Library Imports

In [7]:
```
import numpy as np
import pandas as pd
import geopandas as gpd
import zipfile
import requests
import os
from pyspark.sql import SparkSession
from pyspark import SparkContext as sc
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib as mpl
from geomanip.poly_contains import combine_geom_dfs
from geomanip.endpoint_algorithm import GeneratePoints, find_avg_distance, find_dist_ratio, find_traffic_ratio
import altair as alt

%matplotlib inline
```

Spark session creation and EV registration data extraction

We loaded the registration data into csv format so that we can manipulate it more quickly. The file size is very large in this case.

In [8]:
```
spark = SparkSession \
    .builder \
    .master("local[*]") \
    .appName('NY Registration EDA') \
    .getOrCreate()

# Specify filepath
fp = 'assets/ny_reg.csv'
ny_reg = spark.read.csv(fp, header=True)
```

We created a view of the registration dataframe to use SQL queries. The first row is also shown.

In [9]:
```
ny_reg.createOrReplaceTempView('ny_reg')
ny_reg.first()
```

```
Row(Record Type='BOAT', VIN='999999999999', Registration Class='BOT', City='RYE', State='NY', Zip='10580', Coun
```

Out[9]:
```
ty='WESTCHESTER', Model Year='1940', Make='FAIRF', Body Type='BOAT', Fuel Type='GAS', Unladen Weight=None, Maxi
mum Gross Weight=None, Passengers=None, Reg Valid Date='03/10/2020', Reg Expiration Date='04/30/2023', Color=No
ne, Scofflaw Indicator='N', Suspension Indicator='N', Revocation Indicator='N')
```

Here, we found the number of passenger electric vehicles registered by county using the Spark Dataframe. Only counties in the NYC metro were chosen.

In [10]:
```
query = """
SELECT Zip, COUNT(*) num_veh FROM ny_reg
GROUP BY Zip, `Record Type`, `Registration Class`, `Fuel Type`, `County`, State
HAVING `Record Type` = 'VEH'
    AND `Registration Class` = 'PAS'
    AND `Fuel Type` = 'ELECTRIC'
    AND County IN ('QUEENS', 'KINGS', 'NEW YORK', 'BRONX', 'RICHMOND',
    'WESTCHESTER', 'ROCKLAND', 'NASSAU', 'SUFFOLK')
    AND State = 'NY'
ORDER BY num_veh DESC
"""
zip_veh = pd.DataFrame([x.asDict() for x in spark.sql(query).collect()])
```

Zipcode shapefile creation and zip extraction

We downloaded the US zipcode Shapefile and extracted, then read the file. This was used to match short count locations.

In [11]:
```
if not os.path.exists('assets/us_zipcodes'):
    zipcode_url = 'https://www2.census.gov/geo/tiger/GENZ2018/shp/cb_2018_us_zcta510_500k.zip'
    r_zipcode = requests.get(zipcode_url)
    with open('assets/us_zipcodes.zip', 'wb') as us_zipcodes:
        us_zipcodes.write(r_zipcode.content)
    with zipfile.ZipFile('assets/us_zipcodes.zip', 'r') as us_zipcodes:
        us_zipcodes.extractall('assets/us_zipcodes')
us_zip_shp = gpd.read_file('assets/us_zipcodes/cb_2018_us_zcta510_500k.shp', crs='ESPG:4326')
```

We queried the NY Reg dataframe to find all zip codes in NY State. Used to narrow zip code data.

In [12]:
```
query = """
SELECT DISTINCT(Zip) FROM ny_reg
WHERE State = 'NY'
"""
results = [x.asDict()['Zip'] for x in spark.sql(query).collect()]
```

We selected only zip codes in NY state and with centroids below 41.5°N. We deemed all zip codes below this latitude to be in NYC metro.

In [13]:
```
nyc_zips = us_zip_shp.loc[(us_zip_shp.geometry.centroid.y < 41.5) & (us_zip_shp['ZCTA5CE10'].isin(results))]
```

```
<ipython-input-13-83aa2a482600>:1: UserWarning: Geometry is in a geographic CRS. Results from 'centroid' are li
kely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this operation.

  nyc_zips = us_zip_shp.loc[(us_zip_shp.geometry.centroid.y < 41.5) & (us_zip_shp['ZCTA5CE10'].isin(results))]
```

We extracted the counts with LineString geometries to match road segemnts.

In [14]:
```
if not os.path.exists('assets/ny_aadt.zip'):
    r = requests.get('http://gis.ny.gov/gisdata/fileserver/?DSID=1282&file=NYSDOT_TDV_AADT.zip')
    with open('assets/ny_aadt.zip', 'wb') as aadt:
        aadt.write(r.content)
    with zipfile.ZipFile('assets/ny_aadt.zip', 'r') as aadt:
        aadt.extractall('assets/ny_aadt')
```

Traffic data extraction and manipulation

We then extracted the short count geodatabase and read short count data, then set the crs to that for the NYC region.

In [15]:
```
if not os.path.exists('assets/ny_short_counts_2019.gdb'):
    with zipfile.ZipFile('assets/ny_short_counts_2019.zip', 'r') as traffic_count:
        traffic_count.extractall('assets/ny_short_counts_2019.gdb')
short_count_loc = gpd.read_file('assets/ny_short_counts_2019.gdb')
short_count_loc.crs = 'EPSG:32618'
```

We read the traffic database with linestring objects and set the crs, then extracted only non-null geometries.

In [16]:
```
line_counts = gpd.read_file('assets/ny_traffic.gdb')
line_counts.crs = 'EPSG:32618'
line_counts = line_counts.loc[line_counts['geometry'].notnull()]
```

We merged the short counts and linestring dataframes on the station number and selected only those with short count latitude below 41°N.

In [17]:

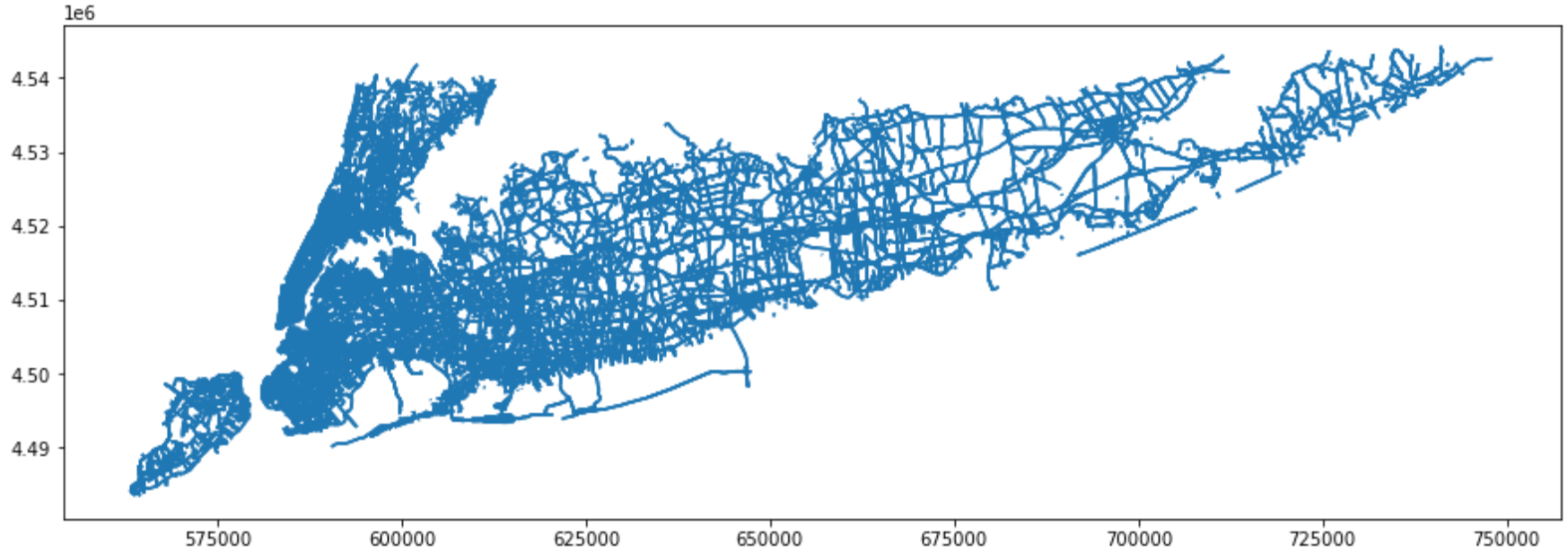
```
traffic_part_line = pd.merge(short_count_loc, line_counts[['RCStation', 'geometry']], left_on='RCSTA', right_on
traffic_part_line.drop_duplicates(inplace=True)
traffic_nyc_line = traffic_part_line.loc[traffic_part_line['LATITUDE'] < 41]
```

This is a visualization of all roads included in this dataframe.

In [18]:

```
full_routes = gpd.GeoDataFrame(traffic_nyc_line, geometry='geometry_y', crs='EPSG:32618').plot(figsize=(15,15))
plt.savefig('full_routes.png')
full_routes
```

Out[18]: <AxesSubplot:>



We selected all unique station counts across the collection years. This was required to have enough stations to create routes.

In [19]:

```
traffic_part = pd.DataFrame()
included_stations = []
for year in np.sort(traffic_part_line['CountYear'].unique())[::-1]:
    df = traffic_part_line.loc[traffic_part_line['CountYear'].astype(int) == year]
    df_unique = df.loc[~df['RCSTA'].isin(included_stations)]
    traffic_part = pd.concat([traffic_part, df_unique], axis=0)
    included_stations.extend(df['RCSTA'].unique().tolist())
```

We selected only segments with North, East, South, or West direction. Directions with the label `0` were bidirectional.

