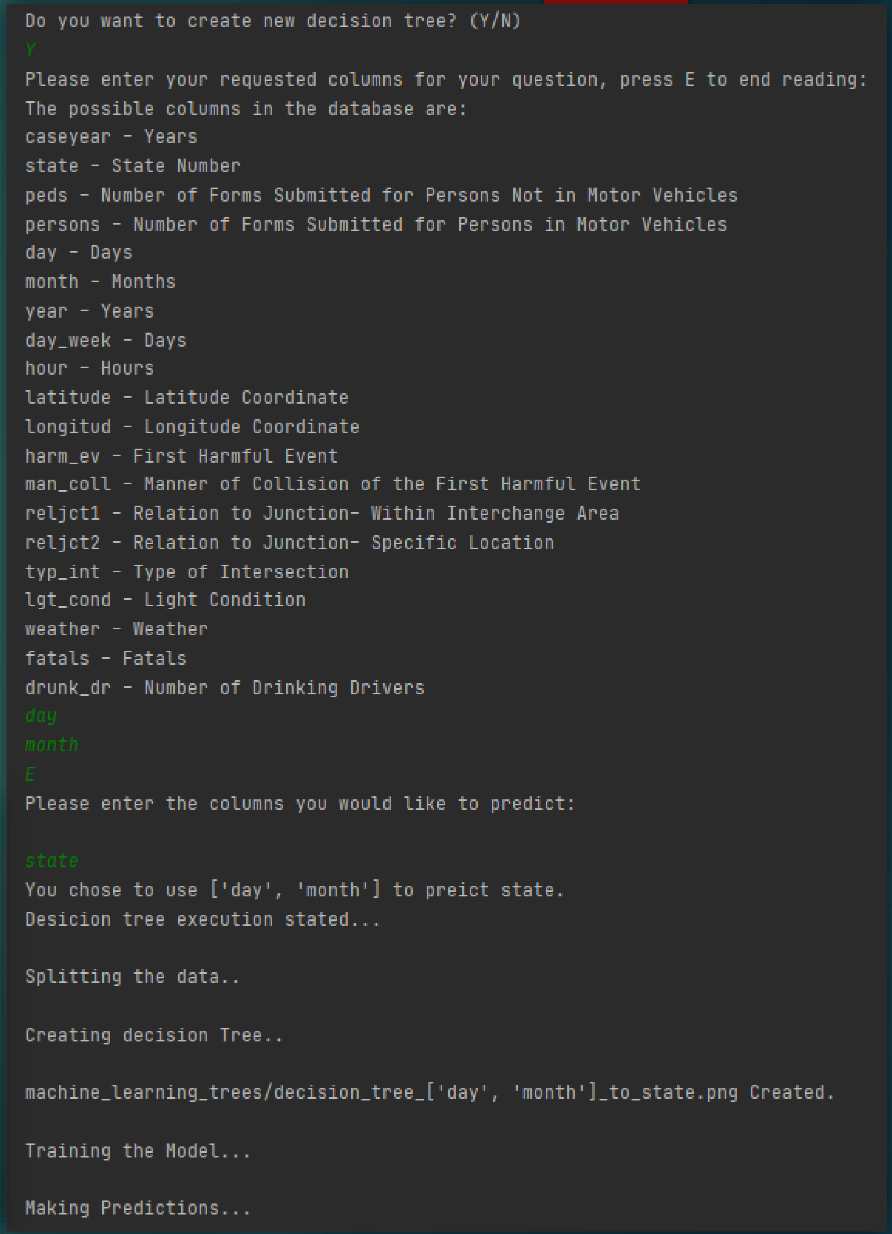
- We added auto prediction option to test different inputs of data:



```
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: {}\nUse the digit id next to the description\n".format(columns1))
        for col in columns1:
            print('{} :'.format(col), end=" ")
            for description in range(len(const.columns_description[col])):
                print('({}) - {}'.format(description, (const.columns_description[col])[description]), end=" ")
            print("\n")

        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.
```



```
Making Predictions...

Your Model Accuracy is %: 9.903948772678763
Do you want to predict? (Y/N)
Y
Please enter your data for predition:
 you should enter in the following format: ['day', 'month']
Use the digit id next to the description

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

month : (0) - January (1) - February (2) - March (3) - April (4) - May (5) - June (6) - July (7) - August (8) - September (9) - October (10) - November (11) - December

0
0
---------------
 Your prediction is: [48]
---------------
```

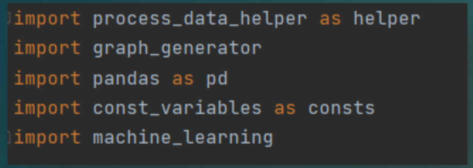


```
Do you want to create new decision tree? (Y/N)
Y
Please enter your requested columns for your question, press E to end reading:
The possible columns in the database are:
caseyear - Years
state - State Number
peds - Number of Forms Submitted for Persons Not in Motor Vehicles
persons - Number of Forms Submitted for Persons in Motor Vehicles
day - Days
month - Months
year - Years
day_week - Days
hour - Hours
latitude - Latitude Coordinate
longitud - Longitude Coordinate
harm_ev - First Harmful Event
man_coll - Manner of Collision of the First Harmful Event
reljct1 - Relation to Junction- Within Interchange Area
reljct2 - Relation to Junction- Specific Location
typ_int - Type of Intersection
lgt_cond - Light Condition
weather - Weather
fatals - Fatals
drunk_dr - Number of Drinking Drivers
day
month
E
Please enter the columns you would like to predict:

state
You chose to use ['day', 'month'] to preict state.
Desicion tree execution stated...

Splitting the data..

Creating decision Tree..

machine_learning_trees/decision_tree_['day', 'month']_to_state.png Created.

Training the Model...

Making Predictions...
```

