How can we control the increasing number of accidents in New York?

In [1]:
```
import json
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import base64
```

Introduction

Business Context. The city of New York has seen a rise in the number of accidents on the roads in the city. They would like to know if the number of accidents have increased in the last few weeks. For all the reported accidents, they have collected details for each accident and have been maintaining records for the past year and a half (from January 2018 to August 2019).

The city has contracted you to build visualizations that would help them identify patterns in accidents, which would help them take preventive actions to reduce the number of accidents in the future. They have certain parameters like borough, time of day, reason for accident, etc. Which they care about and which they would like to get specific information on.

Business Problem. Your task is to format the given data and provide visualizations that would answer the specific questions the client has, which are mentioned below.

Analytical Context. You are given a CSV file (stored in the already created `data` folder) containing details about each accident like date, time, location of the accident, reason for the accident, types of vehicles involved, injury and death count, etc. The delimiter in the given CSV file is `;` instead of the default `,` . You will be performing the following tasks on the data:

1. Extract additional borough data stored in a JSON file
2. Read, transform, and prepare data for visualization
3. Perform analytics and construct visualizations of the data to identify patterns in the dataset

The client has a specific set of questions they would like to get answers to. You will need to provide visualizations to accompany these:

1. How have the number of accidents fluctuated over the past year and a half? Have they increased over the time?
2. For any particular day, during which hours are accidents most likely to occur?
3. Are there more accidents on weekdays than weekends?
4. What are the accidents count-to-area ratio per borough? Which boroughs have disproportionately large numbers of accidents for their size?
5. For each borough, during which hours are accidents most likely to occur?
6. What are the top 5 causes of accidents in the city?

7. What types of vehicles are most involved in accidents per borough?
8. What types of vehicles are most involved in deaths?

Note: To solve this extended case, please read the function docstrings **very carefully**. They contain information that you will need! Also, please don't include `print()` statements inside your functions (they will most likely produce an error in the test cells).

Fetching the relevant data

The client has requested analysis of the accidents-to-area ratio for boroughs. Borough data is stored in a JSON file in the `data` folder (this file was created using data from Wikipedia).

Question

Use the function `json.load()` to load the file `borough_data.json` as a dictionary.

Answer. One possible solution is given below:

In [2]:

```
with open('data/borough_data.json') as f:
    borough_data=json.load(f)
borough_data
```

Out[2]:

```
{'the bronx': {'name': 'the bronx', 'population': 1471160.0, 'area': 42.1},
 'brooklyn': {'name': 'brooklyn', 'population': 2648771.0, 'area': 70.82},
 'manhattan': {'name': 'manhattan', 'population': 1664727.0, 'area': 22.83},
 'queens': {'name': 'queens', 'population': 2358582.0, 'area': 108.53},
 'staten island': {'name': 'staten island',
  'population': 479458.0,
  'area': 58.37}}
```

Question

Similarly, use the `pandas` function `read_csv()` to load the file `accidents.csv` as a DataFrame. Name this DataFrame `df`.

Answer. One possible solution is given below:

In [3]:

```
with open('data/accidents.csv') as f:
    df=pd.read_csv(f, delimiter=';')
```

In [4]:

```
df.head(20)
```

Out[4]:

			ZIP			ON STREET	NUMBER OF	NUMBER OF	NUMBER OF	CONTRIBUTING

In [5]:
```
for col in df[['NUMBER OF PEDESTRIANS INJURED',
       'NUMBER OF PEDESTRIANS KILLED', 'NUMBER OF CYCLIST INJURED',
       'NUMBER OF CYCLIST KILLED']]:
  print(df[col].unique())
```

```
[0 1 2 6 5 3 4]
[0 1 2]
[0 1 2 3]
[0 1]
```

Overview of the data

Let's go through the columns present in the DataFrame:

In [6]:
```
df.columns
```

Out[6]:
```
Index(['DATE', 'TIME', 'BOROUGH', 'ZIP CODE', 'LATITUDE', 'LONGITUDE',
       'ON STREET NAME', 'NUMBER OF PEDESTRIANS INJURED',
       'NUMBER OF PEDESTRIANS KILLED', 'NUMBER OF CYCLIST INJURED',
       'NUMBER OF CYCLIST KILLED', 'NUMBER OF MOTORIST INJURED',
       'NUMBER OF MOTORIST KILLED', 'CONTRIBUTING FACTOR VEHICLE 1',
       'CONTRIBUTING FACTOR VEHICLE 2', 'CONTRIBUTING FACTOR VEHICLE 3',
       'CONTRIBUTING FACTOR VEHICLE 4', 'CONTRIBUTING FACTOR VEHICLE 5',
       'COLLISION_ID', 'VEHICLE TYPE CODE 1', 'VEHICLE TYPE CODE 2',
       'VEHICLE TYPE CODE 3', 'VEHICLE TYPE CODE 4', 'VEHICLE TYPE CODE 5'],
      dtype='object')
```

We have the following columns:

1. **BOROUGH**: The borough in which the accident occurred
2. **COLLISION_ID**: A unique identifier for this collision
3. **CONTRIBUTING FACTOR VEHICLE (1, 2, 3, 4, 5)**: Reasons for the accident
4. **CROSS STREET NAME**: Nearest cross street to the location of the accident
5. **DATE**: Date of the accident
6. **TIME**: Time of the accident
7. **LATITUDE**: Latitude of the accident
8. **LONGITUDE**: Longitude of the accident
9. **NUMBER OF (CYCLISTS, MOTORISTS, PEDESTRIANS) INJURED**: Injuries by category
10. **NUMBER OF (CYCLISTS, MOTORISTS, PEDESTRIANS) KILLED**: Deaths by category
11. **ON STREET NAME**: Street where the accident occurred

Group the available accident data by month.

Hint: You may find the `pandas` functions `pd.to_datetime()` and `dt.to_period()` useful.

In [7]:

```
def ex_2(df):
    """
    Group accidents by month

    Arguments:
    `df`: A pandas DataFrame

    Outputs:
    `monthly_accidents`: The grouped Series
    """

    # YOUR CODE HERE
    df["DATE"] = pd.to_datetime(df["DATE"])
    df["MONTH"] = df["DATE"].dt.to_period("m")

    monthly_accidents = df.groupby(df["MONTH"]).size()


    return monthly_accidents
```

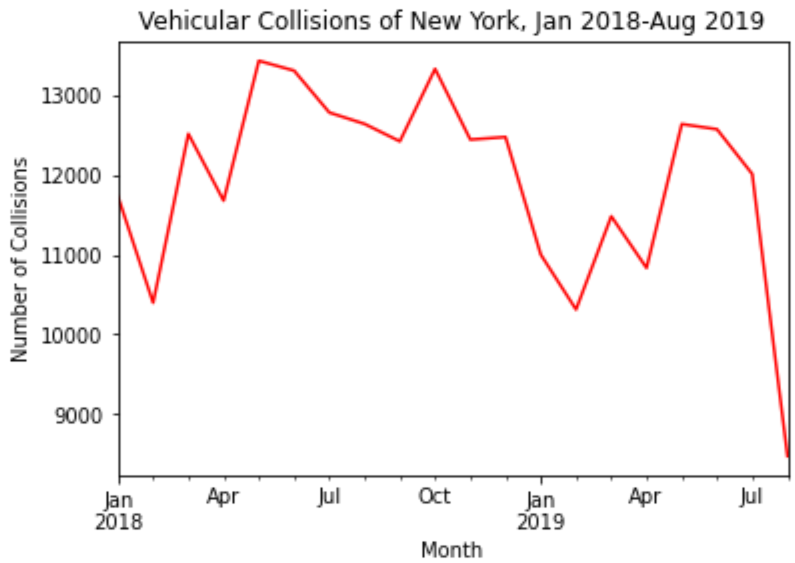
2.2

2.2.1

Generate a line plot of accidents over time.

In [8]:

```
# YOUR CODE HERE
ex_2(df).plot(color = "red")

plt.xlabel("Month")
plt.ylabel("Number of Collisions")
plt.title("Vehicular Collisions of New York, Jan 2018-Aug 2019");
```



2.2.2

Has the number of accidents increased over the past year and a half?

ANSWER

No

Exercise 3

From the plot above, which months seem to have the least number of accidents? What do you think are the reasons behind this?

ANSWER

January, February, and March. It may be because less people are driving due to the weather or for some reason less is reported.

Exercise 4

4.1

Create a new column `HOUR` based on the data from the `TIME` column.

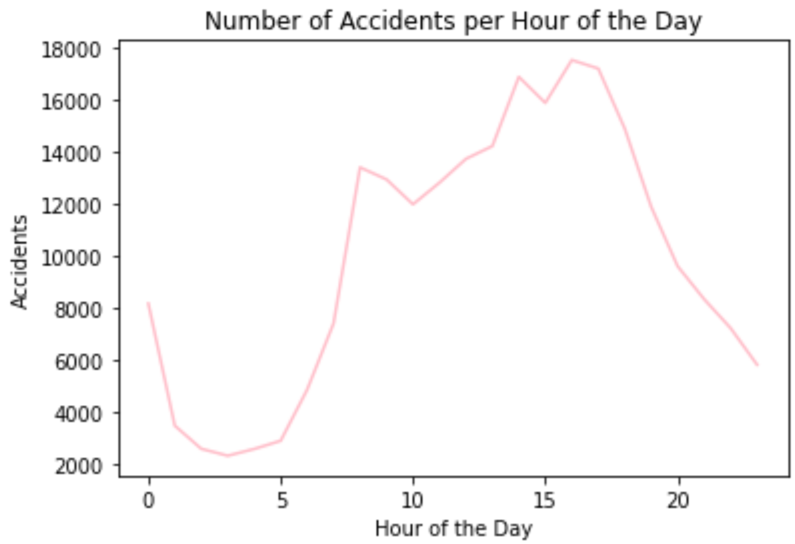
Hint: You may find the `dt.hour` accessor useful.

In [9]:
```python
# workspace
df["TIME"] = pd.to_datetime(df["TIME"])
df["TIME"]
```

Out[9]:
```
0        2021-05-21 12:12:00
1        2021-05-21 16:30:00
2        2021-05-21 19:30:00
3        2021-05-21 13:10:00
4        2021-05-21 22:40:00
                 ...
238517   2021-05-21 15:00:00
238518   2021-05-21 14:00:00
238519   2021-05-21 13:05:00
238520   2021-05-21 17:45:00
238521   2021-05-21 16:38:00
Name: TIME, Length: 238522, dtype: datetime64[ns]
```

In [10]:
```python
# workspace
df["TIME"] = pd.to_datetime(df["TIME"])
df["HOUR"] = df["TIME"].dt.hour
hourly_accidents = df.groupby(df["HOUR"]).size()
hourly_accidents
```

Out[10]:
```
HOUR
0      8160
1      3460
2      2570
3      2302
4      2562
5      2878
6      4844
7      7399
8     13403
9     12939
10    11981
11    12815
12    13731
13    14224
14    16889
15    15886
16    17536
17    17209
18    14899
19    11885
20     9597
21     8330
22     7216
23     5807
```



4.2.2

How does the number of accidents vary throughout a single day?

In [13]:

```
def accidents_p_h_variance(df):
    df["TIME"] = pd.to_datetime(df["TIME"])
    df["HOUR"] = df["TIME"].dt.hour

    return df["HOUR"]

accidents_per_hour_var = accidents_p_h_variance(df).var()
print("The number of accidents throughout the day vary by {:.2f}".format(accidents_per_hour_var))
```

```
The number of accidents throughout the day vary by 31.44
```

Exercise 5

In the above question we have aggregated the number accidents per hour disregarding the date and place of occurrence. What criticism would you give to this approach?

ANSWER

Although the result may be able to give a result that can inform from a quantitative perspective, there is a lot of information missing, like how those days may look from month to month, season to season, location to location, etc.

Exercise 6

6.1

Calculate the number of accidents by day of the week.

Hint: You may find the `dt.weekday` accessor useful.

In [14]:

```
# datetime to weekday
df["DATE"] = pd.to_datetime(df["DATE"])
df["WEEKDAY"] = df["DATE"].dt.weekday
df['DAY_OF_WEEK'] = df["DATE"].dt.day_name()
df["DAY_OF_WEEK"]
```

Out[14]:

```
0         Wednesday
1           Tuesday
2          Thursday
3            Sunday
4           Tuesday
            ...
238517     Saturday
238518     Thursday
238519     Saturday
238520       Monday
238521      Tuesday
Name: DAY_OF_WEEK, Length: 238522, dtype: object
```

In [15]:

```
def ex_6(df):
    """
    Group accidents by day of the week

    Arguments:
    `df`: A pandas DataFrame

    Outputs:
    `weekday_accidents`: The grouped Series
    """

    # YOUR CODE HERE
    df["DATE"] = pd.to_datetime(df["DATE"])
    df["WEEKDAY"] = df["DATE"].dt.weekday
    weekday_accidents = df.groupby(df["WEEKDAY"]).size()

    return weekday_accidents
```

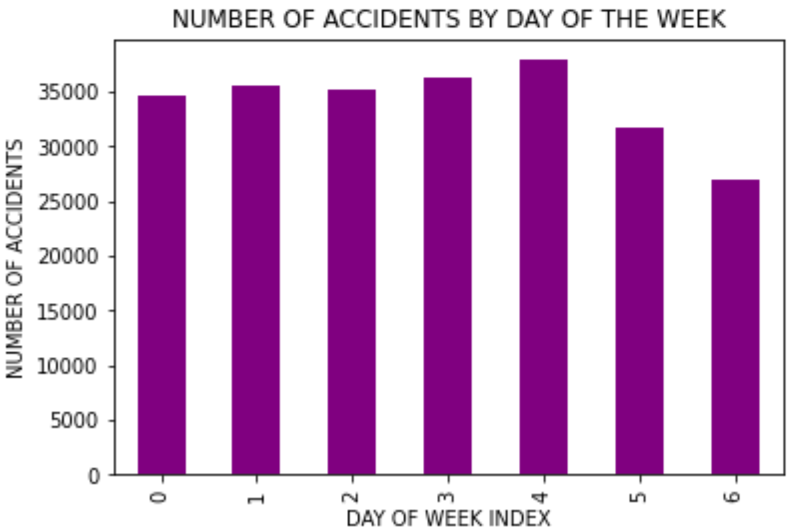
6.2

6.2.1

Plot a bar graph based on the accidents count by day of the week.

In [16]:

```
# YOUR CODE HERE
ex_6(df).plot.bar(color = "purple")

plt.xlabel("DAY OF WEEK INDEX")
plt.ylabel("NUMBER OF ACCIDENTS")
plt.title("NUMBER OF ACCIDENTS BY DAY OF THE WEEK");
```

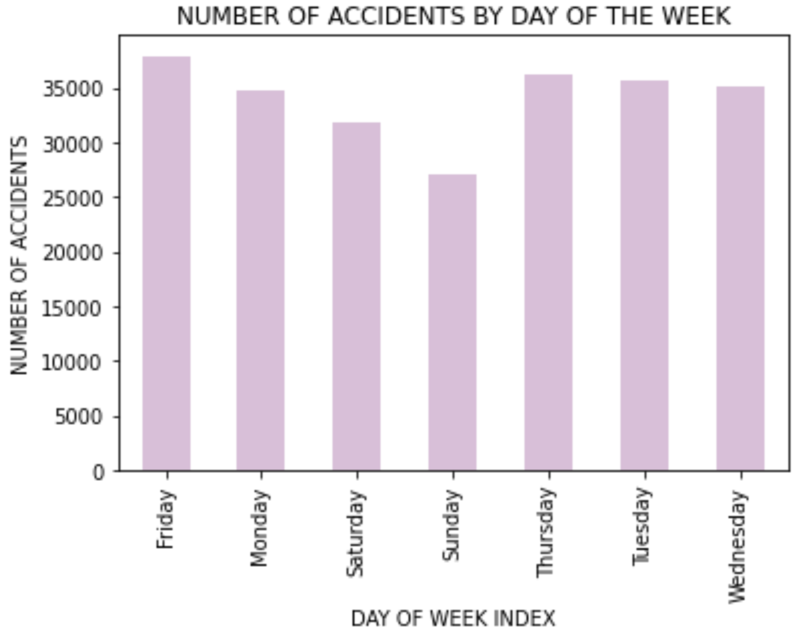


In [17]:

```
def ex_6_name(df): # function for datetime to weekday name

    df["DATE"] = pd.to_datetime(df["DATE"])
    df['DAY_OF_WEEK'] = df["DATE"].dt.day_name()
    weekday_name_accidents = df.groupby(df["DAY_OF_WEEK"]).size()

    return weekday_name_accidents
```

In [18]:

```
ex_6_name(df).plot.bar(color = "thistle") # same but with name of the day of the week in alphabetical order

plt.xlabel("DAY OF WEEK INDEX")
plt.ylabel("NUMBER OF ACCIDENTS")
plt.title("NUMBER OF ACCIDENTS BY DAY OF THE WEEK");
```



6.2.2

How does the number of accidents vary throughout a single week?

In [19]:
```
def accidents_weekday_variance(df):
    df["DATE"] = pd.to_datetime(df["DATE"])
    df["WEEKDAY"] = df["DATE"].dt.weekday

    return df["WEEKDAY"]

accidents_per_week_var = accidents_weekday_variance(df).var()
print("The number of accidents from weekday to weekday vary by {:.2f}".format(accidents_per_week_var))
```

```
The number of accidents from weekday to weekday vary by 3.75
```

Exercise 7

7.1

Calculate the total number of accidents for each borough.

In [20]:
```
# workspace
boroughs = df.groupby(df["BOROUGH"]).size()
boroughs
```

Out[20]:
```
BOROUGH
```

```
BOROUGH
BRONX             37709
BROOKLYN          76253
MANHATTAN         48749
QUEENS            67120
STATEN ISLAND      8691
```

In [21]:

```python
def ex_7_1(df):
    """
    Group accidents by borough

    Arguments:
    `df`: A pandas DataFrame

    Outputs:
    `boroughs`: The grouped Series
    """

    # YOUR CODE HERE
    boroughs = df.groupby(df["BOROUGH"]).size()

    return boroughs
```

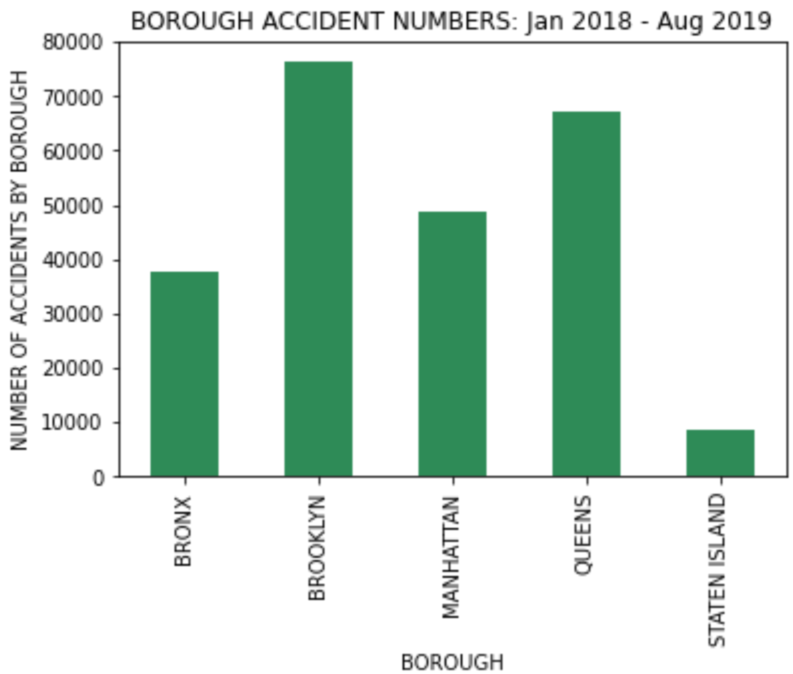
7.2

7.2.1

Plot a bar graph of the previous data.

In [22]:

```python
# YOUR CODE HERE
ex_7_1(df).plot.bar(color = "seagreen")

plt.xlabel("BOROUGH")
plt.ylabel("NUMBER OF ACCIDENTS BY BOROUGH")
plt.title("BOROUGH ACCIDENT NUMBERS: Jan 2018 - Aug 2019");
```



7.2.2

What do you notice in the plot?

ANSWER

- All the boroughs vary from each other by ~ 10k or more
- Brooklyn has the most accidents with around 75k
- Staten Island has a signigicantly less amount than any other borough with ~ < 10k
- The Bronx has almost half as much as Brooklyn
- Manhattan has less than Queens but more than the Bronx

7.3 (hard)

How about per square mile? Calculate the number accidents per square mile for each borough.

Hint: You will have to update the keys in the borough dictionary to match the names in the DataFrame.

In [23]:
```
# workspace
borough_data
```

Out[23]:
```
{'the bronx': {'name': 'the bronx', 'population': 1471160.0, 'area': 42.1},
 'brooklyn': {'name': 'brooklyn', 'population': 2648771.0, 'area': 70.82},
```

```
 'manhattan': {'name': 'manhattan', 'population': 1664727.0, 'area': 22.83},
 'queens': {'name': 'queens', 'population': 2358582.0, 'area': 108.53},
 'staten island': {'name': 'staten island',
  'population': 479458.0,
```

In [24]:

```python
# workspace
borough_data["the bronx"]["area"]
```

Out[24]:

```
42.1
```

In [25]:

```python
# workspace
bd_df = pd.DataFrame.from_dict(borough_data, orient = "index")
bd_df
```

Out[25]:

	name	population	area
the bronx	the bronx	1471160.0	42.10
brooklyn	brooklyn	2648771.0	70.82
manhattan	manhattan	1664727.0	22.83
queens	queens	2358582.0	108.53
staten island	staten island	479458.0	58.37

In [26]:

```python
# workspace
boroughs = ex_7_1(df)
borough_frame = pd.DataFrame(boroughs)

bd_df = pd.DataFrame.from_dict(borough_data, orient = "index")
borough_frame["accidents_per_sq_mi"] = (boroughs.values)/(bd_df["area"].values)
borough_frame["accidents_per_sq_mi"]
```

Out[26]:

```
BOROUGH
BRONX             895.700713
BROOKLYN         1076.715617
MANHATTAN        2135.304424
QUEENS            618.446512
STATEN ISLAND     148.894980
Name: accidents_per_sq_mi, dtype: float64
```

In [27]:

```
def ex_7_3(df, borough_data):
    """
    Calculate accidents per sq mile for each borough

    Arguments:
    `borough_frame`: A pandas DataFrame with the count of accidents per borough
    `borough_data`: A python dictionary with population and area data for each borough


    Outputs:
    `borough_frame`: The same `borough_frame` DataFrame used as input, only with an
    additional column called `accidents_per_sq_mi` that results from dividing
    the number of accidents in each borough by its area. Please call this new column
    exactly `accidents_per_sq_mi` - otherwise the test cells will throw an error.
    """

    boroughs = ex_7_1(df)
    borough_frame = pd.DataFrame(boroughs)
    borough_frame.columns = ["accidents"]

    # YOUR CODE HERE
    bd_df = pd.DataFrame.from_dict(borough_data, orient = "index")
    borough_frame["accidents_per_sq_mi"] = (boroughs.values)/(bd_df["area"].values)

    return borough_frame # This must be a DataFrame, NOT a Series
```

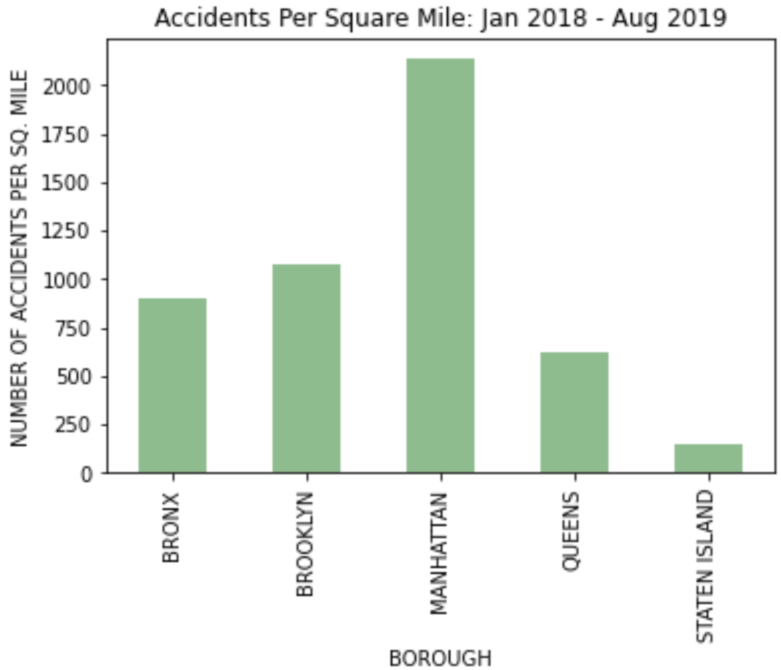
7.4

7.4.1

Plot a bar graph of the accidents per square mile per borough with the data you just calculated.

In [28]:

```
# YOUR CODE HERE
ex_7_3(df, borough_data)["accidents_per_sq_mi"].plot.bar(color = "darkseagreen")

plt.xlabel("BOROUGH")
plt.ylabel("NUMBER OF ACCIDENTS PER SQ. MILE")
plt.title("Accidents Per Square Mile: Jan 2018 - Aug 2019");
```



7.4.2

What can you conclude?

ANSWER

- The Accidents Per Square Mile vary from Total Accidents
- Each borough's accidents per square mile vary from each other
- Bronx and Brooklyn are closest to each other than any other borough
- Staten Island is still significantly lower than the other boroughs
- Manhattan is significantly higher (as opposed to Brooklyn, in the prior graph); maybe due to density

Exercise 8

8.1

Create a Series of the number of accidents per hour and borough.

In [29]:

```python
# workspace

df["TIME"] = pd.to_datetime(df["TIME"])
df["HOUR"] = df["TIME"].dt.hour
hb = df.groupby(["BOROUGH", "HOUR"]).size()
hb
```

Out[29]:

```
BOROUGH        HOUR
BRONX          0       1329
               1        529
               2        402
               3        361
               4        418
                       ...
STATEN ISLAND  19       415
               20       367
               21       268
               22       224
               23       174
Length: 120, dtype: int64
```

In [30]:

```python
# workspace

df["TIME"] = pd.to_datetime(df["TIME"])
df["HOUR"] = df["TIME"].dt.hour
hb = df.groupby(["BOROUGH", "HOUR"]).size()
hb_leveled = hb.groupby(level = 1)
hb_leveled.head(10)
```

Out[30]:

```
BOROUGH        HOUR
BRONX          0       1329
               1        529
               2        402
               3        361
               4        418
                       ...
STATEN ISLAND  19       415
               20       367
               21       268
               22       224
               23       174
Length: 120, dtype: int64
```

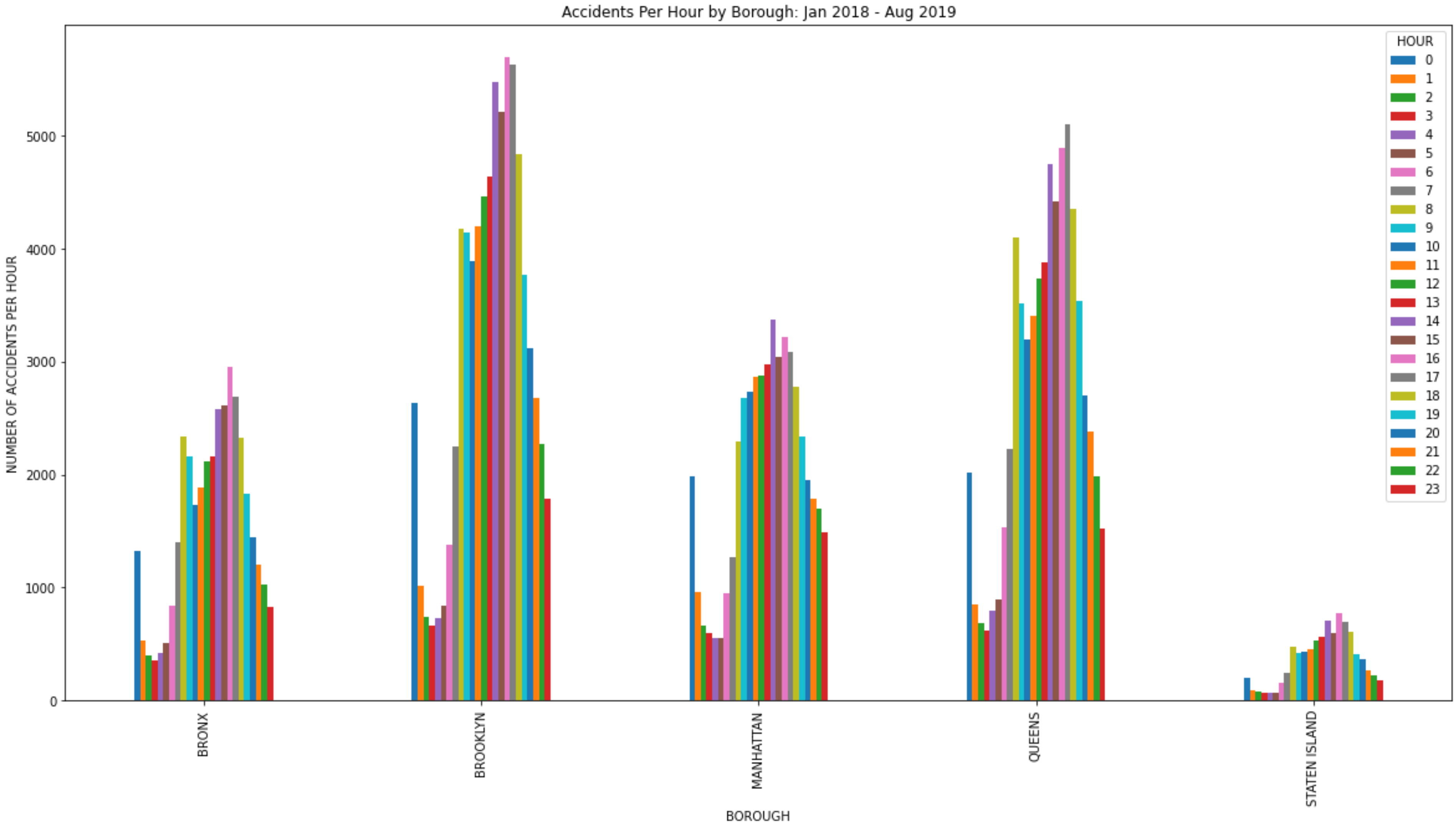
In [31]:
```
def ex_8_1(df):
    """
    Calculate accidents per hour for each borough

    Arguments:
    `df`: A pandas DataFrame


    Outputs:
    `bor_hour`: A Series. This should be the result of doing groupby by borough
    and hour.
    """

    # YOUR CODE HERE
    df["TIME"] = pd.to_datetime(df["TIME"])
    df["HOUR"] = df["TIME"].dt.hour
    bor_hour = df.groupby(["BOROUGH", "HOUR"]).size()

    return bor_hour
```

8.2

8.2.1

Plot a bar graph for each borough showing the number of accidents for each hour of the day.

In [32]:
```
# YOUR CODE HERE
ex_8_1(df).unstack().plot.bar(legend = True, figsize = (20,10))

plt.xlabel("BOROUGH")
plt.ylabel("NUMBER OF ACCIDENTS PER HOUR")
plt.title("Accidents Per Hour by Borough: Jan 2018 - Aug 2019");
```



8.2.2

Which hours have the most accidents for each borough?

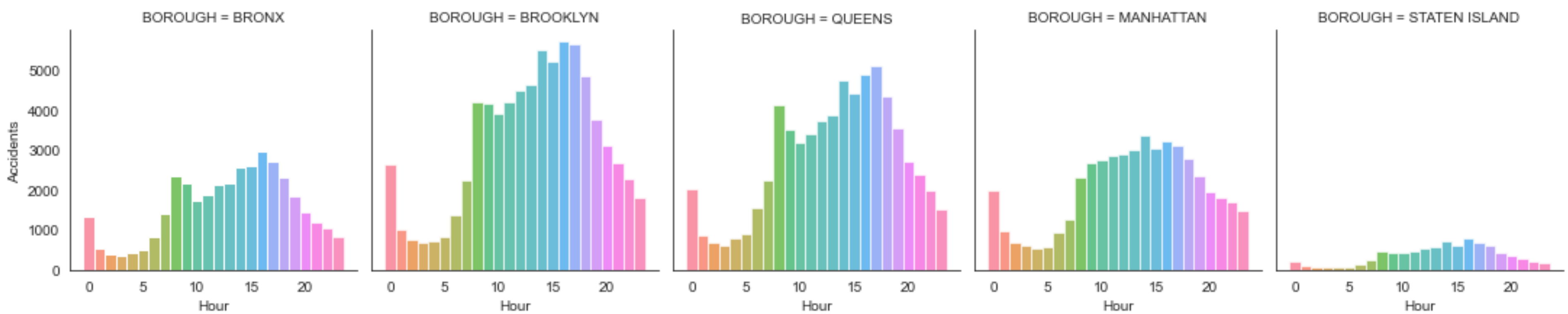
Hint: You can use `sns.FacetGrid` to create a grid of plots with the hourly data of each borough.

In [33]:

```
df["TIME"] = pd.to_datetime(df["TIME"])
df["HOUR"] = df["TIME"].dt.hour
```

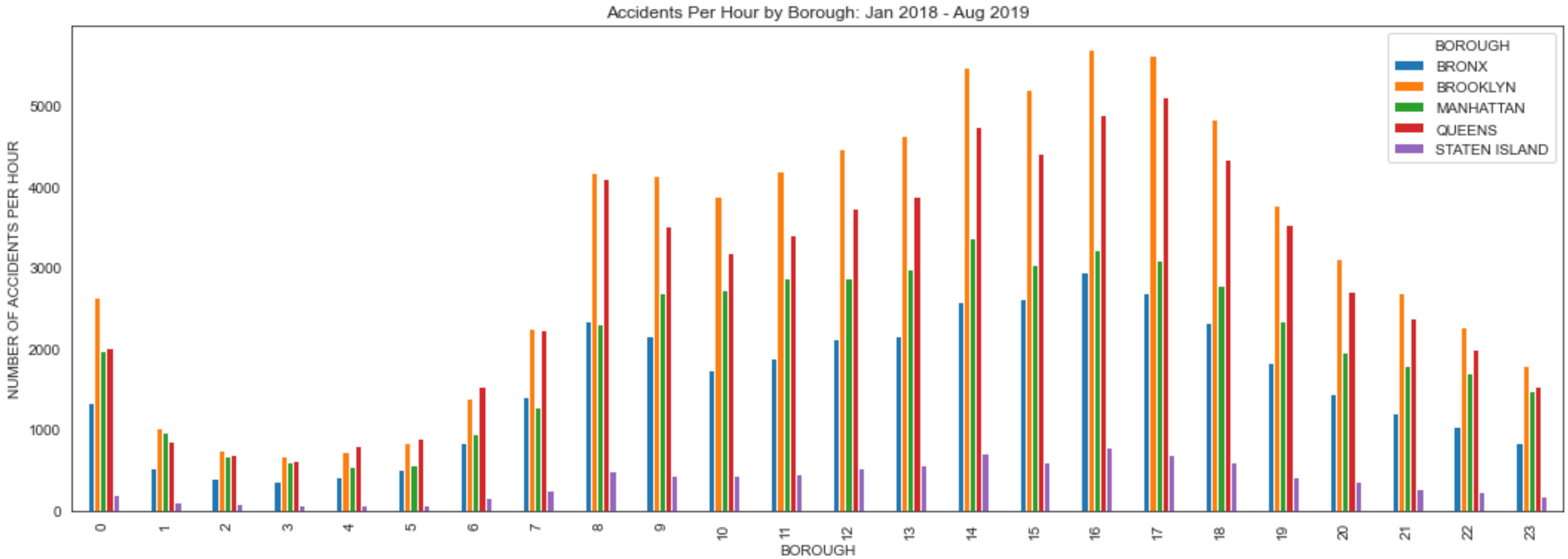
In [34]:

```
# with Seaborn
sns.set_style("white")
fg = sns.FacetGrid(df, col = "BOROUGH", hue = "HOUR")
fg.map_dataframe(sns.histplot, x = "HOUR")
fg.set_axis_labels("Hour", "Accidents");
```



In [35]:

```
# With PyPlot

df["TIME"] = pd.to_datetime(df["TIME"])
df["HOUR"] = df["TIME"].dt.hour
bor_hour = df.groupby(["HOUR", "BOROUGH"]).size()

bor_hour.unstack().plot.bar(legend = True, figsize = (18,6))

plt.xlabel("BOROUGH")
plt.ylabel("NUMBER OF ACCIDENTS PER HOUR")
plt.title("Accidents Per Hour by Borough: Jan 2018 - Aug 2019");
```



Exercise 9 (hard)

Using `contrib_df`, find which 6 factors cause the most accidents. It is important that you avoid double counting the contributing factors of a single accident.

Hint: You can use the **`pd.melt()`** function to take a subset of `df` and convert it from wide format to narrow format.

In [36]:

```
# workspace
contribf_df = df[["DATE", "COLLISION_ID","CONTRIBUTING FACTOR VEHICLE 1", "CONTRIBUTING FACTOR VEHICLE 2", "CONTRIBUTING
contribff_df = contribf_df.melt(id_vars = ["DATE", "COLLISION_ID"],
                                value_vars = ["CONTRIBUTING FACTOR VEHICLE 1", "CONTRIBUTING FACTOR VEHICLE 2", "CONTRIB
                                var_name = "CONTRIBUTING FACTOR VEHICLE #", value_name = "FACTOR")
contrib_df = contribff_df.groupby(["COLLISION_ID", "FACTOR"]).first()
factors_most_acc = contrib_df.value_counts("FACTOR", ascending = False).rename_axis("unique_factors").to_frame("counts")
factors_most_acc
```

Out[36]:

	counts
unique_factors	
Unspecified	190096
Driver Inattention/Distraction	61752
Failure to Yield Right-of-Way	19641

	counts
unique_factors	
Following Too Closely	17293

In [37]:

```
def ex_9(df):
    """
    Finds which 6 factors cause the most accidents, without
    double counting the contributing factors of a single accident.

    Arguments:
    `contrib_df`: A pandas DataFrame.

    Outputs:
    `factors_most_acc`: A pandas DataFrame. It has only 6 elements, which are,
    sorted in descending order, the contributing factors with the most accidents.
    The column with the actual numbers is named `index`.
    """

    # YOUR CODE HERE
    contribf_df = df[["DATE", "COLLISION_ID","CONTRIBUTING FACTOR VEHICLE 1", "CONTRIBUTING FACTOR VEHICLE 2", "CONTRIBU
    contribff_df = contribf_df.melt(id_vars = ["DATE", "COLLISION_ID"],
                                    value_vars = ["CONTRIBUTING FACTOR VEHICLE 1", "CONTRIBUTING FACTOR VEHICLE 2", "CON
                                    var_name = "CONTRIBUTING FACTOR VEHICLE #", value_name = "FACTOR")
    contrib_df = contribff_df.groupby(["COLLISION_ID", "FACTOR"]).first()
    factors_most_acc = contrib_df.value_counts("FACTOR", ascending = False).rename_axis("unique_factors").to_frame("coun

    return factors_most_acc
```

Exercise 10 (hard)

Which 10 vehicle type-borough pairs are most involved in accidents? Avoid double counting the types of vehicles involved in a single accident. You can apply a similar approach to the one used in the previous exercise using `pd.melt()` .

Hint: You may want to include `BOROUGH` as one of your `id_vars` (the other being `index`) in `pd.melt()` . Including `BOROUGH` in your final `.groupby()` is also a good idea.

In [38]:

```
# workspace
bvv_df = df[["DATE", "COLLISION_ID", "BOROUGH", "VEHICLE TYPE CODE 1","VEHICLE TYPE CODE 2","VEHICLE TYPE CODE 3","VEHIC
bvvv_df = bvv_df.melt(id_vars = ["COLLISION_ID", "BOROUGH"],
                      value_vars = ["VEHICLE TYPE CODE 1","VEHICLE TYPE CODE 2","VEHICLE TYPE CODE 3","VEHICLE TYPE CODE
                      var_name = "VEHICLE TYPE CODE", value_name = "VEHICLE TYPE")
bv_df = bvvv_df.groupby(["COLLISION_ID", "BOROUGH", "VEHICLE TYPE"]).first()
vehi_most_acc = bv_df.value_counts(["BOROUGH","VEHICLE TYPE"], ascending = False).rename_axis(["BOROUGH", "VEHICLE"]).re
vehi_most_acc
```

Out[38]:

	BOROUGH	VEHICLE	index
0	BROOKLYN	Sedan	39459
1	QUEENS	Sedan	35103
2	BROOKLYN	Station Wagon/Sport Utility Vehicle	32262
3	QUEENS	Station Wagon/Sport Utility Vehicle	31647
4	MANHATTAN	Sedan	20727
5	BRONX	Sedan	19652
6	MANHATTAN	Station Wagon/Sport Utility Vehicle	16432
7	BRONX	Station Wagon/Sport Utility Vehicle	15434
8	BROOKLYN	PASSENGER VEHICLE	10177
9	MANHATTAN	Taxi	8989

In [39]:

```
def ex_10(df):
    """
    Finds the 10 borough:vehicle type pairs with more accidents, without
    double counting the vehicle types of a single accident.

    Arguments:
    `df`: A pandas DataFrame.

    Outputs:
    `vehi_most_acc`: A pandas DataFrame. It has only 10 elements, which are,
    sorted in descending order, the borough-vehicle pairs with the most accidents.
    The column with the actual numbers is named `index`
    """

    vehi_cols = ['VEHICLE TYPE CODE 1','VEHICLE TYPE CODE 2','VEHICLE TYPE CODE 3','VEHICLE TYPE CODE 4','VEHICLE TYPE C

    # YOUR CODE HERE
    bvv_df = df[["DATE", "COLLISION_ID", "BOROUGH", "VEHICLE TYPE CODE 1","VEHICLE TYPE CODE 2","VEHICLE TYPE CODE 3","V
    bvvv_df = bvv_df.melt(id_vars = ["COLLISION_ID", "BOROUGH"],
                      value_vars = ["VEHICLE TYPE CODE 1","VEHICLE TYPE CODE 2","VEHICLE TYPE CODE 3","VEHICLE TYPE CODE
                      var_name = "VEHICLE TYPE CODE", value_name = "VEHICLE TYPE")
    bv_df = bvvv_df.groupby(["COLLISION_ID", "BOROUGH", "VEHICLE TYPE"]).first()
    vehi_most_acc = bv_df.value_counts(["BOROUGH","VEHICLE TYPE"], ascending = False).rename_axis(["BOROUGH", "VEHICLE"]
    vehi_most_acc

    return vehi_most_acc
```

Exercise 11

In a 2018 interview with The New York Times, New York's mayor de Blasio stated that "*Vision Zero is clearly working*". That year, the number of deaths in traffic accidents in NYC dropped to a historically low 202. Yet, as reported by am New York Metro, the number of fatalities has increased by 30% in the first quarter of 2019 compared to the previous year and the number of pedestrians and cyclists injured has not seen any improvement.

Which of the following BEST describes how you would use the provided data to understand what went wrong in the first quarter of 2019? Please explain the reasons for your choice.

A. Consider the accidents of the first quarter of 2019. Then, check for the most common causes of accidents where pedestrians and cyclists were involved. Give a recommendation based solely on this information.
B. Create a pair of heat maps of the accidents involving injured/killed pedestrians and cyclists in the first quarter of 2018 and 2019. Compare these two to see if there is any change in the concentration of accidents. In critical areas, study the type of factors involved in the accidents. Give a recommendation to visit these areas to study the problem further.
C. The provided data is insufficient to improve our understanding of the situation.
D. None of the above. (If you choose this, please elaborate on what you would do instead.)

Your answer here.

B and C
Comparing the first quarter of 2018 to the first quarter of 2019 would give insight to the degrees of change between the two years and which boroughs and values need to be more closely examined. We would also need more information. For example, weather patterns and how weather may have differed between the two years. Another example of additional data to consider is traffic concentration rates and how that may affect accident rates.

Exercise 12 (hard)

12.1

Calculate the number of deaths caused by each type of vehicle.

Hint 1: As an example of how to compute vehicle involvement in deaths, suppose two people died in an accident where 5 vehicles were involved, and 4 are PASSENGER VEHICLE and 1 is a SPORT UTILITY/STATION WAGON. Then we would add two deaths to both the PASSENGER VEHICLE and SPORT UTILITY/STATION WAGON types.)

Hint 2: You will need to use `pd.melt()` and proceed as in the previous exercises to avoid double-counting the types of vehicles (i.e. you should remove duplicate "accident ID - vehicle type" pairs).

In [40]:

```
# workspace
df.columns
```

Out[40]:
```
Index(['DATE', 'TIME', 'BOROUGH', 'ZIP CODE', 'LATITUDE', 'LONGITUDE',
       'ON STREET NAME', 'NUMBER OF PEDESTRIANS INJURED',
       'NUMBER OF PEDESTRIANS KILLED', 'NUMBER OF CYCLIST INJURED',
       'NUMBER OF CYCLIST KILLED', 'NUMBER OF MOTORIST INJURED',
       'NUMBER OF MOTORIST KILLED', 'CONTRIBUTING FACTOR VEHICLE 1',
       'CONTRIBUTING FACTOR VEHICLE 2', 'CONTRIBUTING FACTOR VEHICLE 3',
       'CONTRIBUTING FACTOR VEHICLE 4', 'CONTRIBUTING FACTOR VEHICLE 5',
       'COLLISION_ID', 'VEHICLE TYPE CODE 1', 'VEHICLE TYPE CODE 2',
       'VEHICLE TYPE CODE 3', 'VEHICLE TYPE CODE 4', 'VEHICLE TYPE CODE 5',
       'MONTH', 'HOUR', 'WEEKDAY', 'DAY_OF_WEEK'],
      dtype='object')
```

In [41]:
```python
# workspace
p = df["NUMBER OF PEDESTRIANS KILLED"].sum()
c = df["NUMBER OF CYCLIST KILLED"].sum()
m = df["NUMBER OF MOTORIST KILLED"].sum()
p+c+m
```

Out[41]: 221

In [42]:
```python
# workspace
ddf = df.copy()
ddf["TOTAL KILLED"] = ddf["NUMBER OF PEDESTRIANS KILLED"] + ddf["NUMBER OF CYCLIST KILLED"] + ddf["NUMBER OF MOTORIST KI
death_df = ddf[ddf["TOTAL KILLED"] > 0].reset_index(drop = True)
death_df
```

Out[42]:

	DATE	TIME	BOROUGH	ZIP CODE	LATITUDE	LONGITUDE	ON STREET NAME	NUMBER OF PEDESTRIANS INJURED	NUMBER OF PEDESTRIANS KILLED	NUMBER OF CYCLIST INJURED	...	VEHICLE TYPE CODE 1
0	2019-06-24	2021-05-21 09:24:00	MANHATTAN	10010.0	NaN	NaN	AVENUE OF AMERICAS	0	0	0	...	Box Truck
1	2019-07-29	2021-05-21 08:41:00	BROOKLYN	11232.0	40.656124	-74.005910	3 AVENUE	0	0	0	...	Station Wagon/Sport Utility Vehicle
2	2019-07-31	2021-05-21 05:08:00	BROOKLYN	11207.0	40.675632	-73.898780	ATLANTIC AVENUE	0	0	0	...	Sedan
3	2019-08-24	2021-05-21 00:05:00	BRONX	10463.0	40.882328	-73.891655	SEDGWICK AVENUE	0	0	0	...	Sedan
4	2019-08-05	2021-05-21 08:43:00	BROOKLYN	11212.0	40.668415	-73.910420	ROCKAWAY AVENUE	0	1	0	...	Station Wagon/Sport Utility Vehicle

	DATE	TIME	BOROUGH	ZIP CODE	LATITUDE	LONGITUDE	ON STREET NAME	NUMBER OF PEDESTRIANS INJURED	NUMBER OF PEDESTRIANS KILLED	NUMBER OF CYCLIST INJURED	...	VEHICLE TYPE CODE 1
...	...	...	...	...	...	...	...	...	...	...	...	...
214	2018-02-03	2021-05-21 02:53:00	BRONX	10467.0	40.871510	-73.870570	BRONX PARK EAST	0	0	0	...	SPORT UTILITY / STATION WAGON
215	2018-01-06	2021-05-21 16:56:00	STATEN ISLAND	10305.0	40.583733	-74.086550	SEAVIEW AVENUE	0	0	0	...	SPORT UTILITY / STATION WAGON
216	2018-01-19	2021-05-21 18:40:00	BROOKLYN	11208.0	40.684956	-73.874570	RIDGEWOOD AVENUE	0	1	0	...	SPORT UTILITY / STATION WAGON
217	2018-01-23	2021-05-21 22:52:00	QUEENS	11366.0	40.724980	-73.794235	UNION TURNPIKE	0	1	0	...	PASSENGER VEHICLE

In [43]:

```
# workspace
kill_melt = death_df.melt(id_vars = ["TOTAL KILLED"],
                          value_vars = ["VEHICLE TYPE CODE 1","VEHICLE TYPE CODE 2","VEHICLE TYPE CODE 3","VEHICLE TYP
                          var_name = "VEHICLE TYPE",
                          value_name = "VEHICLE");
kill_melt
```

Out[43]:

	TOTAL KILLED	VEHICLE TYPE	VEHICLE
0	1	VEHICLE TYPE CODE 1	Box Truck
1	1	VEHICLE TYPE CODE 1	Station Wagon/Sport Utility Vehicle
2	1	VEHICLE TYPE CODE 1	Sedan
3	1	VEHICLE TYPE CODE 1	Sedan
4	1	VEHICLE TYPE CODE 1	Station Wagon/Sport Utility Vehicle
...	...	...	...
1090	1	VEHICLE TYPE CODE 5	PASSENGER VEHICLE
1091	1	VEHICLE TYPE CODE 5	NaN
1092	1	VEHICLE TYPE CODE 5	NaN
1093	1	VEHICLE TYPE CODE 5	NaN
1094	1	VEHICLE TYPE CODE 5	NaN

1095 rows × 3 columns

In [44]:

```
# workspace

result = kill_melt.groupby(["VEHICLE"]).sum()
result.sort_values(by = "TOTAL KILLED", ascending = False)
```

Out[44]:

	TOTAL KILLED
VEHICLE	
Station Wagon/Sport Utility Vehicle	100
Sedan	79
PASSENGER VEHICLE	33
SPORT UTILITY / STATION WAGON	26
Motorcycle	22
Bike	19
Bus	10
Box Truck	8
Pick-up Truck	8
Taxi	5
Tractor Truck Diesel	4
PICK-UP TRUCK	4
Van	3
Tanker	3
BICYCLE	3
Dump	3
BU	2
Concrete Mixer	2
Garbage or Refuse	2
Tow Truck / Wrecker	2
TK	2
Motorbike	1
TN	1

	TOTAL KILLED
VEHICLE	
Beverage Truck	1
VN	1
Utili	1
USPS	1
Convertible	1
TAXI	1
Moped	1
E SCO	1
Stake or Rack	1
MD	1
MOTORCYCLE	1
Minicycle	1
Open Body	1

In [45]:

```python
def ex_12(df):
    """
    Calculate total killed per vehicle type and plot the result
    as a bar graph

    Arguments:
    `df`: A pandas DataFrame.

    Outputs:
    `result`: A pandas DataFrame. Its index should be the vehicle type. Its only
    column should be `TOTAL KILLED`
    """

    # YOUR CODE HERE
    ddf = df.copy()
    ddf["TOTAL KILLED"] = ddf["NUMBER OF PEDESTRIANS KILLED"] + ddf["NUMBER OF CYCLIST KILLED"] + ddf["NUMBER OF MOTORIS
    death_df = ddf[ddf["TOTAL KILLED"] > 0].reset_index(drop = True)

    kill_melt = death_df.melt(id_vars = ["TOTAL KILLED"],
                              value_vars = ["VEHICLE TYPE CODE 1","VEHICLE TYPE CODE 2","VEHICLE TYPE CODE 3","VEHICLE T
                              var_name = "VEHICLE TYPE",
                              value_name = "VEHICLE");

    result = kill_melt.groupby(["VEHICLE"]).sum()
    result.sort_values(by = "TOTAL KILLED", ascending = False)

    return result
```

12.2

12.2.1

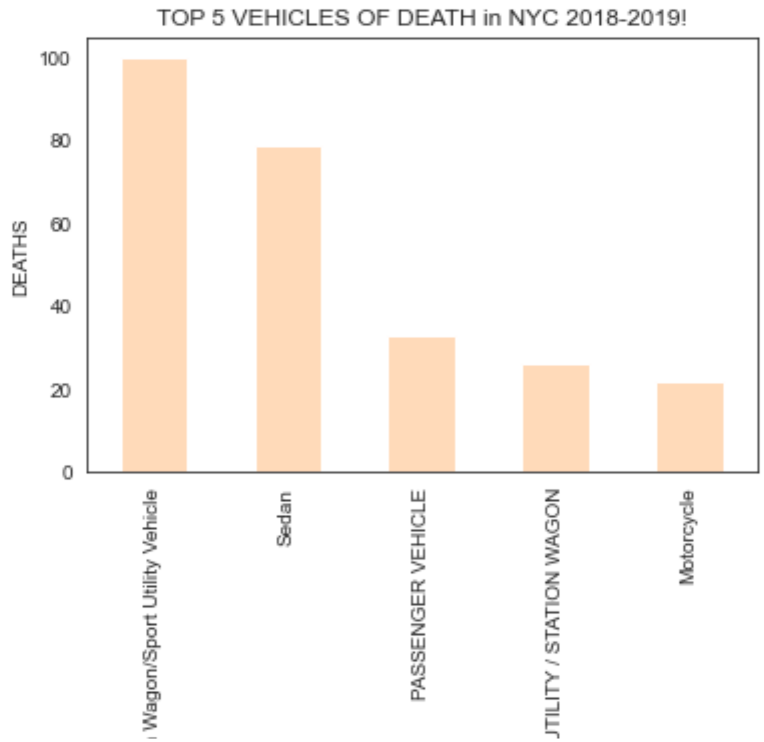
Plot a bar chart for the top 5 vehicles.

In [46]:

```python
# YOUR CODE HERE
ex_12(df).sort_values(by = "TOTAL KILLED", ascending = False).head(5).plot.bar(legend = False, color = "peachpuff")

plt.xlabel("VEHICLE")
plt.ylabel("DEATHS")
plt.title("TOP 5 VEHICLES OF DEATH in NYC 2018-2019!")
;
```

Out[46]:

```
''
```



12.2.2

Which vehicles are most often involved in deaths, and by how much more than the others?

Your answer here.

Station Wagon / Sport Utility Vehicle has the most deaths associatated.

In [47]:
```
ddf = df.copy()
ddf["TOTAL KILLED"] = ddf["NUMBER OF PEDESTRIANS KILLED"] + ddf["NUMBER OF CYCLIST KILLED"] + ddf["NUMBER OF MOTORIST KI
death_df = ddf[ddf["TOTAL KILLED"] > 0].reset_index(drop = True)

kill_melt = death_df.melt(id_vars = ["TOTAL KILLED"],
                          value_vars = ["VEHICLE TYPE CODE 1","VEHICLE TYPE CODE 2","VEHICLE TYPE CODE 3","VEHICLE TYPE
                          var_name = "VEHICLE TYPE",
                          value_name = "VEHICLE");

result = kill_melt.groupby(["VEHICLE"]).sum()
result.sort_values(by = "TOTAL KILLED", ascending = False);
```

In [48]:

```
SWSUV = result.loc["Station Wagon/Sport Utility Vehicle"].iloc[0]
Sed = result.loc["Sedan"].iloc[0]
PV = result.loc["PASSENGER VEHICLE"].iloc[0]
SUSW = result.loc["SPORT UTILITY / STATION WAGON"].iloc[0]
MC = result.loc["Motorcycle"].iloc[0]

Sedd = SWSUV - Sed
PVV = SWSUV - PV
SUSWW = SWSUV - SUSW
MCC = SWSUV - MC

print("The highest killing deathmobile was Station Wagon/Sport Utility Vehicle \nby the following margins for the Top 5
      "\nSWSUV - Sedan =", Sedd, "deaths",
      "\nSWSUV - PASSENGER VEHICLE =", PVV, "deaths",
     "\nSWSUV - SPORT UTILITY / STATION WAGON =", SUSWW, "deaths--this one is sloppy",
     "\nSWSUV - Motorcycle =", MCC, "deaths",)
```

```
The highest killing deathmobile was Station Wagon/Sport Utility Vehicle
by the following margins for the Top 5 Deathmobiles:
SWSUV - Sedan = 21 deaths
SWSUV - PASSENGER VEHICLE = 67 deaths
SWSUV - SPORT UTILITY / STATION WAGON = 74 deaths--this one is sloppy
SWSUV - Motorcycle = 78 deaths
```

Testing cells

In [49]:

```
# Ex. 2
assert type(ex_2(df)) == type(pd.Series([9,1,2])), "Ex. 2 - Your output isn't a pandas Series. If you use .groupby(), it
assert ex_2(df).loc["2018-10"] == 13336, "Ex. 2 - Wrong output! Try using the .size() aggregation function with your .gr
print("Exercise 2.1 looks correct!")
```

```
Exercise 2.1 looks correct!
```

In [50]:

```
# Ex 4
assert type(ex_4(df)) == type(pd.Series([9,1,2])), "Ex. 4 - Your output isn't a pandas Series. If you use .groupby(), it
assert ex_4(df).loc[13] == 14224, "Ex. 4 - Wrong output! Try using the .size() aggregation function with your .groupby()
print("Exercise 4.1 looks correct!")
```

```
Exercise 4.1 looks correct!
```

In [51]:

```
# Ex. 6
assert type(ex_6(df)) == type(pd.Series([9,1,2])), "Ex. 6 - Your output isn't a pandas Series. If you use .groupby(), it
assert max(ex_6(df)) == 37886, "Ex. 6 - Your results don't match ours! Remember that you can use the .size() aggregation
print("Exercise 6.1 looks correct!")
```

```
Exercise 6.1 looks correct!
```

In [52]:

```
# Ex. 7.1
assert type(ex_7_1(df)) == type(pd.Series([9,1,2])), "Ex. 7.1 - Your output isn't a pandas Series. If you use .groupby()
assert max(ex_7_1(df)) == 76253, "Ex. 7.1 - Your results don't match ours! Remember that you can use the .size() aggrega
print("Exercise 7.1 looks correct!")
```

```
Exercise 7.1 looks correct!
```

In [53]:

```
# Ex. 7.3
with open('data/borough_data.json') as f:
    borough_data=json.load(f)
borough_data
e73 = ex_7_3(df, borough_data)
assert "accidents_per_sq_mi" in e73.columns, "Ex. 7.3 - You didn't create an 'accidents_per_sq_mi' in your DataFrame!"
assert round(min(e73["accidents_per_sq_mi"])) == 149, "Ex. 7.3 - Your output doesn't match ours! Remember that you need
print("Exercise 7.3 looks correct!")
```

```
Exercise 7.3 looks correct!
```

In [54]:

```
# Ex. 8.1
assert type(ex_8_1(df)) == type(pd.Series([9,1,2])), "Ex. 9 - Your output isn't a pandas Series. If you use .groupby(),
assert ex_8_1(df).max() == 5701, "Ex. 8.1 - Your numbers don't match ours. If you haven't already, you can try using .si
print("Exercise 8.1 looks correct!")
```

```
Exercise 8.1 looks correct!
```

In [55]:

```
# Ex. 9
assert type(ex_9(df)) == type(pd.Series([9,1,2]).to_frame()), "Ex. 9 - Your output isn't a pandas DataFrame. If you use
assert len(ex_9(df)) == 6, "Ex. 9 - Your output doesn't have six elements. Did you forget to use .head(6)?"
assert int(ex_9(df).sum()) == 316248, "Ex. 9 - Your numbers don't match ours. Are you sure you sorted your Series in des
print("Exercise 9 looks correct!")
```

```
Exercise 9 looks correct!
```

In [56]:

```
# Ex. 10
assert type(ex_10(df)) == type(pd.Series([9,1,2]).to_frame()), "Ex. 10 - Your output isn't a pandas DataFrame. If you us
assert len(ex_10(df)["index"]) == 10, "Ex. 10 - Your output doesn't have 10 elements. Did you forget to use .head(10)?"
assert ex_10(df)["index"].sum() == 229882, "Ex. 10 - Your numbers don't match ours. Are you sure you sorted your Series
print("Exercise 10 looks correct!")
```

Exercise 10 looks correct!

In [57]:

```
# Ex. 12
e12 = ex_12(df)
assert type(e12) == type(pd.Series([9,1,2]).to_frame()), "Ex. 12 - Your output isn't a pandas DataFrame. If you use .gro
assert int(e12.loc["Bike"]) == 19, "Ex. 12 - Your output doesn't match ours! Remember that you need to remove the duplic
print("Exercise 12.1 looks correct!")
```

Exercise 12.1 looks correct!

Attribution

"Vehicle Collisions in NYC 2015-Present", New York Police Department, NYC Open Data terms of use, https://www.kaggle.com/nypd/vehicle-collisions

"Boroughs of New York City", Creative Commons Attribution-ShareAlike License, https://en.wikipedia.org/wiki/Boroughs_of_New_York_City