In [20]:
```
traffic_part = traffic_part.loc[traffic_part['FederalDirection'] != 0]
```

Next, We filtered by only commuter traffic using the `FactorGroup` column.

In [21]:
```
traffic_part = traffic_part.loc[(traffic_part['FactorGroup'] == 30)]
```

We reformed the manipulated dataframe into a GeoDataFrame.

In [22]:
```
traffic_part = gpd.GeoDataFrame(traffic_part, geometry='geometry_y')
traffic_part.crs = 'EPSG:32618'
```

We combined traffic from several periods to form morning rush hour periods.

In [35]:
```
traffic_part['morning_rush_hour'] = (traffic_part['AvgWeekdayInterval_7']
                                     .add(traffic_part['AvgWeekdayInterval_8'])
                                     .add(traffic_part['AvgWeekdayInterval_9'])
                                     .add(traffic_part['AvgWeekdayInterval_10'])
                                    )
traffic_part['afternoon_rush_hour'] = (traffic_part['AvgWeekdayInterval_16']
                                     .add(traffic_part['AvgWeekdayInterval_17'])
                                     .add(traffic_part['AvgWeekdayInterval_18'])
                                     .add(traffic_part['AvgWeekdayInterval_19'])
                                    )
```

We selected only short count locations with a lattitude less than 41.5°N. These were deemed to be within the NYC region.

In [36]:
```
traffic_nyc = traffic_part.loc[traffic_part['LATITUDE'] < 41]
```

Combination with zipcode dataframe and creation of seeds for algorithm

Next, we combined the traffic dataframe with the zipcode dataframe to match specific zipcodes with traffic counts.

In [37]:
```
traffic_nyc = gpd.GeoDataFrame(traffic_nyc, geometry='geometry_x')
traffic_nyc.crs = 'EPSG:32618'

traffic_nyc = combine_geom_dfs(nyc_zips, traffic_nyc, 'geometry', 'geometry_x')
```

We found the top route segments by zipcode for morning and afternoon rush hour. Used to match registration data by zipcode to seeds of endpoint algorithm.

In [38]:
```
traffic_nyc['top_morning'] = traffic_nyc[['morning_rush_hour']] == traffic_nyc.groupby('ZCTA5CE10')[['morning_r
traffic_nyc['top_after'] = traffic_nyc[['afternoon_rush_hour']] == traffic_nyc.groupby('ZCTA5CE10')[['afternoon
```

We selected the columns used by the endpoint algorithm as well as the zipcode column to merge with registration data.

In [39]:
```
top_morning = traffic_nyc.loc[traffic_nyc['top_morning']][['RCSTA', 'FederalDirection', 'geometry_y', 'ZCTA5CE1
top_afternoon = traffic_nyc.loc[traffic_nyc['top_after']][['RCSTA', 'FederalDirection', 'geometry_y', 'ZCTA5CE1
```

Next, we merged the traffic data with registration data to estimate traffic.

In [40]:
```
top_morning = pd.merge(top_morning, zip_veh, left_on='ZCTA5CE10', right_on='Zip')
top_afternoon = pd.merge(top_afternoon, zip_veh, left_on='ZCTA5CE10', right_on='Zip')
```

We dropped the zipcode columns to find only the relevant columns. We then renamed the columns to match the required columns for seeds by the algorithm.

In [41]:
```
top_morning.drop(['ZCTA5CE10', 'Zip'], axis=1, inplace=True)
top_afternoon.drop(['ZCTA5CE10', 'Zip'], axis=1, inplace=True)
top_morning.columns = ['station', 'direction', 'geometry', 'traffic']
top_afternoon.columns = ['station', 'direction', 'geometry', 'traffic']
```

We randomly selected three segments from each period to seed the algorithm.

In [42]:
```
morning_seeds = [top_morning.sample(3, random_state=42).iloc[i].to_dict() for i in range(3)]
after_seeds = [top_afternoon.sample(3, random_state=42).iloc[i].to_dict() for i in range(3)]
```

Running algorithm on various seeds

We used the seeds to evaluate different inputs into the endpoint algorithm and their effects. We created a geojson file to store the results and created a GeoDataFrame.

In [43]:
```
# Check if geojson file already exists. If so, read into dataframe and set crs.
if os.path.exists('assets/traffic_results.geojson'):
```

```
    seed_df = gpd.read_file('assets/traffic_results.geojson')
    seed_df.crs = 'EPSG:32618'

# Otherwise, create geojson file and dataframe.
else:

    # Loop through inputs of traffic exponent and straight path multiplier to test
    # different values. Append to inputs.
    full_inputs = []
    for i in range(1, 4):
        for j in range(2, 5):
            traffic_exp = j * 0.5
            inputs = {'straight_mult': i,
                      'traffic_exp': traffic_exp}
            full_inputs.append(inputs)

    # Loop through both periods and seeds as well as inputs.
    results = []
    for period in ['morning', 'afternoon']:
        traffic_col = f'{period}_rush_hour'

        # Choose relevant seeds
        if period == 'morning':
            seeds = morning_seeds.copy()
        else:
            seeds = after_seeds.copy()

        # Initialize class instance as None.
        point_gen = None

        # Loop through seeds to evaluate different inputs.
        for seed in seeds:
            for inputs in full_inputs:

                # If class instance is not created, create it.
                if not point_gen:
                    point_gen = GeneratePoints(traffic_nyc, radius=100, traffic_col=traffic_col,
                                               straight_mult=inputs['straight_mult'],
                                               traffic_exp=inputs['traffic_exp'])
                # Adjust inputs if class instance already created.
                else:
                    point_gen.straight_mult = inputs['straight_mult']
                    point_gen.traffic_exp = inputs['traffic_exp']

                # Build list of dictionaries for steps.
                result = point_gen.build_pred_steps(seed, min_perc=0.005)
```

```
                metrics = {'inputs': inputs,
                           'dist_ratio': find_dist_ratio(result),
                           'traf_ratio': find_traffic_ratio(result, point_gen.traffic_part),
                           'dist_avg': find_avg_distance(result),
                           'period': period,
                           'result': result
                          }

                # Create variable for printing seed, direction, and inputs when loop completed.
                print_met = metrics.copy()
                print_met.pop('result', None)
                print(print_met)
                print('\n')

                # Append results to list of results.
                results.append(metrics)

    # Create list of dictionaries to convert to geojson file.
    full_dcts = []
    for result in results:
        i = 1
        for seed in result['result']:
            j = 1
            for step in seed:
                part_dct = {'period': result['period'],
                            'route': i,
                            'part': j,
                            'straight_mult': result['inputs']['straight_mult'],
                            'traffic_exp': result['inputs']['traffic_exp'],
                            'dist_ratio': result['dist_ratio'],
                            'traf_ratio': result['traf_ratio'],
                            'dist_avg': result['dist_avg'],
                            'seed_station': seed[0]['station'],
                            'seed_direction': seed[0]['direction'],
                            'seed_traffic': seed[0]['traffic'],
                            'station': step['station'],
                            'direction': step['direction'],
                            'traffic': step['traffic'],
                            'traffic_perc': step['traffic_perc'],
                            'geometry': step['geometry']
                           }
                full_dcts.append(part_dct)
                j += 1
            i += 1

    # Convert results to GeoDataFrame, then save as geojson.
```

```
    seed_df = gpd.GeoDataFrame(full_dcts, geometry='geometry', crs='EPSG:32618')
    seed_df.crs = 'EPSG:32618'
    with open('assets/traffic_results.geojson', 'w') as traffic:
        traffic.write(seed_df.to_json())
```

In order to evaluate the differing straight mult and traffic exp options, we standardized the metric results by their mean and standard deviation by seed, period, and inputs.

In [48]:
```
seed_df[['dist_ratio_z', 'traf_ratio_z', 'dist_avg_z']] = (seed_df
                    .groupby(['period', 'seed_station', 'seed_direction', 'straight_mult', 'traffic_exp'])[['di
                    .transform(lambda x: (x - x.mean()) / x.std()))
```

```
<ipython-input-48-0e26b2464c77>:3: DeprecationWarning: '-' operator will be deprecated. Use the 'difference' me
thod instead.
  .transform(lambda x: (x - x.mean()) / x.std()))
```

We separated the results from the morning and the afternoon rush hours and then selected only the seed inputs and algortihm inputs.

In [49]:
```
morning = seed_df.loc[seed_df['period'] == 'morning']
inputs = morning[['straight_mult', 'traffic_exp', 'seed_station', 'seed_direction']].drop_duplicates()
```

We selected only unique seeds.

