Information Visualization I

School of Information, University of Michigan

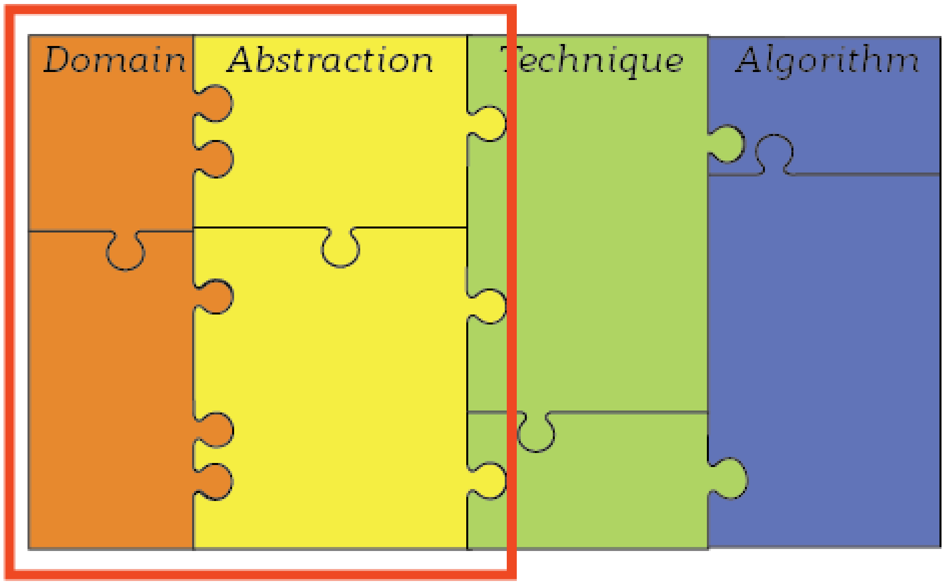
Week 1:

- Domain identification vs Abstract Task extraction
- Pandas Review

Assignment Overview

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 (https://pandas.pydata.org/) as a tool for reading, manipulating, and analyzing datasets in Python.

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)

Resources:

- We're going to be recreating parts of this article by CMAP (https://www.cmap.illinois.gov/) available online (https://www.cmap.illinois.gov/updates/all/-/asset_publisher/UIMfSLnFfMB6/content/crash-scans-show-relationship-between-congestion-and-crash-rates) (CMAP, 2016)

- We'll need the datasets from the city of Chicago. We have downloaded a subset to the local folder /assets (assets/)
  - If you're curious, the original dataset can be found on Chicago Data Portal (https://data.cityofchicago.org/)
    - Chicago Traffic Tracker - Historical Congestion Estimates by Segment - 2011-2018 (https://data.cityofchicago.org/Transportation/Chicago-Traffic-Tracker-Historical-Congestion-Esti/77hq-huss)
    - Traffic Crashes - Crashes (https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if)
- Pandas
  - This assignment is partially a warm-up/reminder of how to use Pandas. We've also created an optional lab for you (see Coursera) if you need more help remembering how to do things in Pandas.
- Altair
  - We will use a python library called Altair (https://altair-viz.github.io/) 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.
  - If you're interested, we made a short 7-minute video (https://www.youtube.com/watch?v=Tg41r3lAYoQ) explaining the very basics of how Grammar of Graphics/Altair works and why we need to transform the data as we do.

Important notes:

1) Grading for this assignment is entirely done by a human grader. They will be running tests on the functions we ask you to create. This means there is no autograding (submitting through the autograder will result in an error). You are expected to test and validate your own code.

2) Keep your notebooks clean and readable. If your code is highly messy or inefficient you will get a deduction.

3) Pay attention to the inputs and return types of your functions. Sometimes things will look right but fail later if you return the wrong kind of object (e.g., Array instead of Series). *Do not* hard-code variables into your functions. *Do not* modify our function definitions.

4) Follow the instructions for submission on Coursera. You will be providing us a generated link to a read-only version of your notebook and a PDF. When turning in your PDF, please use the File -> Print -> Save as PDF option ***from your browser***. Do ***not*** use the File->Download as->PDF option. Complete instructions for this are under Resources in the Coursera page for this class. If you're having trouble with printing, take a look at this video (https://youtu.be/PiO-K7AoWjk).

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 (https://www.cmap.illinois.gov/updates/all/-/asset_publisher/UIMfSLnFfMB6/content/crash-scans-show-relationship-between-congestion-and-crash-rates) and answer the following questions. Think of this as the output report produced by the analyst.

Remember: Domain tasks are questions an analyst might want to answer and/or they might be insights (answers) the analyst wants to communicate to someone else. For example, a retail analyst might want to know: how many fruit did we sell? or what's the relationship between temperature and fruits rotting? A learning analyst would have the domain task: how often do students pass the class? or how does study time correlate with grade? An advertising analyst would ask: how many people clicked on an ad? or what's the relationship between time of day and click through rate?

Abstract tasks are generic: What's the sum of a quantitative variable? or what's the correlation between two variables? Notice we gave two examples for each analyst type and these roughly map to the two abstract questions. You should not use domain language (e.g., accidents) when describing abstract tasks.

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)

1.1 Answer

I believe that a Department of Transportation employee, who likely works under the National Highway Traffic Safety Administration performed this analysis. Their expertise is likely data analytics. Their goal for the article was to lay out as much information as possible about car accidents with hopes that the Department of Transportation would make changes to the highway systems that would lead to reduced crash rates.

Domain tasks :

1) Highway designs have an impact on crash rates. 2) What is the relationship between congestion and crashes? 3) Expressways have a lower crash rate than roadways.

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

1.2 Answer

Abstract tasks

Domain: Highway designs have an impact on crash rates. Abstract: What is the correlation coefficient between the two variables?

Domain: What is the correlation between congestion and crash frequency? Abstract: Is the correlation between the two variables positive or negative?

Domain: Expressways have a lower crash rate than roadways. Abstract: Are there any outliers present?

Part 2. Pandas programming exercise (80 points)

We have provided some code to create visualizations based on these two datasets:

1. Historic Congestion (assets/Pulaski.small.csv.gz)
2. Traffic Crashes (assets/Traffic.Crashes.csv.gz)

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

In [1]:

```
import pandas as pd
import numpy as np
import altair as alt
```

In [2]:

```
# enable correct rendering
alt.renderers.enable('default')
```

Out[2]:

```
RendererRegistry.enable('default')
```

In [3]:

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

Out[3]:

```
DataTransformerRegistry.enable('json')
```

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.

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 grouping based on some columns (e.g., month, day, hour for each segment) and then picking out the first measurement from that group. We're going to write the sampling function to be generic. 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 (https://pandas.pydata.org/pandas-docs/stable/reference/index.html)).

In [4]:

```
def read_csv(filename):

    df = pd.read_csv(filename)
    return df

read_csv('assets/Pulaski.small.csv.gz')
```

Out[4]:

	TIME	SEGMENT_ID	SPEED	STREET	DIRECTION	FROM_STREET	TO_STREET	HO
0	12/31/2018 11:50:23 PM	83	-1	Pulaski	SB	Lake	Washington	
1	12/31/2018 11:50:23 PM	84	20	Pulaski	SB	Chicago	Lake	
2	12/31/2018 11:50:19 PM	78	27	Pulaski	SB	Cermak	26th	
3	12/31/2018 11:50:19 PM	79	27	Pulaski	SB	16th	Cermak	
4	12/31/2018 11:50:19 PM	80	27	Pulaski	SB	Roosevelt	16th	
...	...	...	...	...	...	...	...	
3195445	02/28/2018 05:01:00 PM	95	22	Pulaski	SB	Foster	Lawrence	
3195446	02/28/2018 05:01:00 PM	96	33	Pulaski	SB	Bryn Mawr	Foster	
3195447	02/28/2018 05:01:00 PM	96	33	Pulaski	SB	Bryn Mawr	Foster	
3195448	02/28/2018 05:01:00 PM	97	26	Pulaski	SB	Peterson	Bryn Mawr	
3195449	02/28/2018 05:01:00 PM	97	26	Pulaski	SB	Peterson	Bryn Mawr	

3195450 rows × 10 columns

In [5]:

```
#df = pd.read_csv('assets/Pulaski.small.csv.gz')

#def read_csv(filename):
    #"""Read the csv file from filename (uncompress 'gz' if needed)
    #return the dataframe resulting from reading the columns
    #"""
```

In [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','DIR
                                  'HOUR','DAY_OF_WEEK','MONTH']
```

```
(3195450, 10)
```

In [7]:

```python
grouping_columns =  ['MONTH','DAY_OF_WEEK', 'HOUR', 'SEGMENT_ID']
df = hist_con

def get_group_first_row(df, grouping_columns):
    """Group rows using the grouping_columns argument and return the first
    (you can look at first() for reference). We'll write this function to b
    we want to use it for a different resample.
    return a dataframe without a hierarchical index (important: return with

    See the example link below if you want a better sense of what this shou
    """
    #df = hist_con
    #grouping_columns =  ['MONTH','DAY_OF_WEEK', 'HOUR', 'SEGMENT_ID']
    df = df.groupby(grouping_columns).sample(n=1, random_state=1)
    df = df.set_index(grouping_columns)
    df = df.reset_index()

    return df
get_group_first_row(df, grouping_columns)
    # YOUR CODE HERE
    #raise NotImplementedError()
```

Out[7]:

	MONTH	DAY_OF_WEEK	HOUR	SEGMENT_ID	TIME	SPEED	STREET	DIRECTION
0	2	4	17	19	02/28/2018 05:30:00 PM	32	Pulaski	NB
1	2	4	17	20	02/28/2018 05:01:00 PM	20	Pulaski	NB
2	2	4	17	21	02/28/2018 05:01:00 PM	-1	Pulaski	NB
3	2	4	17	22	02/28/2018 05:30:00 PM	-1	Pulaski	NB
4	2	4	17	23	02/28/2018 05:10:00 PM	23	Pulaski	NB
...	...	...	...	...	...	...	...	...
131581	12	7	23	93	12/08/2018 11:40:06 PM	30	Pulaski	SB
131582	12	7	23	94	12/29/2018 11:20:40 PM	25	Pulaski	SB
131583	12	7	23	95	12/08/2018 11:01:08 PM	-1	Pulaski	SB
131584	12	7	23	96	12/22/2018 11:40:04 PM	-1	Pulaski	SB

	MONTH	DAY_OF_WEEK	HOUR	SEGMENT_ID	TIME	SPEED	STREET	DIRECTION
131585	12	7	23	97	12/15/2018 11:50:38 PM	19	Pulaski	SB

131586 rows × 10 columns

In [8]:

