```
!python -V
```

```
Python 3.11.11
```

```
import numpy as np
import pandas as pd
import geopandas as gpd
import plotly.graph_objects as go
import plotly.express as px
import matplotlib.ticker as ticker
from datetime import datetime
from sklearn import tree
from itertools import chain


%matplotlib inline
```

```
# data recieved from https://catalog.data.gov/dataset/crash-reporting-drivers·
filename = "/content/maryland_crash_data.csv"
df = pd.read_csv(
    filename,
    dtype={
        "Report Number": str,
        "Local Case Number": str,
        # "Crash Date/Time": datetime,
        "Latitude": str,
        "Longitude": float,
        "Vehicle Year": int,
    },
    parse_dates=[
        "Crash Date/Time",
    ],
)
```

```
<ipython-input-3-1ee1d5b2bc70>:3: UserWarning: Could not infer format, so each
  df = pd.read_csv(
```

Project

Data source

As discussed in class, as the professor advised, I am changing my data source to be aligned with the govt data sources. My data source for this mining operation can be found at Data gov site

The data has `192183` rows and `39` columns.

```
features = [
    'Weather',
    'Surface Condition',
    'Light',
    'Driver Substance Abuse',
    'Driver Distracted By',
    'Speed Limit',
    'difference_crash_make_year',
]

result = ['Vehicle Damage Extent']
```

```
# Sanitize to remove absurd make year values
df = df[df["Vehicle Year"] < 2025]
df = df[df["Vehicle Year"] > 1970]
```

```
df["Vehicle Year"] = pd.to_datetime(df["Vehicle Year"], format='%Y')
df["difference_crash_make_year"] = (df["Crash Date/Time"] - df["Vehicle Year"]
```

```
<ipython-input-6-7da753a0cb32>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  df["Vehicle Year"] = pd.to_datetime(df["Vehicle Year"], format='%Y')
<ipython-input-6-7da753a0cb32>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  df["difference_crash_make_year"] = (df["Crash Date/Time"] - df["Vehicle Year
```

```
df = df[df["difference_crash_make_year"] >= 0 ]
```

Standardize the inputs

This way, futher processing can become easier!

First, let's make the text fields lower-case-d

Then let's filter out records that don't work too well, such as nan or unknown or other values

```
for coll in chain(features, result):
  try:
```

```
    df[coll] = df[coll].str.lower()
  except AttributeError as e:
    print(f"Could not convert {coll} to lower case")
```

```
Could not convert Speed Limit to lower case
Could not convert difference_crash_make_year to lower case
```

```
bad_values = {
  'Vehicle Damage Extent': [
      'unknown',
      'other',
      'disabling',
      'vehicle not at scene',
  ],
  'Surface Condition': [
      'unknown',
      'other',
  ],
  'Light': [
      'unknown',
      'other',
  ],
  'Driver Substance Abuse': [
      'unknown',
      'other',
      'unknown, unknown',
  ],
  'Driver Distracted By': [
      'unknown',
      'other',
  ],
}

for coll in chain(features, result):
  df = df[df[coll].notna()]

for coll, values in bad_values.items():
  try:
    df = df[~df[coll].isin(values)]
  except Exception as e:
    print(f"Could not remove unwanted vals for {coll}")
    print(e)
```

```
display(df)
```

	Report Number	Local Case Number	Agency Name	ACRS Report Type	Crash Date/ Time	Route Type	Road
			Montgomery	Injury	2023-08-28	Maryland	NO

5	MCP3348000Z	230051804	Montgomery County Police	Injury Crash	2023-08-28 11:09:00	Maryland (State)	NO
6	MCP302600BD	230046425	Montgomery County Police	Property Damage Crash	2023-07-27 12:30:00	County	GREE
8	MCP3372001V	230065250	Montgomery County Police	Property Damage Crash	2023-11-10 20:24:00	Maryland (State)	GE
17	MCP2962008G	230065146	Montgomery County Police	Property Damage Crash	2023-11-08 14:05:00	County	F H
19	MCP3136006J	230048375	Montgomery County Police	Property Damage Crash	2023-08-08 11:39:00	Maryland (State)	BRAN
...	...	...	...	...	...	...	
192169	MCP312900BT	250003767	MONTGOMERY	Injury Crash	2025-01-27 07:56:00	County Route	
192171	MCP3171002L	250004790	MONTGOMERY	Property Damage Crash	2025-02-02 12:30:00	County Route	CEI BLVD
192172	MCP235300D1	250004139	MONTGOMERY	Property Damage Crash	2025-01-29 06:40:00	Maryland (State) Route	ROC PIKE
192174	MCP283200B9	250004515	MONTGOMERY	Property Damage Crash	2025-01-31 12:31:00	County Route	MAN
192176	MCP3404001T	250004298	MONTGOMERY	Injury Crash	2025-01-30 07:00:00	County Route	LOCK

