Information Visualization I

School of Information, University of Michigan

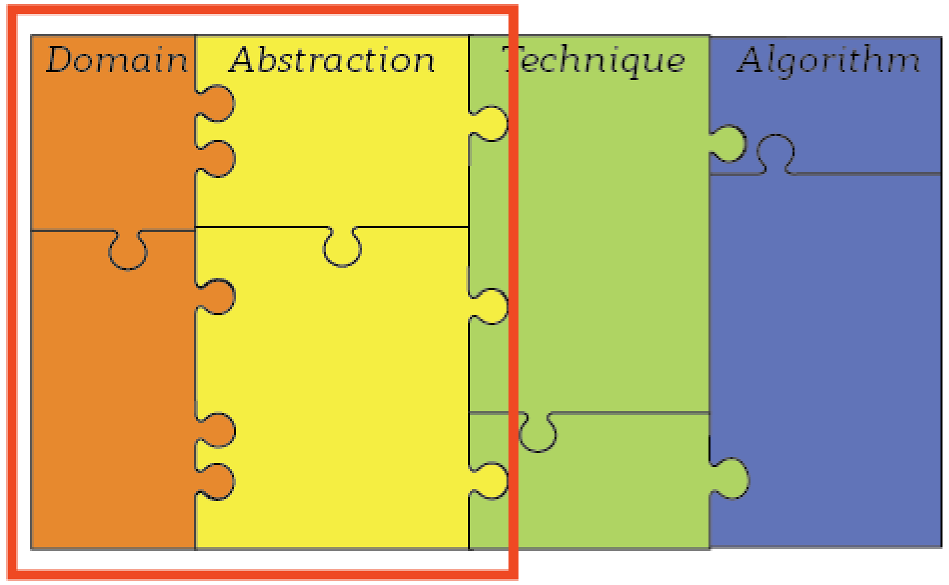
Week 1:

- 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/)](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)
- 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.

Important notes:

1) 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.

2) Pay attention to the 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)

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.

Remember: Domain tasks are questions an analyst (or reader) might need to figure out. 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)

The Chicago Metropolitan Agency for Planning (CMAP), the agency responsible for the study, likely had employees that were well familiar with the greater metropolitan area and had strong statistical and data analytic skills perform the analysis. Their expertise would have been in (big) data collection, storage, and aggregation, and data analysis with a strong ability to identify common trends and patterns within the data. Additionally, the analysts would also have had strong data visualization skills and an ability to not only identify patterns in the data but present the findings in a way that was easily digestible to perhaps a layman audience or an audience unfamiliar with the Chicago area. The goal of the article was to note and detail relationships between crash rates and congestion, as well as other transportation relationships as a means of persuasion. The article mentions "GO TO 2040" frequently throughout, suggesting that highlighting shortcomings in the current Chicago transportation infrastructure that lead to greater crash rates, congestion, etc. is a means of driving support for an initiative that hopes to improve and enhance the infrastructure.

The 3 examples of domain tasks featured in the article

1.) What is the crash frequency on Chicago's major expressways?

2.) What is the relationship between congestion and crash rates?

3.) Is the crash frequency on expressways relative to more arterial roads more or less frequent?

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

Domain Task: What is the crash frequency on Chicago's major expressways?

Corresponding Abstract Task: What is the rate of a quantitative variable?

Domain Task: What is the relationship between congestion and crash rates?

Corresponding Abstract Task: What is the correlation between two variables?

Domain Task: Is the crash frequency on expressways relative to more arterial roads more or less frequent?

Abstract Task: Is the rate of a quantitative variable higher or lower relative to a rate of another quantitative variable?

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 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 (https://pandas.pydata.org/pandas-docs/stable/reference/index.html)).

In [4]:
```python
def read_csv(filename):
    """Read the csv file from filename (uncompress 'gz' if needed)
    return the dataframe resulting from reading the columns
    """
    df = pd.read_csv(filename)
    return df
    #raise NotImplementedError()
```

In [5]:
```python
# 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)
```

In [6]:
```python
def get_group_first_row(df, grouping_columns):
    """Group rows using 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
    """
    first_vals = df.groupby(grouping_columns).first().reset_index()
    return first_vals
    #raise NotImplementedError()
```

In [7]:
```python
# 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 (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.

In [8]:
```
#hidden tests are within this cell
```

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

In [9]:
```
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
    """
    return df.groupby('SEGMENT_ID').agg({'SPEED' : 'mean'}).squeeze()
    #raise NotImplementedError()
```

```
In [10]: # 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()

