assignment1

February 11, 2020

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

[1]:
```
NAME = "Aseem Sachdeva"
```

---

1 Information Visualization I

1.1 School of Information, University of Michigan

1.2 Week 1:

- Domain identification vs Abstract Task extraction
- Pandas Review

1.3 Assignment Overview

1.3.1 The objectives for this week are for you to:

- Review, reflect, and apply the concepts of Domain Tasks and Abstract Tasks. Specifically, given a real context, identify the expert's goals and then abstract the visualization tasks.
- Review and evaluate the domain of Pandas as a tool for reading, manipulating, and analyzing datasets in Python.

1.3.2 The total score of this assignment will be 100 points consisting of:

- Case study reflection: Car congestion and crash rates (20 points)
- Pandas programming exercise (80 points)

1.3.3 Resources:

- We're going to be recreating parts of this article by CMAP available online (CMAP, 2016)

- We'll need the datasets from the city of Chicago. We have downloaded a subset to the local folder /assets
  - If you're curious, the original dataset can be found on Chicago Data Portal
    * Chicago Traffic Tracker - Historical Congestion Estimates by Segment - 2011-2018
    * Traffic Crashes - Crashes
- Altair
  - We will use a python library called Altair for the visualizations. Don't worry about understanding this code. You will only need to prepare the data for the visualization in Pandas. If you do it correctly, our code will produce the visualization for you.

1.4 Part 1. Domain identification vs Abstract Task extraction (20 points)

Read the following article by CMAP Crash scans show the relationship between congestion and crash rates and answer the following questions:

1.4.1 1.1 Briefly describe who you think performed this analysis. What is their expertise? What is their goal for the article? Give 3 examples of domain tasks featured in the article. (10 points)

The fact that the above article is hosted under the Chicaco Metropolitan Agency for Planning's(CMAP) governmental web domain name highly suggests that the analyst who authored the article is affiliated with the government of illinois through the aforementioned organization. The analayst can feasibly tout a degree of expertise in highway/roadway planning on account of the relative ease and accuracy with which they utilize domain specific vocabulary. Three examples of domain-specific tasks discussed in the article are:

1. What is the relationship between highway congestion and crash rates?
2. Do arterial roadways witness crashes at a different rate than highways?
3. Do highways with excess tollways/merge locations witness crashes at a different rate than highways with a more free design?

1.4.2 1.2 For each domain task describe the abstract task (10 points)

1. Find the correlation between variable A and variable B
2. Find the statistical difference between population A and population B
3. Find the statistical difference between population A and population B

1.5 Part 2. Pandas programming exercise (80 points)

We have provided some code to create visualizations based on these two datasets: 1. Historic Congestion 2. Traffic Crashes

Complete each assignment function and run each cell to generate the final visualizations

[2]:
```
import pandas as pd
import numpy as np
import altair as alt

pd.set_option('display.max_columns', None)
```

```
pd.set_option('display.max_rows', None)
```

[3]:
```
# enable correct rendering
alt.renderers.enable('default')
```

[3]:
```
RendererRegistry.enable('default')
```

[4]:
```
# uses intermediate json files to speed things up
alt.data_transformers.enable('json')
```

[4]:
```
DataTransformerRegistry.enable('json')
```

1.5.1 PART A: Historic Congestion (55 points)

For parts 2.1 to 2.5 we will use the Historic Congestion dataset. This dataset contains measures of speed for different segments. For this subsample, the available measures are limited to traffic on Pulaski Road in 2018.

1.5.2 2.1 Read and resample (15 points)

Complete the `read_csv` and `get_group_first_row` functions. Since our dataset is large we want to only grab one measurement per hour for each segment. To do this, we will resample by selecting the first measure for each month, day, hour on each segment. Complete the `get_group_first_row` function to achieve this. Note that the file we are loading is compressed–depending on how you load the file, this may or may not make a difference (you'll want to look at the API documents).

[5]:
```
def read_csv(filename):
    """Read the csv file from filename (uncompress 'gz' if needed)
    return the dataframe resulting from reading the columns
    """
    dataframe = pd.read_csv(filename)
    return dataframe

    #raise NotImplementedError()
```

[6]:
```
# Save the congestion dataframe on hist_con
hist_con = read_csv('assets/Pulaski.small.csv.gz')
print(hist_con.shape)
assert hist_con.shape == (3195450, 10)
assert list(hist_con.columns) ==␣
 ↪['TIME','SEGMENT_ID','SPEED','STREET','DIRECTION','FROM_STREET','TO_STREET',
                                 'HOUR','DAY_OF_WEEK','MONTH']
```

```
(3195450, 10)
```

[7]:
```
def get_group_first_row(df, grouping_columns):
    """Group rows for the grouping columns and return the first row belonging␣
 ↪to each group
    (you can look at first() for reference). We'll write this function to be␣
 ↪more general in case
```

```
    we want to use it for a different resample.
    return a dataframe without a hierarchical index (use default index)

    See the example link below if you want a better sense of what this should␣
 ↪return
    """
    dataframe = df
    dataframe_2 = dataframe.groupby(grouping_columns).first()
    dataframe_2 = dataframe_2.reset_index(level=grouping_columns)
    return dataframe_2
    #raise NotImplementedError()
```

[8]:
```
# test your code, we want segment_rows to be resampled version of hist_con␣
 ↪where we've grouped by the
# properties month, day_of_week, hour, and segment_id and returned the first␣
 ↪measure of each group
segment_rows = get_group_first_row(hist_con, ['MONTH','DAY_OF_WEEK', 'HOUR',␣
 ↪'SEGMENT_ID'])
```

The table should look something like this

[9]:
```
#hidden tests are within this cell
```

1.5.3 2.2 Basic Bar Chart Visualization (10 points)

We want to create a visualization for the *average speed* of each segment (across all the samples). To do this, we're going to want to group by each segment and calculate the average speed on each. Complete this code on the `average_speed_per_segment` function.