```
# test your code, we want segment_rows to be resampled version of hist_con
# properties month, day_of_week, hour, and segment_id and returned the firs
segment_rows = get_group_first_row(hist_con, ['MONTH','DAY_OF_WEEK', 'HOUR'

# ADD YOUR TESTS HERE

segment_rows.sample(5)
assert segment_rows.loc[24854].TIME == '04/13/2018 11:20:40 PM'
```

The table should look something like this (assets/segment_rows.png).

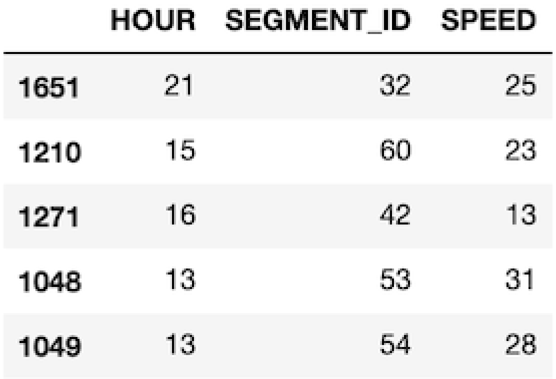
***Note** When we show examples like this, we are sampling (e.g., `segment_rows.sample(5)`) so your table may look different.

If you want to build your own tests from our example tables, you can create an assert test for one of the rows and make sure the values match what you expect. For example we see that the row id 68592 in the example is for 8/27/2018 at 1:50:21 PM. So we could write the test:

```
assert segment_rows.loc[68952].TIME == '08/27/2018 01:50:21 PM'
```

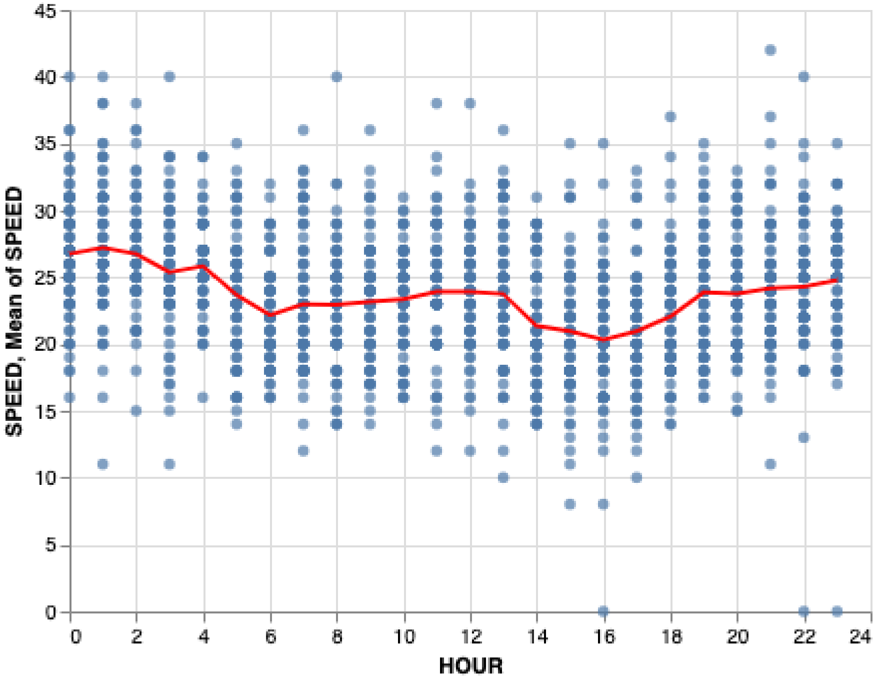
If this assertion failed, you'd get an error message.

Now let's do something a little bit interesting with this. We should now be able to test a theory that traffic speeds vary by hour of day. We're going to create a scatter plot showing hour on the x-axis and speed on the y-axis. We're going to sample only one hour per segment to keep things simple. So for each segment (we have 78 of them) we're going to find the first speed measure for 12am, 1am, 2am, etc. The result will be roughly 1872 points (plus or minus, we have some missing data). On top of that, we will add a line for the mean speed for each hour. To plot this, we need our data to look roughly like this:



	HOUR	SEGMENT_ID	SPEED
1651	21	32	25
1210	15	60	23
1271	16	42	13
1048	13	53	31
1049	13	54	28

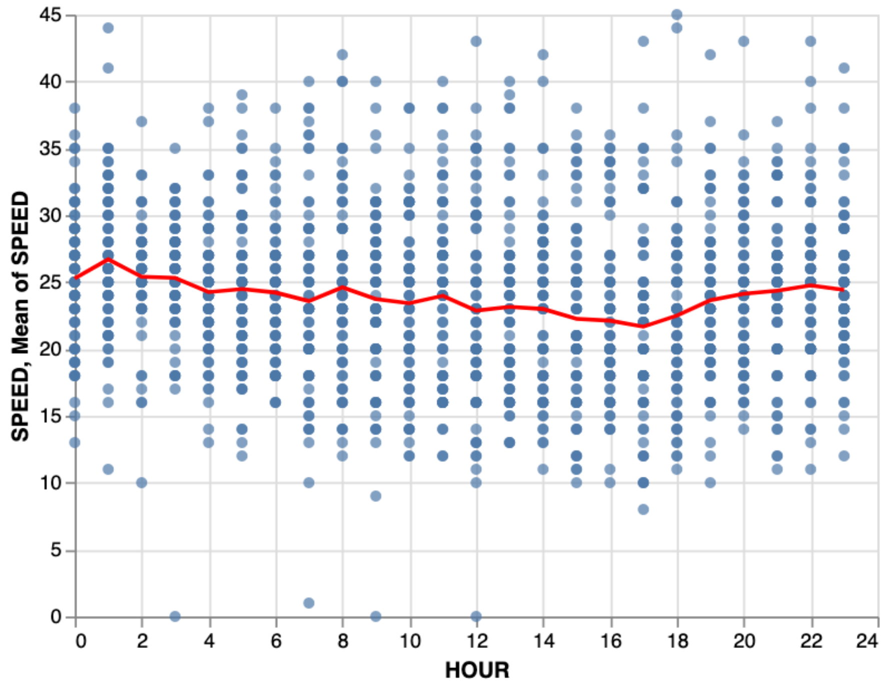
This will allow the encoding system to read row by row, and create a point for each where the X is the hour and Y is the speed. If everything works, you'll see:



Notice the dip in speeds around morning and afternoon rush hours.

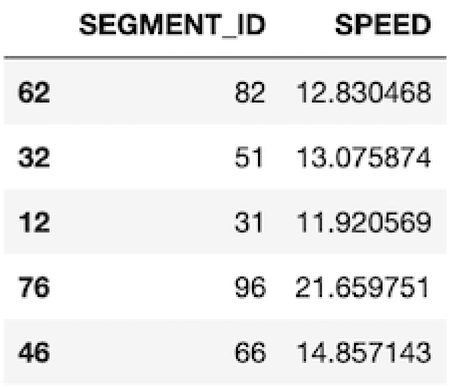
In [9]:
```
def create_mean_speed_vis(indf):
    # input: indf -- the input frame (in style of hist_con above)
    
    # take the history of congestion data and only keep rows where speed >
    srows = indf[indf.SPEED>-1]
    
    # sub-sample for hour/segment
    srows = get_group_first_row(srows, ['HOUR', 'SEGMENT_ID'])
    
    # grab the only columns we care about (strictly speaking, we only need
    srows = srows[['HOUR','SEGMENT_ID','SPEED']]
    
    # create the scatter plot using this data
    distr = alt.Chart(srows).mark_circle().encode(
        x='HOUR:Q',   # x is the HOUR
        y='SPEED:Q'   # y is the speed
    )
    
    # create the line chart on top, we could calculate the means in either
    mean = distr.mark_line(color='red').encode(
        # this "extends" distr, so x is still encoding HOUR
        y='mean(SPEED):Q' # y should now encode the mean of SPEED (at each
    )
    
    # combine the scatter plot and line chart
    return distr+mean

create_mean_speed_vis(hist_con)
```

Out[9]:



2.2 Basic Bar Chart Visualization (10 points)

We want to create a bar chart visualization for the *average speed* of each segment (across all the samples). Our encoder is going to want the data so we that we have one row per segment, with a segment id column (we'll use this for the X placement of the bars) and the average speed (we'll use this for the length of the bar). So something like this:



	SEGMENT_ID	SPEED
62	82	12.830468
32	51	13.075874
12	31	11.920569
76	96	21.659751
46	66	14.857143

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. Make sure your function returns a ***series***.