         # check what's in average_speed
         average_speed
```

```
Out[10]: SEGMENT_ID
         19    12.251926
         20    15.274452
         21    12.141079
         22    12.346769
         23    12.716657
                 ...
         93    13.503260
         94    14.560759
         95    14.959099
         96    21.659751
         97    18.714286
         Name: SPEED, Length: 78, dtype: float64
```

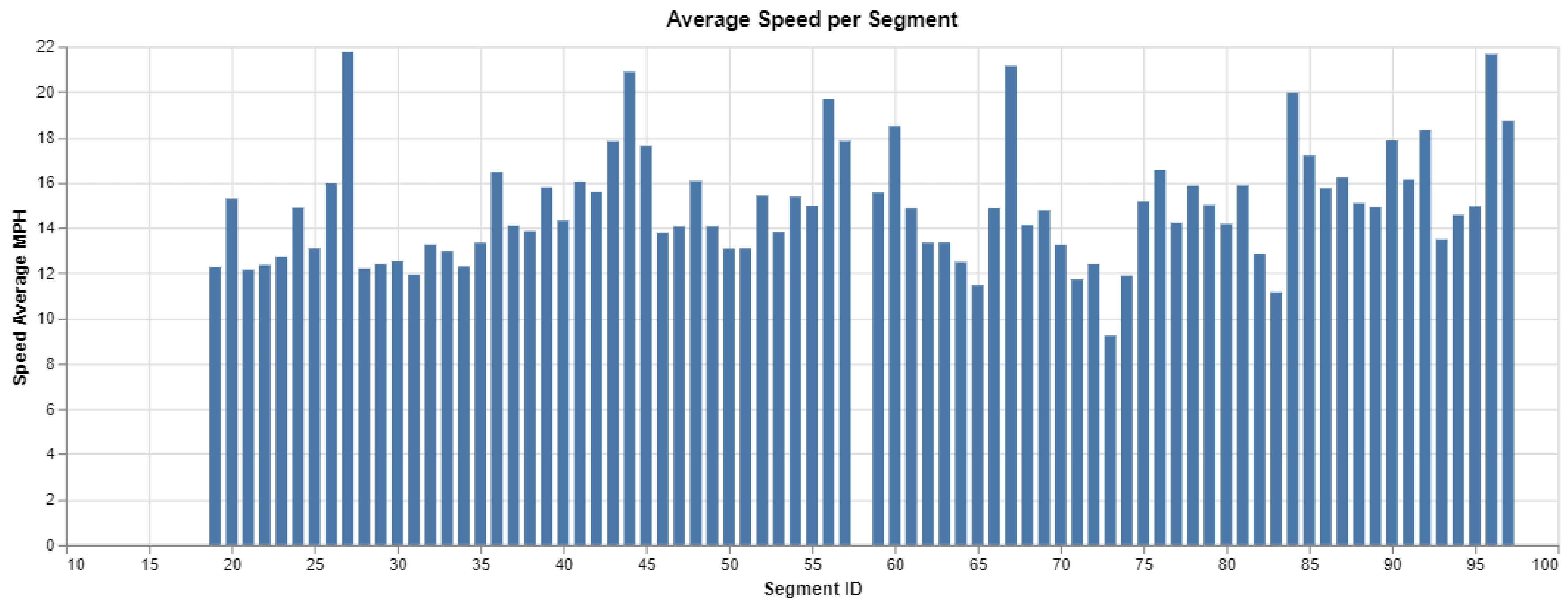
If you got things right, the **series** should look something like this (assets/average_speed.png). You might want to write a test to make sure you are returning the expected type. For example:

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

```
In [11]: #hidden tests are within this cell
```

In [12]:
```python
# let's generate the visualization

