```python
data_path = "gs://us_accidents_projects/US_Accidents_March23.csv"
df = spark.read.csv(data_path, header=True, inferSchema=True)
```

```python
df.columns
```

```
['ID',
 'Source',
 'Severity',
 'Start_Time',
 'End_Time',
 'Start_Lat',
 'Start_Lng',
 'End_Lat',
 'End_Lng',
 'Distance(mi)',
 'Description',
 'Street',
 'City',
 'County',
 'State',
 'Zipcode',
 'Country',
 'Timezone',
 'Airport_Code',
 'Weather_Timestamp',
 'Temperature(F)',
 'Wind_Chill(F)',
 'Humidity(%)',
 'Pressure(in)',
 'Visibility(mi)',
 'Wind_Direction',
 'Wind_Speed(mph)',
 'Precipitation(in)',
 'Weather_Condition',
 'Amenity',
 'Bump',
 'Crossing',
 'Give_Way',
 'Junction',
 'No_Exit',
 'Railway',
 'Roundabout',
 'Station',
 'Stop',
 'Traffic_Calming',
 'Traffic_Signal',
 'Turning_Loop',
 'Sunrise_Sunset',
 'Civil_Twilight',
 'Nautical_Twilight',
 'Astronomical_Twilight']
```

```
df.describe()
```

```
DataFrame[summary: string, ID: string, Source: string, Severity:
string, Start_Lat: string, Start_Lng: string, End_Lat: string,
End_Lng: string, Distance(mi): string, Description: string, Street:
string, City: string, County: string, State: string, Zipcode: string,
Country: string, Timezone: string, Airport_Code: string,
Temperature(F): string, Wind_Chill(F): string, Humidity(%): string,
Pressure(in): string, Visibility(mi): string, Wind_Direction: string,
Wind_Speed(mph): string, Precipitation(in): string, Weather_Condition:
string, Sunrise_Sunset: string, Civil_Twilight: string,
Nautical_Twilight: string, Astronomical_Twilight: string]
```

```
import pandas as pd_10
print('Columns overview')
pd_10.DataFrame(df.dtypes, columns = ['Column Name','Data type'])
```

```
Columns overview
```

```
              Column Name  Data type
0                      ID     string
1                  Source     string
2                Severity        int
3              Start_Time  timestamp
4                End_Time  timestamp
5               Start_Lat     double
6               Start_Lng     double
7                 End_Lat     double
8                 End_Lng     double
9            Distance(mi)     double
10            Description     string
11                 Street     string
12                   City     string
13                 County     string
14                  State     string
15                Zipcode     string
16                Country     string
17               Timezone     string
18           Airport_Code     string
19      Weather_Timestamp  timestamp
20         Temperature(F)     double
21          Wind_Chill(F)     double
22            Humidity(%)     double
```

```
23           Pressure(in)     double
24         Visibility(mi)     double
25         Wind_Direction     string
26        Wind_Speed(mph)     double
27      Precipitation(in)     double
28      Weather_Condition     string
29                Amenity    boolean
30                   Bump    boolean
31               Crossing    boolean
32               Give_Way    boolean
33               Junction    boolean
34                No_Exit    boolean
35                Railway    boolean
36             Roundabout    boolean
37                Station    boolean
38                   Stop    boolean
39        Traffic_Calming    boolean
40         Traffic_Signal    boolean
41           Turning_Loop    boolean
42         Sunrise_Sunset     string
43         Civil_Twilight     string
44      Nautical_Twilight     string
45  Astronomical_Twilight     string
```

```
len(df.columns)
```

```
46
```

```
df.printSchema()
```

```
root
 |-- ID: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Severity: integer (nullable = true)
 |-- Start_Time: timestamp (nullable = true)
 |-- End_Time: timestamp (nullable = true)
 |-- Start_Lat: double (nullable = true)
 |-- Start_Lng: double (nullable = true)
 |-- End_Lat: double (nullable = true)
 |-- End_Lng: double (nullable = true)
 |-- Distance(mi): double (nullable = true)
 |-- Description: string (nullable = true)
 |-- Street: string (nullable = true)
 |-- City: string (nullable = true)
 |-- County: string (nullable = true)
 |-- State: string (nullable = true)
 |-- Zipcode: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Timezone: string (nullable = true)
 |-- Airport_Code: string (nullable = true)
 |-- Weather_Timestamp: timestamp (nullable = true)
 |-- Temperature(F): double (nullable = true)
 |-- Wind_Chill(F): double (nullable = true)
 |-- Humidity(%): double (nullable = true)
 |-- Pressure(in): double (nullable = true)
 |-- Visibility(mi): double (nullable = true)
 |-- Wind_Direction: string (nullable = true)
 |-- Wind_Speed(mph): double (nullable = true)
 |-- Precipitation(in): double (nullable = true)
 |-- Weather_Condition: string (nullable = true)
 |-- Amenity: boolean (nullable = true)
 |-- Bump: boolean (nullable = true)
 |-- Crossing: boolean (nullable = true)
 |-- Give_Way: boolean (nullable = true)
 |-- Junction: boolean (nullable = true)
 |-- No_Exit: boolean (nullable = true)
 |-- Railway: boolean (nullable = true)
 |-- Roundabout: boolean (nullable = true)
 |-- Station: boolean (nullable = true)
 |-- Stop: boolean (nullable = true)
 |-- Traffic_Calming: boolean (nullable = true)
 |-- Traffic_Signal: boolean (nullable = true)
 |-- Turning_Loop: boolean (nullable = true)
 |-- Sunrise_Sunset: string (nullable = true)
 |-- Civil_Twilight: string (nullable = true)
 |-- Nautical_Twilight: string (nullable = true)
 |-- Astronomical_Twilight: string (nullable = true)
```

```
print(f'Out of total {df.count()} rows, Displaying first 2 data
rows:')
df.limit(2).toPandas()
```

```
Out of total 7728394 rows, Displaying first 2 data rows:

23/11/30 02:21:38 WARN package: Truncated the string representation of
a plan since it was too large. This behavior can be adjusted by
setting 'spark.sql.debug.maxToStringFields'.

    ID   Source  Severity          Start_Time            End_Time
Start_Lat  \
0  A-1  Source2         3 2016-02-08 05:46:00 2016-02-08 11:00:00
39.865147
1  A-2  Source2         2 2016-02-08 06:07:59 2016-02-08 06:37:59
39.928059

   Start_Lng  End_Lat  End_Lng  Distance(mi)  ... Roundabout Station
Stop  \
0 -84.058723      NaN      NaN          0.01  ...      False   False
```

```
False
1 -82.831184      NaN       NaN          0.01  ...      False   False
False

  Traffic_Calming Traffic_Signal Turning_Loop Sunrise_Sunset
Civil_Twilight  \
0            False          False        False          Night
Night
1            False          False        False          Night
Night

  Nautical_Twilight Astronomical_Twilight
0             Night                 Night
1             Night                   Day

[2 rows x 46 columns]
```

DATA CLEANING

```
from pyspark import pandas as pd_10

/usr/lib/spark/python/pyspark/pandas/__init__.py:49: UserWarning:
'PYARROW_IGNORE_TIMEZONE' environment variable was not set. It is
required to set this environment variable to '1' in both driver and
executor sides if you use pyarrow>=2.0.0. pandas-on-Spark will set it
for you but it does not work if there is a Spark context already
launched.
  warnings.warn(

from pyspark.sql import functions as ps_10

op = 'Severity'

# find different column types and seggregate to set right defaults
string_cols =  [col[0] for col in df.dtypes if col[1] == "string" ]

num_cols = [col[0] for col in df.dtypes if col[1] == "int" or col[1]
== "double" or col[1] == "float" ]

# output could be null
num_cols.remove(op)

bool_cols = [col[0] for col in df.dtypes if col[1] == "boolean"]

print("String columns - ", string_cols)
print("Numeric columns - ", num_cols)
print("Boolean columns - ", bool_cols)

String columns -  ['ID', 'Source', 'Description', 'Street', 'City',
'County', 'State', 'Zipcode', 'Country', 'Timezone', 'Airport_Code',
'Wind_Direction', 'Weather_Condition', 'Sunrise_Sunset',
'Civil_Twilight', 'Nautical_Twilight', 'Astronomical_Twilight']
Numeric columns -  ['Start_Lat', 'Start_Lng', 'End_Lat', 'End_Lng',
'Distance(mi)', 'Temperature(F)', 'Wind_Chill(F)', 'Humidity(%)',
'Pressure(in)', 'Visibility(mi)', 'Wind_Speed(mph)',
'Precipitation(in)']
Boolean columns -  ['Amenity', 'Bump', 'Crossing', 'Give_Way',
'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station', 'Stop',
'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop']

# now we initialize empty rows for each kind of datatype

df = df.fillna("not_available", string_cols)

df = df.fillna(0 , num_cols)

# for bool column, we really can't initialize with 0 or 1

# Now we count the no of nulls to verify -
col_vs_nulls = {col:df.filter(ps_10.isnull(df[col[0]])).count() for
col in df.dtypes }
print(col_vs_nulls)



{('ID', 'string'): 0, ('Source', 'string'): 0, ('Severity', 'int'): 0,
('Start_Time', 'timestamp'): 0, ('End_Time', 'timestamp'): 0,
('Start_Lat', 'double'): 0, ('Start_Lng', 'double'): 0, ('End_Lat',
'double'): 0, ('End_Lng', 'double'): 0, ('Distance(mi)', 'double'): 0,
('Description', 'string'): 0, ('Street', 'string'): 0, ('City',
'string'): 0, ('County', 'string'): 0, ('State', 'string'): 0,
('Zipcode', 'string'): 0, ('Country', 'string'): 0, ('Timezone',
'string'): 0, ('Airport_Code', 'string'): 0, ('Weather_Timestamp',
'timestamp'): 120228, ('Temperature(F)', 'double'): 0,
('Wind_Chill(F)', 'double'): 0, ('Humidity(%)', 'double'): 0,
('Pressure(in)', 'double'): 0, ('Visibility(mi)', 'double'): 0,
('Wind_Direction', 'string'): 0, ('Wind_Speed(mph)', 'double'): 0,
('Precipitation(in)', 'double'): 0, ('Weather_Condition', 'string'):
0, ('Amenity', 'boolean'): 0, ('Bump', 'boolean'): 0, ('Crossing',
'boolean'): 0, ('Give_Way', 'boolean'): 0, ('Junction', 'boolean'): 0,
('No_Exit', 'boolean'): 0, ('Railway', 'boolean'): 0, ('Roundabout',
'boolean'): 0, ('Station', 'boolean'): 0, ('Stop', 'boolean'): 0,
('Traffic_Calming', 'boolean'): 0, ('Traffic_Signal', 'boolean'): 0,
('Turning_Loop', 'boolean'): 0, ('Sunrise_Sunset', 'string'): 0,
('Civil_Twilight', 'string'): 0, ('Nautical_Twilight', 'string'): 0,
('Astronomical_Twilight', 'string'): 0}

# col_vs_nulls = {col:df.filter(ps_10.isnan(df[col[0]])).count() for
col in df.dtypes if col[1]!='timestamp' }
# print(col_vs_nulls)
# pd_10.isna(df) won't work we need to use pysql functions to find
null/ na

state_map = {'AK': 'Alaska',
 'AL': 'Alabama',
 'AR': 'Arkansas',
 'AS': 'American Samoa',
 'AZ': 'Arizona',
 'CA': 'California',
 'CO': 'Colorado',
 'CT': 'Connecticut',
 'DC': 'District of Columbia',
 'DE': 'Delaware',
 'FL': 'Florida',
 'GA': 'Georgia',
 'GU': 'Guam',
 'HI': 'Hawaii',
 'IA': 'Iowa',
 'ID': 'Idaho',
 'IL': 'Illinois',
 'IN': 'Indiana',
 'KS': 'Kansas',
 'KY': 'Kentucky',
 'LA': 'Louisiana',
 'MA': 'Massachusetts',
 'MD': 'Maryland',
 'ME': 'Maine',
 'MI': 'Michigan',
 'MN': 'Minnesota',
 'MO': 'Missouri',
 'MP': 'Northern Mariana Islands',
 'MS': 'Mississippi',
 'MT': 'Montana',
 'NC': 'North Carolina',
 'ND': 'North Dakota',
 'NE': 'Nebraska',
 'NH': 'New Hampshire',
 'NJ': 'New Jersey',
 'NM': 'New Mexico',
 'NV': 'Nevada',
 'NY': 'New York',
 'OH': 'Ohio',
 'OK': 'Oklahoma',
 'OR': 'Oregon',
 'PA': 'Pennsylvania',
 'PR': 'Puerto Rico',
 'RI': 'Rhode Island',
 'SC': 'South Carolina',
 'SD': 'South Dakota',
 'TN': 'Tennessee',
 'TX': 'Texas',
 'UT': 'Utah',
 'VA': 'Virginia',
 'VI': 'Virgin Islands',
 'VT': 'Vermont',
 'WA': 'Washington',
 'WI': 'Wisconsin',
 'WV': 'West Virginia',
 'WY': 'Wyoming'}
```

DATA ANALYSIS

Here we find the US states which has most severe accidents since 2020

```
from pyspark.sql import SparkSession


from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def map_state_code_to_name(code):
    return state_map.get(code, None)  # Returns None if the code is
not found

map_state_udf = udf(map_state_code_to_name, StringType())

accidents_with_state_names = df.withColumn("StateName",
map_state_udf(df["State"]))

#accidents_with_state_names.select("State", "StateName").show()
unique_states_df =
accidents_with_state_names.select("StateName").distinct()
unique_states_df.show()

[Stage 118:==================================================>    (21
+ 2) / 23]

+--------------------+
|           StateName|
+--------------------+
|                Utah|
|           Minnesota|
|                Ohio|
|              Oregon|
|            Arkansas|
|               Texas|
|        North Dakota|
|        Pennsylvania|
|         Connecticut|
|            Nebraska|
|             Vermont|
|              Nevada|
|          Washington|
|            Illinois|
|            Oklahoma|
|District of Columbia|
|            Delaware|
|          New Mexico|
|       West Virginia|
|            Missouri|
+--------------------+
only showing top 20 rows
```

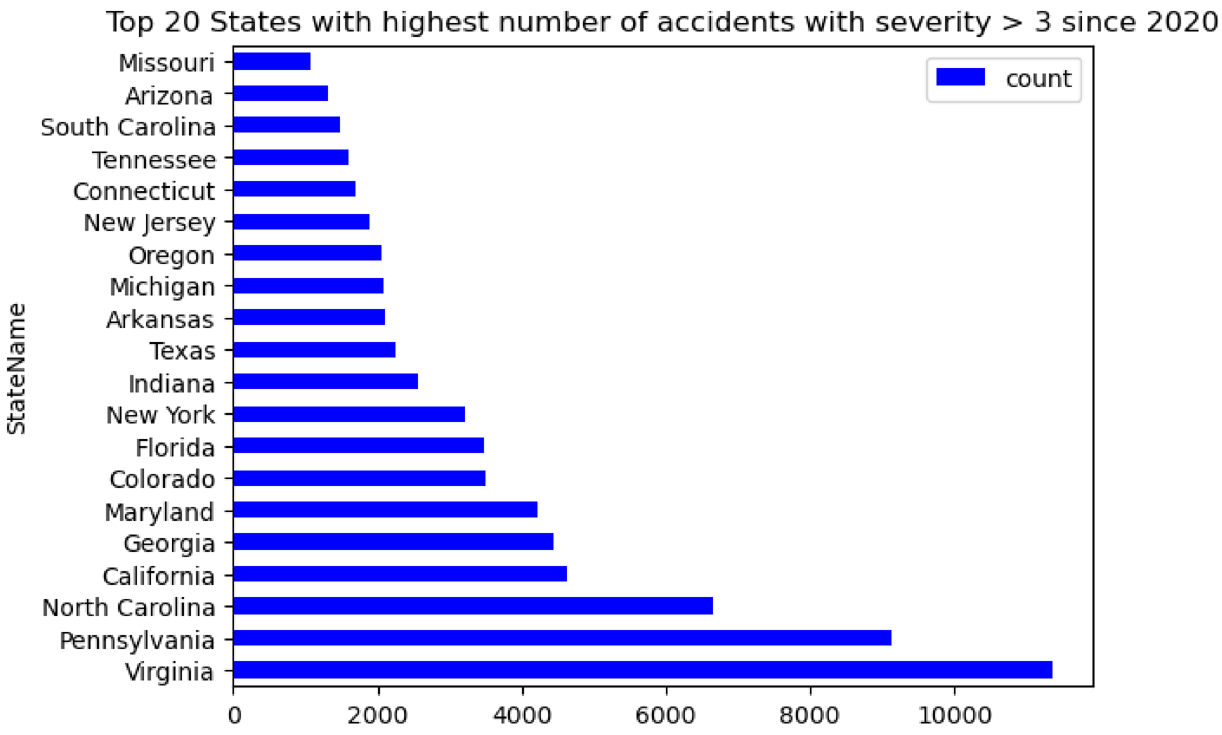
```
df = accidents_with_state_names

states_with_severe_accidents = df.select(df.StateName, df.Severity,
df.Start_Time).where(ps_10.year(df.Start_Time)>"2020").filter(df.Sever
ity > 3).groupBy(df.StateName).count().orderBy("count",
ascending=False)

states_with_severe_accidents_pd =
states_with_severe_accidents.toPandas()
```

```
print(len(states_with_severe_accidents_pd))
```