```
In [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 t
             """
             avg_speed = df.groupby(['SEGMENT_ID']).mean()
             avg_speed = avg_speed['SPEED']

             return avg_speed
             # YOUR CODE HERE
             #raise NotImplementedError()
         average_speed_per_segment(df)
```

```
Out[10]: SEGMENT_ID
         19    14.262480
         20    16.091586
         21    13.947909
         22    13.605072
         23    13.582444
                 ...
         93    13.387580
         94    14.295921
         95    14.440354
         96    20.697757
         97    18.142383
         Name: SPEED, Length: 78, dtype: float64
```

In [11]:

```
# reset to a "clean" segment_rows
segment_rows = get_group_first_row(hist_con, ['MONTH','DAY_OF_WEEK', 'HOUR'

# calculate the average speed per segment
average_speed = average_speed_per_segment(segment_rows)

# ADD YOUR TESTS HERE
assert type(average_speed) == pd.core.series.Series

# check what's in average_speed
average_speed
```

Out[11]:

```
SEGMENT_ID
19    13.730290
20    16.354475
21    13.870777
22    12.951986
23    13.006520
        ...
93    13.739182
94    14.531713
95    14.537048
96    20.583877
97    18.473622
Name: SPEED, Length: 78, dtype: float64
```

If you got things right, the **series** should look something like this (assets/average_speed.png). You might want to write a test to make sure you are returning the expected type. For example:

```
assert type(average_speed) == pd.core.series.Series
```

In [12]:

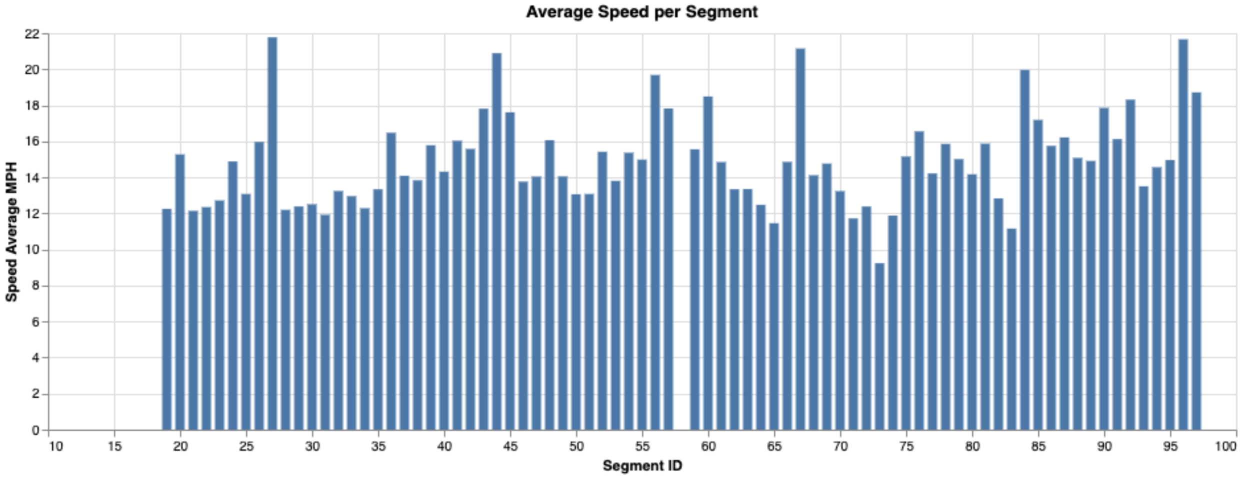
```
# make a dataframe from the average_speed
def get_average_speed_df(indf):
    # input: indf the input data frame (like hist_con)
    indf = hist_con
    # reset segment rows
    segment_rows = get_group_first_row(indf, ['MONTH','DAY_OF_WEEK', 'HOUR'

    # calculate the average speed
    average_speed = average_speed_per_segment(segment_rows)
    # create the data frame
    asdf = pd.DataFrame(average_speed).reset_index()
    #return the frame
    return asdf
```

In [13]:
```
# see what's inside
average_speed_df = get_average_speed_df(hist_con)

# ADD YOUR TESTS HERE
assert type(average_speed) == pd.core.series.Series

# print a sample
average_speed_df.sample(5)
```

Out[13]:

	SEGMENT_ID	SPEED
30	49	14.352104
58	78	13.797273
35	54	14.534677
61	81	16.307054
20	39	16.271488

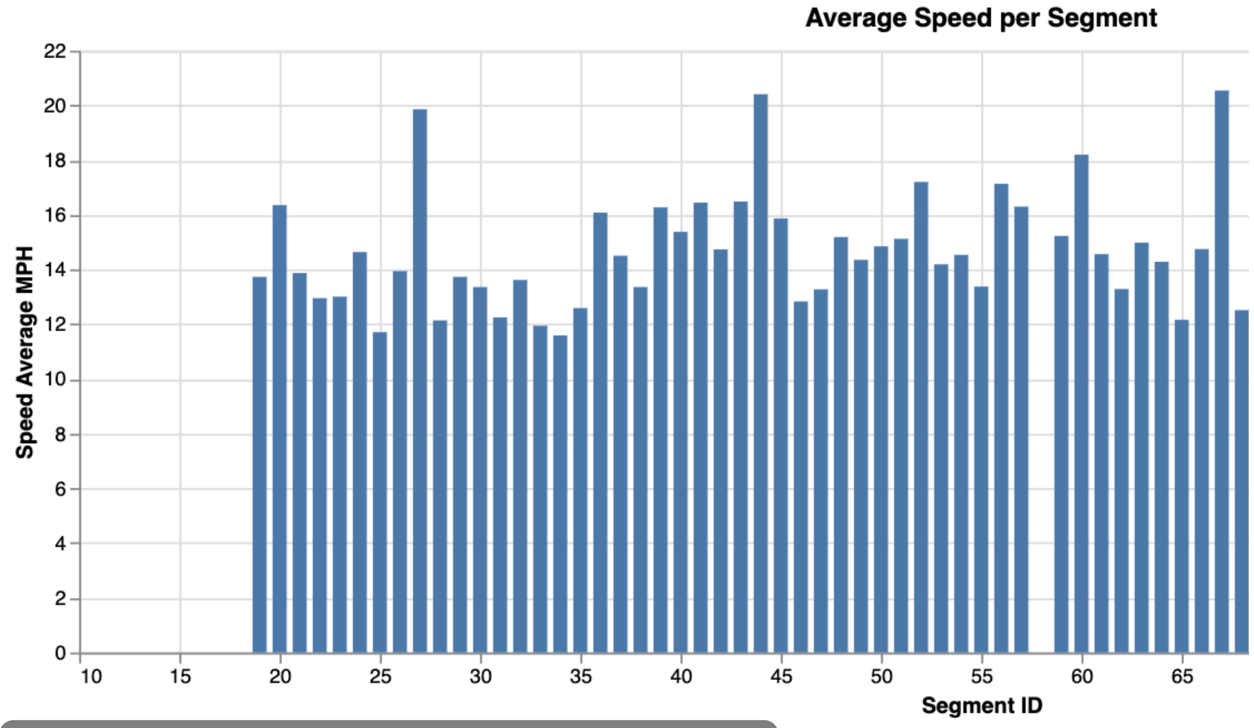
Ok, now we can build our visualization. If your code is correct, you should seem something like:



In [14]:

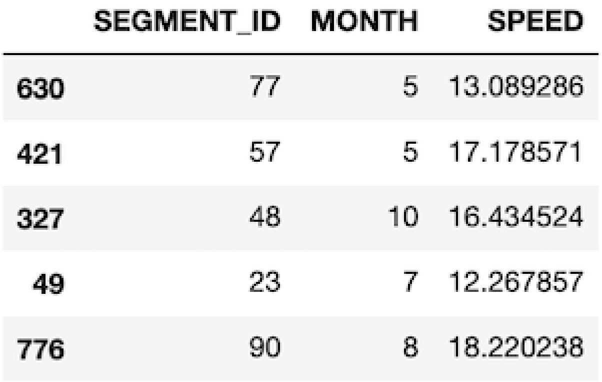
```
# let's generate the visualization

def create_average_speed_per_segment_vis(visdf):
    # visdf: frame to visualize

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

    # we're going to "encode" the variables, more on this next assignment
    encoding = base.encode(
        x= alt.X(                   # encode SEGMENT_ID as a quantiative varia
                'SEGMENT_ID:Q',
                title='Segment ID',
                scale=alt.Scale(zero=False)    # we don't need to start at 0
        ),
        y=alt.Y(
                'sum(SPEED):Q',    # encode the sum of speed for the segment
                title='Speed Average MPH'
        ),
    )

    # we're going to use a bar chart and set various parameters (like bar s
    return encoding.mark_bar(size=7).properties(title='Average Speed per Se

create_average_speed_per_segment_vis(average_speed_df)
```

Out[14]:



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. We're going to use a heatmap style calendar visualization (think GitHub) check-in history. Our encoder is going to make a square for each segment/month. The segment id will tell us where on the x-axis to put the square and the month value will tell us where on the y-axis. We will also want the mean speed as a column (for that month/segment) which we'll encode using color. What we're working towards is a dataframe that looks something like:



	SEGMENT_ID	MONTH	SPEED
630	77	5	13.089286
421	57	5	17.178571
327	48	10	16.434524
49	23	7	12.267857
776	90	8	18.220238

We're going to do part of this for you. First, we need you to use a pivot table to get us part way there. For the pivot table we want a table where the index is the month, and each column is a segment id. We will put the average speed in the cells.

Complete the `create_pivot_table` function for this. The table you output should look something like [this (assets/pivot_table.png)](assets/pivot_table.png)

In [15]:
```
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
    """
    avg_speed = df.groupby(['SEGMENT_ID', 'MONTH']).mean()
    avg_speed = avg_speed['SPEED']
    avg_speed = avg_speed.reset_index()
    avg_speed = avg_speed.pivot(index='MONTH', columns='SEGMENT_ID', values

    return avg_speed

create_pivot_table(df)
    # YOUR CODE HERE
    #raise NotImplementedError()
```

Out[15]:
```
SEGMENT_ID         19          20          21          22          23          24          25
     MONTH
         2  13.732143   16.732143   11.910714   14.053571   14.035714   17.892857   13.625000   16.3214
         3  13.696891   15.751531   13.596561   13.450777   13.125765   14.778380   11.543335   13.7326
         4  14.061928   16.287693   13.964986   13.725895   13.561275   14.963157   11.776065   14.0799
         5  13.949210   16.166464   13.574241   13.415310   13.048603   14.879222   11.334629   13.7642
         6  13.828460   15.916667   13.503168   13.014620   13.123538   14.423733   11.328947   13.4220
         7  13.559178   15.578077   13.136310   12.795653   13.008268   14.071108   10.940704   13.3753
         8  15.073286   16.416864   14.658786   14.190439   14.126872   15.925401   11.957709   14.3596
         9  14.652677   16.147612   13.756392   13.508683   13.220936   15.138688   11.406175   13.9126
        10  15.353257   16.575298   14.929022   14.075181   14.194023   16.096428   12.356993   14.5370
        11  14.897021   16.529019   14.694658   14.398562   14.521572   16.328197   13.125064   14.6183
        12  13.651316   15.620536   13.787359   13.573543   13.971335   14.970630   12.178102   13.6118