# 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(                # encode SEGMENT_ID a sa quantiative variable on the X axis
            '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 as a quantitative variable on Y
            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)
```

Out[12]:



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

In [13]:
```
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
    """
    return pd.pivot_table(df, values = 'SPEED', index = 'MONTH',
                          columns = 'SEGMENT_ID', aggfunc = np.mean)
    #raise NotImplementedError()
```

In [14]:
```
# run the code and see what's in the table
pivot_table = create_pivot_table(segment_rows)
pivot_table
```

Out[14]:

SEGMENT_ID	19	20	21	22	23	24	25	26	27	28	...	88	89	90	91	92
MONTH																
2	6.857143	16.142857	13.571429	19.571429	18.285714	15.857143	11.285714	10.142857	25.000000	20.571429	...	17.000000	14.714286	19.000000	17.857143	20.857143
3	10.773810	14.863095	11.696429	11.815476	13.583333	16.244048	12.398810	15.529762	21.779762	12.422619	...	15.130952	16.470238	17.744048	16.095238	18.095238
4	11.744048	14.958333	11.791667	12.071429	13.208333	16.779762	14.136905	18.339286	22.232143	11.589286	...	14.958333	14.642857	17.702381	15.386905	18.488095
5	11.357143	14.738095	11.369048	11.916667	12.023810	13.220238	11.505952	15.095238	22.857143	11.892857	...	14.154762	12.553571	16.184524	15.130952	17.952381
6	11.630952	14.583333	13.011905	12.279762	12.428571	14.678571	12.690476	15.244048	22.309524	12.619048	...	16.089286	14.869048	17.511905	15.220238	19.035714
7	11.755952	13.595238	10.880952	12.238095	12.267857	14.321429	13.232143	14.964286	22.232143	11.958333	...	17.220238	15.511905	19.476190	15.630952	18.666667
8	12.988095	15.446429	12.303571	13.315476	13.023810	15.827381	12.988095	16.946429	22.244048	12.535714	...	14.863095	13.880952	18.220238	15.196429	17.994048
9	13.970238	17.059524	14.398810	13.452381	12.017857	14.869048	12.571429	15.630952	21.571429	12.464286	...	16.178571	14.916667	17.922619	14.101190	16.833333
10	13.708333	15.666667	12.434524	13.041667	12.422619	13.714286	13.613095	15.922619	20.333333	13.119048	...	14.291667	15.351190	18.059524	19.273810	18.119048
11	12.970238	16.107143	11.922619	11.476190	12.125000	15.607143	14.327381	16.815476	20.553571	12.910714	...	13.422619	15.821429	18.250000	16.357143	17.922619
12	11.845238	15.690476	11.541667	11.559524	13.833333	13.523810	13.375000	15.404762	21.446429	10.113095	...	14.380952	15.101190	17.404762	18.755952	19.898810

11 rows × 78 columns

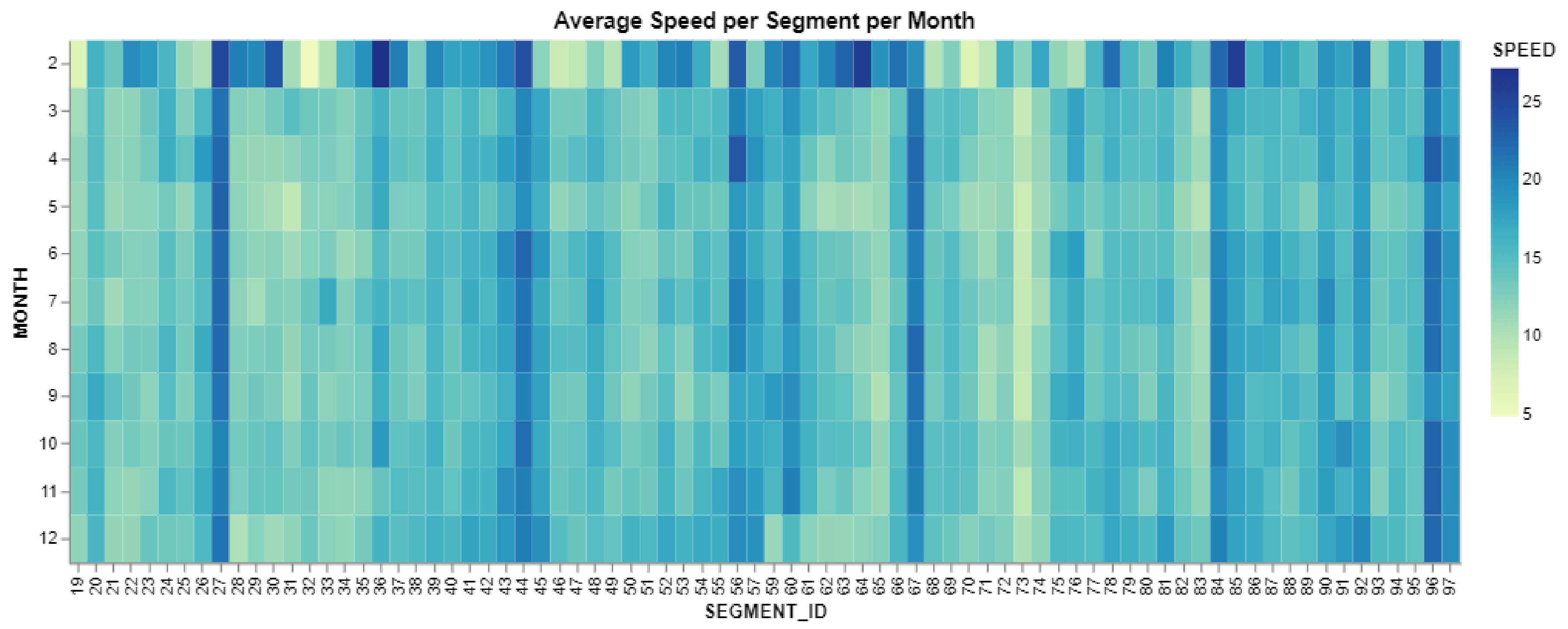
The table should look something like this (assets/pivot_table.png)

As before, we can write a "test" based on this example. For example, here 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 [15]: # 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)
```

Out[15]:



In [16]:
```python
# 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"
```

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_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`)
- Grab a specific column from the dataframe (`select_column`)