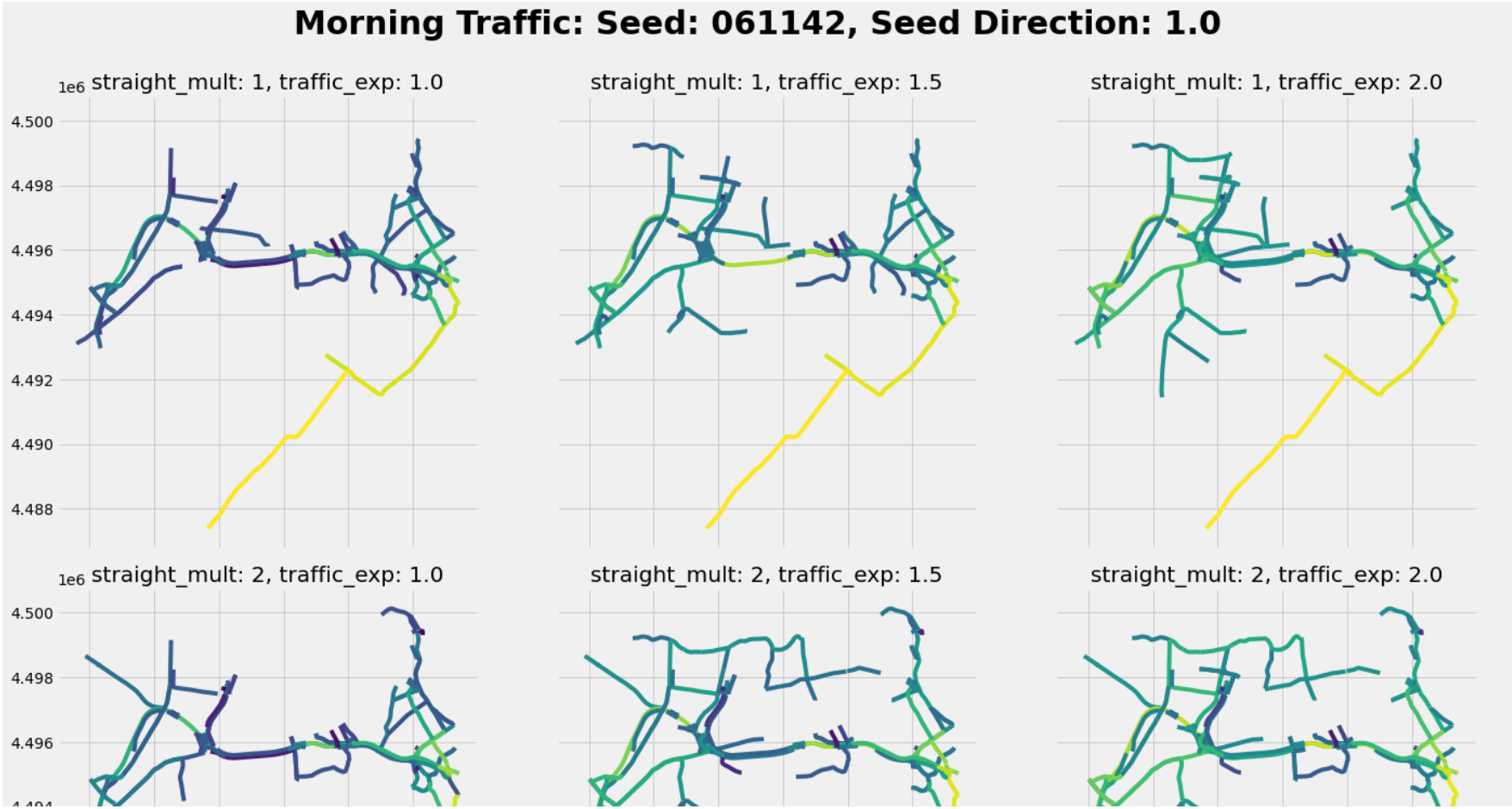
In [50]:
```
seeds = inputs[['seed_station', 'seed_direction']].drop_duplicates()
```

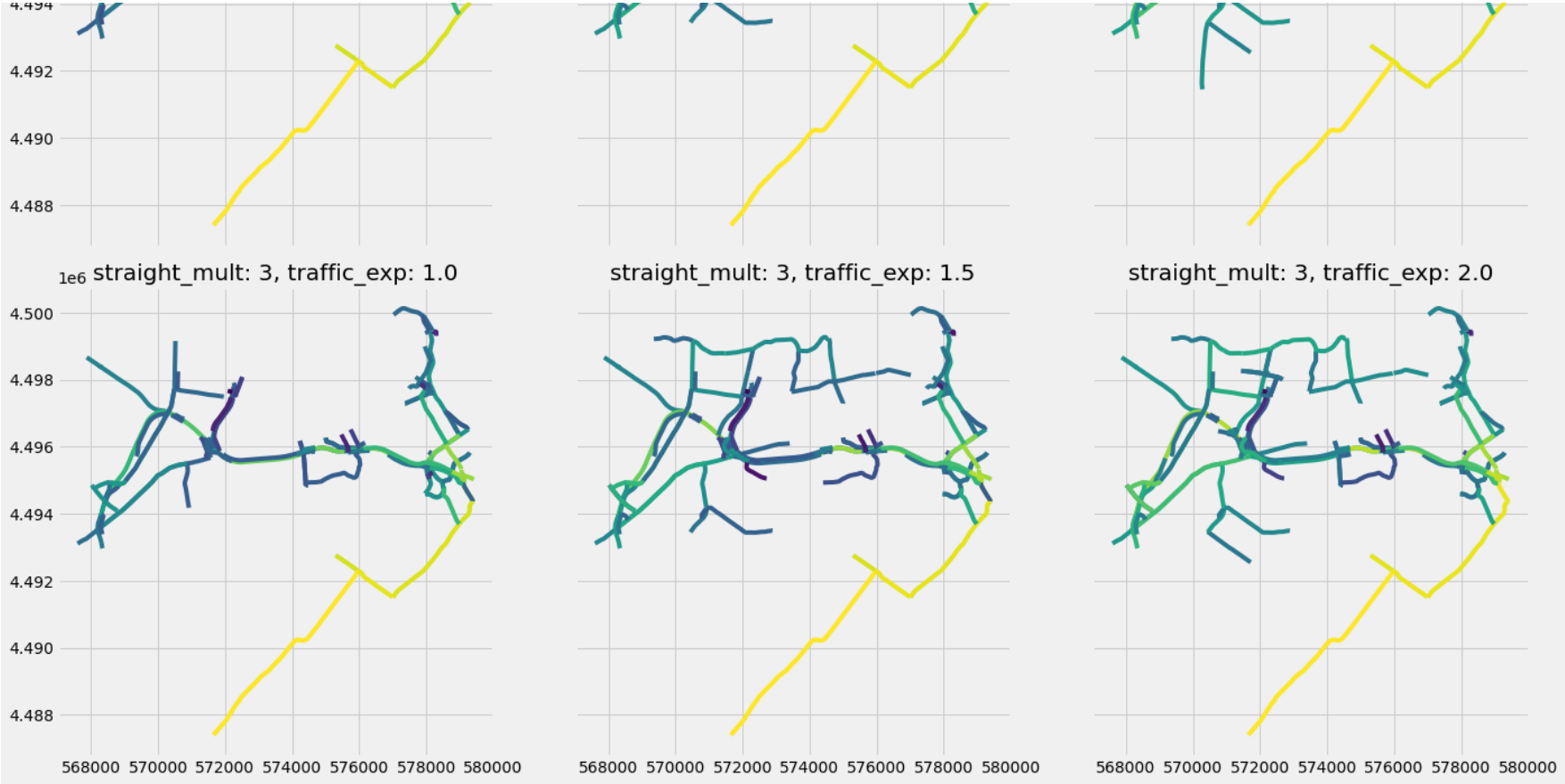
Visualizing created routes

We then created visualization to illustrate the effects of changing the straight multiplier and the traffic exponent for the seed stations. For this seed the area covered by the network increased with both an increase in the straight multiplier and the traffic exponent. The change was especially apparent with an increase in the traffic exponent.

In [51]:
```
%matplotlib inline

seed = inputs.loc[(inputs['seed_station'] == seeds.iloc[0]['seed_station']) & (inputs['seed_direction'] == seed
size = seed.shape[0]

plt.style.use('fivethirtyeight')
fig, axes = plt.subplots(int(size / 3), 3, figsize=(20, 20), sharex=True, sharey=True)
```

```
for j in range(3):
    for i in range(size // 3):
        dct = seed.iloc[i * 3 + j].to_dict()
        df = morning.loc[(morning['straight_mult'] == dct['straight_mult']) &
                         (morning['traffic_exp'] == dct['traffic_exp']) &
                         (morning['seed_station'] == dct['seed_station']) &
                         (morning['seed_direction'] == dct['seed_direction'])
                        ]
        df = df.groupby(['station', 'direction']).agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
        df = gpd.GeoDataFrame(df, geometry='geometry')
        df['log_traffic'] = np.log(df['traffic'])
        df.plot(column='log_traffic', ax=axes[i, j])
        axes[i, j].set_title('straight_mult: {}, traffic_exp: {}'.format(dct['straight_mult'], dct['traffic_exp
fig.suptitle('Morning Traffic: Seed: {}, Seed Direction: {}'.format(dct['seed_station'], dct['seed_direction'])
             fontweight='bold')
fig.tight_layout()
```