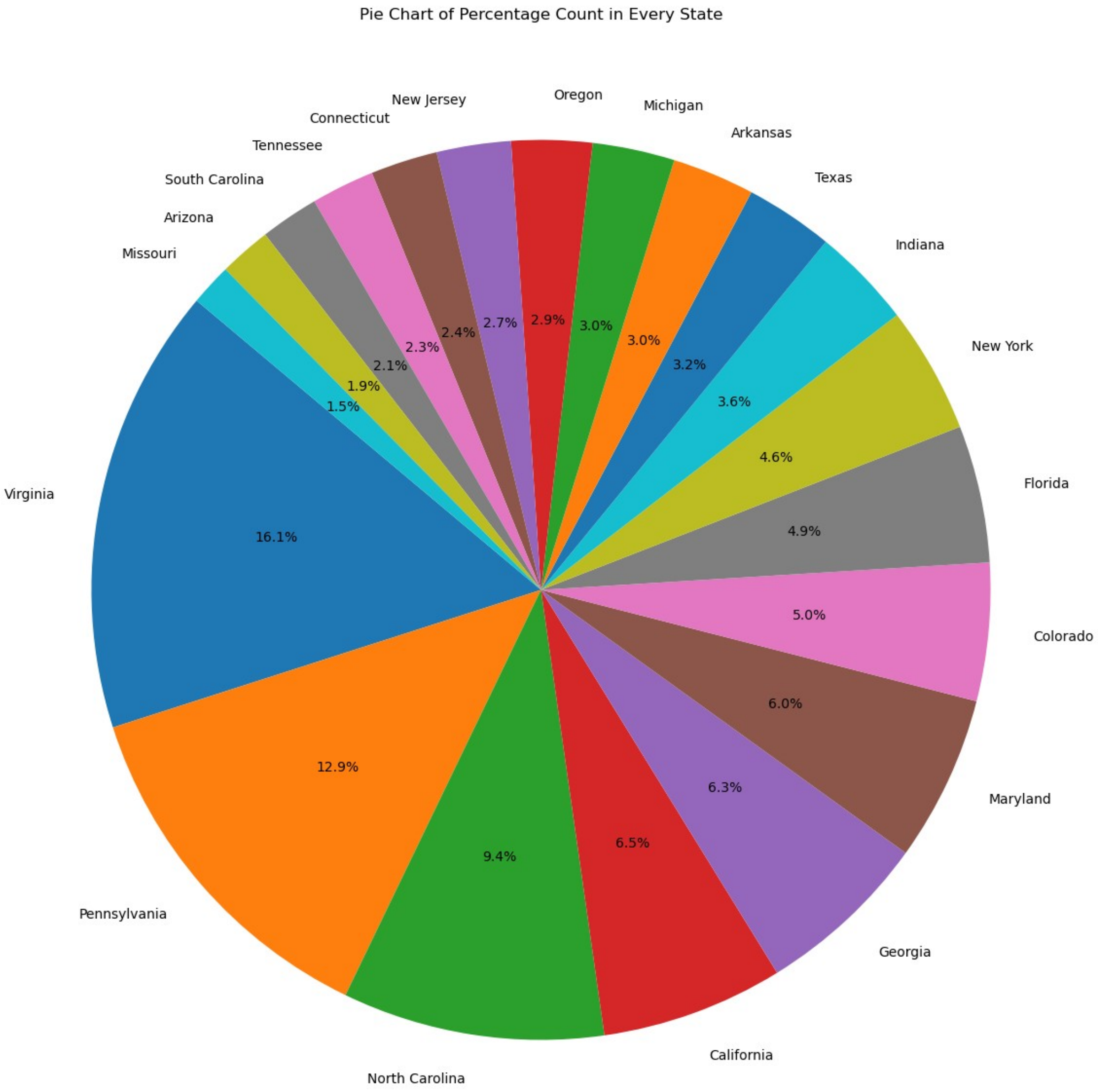
```
49
```

```
states_with_severe_accidents_pd[:20].plot(kind='barh', x='StateName',
y='count', color="blue", title='Top 20 States with highest number of
accidents with severity > 3 since 2020')
```

```
<AxesSubplot:title={'center':'Top 20 States with highest number of
accidents with severity > 3 since 2020'}, ylabel='StateName'>
```



```
states_with_severe_accidents_pd["percentage"] =
states_with_severe_accidents_pd["count"] /
states_with_severe_accidents_pd["count"].sum()

import matplotlib.pyplot as plt_10

# Create a pie chart
plt_10.figure(figsize=(15, 15))
plt_10.pie(states_with_severe_accidents_pd[:20]['percentage'],
labels=states_with_severe_accidents_pd[:20]['StateName'],
autopct='%1.1f%%', startangle=140)
plt_10.title('Pie Chart of Percentage Count in Every State')
plt_10.show()
```


Pie Chart of Percentage Count in Every State

The next idea is to understand what weather conditions influence the accidents of high severity

```
weather_condt_affecting_accidents = (
    df.select(df.Weather_Condition, df.Severity, df.Start_Time)
#     .where(ps_10.year(df.Start_Time) > "2016")
    .filter(df.Severity > 3)
    .groupBy(df.Weather_Condition)
    .count()
    .orderBy("count", ascending=False)
)
```

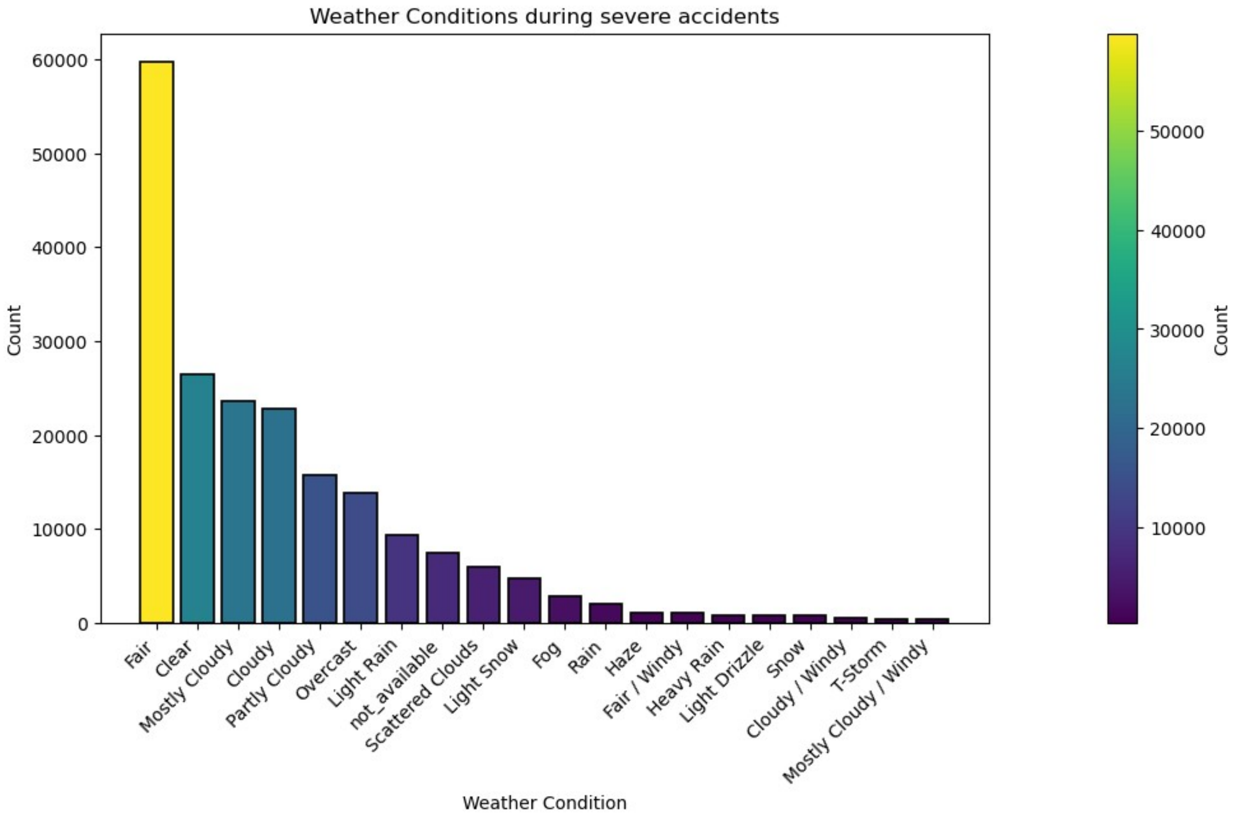
```
weather_condt_affecting_accidents_pd =
weather_condt_affecting_accidents.toPandas()




print(weather_condt_affecting_accidents_pd[:20])
```

```
        Weather_Condition  count
0                    Fair  59783
1                   Clear  26479
2           Mostly Cloudy  23555
3                  Cloudy  22828
4           Partly Cloudy  15700
5                Overcast  13867
6              Light Rain   9360
7           not_available   7429
8        Scattered Clouds   5971
9              Light Snow   4759
10                    Fog   2774
11                   Rain   2029
12                   Haze   1073
13           Fair / Windy   1014
14             Heavy Rain    753
15          Light Drizzle    750
16                   Snow    718
17         Cloudy / Windy    455
18                T-Storm    409
19  Mostly Cloudy / Windy    407
```

```
from matplotlib.cm import get_cmap
import matplotlib.pyplot as plt
import pandas as pd

def draw_bar_chart(data, title: str):
    # Remove or replace None values
    data = data[data['Weather_Condition'].notna()]  # This line
removes rows where 'Weather_Condition' is None
    # data['Weather_Condition'] =
data['Weather_Condition'].fillna('Unknown')  # Alternatively, replace
None with 'Unknown'

    # Proceed with your existing code
    weather_accident_top20 = data

    # Create a bar chart with colors based on the 'count' column
    plt.figure(figsize=(12, 6))

    # colormap
    cmap = get_cmap('viridis')
    # Normalize 'count' values to be used as color intensities
    normalize =
plt.Normalize(vmin=weather_accident_top20['count'].min(),
vmax=weather_accident_top20['count'].max())
    colors = cmap(normalize(weather_accident_top20['count']))

    # Create the bar chart with colormap
    bars = plt.bar(weather_accident_top20['Weather_Condition'],
weather_accident_top20['count'], color=colors, edgecolor='black',
linewidth=1.2)

    # Adding colorbar for reference
    sm = plt.cm.ScalarMappable(cmap=cmap, norm=normalize)
    sm.set_array([])
    cbar = plt.colorbar(sm, pad=0.1)
    cbar.set_label('Count')

    # Adding labels and title
    plt.xlabel('Weather Condition')
    plt.ylabel('Count')
    plt.title(title)
    plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for
better readability

    # Display the plot
    plt.show()

print(weather_condt_affecting_accidents_pd[:20].count)
```

```
<bound method DataFrame.count of          Weather_Condition  count
0                     Fair  59783
1                    Clear  26479
2            Mostly Cloudy  23555
3                   Cloudy  22828
4            Partly Cloudy  15700
5                 Overcast  13867
6               Light Rain   9360
7            not_available   7429
8         Scattered Clouds   5971
9               Light Snow   4759
10                     Fog   2774
11                    Rain   2029
12                    Haze   1073
13            Fair / Windy   1014
14              Heavy Rain    753
15           Light Drizzle    750
16                    Snow    718
17          Cloudy / Windy    455
18                 T-Storm    409
19   Mostly Cloudy / Windy    407>
```

```
draw_bar_chart(weather_condt_affecting_accidents_pd[:20],
title="Weather Conditions during severe accidents")
```



```
weather_severity_count = (
    df.select(df.Weather_Condition, df.Severity, df.Start_Time)
    .where(ps_10.year(df.Start_Time) > "2020")
#     .filter(df.Severity > 3)
    .groupBy(df.Weather_Condition, df.Severity)
    .count()
    .orderBy(["Severity","count"], ascending=False)
)

weather_severity_count_pd = weather_severity_count.toPandas()



print(weather_severity_count_pd[:20])
```

```
          Weather_Condition  Severity  count
0                      Fair         4  37446
1                    Cloudy         4  13436
2             Mostly Cloudy         4   8360
3             Partly Cloudy         4   6049
4             not_available         4   3498
```