In [17]:
```python
def sort_by_col(df, sorting_columns):
    """Sort the rows of df by the columns (sorting_columns)
    return the sorted dataframe
    """
    return df.sort_values(by=sorting_columns)
    #raise NotImplementedError()
```

In [18]:
```python
segment_rows = sort_by_col(segment_rows, ['SEGMENT_ID'])
```

In [19]:
```python
#hidden tests are within this cell
```

In [20]:
```python
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).The column should be a datetime type.
    """
    df['TIME_HOURS'] = pd.to_datetime(df['TIME']).dt.round('H')
    return df
    #raise NotImplementedError()
```

In [21]:
```python
segment_rows = time_to_hours(segment_rows)
```

In [22]:
```python
#hidden tests are within this cell
```

In [23]:
```python
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
    """
    return df[df['DIRECTION'] == traffic_orientation]
    #raise NotImplementedError()
```

In [24]:
```python
sb = filter_orientation(segment_rows, 'SB')
nb = filter_orientation(segment_rows, 'NB')
```

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

In [25]:
```python
#hidden tests are within this cell
```

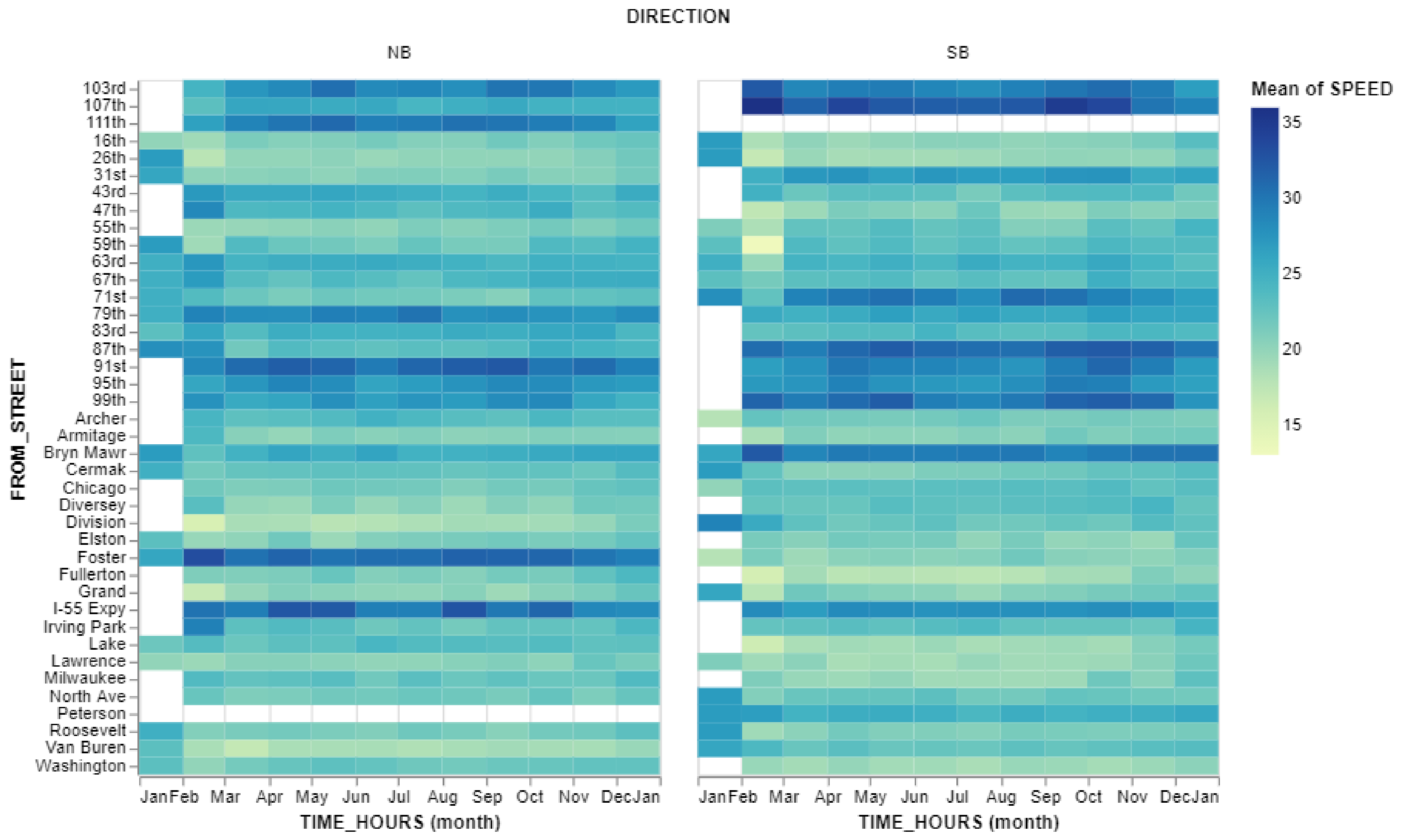
In [26]:
```python
def select_column(df, column_name):
    """ Select a column from the df
    return a series with the desired column
    """
    return df[column_name]
    #raise NotImplementedError()
```

In [27]:
```
#hidden tests are within this cell
```

In [28]:
```
# we're going to remove speeds of -1 (no data)
sb = sb[sb.SPEED > -1]
nb = nb[nb.SPEED > -1]
```

In [29]:
```
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
)
```

Out[29]:



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: What is the relationship of average speed on various streets and at various times of the year between Northbound and Southbound directions?