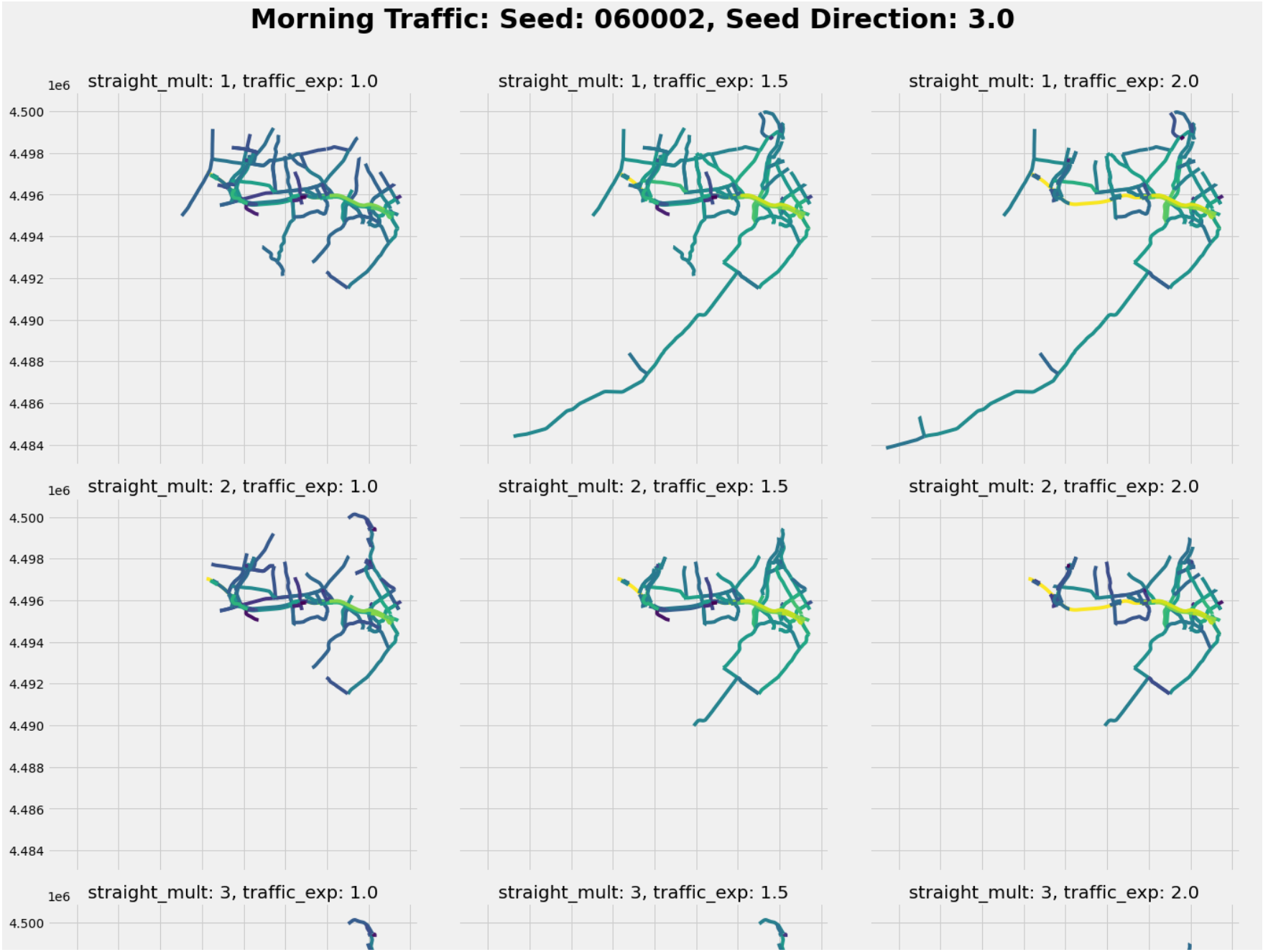


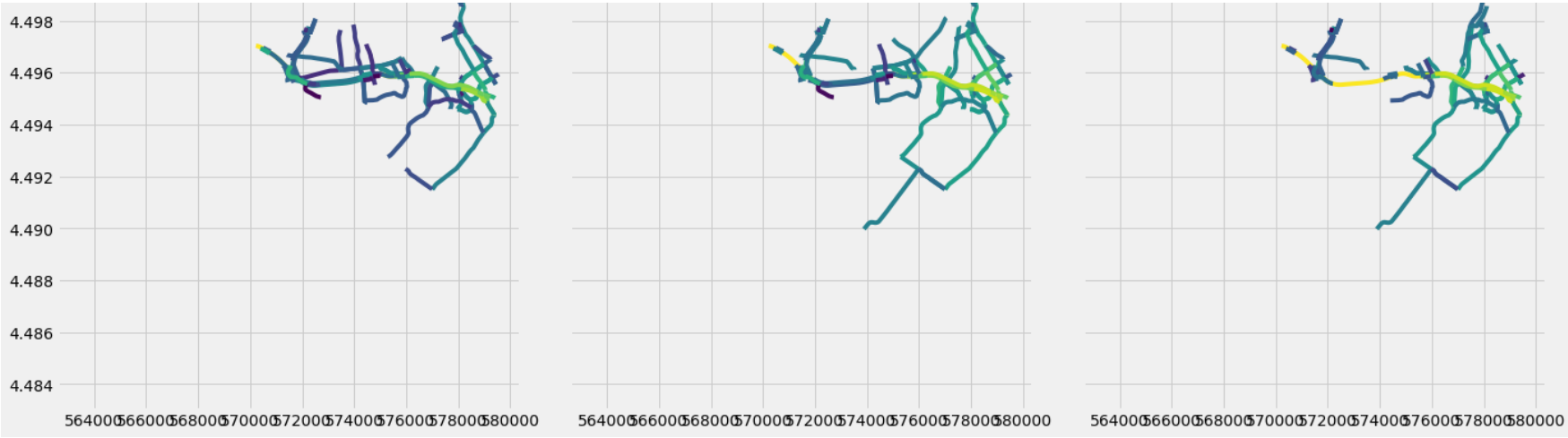


In [52]:

```
seed = inputs.loc[(inputs['seed_station'] == seeds.iloc[1]['seed_station']) & (inputs['seed_direction'] == seed
size = seed.shape[0]

plt.style.use('fivethirtyeight')
fig, axes = plt.subplots(int(size / 3), 3, figsize=(20, 20), sharex=True, sharey=True)
for j in range(3):
    for i in range(size // 3):
        dct = seed.iloc[i * 3 + j].to_dict()
        df = morning.loc[(morning['straight_mult'] == dct['straight_mult']) &
                         (morning['traffic_exp'] == dct['traffic_exp']) &
                         (morning['seed_station'] == dct['seed_station']) &
                         (morning['seed_direction'] == dct['seed_direction'])
                        ]
        df = df.groupby(['station', 'direction']).agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
        df = gpd.GeoDataFrame(df, geometry='geometry')
        df['log_traffic'] = np.log(df['traffic'])
        df.plot(column='log_traffic', ax=axes[i, j])
```

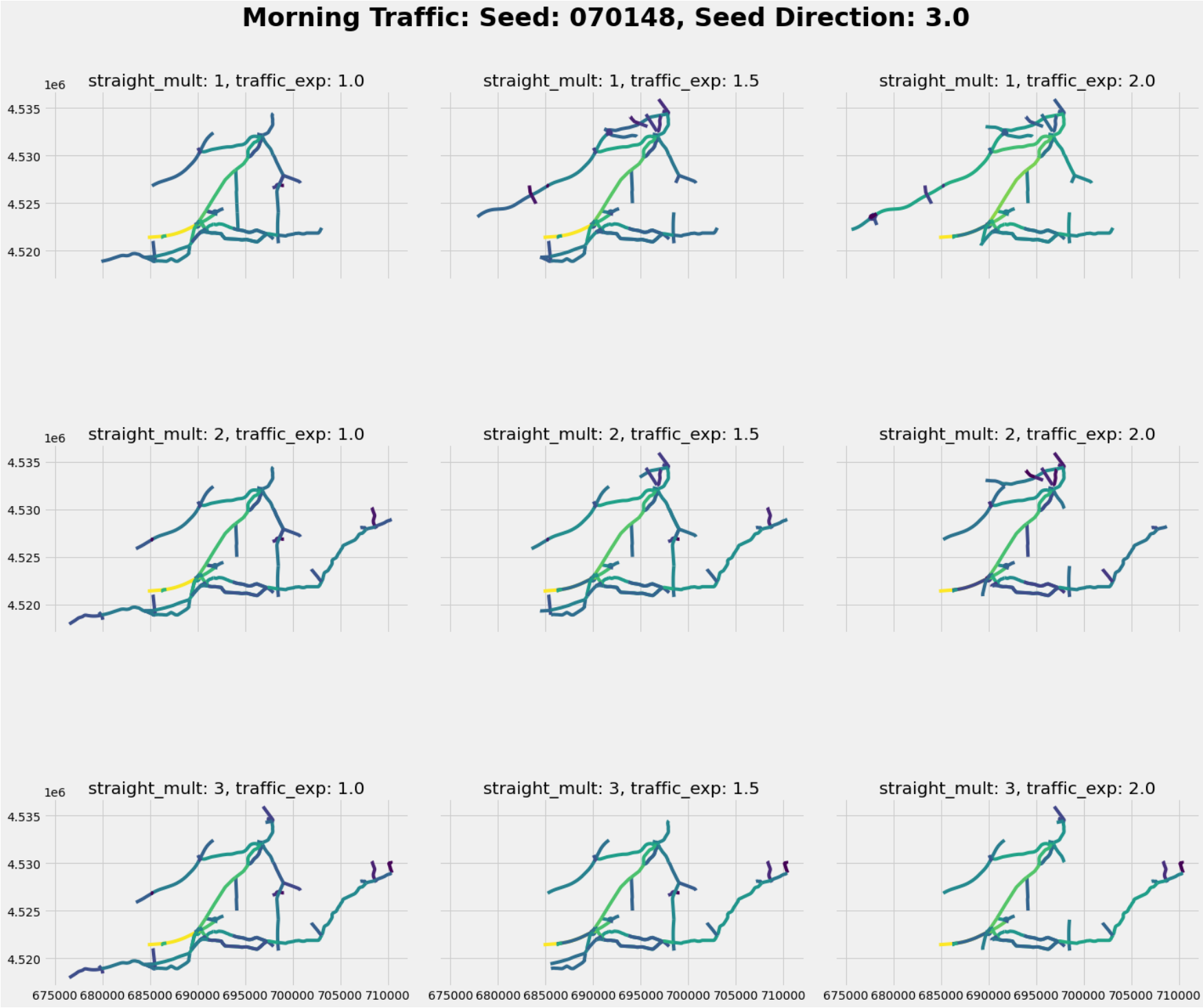
```
        axes[i, j].set_title('straight_mult: {}, traffic_exp: {}'.format(dct['straight_mult'], dct['traffic_exp
fig.suptitle('Morning Traffic: Seed: {}, Seed Direction: {}'.format(dct['seed_station'], dct['seed_direction'])
             fontweight='bold')
fig.tight_layout()
```





