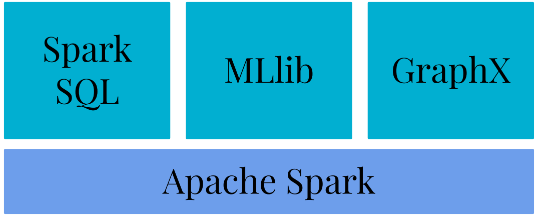
Traffic Simulation in NYC

Team Violet Noise
Ameya, Emma, Helene, Kristin

Big Data Application Development CSCI GA 2437

Motivation and Concept

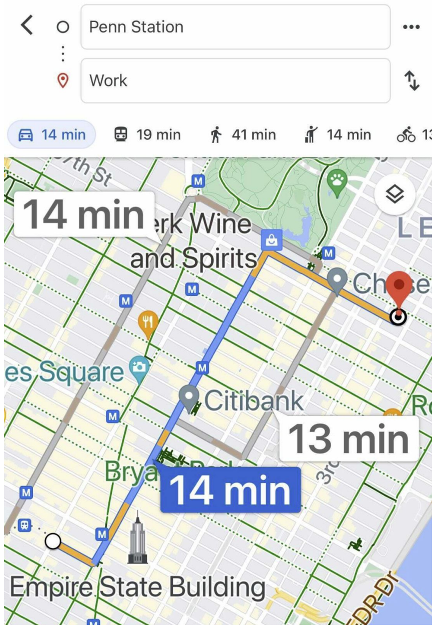


Goals

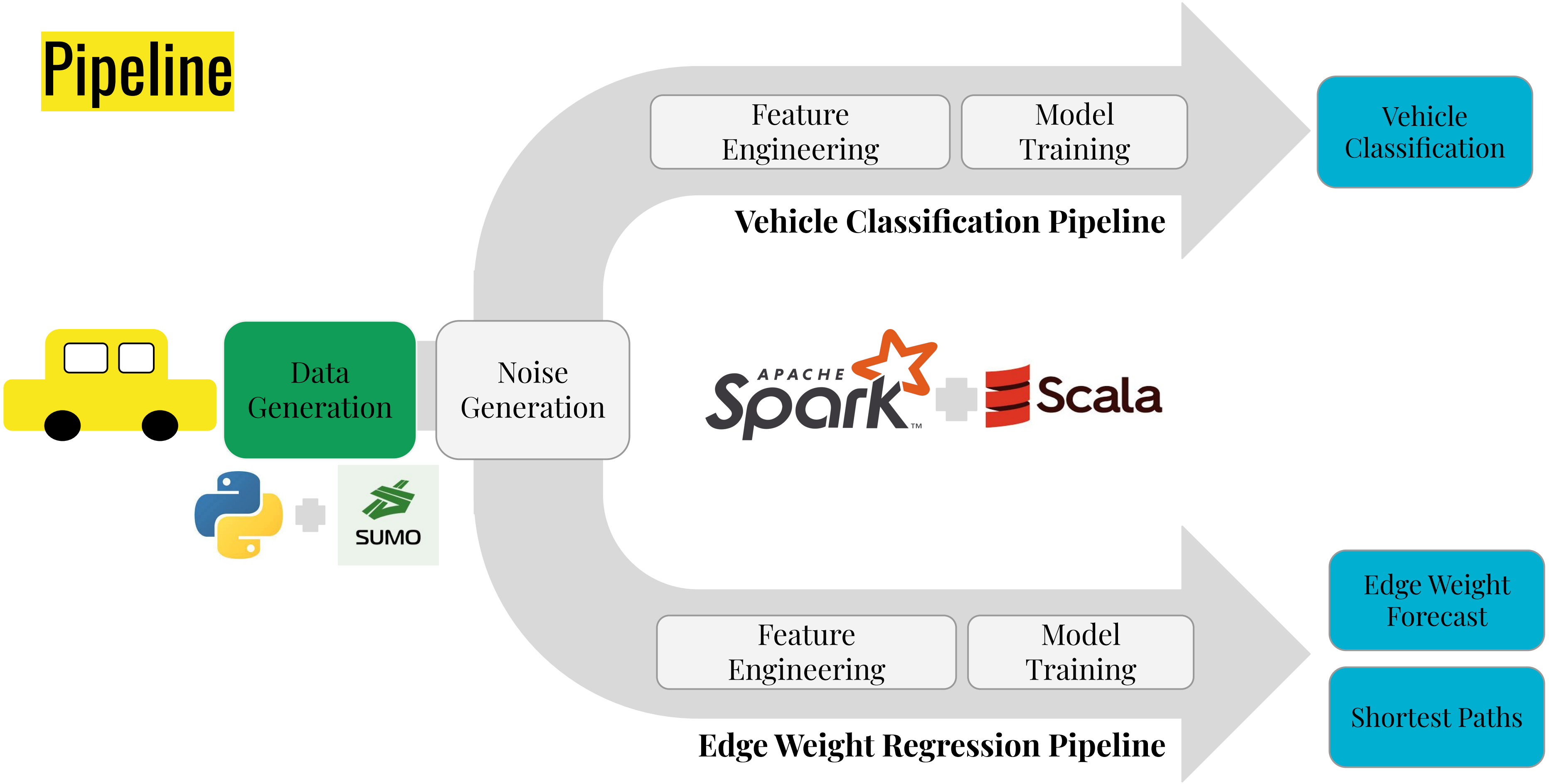
1. Combine several main components of Apache Spark
    - Data processing
    - Machine learning
    - Graph algorithms
2. Try different types of algorithms to see how they handle graph features, and graph based noise

Concept

1. Simulate data for traffic in NYC
2. Assume that we collect the data through some IoT framework - we only have GPS type signals and general metadata about the graph
3. Attempt to solve for shortest path recommendation and vehicle identification



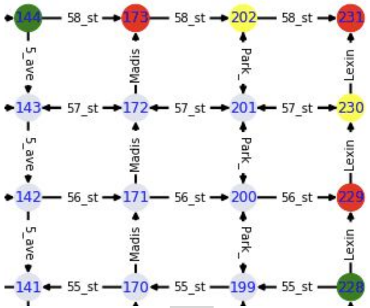
Pipeline

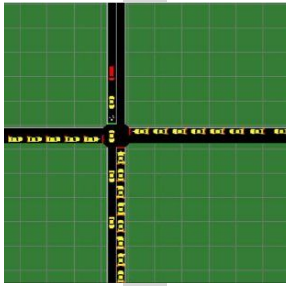

Feature Engineering
Model Training
Vehicle Classification
Vehicle Classification Pipeline
Data Generation
Noise Generation
SUMO
APACHE Spark
Scala
Feature Engineering
Model Training
Edge Weight Forecast
Shortest Paths
Edge Weight Regression Pipeline

Data Generation

Data Generation

1. Created a graph to mimic midtown manhattan
    - Nodes are intersections
    - Edges are streets
    - “Source” and “Sink” nodes along the edge allow traffic flow
    - Edges have varying sizes, 2-4 lanes
2. Calculated all reasonable paths for every pair of nodes
3. Assign cars to these paths at regular intervals
4. Create instructions in XML for sumo to implement our traffic scenario
5. Extract trace data for all vehicles from sumo ~ **550 GB for 70 days**

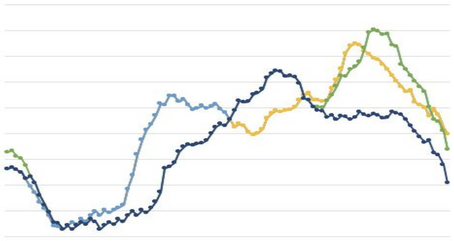




```
1.5 G     4.4 G     /user/jl11257/big_data_project/traces/demo
54.9 G    164.8 G   /user/jl11257/big_data_project/traces/noised
61.4 G    184.2 G   /user/jl11257/big_data_project/traces/processed
550.3 G   1.6 T     /user/jl11257/big_data_project/traces/raw
```

Features Embedded in Data Generation

1. Created time series for car arrival rates for each “source”



```
cars = int(np.random.poisson(rate))
spread_cars = [min(i,2) for i in np.histogram(list(range(cars)),bins=900,density=False)[0]]
timestamps = [i + minute*60 + start_tm_seconds for i in range(900)]
cars_per_sec = [i for i in zip(timestamps, spread_cars) if i[1] > 0]
```

2. Different vehicles have different sizes and speed properties

```
<vType accel="1.0" decel="3.5" id="Bus" length="15.0" maxSpeed="8.0" sigma="0.5" >
<vType accel="2.7" decel="4.6" id="Car1" length="4.0" maxSpeed="11.2" sigma="0.5">
<vType accel="2.4" decel="4.5" id="Car2" length="5.0" maxSpeed="11.2" sigma="0.5">
<vType accel="1.9" decel="4.3" id="Car3" length="7.0" maxSpeed="11.2" sigma="0.5">
```

3. Buses travel on a separate schedule, with specific routes and timed stops

```
{"busroute": "230|27",
"active_hrs": [5, 7, 9, 11, 13, 15, 17, 19, 21],
"start_min": 2,
"stops": ["station15"]}
```

Adding Noise

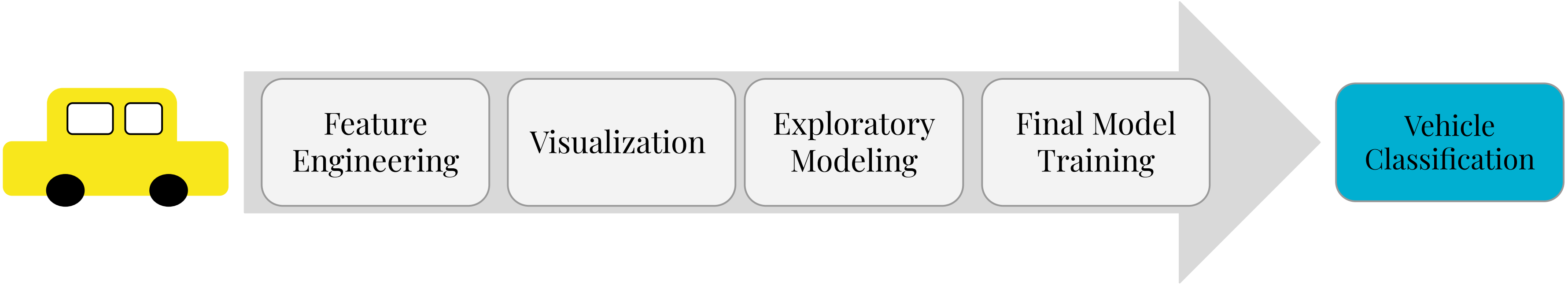
1. **Random traffic jams**
    - 1-4 per day
    - 3-9 min per jam
2. **Making data more realistic** (~messy IOT device data)
    - 5% data loss from faulty GPS signals
    - Cartesian coordinates » GPS coordinates
    - Dropped speed feature from raw data

```
val accident = udf((choice: Int) => {
                val windowlength = if ((choice % 2) == 0) 3 else 5
                val startTime = if ((choice % 2) == 0) 7 else 14
                val duration = rand_gen.nextInt(max_accident_duration-min_accident_duration)+min_accident_duration
                val start_time = rand_gen.nextInt(windowlength*60*60-duration+1)+(startTime*60*60)
                val end_time = start_time+duration
                val node = accident_candidates(rand_gen.nextInt(accident_candidates.length))(0).toString
                Accident(start_time, end_time, node)
            }
        )
```

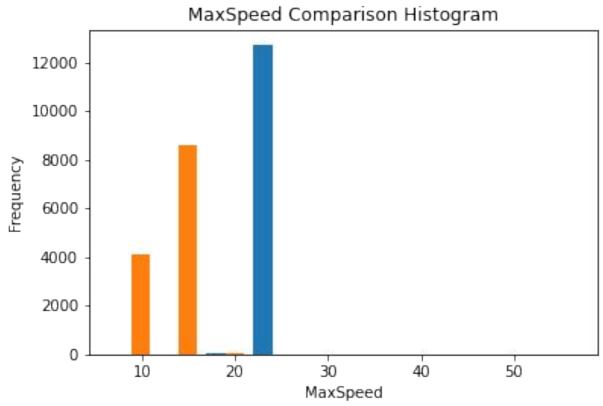
```
val transformXY = udf((x: Double, y: Double) => {
                val xorig = if (x < 0) 0.0 else x / 100.0
                val yorig = if (y < 0) 0.0 else y / 100.0
                val mcoords = new DenseMatrix(3, 1, Array(xorig, yorig, 1))
                val output = transformer.multiply(mcoords)
                val latitude = BigDecimal(output.apply(0,0) / output.apply(2,0)).setScale(6,
BigDecimal.RoundingMode.HALF_UP).toDouble
                val longitude = BigDecimal(output.apply(1,0) / output.apply(2,0)).setScale(6,
BigDecimal.RoundingMode.HALF_UP).toDouble
                GpsCoord(latitude, longitude)
            })
```

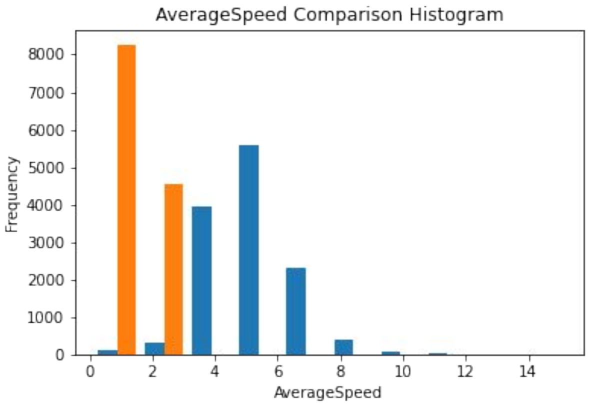
Vehicle Classification

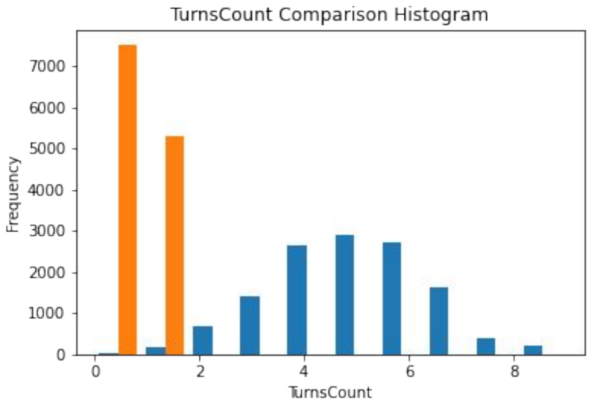
Vehicle Classification Pipeline



Feature Visualizations

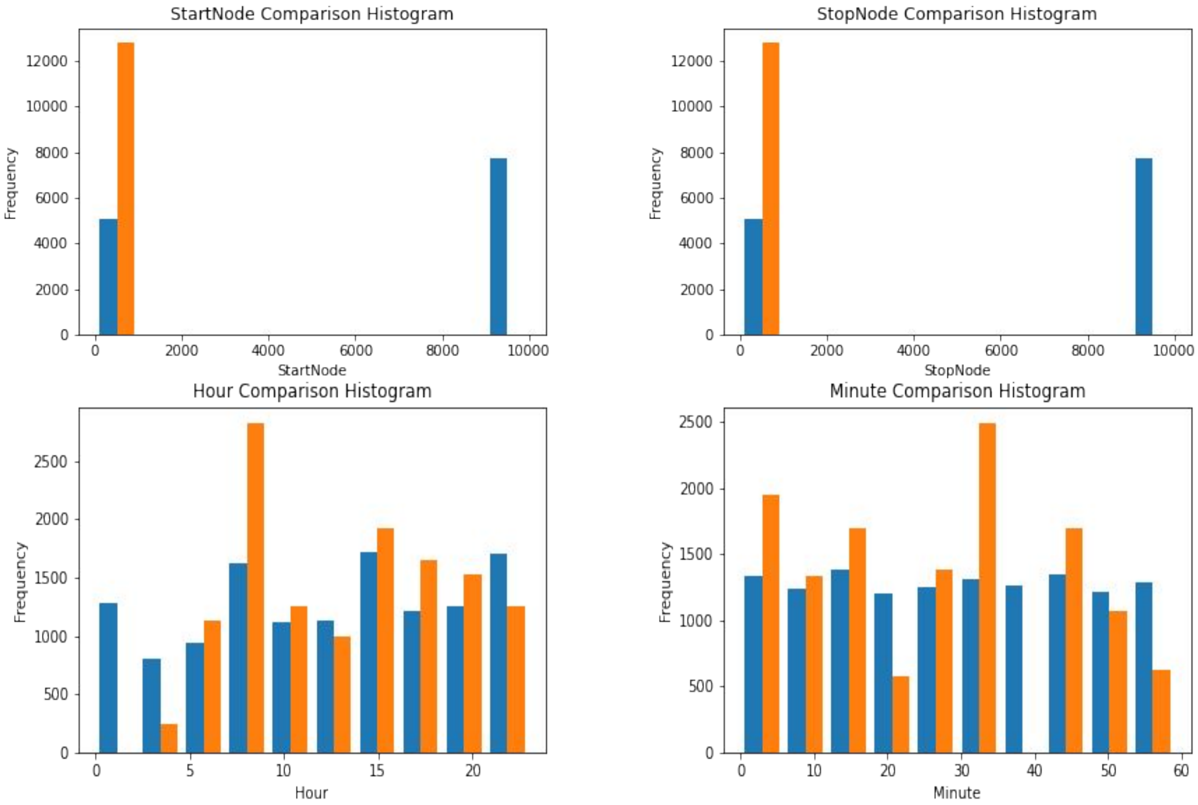






MaxSpeed, AverageSpeed, and TurnsCount features of Car(blue) and Bus(orange) comparison

Feature Visualizations



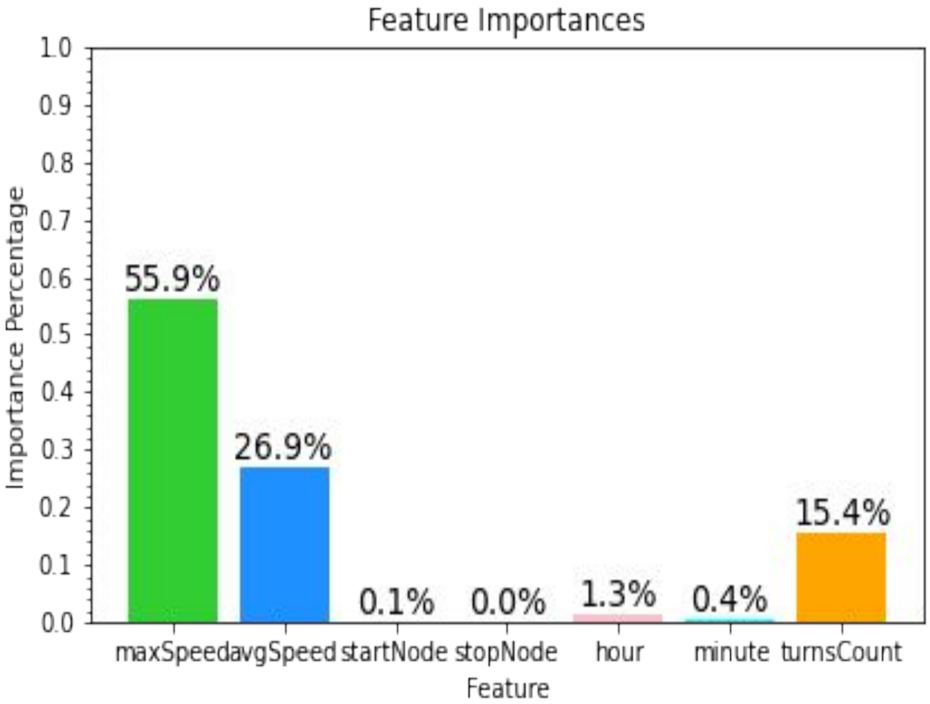
StartNode, StopNode, Hour, Minute features of Car(blue) and Bus(orange) comparison

Exploratory Model Results - Without Noise

Model	auPR(Down Sampled Data)	auPR(Raw Test Data)
Linear SVM	0.010614	0.001081
Linear Regression	0.999609	0.998593
Random Forest Classifier	1.0	1.0

- Random Forest Classifier Model works best in these 3 models.
- We will focus on seeing how Random Forest Classifier Model reacts to noise in our data in the next stage.
- We also add cross validation to do parameter tuning and avoid overfitting.

Feature Importances Visualization



- MaxSpeed, AverageSpeed and TurnsCount were relevant features
- StartNode, StopNode, Hour and Minute were not that relevant features
- Relevant features were chosen for later model training

Cross Validation Results With Noise

Model	auPR(Down Sampled Data)	auPR(Raw Test Data)	Features Included
Random Forest Classifier combined with Cross Validator	0.991220	0.919165	3 features (MaxSpeed, AverageSpeed and TurnsCount)
	0.021272	0.002245	5 features (adding startNode and stopNode features)
	0.992295	0.923198	4 features (adding hour feature)

Summary of Findings

Adding startNode and stopNode features:

- The result gets worse.
- So not adding these two features to model training.

Adding Hour feature:

- The result improves a bit.
- So final model trained with four features: MaxSpeed, AverageSpeed, TurnsCount, Hour.

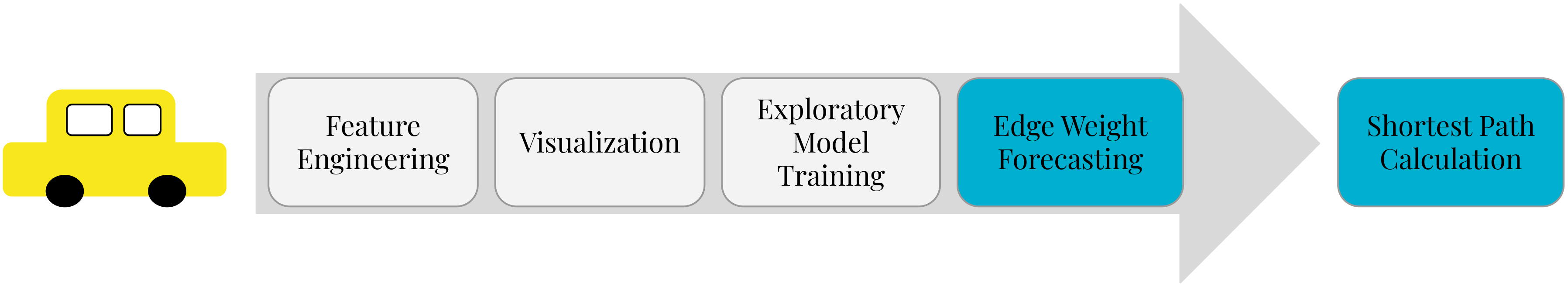
Sample Prediction Output from Test Data

```
There were 1474 buses and 655148 cars in the data set provided.
The model predicted 1595 buses and 655027 cars.
Area under precision-recall curve = 0.923198
```

Because we used auPR metric to train and the data is unbalanced, we will have slightly more false positives but with a better recall on the buses.

Edge Weight Forecasting

Shortest Path Pipeline



Feature Engineering

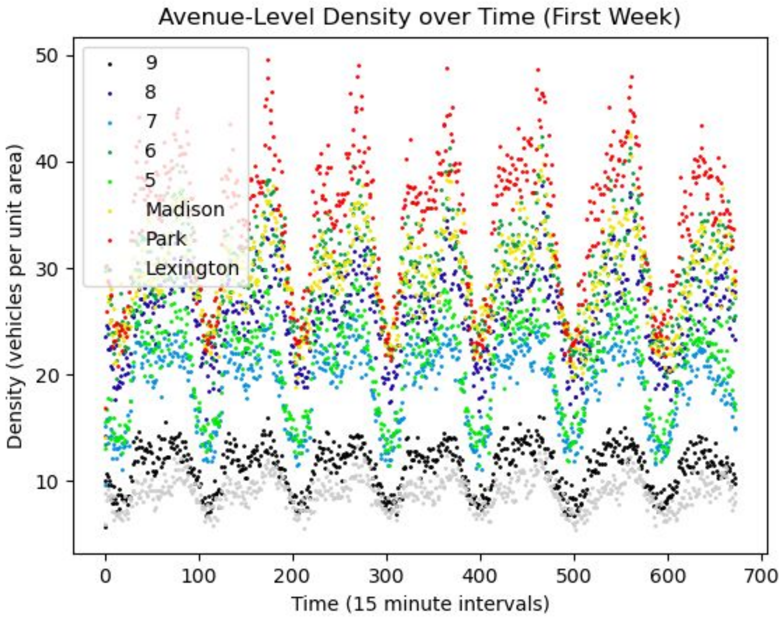
Obtained edge weight features using Spark SQL

- Customized measure of edge weight (for shortest path algorithm)
  - density: $\frac{numOf\ Vehicles \cdot vehicleSize}{totalEdgeArea}$
- Edge Features
  - density at time *t*: d_t
  - change of density: $\Delta d_{t-1,t}, \Delta d_{t-2,t}, \Delta d_{t-3,t}$
  - Time of the day as cyclical variable, formula we adapt $f(t) = \sin\left(\frac{minuteOf\ Day \cdot 2\pi}{24 \cdot 60}\right)$
- Graph Features
  - is edge two way & is edge in bus route
  - One hot encoding these two features

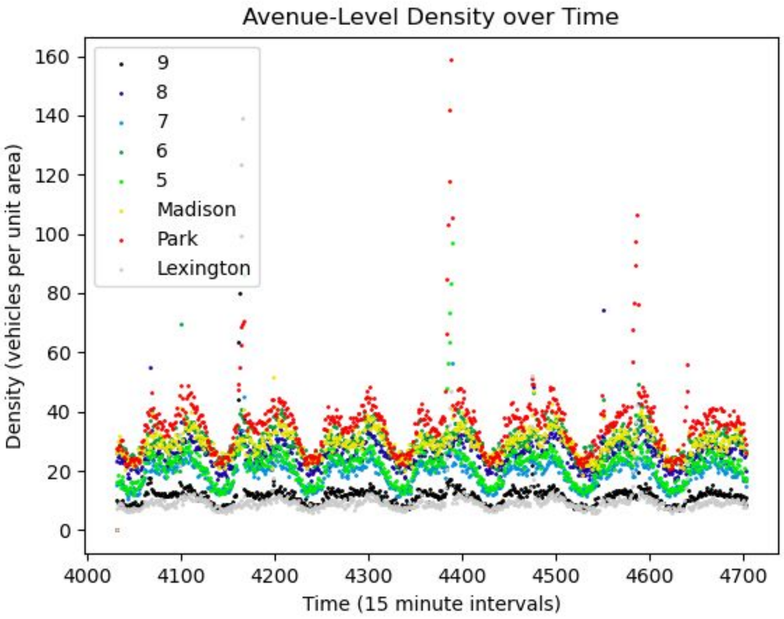
Custom train / test splits to handle time series data and data leakage

Feature Visualizations

Before Noise (1 week)



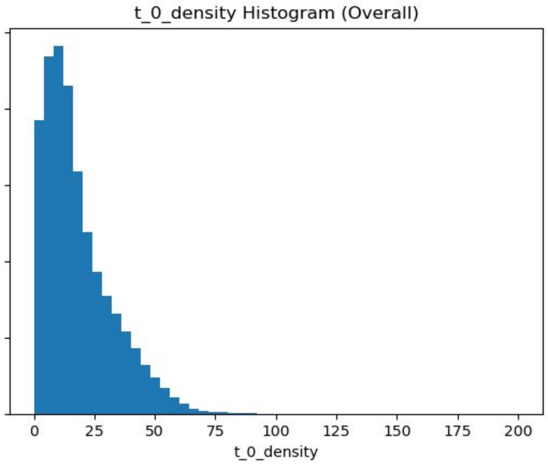
After Noise (1 week)



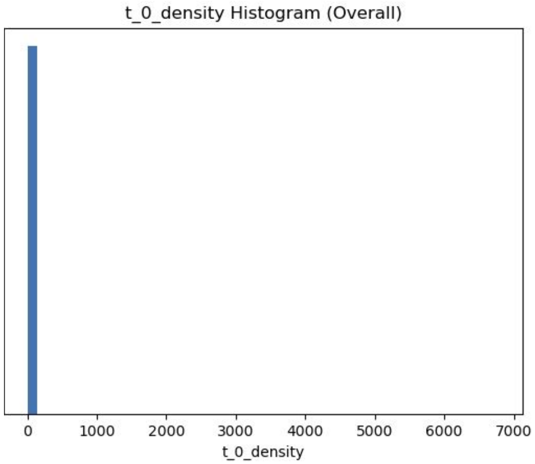
Feature Visualizations

Distributions of the Independent Variable

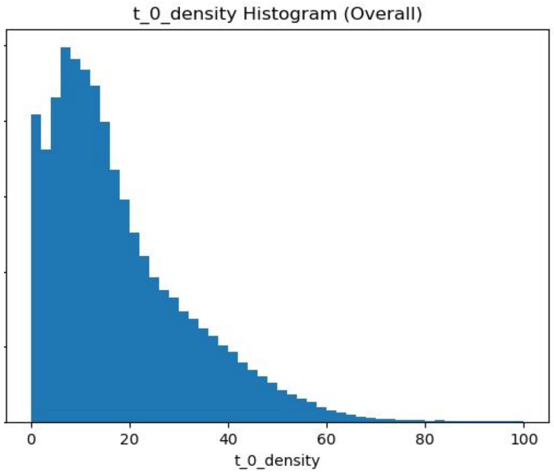
Before Noise



After Noise - All



After Noise - Truncated



Poisson Distribution

Exploratory Modeling Result

In sample data R square, use **fold zero** to calculate out sample data R square

Without Noise

R Square	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
Linear Regression	0.88 (0.87)	0.88	0.87	0.87	0.87
Generalized Linear	0.88 (0.87)	0.88	0.87	0.87	0.87
Random Forest	0.82 (0.82)	0.81	0.81	0.82	0.81
Gradient Boosted Tree	0.88 (0.87)	0.87	0.88	0.88	0.87

With Noise

R Square	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
Linear Regression	0.68 (0.83)	0.88	0.80	0.85	0.87
Generalized Linear	-117 (-77)	-23	-8	-76	-162
Random Forest	0.11 (0.06)	0.09	0.13	0.11	0.10
Gradient Boosted Tree	0.71 (0.55)	0.68	0.61	0.62	0.63

With Noise & Filtering Outlier

R Square	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
Linear Regression	0.15 (0.12)	0.61	0.25	0.36	0.38
Generalized Linear	-1.7 (-4.5)	0.12	-634	-15	-2.6
Random Forest	0.78 (0.78)	0.79	0.80	0.80	0.79
Gradient Boosted Tree	0.84 (0.83)	0.85	0.86	0.86	0.85

With Noise & Add Extra Features

R Square	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
Linear Regression	0.68 (0.83)	0.88	0.80	0.85	0.87
Generalized Linear	-119 (-79)	-32	-5	-80	-163
Random Forest	0.11 (0.07)	0.10	0.13	0.12	0.11
Gradient Boosted Tree	0.59 (0.44)	0.54	0.75	0.55	0.58

Shortest Path Algorithm

Model Final Output (linear regression)

- Generate Recommend Path at Week 9 Monday 1PM node 0-8
- True Shortest Path
  - 9_30to8_30 8_30to8_31 8_31to8_32 8_32to8_33 8_33to8_34 8_34to8_35 8_35to8_36 8_36to8_37 8_37to8_38 8_38to8_39 8_39to9_39 9_39to9_38
- Predicted Shortest Path
  - 9_30to8_30 8_30to8_31 8_31to8_32 8_32to8_33 8_33to8_34 8_34to8_35 8_35to8_36 8_36to8_37 8_37to8_38 8_38to8_39 8_39to9_39 9_39to9_38
- Source of Error
  - Mainly come from unstable edge (plot on the right)

edge	MAE	RMSE
6_42to5_42	20.46341874	33.40086947
Park_54toPark_53	16.97218526	27.37084394
Park_49toPark_50	13.77609672	24.13617799
7_42to8_42	13.99103017	19.3966747
6_49to6_50	13.06004144	18.24747077
6_34to5_34	9.993132689	16.80321039
Park_36toPark_35	10.96373559	16.75342717
5_52to5_51	10.9948562	15.8060884
Madison_47toMadison_48	9.790172074	15.68596128
Madison_57to5_57	8.016554576	12.39315383
Park_55toPark_54	7.663552521	11.92563636
Park_38toPark_39	9.33225279	11.85758805
Park_58toPark_57	8.490048711	11.19763564
Park_46toPark_47	8.33578231	11.02605693
Park_47toPark_46	7.882120061	10.9333112
5_40to5_39	7.775983013	10.81476376
Park_42toPark_43	7.847863891	10.45676273
Park_48toPark_49	7.644048104	10.02472836
Park_44toPark_45	7.315247051	9.363016631
Park_50toPark_51	7.261492887	9.348443243
8_46to8_47	6.782689898	8.937917834
7_37to7_36	6.884931109	8.80444422
7_49to7_48	6.810883706	8.704506702
8_36to8_37	6.769252371	8.450968421
Park_52toPark_53	6.679018084	8.363973546
Park_49toPark_48	6.582704206	8.33130152
Park_50toPark_49	6.097223714	7.95653521
6_36to6_37	6.018124317	7.942292519

Conclusion & Future Steps

All models perform well on no noise data

- Big challenge to make Spark work with a time series problem

After we add noise on data, the story changes...

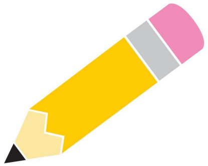
- Poisson works best for no noise data, not after we add noise and ruin the distribution
- Linear Regression is highly sensitive to the feature distribution and outliers in our noise
- Tree based model is robust for different distribution, but gets penalized by extra useless features
- **It's nearly impossible to find a magic one-size-fits-all model**

Future Steps

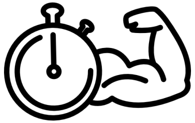
- Consider applying different model to different time segments or subgraphs
  - As long as we are clear about the noise source

Project Takeaways

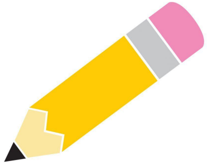
Project Takeaways



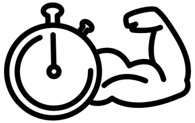
1. Container failures and data shuffle may lead to “directory already exists” error when writing data
    - Change write mode to ‘append’
2. Parquet format is better suited for Spark code than CSV format
3. Caching should only be used for high iteration access
4. Data shuffles can make performance suffer
    - Repartition before joins to avoid shuffle
    - Use map instead of repeated joins
    - Data size should be small for broadcast
    - Runtime improved from 3 days to 10 minutes



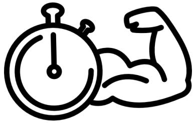
Project Takeaways



5. Optimizing spark-submit parameters can improve performance
    - Speed up joins and windowing with more executors, cores per executor, and driver/executor memory
    - For caching of large datasets, more driver memory is required



6. Window functions partitionBy and orderBy can be used to compute time-series lag features - with caveats!
7. Cross-validation runs sequentially by default



8. Spark error detection does not catch everything
    - Model trained on infinity values due to floating point error without complaints from Spark
    - Spark ran GraphX code with negative cycles, repeatedly self-healing failed tasks which ultimately could have overloaded the cluster

Demo Instructions

Demo Instructions from README.md

BDAD_Violet_Noise

Team Members

- Ameya - as12366
- Kristin - jl11257
- Emma - yl3750
- Helene - hls327

Running the Demo

Data is designed to be simulated one day at a time, each output from the microsimulation is an ~8GB file. To save time, several pre-processing steps of the data pipeline have already been run so that you can test on a small set of data. Steps completed include generating a new day of data, parsing into parquet format, adding noise, and selecting a subset to test. See the following scripts for details:

`runscripts/simulateForEval.sh`

`runscripts/prepforEval.sh`

You are testing on ~2 hours of data, a Monday morning between 7 and 9 AM.

The data is stored in `/user/jl11257/big_data_project/traces/demo/morningsample`

Please Note: The zip file must be unpacked in the top level of your user's /home/ directory on PEEL. If it is not unpacked at the top level directory, the demo will not work.

To run the demo, simply run `source runscripts/demo.sh` in your home user directory.

If you wish to re-run the demo, you will want to run the following command first.

```
hdfs dfs -rm -r /user/$(whoami)/violetnoisesummary
```

In the event of folder permission issues please contact hls327@nyu.edu.

Noise and Demo Data

Traffic accidents are simulated to occur between 7 and 10 AM and 2 and 7 PM

- In our Demo data set, we have two accidents in the morning
- We've selected the hours between 7 and 9 AM - which is the noisiest time of day
- Model performance in the demo data is considered to be the worst case scenario
- In addition, data is freshly simulated - it has not been used in any model training

Expected Demo Output

```
####################################
Team Violet Noise
Team Members as12366 yl3750 jl11257 hls327
Traffic Simulation
Demo Run
####################################
   -
               __
 --          ~( @\   \
---   _________]_[__/_>________
     /  ____ \ <>     |  ____  \
    =\_/ __ \_\_______|_/ __ \__D
________(__)_____________(__)____
####################################


Checking sufficient permissions...
Input data permissions are OK
Model permissions are OK
Making local hdfs directories for demo output
Running feature calculations for the vehicle classification model, please allow 1-2 mins
Vehicle feature spark log is in vehiclefeaturelog.txt
0        0        /user/hls327/violetnoisesummary/vehiclefeatures/_SUCCESS
101.5 K  304.5 K  /user/hls327/violetnoisesummary/vehiclefeatures/part-00000-cefb2c02-9ebe-447a-8e29-03e5e6f5fcde-c000.snappy.parquet
100.3 K  300.9 K  /user/hls327/violetnoisesummary/vehiclefeatures/part-00001-cefb2c02-9ebe-447a-8e29-03e5e6f5fcde-c000.snappy.parquet
Running feature calculations for the edge weight regression model, please allow 1-2 mins
Edge weight feature spark log is in edgefeaturelog.txt
0       0        /user/hls327/violetnoisesummary/edgefeatures/_SUCCESS
43.8 K  131.4 K  /user/hls327/violetnoisesummary/edgefeatures/part-00000-e654a7b5-d22b-4d28-9aa4-fbac5041637b-c000.snappy.parquet
36.0 K  108.0 K  /user/hls327/violetnoisesummary/edgefeatures/part-00001-e654a7b5-d22b-4d28-9aa4-fbac5041637b-c000.snappy.parquet
38.8 K  116.5 K  /user/hls327/violetnoisesummary/edgefeatures/part-00002-e654a7b5-d22b-4d28-9aa4-fbac5041637b-c000.snappy.parquet
41.7 K  125.2 K  /user/hls327/violetnoisesummary/edgefeatures/part-00003-e654a7b5-d22b-4d28-9aa4-fbac5041637b-c000.snappy.parquet
43.4 K  130.1 K  /user/hls327/violetnoisesummary/edgefeatures/part-00004-e654a7b5-d22b-4d28-9aa4-fbac5041637b-c000.snappy.parquet
43.3 K  130.0 K  /user/hls327/violetnoisesummary/edgefeatures/part-00005-e654a7b5-d22b-4d28-9aa4-fbac5041637b-c000.snappy.parquet
Running car classification predictions on sample data, please allow 1-2 mins
Car prediction spark log is in carclassifylog.txt
There were 48 buses and 12190 cars in the data set provided

The model predicted 147 buses and 12091 cars

Confusion matrix:
12088.0  102.0
3.0      45.0

Area under precision-recall curve = 0.29667869149620957
```

Expected Demo Output

```
Running shortest path forecast on sample data, please allow 2-3 mins
Edge forecast spark log is in edgeforecast.txt
Root Mean Squared Error (RMSE) on out sample test data = 17.77699543550038
Mean squared error (MSE) on out sample test data = 316.02156671380123
Regression through the origin(R2) on out sample test data = -0.5941405438580856
Mean absolute error (MAE) on out sample test data = 9.688900768369908

True Shortest Path

9_30to8_30 8_30to7_30 7_30to6_30 6_30to5_30 5_30toMadison_30 Madison_30toPark_30 Park_30toLexington_30 Lexington_30toLexington_31 Lexington_31toL
xington_32 Lexington_32toLexington_33 Lexington_33toLexington_34 Lexington_34toLexington_35 Lexington_35toLexington_36 Lexington_36toLexington_37
Lexington_37toLexington_38 Lexington_38toLexington_39 Lexington_39toPark_39 Park_39toMadison_39 Madison_39to5_39 5_39to6_39 6_39to7_39 7_39to8_39
8_39to9_39 9_39to9_38
143
Predicted Shortest Path

9_30to8_30 8_30to8_31 8_31to8_32 8_32to8_33 8_33to8_34 8_34to8_35 8_35to8_36 8_36to8_37 8_37to8_38 8_38to8_39 8_39to9_39 9_39to9_38
201


               ,___,
             .'     '.
            | STOP  |
             '.___.'
                ||
                ||
                ||
              \||///
          ^^^^^^^^^^^^^^^

        Demo is complete!
####################################
```

Thank You!