Altair Exercises

This notebook will explore multiple different visualizations in Altair.

Part 1

The following exercise is based on the 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)](https://www.cmap.illinois.gov/updates/all/-/asset_publisher/UIMfSLnFfMB6/content/crash-scans-show-relationship-between-congestion-and-crash-rates).

In [1]:

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

In [2]:

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

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

Out[2]:

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

In [3]:

```
# Save the congestion dataframe on hist_con
hist_con = pd.read_csv('../assets/Pulaski.small.csv.gz', compression='
```

In [4]:

```
def get_group_first_row(df, grouping_columns):
    """Group rows using the grouping columns and return the first row
    """
    grouped_df = df.groupby(grouping_columns, as_index=False).first()

    return grouped_df
```

In [5]:

```
# test your code, we want segment_rows to be resampled version of hist
# properties month, day_of_week, hour, and segment_id and returned the
segment_rows = get_group_first_row(hist_con, ['MONTH','DAY_OF_WEEK', '
segment_rows.head(5)
```

Out[5]:

	MONTH	DAY_OF_WEEK	HOUR	SEGMENT_ID	TIME	SPEED	STREET	DIRECTION	F
0	2	4	17	19	02/28/2018 05:40:00 PM	-1	Pulaski	NB	
1	2	4	17	20	02/28/2018 05:40:00 PM	15	Pulaski	NB	
2	2	4	17	21	02/28/2018 05:40:00 PM	29	Pulaski	NB	
3	2	4	17	22	02/28/2018 05:40:00 PM	28	Pulaski	NB	
4	2	4	17	23	02/28/2018 05:40:00 PM	23	Pulaski	NB	

Basic Bar Chart Visualization

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.

In [6]:

```
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 i
    """
    df0 = df.groupby(['SEGMENT_ID']).mean()
    s = df0['SPEED']

    return s

average_speed_per_segment(segment_rows)
```

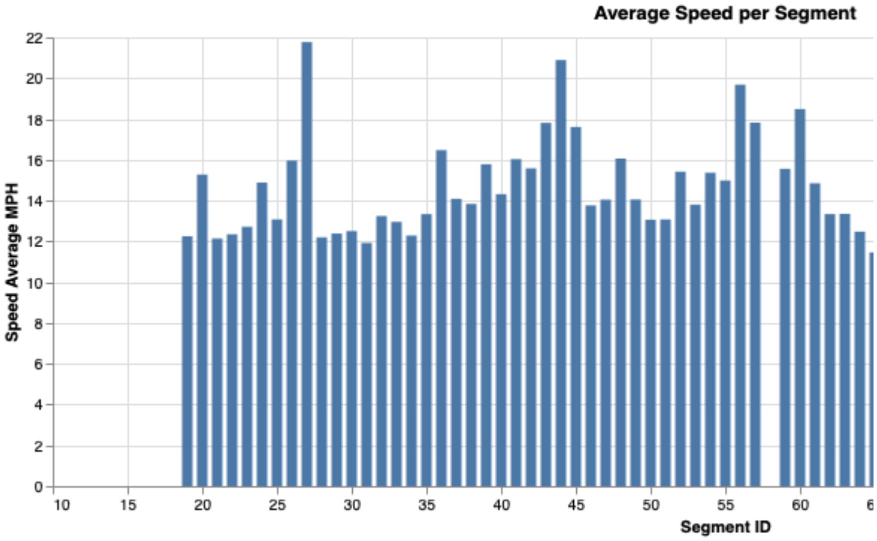
Out[6]:

```
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
```

In [7]:

```
# 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
encoding.mark_bar(size=7).properties(title='Average Speed per Segment'
```

Out[7]:



Create a Basic Pivot Table

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.

In [8]:

```
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 segmen
    """
    df0 = pd.pivot_table(df,
                         values='SPEED',
                         index=['MONTH'],
                         columns=['SEGMENT_ID'],
                         aggfunc='mean'
                        )

    return df0
```

In [9]:

```
pivot_table = create_pivot_table(segment_rows)
pivot_table.head()
```

Out[9]:

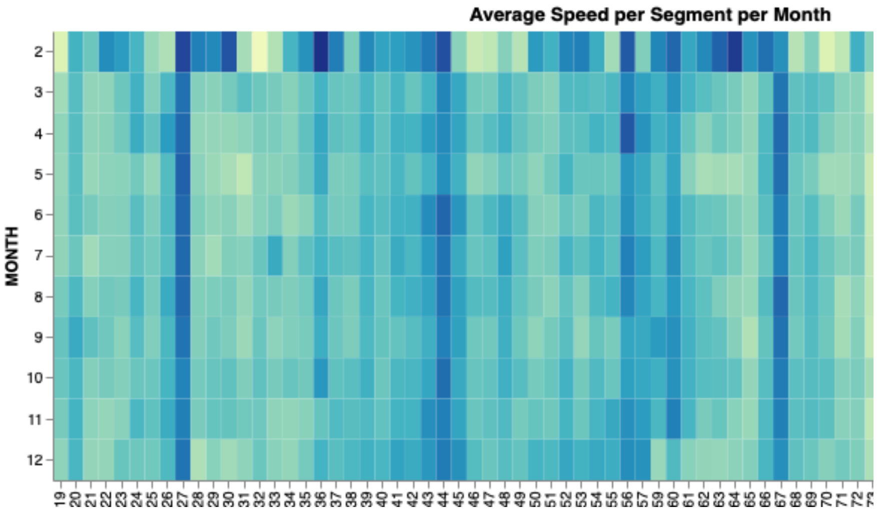
SEGMENT_ID	19	20	21	22	23	24	25	
MONTH								
2	6.857143	16.142857	13.571429	19.571429	18.285714	15.857143	11.285714	10
3	10.773810	14.863095	11.696429	11.815476	13.583333	16.244048	12.398810	15
4	11.744048	14.958333	11.791667	12.071429	13.208333	16.779762	14.136905	18
5	11.357143	14.738095	11.369048	11.916667	12.023810	13.220238	11.505952	15
6	11.630952	14.583333	13.011905	12.279762	12.428571	14.678571	12.690476	15

5 rows × 78 columns

In [10]:

```
# we're going to implement a transformation to put the pivot table
# into a 'long form' because it is easier to specify the visualization

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
# figure out the horizontal placement (x), the month as the vertical (

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
```

Out[10]:



SEGMENT_ID

Sorting, Transforming, and Filtering

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`)

In [11]:
```
def sort(df, sorting_columns):
    """Sort the rows by the columns
    return the sorted dataframe
    """
    df0 = df.copy()
    df0 = df0.sort_values(by=sorting_columns, ascending=True)

    return df0
```

In [12]:
```
segment_rows = sort(segment_rows, ['SEGMENT_ID'])
```

In [13]:
```
def time_to_hours(df):
    """ Add a column (called TIME_HOURS) based on the data in the TIME
    the value to the nearest hour.  For example, if the original TIME
    ‘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_H
    actually a proper datetime and not a string).
    """
    df0 = df.copy()
    df0['TIME'] = pd.to_datetime(df0['TIME'])
    df0['TIME_HOURS'] = df0['TIME'].dt.round(freq='H')

    return df0
```

In [14]:
```
segment_rows = time_to_hours(segment_rows)
segment_rows.head()
```

Out[14]:

	MONTH	DAY_OF_WEEK	HOUR	SEGMENT_ID	TIME	SPEED	STREET	DIRECTION
0	2	4	17	19	2018- 02-28	-1	Pulaski	NE

					17:40:00			
22542	4	5	18	19	2018-04-26 18:50:23	-1	Pulaski	NE
108420	11	2	15	19	2018-11-26 15:50:26	16	Pulaski	NE
22620	4	5	19	19	2018-04-26 19:50:21	-1	Pulaski	NE
108342	11	2	14	19	2018-11-26 14:40:10	26	Pulaski	NE

In [15]:

```
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 orie
    """
    df0 = df.copy()
    df0 = df0[df0['DIRECTION'] == traffic_orientation]

    return df0
```

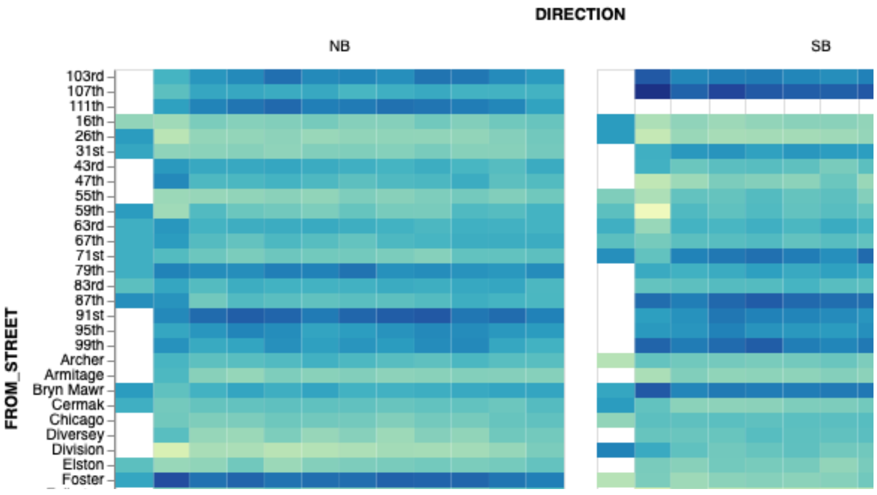
In [16]:

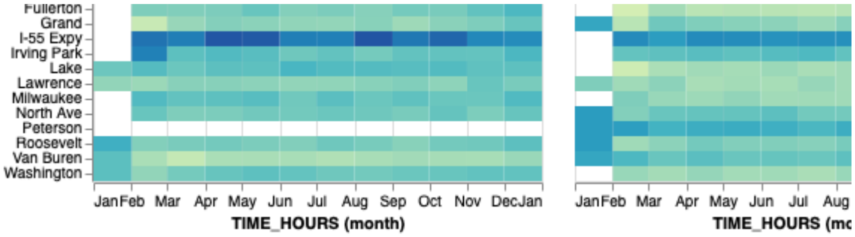
```
sb = filter_orientation(segment_rows, 'SB')
nb = filter_orientation(segment_rows, 'NB')

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

In [17]:

```
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[17]:





Traffic Heatmap Visualization

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. We will work with accidents recorded on Pulaski Road.

In [18]:

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

In [19]:

```
def bin_crashes(df):
    """ Assign each crash instance a category (bin) every 300 house nu
    Return a new dataframe with a column called BIN where each value i
    i.e. 0 is the label for records with street number n, such that 1
    300 is the label for records with with n at 301 <= n <= 600, and s
    """
    df0 = df.copy()

    bin_values = list(range(0,max(list(df0['STREET_NO']))+300,300))
    labels = list(range(0,max(list(df0['STREET_NO'])),300))

    df0['BIN'] = pd.cut(df0['STREET_NO'], bin_values, labels=labels)
    #s = df0['BIN'].value_counts()

    return df0
```

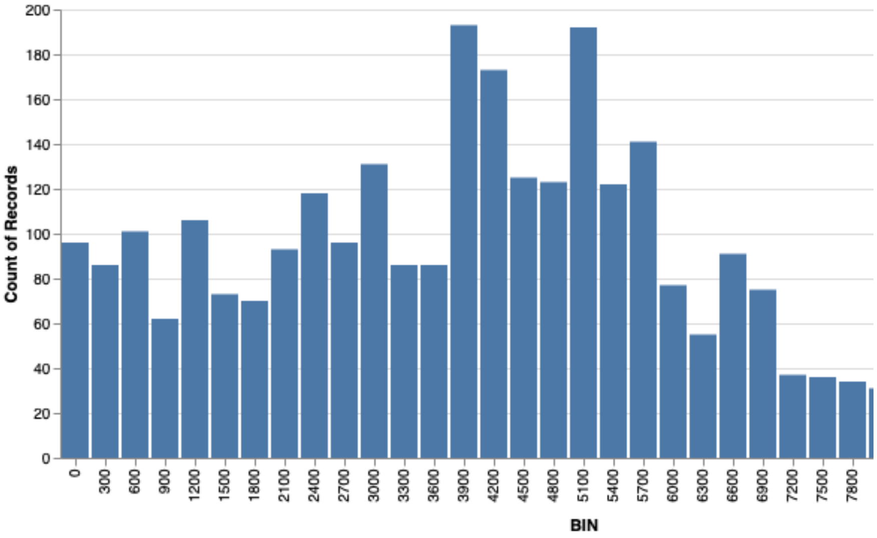
In [20]:

```
binned_df = bin_crashes(crashes_pulaski)
```

In [21]:

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

Out[21]:



In [22]:
```
def calculate_group_aggregates(df):
    """
    Return a df with the count of accidents in a 'ACCIDENT_COUNT' colu
    """
    df0 = (df.groupby(['BIN','STREET_DIRECTION'])
           .agg({'CRASH_DATE' :'count','INJURIES_TOTAL':'sum'})
           .reset_index()
           .rename(columns={'CRASH_DATE':'ACCIDENT_COUNT',
                            'INJURIES_TOTAL':'INJURIES_SUM'})
         )

    return df0
```

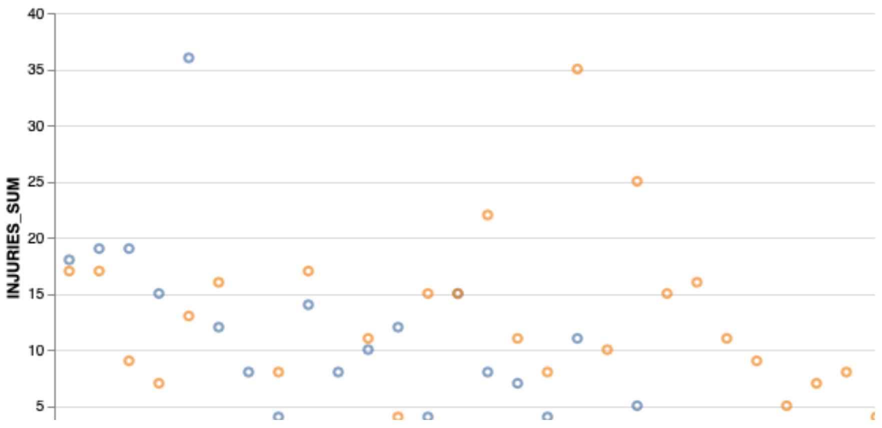
In [23]:
```
aggregates = calculate_group_aggregates(binned_df)
aggregates.head(14)
```

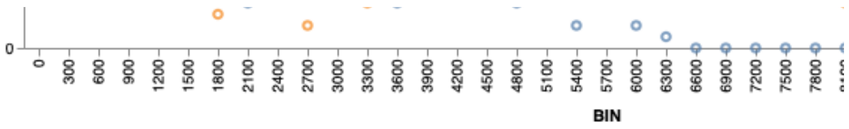
Out[23]:

	BIN	STREET_DIRECTION	ACCIDENT_COUNT	INJURIES_SUM
0	0	N	52	18.0
1	0	S	44	17.0
2	300	N	37	19.0
3	300	S	49	17.0
4	600	N	70	19.0
5	600	S	31	9.0
6	900	N	45	15.0
7	900	S	17	7.0
8	1200	N	76	36.0
9	1200	S	30	13.0
10	1500	N	42	12.0
11	1500	S	31	16.0
12	1800	N	50	8.0
13	1800	S	20	3.0

In [24]:
```
alt.Chart(aggregates).mark_point().encode(
    alt.Color('STREET_DIRECTION'),
    alt.X('BIN'),
    alt.Y('INJURIES_SUM')
)
```

Out[24]:





Sort the Street Ranges

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

In [25]:

```
crashed_range = list(range(0, crashes_pulaski.STREET_NO.max()+1000, 30
sort = ['N ' + str(s) for s in crashed_range[::-1]] + ['S ' + str(s) f

def categorical_sorting(df, sort):
    """ Create a column called ORDER_LABEL that contains a concatenati
    Set the sort order of this column to the provided sort array (the
    of the array)
    Sort the dataframe by this column
    """
    df0 = df.copy()
    df0['ORDER_LABEL'] = df0['STREET_DIRECTION'].astype(str) + ' ' + c

    df0.ORDER_LABEL = pd.Categorical(df0.ORDER_LABEL,
                                     categories=sort,
                                     ordered=True
                                    )

    return df0
```

In [26]:

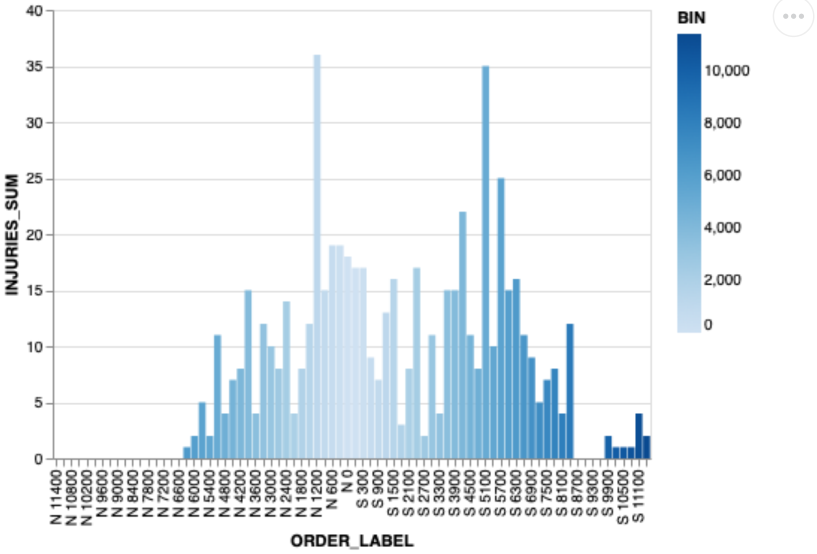
```
sorted_groups = categorical_sorting(aggregates, sort)
```

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 debugging.

In [27]:

```
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
)
```

Out[27]:

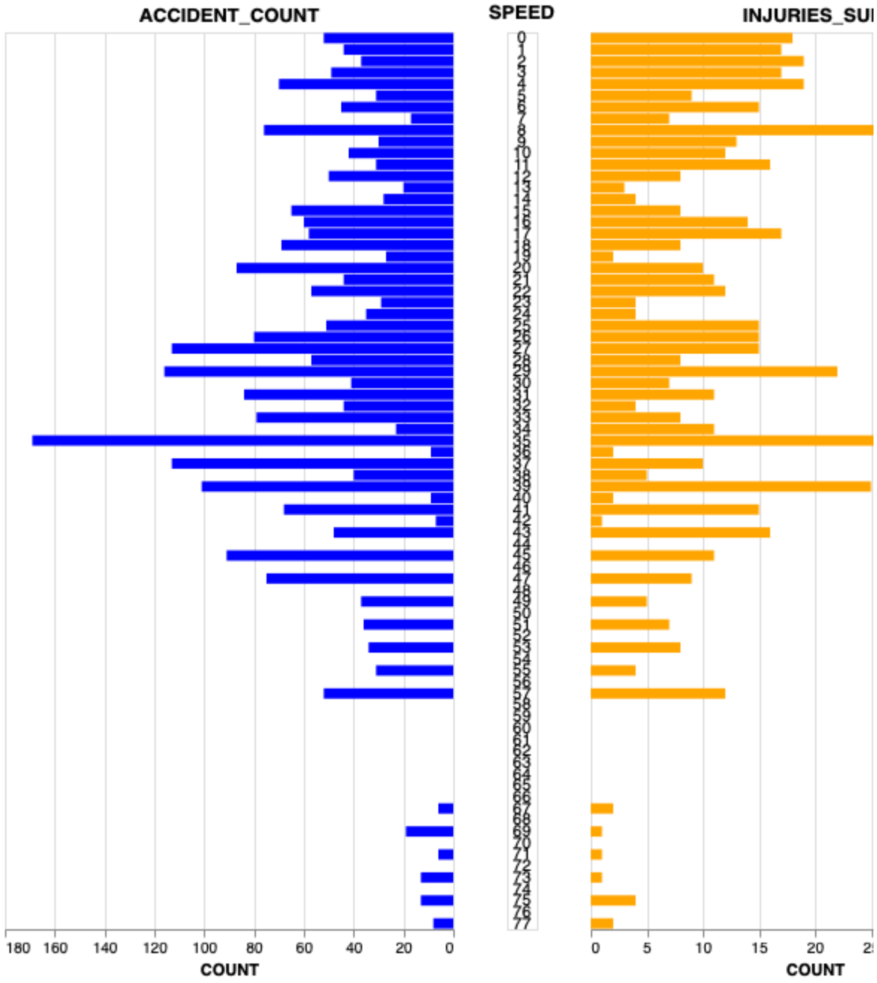


Ok, let's actually make a useful visualization using some of the dataframes we've created.

In [28]:

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

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

a = alt.Chart(bar_sorted_groups).mark_bar().transform_filter(alt.datum
    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', 'INJURIE
                                    range=['blue', 'orange']))
).properties(
    title='ACCIDENT_COUNT',
    width=300,
    height=600
)

# middle "chart" which actually won't be a chart, just a bunch of labe
b = alt.Chart(bar_sorted_groups).mark_bar().transform_filter(alt.datum
    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
    x='COUNT:Q',
    y=alt.Y('SPEED:O',axis=None),
    color=alt.Color('TYPE:N',
                    legend=None,
                    scale=alt.Scale(domain=['ACCIDENT_COUNT', 'INJURIE
                    scale=alt.Scale(domain=['ACCIDENT_COUNT', 'INJURIE
                                    range=['blue', 'orange']))
).properties(
    title='INJURIES_SUM',
    width=300,
    height=600
)

# put them all together

a | b | c
```

Out[28]:



Exercise adapted and modified from UMSI homework assignment for SIADS 522.