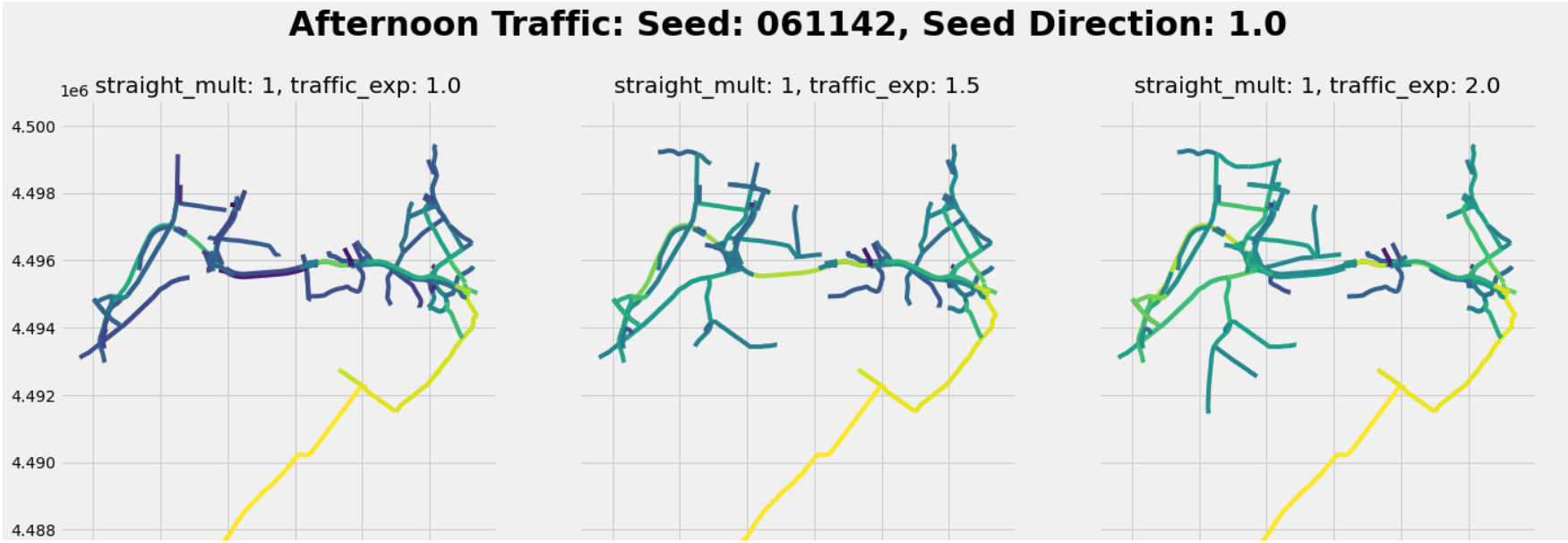
In [53]:

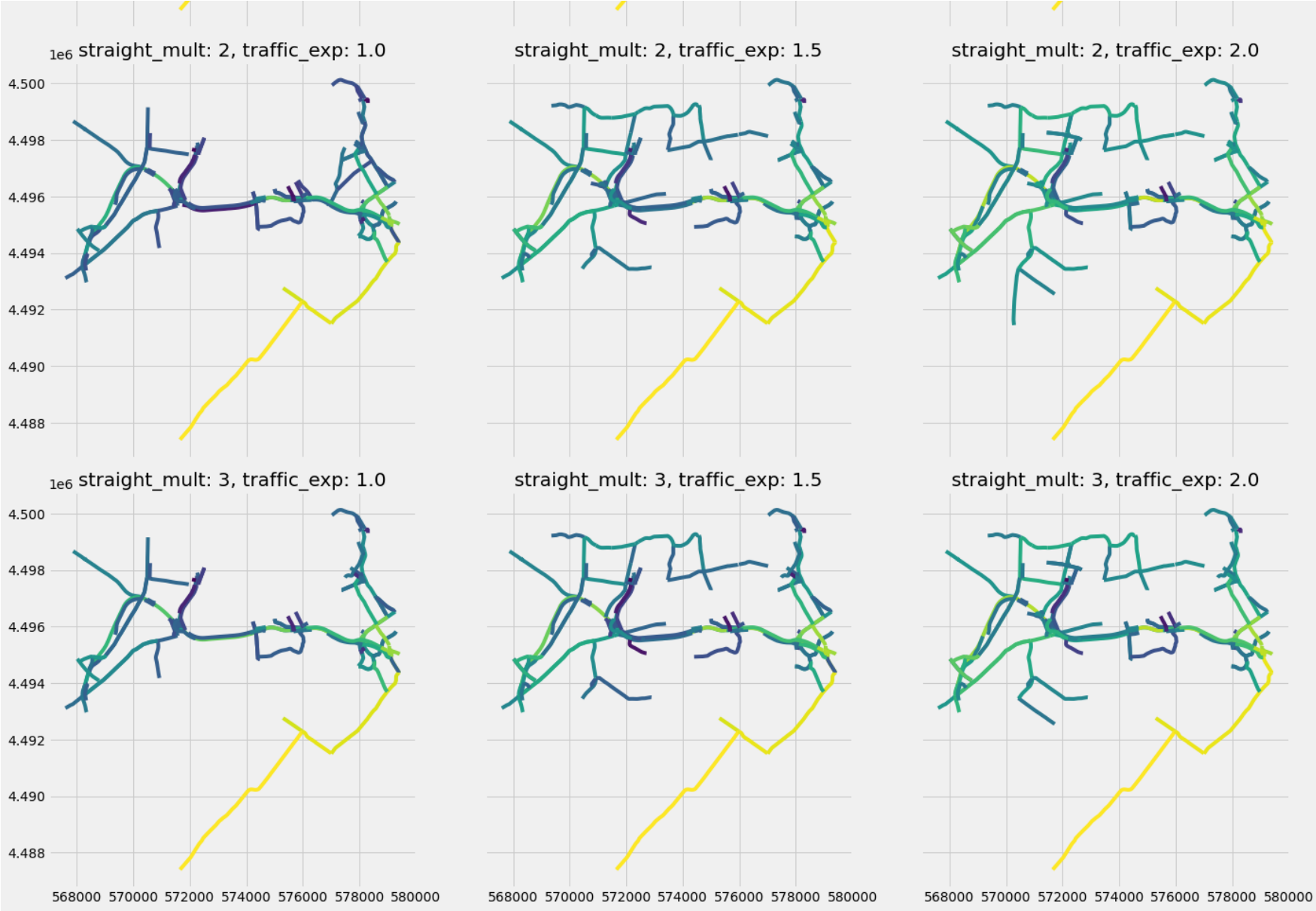
```
seed = inputs.loc[(inputs['seed_station'] == seeds.iloc[2]['seed_station']) & (inputs['seed_direction'] == seed
size = seed.shape[0]

plt.style.use('fivethirtyeight')
fig, axes = plt.subplots(int(size / 3), 3, figsize=(20, 20), sharex=True, sharey=True)
for j in range(3):
    for i in range(size // 3):
        dct = seed.iloc[i * 3 + j].to_dict()
        df = morning.loc[(morning['straight_mult'] == dct['straight_mult']) &
                         (morning['traffic_exp'] == dct['traffic_exp']) &
                         (morning['seed_station'] == dct['seed_station']) &
                         (morning['seed_direction'] == dct['seed_direction'])
                        ]
        df = df.groupby(['station', 'direction']).agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
        df = gpd.GeoDataFrame(df, geometry='geometry')
        df['log_traffic'] = np.log(df['traffic'])
        df.plot(column='log_traffic', ax=axes[i, j])
        axes[i, j].set_title('straight_mult: {}, traffic_exp: {}'.format(dct['straight_mult'], dct['traffic_exp
fig.suptitle('Morning Traffic: Seed: {}, Seed Direction: {}'.format(dct['seed_station'], dct['seed_direction'])
             fontweight='bold')
fig.tight_layout()
```


Morning Traffic: Seed: 070148, Seed Direction: 3.0
straight_mult: 1, traffic_exp: 1.0
straight_mult: 1, traffic_exp: 1.5
straight_mult: 1, traffic_exp: 2.0
straight_mult: 2, traffic_exp: 1.0
straight_mult: 2, traffic_exp: 1.5
straight_mult: 2, traffic_exp: 2.0
straight_mult: 3, traffic_exp: 1.0
straight_mult: 3, traffic_exp: 1.5
straight_mult: 3, traffic_exp: 2.0
1e6
4.535
4.530
4.525
4.520
675000 680000 685000 690000 695000 700000 705000 710000

In [54]:
```
after = seed_df.loc[seed_df['period'] == 'morning']
inputs = after[['straight_mult', 'traffic_exp', 'seed_station', 'seed_direction']].drop_duplicates()
```