62916 rows × 40 columns

Visualization

Here we can plot all the accidents in our dataset and see how clustered the accidents are and we can also see what year cars are more accident prone and if it has anything to do with how old the cars are.

There might be other interesting ideas that might pop up after checking out the visualizations!

```
fig_vehicle_year = px.scatter_geo(df, lat="Latitude", lon="Longitude", color='
fig_vehicle_year.show()
```

```
fig_vehicle_difference_year = px.scatter_geo(df, lat="Latitude", lon="Longitud
fig_vehicle_difference_year.show()
```

```
1 display(df.groupby(["difference_crash_make_year"])["difference_crash_make_yea
```

	difference_crash_make_year
difference_crash_make_year	
0	3177
1	4646
2	4615
3	4608
4	4580
5	4352
6	4147
7	3789
8	3708
9	3406
10	3257
11	2892
12	2594
13	2369
14	2003
15	1716
16	1506
17	1281
18	1031
19	839

20	622
21	480
22	345
23	248
24	174
25	135
26	96
27	65
28	61
29	48
30	21
31	18
32	19
33	18
34	18
35	12
36	5
37	3
38	1
39	3
40	3
41	1
42	1
43	1
44	1
47	1

dtype: int64

Corr, Mean, Median

I'm still understanding what useful knowledge can be extracted to use to make safer cars.

I will use Corr, Mean, Median to help determine if cars have gotten safer over time.

```
df[[
    "Vehicle Year",
    "difference_crash_make_year"
]].corr(min_periods=3)
```

	Vehicle Year	difference_crash_make_year
Vehicle Year	1.000000	-0.883547
difference_crash_make_year	-0.883547	1.000000

```
df[[
    # "Vehicle Year",
    "difference_crash_make_year"
]].median(
    axis=0,
)
```

	0
difference_crash_make_year	7.0

dtype: float64

```
df[[
    # "Vehicle Year",
    "difference_crash_make_year"
]].mean(
    axis=0,
)
```

	0
difference_crash_make_year	7.865805

dtype: float64

```
df[[
    "Vehicle Make",
]].value_counts().idxmax()
```

```
('TOYOTA',)
```

Findings

Findings

As we can see, on average, we can expect accidents to start occuring after the car is around 7~8 years of age. This dataset also highlights that the 2012 models are more likely to be the ones in accident. This could also just be a red herring that is throwing off our data since a lot of crashes can go unreported.

Algo selection

I would like to explore a few different ideas. But one of the algorithms we were taught in class was decision tree. I would like to explore more columns with decision tree to determine rates of accidents given the factors, such as:

- Weather
- Driver substance abuse
- Speed limit of road

These are our feature-set.

We will use features to predict how bad the accident turns out to be, based on the damage occurred to the vehicle.

A major reason to use the decision tree is that it is easy to see the outcome. This will produce a very clean algebraic equation that can help customers buy the right car for them!