11 rows × 78 columns
```

In [16]:
```
# go back to our original sample for segment_rows
segment_rows = get_group_first_row(hist_con, ['MONTH','DAY_OF_WEEK', 'HOUR'
```

In [17]:
```
# run the code and see what's in the table
pivot_table = create_pivot_table(segment_rows)
pivot_table
```

Out[17]:

SEGMENT_ID	19	20	21	22	23	24	25	
MONTH								
2	20.142857	11.857143	8.142857	11.285714	15.857143	21.428571	13.714286	17.5714
3	14.892857	15.523810	12.589286	12.809524	11.488095	13.160714	11.255952	12.4821
4	11.583333	16.261905	13.172619	12.000000	13.571429	13.380952	11.541667	14.3690
5	12.946429	16.410714	13.041667	13.029762	10.452381	14.535714	10.428571	13.3571
6	14.053571	16.678571	12.976190	13.428571	12.369048	14.869048	12.101190	13.4047
7	13.833333	15.214286	14.684524	13.011905	12.934524	13.607143	10.857143	14.1190
8	11.994048	16.565476	15.083333	12.904762	14.184524	16.327381	13.083333	14.8273
9	14.940476	17.059524	13.422619	12.684524	13.261905	14.791667	10.047619	13.3154
10	13.839286	15.803571	13.726190	12.196429	13.702381	15.464286	11.964286	14.8392
11	14.988095	16.255952	15.726190	14.029762	13.244048	15.130952	12.797619	14.0535
12	13.964286	17.958333	14.523810	13.494048	14.738095	14.845238	12.934524	14.4940

11 rows × 78 columns

As before, we can write a "test" based on this example. For example, here (assets/pivot_table.png) we see that in March (Month 3) segment 21 had a value of ~11.696, so we could write the test:

```
assert round(pivot_table.loc[3,21],3) == 11.696
```

In [18]:
```
# add your tests here
assert round(pivot_table.loc[3,21],3) == 12.589
```

In [19]:
```
# we're going to implement a transformation to put the pivot table into a '
# is easier to specify the visualization.
def make_long_form(sourceTable):
    # sourceTable: the original table to modify
    hm_pivot_table = sourceTable.copy().unstack().reset_index()
    hm_pivot_table['SPEED'] = hm_pivot_table[0]
    hm_pivot_table.drop(0,axis=1,inplace=True)
    return(hm_pivot_table)
```

In [20]:

```python
# you can see what's inside the long form
longformASSM = make_long_form(pivot_table)
longformASSM.sample(5)
```

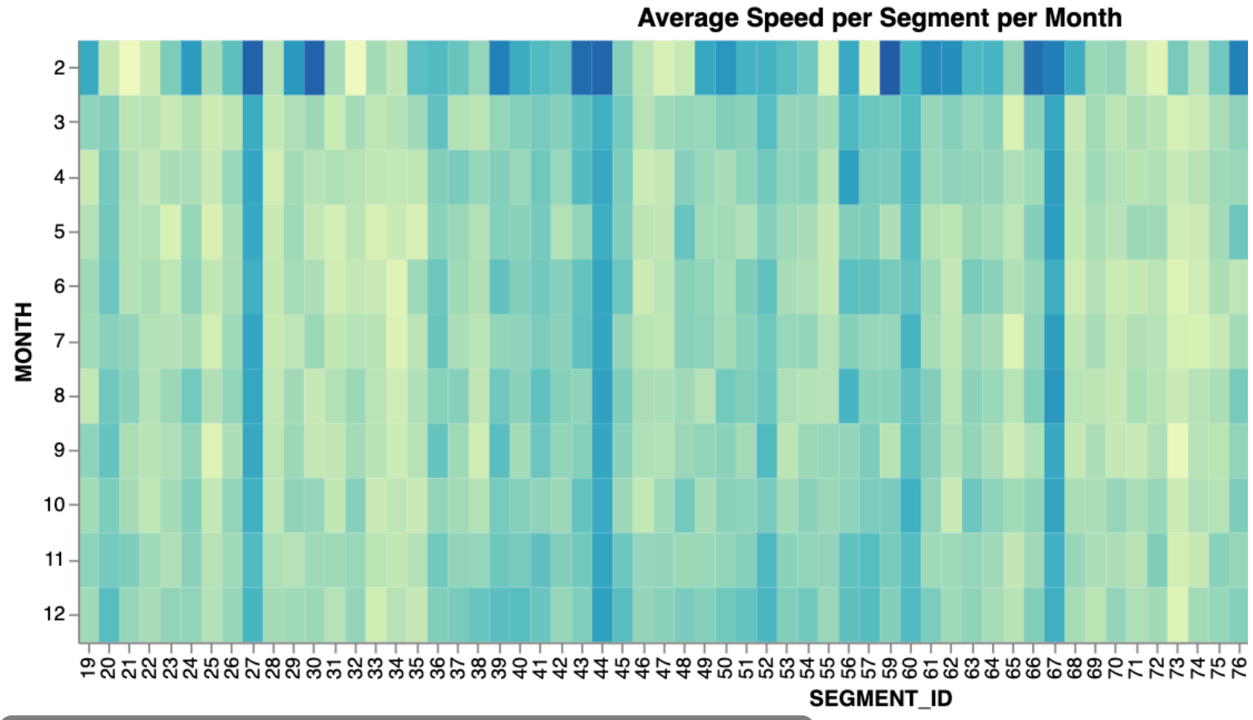
Out[20]:

	SEGMENT_ID	MONTH	SPEED
427	57	11	17.952381
728	86	4	15.101190
551	70	3	12.535714
672	81	3	14.869048
280	44	7	20.452381

In [21]:

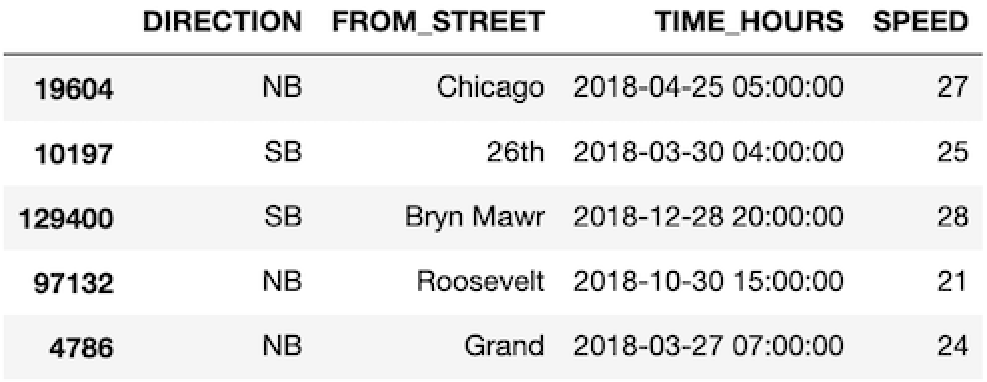
```python
# create the visualization. We're going to use rectangles (a heat map of so
# figure out the horizontal placement (x), the month as the vertical (y) an
def create_speed_month_segment_vis(visframe):
    # visframe: the frame to visualize

    # using rectangles
    encoding = alt.Chart(visframe).mark_rect().encode(
        x='SEGMENT_ID:O',   # segments on the x axis, ordinal encoding so o
        y='MONTH:O',        # month, ordinal encoding so ordered
        color='SPEED:Q'     # color based on speed, quantitative encoding
    )

    # adjust title, height, width and return
    return encoding.properties(title='Average Speed per Segment per Month',

create_speed_month_segment_vis(longformASSM)
```

Out[21]:



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. In the end, we'll want something roughly like:



	DIRECTION	FROM_STREET	TIME_HOURS	SPEED
19604	NB	Chicago	2018-04-25 05:00:00	27
10197	SB	26th	2018-03-30 04:00:00	25
129400	SB	Bryn Mawr	2018-12-28 20:00:00	28
97132	NB	Roosevelt	2018-10-30 15:00:00	21
4786	NB	Grand	2018-03-27 07:00:00	24

To do this:

- We're going to need to sort the dataframe by one or more columns (this is the `sort_by_col` 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`)

In [22]:

```python
def sort_by_col(df, sorting_columns):
    """Sort the rows of df by the columns (sorting_columns)
    return the sorted dataframe
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    return df.sort_values(sorting_columns)
```

In [23]:

```
# test it out
segment_rows = sort_by_col(segment_rows, ['SEGMENT_ID'])
segment_rows.sample(5)
```

Out[23]:

	MONTH	DAY_OF_WEEK	HOUR	SEGMENT_ID	TIME	SPEED	STREET	DIRECTION
109070	11	2	23	45	11/19/2018 11:10:18 PM	20	Pulaski	NB
82741	9	2	21	81	09/24/2018 09:20:46 PM	18	Pulaski	SB
123455	12	3	15	79	12/25/2018 03:20:44 PM	19	Pulaski	SB
116801	11	7	2	54	11/24/2018 02:20:40 AM	-1	Pulaski	NB
111075	11	4	1	22	11/14/2018 01:50:20 AM	-1	Pulaski	NB

In [24]:

```
def time_to_hours(df):
    """ Add a column (called TIME_HOURS) based on the data in the TIME colu
    the value to the nearest hour.  For example, if the original TIME row s
    ‘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 c
    actually a proper datetime and not a string).The column should be a dat
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    df['TIME_HOURS'] = df['TIME'].astype('datetime64[h]')
    return df

time_to_hours(df)
```

Out[24]:

	TIME	SEGMENT_ID	SPEED	STREET	DIRECTION	FROM_STREET	TO_STREET	HO
0	12/31/2018 11:50:23 PM	83	-1	Pulaski	SB	Lake	Washington	
1	12/31/2018 11:50:23 PM	84	20	Pulaski	SB	Chicago	Lake	
2	12/31/2018 11:50:19 PM	78	27	Pulaski	SB	Cermak	26th	
3	12/31/2018 11:50:19 PM	79	27	Pulaski	SB	16th	Cermak	
4	12/31/2018 11:50:19 PM	80	27	Pulaski	SB	Roosevelt	16th	
...	...	...	...	...	...	...	...	
3195445	02/28/2018 05:01:00 PM	95	22	Pulaski	SB	Foster	Lawrence	
3195446	02/28/2018 05:01:00 PM	96	33	Pulaski	SB	Bryn Mawr	Foster	
3195447	02/28/2018 05:01:00 PM	96	33	Pulaski	SB	Bryn Mawr	Foster	
3195448	02/28/2018 05:01:00 PM	97	26	Pulaski	SB	Peterson	Bryn Mawr	
3195449	02/28/2018 05:01:00 PM	97	26	Pulaski	SB	Peterson	Bryn Mawr	

3195450 rows × 11 columns

In [25]:
```
# we can test this out
segment_rows = time_to_hours(segment_rows)
segment_rows.sample(5)
```

Out[25]:

	MONTH	DAY_OF_WEEK	HOUR	SEGMENT_ID	TIME	SPEED	STREET	DIRECTION
116358	11	6	20	80	11/02/2018 08:40:07 PM	-1	Pulaski	SB
72592	8	4	11	72	08/29/2018 11:10:22 AM	26	Pulaski	SB
86434	9	4	21	29	09/19/2018 09:20:45 PM	23	Pulaski	NB
14389	4	1	9	56	04/22/2018 09:30:53 AM	41	Pulaski	NB
15001	4	1	17	44	04/08/2018 05:20:36 PM	-1	Pulaski	NB

In [26]:
```
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 orientati
    df: original traffic data frame
    traffic_orientation: a string, one of "SB" or "NB"
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    df['DIRECTION'] == traffic_orientation
    df = df[df['DIRECTION']== traffic_orientation]
    return df
```

In [27]:
```
# let's filter down to a south bound and north bound table
sb = filter_orientation(segment_rows, 'SB')
nb = filter_orientation(segment_rows, 'NB')
```

The sb table should look like this (assets/sb.png)

In [28]:

```
# let's look at a sample. You might want to build some assert tests here
sb.sample(5)
```

Out[28]:

	MONTH	DAY_OF_WEEK	HOUR	SEGMENT_ID	TIME	SPEED	STREET	DIRECTION
105197	10	7	21	73	10/06/2018 09:50:21 PM	-1	Pulaski	SB
3397	3	2	12	63	03/05/2018 12:50:21 PM	23	Pulaski	SB
127052	12	5	13	88	12/06/2018 01:30:59 PM	21	Pulaski	SB
77135	8	6	21	91	08/03/2018 09:40:07 PM	-1	Pulaski	SB
71179	8	3	17	63	08/07/2018 05:10:26 PM	25	Pulaski	SB

In [29]:

```
# let's put it all together to generate our table
def get_sbnb(indf):
    # input: indf, a hist_con shaped data frame

    # go back to our original sample for segment_rows
    segment_rows = get_group_first_row(indf, ['MONTH','DAY_OF_WEEK', 'HOUR'

    # use our new functions
    segment_rows = sort_by_col(segment_rows, ['SEGMENT_ID'])
    segment_rows = time_to_hours(segment_rows)
    sb = filter_orientation(segment_rows, 'SB')
    nb = filter_orientation(segment_rows, 'NB')

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

    # now append the columns and just select the sub columns we care about
    sbnb = sb.append(nb)[['DIRECTION','FROM_STREET','TIME_HOURS','SPEED']]
    return(sbnb)
```

```
In [30]: # let's see what's inside
         sbnb = get_sbnb(hist_con)
         sbnb.sample(5)
```

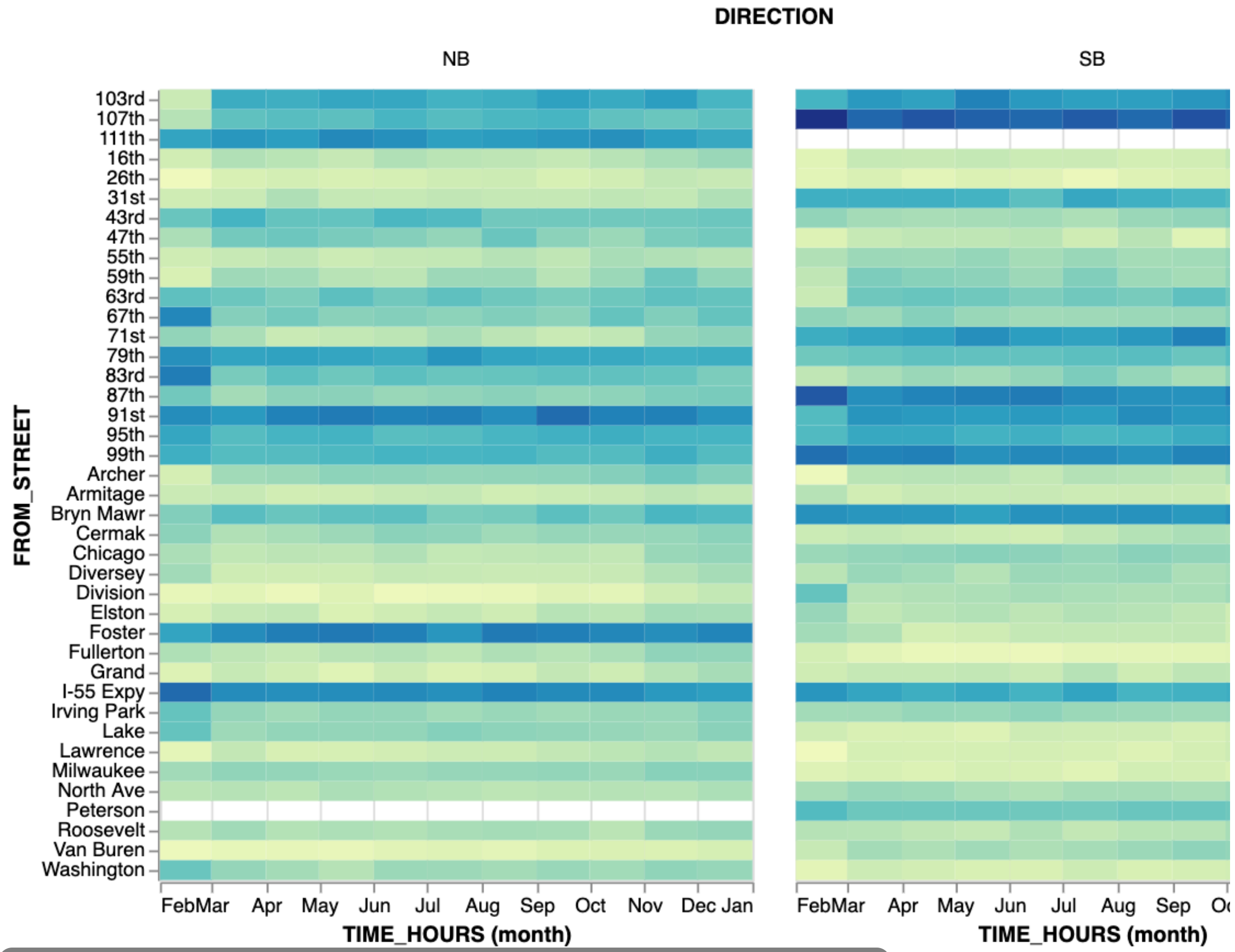
Out[30]:

	DIRECTION	FROM_STREET	TIME_HOURS	SPEED
108692	NB	Bryn Mawr	2018-11-12 18:00:00	23
34039	NB	Fullerton	2018-05-02 21:00:00	29
63062	NB	Bryn Mawr	2018-07-27 09:00:00	24
94640	NB	Chicago	2018-10-15 06:00:00	23
21800	NB	Bryn Mawr	2018-04-26 08:00:00	19

In [31]:

```
# create the visualization, but it's your bonus (2.5) to describe what's go
def create_speed_direction_vis(visdf):
    alt.data_transformers.disable_max_rows()
    return alt.Chart(visdf).mark_rect().encode(
        x='month(TIME_HOURS):T',
        y='FROM_STREET:N',
        color='mean(SPEED):Q',
        facet='DIRECTION:N'
    ).properties(
        width=300,
        height=400
    )

create_speed_direction_vis(sbnb)
```

Out[31]:



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.

2.5 Answer

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

In [32]:

```
# load the crash data
crashes = read_csv('assets/Traffic.Crashes.csv.gz')

# just grab the pulaski road data
crashes_pulaski = crashes[crashes.STREET_NAME == 'PULASKI RD']
```

2.6 Calculate summary statistics for grouped streets (15 points)

We want to get a few summary visualizations like where crashes are happening on Pulaski Road (by which house number). We're going to bin the records by house number to start. Think of bins as vaguely representing "street blocks" (it's obviously not quite right).

- Group the streets every 300 units (street numbers). Hint: You can use the `pd.cut` function

The second visualization will tell us around which houses accidents are happening.

- 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

In [33]:

```
bin_values = np.arange(0,15000,300)
df = crashes_pulaski

def bin_crashes(df):
    """ Assign each crash instance a category (bin) every 300 house number
    Return a new dataframe with a column called BIN where each value is the
    i.e. 0 is the label for records with street number n, where 1 <= n <= 3
    300 is the label for records with n at 301 <= n <= 600, and so on.
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    df['BIN'] = pd.cut(crashes_pulaski['STREET_NO'], bin_values, labels=bin
    return df
```

In [34]:

```python
binned_df = bin_crashes(crashes_pulaski)

# sample the values to see what's in your new DF (we only care about street
binned_df.sample(5)[['STREET_NO','BIN']]
```

Out[34]:

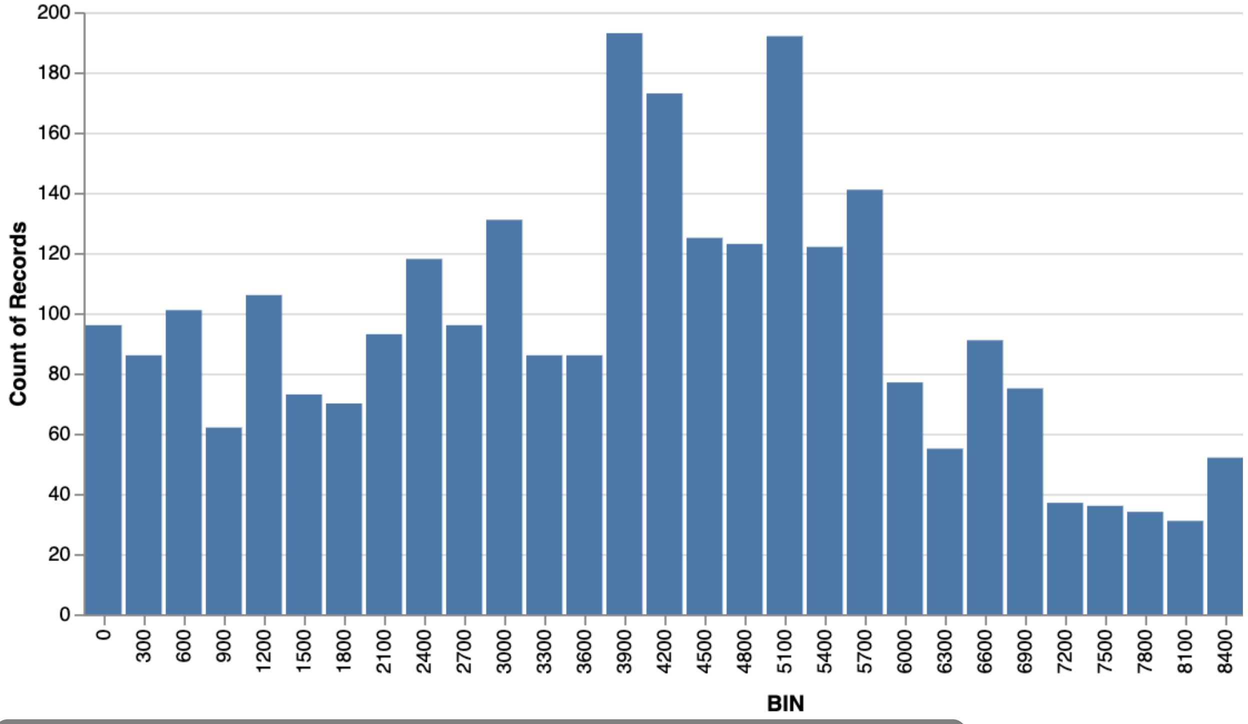
	STREET_NO	BIN
59136	3828	3600
2966	3922	3900
50204	728	600
74374	7030	6900
93295	101	0

A sample of the relevant columns from the table would look something like [this (assets/binned_df.png)](assets/binned_df.png). We can also create a histogram of street numbers to see which are the most prevalent. It should look something like [this (assets/street_no.png)](assets/street_no.png).

In [35]:

```python
def create_street_histogram_vis(visf):
    # create this vis
    return alt.Chart(binned_df).mark_bar().encode(
        alt.X('BIN'),
        alt.Y('count()')
    )

create_street_histogram_vis(bin_crashes(crashes_pulaski))
```

Out[35]:



In [36]:
```
binned_df = bin_crashes(crashes_pulaski)
def calculate_group_aggregates(df):
    """
    There are *accidents* and *injuries* (could be 0 people got hurt, could
    There’s one row per accident at the moment, so we want to know how many
    happened in each BIN/STREET_DIRECTION (this will be the count) and how r

    Return a df with the count of accidents in a column named 'ACCIDENT_COUN
    bin (the count) and how many injuries (the sum) in a column named 'INJUR

    Replace NaN with 0
    """

    cleaned_injuries = df.groupby(['BIN', 'STREET_DIRECTION']).sum() # SUM
    cleaned_injuries = cleaned_injuries.reset_index()
    cleaned_injuries['INJURIES_SUM'] = cleaned_injuries['INJURIES_TOTAL']
    cleaned_injuries = cleaned_injuries.drop(['INJURIES_TOTAL'], axis=1)

    cleaned_accidents = df.groupby(['BIN']).count() #COUNT
    cleaned_accidents = cleaned_accidents.drop(['STREET_DIRECTION', 'INJURIE
    cleaned_accidents['ACCIDENT_COUNT']= cleaned_accidents['RD_NO']
    cleaned_accidents = cleaned_accidents.drop(['RD_NO'], axis=1)

    merged = cleaned_injuries.merge(cleaned_accidents, how='right', on='BIN
    merged = merged.fillna(0)

    merged = merged.drop(['POSTED_SPEED_LIMIT_x', 'LANE_CNT_x', 'STREET_NO_x
                         'INJURIES_FATAL_x', 'INJURIES_INCAPACITATING_x', 'I
                         'INJURIES_REPORTED_NOT_EVIDENT_y', 'INJURIES_NO_INI
                         'CRASH_HOUR_y', 'CRASH_DAY_OF_WEEK_y', 'CRASH_MONTH
                         'LOCATION','INJURIES_NON_INCAPACITATING_y', 'INJURI
                         'MOST_SEVERE_INJURY', 'NUM_UNITS_y', 'WORKERS_PRESE
                         'DOORING_I', 'INJURIES_REPORTED_NOT_EVIDENT_x','INJ
                         'CRASH_HOUR_x', 'CRASH_DAY_OF_WEEK_x', 'CRASH_MONTH
                         'PHOTOS_TAKEN_I', 'BEAT_OF_OCCURRENCE_y', 'STREET_N
                         'PRIM_CONTRIBUTORY_CAUSE', 'DATE_POLICE_NOTIFIED',
                         'INTERSECTION_RELATED_I', 'CRASH_TYPE', 'DEVICE_CON
                         'POSTED_SPEED_LIMIT_y', 'REPORT_TYPE', 'ROAD_DEFECT
                         'LANE_CNT_y', 'TRAFFICWAY_TYPE', 'FIRST_CRASH_TYPE
                         'CRASH_DATE', 'CRASH_DATE_EST_I', 'LONGITUDE_x', 'L

    return merged

calculate_group_aggregates(df)
    # YOUR CODE HERE
    #raise NotImplementedError()
```

Out[36]:
```
   BIN  STREET_DIRECTION  INJURIES_SUM  ACCIDENT_COUNT
0    0                 N          18.0              96
1    0                 S          17.0              96
2  300                 N          19.0              86
```

	BIN	STREET_DIRECTION	INJURIES_SUM	ACCIDENT_COUNT
3	300	S	17.0	86
4	600	N	19.0	101
...	...	...	...	...
93	13800	S	0.0	0
94	14100	N	0.0	0
95	14100	S	0.0	0
96	14400	N	0.0	0
97	14400	S	0.0	0

98 rows × 4 columns

In [37]:

```
aggregates = calculate_group_aggregates(binned_df)

# check the data
aggregates.head(15)

#aggregates.sample(15)
```

Out[37]:

	BIN	STREET_DIRECTION	INJURIES_SUM	ACCIDENT_COUNT
0	0	N	18.0	96
1	0	S	17.0	96
2	300	N	19.0	86
3	300	S	17.0	86
4	600	N	19.0	101
5	600	S	9.0	101
6	900	N	15.0	62
7	900	S	7.0	62
8	1200	N	36.0	106
9	1200	S	13.0	106
10	1500	N	12.0	73
11	1500	S	16.0	73
12	1800	N	8.0	70
13	1800	S	3.0	70
14	2100	N	4.0	93

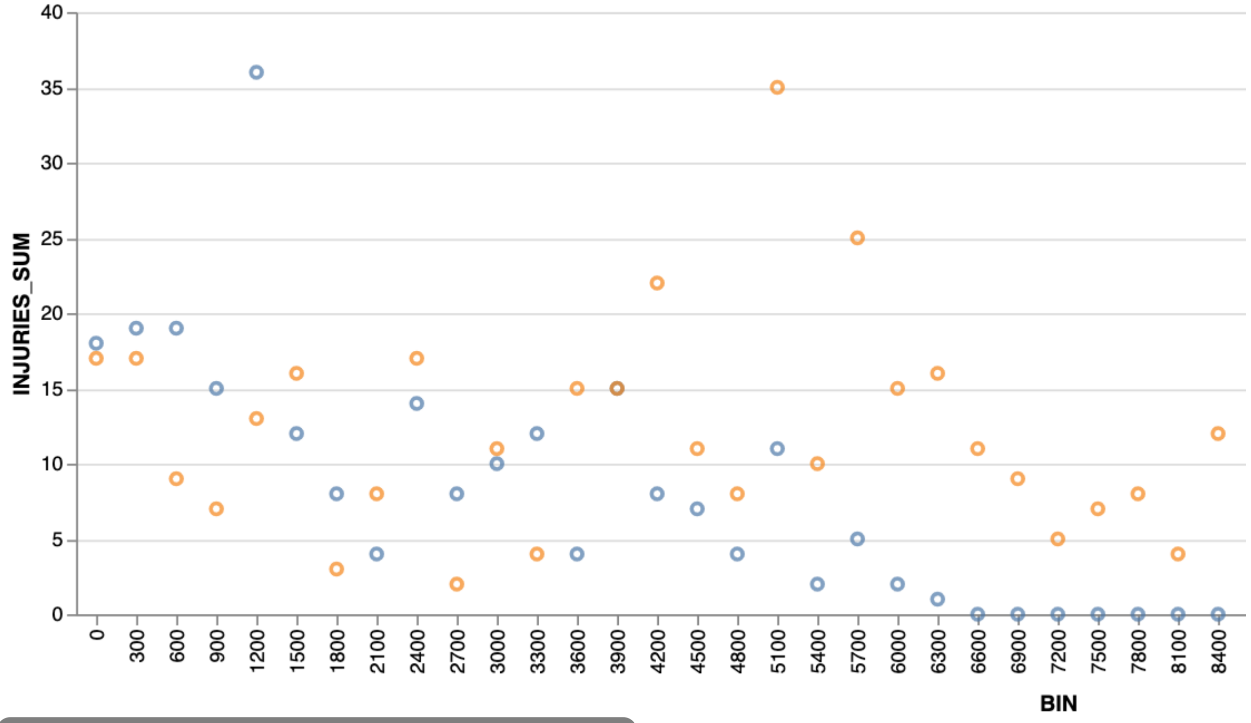
The table should look like this (assets/2.6_aggregate_1.png)

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 (assets/direction_injuries.png)](assets/direction_injuries.png)

In [38]:

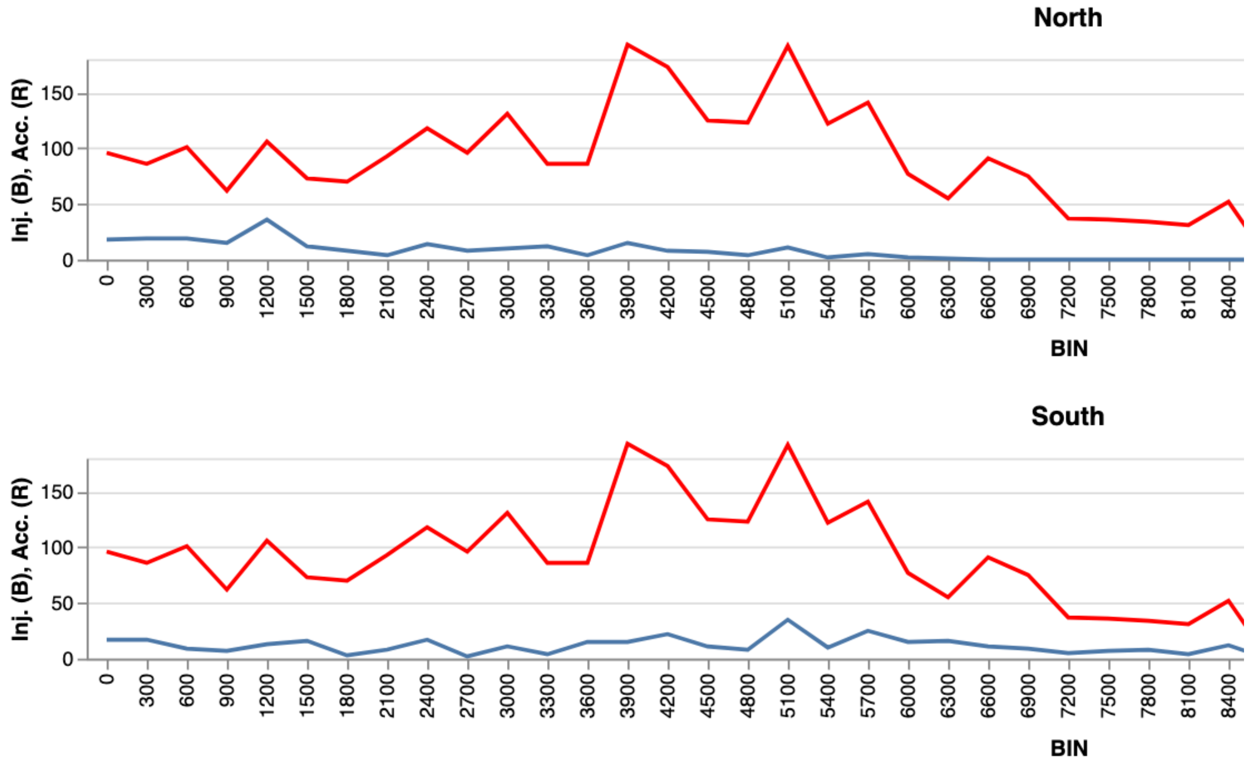
```
def create_injuries_sum_chart_vis(visdf):
    return alt.Chart(visdf).mark_point().encode(
        alt.Color('STREET_DIRECTION'),
        alt.X('BIN'),
        alt.Y('INJURIES_SUM')
    )

create_injuries_sum_chart_vis(aggregates)
```

Out[38]:



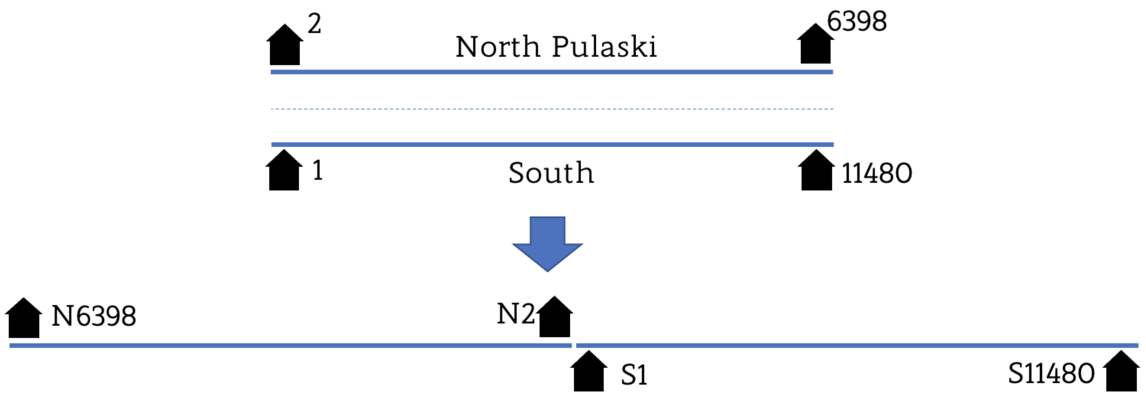
In [39]:
```
# we can also look at the differences between injuries and accidents for a
# both directions so you can see the difference

def create_injuries_vs_accident_vis(visdf,chart_title):
    c1 = alt.Chart(visdf).mark_line().encode(
        alt.X('BIN'),
        alt.Y('INJURIES_SUM',scale=alt.Scale(domain=(0, 170)), title='Inj.
    )

    c2 = c1.mark_line(color='red').encode(
        alt.Y('ACCIDENT_COUNT',scale=alt.Scale(domain=(0, 170)), title='Acc
    )
    return (c1+c2).properties(title=chart_title,height=100)

def create_compound_i_vs_a_vis(visdf):
    north = create_injuries_vs_accident_vis(aggregates[aggregates.STREET_DI
    south = create_injuries_vs_accident_vis(aggregates[aggregates.STREET_DI
    return north & south

create_compound_i_vs_a_vis(aggregates)
```

Out[39]:



2.7 Sort the street ranges (10 points)

Because the street has both North and South addresses we are going to "stretch" it so the bins range from the highest North street value down to the lowest and then going from lowest South to highest South. Something like this (but we're going to used the binned values, instead of the "raw" house numbers, in the end):



Altair will use the sort order in the dataframe so if we sort the frame this way, that's what we'll have.

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

In [40]:

```
# pulaski_sort_order will be a useful way for you to bin
crashed_range = list(range(0, crashes_pulaski.STREET_NO.max()+1000, 300))
pulaski_sort_order = ['N ' + str(s) for s in crashed_range[::-1]] + ['S ' +
```

In [41]:

```
def categorical_sorting(df, sorder):
    """ Create a column called ORDER_LABEL that contains a concatenation of
    Set the sort order of this column to the provided sort array (sorder: t
    the same order of the array, we can pass in pulaski_sort_order as below
    Sort the dataframe (df) by this column
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    df['ORDER_LABEL'] = df['STREET_DIRECTION'].astype(str) + ' ' + df['BIN'
    df['ORDER_LABEL'] = pd.Categorical(df['ORDER_LABEL'], categories = sord
    df.sort_values(by = 'ORDER_LABEL')

    return df

#categorical_sorting(df, sorder)
```

In [42]:
```
print(pulaski_sort_order)
sorted_groups = categorical_sorting(aggregates, pulaski_sort_order)

# check the values
sorted_groups.head(15)

#test
#assert sorted_groups['ORDER_LABEL'].iloc[0] == sort_order[1]
#assert sorted_groups['ORDER_LABEL'].iloc[0] > sorted_groups['ORDER_LABEL']
```

```
['N 12300', 'N 12000', 'N 11700', 'N 11400', 'N 11100', 'N 10800', 'N 105
00', 'N 10200', 'N 9900', 'N 9600', 'N 9300', 'N 9000', 'N 8700', 'N 840
0', 'N 8100', 'N 7800', 'N 7500', 'N 7200', 'N 6900', 'N 6600', 'N 6300',
'N 6000', 'N 5700', 'N 5400', 'N 5100', 'N 4800', 'N 4500', 'N 4200', 'N
3900', 'N 3600', 'N 3300', 'N 3000', 'N 2700', 'N 2400', 'N 2100', 'N 180
0', 'N 1500', 'N 1200', 'N 900', 'N 600', 'N 300', 'N 0', 'S 0', 'S 300',
'S 600', 'S 900', 'S 1200', 'S 1500', 'S 1800', 'S 2100', 'S 2400', 'S 27
00', 'S 3000', 'S 3300', 'S 3600', 'S 3900', 'S 4200', 'S 4500', 'S 480
0', 'S 5100', 'S 5400', 'S 5700', 'S 6000', 'S 6300', 'S 6600', 'S 6900',
'S 7200', 'S 7500', 'S 7800', 'S 8100', 'S 8400', 'S 8700', 'S 9000', 'S
9300', 'S 9600', 'S 9900', 'S 10200', 'S 10500', 'S 10800', 'S 11100', 'S
11400', 'S 11700', 'S 12000', 'S 12300']
```

Out[42]:

	BIN	STREET_DIRECTION	INJURIES_SUM	ACCIDENT_COUNT	ORDER_LABEL
0	0	N	18.0	96	N 0
1	0	S	17.0	96	S 0
2	300	N	19.0	86	N 300
3	300	S	17.0	86	S 300
4	600	N	19.0	101	N 600
5	600	S	9.0	101	S 600
6	900	N	15.0	62	N 900
7	900	S	7.0	62	S 900
8	1200	N	36.0	106	N 1200
9	1200	S	13.0	106	S 1200
10	1500	N	12.0	73	N 1500
11	1500	S	16.0	73	S 1500
12	1800	N	8.0	70	N 1800
13	1800	S	3.0	70	S 1800
14	2100	N	4.0	93	N 2100

The table should look like [this (assets/sorted_groups.png)](assets/sorted_groups.png)

You can test your code a few ways. First, we gave you the sort order, so you know what the ORDER_LABEL of the first row should be:

```
assert sorted_groups['ORDER_LABEL'].iloc[0] == sort_order[1]
```

(it might be sort_order[0] depending on how you did the label)

You also know that the first item should be "greater" than the second, so you can test:

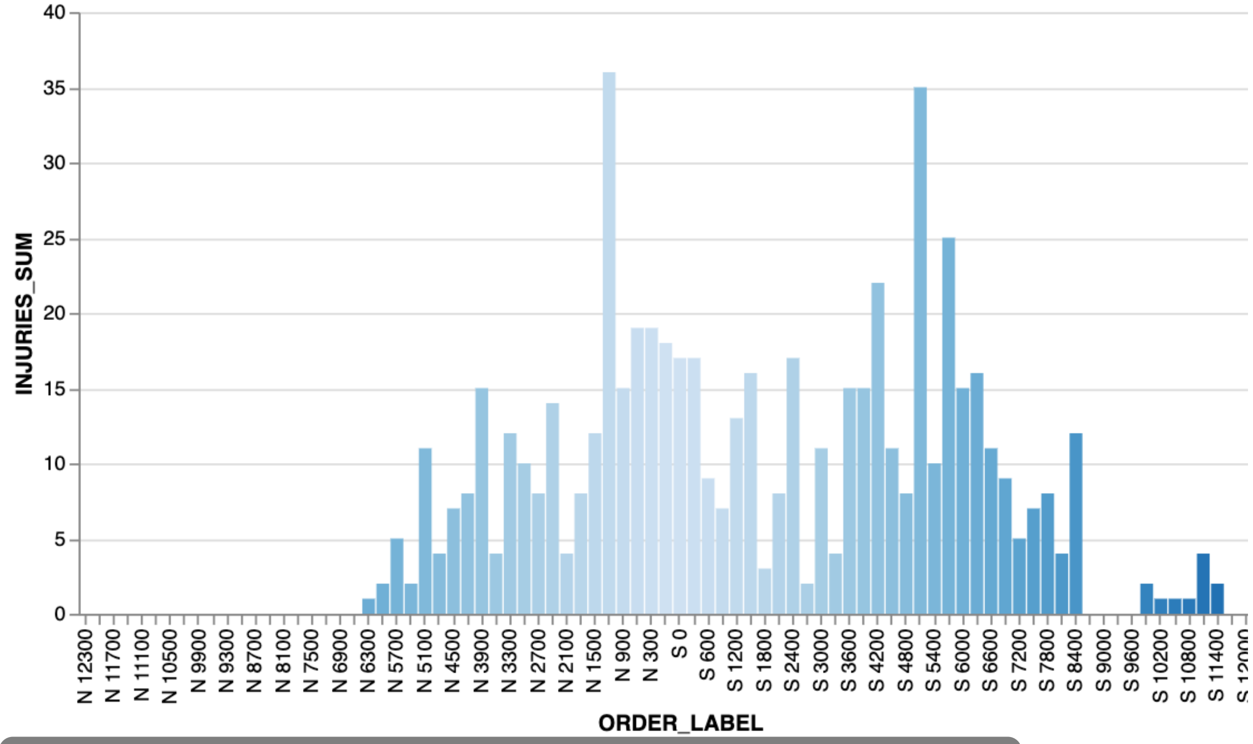
```
 assert sorted_groups['ORDER_LABEL'].iloc[0] >
sorted_groups['ORDER_LABEL'].iloc[1]
```

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 (assets/order_injuries.png)

If your X axis cutoffs are a bit different, that's fine.

```
In [43]: def create_sorted_pulaski_histogram_vis(visframe,sorder):
             # creates a histogram based on the calculated values in visframe
             # assumes an ORDER_LABEL, INJURIES_SUM, and BIN columns
             # color will double encode the bin value (which is the X)
             return alt.Chart(visframe).mark_bar().encode(
                 alt.X('ORDER_LABEL:O', sort=sorder),
                 alt.Y('INJURIES_SUM:Q'),
                 alt.Color('BIN:Q')
             ).properties(
                 width=600
             )

         create_sorted_pulaski_histogram_vis(sorted_groups,pulaski_sort_order)
```

Out[43]:

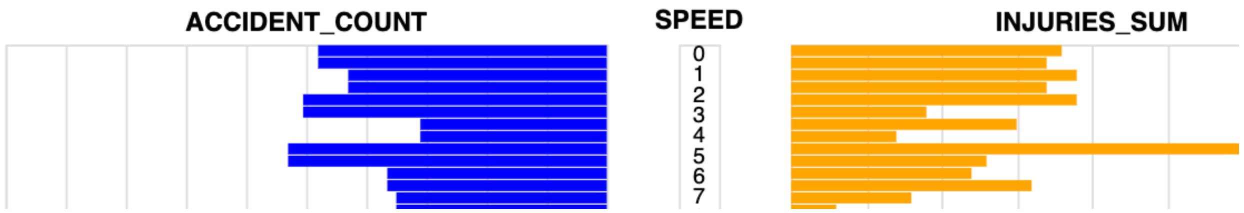


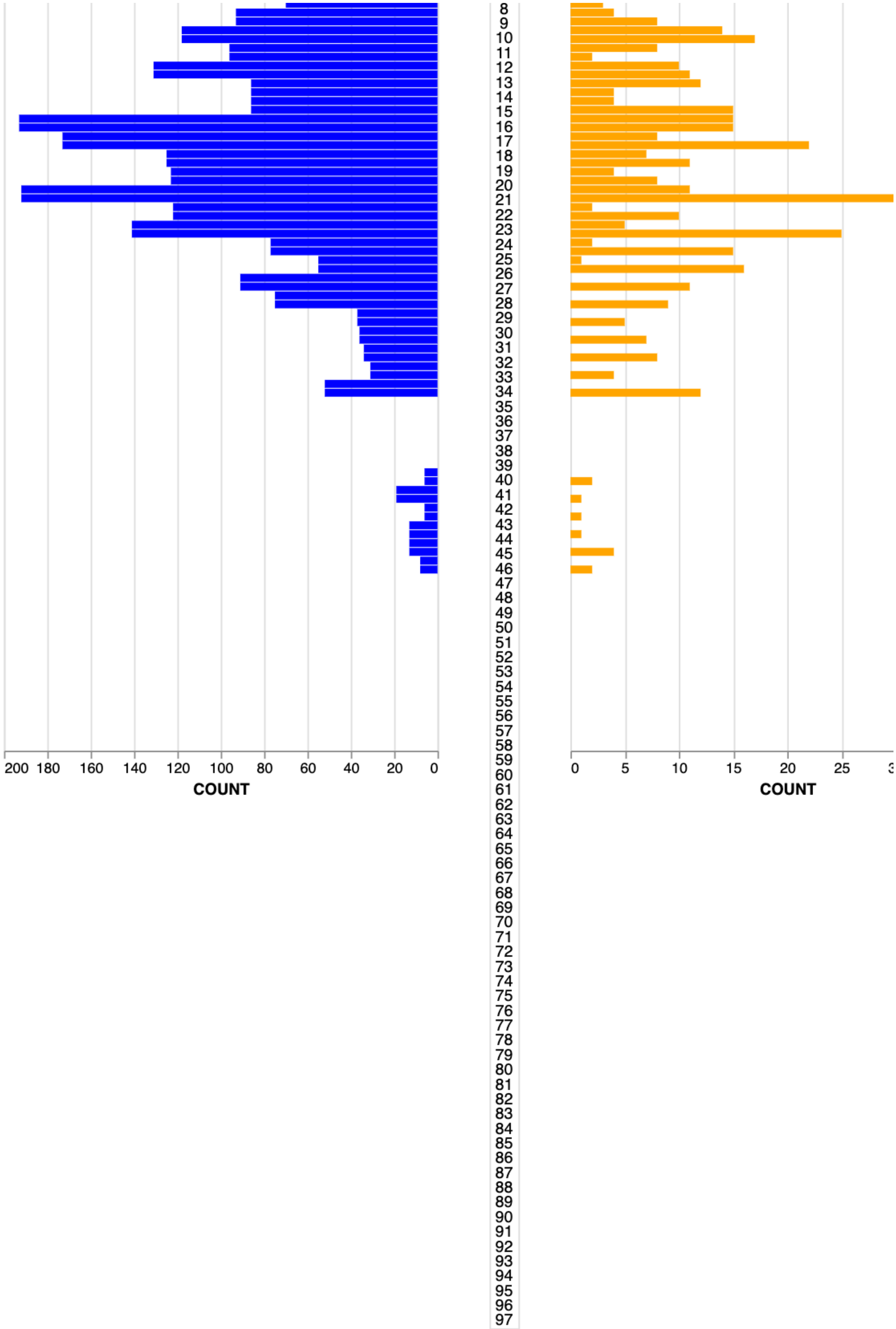
Ok, let's actually make a useful visualization using some of the dataframes we've created. As a bonus, we're going to ask you what you would use this for.

In [44]:

```
# to make the kind of chart we are interested in we're going to build it ou
# put them together at the end

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

# Note that we cheated a bit. The actual speed column (POSTED_SPEED) doesn'
# example, so we're using the level_1 variable (it's an index variable) as
# Just assume this actually is the speed at which the accident happened.

a = alt.Chart(bar_sorted_groups).mark_bar().transform_filter(alt.datum.TYPE
    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
    y=alt.Y('SPEED:O', axis=None),
    text=alt.Text('SPEED:Q')
).mark_text().properties(title='SPEED',
                         width=20,
                         height=1000)

# and the right most chart
c = alt.Chart(bar_sorted_groups).mark_bar().transform_filter(alt.datum.TYPE
    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
```

Out[44]:




8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
200 180 160 140 120 100 80 60 40 20 0
COUNT
0 5 10 15 20 25 3
COUNT

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. See the comment in the code about "speed."

2.8 Answer