Abstract Task: What is the correlation between two quantitative variables?

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 [30]:
```
crashes = read_csv('assets/Traffic.Crashes.csv.gz')
crashes_pulaski = crashes[crashes.STREET_NAME == 'PULASKI RD']
```

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

In [31]:
```
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, where 1 <= n <= 300
    300 is the label for records with n at 301 <= n <= 600, and so on.
    """
    bin_values = np.arange(0, df['STREET_NO'].max() + 300, 300)
    df['BIN'] = pd.cut(crashes_pulaski['STREET_NO'], bin_values, labels = bin_values[0:-1])
    return df
    #raise NotImplementedError()
```

```
In [32]: binned_df = bin_crashes(crashes_pulaski)

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

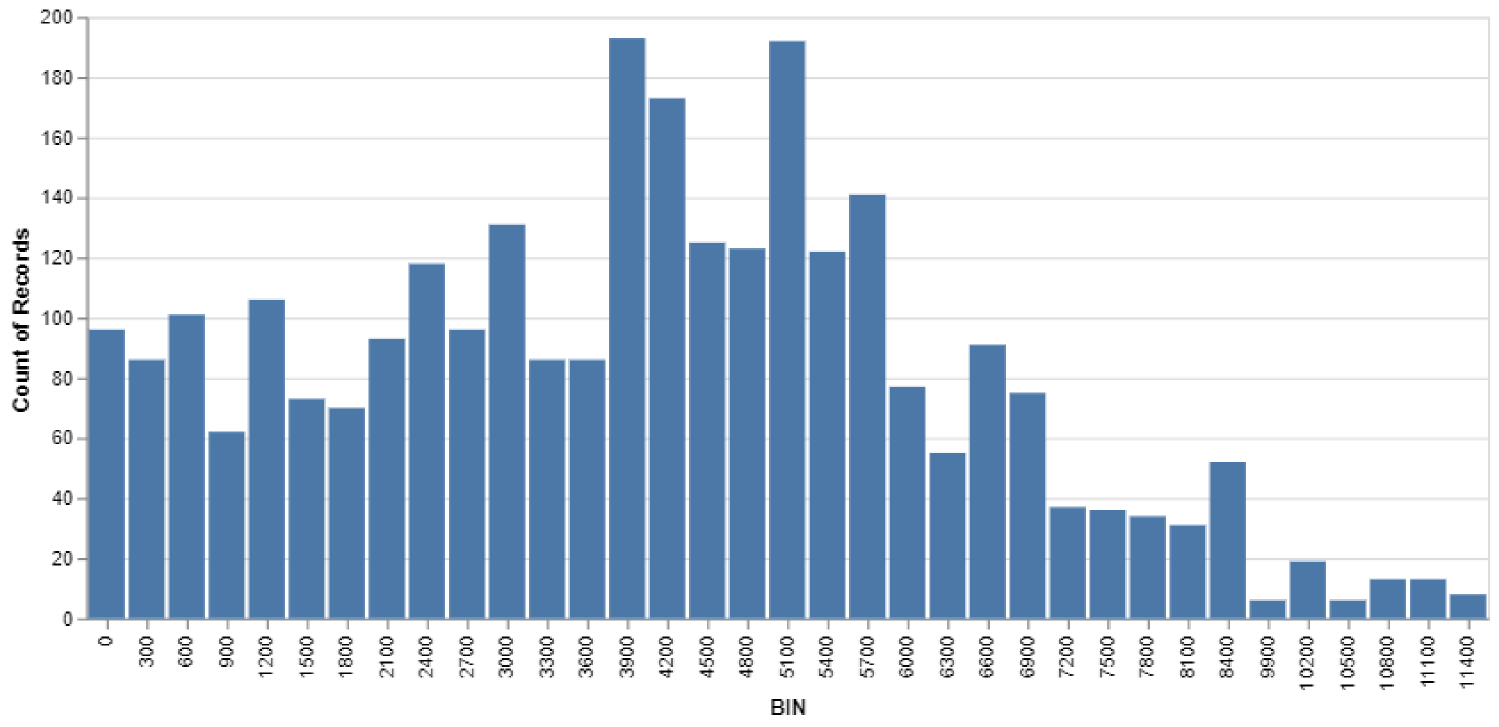
Out[32]:

	STREET_NO	BIN
22205	1627	1500
34402	5500	5400
102336	4600	4500
85688	2220	2100
93846	200	0

A sample of the relevant columns from the table would look something like this (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).

In [33]:
```python
# create this vis
alt.Chart(binned_df).mark_bar().encode(
    alt.X('BIN'),
    alt.Y('count()')
)
```

Out[33]:



In [34]:
```
#hidden tests are within this cell
```

In [35]:
```
def calculate_group_aggregates(df):
    """
    There are *accidents* and *injuries* (could be 0 people got hurt, could be more).
    There’s one row per accident at the moment, so we want to know how many accidents
    happened in each BIN/STREET_DIRECTION (this will be the count) and how many injuries (which will be the sum).

    Return a df with the count of accidents in a column named 'ACCIDENT_COUNT' (how many accidents happened in each
    bin (the count) and how many injuries (the sum) in a column named 'INJURIES_SUM'

    Replace NaN with 0
    """
    aggregate = df.groupby(['BIN', 'STREET_DIRECTION'])['INJURIES_TOTAL'].agg(['count','sum']).fillna(0).reset_index()
    aggregate = aggregate.rename(columns = {'count': 'ACCIDENT_COUNT', 'sum': 'INJURIES_SUM'})
    return aggregate
    #raise NotImplementedError()
```

In [36]:
```
aggregates = calculate_group_aggregates(binned_df)

# check the data
#aggregates.head(15)

aggregates.sample(15)
```

Out[36]:

	BIN	STREET_DIRECTION	ACCIDENT_COUNT	INJURIES_SUM
62	9300	N	0.0	0.0
18	2700	N	69.0	8.0
30	4500	N	41.0	7.0
33	4800	S	79.0	8.0
77	11400	S	8.0	2.0
49	7200	S	36.0	5.0
3	300	S	49.0	17.0
66	9900	N	0.0	0.0
45	6600	S	91.0	11.0
46	6900	N	0.0	0.0
7	900	S	17.0	7.0
76	11400	N	0.0	0.0
23	3300	S	29.0	4.0
72	10800	N	0.0	0.0
56	8400	N	0.0	0.0

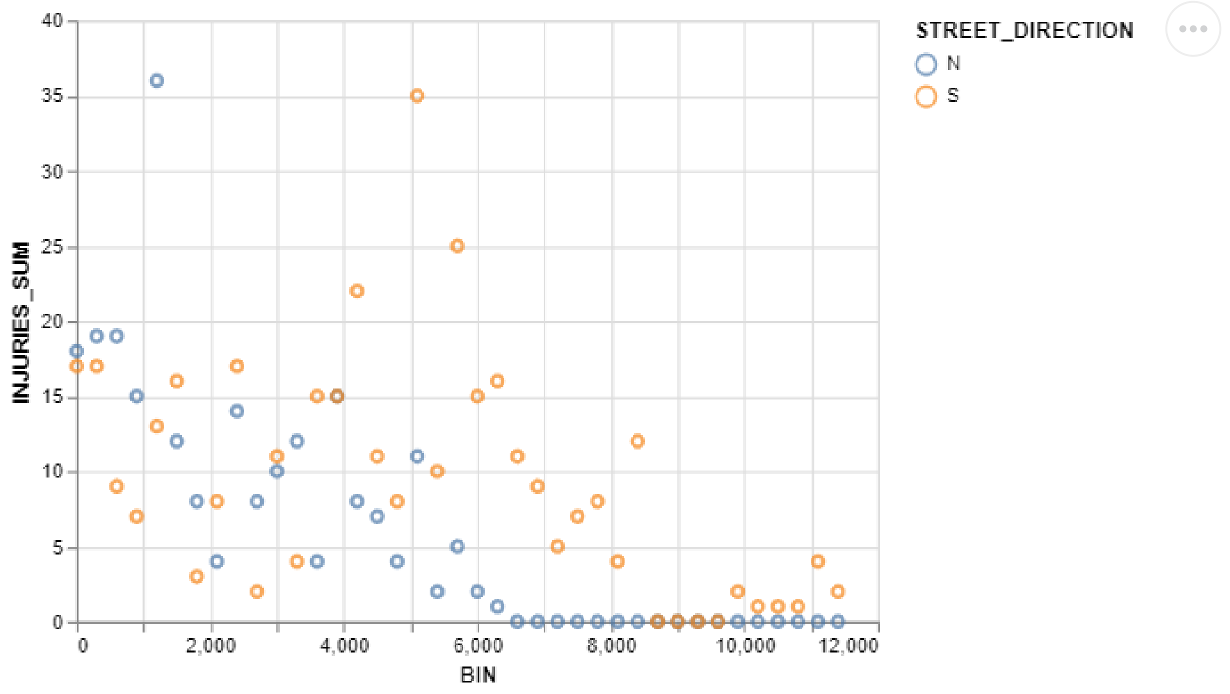
The table should look like [this (assets/2.6_aggregate_1.png)](assets/2.6_aggregate_1.png)

In [37]:
```
#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 (assets/direction_injuries.png)](assets/direction_injuries.png)

```
In [38]: alt.Chart(aggregates).mark_point().encode(
             alt.Color('STREET_DIRECTION'),
             alt.X('BIN'),
             alt.Y('INJURIES_SUM')
         )
```

Out[38]:



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.

In [39]:
```
crashed_range = list(range(0, crashes_pulaski.STREET_NO.max()+1000, 300))
sort_order = ['N ' + str(s) for s in crashed_range[::-1]] + ['S ' + str(s) for s in crashed_range]
def categorical_sorting(df, sorder):
    """ 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 (sorder: the elements of this column should be in
    the same order of the array)
    Sort the dataframe (df) by this column
    """
    df['ORDER_LABEL'] = df['STREET_DIRECTION'] + ' ' + df['BIN'].astype(str)

    df['ORDER_LABEL'] = pd.Categorical(df['ORDER_LABEL'], categories = sorder)
    return df.sort_values(by = 'ORDER_LABEL')
    #raise NotImplementedError()
```

```
In [40]: sorted_groups = categorical_sorting(aggregates, sort_order)

         # check the values
         sorted_groups.sample(15)
```

Out[40]:

	BIN	STREET_DIRECTION	ACCIDENT_COUNT	INJURIES_SUM	ORDER_LABEL
65	9600	S	0.0	0.0	S 9600
63	9300	S	0.0	0.0	S 9300
66	9900	N	0.0	0.0	N 9900
35	5100	S	169.0	35.0	S 5100
49	7200	S	36.0	5.0	S 7200
61	9000	S	0.0	0.0	S 9000
28	4200	N	57.0	8.0	N 4200
41	6000	S	68.0	15.0	S 6000
8	1200	N	76.0	36.0	N 1200
7	900	S	17.0	7.0	S 900
36	5400	N	9.0	2.0	N 5400
20	3000	N	87.0	10.0	N 3000
9	1200	S	30.0	13.0	S 1200
25	3600	S	51.0	15.0	S 3600
47	6900	S	74.0	9.0	S 6900

The table should look like this (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]
```

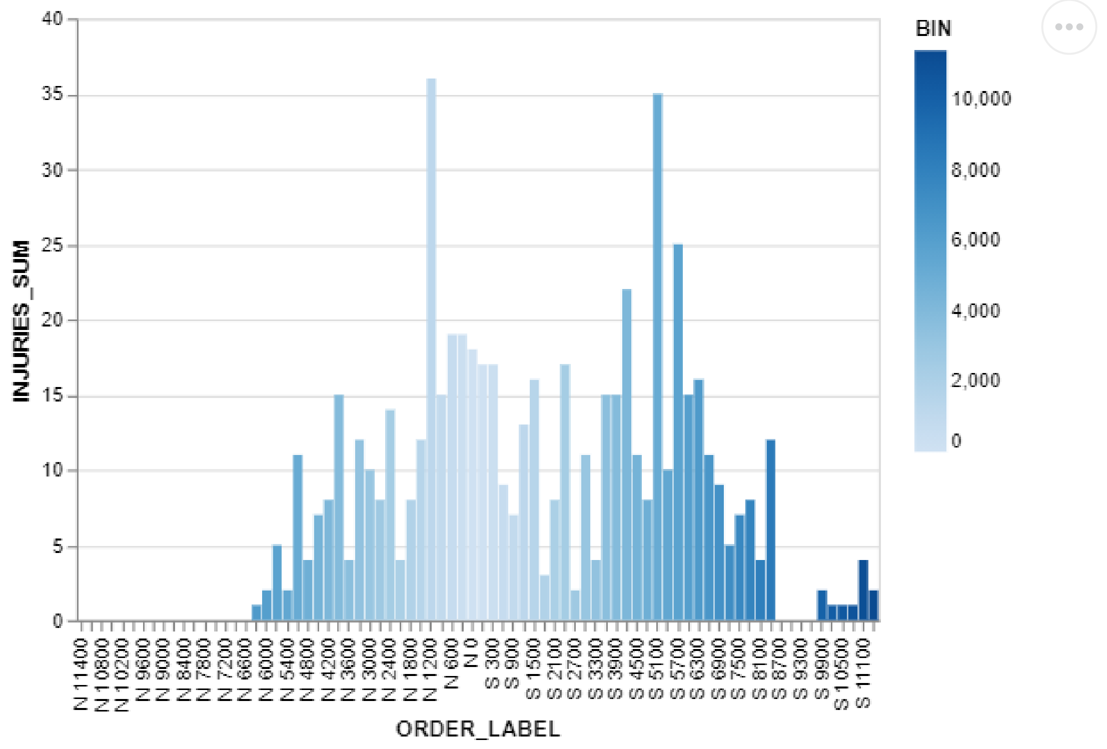
```
In [41]: #hidden tests are within this cell
```

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 (assets/order_injuries.png)](assets/order_injuries.png)

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

```
In [42]: alt.Chart(sorted_groups).mark_bar().encode(
             alt.X('ORDER_LABEL:O', sort=sort_order),
             alt.Y('INJURIES_SUM:Q'),
             alt.Color('BIN:Q')
         ).properties(
             width=400
         )
```

Out[42]:

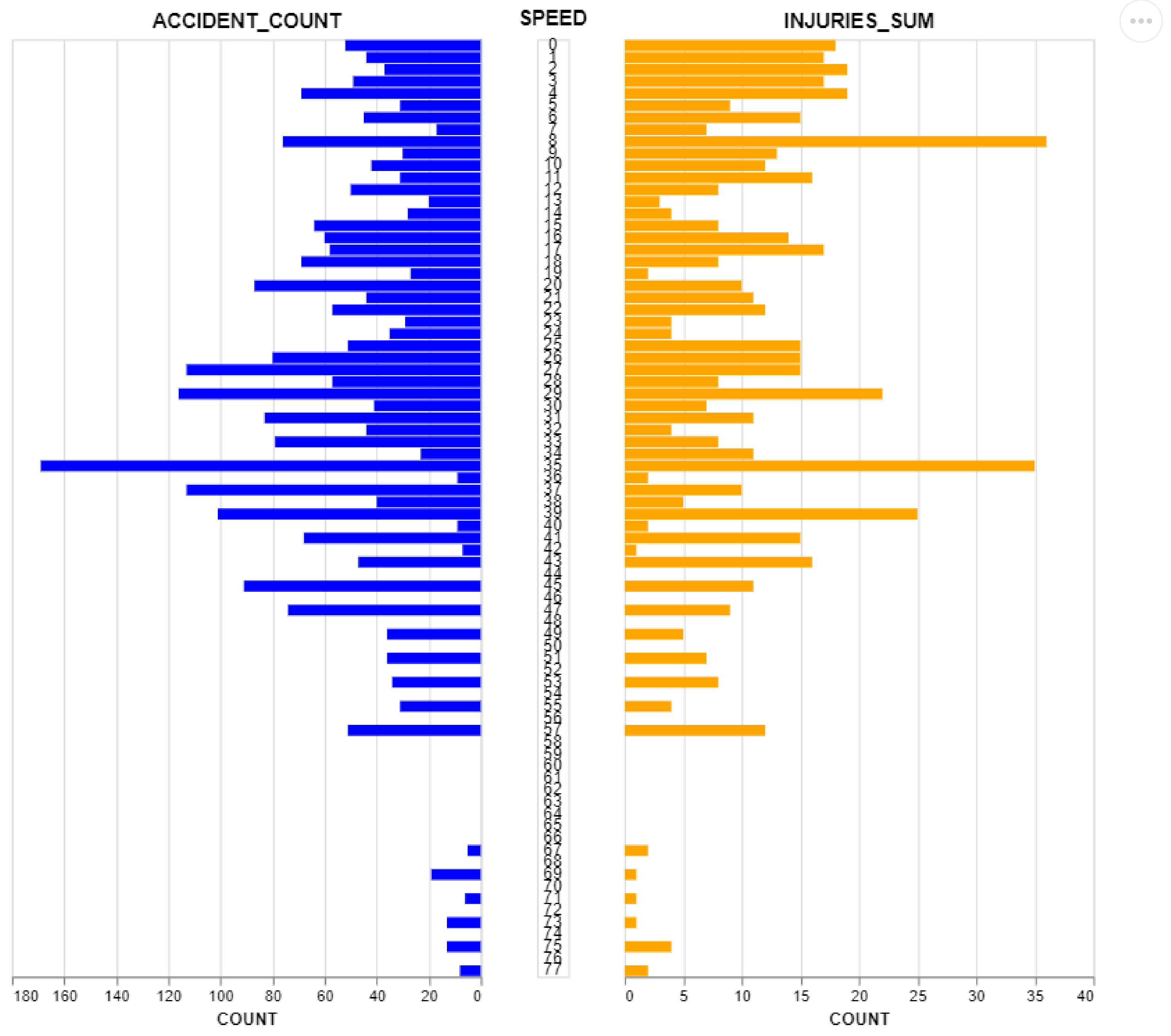


Ok, let's actually make a useful visualization using some of the dataframes we've created. As a bonus, we're going to ask you what you would use this for.

[illegible]

In [43]:
```python
# 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)

# Note that we cheated a bit. The actual speed column (POSTED_SPEED) doesn't have enough variation for this
# example, so we're using the level_1 variable (it's an index variable) as a fake SPEED.
# 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 == '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
```

Out[43]:



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."

Domain Task: What is the relationship of total accidents and total injuries relative to average speed?

Abstract Task: What is the correlation between quantitative variables?