In [55]:
```
seed = inputs.loc[(inputs['seed_station'] == seeds.iloc[0]['seed_station']) & (inputs['seed_direction'] == seed
size = seed.shape[0]

plt.style.use('fivethirtyeight')
fig, axes = plt.subplots(int(size / 3), 3, figsize=(20, 20), sharex=True, sharey=True)
for j in range(3):
    for i in range(size // 3):
        dct = seed.iloc[i * 3 + j].to_dict()
        df = after.loc[(after['straight_mult'] == dct['straight_mult']) &
                       (after['traffic_exp'] == dct['traffic_exp']) &
                       (after['seed_station'] == dct['seed_station']) &
                       (after['seed_direction'] == dct['seed_direction'])
                      ]
        df = df.groupby(['station', 'direction']).agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
        df = gpd.GeoDataFrame(df, geometry='geometry')
        df['log_traffic'] = np.log(df['traffic'])
        df.plot(column='log_traffic', ax=axes[i, j])
        axes[i, j].set_title('straight_mult: {}, traffic_exp: {}'.format(dct['straight_mult'], dct['traffic_exp
fig.suptitle('Afternoon Traffic: Seed: {}, Seed Direction: {}'.format(dct['seed_station'], dct['seed_direction'
             fontweight='bold')
fig.tight_layout()
```

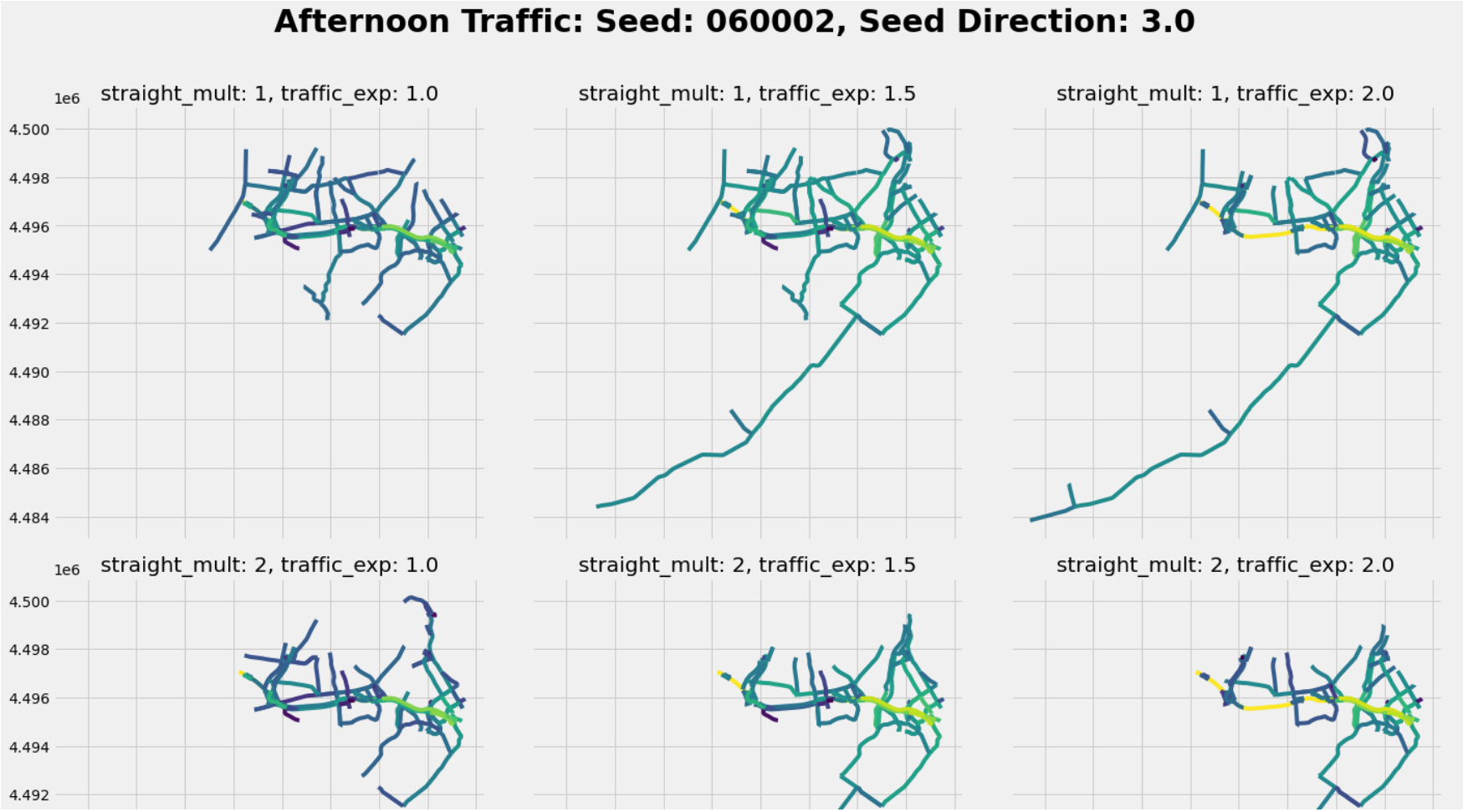


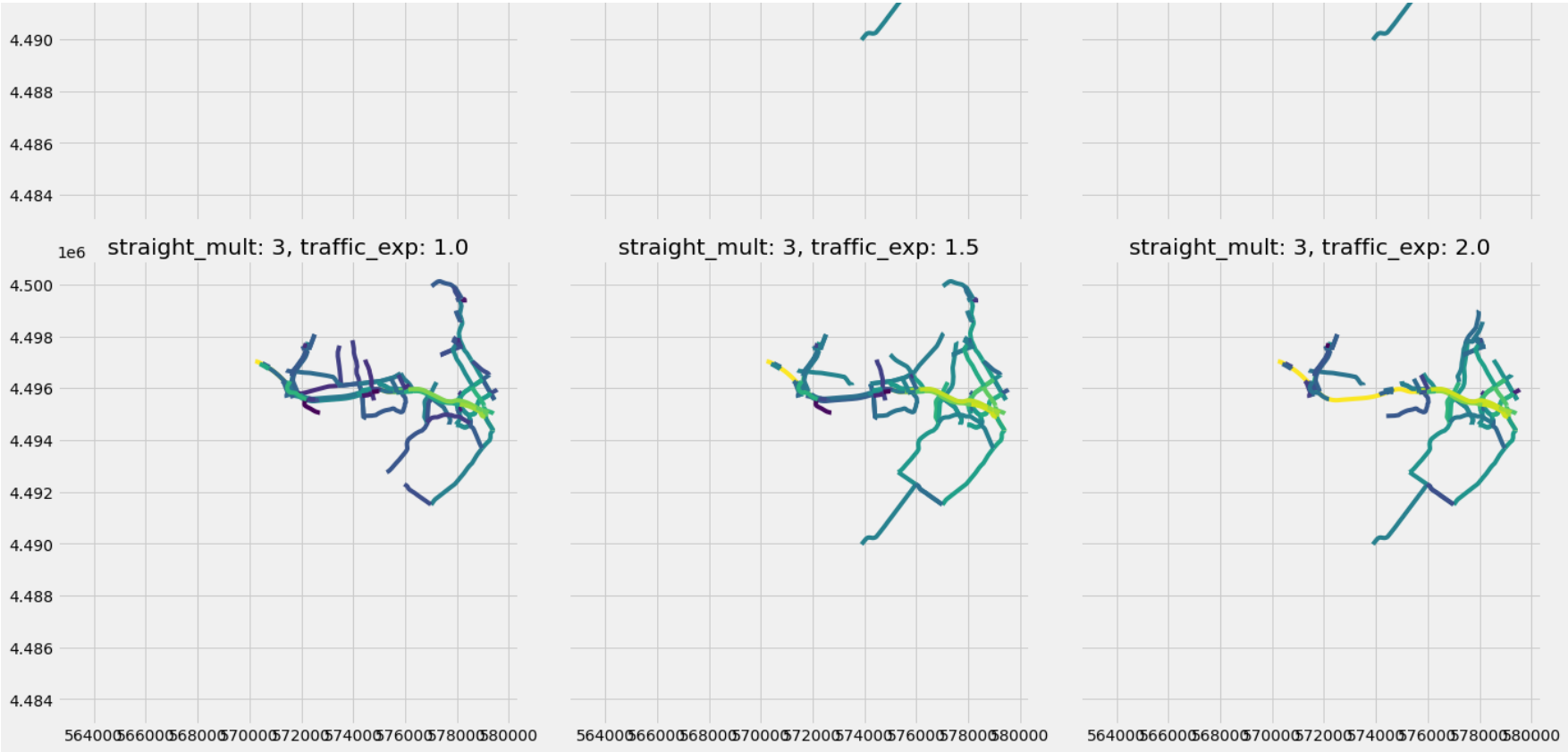


In [56]:

```
seed = inputs.loc[(inputs['seed_station'] == seeds.iloc[1]['seed_station']) & (inputs['seed_direction'] == seed
size = seed.shape[0]

plt.style.use('fivethirtyeight')
fig, axes = plt.subplots(int(size / 3), 3, figsize=(20, 20), sharex=True, sharey=True)
for j in range(3):
```

```
    for i in range(size // 3):
        dct = seed.iloc[i * 3 + j].to_dict()
        df = after.loc[(after['straight_mult'] == dct['straight_mult']) &
                       (after['traffic_exp'] == dct['traffic_exp']) &
                       (after['seed_station'] == dct['seed_station']) &
                       (after['seed_direction'] == dct['seed_direction'])
                      ]
        df = df.groupby(['station', 'direction']).agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
        df = gpd.GeoDataFrame(df, geometry='geometry')
        df['log_traffic'] = np.log(df['traffic'])
        df.plot(column='log_traffic', ax=axes[i, j])
        axes[i, j].set_title('straight_mult: {}, traffic_exp: {}'.format(dct['straight_mult'], dct['traffic_exp
fig.suptitle('Afternoon Traffic: Seed: {}, Seed Direction: {}'.format(dct['seed_station'], dct['seed_direction'
             fontweight='bold')
fig.tight_layout()
```