This is precisely the model I have chosen to implement

After working on it, and enhancing it and playing around with inputs and parameters, this seems to be not a terrible choice!

```
remap = {
    'Driver Substance Abuse': {
        'none detected': False,
        'alcohol contributed': True,
        'alcohol present': True,
        'illegal drug present': True,
        'illegal drug contributed': True,
        'medication present': True,
        'combined substance present': True,
        'medication contributed': True,
        'combination contributed': True,
        'not suspect of alcohol use, not suspect of drug use': False,
        'suspect of alcohol use, not suspect of drug use': True,
        'suspect of alcohol use, unknown': True,
        'unknown, not suspect of drug use': False,
```

```
        'not suspect of alcohol use, unknown': False,
        'suspect of alcohol use, suspect of drug use': True,
        'not suspect of alcohol use, suspect of drug use': True,
    },
    'Light': {
        'dark -- unknown lighting': 'dark',
        'dark - unknown lighting': 'dark',
    },
}

pd.set_option('future.no_silent_downcasting', True)
for feature, mapping in remap.items():
  try:
    df = df.replace({ feature: mapping })
  except Exception as e:
    print(f"Could not remap {feature}")
    print(e)
```

```
for feature in chain(features, result):
  print(f"{feature}: {list(df[feature].unique())}")
```

```
Weather: ['cloudy', 'clear', 'raining', 'snow', 'foggy', 'wintry mix', 'other'
Surface Condition: ['dry', 'wet', 'ice', 'slush', 'water(standing/moving)', 's
Light: ['daylight', 'dark lights on', 'dawn', 'dusk', 'dark no lights', 'dark'
Driver Substance Abuse: [False, True]
Driver Distracted By: ['not distracted', 'looked but did not see', 'other dist
Speed Limit: [np.int64(30), np.int64(35), np.int64(40), np.int64(5), np.int64(
difference_crash_make_year: [np.int64(13), np.int64(1), np.int64(7), np.int64(
Vehicle Damage Extent: ['functional', 'superficial', 'no damage', 'destroyed']
```

Narrow down

Here we will narrow down our df to use only some columns, like our feature columns and outcome column(s)

```
df = df[chain(features, result)]
```

ONE HOT ENCODING

Other than the great models, I have also learnt that pre-processing is very important! One Hot Encoding is another useful option in preprocessing. This has been absolutely eye-opening for me when it comes to data processing and extracting information and knowledge from it using mining methods.

```
for coll in chain(features, result):
  if coll in [
      'Driver Substance Abuse',
      'Speed Limit',
      'difference_crash_make_year',
      'Vehicle Damage Extent',
  ]:
    continue
  try:
    tmp = pd.get_dummies(df[coll], prefix=coll.lower(), prefix_sep='_')
    df = df.drop(coll, axis = 1)
    df = df.join(tmp)
  except Exception as e:
    print(f"Could not convert {coll} to one hot encoding")
    print(e)
```

```
display(list(df.columns))
```

```
['Driver Substance Abuse',
 'Speed Limit',
 'difference_crash_make_year',
 'Vehicle Damage Extent',
 'weather_blowing sand, soil, dirt',
 'weather_blowing snow',
 'weather_clear',
 'weather_cloudy',
 'weather_fog, smog, smoke',
 'weather_foggy',
 'weather_freezing rain or freezing drizzle',
 'weather_other',
 'weather_rain',
 'weather_raining',
 'weather_severe crosswinds',
 'weather_severe winds',
 'weather_sleet',
 'weather_sleet or hail',
 'weather_snow',
 'weather_unknown',
 'weather_wintry mix',
 'surface condition_dry',
 'surface condition_ice',
 'surface condition_ice/frost',
 'surface condition_mud, dirt, gravel',
 'surface condition_oil',
 'surface condition_sand',
 'surface condition_slush',
 'surface condition_snow',
 'surface condition_water (standing, moving)',
 'surface condition_water(standing/moving)',
 'surface condition_wet',
 'light_dark',
 'light_dark - lighted',
```

```
    'light_dark - not lighted',
    'light_dark lights on',
    'light_dark no lights',
    'light_dawn',
    'light_daylight',
    'light_dusk',
    'driver distracted by_adjusting audio and or climate controls',
    'driver distracted by_by moving object in vehicle',
    'driver distracted by_by other occupants',
    'driver distracted by_dialing cellular phone',
    'driver distracted by_distracted by outside person object or event',
    'driver distracted by_eating or drinking',
    'driver distracted by_inattentive or lost in thought',
    'driver distracted by_looked but did not see',
    'driver distracted by_manually operating (dialing, playing game, etc.)',
    'driver distracted by_no driver present',
    'driver distracted by_not distracted',
    'driver distracted by_other action (looking away from task, etc.)',
    'driver distracted by_other cellular phone related',
    'driver distracted by_other distraction',
    'driver distracted by_other electronic device (navigational palm pilot)',
    'driver distracted by_smoking related',
    'driver distracted by_talking or listening to cellular phone',
    'driver distracted by_talking/listening',
    'driver distracted by_texting from a cellular phone',
    'driver distracted by_using device object brought into vehicle',
    'driver distracted by_using other device controls integral to vehicle']
```

```
results = [
    # 'vehicle damage extent_destroyed',
    # 'vehicle damage extent_no damage',
    # 'vehicle damage extent_functional',
    # 'vehicle damage extent_superficial',
    'Vehicle Damage Extent',
]

features = [
    'Driver Substance Abuse',
    'Speed Limit',
    'difference_crash_make_year',
    'weather_blowing sand, soil, dirt',
    'weather_blowing snow',
    'weather_clear',
    'weather_cloudy',
    'weather_fog, smog, smoke',
    'weather_foggy',
    'weather_freezing rain or freezing drizzle',
    'weather_other',
    'weather_rain',
    'weather_raining',
    'weather_severe crosswinds',
    'weather_severe winds',
```

```
    'weather_sleet',
    'weather_sleet or hail',
    'weather_snow',
    'weather_unknown',
    'weather_wintry mix',
    'surface condition_dry',
    'surface condition_ice',
    'surface condition_ice/frost',
    'surface condition_mud, dirt, gravel',
    'surface condition_oil',
    'surface condition_sand',
    'surface condition_slush',
    'surface condition_snow',
    'surface condition_water (standing, moving)',
    'surface condition_water(standing/moving)',
    'surface condition_wet',
    'light_dark',
    'light_dark - lighted',
    'light_dark - not lighted',
    'light_dark lights on',
    'light_dark no lights',
    'light_dawn',
    'light_daylight',
    'light_dusk',
    'driver distracted by_adjusting audio and or climate controls',
    'driver distracted by_by moving object in vehicle',
    'driver distracted by_by other occupants',
    'driver distracted by_dialing cellular phone',
    'driver distracted by_distracted by outside person object or event',
    'driver distracted by_eating or drinking',
    'driver distracted by_inattentive or lost in thought',
    'driver distracted by_looked but did not see',
    'driver distracted by_manually operating (dialing, playing game, etc.)',
    'driver distracted by_no driver present',
    'driver distracted by_not distracted',
    'driver distracted by_other action (looking away from task, etc.)',
    'driver distracted by_other cellular phone related',
    'driver distracted by_other distraction',
    'driver distracted by_other electronic device (navigational palm pilot)',
    'driver distracted by_smoking related',
    'driver distracted by_talking or listening to cellular phone',
    'driver distracted by_talking/listening',
    'driver distracted by_texting from a cellular phone',
    'driver distracted by_using device object brought into vehicle',
    'driver distracted by_using other device controls integral to vehicle',
]
```

Train Test

Here we will set up the training data and the test data and use it to generate and score a decision tree.

```
msk = np.random.rand(len(df)) < 0.8

df_train = train = df[msk]
df_test = df[~msk]
```

```
df_features = df_train[features]
df_result = df_train[results]

dtree = tree.DecisionTreeClassifier()
dtree = dtree.fit(df_features, df_result)
```

```
score = dtree.score(df_test[features], df_test[results])
print(f"Score: {score}")
```

```
Score: 0.45638801261829653
```

Score

So this is probably my 5th or 7th iteration

Earlier I got even worse scores

But playing around, I was able to see what works.

Some of my learnings:

- One Hot Encoding is a must, or else it can get pretty bad interpretting certain attributes as categories
- Encoding of the resultant category is not as necessary and is better not to encode it
- Geographical data points are ONLY useful if we use geo encodings so that the data points' closeness can be interpretted by computers. Seeing it on a map is not enough. A deep dive in the near future will lead me to Geographical databases or geo databses. They are super cool!
- In terms of findings, I think I am confident to say (after many selections of features and resultants etc.) that these features highlighted in this version of my notebook is really really good! however, further fine tuning is necessary! I wish to expand this project into the future to see its potential!
- I was also reading that there are more types of decision trees even, and this package may not have them all! I would love to code one up from scratch!

not have them all. I would love to code one up from scratch.