[10]:
```
def average_speed_per_segment(df):
    """Group rows by SEGMENT_ID and calculate the mean of each
    return a series where the index is the segment id and each value is the␣
 ↪average speed per segment
    """
    dataframe = df
    dataframe = dataframe.groupby('SEGMENT_ID').mean()
    dataframe = dataframe.drop(['MONTH', 'DAY_OF_WEEK', 'HOUR'], axis=1)
    return dataframe

    #raise NotImplementedError()
```

[11]:
```
average_speed_per_segment(segment_rows)
```

[11]:

SEGMENT_ID	SPEED
19	12.251926
20	15.274452
21	12.141079
22	12.346769
23	12.716657

24	14.882632
25	13.076467
26	15.965027
27	21.769413
28	12.197392
29	12.384114
30	12.505631
31	11.920569
32	13.241257
33	12.955542
34	12.285714
35	13.336692
36	16.472436
37	14.086544
38	13.835803
39	15.781269
40	14.314167
41	16.032009
42	15.579135
43	17.809129
44	20.890338
45	17.608773
46	13.761707
47	14.046236
48	16.059870
49	14.053942
50	13.061055
51	13.075874
52	15.416123
53	13.803201
54	15.364552
55	14.982810
56	19.681091
57	17.820391
59	15.556609
60	18.486070
61	14.847659
62	13.339063
63	13.350919
64	12.475400
65	11.455839
66	14.857143
67	21.148192
68	14.117368
69	14.763485
70	13.235329
71	11.719028

```
72          12.381743
73           9.231772
74          11.877297
75          15.154120
76          16.549496
77          14.215768
78          15.854179
79          15.011855
80          14.169532
81          15.867813
82          12.830468
83          11.155898
84          19.956728
85          17.186722
86          15.746295
87          16.215175
88          15.077060
89          14.911085
90          17.852401
91          16.122110
92          18.311203
93          13.503260
94          14.560759
95          14.959099
96          21.659751
97          18.714286
```

[12]:
```
# calculate the average speed per segment
average_speed = average_speed_per_segment(segment_rows)

# create labels for the visualization
labels = average_speed.index.astype(str)

# grab the values from the table
values = pd.DataFrame(average_speed).reset_index()

# create a chart
base = alt.Chart(values)

# we're going to "encode" the variables, more on this next assignment
encoding = base.encode(
    x= alt.X(
            'SEGMENT_ID:Q',
            title='Segment ID',
            scale=alt.Scale(zero=False)
    ),
    y=alt.Y(
```

```
            'sum(SPEED):Q',
            title='Speed Average MPH'
    ),
)

# we're going to use a bar chart and set various parameters (like bar size and␣
 ↪title) to make it readable
encoding.mark_bar(size=7).properties(title='Average Speed per␣
 ↪Segment',height=300, width=900)
```

[12]:
```
alt.Chart(...)
```

The table should look something like this

[13]:
```
#hidden tests are within this cell
```

1.5.4 2.3 Create a basic pivot table (10 points)

For the next visualization, we need a more complex transformation that will allow us to see the average speed for each month. To do this, we will create a pivot table where the index is the month, and each column is a segment id. We will put the average speed in the cells. From the table, we'll be able to find the month (by index)–giving us the row, and pick the column corresponding to the segment we care about.

Complete the `create_pivot_table` function for this

[14]:
```
def create_pivot_table (df):
    """return a pivot table where:
    each row i is a month
    each column j is a segment id
    each cell value is the average speed for the month i in the segment j
    """
    dataframe = df
    table = pd.pivot_table(dataframe, values='SPEED', index=['MONTH'],
                    columns=['SEGMENT_ID'], aggfunc=np.mean)
    return table

    #raise NotImplementedError()
```

[15]:
```
# run the code and see what's in the table
pivot_table = create_pivot_table(segment_rows)
pivot_table
```

[15]:
```
SEGMENT_ID         19         20         21         22         23         24  \
MONTH
2           6.857143  16.142857  13.571429  19.571429  18.285714  15.857143
3          10.773810  14.863095  11.696429  11.815476  13.583333  16.244048
4          11.744048  14.958333  11.791667  12.071429  13.208333  16.779762
5          11.357143  14.738095  11.369048  11.916667  12.023810  13.220238
6          11.630952  14.583333  13.011905  12.279762  12.428571  14.678571
7          11.755952  13.595238  10.880952  12.238095  12.267857  14.321429
```

8	12.988095	15.446429	12.303571	13.315476	13.023810	15.827381
9	13.970238	17.059524	14.398810	13.452381	12.017857	14.869048
10	13.708333	15.666667	12.434524	13.041667	12.422619	13.714286
11	12.970238	16.107143	11.922619	11.476190	12.125000	15.607143
12	11.845238	15.690476	11.541667	11.559524	13.833333	13.523810

SEGMENT_ID	25	26	27	28	29	30 \
MONTH						
2	11.285714	10.142857	25.000000	20.571429	20.000000	23.857143
3	12.398810	15.529762	21.779762	12.422619	12.059524	13.107143
4	14.136905	18.339286	22.232143	11.589286	11.505952	11.470238
5	11.505952	15.095238	22.857143	11.892857	11.190476	10.440476
6	12.690476	15.244048	22.309524	12.619048	11.863095	12.136905
7	13.232143	14.964286	22.232143	11.958333	10.755952	12.523810
8	12.988095	16.946429	22.244048	12.535714	12.720238	12.970238
9	12.571429	15.630952	21.571429	12.464286	13.470238	12.702381
10	13.613095	15.922619	20.333333	13.119048	13.863095	14.297619
11	14.327381	16.815476	20.553571	12.910714	13.952381	14.029762
12	13.375000	15.404762	21.446429	10.113095	12.142857	10.904762

SEGMENT_ID	31	32	33	34	35	36 \
MONTH						
2	10.571429	4.857143	9.857143	15.857143	19.142857	27.142857
3	14.708333	13.547619	13.071429	12.208333	13.791667	16.130952
4	11.922619	12.785714	12.833333	12.148810	14.285714	17.303571
5	9.166667	12.130952	12.059524	12.428571	13.773810	16.964286
6	10.815476	12.154762	12.821429	11.166667	12.065476	15.077381
7	12.232143	13.702381	16.982143	12.398810	14.476190	15.982143
8	11.517857	13.238095	13.017857	12.595238	13.279762	17.375000
9	11.101190	13.726190	11.904762	12.583333	12.922619	16.630952
10	12.601190	14.351190	13.059524	13.898810	13.303571	18.750000
11	13.380952	13.458333	11.821429	11.690476	12.071429	13.976190
12	11.815476	13.666667	12.113095	11.589286	13.154762	16.089286

SEGMENT_ID	37	38	39	40	41	42 \
MONTH						
2	20.714286	12.714286	19.857143	17.714286	18.142857	19.000000
3	13.916667	13.619048	15.714286	14.047619	15.857143	13.809524
4	14.404762	13.934524	16.232143	14.333333	16.273810	16.047619
5	12.851190	12.946429	14.732143	14.244048	15.690476	14.005952
6	13.148810	13.130952	15.827381	14.904762	16.208333	16.226190
7	14.833333	14.380952	15.952381	14.172619	16.946429	15.636905
8	13.625000	12.791667	15.821429	13.720238	15.928571	16.357143
9	13.589286	12.660714	15.363095	14.011905	14.107143	14.815476
10	14.291667	14.678571	16.345238	13.601190	15.559524	15.732143
11	15.202381	14.767857	15.309524	14.214286	15.970238	16.089286
12	14.726190	15.494048	16.345238	15.750000	17.690476	16.928571

SEGMENT_ID	43	44	45	46	47	48	\
MONTH							
2	21.000000	24.285714	12.000000	8.428571	9.142857	12.285714	
3	16.041667	20.166667	17.232143	13.214286	13.041667	15.678571	
4	18.267857	19.827381	17.476190	13.726190	14.797619	16.404762	
5	15.940476	19.369048	16.255952	11.684524	12.375000	13.857143	
6	19.547619	22.577381	18.827381	14.107143	15.428571	17.101190	
7	18.363095	21.434524	16.970238	13.863095	14.434524	17.952381	
8	17.678571	21.482143	17.863095	15.095238	14.892857	16.714286	
9	16.464286	20.529762	17.863095	13.476190	13.321429	16.464286	
10	17.446429	21.928571	17.071429	13.916667	14.172619	16.434524	
11	19.410714	20.577381	17.529762	13.755952	14.404762	15.321429	
12	18.797619	20.869048	19.232143	15.000000	13.797619	14.827381	

SEGMENT_ID	49	50	51	52	53	54	\
MONTH							
2	9.714286	18.428571	16.285714	20.142857	20.571429	16.714286	
3	14.136905	12.666667	12.125000	15.285714	15.226190	14.690476	
4	14.113095	12.970238	12.559524	14.500000	14.613095	16.101190	
5	13.077381	11.875000	13.220238	15.910714	13.708333	13.755952	
6	15.095238	12.327381	12.107143	13.892857	12.976190	15.559524	
7	14.636905	12.607143	12.869048	15.940476	14.470238	16.035714	
8	14.190476	12.892857	11.982143	14.226190	12.375000	15.386905	
9	13.488095	11.863095	13.089286	14.690476	11.488095	13.803571	
10	14.803571	13.107143	13.803571	16.208333	13.178571	15.291667	
11	13.089286	13.958333	13.404762	15.928571	13.577381	15.869048	
12	14.089286	16.119048	15.464286	17.380952	16.136905	17.095238	

SEGMENT_ID	55	56	57	59	60	61	\
MONTH							
2	10.714286	23.428571	12.571429	20.142857	22.428571	17.285714	
3	15.446429	20.238095	17.833333	16.517857	19.136905	16.053571	
4	15.238095	23.678571	19.023810	16.244048	17.648810	13.809524	
5	13.505952	18.696429	17.178571	14.440476	17.559524	12.125000	
6	14.523810	19.214286	16.875000	14.910714	18.190476	15.136905	
7	14.613095	20.494048	18.178571	15.398810	19.226190	15.047619	
8	16.047619	20.029762	17.571429	15.880952	18.589286	15.714286	
9	12.952381	17.446429	17.065476	18.458333	19.303571	16.291667	
10	13.767857	18.005952	17.238095	16.547619	19.214286	16.166667	
11	17.202381	19.315476	18.315476	15.642857	20.595238	15.880952	
12	16.708333	19.535714	19.142857	11.333333	15.232143	12.148810	

SEGMENT_ID	62	63	64	65	66	67	\
MONTH							
2	19.857143	22.714286	26.142857	19.142857	21.714286	19.285714	
3	14.732143	13.732143	12.952381	11.571429	13.958333	21.351190	

4	11.934524	13.535714	12.714286	11.535714	15.357143	22.172619
5	10.440476	10.803571	10.577381	11.273810	15.601190	21.827381
6	13.845238	13.636905	12.523810	11.726190	14.696429	20.779762
7	13.797619	14.470238	13.089286	11.428571	13.750000	20.607143
8	14.440476	12.750000	11.761905	11.440476	15.041667	22.303571
9	14.607143	14.208333	12.285714	10.059524	14.011905	21.577381
10	14.839286	14.672619	14.184524	11.541667	14.821429	20.994048
11	12.869048	13.803571	12.160714	11.279762	15.285714	20.553571
12	11.613095	11.505952	11.934524	12.380952	15.761905	19.392857

SEGMENT_ID	68	69	70	71	72	73	\
MONTH							
2	9.714286	12.571429	7.000000	9.142857	16.428571	12.142857	
3	14.410714	15.095238	14.190476	12.142857	11.988095	8.577381	
4	14.750000	15.244048	12.767857	11.720238	12.017857	9.833333	
5	13.696429	12.916667	10.904762	10.970238	11.636905	8.184524	
6	13.755952	15.005952	12.630952	11.136905	13.005952	8.654762	
7	14.011905	15.523810	13.571429	12.386905	12.904762	8.904762	
8	13.791667	15.285714	13.696429	10.547619	11.803571	9.059524	
9	13.244048	15.053571	13.386905	10.696429	12.434524	8.726190	
10	14.648810	14.922619	14.684524	12.375000	12.839286	10.797619	
11	14.636905	15.029762	14.488095	12.089286	12.392857	9.196429	
12	14.410714	13.648810	12.291667	13.232143	12.625000	10.261905	

SEGMENT_ID	74	75	76	77	78	79	\
MONTH							
2	17.428571	11.571429	10.142857	15.857143	21.857143	15.714286	
3	11.785714	14.916667	17.666667	14.773810	15.952381	15.351190	
4	11.488095	13.863095	17.398810	13.821429	16.244048	14.738095	
5	10.863095	13.071429	14.678571	13.089286	14.517857	13.613095	
6	11.779762	16.815476	18.244048	12.333333	15.029762	14.904762	
7	10.702381	15.178571	16.910714	14.107143	15.059524	15.190476	
8	12.142857	14.869048	16.851190	13.392857	14.833333	13.613095	
9	12.488095	16.827381	17.500000	13.678571	14.928571	14.773810	
10	12.833333	15.976190	16.375000	15.607143	17.095238	16.214286	
11	12.571429	14.559524	14.613095	16.101190	17.190476	15.071429	
12	11.886905	15.613095	15.523810	15.184524	17.440476	16.619048	

SEGMENT_ID	80	81	82	83	84	85	\
MONTH							
2	13.428571	20.428571	16.714286	13.857143	22.428571	25.714286	
3	14.625000	15.279762	13.035714	10.089286	19.595238	16.928571	
4	14.744048	15.940476	12.809524	11.011905	19.375000	15.428571	
5	13.517857	14.601190	11.255952	9.797619	18.517857	15.910714	
6	14.398810	14.672619	12.577381	11.547619	20.083333	17.857143	
7	15.184524	16.369048	12.839286	10.267857	20.208333	17.565476	
8	13.261905	15.029762	12.392857	10.773810	20.446429	17.553571	

```
9           12.535714  15.690476  12.541667  11.083333  20.375000  17.654762
10          15.297619  16.392857  12.785714  11.517857  20.815476  17.238095
11          12.851190  16.160714  13.535714  11.869048  19.559524  17.851190
12          15.309524  18.351190  14.369048  13.488095  20.488095  17.523810

SEGMENT_ID         86         87         88         89         90         91  \
MONTH
2           16.142857  18.428571  17.000000  14.714286  19.000000  17.857143
3           15.851190  15.833333  15.130952  16.470238  17.744048  16.095238
4           14.422619  15.476190  14.958333  14.642857  17.702381  15.386905
5           13.886905  15.892857  14.154762  12.553571  16.184524  15.130952
6           16.053571  17.964286  16.089286  14.869048  17.511905  15.220238
7           15.589286  17.791667  17.220238  15.511905  19.476190  15.630952
8           17.154762  16.886905  14.863095  13.880952  18.220238  15.196429
9           15.160714  15.523810  16.178571  14.916667  17.922619  14.101190
10          15.517857  16.464286  14.291667  15.351190  18.059524  19.273810
11          16.767857  14.869048  13.422619  15.821429  18.250000  16.357143
12          17.041667  15.357143  14.380952  15.101190  17.404762  18.755952

SEGMENT_ID         92         93         94         95         96         97
MONTH
2           20.857143  12.000000  16.857143  14.857143  22.285714  17.857143
3           18.095238  13.994048  15.875000  14.761905  20.761905  17.625000
4           18.488095  14.250000  14.803571  16.535714  23.005952  19.803571
5           17.952381  12.607143  12.976190  14.065476  20.071429  17.190476
6           19.035714  14.071429  14.315476  15.410714  21.815476  18.988095
7           18.666667  13.630952  14.857143  15.130952  21.845238  18.601190
8           17.994048  13.648810  13.666667  15.619048  21.928571  18.815476
9           16.833333  11.952381  13.005952  15.321429  19.440476  17.755952
10          18.119048  13.244048  15.160714  14.321429  22.636905  19.482143
11          17.922619  12.434524  15.607143  14.071429  22.785714  19.321429
12          19.898810  15.261905  15.244048  14.357143  22.279762  19.595238
```

The table should look something like this

[16]:
```
# we're going to implement a transformation to put the pivot table into a 'long␣
 ↪form' because it
# is easier to specify the visualization. You can print out hm_pivot_table to␣
 ↪see what it looks like
hm_pivot_table = pivot_table.copy().unstack().reset_index()
hm_pivot_table['SPEED'] = hm_pivot_table[0]
hm_pivot_table.drop(0,axis=1,inplace=True)

# create the visualization. We're going to use rectangles (a heat map of sorts).
 ↪ We'll use the segment_id to
# figure out the horizontal placement (x), the month as the vertical (y) and␣
 ↪use color to encode the speed.
encoding = alt.Chart(hm_pivot_table).mark_rect().encode(
```

```
    x='SEGMENT_ID:O',
    y='MONTH:O',
    color='SPEED:Q'
)

encoding.properties(title='Average Speed per Segment per Month',height=300,␣
 ↪width=800)
```

[16]:
```
alt.Chart(...)
```

[17]:
```
# test function
pivot_table = create_pivot_table(segment_rows)
# check that the rows are months and columns are segments
assert pivot_table.shape == (11, 78), "Problem 2.3, first test"
# check that the value is the average
assert int(pivot_table.loc[2,19]) == 6, "Problem 2.3, second test"
assert int(pivot_table.loc[3,19]) == 10, "Problem 2.3, third test"
```

1.5.5 2.4 Sorting, Transforming, and Filtering (20 points)

Without telling you too much about the visualization we want to create next (that's part of the bonus below), we need to get the data into a form we can use. - We're going to need to sort the dataframe by one or more columns (this is the `sort` function). - We'll want to create a derivative column that is the time of the measurement rounded to the nearest hour (`time_to_hours`) - We need to "facet" the data into groups to generate different visualizations. - We need a function that selects part of the dataframe that matches a specific characteristic (`filter_orientation`) - Grab a specific column from the dataframe (`select_column`)

[18]:
```
def sort(df, sorting_columns):
    """Sort the rows by the columns
    return the sorted dataframe
    """
    dataframe = df
    dataframe = dataframe.sort_values(sorting_columns)
    return dataframe
    # YOUR CODE HERE
    #raise NotImplementedError()
```

[19]:
```
segment_rows = sort(segment_rows, ['SEGMENT_ID'])
```

[20]:
```
#hidden tests are within this cell
```

[21]:
```
def time_to_hours(df):
    """ Add a column (called TIME_HOURS) based on the data in the TIME column␣
 ↪and rounded up
    the value to the nearest hour.  For example, if the original TIME row said:
    02/28/2018 05:40:00 PM we want 2018-02-28 18:00:00
    (the change is that 5:40pm was rounded up to 6:00pm and the TIME_HOUR␣
 ↪column is
    actually a proper datetime and not a string).
```

```
    """
    dataframe = df
    dataframe['TIME_HOURS'] = pd.to_datetime(dataframe['TIME']).dt.round('H')
    #dataframe['TIME_HOURS'] = dataframe['TIME_HOURS'].dt.round('H')
    return dataframe
    #raise NotImplementedError()
```

[22]:
```
segment_rows = time_to_hours(segment_rows)
```

[23]:
```
#hidden tests are within this cell
```

[24]:
```
def filter_orientation(df, traffic_orientation):
    """ Filter the rows according to the traffic orientation
    return a df that is a subset of the original with the desired orientation
    """
    dataframe = df
    dataframe_subset = dataframe[dataframe['DIRECTION'] == traffic_orientation]
    return dataframe_subset
    #raise NotImplementedError()
```

[25]:
```
sb = filter_orientation(segment_rows, 'SB')
nb = filter_orientation(segment_rows, 'NB')
```

The sb table should look something like this

[26]:
```
#hidden tests are within this cell
```

[27]:
```
def select_column(df, column_name):
    """ Select a column from the df
    return a series with the desired column
    """
    dataframe = df
    dataframe_selected_column = dataframe[column_name]
    return dataframe_selected_column
    #raise NotImplementedError()
```

[28]:
```
#hidden tests are within this cell
```

[29]:
```
# we're going to remove speeds of -1 (no data)
sb = sb[sb.SPEED > -1]
nb = nb[nb.SPEED > -1]
```

[30]:
```
alt.data_transformers.disable_max_rows()
alt.Chart(sb.append(nb)).mark_rect().encode(
    x='month(TIME_HOURS):T',
    y='FROM_STREET:N',
    color='mean(SPEED):Q',
    facet='DIRECTION:N'
).properties(
    width=300,
    height=400
)
```

[30]:
```
alt.Chart(...)
```

1.5.6 2.5 (Bonus) Traffic heatmap visualization (up to 2 points)

Looking at the visualization above (the one showing Northbound versus Southbound facets), what domain/abstract tasks are fulfilled by this visualization? List at least one domain task and the corresponding abstract task.

Domain Task: Is there a relationship between time and traffic congestion on northbound lanes and southbound lanes, and is there a significant difference between both populations?

Abstract Tasks: (1) Find the correlation between variable A and variable B (2) Find the statistical difference between population A and population B

1.5.7 PART B: Crashes (25 points)

For parts 2.6 and 2.7 we will use the Crashes dataset. This dataset contains crash entries recording the time of the accident, the street, and the street number where the accident occurred. You will work with accidents recorded on Pulaski Road

[31]:
```
crashes = read_csv('assets/Traffic.Crashes.csv.gz')
crashes_pulaski = crashes[crashes.STREET_NAME == 'PULASKI RD']
```

[32]:
```
crashes_pulaski['STREET_NO'].max()
```

[32]:
```
11480
```

1.5.8 2.6 Calculate summary statistics for grouped streets (15 points)

- Group the streets every 300 units (street numbers). Hint: You can use the `pd.cut` function
- Calculate the number of accidents (count rows) and the total of injuries (sum injuries total) for each of these 300-chunk road segments. Do this *for each direction*.

Complete bin_crashes and calculate_group_aggregates functions for this

[33]:
```
def bin_crashes(df):
    """ Assign each crash instance a category (bin) every 300 house number␣
↪units starting from 0
    Return a new dataframe with a column called BIN where each value is the␣
↪start of the bin
    i.e. 0 is the label for records with street number n, such that 1 <= n <=␣
↪300
    300 is the label for records with with n at 301 <= n <= 600, and so on.
    """
    bin_values = np.arange(0,12000,300)
    df['BIN'] = pd.cut(df['STREET_NO'],bin_values,labels=bin_values[0:-1])
    return df
    #raise NotImplementedError()
```

[34]:
```
binned_df = bin_crashes(crashes_pulaski)
```

[35]:
```
binned_df.head()
```

[35]:

	RD_NO	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT
6	JC100005	NaN	12/31/2018 11:45:00 PM	25
51	JB574321	NaN	12/31/2018 09:20:00 PM	30
126	JB574112	NaN	12/31/2018 05:30:00 PM	30
133	JC111739	NaN	12/31/2018 05:30:00 PM	30
240	JB573809	NaN	12/31/2018 01:30:00 PM	30

	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION
6	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	UNKNOWN
51	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	RAIN
126	NO CONTROLS	NO CONTROLS	RAIN
133	NO CONTROLS	NO CONTROLS	RAIN
240	NO CONTROLS	NO CONTROLS	RAIN

	LIGHTING_CONDITION	FIRST_CRASH_TYPE
6	DARKNESS, LIGHTED ROAD	TURNING
51	DARKNESS, LIGHTED ROAD	REAR END
126	DARKNESS, LIGHTED ROAD	PARKED MOTOR VEHICLE
133	DUSK	REAR END
240	DAYLIGHT	TURNING

	TRAFFICWAY_TYPE	LANE_CNT	ALIGNMENT
6	NOT DIVIDED	4.0	STRAIGHT AND LEVEL
51	NOT DIVIDED	4.0	STRAIGHT AND LEVEL
126	PARKING LOT	0.0	STRAIGHT AND LEVEL
133	NOT DIVIDED	NaN	STRAIGHT AND LEVEL
240	DIVIDED - W/MEDIAN (NOT RAISED)	4.0	STRAIGHT AND LEVEL

	ROADWAY_SURFACE_COND	ROAD_DEFECT	REPORT_TYPE
6	UNKNOWN	NO DEFECTS	NOT ON SCENE (DESK REPORT)
51	WET	NO DEFECTS	ON SCENE
126	WET	NO DEFECTS	NOT ON SCENE (DESK REPORT)
133	WET	NO DEFECTS	NOT ON SCENE (DESK REPORT)
240	WET	NO DEFECTS	ON SCENE

	CRASH_TYPE	INTERSECTION_RELATED_I
6	NO INJURY / DRIVE AWAY	Y
51	NO INJURY / DRIVE AWAY	NaN
126	NO INJURY / DRIVE AWAY	NaN
133	NO INJURY / DRIVE AWAY	NaN
240	INJURY AND / OR TOW DUE TO CRASH	NaN

	NOT_RIGHT_OF_WAY_I	HIT_AND_RUN_I	DAMAGE	DATE_POLICE_NOTIFIED
6	NaN	Y	OVER $1,500	12/31/2018 11:55:00 PM
51	NaN	Y	OVER $1,500	12/31/2018 09:24:00 PM
126	Y	NaN	$501 - $1,500	12/31/2018 06:03:00 PM
133	NaN	NaN	$501 - $1,500	01/10/2019 02:42:00 PM

```
240               NaN           NaN    OVER $1,500  12/31/2018 01:33:00 PM

                  PRIM_CONTRIBUTORY_CAUSE  \
6               IMPROPER TURNING/NO SIGNAL
51   FAILING TO REDUCE SPEED TO AVOID CRASH
126                     IMPROPER BACKING
133                       NOT APPLICABLE
240                   IMPROPER LANE USAGE

                SEC_CONTRIBUTORY_CAUSE  STREET_NO STREET_DIRECTION  \
6                   IMPROPER LANE USAGE       2200                S
51                       NOT APPLICABLE       4699                S
126  DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       5139                S
133                      NOT APPLICABLE       3900                S
240                 UNABLE TO DETERMINE       5301                S

    STREET_NAME  BEAT_OF_OCCURRENCE PHOTOS_TAKEN_I STATEMENTS_TAKEN_I  \
6    PULASKI RD               1013            NaN               NaN
51   PULASKI RD                821            NaN               NaN
126  PULASKI RD                822            NaN               NaN
133  PULASKI RD                815            NaN               NaN
240  PULASKI RD                822            NaN               NaN

    DOORING_I WORK_ZONE_I WORK_ZONE_TYPE WORKERS_PRESENT_I  NUM_UNITS  \
6         NaN           N            NaN               NaN        2.0
51        NaN         NaN            NaN               NaN        2.0
126       NaN         NaN            NaN               NaN        2.0
133       NaN         NaN            NaN               NaN        2.0
240       NaN         NaN            NaN               NaN        2.0

         MOST_SEVERE_INJURY  INJURIES_TOTAL  INJURIES_FATAL  \
6    NO INDICATION OF INJURY             0.0             0.0
51   NO INDICATION OF INJURY             0.0             0.0
126  NO INDICATION OF INJURY             0.0             0.0
133  NO INDICATION OF INJURY             0.0             0.0
240  NO INDICATION OF INJURY             0.0             0.0

     INJURIES_INCAPACITATING  INJURIES_NON_INCAPACITATING  \
6                        0.0                          0.0
51                       0.0                          0.0
126                      0.0                          0.0
133                      0.0                          0.0
240                      0.0                          0.0

     INJURIES_REPORTED_NOT_EVIDENT  INJURIES_NO_INDICATION  INJURIES_UNKNOWN  \
6                              0.0                     3.0               0.0
51                             0.0                     2.0               0.0
```

```
126                          0.0                    2.0              0.0
133                          0.0                    4.0              0.0
240                          0.0                    6.0              0.0

     CRASH_HOUR  CRASH_DAY_OF_WEEK  CRASH_MONTH   LATITUDE  LONGITUDE  \
6            23                  2           12  41.851521 -87.724905
51           21                  2           12  41.807977 -87.723443
126          17                  2           12  41.799320 -87.723191
133          17                  2           12  41.822564 -87.724187
240          13                  2           12  41.796759 -87.723116

                                      LOCATION   BIN
6     POINT (-87.72490478675 41.851521442331)  2100
51   POINT (-87.723442855227 41.807977202659)  4500
126   POINT (-87.72319071101 41.799320431921)  5100
133  POINT (-87.724187479133 41.822563835982)  3600
240   POINT (-87.723115903434 41.79675909896)  5100
```

A sample of the relevant columns from the table would look something like this. We can also create a histogram of street numbers to see which are the most prevalent. It should look something like this.

[36]:
```
# create this vis
alt.Chart(binned_df).mark_bar().encode(
    alt.X('BIN'),
    alt.Y('count()')
)
```

[36]:
```
alt.Chart(...)
```

[37]:
```
#hidden tests are within this cell
```

[38]:
```
def calculate_group_aggregates(df):
    """
    Return a df with the count of accidents in a 'ACCIDENT_COUNT' column and 'INJURIES_SUM'
    """
    dataframe = df
    grouping_columns = ['BIN', 'STREET_DIRECTION']
    dataframe = dataframe.groupby(grouping_columns)['INJURIES_TOTAL'].agg({'ACCIDENT_COUNT': 'count', 'INJURIES_SUM': 'sum'})
    dataframe = dataframe.reset_index(level=grouping_columns)

    return dataframe
    #raise NotImplementedError()
```

[39]:
```
aggregates = calculate_group_aggregates(binned_df)
aggregates.head(5)
```

[39]:
```
   BIN STREET_DIRECTION  ACCIDENT_COUNT  INJURIES_SUM
0    0                N            52.0          18.0
```

```
1    0                S            44.0          17.0
2  300                N            37.0          19.0
3  300                S            49.0          17.0
4  600                N            69.0          19.0
```

The table should look something like this

[40]:
```
#hidden tests are within this cell
```

Just for fun, here's a plot of injuries in the North and South directions based on bin. This may also help you debug your code. Depending on whether you removed N/A or if you hardcoded things, you may see slight differences. Here's what it might look like

[41]:
```
alt.Chart(aggregates).mark_point().encode(
    alt.Color('STREET_DIRECTION'),
    alt.X('BIN'),
    alt.Y('INJURIES_SUM')
)
```

[41]:
```
alt.Chart(...)
```

1.5.9 2.7 Sort the street ranges (10 points)

- Sort the dataframe so North streets are in descending order and South streets are in ascending order
- You are provided with a 'sort' arrray that contains this desired order. Use a categorical (pd.Categorial) column to order the dataframe according to this array.

[42]:
```
crashed_range = list(range(0, crashes_pulaski.STREET_NO.max()+1000, 300))
sort = ['N ' + str(s) for s in crashed_range[::-1]] + ['S ' + str(s) for s in␣
 ↪crashed_range]
def categorical_sorting(df, sort):
    """ Create a column called ORDER_LABEL that contains a concatenation of the␣
 ↪street direction and the street range
    Set the sort order of this column to the provided sort array (the elements␣
 ↪of this column should be in the same order
    of the array)
    Sort the dataframe by this column
    """
    dataframe = df
    dataframe['BIN'] = dataframe['BIN'].astype(str)
    dataframe['STREET_DIRECTION'] = dataframe['STREET_DIRECTION'].astype(str)
    dataframe['ORDER_LABEL'] = dataframe['STREET_DIRECTION'] + ' ' + ␣
 ↪dataframe['BIN']
    dataframe['ORDER_LABEL'] = pd.Categorical(dataframe['ORDER_LABEL'],
                          categories=sort,
                          ordered=True)

    dataframe.sort_values('ORDER_LABEL', inplace=True)
```

```
    return dataframe
    #raise NotImplementedError()
```

[43]:
```
sorted_groups = categorical_sorting(aggregates, sort)
```

The table should look something like this

[44]:
```
#hidden tests are within this cell
```

Again, just for kicks, let's see where injuries happen. We're going to color bars by the bin and preserve our ascending/descending visualization. We can probably imagine other (better) ways to visualize this data, but this may be useful for you to debug. The visualization should look something like this

[45]:
```
alt.Chart(sorted_groups).mark_bar().encode(
    alt.X('ORDER_LABEL:O', sort=sort),
    alt.Y('INJURIES_SUM:Q'),
    alt.Color('BIN:Q')
).properties(
    width=400
)
```

[45]:
```
alt.Chart(...)
```

Ok, let's actually make a useful visualization using some of the dataframes we've created. As a bonus, we're going to ask you what you would use this for.

[46]:
```
# to make the kind of chart we are interested in we're going to build it out of␣
 ↪three different charts and
# put them together at the end

# this is going to be the left chart
bar_sorted_groups = sorted_groups[['ACCIDENT_COUNT','INJURIES_SUM']].unstack().
 ↪reset_index() \
    .rename({'level_0':'TYPE','level_1':'SPEED',0:'COUNT'},axis=1)

a = alt.Chart(bar_sorted_groups).mark_bar().transform_filter(alt.datum.TYPE ==␣
 ↪'ACCIDENT_COUNT').encode(
    x=alt.X('COUNT:Q',sort='descending'),
    y=alt.Y('SPEED:O',axis=None),
    color=alt.Color('TYPE:N',
                    legend=None,
                    scale=alt.Scale(domain=['ACCIDENT_COUNT', 'INJURIES_SUM'],
                                    range=['blue', 'orange']))
).properties(
    title='ACCIDENT_COUNT',
    width=300,
    height=600
)

# middle "chart" which actually won't be a chart, just a bunch of labels
```

```
b = alt.Chart(bar_sorted_groups).mark_bar().transform_filter(alt.datum.TYPE ==␣
 ↪'ACCIDENT_COUNT').encode(
    y=alt.Y('SPEED:O', axis=None),
    text=alt.Text('SPEED:Q')
).mark_text().properties(title='SPEED',
                         width=20,
                         height=600)

# and the right most chart
c = alt.Chart(bar_sorted_groups).mark_bar().transform_filter(alt.datum.TYPE ==␣
 ↪'INJURIES_SUM').encode(
    x='COUNT:Q',
    y=alt.Y('SPEED:O',axis=None),
    color=alt.Color('TYPE:N',
                    legend=None,
                    scale=alt.Scale(domain=['ACCIDENT_COUNT', 'INJURIES_SUM'],
                                    range=['blue', 'orange']))
).properties(
    title='INJURIES_SUM',
    width=300,
    height=600
)

# put them all together

a | b | c
```

[46]: `alt.HConcatChart(...)`

1.6 2.8 (Bonus) Accident barchart visualization (up to 2 points)

Looking at the visualization we generated above (part 2.7), what domain/abstract tasks are fulfilled by this visualization? List at least one domain task and the corresponding abstract task.

Domain Task: What is the relationship between speed and accident count, and speed and total injuries?

Abstract Task: Find the correlation between variable A and variable B