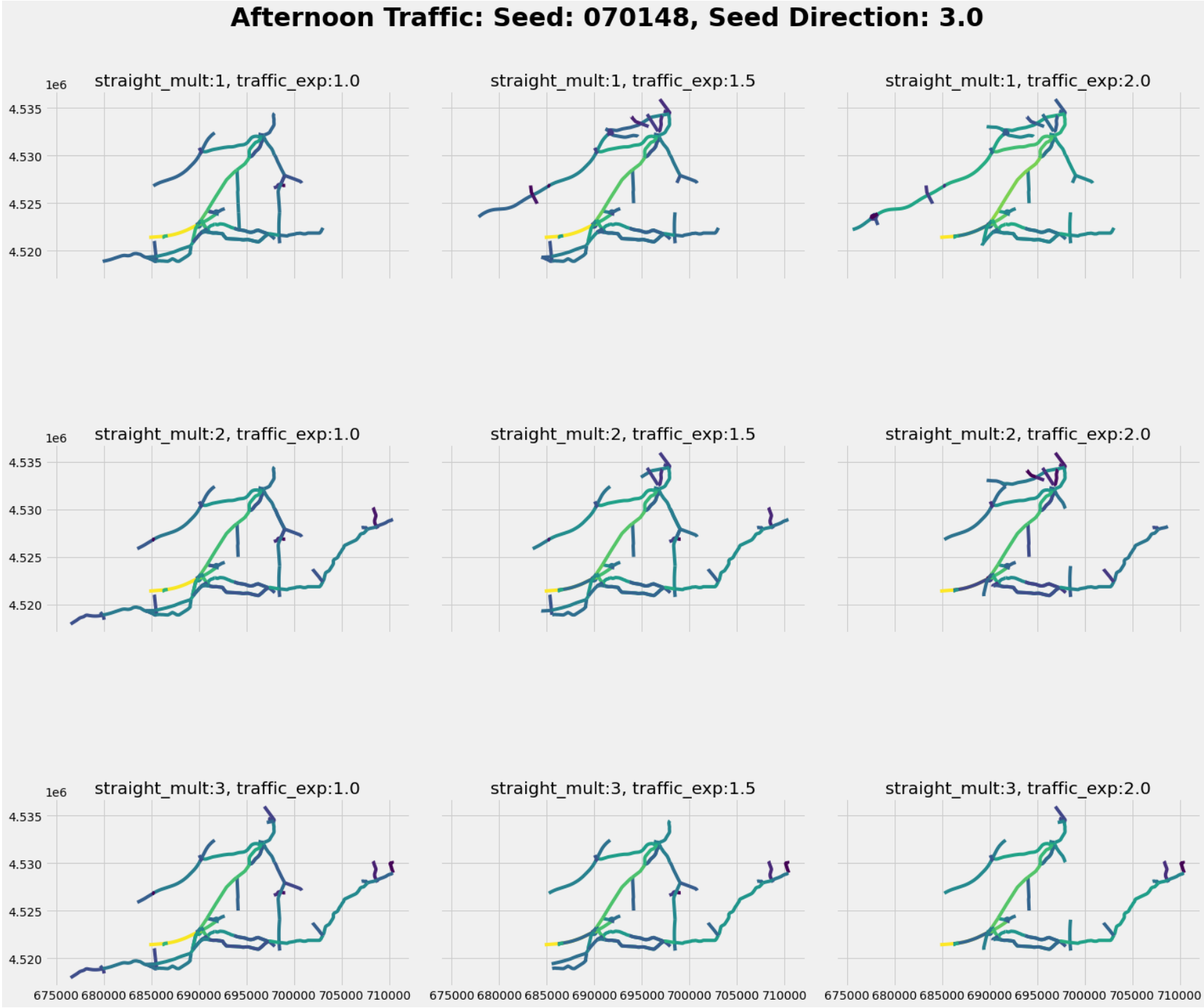




In [57]:

```
seed = inputs.loc[(inputs['seed_station'] == seeds.iloc[2]['seed_station']) & (inputs['seed_direction'] == seed
size = seed.shape[0]

plt.style.use('fivethirtyeight')
fig, axes = plt.subplots(int(size / 3), 3, figsize=(20, 20), sharex=True, sharey=True)
for j in range(3):
    for i in range(size // 3):
        dct = seed.iloc[i * 3 + j].to_dict()
        df = after.loc[(after['straight_mult'] == dct['straight_mult']) &
                       (after['traffic_exp'] == dct['traffic_exp']) &
                       (after['seed_station'] == dct['seed_station']) &
                       (after['seed_direction'] == dct['seed_direction'])
                      ]
        df = df.groupby(['station', 'direction']).agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
        df = gpd.GeoDataFrame(df, geometry='geometry')
        df['log_traffic'] = np.log(df['traffic'])
        df.plot(column='log_traffic', ax=axes[i, j])
        axes[i, j].set_title('straight_mult:{}, traffic_exp:{}'.format(dct['straight_mult'], dct['traffic_exp']
```

```
fig.suptitle('Afternoon Traffic: Seed: {}, Seed Direction: {}'.format(dct['seed_station'], dct['seed_direction'
             fontweight='bold')
fig.tight_layout()
```


Afternoon Traffic: Seed: 070148, Seed Direction: 3.0
straight_mult:1, traffic_exp:1.0
straight_mult:1, traffic_exp:1.5
straight_mult:1, traffic_exp:2.0
straight_mult:2, traffic_exp:1.0
straight_mult:2, traffic_exp:1.5
straight_mult:2, traffic_exp:2.0
straight_mult:3, traffic_exp:1.0
straight_mult:3, traffic_exp:1.5
straight_mult:3, traffic_exp:2.0
1e6
4.535
4.530
4.525
4.520
675000 680000 685000 690000 695000 700000 705000 710000

Visualizing metric variation by model inputs

Next, we combined the results by seed station and input. we then standardized the metrics for visualization.

In [58]:
```
results = (seed_df
 .groupby(['seed_station', 'seed_direction', 'period', 'straight_mult', 'traffic_exp'])
 .agg({'dist_ratio': 'first', 'traf_ratio': 'first','dist_avg': 'first'})
).reset_index()
results[['dist_ratio_stand', 'traf_ratio_stand', 'dist_avg_stand']] = (results
    .groupby(['seed_station', 'seed_direction', 'period'])
            [['dist_ratio', 'traf_ratio', 'dist_avg']]
    .transform(lambda x: (x - x.mean()) / x.std()))
results.drop(['dist_ratio', 'traf_ratio', 'dist_avg'], axis=1, inplace=True)
```

We stacked the metrics in order to visualize them in Altair. We then combined the seed station and direction into one column, then the two inputs into one column to form unique identifiers.

In [59]:
```
results = results.set_index(['seed_station', 'seed_direction', 'period', 'straight_mult', 'traffic_exp']).stack
results['station_dir'] = results.apply(lambda x: x['seed_station'] + '\t' + str(x['seed_direction']), axis=1)
results.rename({'level_5': 'metric', 0: 'metric_value'}, axis=1, inplace=True)
results['straight_traf'] = results.apply(lambda x: str(x['straight_mult']) + '\t' + str(x['traffic_exp']), axis
```

We combined the metric column and converted the period to title format.

In [60]:
```
results['metric'] = results['metric'].map(lambda x: 'Standardized Distance Ratio' if x == 'dist_ratio_stand'
                                          else 'Standardized Traffic Ratio' if x == 'traf_ratio_stand'
                                          else 'Standardized Average Distance')
results['period'] = results['period'].map(lambda x: x.title())
```

We then visualized the standardized evaluation metrics as well as the inputs for each station to examine which input combinations should be used. A straight multiplier of 3 and a traffic exponent of 1 were chosen.

In [61]:
```
seeds = results['station_dir'].unique()

alt.themes.enable('fivethirtyeight')
alt.Chart(results.loc[results['station_dir'] == seeds[0]]).mark_bar().encode(
    x=alt.X('straight_traf:N', axis=alt.Axis(
    labelExpr='"Straight Mult: " + split(datum.label, "\t")[0] + ",\tTraffic Exp: " + split(datum.label, "\t")[
    labelAngle=-45,
    title=None
```

```
    )),
    y=alt.Y('metric_value:Q', title=None),
    row=alt.Row('period:N', title=None),
    column=alt.Column('metric:N', title=None)
).properties(title='Scores for seed station {} in direction {}'
             .format(seeds[0].split('\t')[0], seeds[0].split('\t')[1]))
```

Out[61]:

In [62]:

```
alt.Chart(results.loc[results['station_dir'] == seeds[1]]).mark_bar().encode(
    x=alt.X('straight_traf:N', axis=alt.Axis(
    labelExpr='"Straight Mult: " + split(datum.label, "\t")[0] + ",\tTraffic Exp: " + split(datum.label, "\t")[
    labelAngle=-45,
    title=None
    )),
    y=alt.Y('metric_value:Q', title=None),
    row=alt.Row('period:N', title=None),
    column=alt.Column('metric:N', title=None)
).properties(title='Scores for seed station {} in direction {}'
             .format(seeds[1].split('\t')[0], seeds[1].split('\t')[1]))
```

Out[62]:

In [63]:

```
alt.Chart(results.loc[results['station_dir'] == seeds[2]]).mark_bar().encode(
    x=alt.X('straight_traf:N', axis=alt.Axis(
    labelExpr='"Straight Mult: " + split(datum.label, "\t")[0] + ",\tTraffic Exp: " + split(datum.label, "\t")[
    labelAngle=-45,
    title=None
    )),
    y=alt.Y('metric_value:Q', title=None),
    row=alt.Row('period:N', title=None),
    column=alt.Column('metric:N', title=None)
).properties(title='Scores for seed station {} in direction {}'
             .format(seeds[2].split('\t')[0], seeds[2].split('\t')[1]))
```