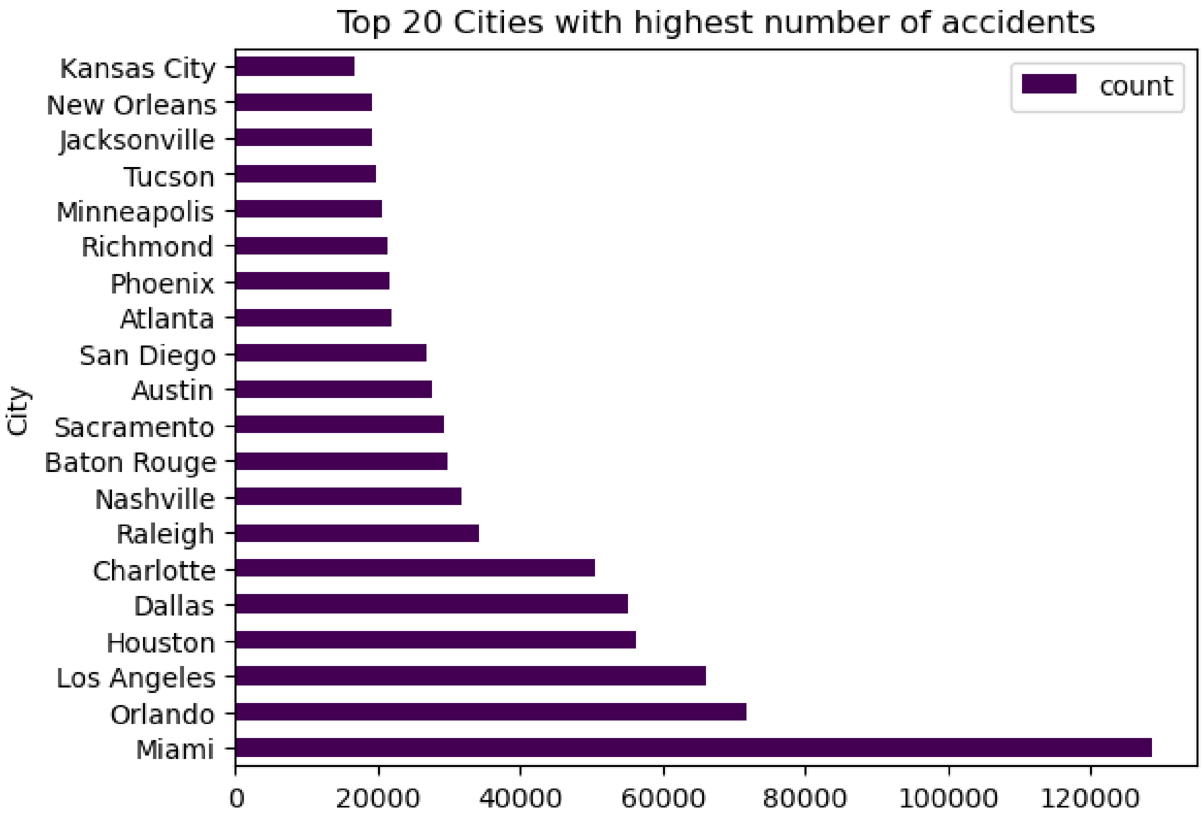
```
5                   Light Rain         4   3457
6                   Light Snow         4   2008
7                          Fog         4   1476
8                         Rain         4    756
9                 Fair / Windy         4    703
10                  Wintry Mix         4    336
11                        Haze         4    331
12                        Snow         4    324
13                  Heavy Rain         4    312
14               Light Drizzle         4    261
15              Cloudy / Windy         4    254
16                     T-Storm         4    237
17  Thunder in the Vicinity         4    235
18     Mostly Cloudy / Windy         4    204
19        Light Snow / Windy         4    162
```

```
top_20_cities = (
    df.select(df.City, df.Start_Time)
    .where(ps_10.year(df.Start_Time) > "2020")
#     .filter(df.Severity > 3)
    .groupBy(df.City)
    .count()
    .orderBy(["count"], ascending=False)
)

top_20_cities_pd = top_20_cities.toPandas()




print(top_20_cities_pd)
```

```
               City   count
0             Miami  128627
1           Orlando   71727
2       Los Angeles   66000
3           Houston   56210
4            Dallas   55086
...             ...     ...
11951  North Greece       1
11952         Leoma       1
11953        Wolsey       1
11954   District 13       1
11955     Red Cloud       1

[11956 rows x 2 columns]
```

```
top_20_cities_pd[:20].plot(kind='barh', x='City', y='count',
colormap='viridis', title='Top 20 Cities with highest number of
accidents')
```

```
<AxesSubplot:title={'center':'Top 20 Cities with highest number of
accidents'}, ylabel='City'>
```



Miami seems to be the city have the highest number of accidents, (could be because it's a party town :P)

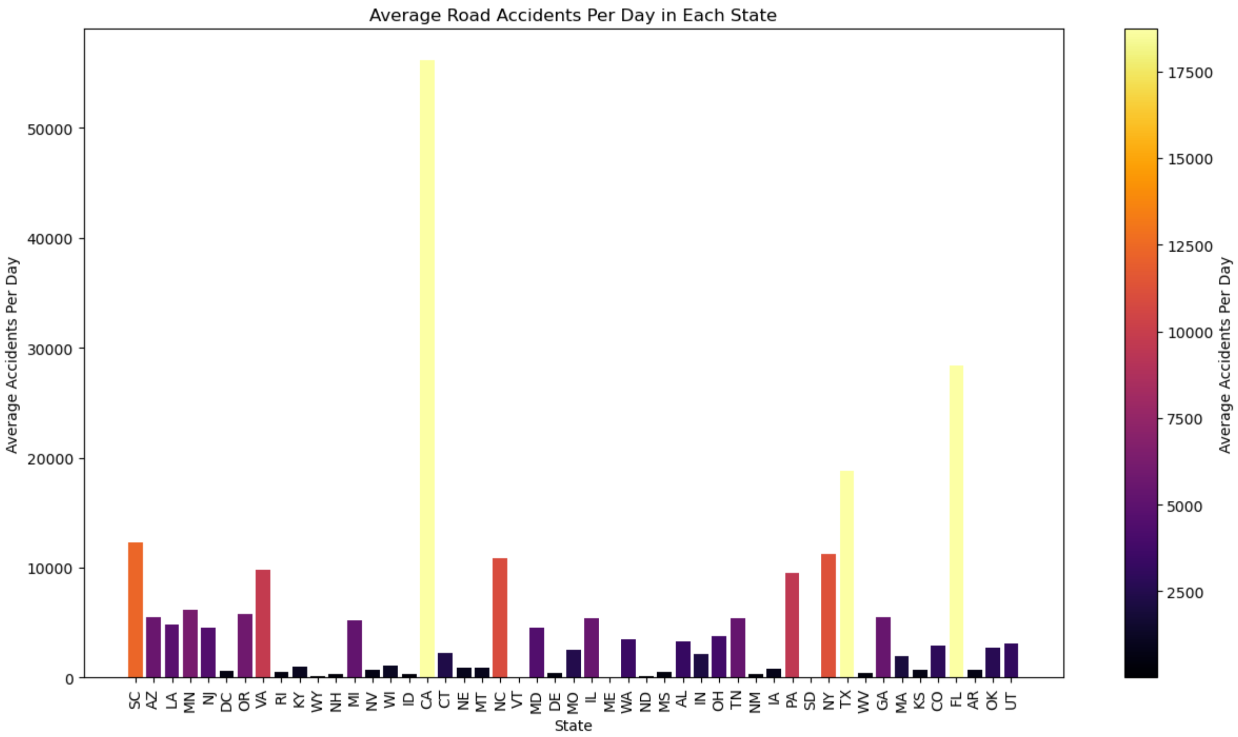
```
df = df.withColumn("Day",
ps_10.dayofmonth(ps_10.to_date(df["Start_Time"])))

df.head()
```

```
Row(ID='A-1', Source='Source2', Severity=3,
Start_Time=datetime.datetime(2016, 2, 8, 5, 46),
End_Time=datetime.datetime(2016, 2, 8, 11, 0), Start_Lat=39.865147,
Start_Lng=-84.058723, End_Lat=0.0, End_Lng=0.0, Distance(mi)=0.01,
Description='Right lane blocked due to accident on I-70 Eastbound at
Exit 41 OH-235 State Route 4.', Street='I-70 E', City='Dayton',
County='Montgomery', State='OH', Zipcode='45424', Country='US',
Timezone='US/Eastern', Airport_Code='KFFO',
Weather_Timestamp=datetime.datetime(2016, 2, 8, 5, 58),
Temperature(F)=36.9, Wind_Chill(F)=0.0, Humidity(%)=91.0,
Pressure(in)=29.68, Visibility(mi)=10.0, Wind_Direction='Calm',
Wind_Speed(mph)=0.0, Precipitation(in)=0.02, Weather_Condition='Light
Rain', Amenity=False, Bump=False, Crossing=False, Give_Way=False,
Junction=False, No_Exit=False, Railway=False, Roundabout=False,
Station=False, Stop=False, Traffic_Calming=False,
Traffic_Signal=False, Turning_Loop=False, Sunrise_Sunset='Night',
Civil_Twilight='Night', Nautical_Twilight='Night',
Astronomical_Twilight='Night', StateName='Ohio', Day=8)
```

```
# Grouping and aggregating
daily_accidents = df.groupBy("State",
"Day").agg(ps_10.count("*").alias("Accidents"))

# Calculating the average
average_accidents =
daily_accidents.groupBy("State").agg(ps_10.avg("Accidents").alias("Ave
rage Accidents Per Day"))

average_accidents.show()
```

```
[Stage 154:=====================================================>  (22
+ 1) / 23]

+-----+------------------------+
|State|Average Accidents Per Day|
+-----+------------------------+
|   SC|       12340.548387096775|
|   AZ|       5503.5161290322585|
|   LA|        4829.064516129032|
|   MN|        6196.258064516129|
|   NJ|        4539.322580645161|
|   DC|        600.9677419354839|
|   OR|       5795.4838709677415|
|   VA|         9783.90322580645|
|   RI|        547.4516129032259|
|   KY|       1040.4516129032259|
|   WY|       121.19354838709677|
|   NH|        329.4516129032258|
|   MI|        5231.967741935484|
|   NV|        698.8709677419355|
|   WI|        1118.967741935484|
|   ID|       366.96774193548384|
|   CA|        56175.25806451613|
|   CT|        2290.483870967742|
|   NE|        931.2903225806451|
|   MT|        919.2258064516129|
+-----+------------------------+
only showing top 20 rows
```

```
average_accidents_pd = average_accidents.toPandas()




import matplotlib.pyplot as plt_10

# Plot state vs average accidents per day

# Convert the 'Average Accidents Per Day' to a numpy array for easy
manipulation
values = average_accidents_pd['Average Accidents Per Day'].to_numpy()

# Normalize the values to fit the color map
norm = plt_10.Normalize(values.min(), values.max()/3)

# Choose a colormap
cmap = plt_10.cm.inferno

# Apply the colormap
colors = cmap(norm(values))

# Create the plot
plt_10.figure(figsize=(15, 8))
plt_10.bar(average_accidents_pd['State'], values, color=colors)
plt_10.colorbar(plt_10.cm.ScalarMappable(norm=norm, cmap=cmap),
ax=plt_10.gca(), label='Average Accidents Per Day')


plt_10.xlabel('State')
plt_10.ylabel('Average Accidents Per Day')
plt_10.title('Average Road Accidents Per Day in Each State')
plt_10.xticks(rotation=90)
plt_10.show()
```



```
# Sort and take the top 10 states with the highest average accidents
top_states = average_accidents_pd.sort_values(by='Average Accidents
Per Day', ascending=False).head(10)

# geo plot need to do
```

Top 20 dangerous streets for maximum accidents

```
top_20_streets = (
    df.select(df.Street, df.Severity)
    .groupBy(df.Street)
    .count()
    .orderBy(["count"], ascending=False)
)

top_20_streets_pd = top_20_streets.toPandas()



print(len(top_20_streets_pd))
```

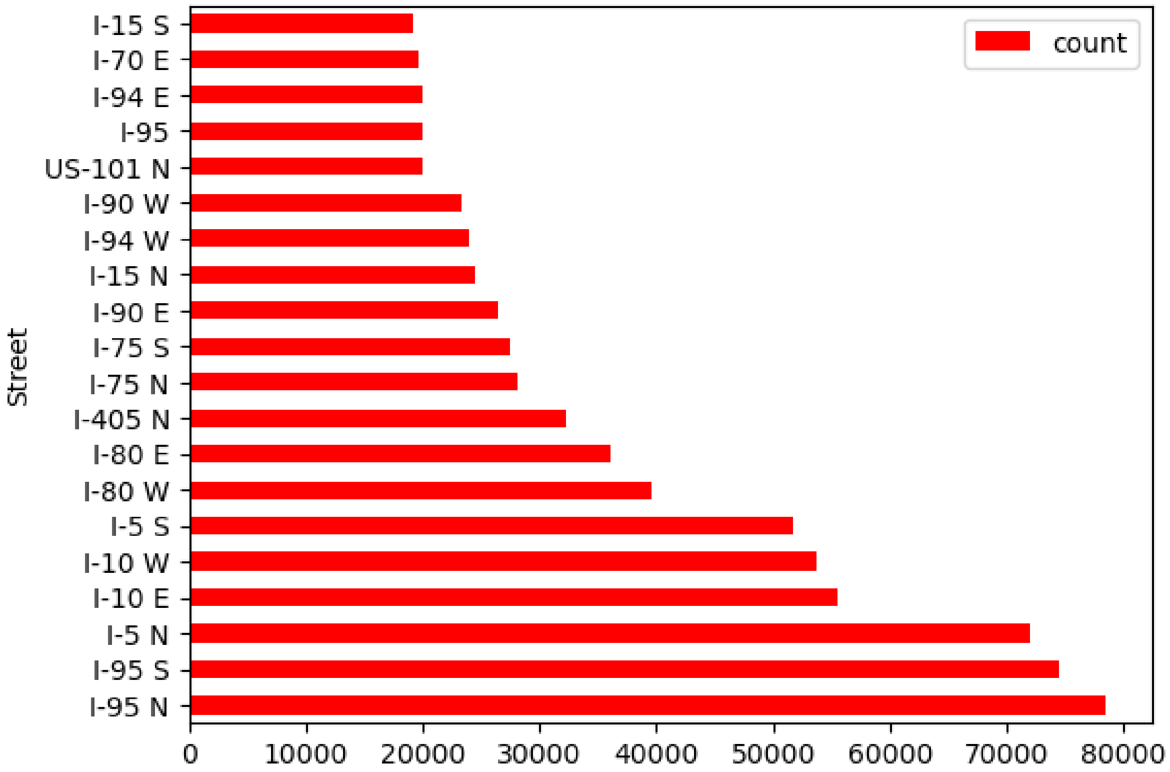
```
336307
```

```
print(top_20_streets_pd[:20])
```

```
      Street  count
0     I-95 N  78430
1     I-95 S  74528
2      I-5 N  71968
3     I-10 E  55572
4     I-10 W  53725
5      I-5 S  51781
6     I-80 W  39662
7     I-80 E  36113
8    I-405 N  32364
9     I-75 N  28166
10    I-75 S  27546
11    I-90 E  26426
12    I-15 N  24470
13    I-94 W  24003
14    I-90 W  23279
15  US-101 N  20041
16      I-95  20028
17    I-94 E  19940
18    I-70 E  19697
19    I-15 S  19230
```

```
top_20_streets_pd[:20].plot(kind='barh', x='Street', y='count',
colormap='autumn')
```

```
<AxesSubplot:ylabel='Street'>
```

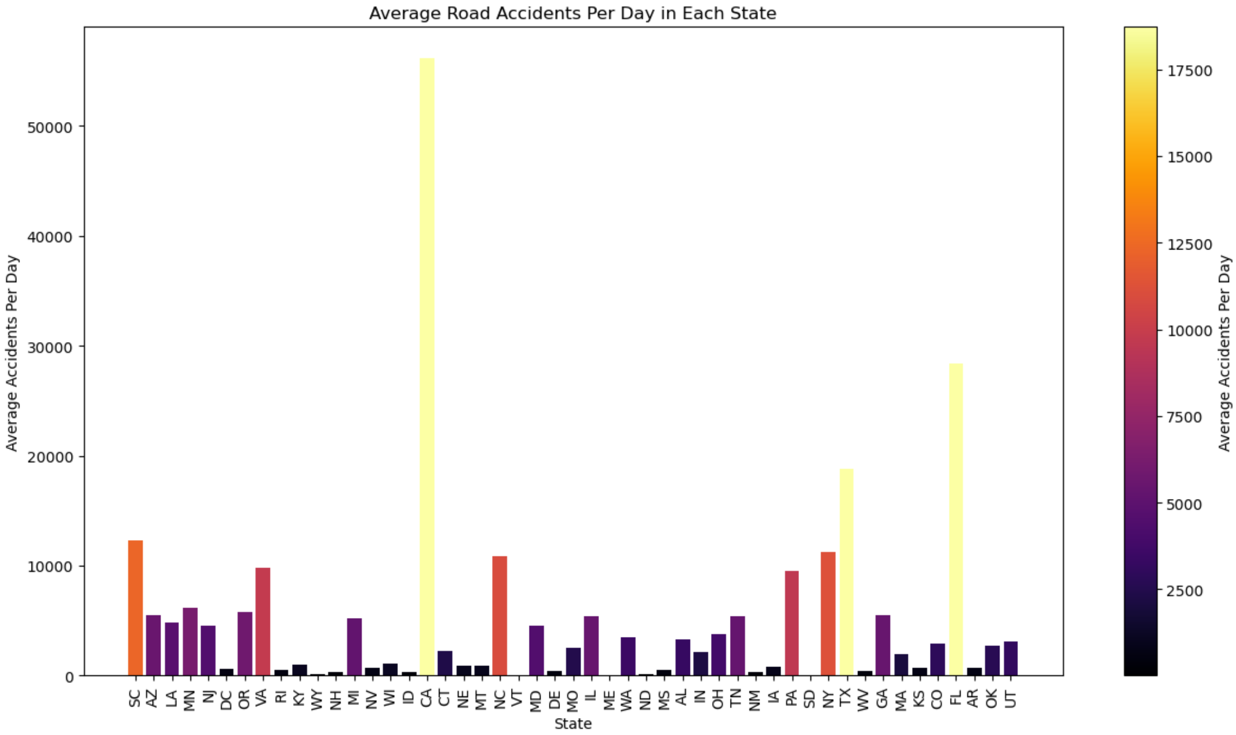


Daily Average per state

```
df = df.withColumn("Day",
ps_10.dayofmonth(ps_10.to_date(df["Start_Time"])))

df.head()
```

```
Row(ID='A-1', Source='Source2', Severity=3,
Start_Time=datetime.datetime(2016, 2, 8, 5, 46),
End_Time=datetime.datetime(2016, 2, 8, 11, 0), Start_Lat=39.865147,
Start_Lng=-84.058723, End_Lat=0.0, End_Lng=0.0, Distance(mi)=0.01,
Description='Right lane blocked due to accident on I-70 Eastbound at
Exit 41 OH-235 State Route 4.', Street='I-70 E', City='Dayton',
County='Montgomery', State='OH', Zipcode='45424', Country='US',
Timezone='US/Eastern', Airport_Code='KFFO',
Weather_Timestamp=datetime.datetime(2016, 2, 8, 5, 58),
Temperature(F)=36.9, Wind_Chill(F)=0.0, Humidity(%)=91.0,
Pressure(in)=29.68, Visibility(mi)=10.0, Wind_Direction='Calm',
Wind_Speed(mph)=0.0, Precipitation(in)=0.02, Weather_Condition='Light
Rain', Amenity=False, Bump=False, Crossing=False, Give_Way=False,
Junction=False, No_Exit=False, Railway=False, Roundabout=False,
Station=False, Stop=False, Traffic_Calming=False,
Traffic_Signal=False, Turning_Loop=False, Sunrise_Sunset='Night',
Civil_Twilight='Night', Nautical_Twilight='Night',
Astronomical_Twilight='Night', StateName='Ohio', Day=8)

# Grouping and aggregating
daily_accidents = df.groupBy("State",
"Day").agg(ps_10.count("*").alias("Accidents"))


# Calculating the average
average_accidents =
daily_accidents.groupBy("State").agg(ps_10.avg("Accidents").alias("Average Accidents Per Day"))

# Display result
average_accidents.show()

[Stage 175:=====================================================>  (22
+ 1) / 23]

+-----+-------------------------+
|State|Average Accidents Per Day|
+-----+-------------------------+
|   SC|       12340.548387096775|
|   AZ|       5503.5161290322585|
|   LA|        4829.064516129032|
|   MN|        6196.258064516129|
|   NJ|        4539.322580645161|
|   DC|        600.9677419354839|
|   OR|       5795.4838709677415|
|   VA|         9783.90322580645|
|   RI|        547.4516129032259|
|   KY|       1040.4516129032259|
|   WY|       121.19354838709677|
|   NH|        329.4516129032258|
|   MI|        5231.967741935484|
|   NV|        698.8709677419355|
|   WI|        1118.967741935484|
|   ID|       366.96774193548384|
|   CA|        56175.25806451613|
|   CT|        2290.483870967742|
|   NE|        931.2903225806451|
|   MT|        919.2258064516129|
+-----+-------------------------+
only showing top 20 rows




average_accidents_pd = average_accidents.toPandas()
```

```python
import matplotlib.pyplot as plt_10

# Plot state vs average accidents per day

# Convert the 'Average Accidents Per Day' to a numpy array for easy
manipulation
values = average_accidents_pd['Average Accidents Per Day'].to_numpy()

# Normalize the values to fit the color map
norm = plt_10.Normalize(values.min(), values.max()/3)

# colormap - inferno, viridis, plasma
cmap = plt_10.cm.inferno

# Apply the colormap on the normalized values
colors = cmap(norm(values))

# Create the plot
plt_10.figure(figsize=(15, 8))
plt_10.bar(average_accidents_pd['State'], values, color=colors)
plt_10.colorbar(plt_10.cm.ScalarMappable(norm=norm, cmap=cmap),
ax=plt_10.gca(), label='Average Accidents Per Day')


plt_10.xlabel('State')
plt_10.ylabel('Average Accidents Per Day')
plt_10.title('Average Road Accidents Per Day in Each State')
plt_10.xticks(rotation=90)
plt_10.show()
```



```
temp = df.select(df.ID, df.Severity).filter(df.Severity > 3)

temp.collect()[:5]
```

```
[Row(ID='A-620', Severity=4),
 Row(ID='A-1198', Severity=4),
 Row(ID='A-1902', Severity=4),
 Row(ID='A-4144', Severity=4),
 Row(ID='A-4965', Severity=4)]
```