Code Explanations and Project Structure

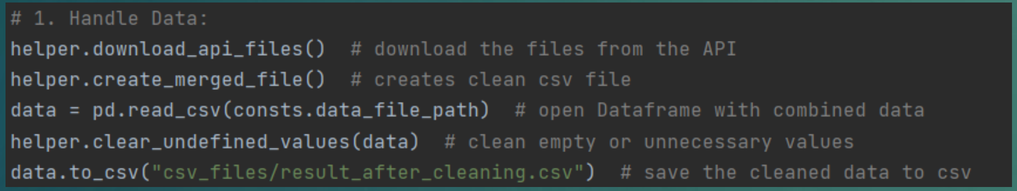
- main.py

Imports:



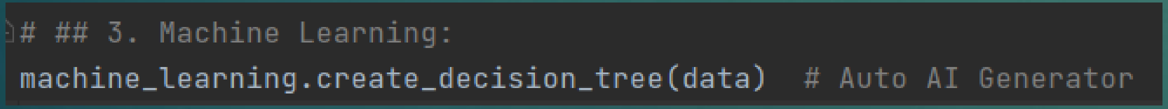
```
import process_data_helper as helper
import graph_generator
import pandas as pd
import const_variables as consts
import machine_learning
```

Getting, Handling and Cleaning the Data:



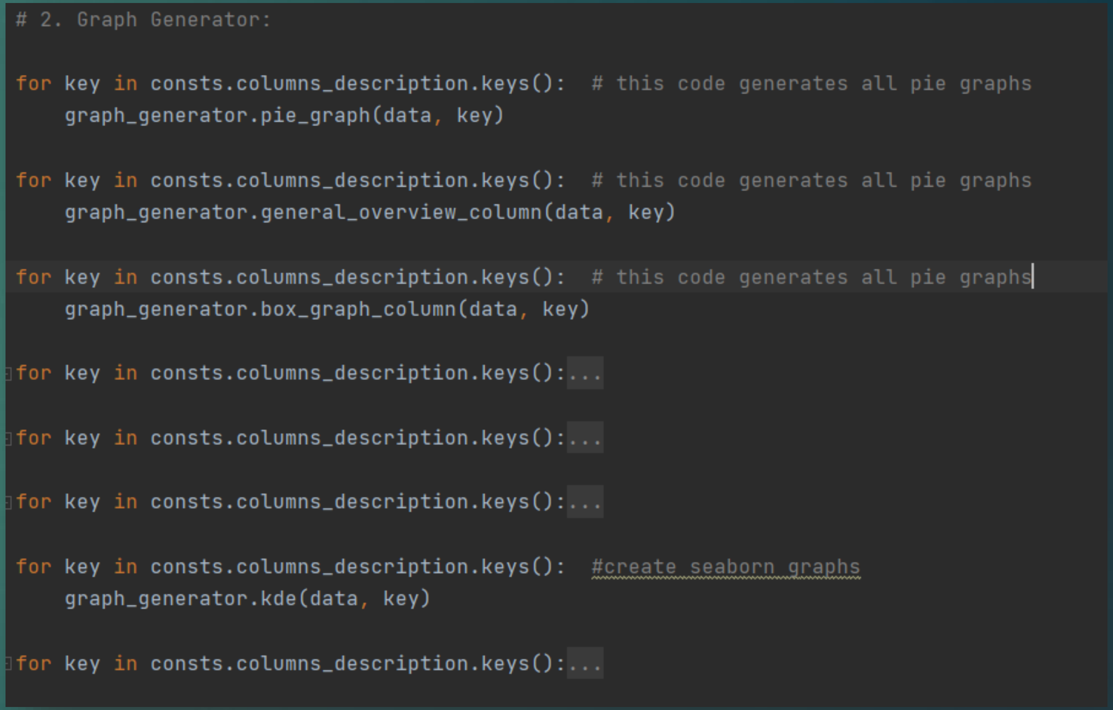
```
# 1. Handle Data:
helper.download_api_files()  # download the files from the API
helper.create_merged_file()  # creates clean csv file
data = pd.read_csv(consts.data_file_path)  # open Dataframe with combined data
helper.clear_undefined_values(data)  # clean empty or unnecessary values
data.to_csv("csv_files/result_after_cleaning.csv")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

Executing AI Machine Learning Process:



```
# ## 3. Machine Learning:
machine_learning.create_decision_tree(data)  # Auto AI Generator
```

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():...

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

for key in consts.columns_description.keys():  #create seaborn graphs
    graph_generator.kde(data, key)

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.

DRIVE
SAFELY