Out[63]:

In [64]:

```
alt.Chart(results.loc[results['station_dir'] == seeds[3]]).mark_bar().encode(
    x=alt.X('straight_traf:N', axis=alt.Axis(
    labelExpr='"Straight Mult: " + split(datum.label, "\t")[0] + ",\tTraffic Exp: " + split(datum.label, "\t")[
    labelAngle=-45,
```

```
    title=None
    )),
    y=alt.Y('metric_value:Q', title=None),
    row=alt.Row('period:N', title=None),
    column=alt.Column('metric:N', title=None)
).properties(title='Scores for seed station {} in direction {}'
             .format(seeds[2].split('\t')[0], seeds[2].split('\t')[1]))
```

Out[64]:

In [65]:

```
alt.Chart(results.loc[results['station_dir'] == seeds[4]]).mark_bar().encode(
    x=alt.X('straight_traf:N', axis=alt.Axis(
    labelExpr='"Straight Mult: " + split(datum.label, "\t")[0] + ",\tTraffic Exp: " + split(datum.label, "\t")[
    labelAngle=-45,
    title=None
    )),
    y=alt.Y('metric_value:Q', title=None),
    row=alt.Row('period:N', title=None),
    column=alt.Column('metric:N', title=None)
).properties(title='Scores for seed station {} in direction {}'
             .format(seeds[2].split('\t')[0], seeds[2].split('\t')[1]))
```

Out[65]:

In [66]:

```
alt.Chart(results.loc[results['station_dir'] == seeds[5]]).mark_bar().encode(
    x=alt.X('straight_traf:N', axis=alt.Axis(
    labelExpr='"Straight Mult: " + split(datum.label, "\t")[0] + ",\tTraffic Exp: " + split(datum.label, "\t")[
    labelAngle=-45,
    title=None
    )),
    y=alt.Y('metric_value:Q', title=None),
    row=alt.Row('period:N', title=None),
    column=alt.Column('metric:N', title=None)
).properties(title='Scores for seed station {} in direction {}'
             .format(seeds[2].split('\t')[0], seeds[2].split('\t')[1]))
```

Out[66]:

Choropleth maps

We selected only the data for the chosen metric values. We then reformed the dataframe into a GeoDataFrame object and converted crs to lat-long.

In [68]:
```
part = seed_df.loc[(seed_df['straight_mult'] == 3) & (seed_df['traffic_exp'] == 1) & (seed_df['period'] == 'mor
part = gpd.GeoDataFrame(part, geometry='geometry')
part.crs = 'EPSG:32618'
part = part.to_crs(4326)
```

We grouped by station and dirction to find the total traffic by segment. We then created a buffer around the line objects to aid in visualization. We calcualted the logrithms of traffic counts to better standardize the values.

In [73]:
```
part['sta_dir'] = part.apply(lambda x: str(x['station']) + '\t' + str(x['direction']), axis=1)
traffic_part = part.groupby('sta_dir').agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
traffic_part = gpd.GeoDataFrame(traffic_part, geometry='geometry', crs=4326)
traffic_part['geometry'] = traffic_part.geometry.buffer(0.0005)
traffic_part['log_traf'] = np.log(traffic_part['traffic'])
```

```
<ipython-input-73-9da0dc277d24>:4: UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are like
ly incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this operation.

  traffic_part['geometry'] = traffic_part.geometry.buffer(0.0005)
```

This is a choropleth map of the combined traffic from seeds at morning rush hour. The seeds were located on Long Island and Staten Island. The log of traffic is visible with a further zoom.

In [86]:
```
import folium

m = folium.Map(location=[40.71, -74], zoom_start=9)

folium.Choropleth(
    geo_data=traffic_part,
    data=traffic_part,
    columns=['sta_dir', 'log_traf'],
    key_on="feature.properties.sta_dir",
    fill_opacity=0.8,
    line_opacity=0.5
).add_to(m)
m
```

Out[86]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [87]:
```
part_after = seed_df.loc[(seed_df['straight_mult'] == 3) & (seed_df['traffic_exp'] == 1) & (seed_df['period'] =
part_after = gpd.GeoDataFrame(part_after, geometry='geometry')
part_after.crs = 'EPSG:32618'
part_after = part_after.to_crs(4326)
```

In [88]:
```
part_after['sta_dir'] = part_after.apply(lambda x: str(x['station']) + '\t' + str(x['direction']), axis=1)
traffic_part_after = part_after.groupby('sta_dir').agg({'traffic': 'sum', 'geometry': 'first'}).reset_index()
traffic_part_after = gpd.GeoDataFrame(traffic_part_after, geometry='geometry', crs=4326)
traffic_part_after['geometry'] = traffic_part_after.geometry.buffer(0.0005)
traffic_part_after['log_traf'] = np.log(traffic_part_after['traffic'])
```

```
<ipython-input-88-7f98422b6d37>:4: UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are like
ly incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this operation.
```

```
traffic_part_after['geometry'] = traffic_part_after.geometry.buffer(0.0005)
```

This is the same choropleth map for afternoon rush hour. This time the seeds are more centered on the city. The log of traffic is visible with a further zoom.

In [89]:
```
m = folium.Map(location=[40.71, -74])

folium.Choropleth(
    geo_data=traffic_part_after,
    data=traffic_part_after,
    columns=['sta_dir', 'log_traf'],
    key_on="feature.properties.sta_dir",
    fill_opacity=0.8,
    line_opacity=0.5
).add_to(m)
m
```

Out[89]: Make this Notebook Trusted to load map: File -> Trust Notebook

In order to create a dataset for electric vehicle travel, we narrowed the traffic to only morning rush hour. We assumed that this indicates commutes from home to work. Using afternoon rush hour is likely unrepresentative of travel patterns because we are using registration data by zipcode as a proxy.

In [91]:
```
top_morning = top_morning.sort_values('traffic', ascending=False)
```

We instanciated the class with a straight multiplier of 2 and a traffic exponent of 2.

In [92]:
```
gen_point = GeneratePoints(traffic_nyc, straight_mult=2, traffic_exp=2)
```

We built a dataframe for trips from the top 100 zipcodes based on electric vehicle registration. We chose a stopping point of 0.5% of total EV traffic from each zipcode.

In [93]:
```
if not os.path.exists('assets/top_routes.geojson'):

    # Choose top 100 locations with minuimum percentage of 0.5%.
    result = gen_point.build_full_df(top_morning.iloc[:100], min_perc=0.005, fp='')

    # Find number of routes. Then, use this to standardize traffic by number of routes.
    result['route_num'] = result.groupby(['seed_direction', 'seed_traffic'])[['route']].transform('max')
    result = result.loc[result['traffic'] != 0]
    result['traffic_adj'] = result['traffic'].div(result['route_num'])

    # Reform dataframe and set crs.
    result = gpd.GeoDataFrame(result, geometry='geometry')
    result.crs = 'EPSG:32618'

    # Change crs to 4326, which is latlong.
    result = result.to_crs('EPSG:4326')
    with open('assets/top_routes.geojson', 'w') as top_routes:
        top_routes.write(result.to_json())
else:
    result = gpd.read_file('assets/top_routes.geojson')
    result.crs = 'EPSG:4326'
```

We then grouped by station and direction to find total traffic at each location.

In [99]:
```
result_per_station = result.groupby(['station', 'direction']).agg({'traffic_adj': 'sum', 'geometry': 'first'}).
```

We created a buffer around each line to aid in visualization. Then we took the log of traffic to differentiate traffic better in the choropleth.

In [100…
```
result_per_station['geometry_buffer'] = gpd.GeoSeries(result_per_station['geometry']).buffer(0.0003)
result_per_station['log_traf'] = np.log(result_per_station['traffic_adj'])
result_per_station.rename({'geometry': 'geom_line', 'geometry_buffer': 'geometry'}, axis=1, inplace=True)
result_per_station.drop('geom_line', axis=1, inplace=True)
```

We converted back to a dataframe and set crs.

In [101…
```
result_per_station = gpd.GeoDataFrame(result_per_station, geometry='geometry')
result_per_station.crs = 'EPSG:4326'
```

In [102…
```
result_per_station.head()
```

Out[102…

	station	direction	traffic_adj	geometry	log_traf
0	010001	3.0	0.002918	POLYGON ((-73.92800 40.84561, -73.92793 40.845...	-5.836828
1	010001	7.0	3.857422	POLYGON ((-73.92721 40.84481, -73.92721 40.844...	1.349999
2	010002	3.0	10.217468	POLYGON ((-73.92692 40.84536, -73.92691 40.845...	2.324099
3	010002	7.0	0.281236	POLYGON ((-73.91751 40.84474, -73.91792 40.844...	-1.268561
4	010003	1.0	0.050365	POLYGON ((-73.90581 40.84450, -73.90585 40.844...	-2.988455

We combined station and direction to create a distinct index.

In [103…
```
result_per_station['sta_dir'] = result_per_station.apply(lambda x: x['station'] + ' ' + str(x['direction']), ax
```

This is a map of the route network with derived EV traffic by segment. Zoom in to see more detail.

In [104…
```
m = folium.Map(location=[40.71, -74])

folium.Choropleth(
    geo_data=result_per_station,
```

```
    data=result_per_station,
    columns=['sta_dir', 'log_traf'],
    key_on="feature.properties.sta_dir",
    fill_opacity=0.8,
    line_opacity=0.5
).add_to(m)
m
```

Out[104… Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]: