McGill | School of Continuing Studies | École d'éducation permanente

DATA SCIENCE AND MACHINE LEARNING

CAPSTONE PROJECT

Emergency Call Centre Human Resource Management Based on Hourly Fire Department Intervention Data

TEAM: KERAS THE PYTHON

Obaro Agbanagba

Mustafizur Bhuiyan

Razvan Lungu

Tashlin Reddy

Jun Shao

Professors:

Nabil Beitinjaneh

Alejandro Gutierrez Lopez

Nicolas Feller

Due Date: **Monday March 18, 2019**

Table of Contents

1. Objective: 1

2. Introduction 2

3. Literature Review 2

4. Project Frame 5

4.1 Risks and Constraints 5

4.2 Stakeholders 6

4.3 Data Process 6

4.4 Project Management 7

4.4.1 Work Breakdown Structure 7

4.4.2 Project Scheduling 8

5. Data Analysis 10

5.1 Data Modelling 10

5.2 Data Prep and Augmentation 11

5.3 Initial Findings 11

5.4 Correlations 15

6. Discussions: Predictions and Results 18

6.1 Multivariate Time Series Forecasting with LSTMs in Keras 18

6.2 Linear Regression with Keras 22

6.3 Facebook Prophet 24

7. Conclusion 28

8. References 30

9. Appendix 31

A1. LSTM model 31

A2. MLP model 38

A3. Facebook Prophet Model 41

A4. Linear regression model and fully-connected model 46

1. Objective:

This report will discuss the various data analysis tools and prediction models that were processed in order to predict the number of interventions in a given hour of Montreal. Given this

prediction, emergency call centres can efficiently manage their scheduling and employ either part time or full time employees for specific range of work hours.

2. Introduction

The service de Securite Incendie de Montreal (SIM) is responsible for fire and rescue operations in Montreal, Quebec. The department offers High Angle Rescue, Collapse, Rescue, HazMat Response, Ice Rescue, and Nautical Rescue since 1863. The SIM agency however was established in 2002 and it currently employs about 2700 firemen and receives about 132 000 calls annually. The calls are mainly categorized as first respondent, fire, structural, or naval incidents. The human resources of Montreal's emergency call centre currently requires information on the hourly traffic of calls in order to determine how many employees they must employ for a given range of hours. Their human resource management will be firmly based on only the fire department calls and interventions.

Interestingly, the City of Montreal webportal hosts the firefighter datasets, which includes the interventions, the type of interventions, and fire stations since 2005, falls The dataset is offered in csv format, which makes using the pandas function 'read_csv' a simple tool to create a dimensional labelled array for them.

Moreover, the fire department datasets that were imported from the City of Montreal portal includes dictionaries labelled as type of intervention, date, barrack, location, and number of units deployed. Initially, the datasets were analysed for their content and the dictionaries were compared for correlation. Detailed information on its analysis and conclusions are further discussed in proceeding sections of this report. Regarding our particular fire department datasets, there has been few data science processes discussed by several online articles. These articles, which shared similar analysis and predictions on the datasets, were reviewed with the attempt of conducting literature reviews and acquiring further insights for creating our data science models.

3. Literature Review

We have reviewed many articles and other forms of literature to gather many perspectives when handling fire department datasets. When reviewing 'Predicting Emergency

Incidents in San Diego" article, the authors were trying to predict the allocation of resources in order to save more lives. San Diego has a reputation of generating fire department response, just in year 2015 the city had 1.3M incidents. The incidents caused roughly 15,000 injuries and $14.3B of property damages. Such numbers created a requirement for better fire department resource allocation.

The paper describes the students attempt to use historic data that lists emergency incidents in the San Diego region, and use the data to predict future incidents of various types. Types may included first response, fires, rescues, etc. Their initial processing involved using the Google API to transform street addresses in latitude and longitude coordinates. We on the other hand, will be interested in doing the inverse, since our data already provides these coordinates. The coordinates will be transformed into postal code areas.

Furthermore, the Stanford students used Decision Tree regressions, iterative clustering, and Neural Networks to create a prediction model. Although they proved to produce sufficient predictions on the data, they felt that additional census data needed to be merged to the initial dataset in order to augment its quality. Below is a table of prediction models that was tried in their research. The authors claim that Neural Network with clustering produced the least amount of root mean squared error, 1.977 (incidents/hour). Regarding this, keras can be a potential framework to effectively digest the test/train data.

Experimental Model: (20 x 20 sized regions)	**RMSE:** (incidents/hour)
Neural Nets (no clustering)	3.161
Max Depth DTR	2.013
Iterative Clustering with Neural Nets	1.977

Table 1: RMSE for Prediction Models

Another article, hosted in Medium.com, Cyril Pecararo presents a data science framework to predict and evaluate the turnout time and travel time of the SIM dataset. Due to the need of a prediction model involving consistent data, he explains finding it difficult to clean the irregularities in the initial data exploration phase of filtering 800,000 entries. Irregularities were classified as manual operations such as an officer pressing a button located in the vehicle to signal his departure or arrival. XGBoost was used to predict the turnout time using features such

as day, season, type of unit, type of incident and information about the fire station of origin. The article describes that XGBoost generates a smaller mean absolute error when comparing to a baseline for predicting turnout times. This article was very interesting as it used the same dataset as ours while attempting to predict the response times of firefighters.

Response time is very critical for firefighters and for those in need. That is why the Amsterdam Fire Service (AFS) is implementing the use of twitter to communicate the incidents in real time and send operational codes to all personnel. Forbes released an article that describes this idea of sharing data and analytics to generate a community of faster responding firefighters. As much of the response time is used to prepare for the incidents, such as hostlering the right equipment and bringing the right engine, knowing the risk profile in advance can be very advantageous. The article discusses having a predictive model to predict the risk profile of the area of the incident. This model could be based on previous intervention data in the target region.

A lot of what we are going to try to predict will be more or less similar to these discussed articles as we will also be modelling with previous intervention data. With the help of incident type, target area, requested barrack, and additional census data, we should have a well rounded model for our fire department dataset. Below is an example of Uber's prediction model for identifying the estimated time of arrival once a request for an uber was made.

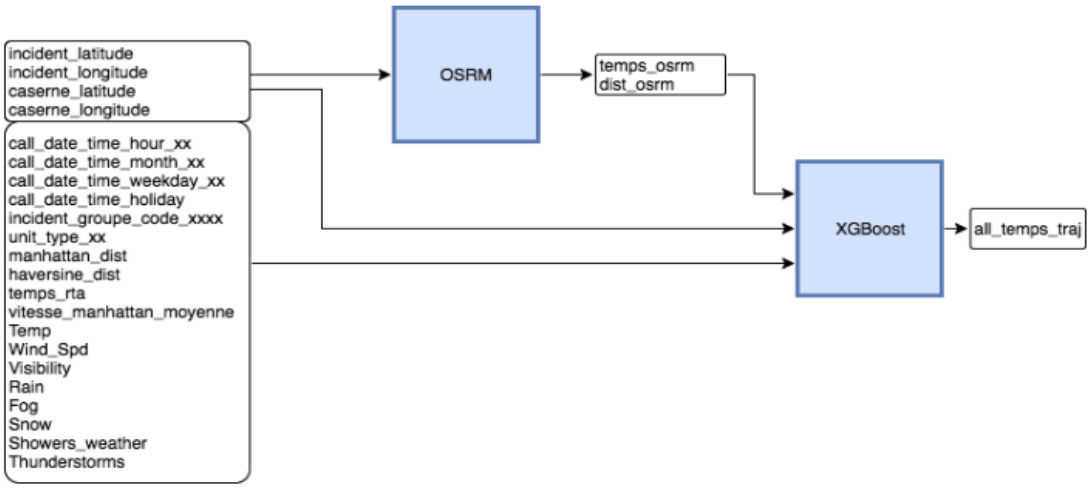


Figure 1: Uber's XGBoost Model

4. Project Frame

Analysing data requires identifying several requirements, starting from loading the data to producing conclusive results. Requirements can include addressing questions such as: “Why are we analysing the data?” “Who will benefit from the results?” and many others. This section will provide the details of the project frame when analysing the fire department data. The project frame of any project should include identifying the risk, constraints, stakeholders, timeline, and budget.

4.1 Risks and Constraints

Risks and constraints play an influential factor for any project. Surpassing risk may jeopardise the completion of the project, while not maintaining within constraints may require overtime or even over budgeting.

Time is a significant constraint for our project because we are given a solid timeframe to complete the analysis and produce conclusive results. In effect we are given 3 months, which is the duration of the Capstone course to analyze and make conclusive predictions of the fire department data. Additionally we were also granted accounts to the Google Cloud Platform and DataCamp for the duration of 365 days and 6 months respectively.

Cost is another significant constraint for a project. Our cost is very minimal because it is an academic project, but the academic institution of McGill University has invested large amount of money to provide the students with various licenses: Slack and Google Cloud Platform (GCP). In terms of using the GCP to conduct our project, we are constrained to 50$ worth of credits each. The credits can be used for the various services offered on the GCP.

Quality and quantity are also factors, which can be seen as risks and constraints to our project. The quality of data is important when we will be analysing and producing conclusive results. For example, a dataset without various attributes may not provide much information nor allow for much classification. Our datasets include dictionaries comprised of type, location, category of severity, date, and time for approximately 1.2 million incidents from 2005 to today. In terms of quantity, the data sets have various attributes for us to consider: population density, population, type of intervention, units deployed, barrack id, location, and date.

4.2 Stakeholders

The entities that will be interested in the analysis and results of the data are seen to be as stakeholders. The Ville de Montreal, the SIM, or even any research council can benefit from the results of our data analysis and prediction. The city of Montreal should be interested in knowing how to manage the emergency call centre employee schedules, while the SIM would directly interested in knowing the number of fire department calls for a specific hour.

4.3 Data Process

For our project, we will use the cross industry standard process for data mining (CRISP-DM). The process involves six phases: Business understanding, data understanding, data preparation, modelling, evaluation, and deployment. Each phase is named explicitly to its function along the project cycle. Based on many researches, the CRISP-DM process has been widely used for many data mining models. We will further discuss the implementation of the model and its phases during our project analysis section of this report. The cycle of the CRISP-DM process is displayed below.

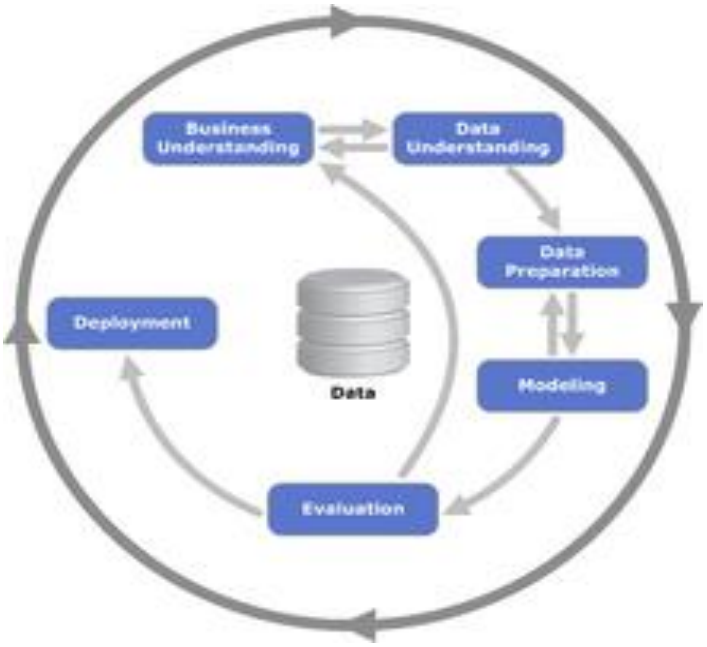


Figure 2: CRISP-DM

4.4 Project Management

Every project must be managed in an efficient type of way in order to meet team member availabilities, deadlines, and resources. The sections below will describe the concepts and tools used in order to organize the project and produce deliverables.

4.4.1 Work Breakdown Structure

During the first class of the Capstone project course, we were assigned individual roles as an identity. The purpose of roles assignment was to allow formation of teams. Each one of our members were given a distinct identity, but we all contributed to every topic of the project. Each teammate helped capture business requirements, analyse data, program prediction models, and prepared presentations. Below is an image of the Lord of the Rings based roles that was given to each individual for team formation.

Figure 3: Team Roles

Considering the work breakdown structure (WBS), the chart below displays the various tasks for the project timeframe. The WBS incorporates the CRISP-DM model, which has highlighted phases in orange. Once the structure and data project model was outlined and the tasks were understood, a project schedule was created.

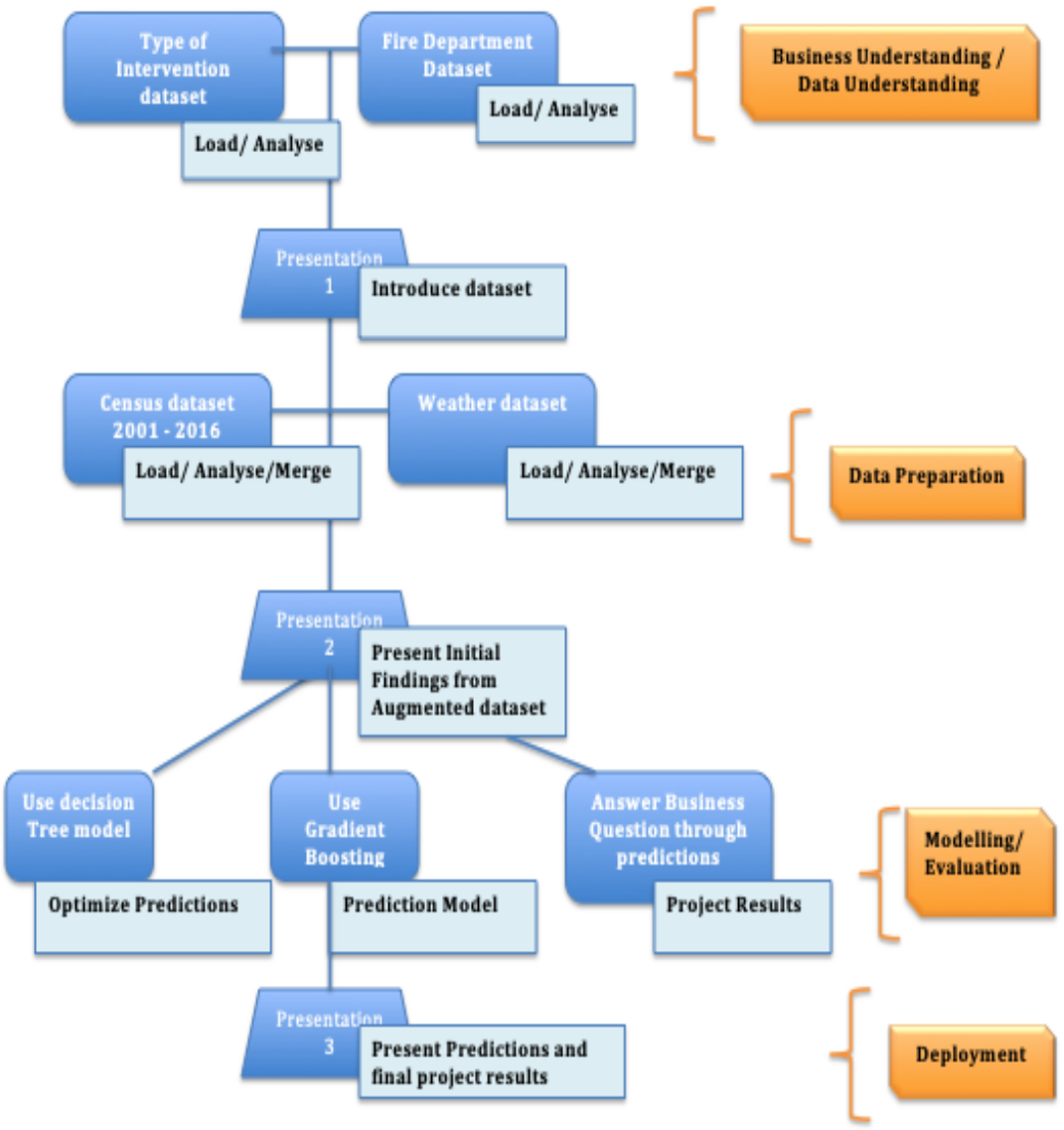


Figure 4: Work Breakdown Structure

4.4.2 Project Scheduling

Every project has a schedule that must be followed. Schedules, include assigned tasks, deadlines, and future objectives. A schedule helps a project stay within the constraints of time. We have used a Gantt chart to keep record of our deadlines and project timeline. The chart is divided into three major dividends, the first, second, and third presentations. Before the first presentation, our team has explored the data and discussed initial impressions. Up to the second the presentation, we had prepared our initial findings from analysing the data. By the end of the third presentation, we have prepared our models and predictions based on the data. Below is the Gantt chart that was prepared to gain traceability of our project timeline.

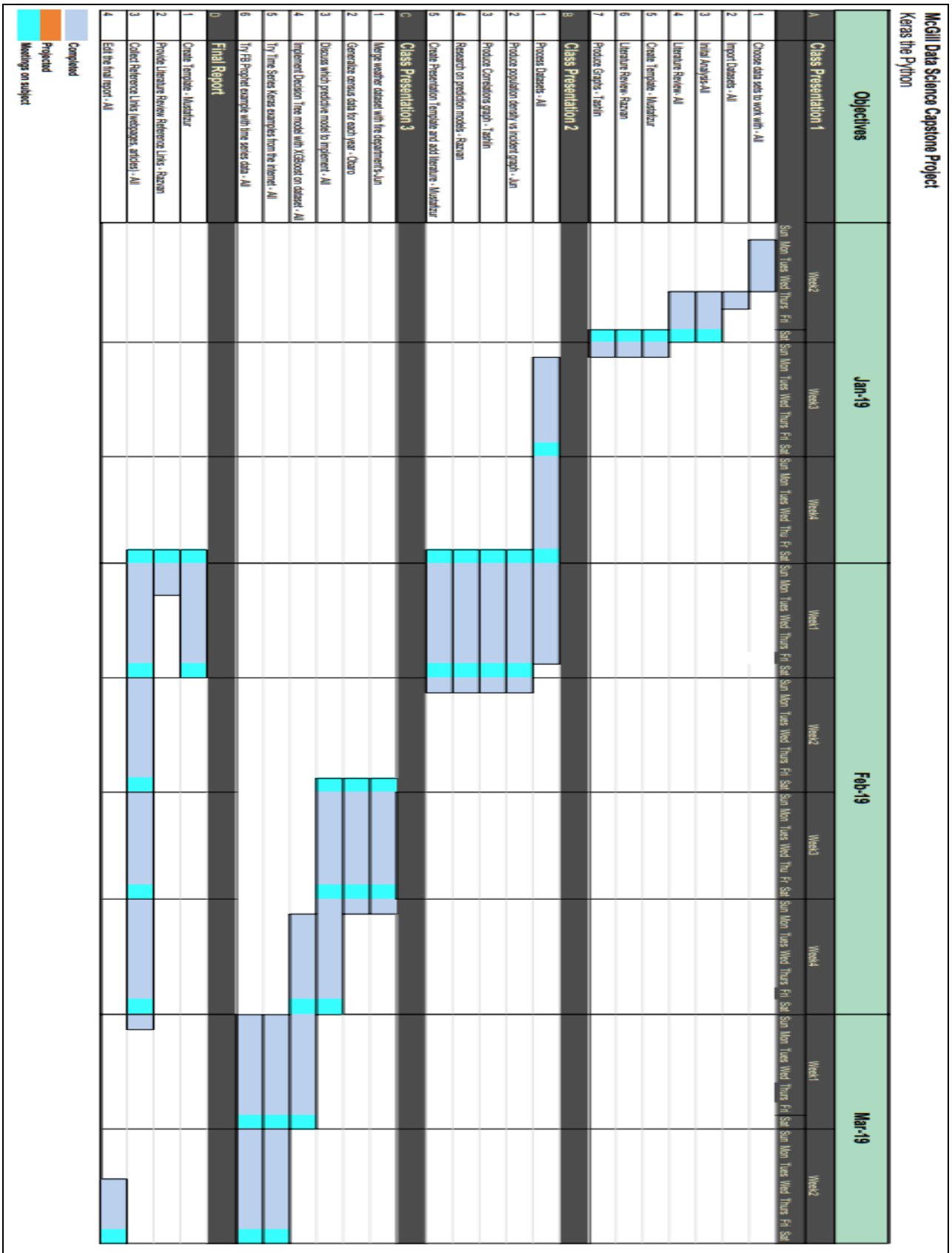


Figure 5: Gantt Chart

5. Data Analysis

This section will detail the data analysis section of the project. Analysis was performed with the use of GCP, Tableau, and Python.

5.1 Data Modelling

There are various datasets available online that may be useful to our project. for our case, we will use the Census 2011/2016, weather, and fire department datasets to acquire sufficient data. The phases of data modelling in order to eventually achieve conclusive results will include the conceptual, logical, and physical phase. The conceptual phase relates to defining stakeholders, clarifying what we are looking for, cleaning the data sets, and producing initial findings from them. Our initial findings focus mainly on the fire department dataset and how each feature can be elaborated to deliver important information for the project.

The logical phase will include the analysis portion of the data. It will allow us to understand what the data is trying to explain and how various features are correlated to each other. Before making any conclusive results or predictions, a thorough analysis of the data should be performed.

Finally, the physical phase will include the results and predictions that will be obtained from the analysis. Such conclusions will allow us to deliver concrete information to the stakeholders in order to influence their decisions on fire department resource allocation.

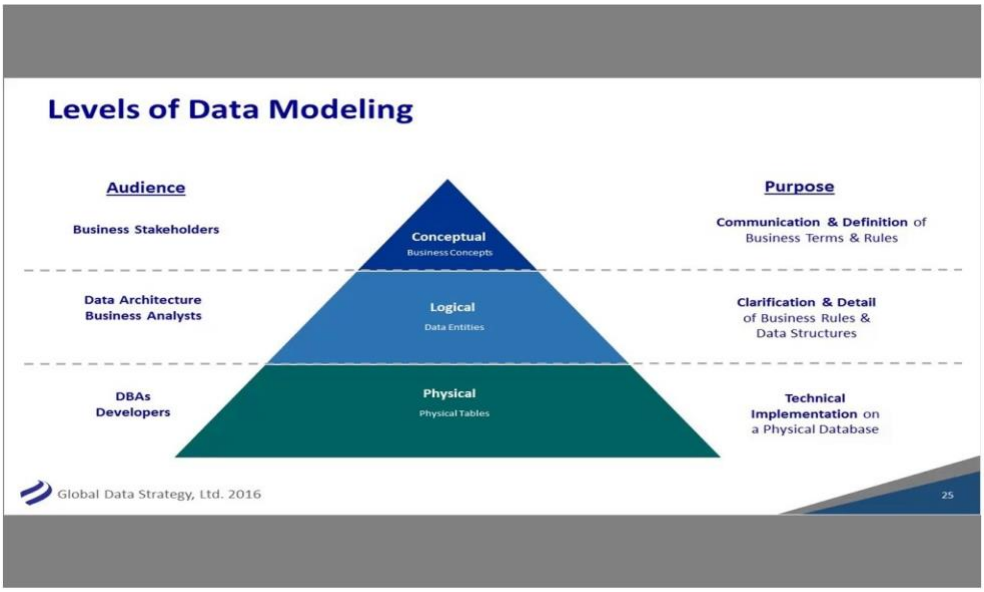


Figure 6: Data Model Pyramid

5.2 Data Prep and Augmentation

The data prep of the initial fire intervention data involved primarily using the pandas library in python. We first found how many NaN values there were, since there were roughly 500 out 1.3 million, we decided to drop those rows. The rest of the dataset was quite clean.

Next, we augmented the data with census data from 2016,2011, and 2006. Merging the census dataset with the intervention dataset based on the ville. Unfortunately, zoning changes throughout the years made this a bit more difficult. To solve this issue, we used the lower common denominator, for example, we consider Ahunstic/Cartierville one area. The other major issue was that census data is only collected every 5 years. We wanted yearly data. To solve this issue we used the "pandas.DataFrame.interpolate()" method. Basically, it created a linear function between two points and interpolated the data. In this case the population was simulated.

Finally, we used hourly weather data from Government of Canada portal to augment our original dataset. This weather encompassed the whole city of Montreal. We grouped the number of interventions per hour per area. This allowed us to merge the weather to our dataset. It is understood that a dataset with various attributes can offer extensive classifications and modelling for analysis. Our prepared dataset resulted in approximately, 90 000 hours for each area, about 2.6 million data points in total.

5.3 Initial Findings

We produced several graphs that will be used to elaborate on the initial findings from the datasets. Firstly, we produced the map of Montreal through Tableau. The map displays the different boroughs and their colour code. This was achieved by using the longitude and latitude information of each borough from the dataset.

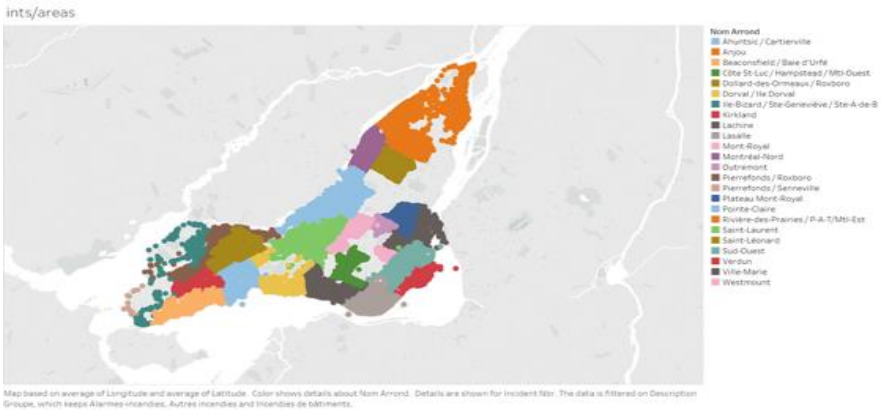


Figure 7: Montreal Boroughs

Each borough has its associative barrack and each barrack is seen to have recorded their previous interventions. The graph below displays the various barracks across the map of Montreal and their number of incidents. It is noted that that a borough can have multiple barracks. Based on the graph below and the previous map, we can say that Verdun's barrack 65, and Ville Marie's barrack 30, seem to be very occupied with interventions.

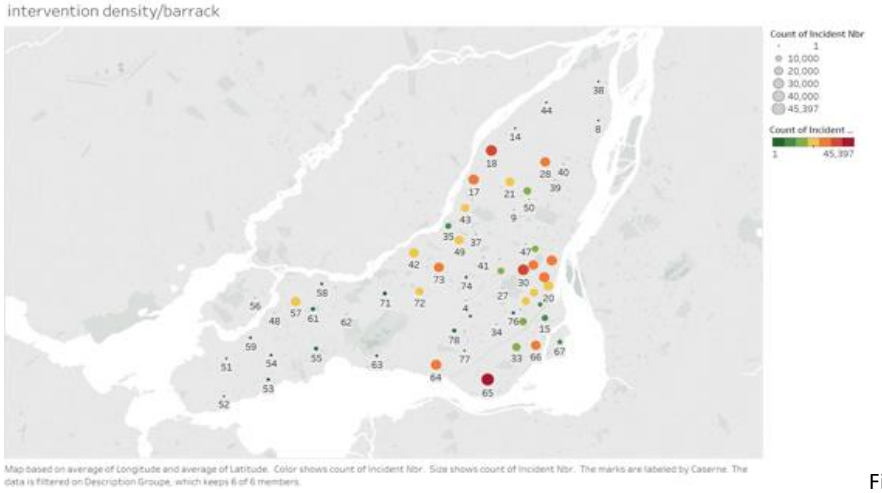


Figure 8: Intervention/ barrack

The fire department dataset further included the date of each intervention for the last 14 years (2003-2017) The rate of increase of incidents throughout the years are displayed in the line graph below. The trend line is an estimate of the actual plotted line. As seen from the plot, there is a significant rise in number of incidents between 2007 and 2010.

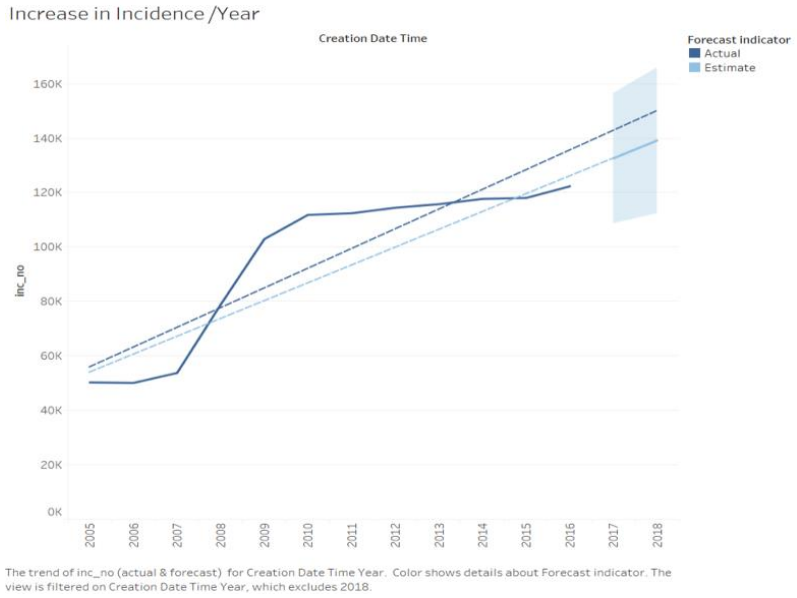


Figure 9: Increase in incidents/year

However, the above graph shows the cumulative incidents without diversifying. The incidents actually have a type, whether it is first respondent, false alarm, fire, structural, etc. The next graph will clearly show why there was an increase in interventions; it is because the fire department have started answering as first respondents for all dispatched incidents since 2007. The graph below displays specifically the month of May through the years and shows the difference in types of interventions. First respondents is the primary type of intervention since 2007.

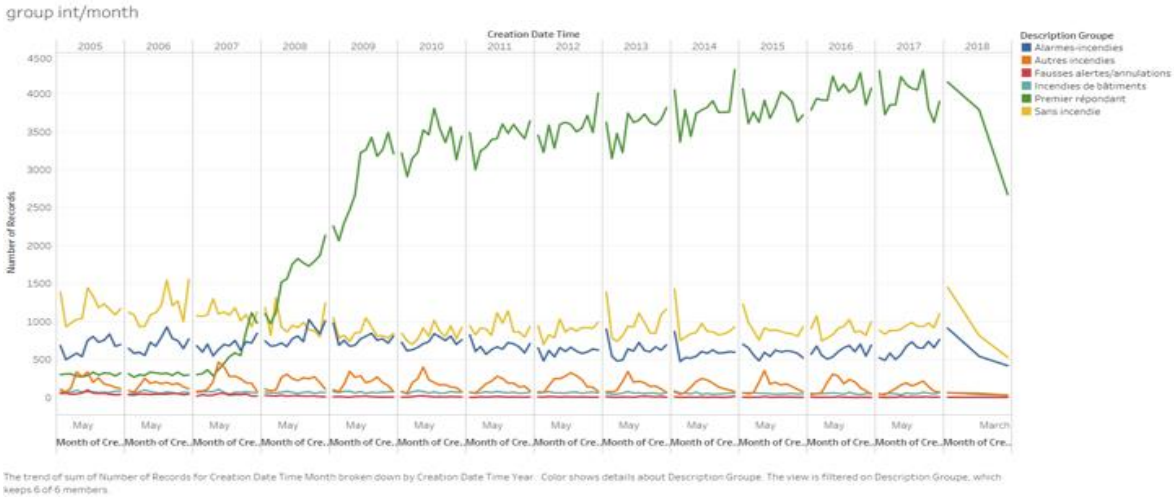


Figure 10: intervention/month

Since the above month displayed only the month of May for various years, we were also interested in finding out the trend of incidents depending on the various months of the year. The graph below displays the number of incidents for the months of 2011. As we assumed, the summer and winter months show a significant rise in incidents throughout the year.

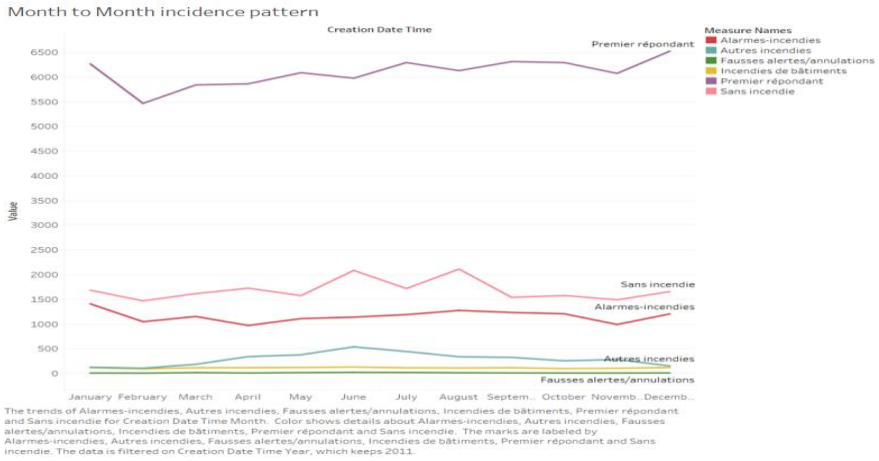


Figure 11: Monthly incident pattern

Additionally, in order to dig deeper to find out which days of the week would be more susceptible, we produced the graph below. Interestingly it shows that Friday is the popular day for fire department incidents. The graph displays interventions for days of the week between 2005 - 2018. The percentage breakdown of the different types of interventions between the 13 years is displayed below it. First responder interventions equal to about 58% of the total interventions, while interventions related to fires (fire alarm, building fires, other fires) amount to 19.65%

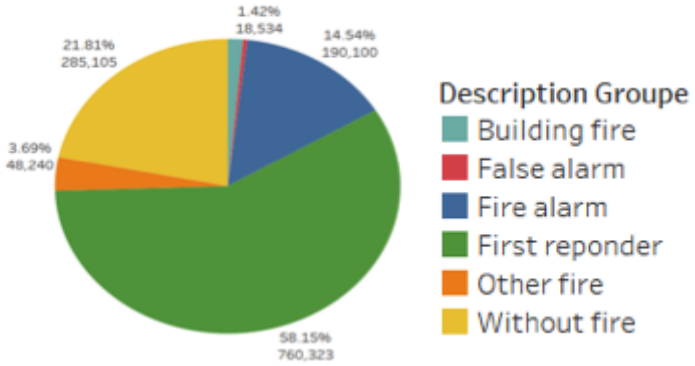


Figure 12: Pie chart of intervention type

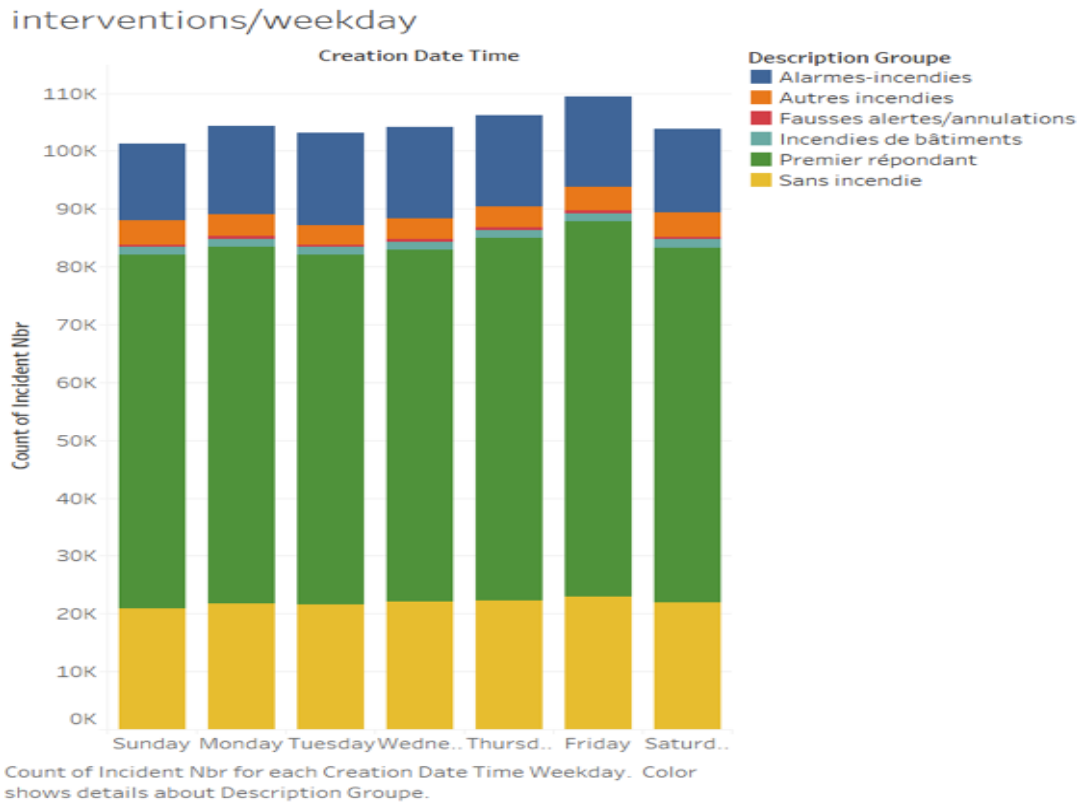


Figure 13: Intervention per Weekday

To further expand on the weekday graph above, we scattered the count of incidents throughout the hours of a given day. Interestingly, we noticed that the most amount of interventions (> 1400) happen between 15h and 16h. Specifically, Friday has history of hosting the most number of interventions during 17h.

Description Groupe / Creation Date Time
Without fire

Count of Incident ..
218 1,583

Hour of C..	Sunday	Mond..	Tuesd..	Wedn..	Thurs..	Friday	Satur..
0	735	469	476	557	506	538	710
1	562	394	343	395	357	429	558
2	436	327	260	319	323	327	449
3	507	272	254	266	301	307	486
4	366	229	218	238	252	260	375
5	308	266	257	253	271	306	289
6	349	411	423	420	411	436	360
7	386	724	665	728	738	721	472
8	513	931	923	917	974	969	674
9	705	1,008	1,004	1,098	1,040	925	831
10	919	1,072	1,030	1,137	1,181	1,078	1,013
11	1,028	1,137	1,183	1,164	1,193	1,151	1,144
12	1,184	1,119	1,135	1,192	1,170	1,235	1,175
13	1,206	1,161	1,156	1,231	1,220	1,333	1,222
14	1,303	1,215	1,165	1,231	1,279	1,292	1,306
15	1,362	1,258	1,263	1,285	1,377	1,426	1,310
16	1,323	1,443	1,321	1,391	1,421	1,533	1,341
17	1,337	1,556	1,526	1,583	1,572	1,547	1,399
18	1,388	1,515	1,542	1,447	1,555	1,488	1,374
19	1,307	1,339	1,340	1,351	1,252	1,385	1,296
20	1,123	1,213	1,248	1,168	1,155	1,309	1,235
21	1,037	1,048	1,138	1,059	1,077	1,111	1,078
22	859	899	935	923	913	1,051	950
23	671	693	769	685	767	860	830

Figure 13: Intervention per hour

5.4 Correlations

Once the fire department dataset was thoroughly addressed, the census, and weather data was merged in order augment the dataset. Features such as population density, total population, income, age, temperature, humidity, and dew point shall be used for the dataset analysis. Initially, the census data was used to find correlations with the fire department dataset. After running a Pearson correlation, which provides a linear relationship between two variables, we noticed that income and interventions are negatively related. Population and interventions are positively related. Areas with low income are more susceptible to having interventions, this may be due to lower education or reduced living conditions. And areas where population is large, have larger number of interventions. Furthermore, population of an area can also be seen as positively related to interventions; areas that have more people - will likely produce more interventions in a given time. The Pearson correlation graph below displays our analysis stated above.

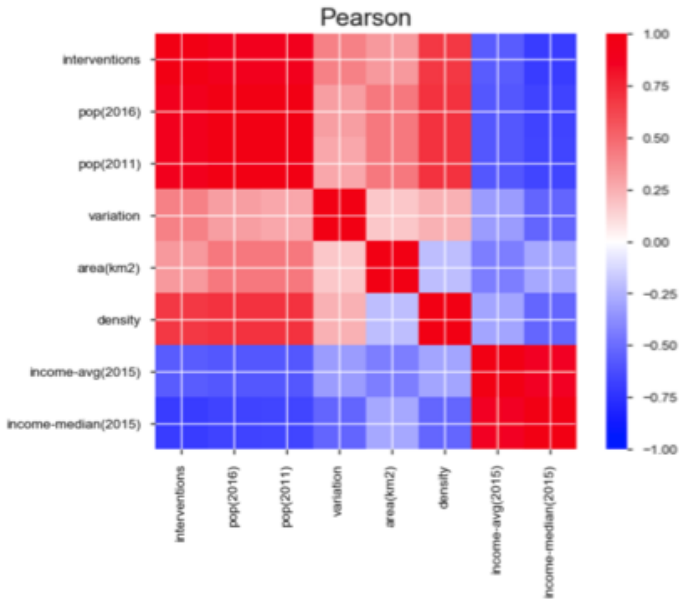


Figure 14: Correlation Chart

The population and its density are important factors when analysing the number of interventions. We believe that there is definitely a relation between the population density and number of interventions, but from the bubble chart below, it suggests that it is not quite so. Population density is not so related to number of incidents. Ville marie, having less population density than Plateau Mont Royal, has more number of interventions. The same analysis is true for Ahuntsic, which has less population density but more number of incidents than Plateau Mont Royal. Maybe population, instead of population density is a better predictor for number of interventions.

Figure 15: Bubble Chart of intervention/population density

The histogram below further elaborates on the bubble chart and demonstrates a distributed visual for each borough. We can start to assume that population, instead of population density is a better predictor for number of interventions. Ville Marie and Ahuntsic host the most number of interventions because of their larger populations. Plateau has the largest population density but does not produce the most number of fire department interventions.

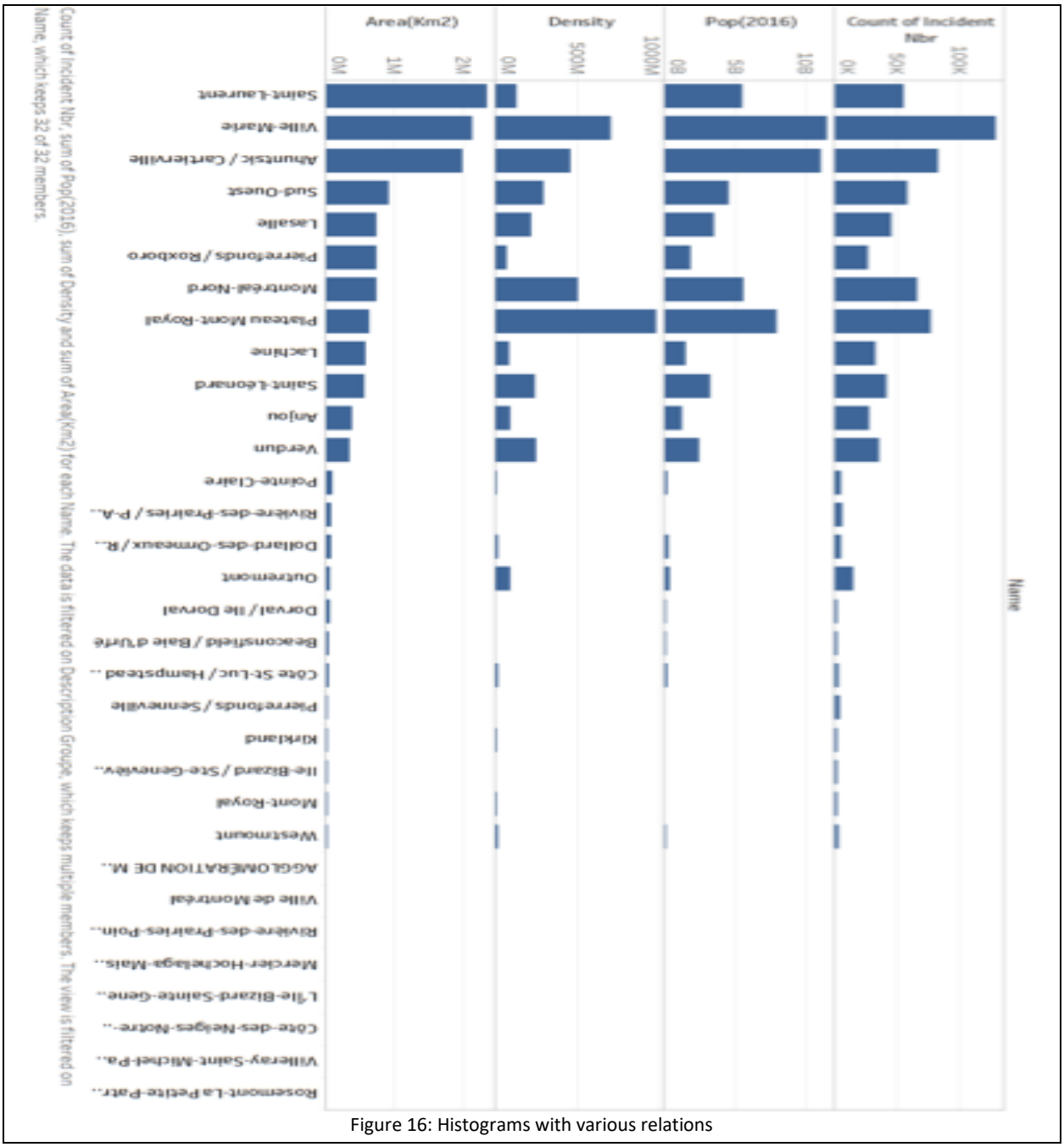


Figure 16: Histograms with various relations

6. Discussions: Predictions and Results

Machine Learning will be applied in order to train and test the dataset and make predictions based on the inputs. We will be trying to predict the number of interventions per any given hour. For comparisons, we have programmed three different predictor models. LSTM with Keras, MLP with Keras, Linear Regression with Keras, and Facebook Prophet. Models can also be further fine tuned for optimization, fine tuning can include adding layers, gradients, and configuring optimizer parameters.

6.1 Multivariate Time Series Forecasting with LSTMs in Keras

Initially, deep neural networks was applied using the Keras library and cascaded input layers. Each layer used the outputs of the previous layers as inputs until the final layer. The Keras model used Sequence() to initiate the layer modelling. Once the model was defined, it can be compiled with the use of the loss factor and optimizer for each layer. Finally, the compiled model was fitted and predictions were made. The diagram below displays the workflow of a general keras neural network.

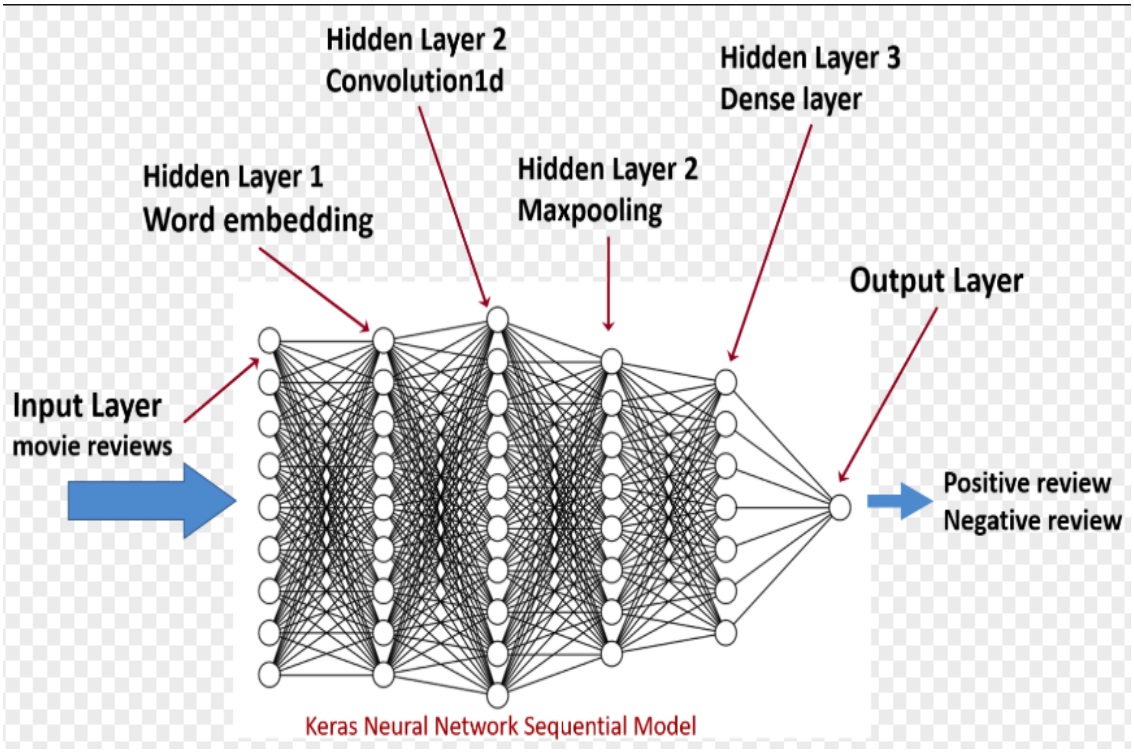


Figure 17: Neural Network

Since our dataset is of time series category, research shows that multivariate LSTM based neural network should be a good choice for forecasting - at least as a baseline. With our multivariable dataset, we can arrange our dataset to gain a prediction of the number of interventions within a given hour. The first step of creating the LSTM model for our dataset involved transforming it into a supervised learning problem and normalizing the input variables. Below is the code snippet for defining the function to transforming the dataset into a supervised learning problem.

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
  # put it all together
  agg = pd.concat(cols, axis=1)
  agg.columns = names
  # drop rows with NaN values
  if dropnan:
    agg.dropna(inplace=True)
  return agg
```

Figure 18: series to supervised

The above function definition takes the indexed time series data and transforms it into x and y variables in order allow output variables to be predicted by the input variables. Once we execute the above function we are able to display the transformed dataset, there are 16 input series variables and 1 output variable (intervention for a given hour var16(t)). Below is the snippet of the code and the printed input & output variables.

```
from sklearn import preprocessing as pp

# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = pp.MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)

#drop columns we don't want to predict
reframed.drop(reframed[['var1(t)', 'var2(t)', 'var3(t)', 'var4(t)',
       'var5(t)', 'var6(t)', 'var7(t)', 'var8(t)', 'var9(t)', 'var10(t)',
       'var11(t)', 'var12(t)', 'var13(t)', 'var14(t)', 'var15(t)']], axis=1, inplace=True)
print(reframed.head())
```

```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1   0.117647   0.369010   0.461938   0.875000   0.166667        1.0
2   0.023529   0.376997   0.461938   0.829545   0.166667        0.0
3   0.035294   0.399361   0.463668   0.738636   0.166667        1.0
4   0.129412   0.410543   0.472318   0.727273   0.166667        0.0
5   0.117647   0.408946   0.475779   0.750000   0.166667        0.0

   var7(t-1)  var8(t-1)  var9(t-1)  var10(t-1)  var11(t-1)  var12(t-1)  \
1   0.944444   0.399170   0.654419    0.007407         0.0         0.0
2   0.000000   0.399170   0.654419    0.000000         0.0         0.0
3   0.944444   0.597656   0.668579    0.007407         0.0         0.0
4   0.000000   0.597656   0.668579    0.000000         0.0         0.0
5   0.000000   0.597656   0.668579    0.000000         0.0         0.0

   var13(t-1)  var14(t-1)  var15(t-1)  var16(t-1)  var16(t)
1         0.0         0.0    0.633823       0.125     0.000
2         0.0         0.0    0.633823       0.000     0.125
3         0.0         0.0    0.633823       0.125     0.000
4         0.0         0.0    0.633823       0.000     0.000
5         0.0         0.0    0.633823       0.000     0.000
```

Figure 19: Reframing

Once the dataset was reframed, the values were split into test/train categories for keras model fitting. Initially the data points were fit for 8 years of historic data and the inputs were shaped into a 3D format. LSTM expects the input to be of shape [samples, timestep, features]. Furthermore, the keras libraries were loaded and the splitted dataset was used for multilayer prediction modelling. Our model used the Mean Absolute Error 'MAE' loss function and the 'adam' optimizer for stochastic gradient descending. Adam as an optimizer, performs an adaptive learning rate process, this means that it calculates individual learning rates for various parameters in order to optimize the prediction and reduce loss

Initially we modelled our layers to have 92 input layers, 50 epochs, and a batch size of 100. Running the model created a plot, which is shown below. As we can see, there is a significant decrease in loss between the test and train values as the epochs are processed. We wonder if we are under fitting the training data because we obtain a Root Mean Squared Error (RMSE) of 4.549.

```
[ ]  # plot history
     pyplot.plot(history.history['loss'], label='train')
     pyplot.plot(history.history['val_loss'], label='test', color='orange')
     pyplot.legend()
     pyplot.xlabel('epochs')
     pyplot.ylabel('loss')
     pyplot.title('Line Plot of Train Test Loss of Multivariable LSTM ')
     pyplot.show()
```

Figure 20: Line Plot Train Test Loss, 92 input layers

To further investigate, we experimented with tuning the prediction model by adjusting the input layer size to 50, while keeping the same number of batch size to 100 and epochs to 50. Below is the plot with the adjusted model. We can see that the test data is not stable and fluctuates indefinitely when compared with the train data. There is some loss between the two but RMSE was resulted to have decreased to 4.541.

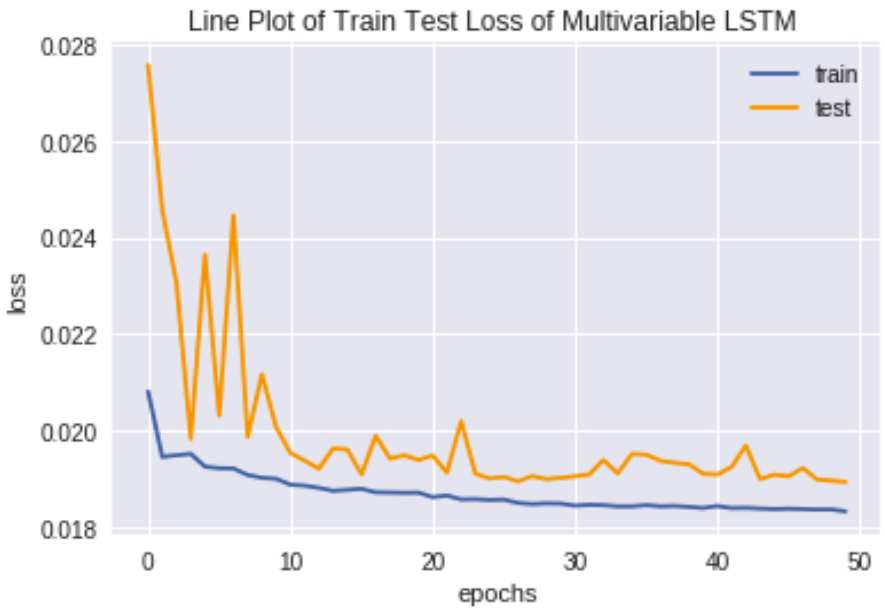


Figure 21: Line plot with 50 input layers

6.2 Linear Regression with Keras

Further exploration was made by creating a linear regression model and NN prediction with keras. Below is the code written to split the dataset into test train sets and in order to apply the linear regression predictor model.

```
Y_values=  data[['nombre_unites',"#int"]].values
X_values = data[data.columns[:8]].values
X_values=scale(X_values)
X_values1 = data[data.columns[8:-2]].values
X_values= np.concatenate((X_values, X_values1), axis=1)

X_train, X_test, y_train, y_test = train_test_split(X_values,Y_values,test_size=0.3,random_state=40)
```

```
# Train a model and predict the Number of interventions in a special hour
Ic2=LinearRegression()
Ic2.fit(X_train,y_train[:,1])
b=Ic2.score(X_test,y_test[:,1])

int_hat=Ic2.predict(X_test)
```

Figure 22: Linear Regression

The graph below displays the relation between the true value and the prediction value. The mse was 0.0353, which is not bad. Any mean squared error value close to zero is a good metric.

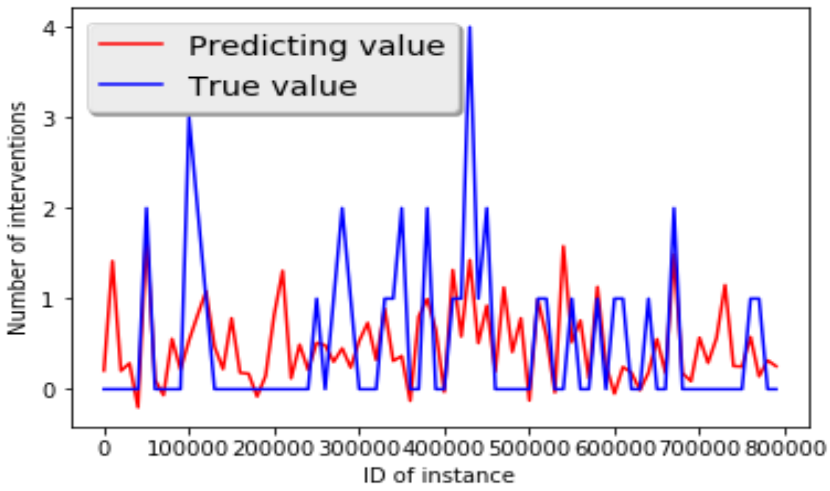


Figure 23: Comparison between the result of Linear Regression model and True values

The Neural Network Keras model was again Sequential, with a Dense input dimension of 69 units. We were experimenting with the tanh activation. Tanh is mainly used for classification; it performs well by keeping negative values in a negative domain and keeping zero values near the zero of the graph. Below we can see the keras model that we fitted after splitting the data. The regression loss function was 'mse' and the optimizer, 'sgd'. To elaborate, Mean Squared Error is the sum of differences between the target and predicted values, while Stochastic Gradient Descent works well with sparse data as ours. The model was run for 100 epochs with a batch size of 50.

```
model=Sequential()
model.add(Dense(units=100, activation='tanh', input_dim=62))
model.add(Dense(units=150, activation='tanh'))
model.add(Dense(units=50, activation='tanh'))
model.add(Dense(units=2, activation='linear'))

model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=['mse'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 100)               6300
_________________________________________________________________
dense_6 (Dense)              (None, 150)               15150
_________________________________________________________________
dense_7 (Dense)              (None, 50)                7550
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 102
=================================================================
Total params: 29,102
Trainable params: 29,102
Non-trainable params: 0
```

Figure 24: Neural Network

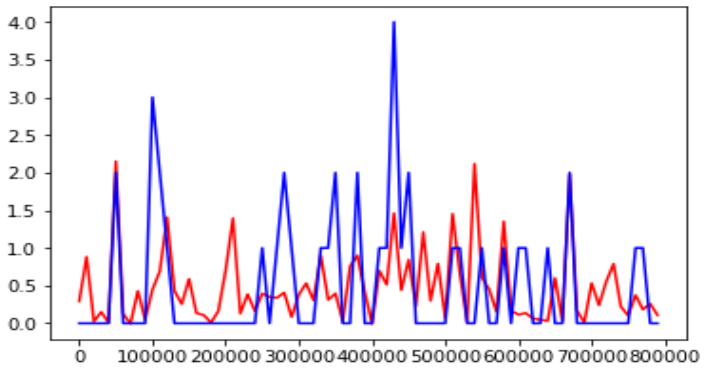


Figure 25: Comparison between the result of NN model and True values

The graph above displays the difference between the predicted result and true values when running the keras NN. The average mse using neural network was 0.4098, which is higher when compared to linear regression modelling. We wonder if the Linear Regression model was a better predictor for our dataset.

6.3 Facebook Prophet

Additionally, Facebook Prophet was also investigated and used for our prediction modelling. As researched, the forecasting model performs well within time series data on different time scales: yearly, weekly, and daily.

Prophet is an open source released by Facebook in order to provide some useful guidance for producing forecast at scale. It is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

Prophet follows the sklearn model API. The input to Prophet is a dataframe with two columns: ds and y. The ds is a date/time stamp column while the y column must be numeric, and represents the measurement we wish to forecast.

The model was tested with several date/time aggregations, at borough or city level with the number of interventions being the forecasted value (y). The date/time input used was hourly, 6 hours slots and daily. Below we present the results for an hourly aggregation of interventions at

city level. Overlap of the predicted values (dark blue), range of confidence 80% (light blue) and actual data (black dots)

```
m = Prophet()
m.fit(df)
future = m.make_future_dataframe(periods=1000, freq='H')
fcst = m.predict(future)
fig = m.plot(fcst)
```

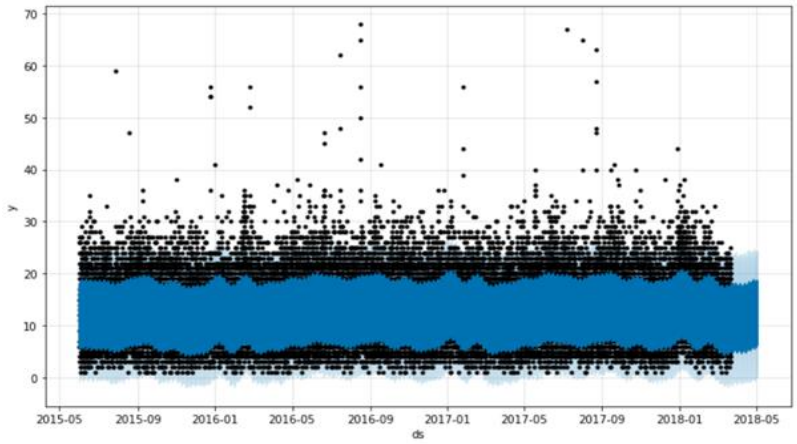


Figure 26: FB Prophet plot

The Prophet model displays the trends and seasonality detected with one simple function:

```
fig = m.plot_components(fcst)
```

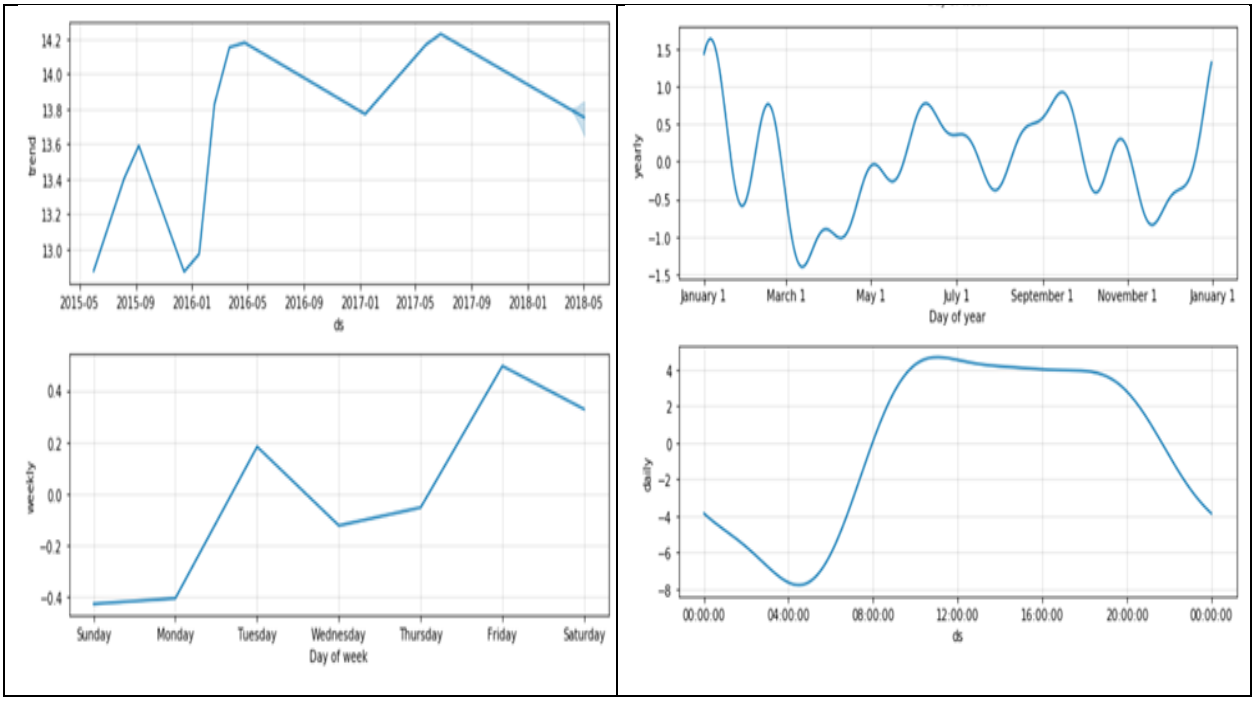


Figure 27: Prophet Trends

The cross-validation was performed using the built-in functions.

```
from fbprophet.diagnostics import cross_validation
df_cv = cross_validation(m, initial='700 days', period='700 days',horizon = '10 days')
```

The model was able to capture the daily seasonality however it did not capture the spikes in interventions correctly which led to a large mean average percentage error.

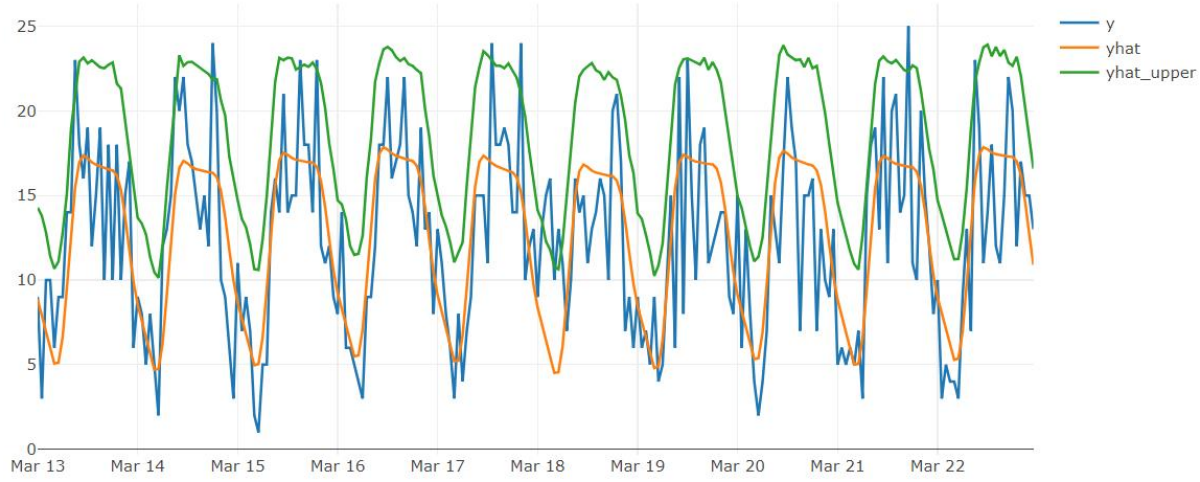


Figure 28: Cross Validation

```
from fbprophet.plot import plot_cross_validation_metric
fig = plot_cross_validation_metric(df_cv, metric='mape')
```

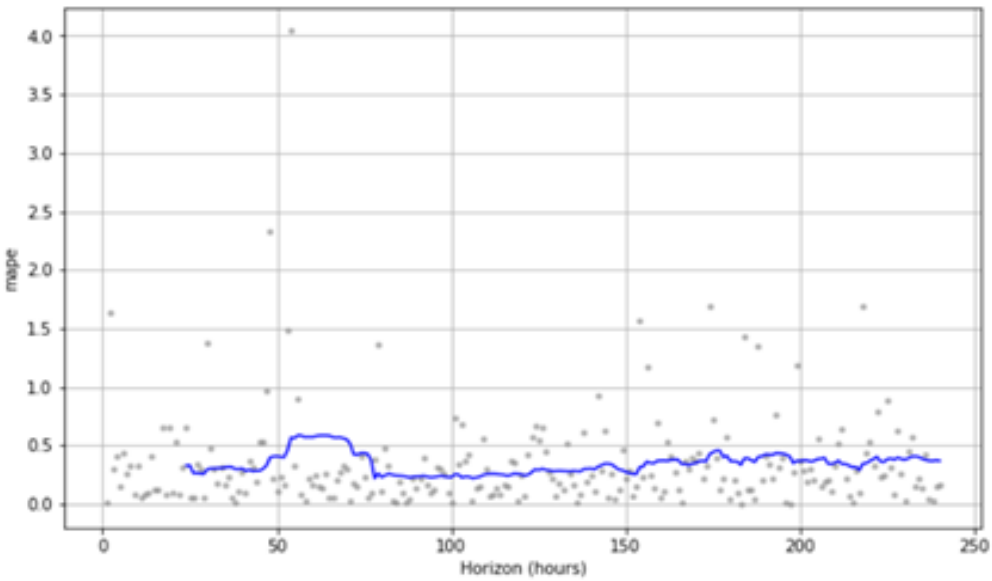


Figure 29: mape metric

By aggregating actual data and forecasted data at 6 hours intervals we were able to reduce the MAPE to under 13%. The time slots correspond to 0-6am; 7-12am; 1-6pm and 7-23pm.

```
In [11]: import numpy as np
         def mean_absolute_percentage_error(y_true, y_pred):
             y_true, y_pred = np.array(y_true), np.array(y_pred)
             return np.mean(np.abs((y_true - y_pred) / y_true))

         mape = mean_absolute_percentage_error(df_cv6.y, df_cv6.yhat)
         print(mape)

         0.1283079566272239
```

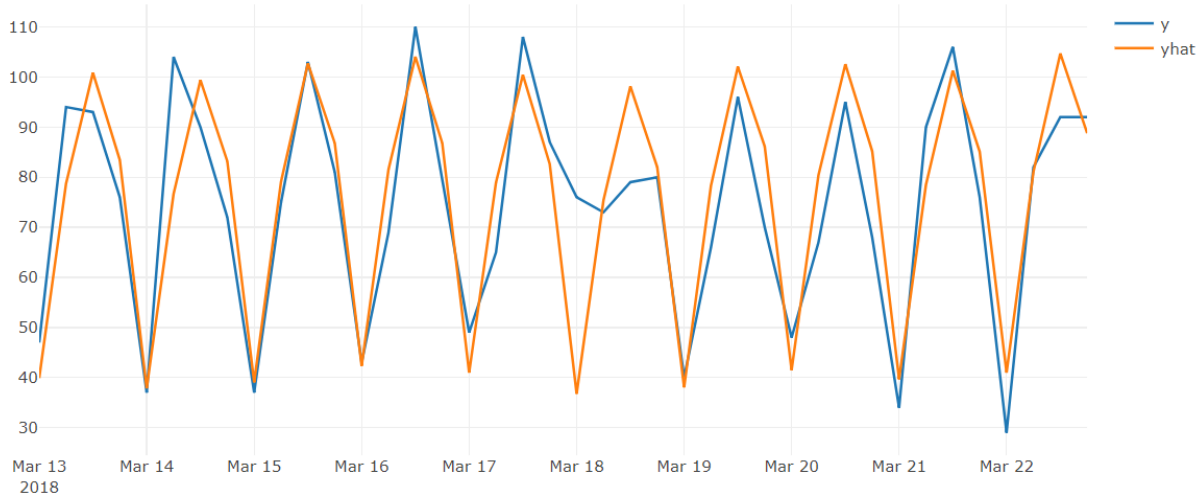


Figure 30: Cross Validation

7. Conclusion

At the very beginning of the project we were asked to derive a problem statement from the first glance of the data. We felt that there was a need to make the emergency call centre as one of the primary stakeholders for our data analysis project. We believed that they can use our data and intervention/hour prediction to properly manage the schedules of their employees. This would mean that through human resource management, they can employ part time employees during non peak hours and full time employees during peak hours.

As we conducted our data prep and data wrangling, it became apparent that this was one of the most important parts and our biggest time sinks. We also found that since we were trying to predict interventions per hour, all of the data that augmented our original dataset, should ideally, add information at the hourly scale. Other than weather data, we found this quite a hard task. Census data was collected every half decade, and many of the hourly data sources such as criminal activity offered little to no benefit to our predictive model or was just too messy to merge.

Throughout our data analysis phase, we discovered a lot of information and derived many correlations and assumptions from the datasets we used. We saw that Fridays were the busiest days for firemen and 17h00 had the most potential during the day for an intervention to occur. We created various visuals to display the information through histograms, bubble charts, and line graphs. The visuals helped us easily see the areas of population and population density and how they affect the number of incidents in that particular area. The data analysis phase helped us further develop our skills in tableau and gave us an insight of the quality of the data. However, we hoped that more quantitative historic data was made available in order to increase the quality of our analysis and potentially our predictions.

Once the preliminary data analysis phase was conducted, the data was further fed into various prediction models in order to predict the incidents per hour. For our LSTM model we split the data sequentially by years, we used the first 6 years (2008-2013) for training, the next 2 years for cross-validation (2013-15), and the rest (2015-2018.3) for testing. We held back the testing data from any part of the algorithm till it came time for the prediction. We also attempted MLP modelling splitting the data 80/20, using a sparse categorical entropy loss and sparse

categorical accuracy metric. From our predictive modelling we noted that Weather has no predictive effect on number of interventions at any given time. Nor does time, day, or week. Population may be correlated with total number of interventions but still has no predictive effect as to when an intervention might occur, only that the probability of an intervention occurring increases. Also, of note is that population density has low effect on the total number of interventions. This was assumed during the early stages of the data analysis.

In effect, our predictive modelling is inconclusive in which we could not properly predict the number of incident calls in a given hour to the emergency call centres. With that regard we could not offer the stakeholder with proper knowledge on how to manage their full time/part time employees and whether to assign more or fewer employees for certain peak times.

Yet, this project has helped us manage a team based data science task and further develop our skills in Python, Tableau, Alteryx, GCP, and Data science. We had many constraints such as quality of data, time, and programming resources but we managed to produce many lessons learnt and valuable information from our data analysis and prediction models. We would like to thank all the professors and TAs who have supported us throughout this project and hope to further develop our skills in Data Science and Machine Learning.

8. References

- Cyril Pecoraro, “Predicting the response times of firefighters using data-science - Medium.com” (March 2, 2018) https://medium.com/crim/predicting-the-response-times-of-firefighters-using-data-science-da79f6965f93

- Bernard Marr, “Is Big Data Analytics the Secret to Successful Fire Fighting - Forbes.com” (January 13, 2017) https://www.forbes.com/sites/bernardmarr/2017/01/13/is-big-data-analytics-the-secret-to-successful-fire-fighting/#5bc69f2845d0

- NFPA Responder Forum, “Data in the Fire Service - Can We Improve Our Planning and Response While Documenting Our Value to Our Community?”, (2015) https://www.nfpa.org/-/media/Files/News-and-Research/Resources/Fire-service/Responder-Forum/2015-NFPA-Responders-Forum-Fire-Service-Data.ashx?la=en

- Tyler Romero,“Barnes Cipollone Romero-Predicting Emergency Incidents In San Diego - Stanford.edu”, (2016) http://cs229.stanford.edu/proj2016/report/.pdf

- XGBoost Documentation, (2016) https://xgboost.readthedocs.io/en/latest/

- Government of Canada, Hourly Data Report (2018) http://climate.weather.gc.ca/historical_data/search_historic_data_e.html

- Ville de Montreal, “Traffic lights - counting vehicles and pedestrians at intersections with lights”, (2018) http://donnees.ville.montreal.qc.ca/dataset/comptage-vehicules-pietons

- Google, “What is Geocoding, Developer Guide” (2018) https://developers.google.com/maps/documentation/geocoding/intro#ReverseGeocoding

- J. Flood, " The Fires: How a Computer Formula, Big Ideas, and the Best of Intentions Burned Down New York City--and Determined the Future of Cities” [Paperback]. Available: Riverhead Books (2011)

- S. Kempe, “LDM Webinar: Data Modelling for Big Data” (2016) https://www.dataversity.net/ldm-webinar-data-modeling-big-data/

- Facebook Prophet paper: Sean J. Taylor, Benjamin Letham (2018) Forecasting at scale. The American Statistician 72(1):37-45 (https://peerj.com/preprints/3190.pdf)

- FB Prophet HTML documentation: https://facebook.github.io/prophet/docs/quick_start.html

9. Appendix

A1. LSTM model

LSTM prediction attempt for Interventions per hour

```
In [540]: import pandas as pd
          from datetime import datetime

In [541]: def parse(x):
              return datetime.strptime(x, '%Y %m %d %H')

In [542]: dataset = pd.read_csv('last_one.csv',  parse_dates = [['year', 'month', 'da

In [543]: dataset.drop('Unnamed: 0', axis=1, inplace=True)

In [544]: dataset.index.name = 'date'

In [545]: dataset.head()
```

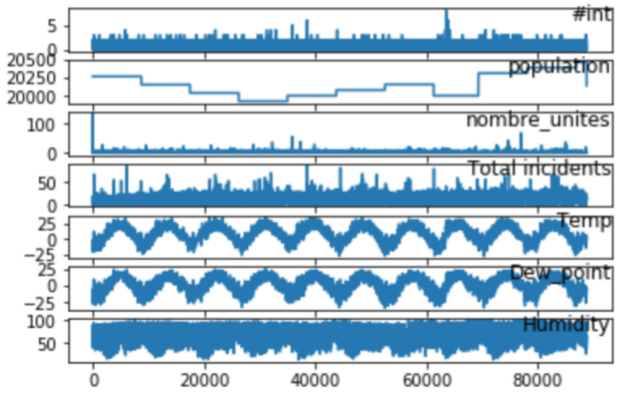
Out[545]:

date	ville	Total incidents	Temp	Dew_point	Humidity	weekday	caserne	division	latitude
2008-01-01 00:00:00	Ahuntsic / Cartierville	11	-4.7	-6.3	89.0	1	42.5	21.0	45.556506
2008-01-01 01:00:00	Ahuntsic / Cartierville	3	-4.2	-6.3	85.0	1	0.0	0.0	45.581554
2008-01-01 02:00:00	Ahuntsic / Cartierville	4	-2.8	-6.2	77.0	1	42.0	21.0	45.524449
2008-01-01 03:00:00	Ahuntsic / Cartierville	12	-2.1	-5.7	76.0	1	38.5	21.0	45.548322
2008-01-01 04:00:00	Ahuntsic / Cartierville	11	-2.2	-5.5	78.0	1	43.0	21.0	45.570641

In [546]:
```
#predict only for westmount as trial
dataset = dataset[dataset['ville']=="Westmount"]
dataset.drop('ville', axis=1, inplace=True)
```

In [ ]:
```

```

In [547]:
```
from matplotlib import pyplot
%matplotlib inline
# load dataset
values = dataset.values
# specify columns to plot
groups = [15, 14, 9, 0, 1, 2, 3]
i = 1
# plot each column
pyplot.figure()
for group in groups:
    pyplot.subplot(len(groups), 1, i)
    pyplot.plot(values[:, group])
    pyplot.title(dataset.columns[group], y=0.5, loc='right')
    i += 1
pyplot.show()
```



In [548]:
```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

In [549]:
```
from sklearn import preprocessing as pp

# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = pp.MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)

#drop columns we don't want to predict
reframed.drop(reframed[['var1(t)', 'var2(t)', 'var3(t)', 'var4(t)',
       'var5(t)', 'var6(t)', 'var7(t)', 'var8(t)', 'var9(t)', 'var10(t)',
       'var11(t)', 'var12(t)', 'var13(t)', 'var14(t)', 'var15(t)']], axis=1
print(reframed.head())
```

```
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
cv_X, cv_y = cv[:, :-1], cv[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D [samples, timesteps, features]

train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
cv_X = cv_X.reshape((cv_X.shape[0], 1, cv_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, cv_X.shape, cv_y.shape, test_X.shape, t
```

```
(52560, 1, 16) (52560,) (17520, 1, 16) (17520,) (18717, 1, 16) (18717,)
```

In [557]:
```
eras.models import Sequential
eras.layers import Dense, Dropout, Flatten
eras.layers import Embedding
eras.layers import LSTM
eras.optimizers import adam
eras.callbacks import EarlyStopping
eras import metrics
eras import losses

ck = EarlyStopping(monitor='val_loss', min_delta=0, patience=5, verbose=2, m
acks = [callback],
```

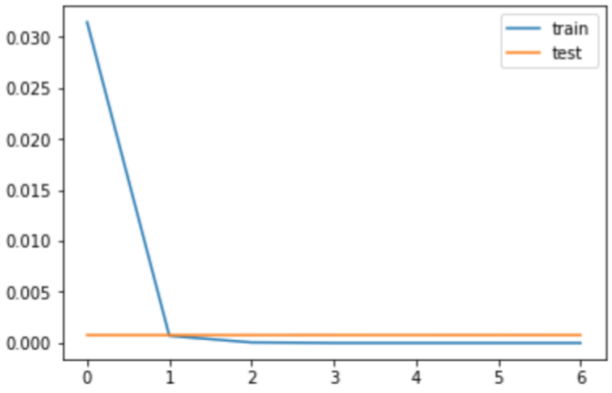
In [569]:
```
Sequential()
ld(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
dd(Flatten())
ld(Dense(2, activation='relu'))
mpile(loss=losses.sparse_categorical_crossentropy, optimizer='adam', metrics
twork
= model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(cv_
istory
lot(history.history['loss'], label='train')
lot(history.history['val_loss'], label='test')
egend()
how()
```

```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1   0.117647   0.369010   0.461938   0.875000   0.166667        1.0
2   0.023529   0.376997   0.461938   0.829545   0.166667        0.0
3   0.035294   0.399361   0.463668   0.738636   0.166667        1.0
4   0.129412   0.410543   0.472318   0.727273   0.166667        0.0
5   0.117647   0.408946   0.475779   0.750000   0.166667        0.0

   var7(t-1)  var8(t-1)  var9(t-1)  var10(t-1)  var11(t-1)  var12(t-1)  \
1   0.944444   0.399170   0.654419    0.007407         0.0         0.0
2   0.000000   0.399170   0.654419    0.000000         0.0         0.0
3   0.944444   0.597656   0.668579    0.007407         0.0         0.0
4   0.000000   0.597656   0.668579    0.000000         0.0         0.0
5   0.000000   0.597656   0.668579    0.000000         0.0         0.0

   var13(t-1)  var14(t-1)  var15(t-1)  var16(t-1)  var16(t)
1         0.0         0.0    0.633823       0.125     0.000
2         0.0         0.0    0.633823       0.000     0.125
3         0.0         0.0    0.633823       0.125     0.000
4         0.0         0.0    0.633823       0.000     0.000
5         0.0         0.0    0.633823       0.000     0.000
```

In [ ]:

In [550]:
```
# split into train and test sets

values = reframed.values


#train data on first 6 year
#cross validate on next 2 year
#test on following ~2 years

n_train_hours = 365 * 24 * 6
n_cv_hours = 365 *24 *8

#split train, cv(cross_validation), test
train = values[:n_train_hours, :]
cv = values[n_train_hours:n_cv_hours, :]
test = values[n_cv_hours:, :]
```

```
52560/52560 [==============================] - 8s 159us/step - loss: 0.03
14 - sparse_categorical_accuracy: 0.8685 - val_loss: 7.7446e-04 - val_spa
rse_categorical_accuracy: 0.8671
Epoch 2/50
52560/52560 [==============================] - 4s 78us/step - loss: 7.029
3e-04 - sparse_categorical_accuracy: 0.8692 - val_loss: 7.7239e-04 - val_
sparse_categorical_accuracy: 0.8671
Epoch 3/50
52560/52560 [==============================] - 5s 96us/step - loss: 5.406
3e-05 - sparse_categorical_accuracy: 0.8692 - val_loss: 7.7565e-04 - val_
sparse_categorical_accuracy: 0.8671
Epoch 4/50
52560/52560 [==============================] - 5s 96us/step - loss: 1.226
4e-06 - sparse_categorical_accuracy: 0.8692 - val_loss: 7.7669e-04 - val_
sparse_categorical_accuracy: 0.8671
Epoch 5/50
52560/52560 [==============================] - 5s 98us/step - loss: 1.193
8e-06 - sparse_categorical_accuracy: 0.8692 - val_loss: 7.7802e-04 - val_
sparse_categorical_accuracy: 0.8671
Epoch 6/50
52560/52560 [==============================] - 5s 99us/step - loss: 1.156
5e-06 - sparse_categorical_accuracy: 0.8692 - val_loss: 7.7974e-04 - val_
sparse_categorical_accuracy: 0.8671
Epoch 7/50
52560/52560 [==============================] - 5s 89us/step - loss: 1.112
3e-06 - sparse_categorical_accuracy: 0.8692 - val_loss: 7.8191e-04 - val_
sparse_categorical_accuracy: 0.8671
Epoch 00007: early stopping
```



In [578]:

```
from math import sqrt
from numpy import concatenate
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error

yhat = model.predict(test_X)

test_X_re = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
test_yhat = yhat[:,0].reshape((len(yhat[:,0]), 1))
inv_yhat = concatenate((test_yhat, test_X_re[:, :-1]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X_re[:, :-1]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 9.596
```

A2. MLP model

Attempt at creating MLP for Intervention Prediction

In [62]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [63]:
```
df = pd.read_csv("last_one.csv")
```

In [64]:
```
df.drop('Unnamed: 0', axis = 1, inplace = True)
```

In [65]:
```
#one hot encode neighborhoods
test = pd.get_dummies(df,prefix=['ville'])
```

In [66]:
```
test.shape
```

Out[66]:
```
(2663940, 50)
```

In [67]:
```
from sklearn import preprocessing as pp

df_y = test['#int'];
df_X = test.drop("#int", axis = 1);
df_X['#int'] = df_y
values = df_X.values
values = values.astype('float32')
# normalize features
scaler = pp.MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
X, y = scaled[:, :-1], scaled[:, -1]
```

In [68]:
```
from sklearn.model_selection import train_test_split
```

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)
```

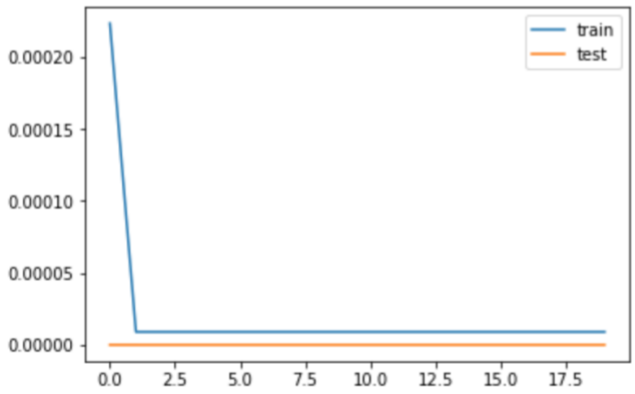
In [70]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras import metrics, losses
```

In [71]:
```
model = Sequential()
model.add(Dense(100, activation='relu', input_dim=49))
model.add(Dense(50))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer='adam', loss=losses.sparse_categorical_crossentrop
y, metrics = [metrics.sparse_categorical_accuracy])
```

In [72]:
```
history = model.fit(X_train, y_train, epochs=20, batch_size=200, validatio
n_data=(X_test, y_test),  verbose=1, shuffle=False)
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```

```
Train on 1784839 samples, validate on 879101 samples
Epoch 1/20
1784839/1784839 [==============================] - 31s 17us/step - loss:
2.2333e-04 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1962e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 2/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1499e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1922e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 3/20
1784839/1784839 [==============================] - 17s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 4/20
1784839/1784839 [==============================] - 17s 10us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 5/20
1784839/1784839 [==============================] - 17s 10us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 6/20
1784839/1784839 [==============================] - 17s 10us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 7/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 8/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 9/20
```

```
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 16/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 17/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 18/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 19/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
Epoch 20/20
1784839/1784839 [==============================] - 16s 9us/step - loss:
9.1498e-06 - sparse_categorical_accuracy: 0.7032 - val_loss: 1.1921e-07 -
val_sparse_categorical_accuracy: 0.7034
```



In [74]:
```
1873489/df_y.shape[0]
```

Out[74]:
```
0.7032774762194344
```

Unfortunately can see that the algorithm is only outputting Zero as the predictive quantity. If we only predict Zero interventions per hour we arrive at 70% accuracy, although this is NOT what we want. Seems to be little to no predictive power.

A3. Facebook Prophet Model

In [1]:
```
import pandas as pd
from fbprophet import Prophet
import matplotlib.pyplot as plt
from fbprophet.diagnostics import cross_validation
from fbprophet.diagnostics import performance_metrics
import plotly.plotly as py
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode()
```

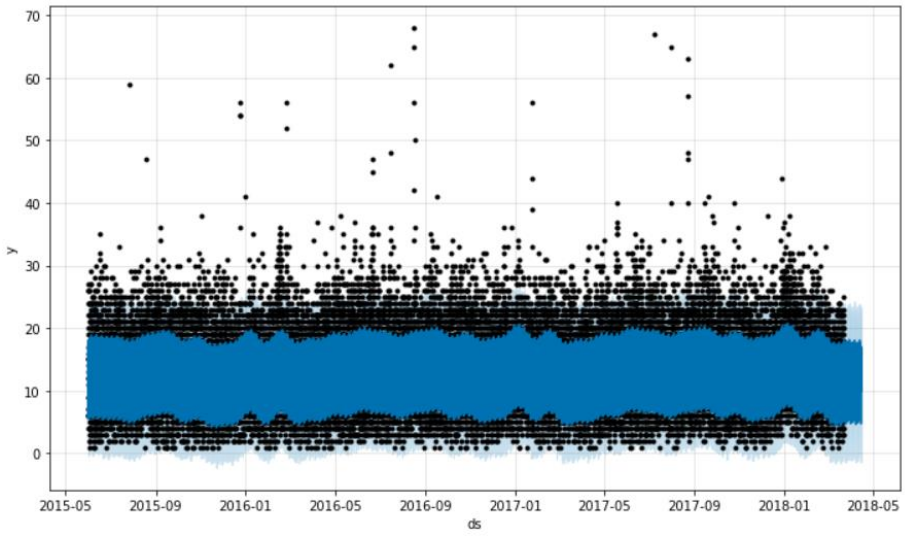
In [2]:
```
filename = "C:/Users/rflun/Downloads/lastone.csv"
X = pd.read_csv(filename)
X['month'] = X.month.map("{:02}".format)
X['day'] = X.day.map("{:02}".format)
X['hour'] = X.hour.map("{:02}".format)
X['date']=X.year.map(str) + "-" + X.month + "-"+ X.day+ " " +X.hour +":00:00"
X['date']=pd.to_datetime(X['date'])
df = X[['date','#int']]
df = df.groupby(['date']).sum()
df = df.reset_index()
df.columns = ['ds','y']
df = df[(df['ds'] >= '2015-06-01 00:00:00') & (df['ds'] <= '2018-03-22 23:00:00')]
#df.loc[(df['ds'] > '2015-05-01 00:00:00') & (df['ds'] < '2015-05-30 23:00:00'), 'y'] = None
df.head()
```

Out[2]:

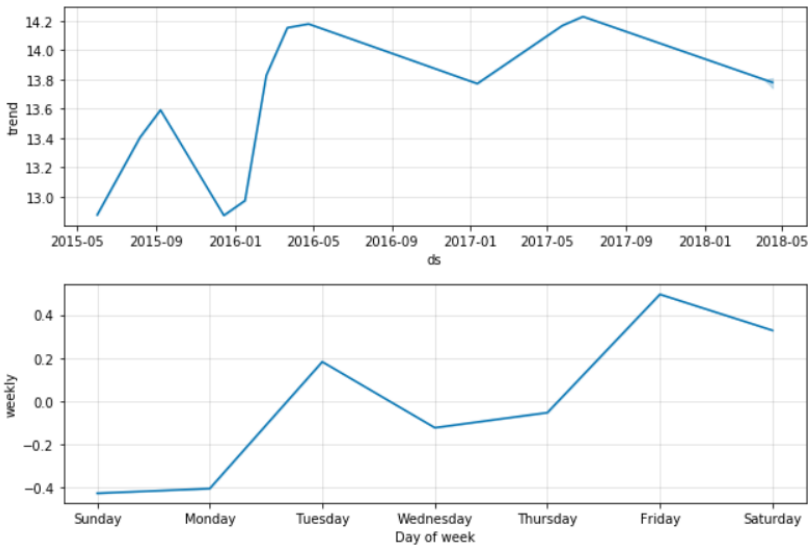
	ds	y
64182	2015-06-01 00:00:00	11
64183	2015-06-01 01:00:00	7
64184	2015-06-01 02:00:00	6
64185	2015-06-01 03:00:00	6
64186	2015-06-01 04:00:00	6

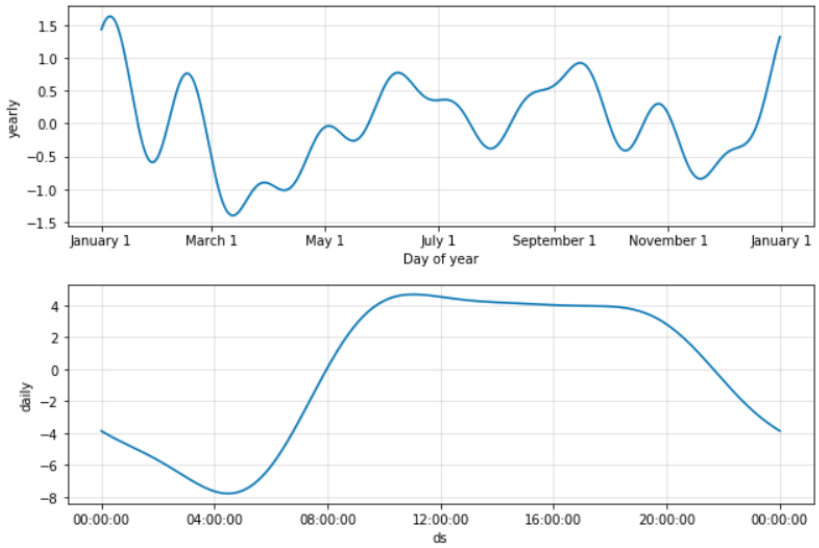
In [3]:
```
m = Prophet()
m.fit(df)
future = m.make_future_dataframe(periods=576, freq='H')
fcst = m.predict(future)
fig = m.plot(fcst)
```

```
C:\Users\rflun\Anaconda3\envs\TF - keras\lib\site-packages\pystan\misc.py:399: FutureWarning:

Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `
np.float64 == np.dtype(float).type`.
```



In [4]:
```
fig = m.plot_components(fcst)
```





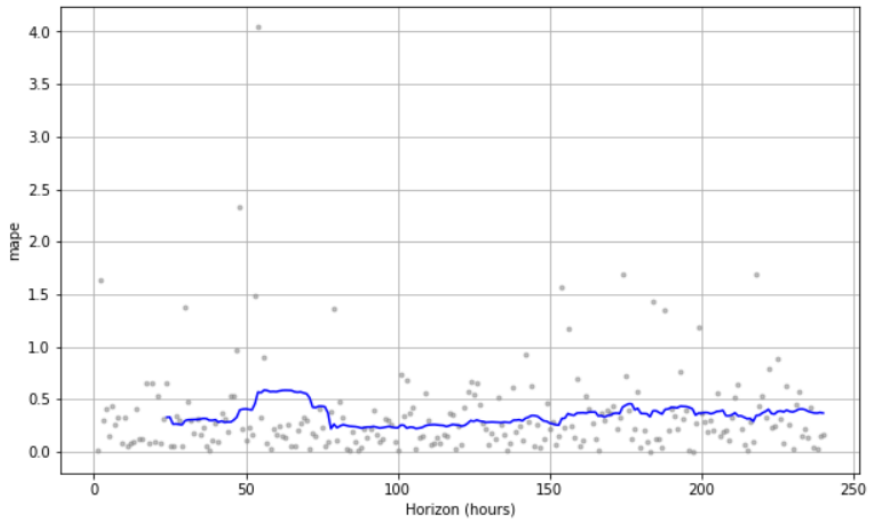
In [5]:
```
from fbprophet.diagnostics import cross_validation
df_cv = cross_validation(m, initial='700 days', period='700 days',horizon = '10 days')
df_cv.tail(5)
```

```
INFO:fbprophet:Making 1 forecasts with cutoffs between 2018-03-12 23:00:00 and 2018-03-12 23:00:00
```

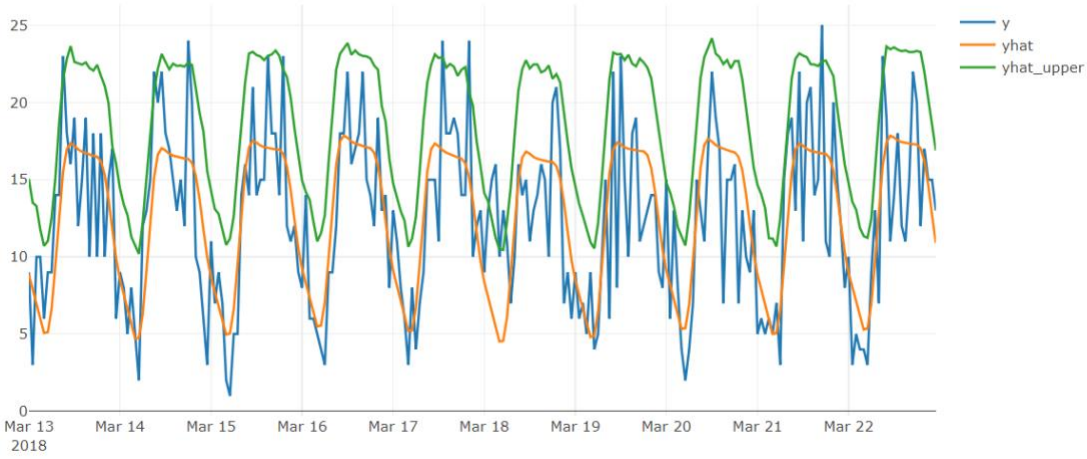
Out[5]:

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
235	2018-03-22 19:00:00	17.023549	11.191501	23.266326	12	2018-03-12 23:00:00
236	2018-03-22 20:00:00	16.194060	10.827799	21.956656	17	2018-03-12 23:00:00
237	2018-03-22 21:00:00	14.663420	8.572589	20.246980	15	2018-03-12 23:00:00
238	2018-03-22 22:00:00	12.718541	6.663141	18.661981	15	2018-03-12 23:00:00
239	2018-03-22 23:00:00	10.896216	5.017019	16.888360	13	2018-03-12 23:00:00

In [6]:
```
from fbprophet.plot import plot_cross_validation_metric
fig = plot_cross_validation_metric(df_cv, metric='mape')
```



In [7]:
```
py.iplot([go.Scatter(x=df_cv['ds'], y=df_cv['y'], name = 'y'),
          go.Scatter(x=df_cv['ds'], y=df_cv['yhat'], name = 'yhat'),
          go.Scatter(x=df_cv['ds'], y=df_cv['yhat_upper'], name = 'yhat_upper')])
```



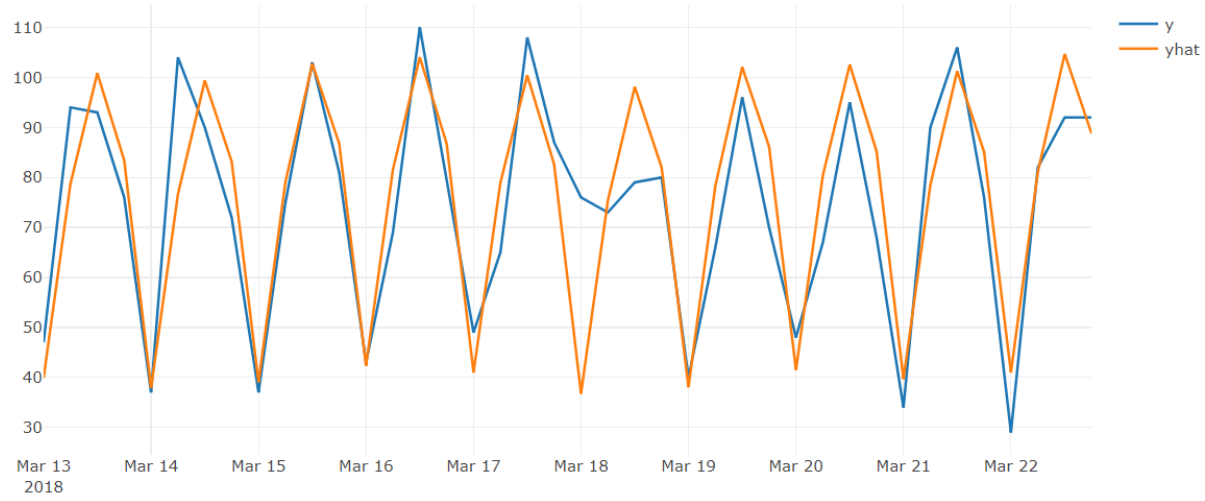
In [8]:
```
df_cv6 = df_cv.resample('6H', on='ds').sum()
df_cv6.head()
```

Out[8]:

ds	yhat	yhat_lower	yhat_upper	y
2018-03-13 00:00:00	39.897297	4.958072	75.357710	47
2018-03-13 06:00:00	78.759201	43.651673	114.383025	94
2018-03-13 12:00:00	100.838761	65.691374	134.526167	93
2018-03-13 18:00:00	83.425464	48.867425	118.339434	76
2018-03-14 00:00:00	37.863939	2.513073	72.716663	37

In [9]:
```
df_cv6 = df_cv6.reset_index()
```

In [10]:
```
py.iplot([go.Scatter(x=df_cv6['ds'], y=df_cv6['y'], name = 'y'),
          go.Scatter(x=df_cv6['ds'], y=df_cv6['yhat'], name = 'yhat')])
```



In [11]:
```
import numpy as np
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true))

mape = mean_absolute_percentage_error(df_cv6.y, df_cv6.yhat)
print(mape)
```

```
0.1283079566272239
```

A4. Linear regression model and fully-connected model

1. Linear Regression

In [193]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import scale
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as MSE
import matplotlib.pyplot as plt
%matplotlib inline

path='D:/Projects/Capstone/'
filename=path+'intervention.csv'
data=pd.read_csv(filename,encoding='ISO-8859-1',na_filter=True)
```

In [194]:
```
data.head()
```

Out[194]:

	Unnamed: 0	ville	year	month	day	hour	Total incidents	Temp	Dew_point	Humidity	...	division	latitude	longitude	nombre_unites	area
0	0	Ahuntsic / Cartierville	2008	1	1	0	11	-4.7	-6.3	89.0	...	21.0	45.556506	-73.685032	2.0	24.1
1	1	Ahuntsic / Cartierville	2008	1	1	1	3	-4.2	-6.3	85.0	...	21.0	45.581554	-73.651107	1.0	24.1
2	2	Ahuntsic / Cartierville	2008	1	1	2	4	-2.8	-6.2	77.0	...	21.0	45.524449	-73.737146	1.0	24.1
3	3	Ahuntsic / Cartierville	2008	1	1	3	12	-2.1	-5.7	76.0	...	21.0	45.548322	-73.677277	3.0	24.1
4	4	Ahuntsic / Cartierville	2008	1	1	4	11	-2.2	-5.5	78.0	...	21.0	45.570641	-73.656907	1.0	24.1

5 rows × 22 columns

In [196]:
```
dict_ville={}
for i in range(30):
    dict_ville[ville[i]]=i
dict_ville
```

Out[196]:
```
{'Ahuntsic / Cartierville': 26,
 'Anjou': 9,
 "Beaconsfield / Baie d'UrfÃ©": 27,
 'CÃ´te St-Luc / Hampstead / Mtl-Ouest': 11,
 'CÃ´te-des-Neiges / Notre-Dame-de-GrÃ¢ce': 19,
 'Dollard-des-Ormeaux / Roxboro': 21,
 'Dorval / Ile Dorval': 14,
 'Ile-Bizard / Ste-GeneviÃ¨ve / Ste-A-de-B': 23,
 'Kirkland': 4,
 "L'Ã\x8Ele-Bizard-Sainte-GeneviÃ¨ve": 0,
 'Lachine': 8,
 'Lasalle': 7,
 'Mercier-Hochelaga-Maisonneuve': 29,
 'Mont-Royal': 20,
 'MontrÃ©al-Nord': 16,
 'Outremont': 17,
 'Pierrefonds / Roxboro': 5,
 'Pierrefonds / Senneville': 22,
 'Plateau Mont-Royal': 15,
 'Pointe-Claire': 13,
 'RiviÃ¨re-des-Prairies / P-A-T/Mtl-Est': 10,
 'RiviÃ¨re-des-Prairies / Pointe-aux-Trembles': 24,
 'Rosemont-La Petite-Patrie': 12,
 'Saint-Laurent': 25,
 'Saint-LÃ©onard': 28,
 'Sud-Ouest': 2,
 'Verdun': 18,
 'Ville-Marie': 6,
 'Villeray-Saint-Michel-Parc-Extension': 1,
 'Westmount': 3}
```

Hard-code ville and hour

In [197]:
```
data.shape
```

Out[197]:
```
(2663940, 22)
```

In [198]:
```
len(data)
```

Out[198]:
```
2663940
```

In [199]:
```
ville_=data['ville'].values
index=dict_ville[ville_[0]]
index
```

Out[199]:
```
26
```

In [200]:
```
district_code=np.zeros((30,len(data)))
ville_=data['ville'].values
for i in range(len(ville_)):
    index=dict_ville[ville_[i]]
    district_code[index][i]=1
```

In [201]:
```
hour_code=np.zeros((24,len(data)))
hour_=data['hour'].values
for i in range(len(hour_)):
    index=hour_[i]
    hour_code[index][i]=1
```

In [202]:
```
data.columns
```

Out[202]:
```
Index(['Unnamed: 0', 'ville', 'year', 'month', 'day', 'hour',
       'Total incidents', 'Temp', 'Dew_point', 'Humidity', 'weekday',
       'caserne', 'division', 'latitude', 'longitude', 'nombre_unites',
       'area(km2)', 'density', 'income-avg(2015)', 'income-median(2015)',
       'population', '#int'],
      dtype='object')
```

In [203]:
```
data=data[['Temp', 'Dew_point', 'Humidity', 'weekday','nombre_unites','area(km2)', 'density', 'income-avg(2015)', 'income-median
(2015)',
       'population', '#int']]
```

In [204]:
```
data.shape
```

Out[204]:
```
(2663940, 11)
```

In [205]:
```
name=['v1','v2','v3','v4','v5','v6','v7','v8','v9','v10','v11','v12','v13','v14','v15','v16','v17','v18','v19','v20','v21','v22'
,'v23','v24','v25','v26','v27','v28','v29','v30']
for i in range(30):
    data[name[i]]=district_code[i]
```

In [206]:
```
name=['h0','h1','h2','h3','h4','h5','h6','h7','h8','h9','h10','h11','h12','h13','h14','h15','h16','h17','h18','h19','h20','h21',
'h22','h23']
for i in range(24):
    data[name[i]]=hour_code[i]
```

In [207]:
```
data.head()
```

Out[207]:

	Temp	Dew_point	Humidity	weekday	nombre_unites	area(km2)	density	income-avg(2015)	income-median(2015)	population	...	h14	h15	h16	h17	h
0	-4.7	-6.3	89.0	1	2.0	24.16	5556.498344	70105.0	51169.0	122475.1932	...	0.0	0.0	0.0	0.0	0
1	-4.2	-6.3	85.0	1	1.0	24.16	5556.498344	70105.0	51169.0	122475.1932	...	0.0	0.0	0.0	0.0	0
2	-2.8	-6.2	77.0	1	1.0	24.16	5556.498344	70105.0	51169.0	122475.1932	...	0.0	0.0	0.0	0.0	0
3	-2.1	-5.7	76.0	1	3.0	24.16	5556.498344	70105.0	51169.0	122475.1932	...	0.0	0.0	0.0	0.0	0
4	-2.2	-5.5	78.0	1	1.0	24.16	5556.498344	70105.0	51169.0	122475.1932	...	0.0	0.0	0.0	0.0	0

5 rows × 65 columns

In [208]:
```
data=data.drop(['weekday'], axis=1)
# data
```

In [209]:
```
columns=data.columns.tolist()
columns=columns[:3]+columns[4:9]+columns[10:]+[columns[3]]+[columns[9]]
columns
```

Out[209]:
```
['Temp',
 'Dew_point',
 'Humidity',
 'area(km2)',
 'density',
 'income-avg(2015)',
 'income-median(2015)',
 'population',
 'v1',
```

In [210]:
```
data=data[columns]
```

In [212]:
```
scores=[[]]
```

In [213]:
```
Y_values=  data[['nombre_unites',"#int"]].values
X_values = data[data.columns[:8]].values
X_values=scale(X_values)
X_values1 = data[data.columns[8:-2]].values
X_values= np.concatenate((X_values, X_values1), axis=1)

X_train, X_test, y_train, y_test = train_test_split(X_values,Y_values,test_size=0.3,random_state=40)
```

In [214]:
```
y_test
```

Out[214]:
```
array([[ 1.,  0.],
       [ 2.,  0.],
       [ 1.,  0.],
       ...,
       [ 1.,  0.],
       [ 4.,  0.],
       [ 3.,  0.]])
```

In [215]:
```
# Train a model and predict the Number of units deployed for a special hour
Ic1=LinearRegression()
Ic1.fit(X_train,y_train[:,0])
a=Ic1.score(X_test,y_test[:,0])
units_hat=Ic1.predict(X_test)

# Train a model and predict the Number of interventions in a special hour
Ic2=LinearRegression()
Ic2.fit(X_train,y_train[:,1])
b=Ic2.score(X_test,y_test[:,1])

int_hat=Ic2.predict(X_test)
```

In [216]:
```
units_hat
```

Out[216]:
```
array([ 1.92919922,  1.70117188,  2.02062988, ...,  1.75527954,
        1.82113647,  1.92474365])
```

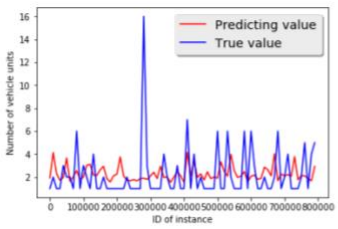
In [217]:
```
int_hat
```

Out[217]:
```
array([ 0.20603561,  0.2299614 ,  0.76732254, ...,  0.46818161,
        0.13062286, -0.13461685])
```

Comparing the result of prediction and ture value

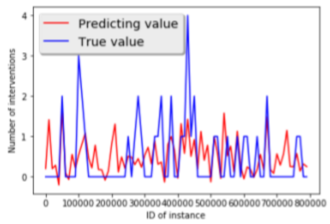
In [223]:
```
fig, ax = plt.subplots()
ax.plot(range(0,len(y_test),10000),units_hat[range(0,len(y_test),10000)],'r-', label='Predicting value')
ax.plot(range(0,len(y_test),10000),y_test[:,0][range(0,len(y_test),10000)],'b-', label='True value')
# plt.plot(range(0,len(y_test),10000),y_hat[:,0][range(0,len(y_test),10000)],'r-',range(0,len(y_test),10000),y_test[:,0][range
(0,len(y_test),10000)],'b-')
legend = ax.legend(shadow=True, fontsize='x-large')
plt.xlabel("ID of instance")
plt.ylabel("Number of vehicle units")
# plt.plot(range(0,len(y_test),10000),units_hat[range(0,len(y_test),10000)],'r-',range(0,len(y_test),10000),y_test[:,0][range(0,
len(y_test),10000)],'b-')
```

Out[223]:
```
<matplotlib.text.Text at 0x23d55b26e48>
```



In [224]:
```
fig, ax = plt.subplots()
ax.plot(range(0,len(y_test),10000),int_hat[range(0,len(y_test),10000)],'r-', label='Predicting value')
ax.plot(range(0,len(y_test),10000),y_test[:,1][range(0,len(y_test),10000)],'b-', label='True value')

legend = ax.legend(shadow=True, fontsize='x-large')
plt.xlabel("ID of instance")
plt.ylabel("Number of interventions")

# plt.plot(range(0,len(y_test),10000),int_hat[range(0,len(y_test),10000)],'r-',range(0,len(y_test),10000),y_test[:,0][range(0,le
n(y_test),10000)],'b-')
```

Out[224]:
```
<matplotlib.text.Text at 0x23d74fda6a0>
```



In [221]:
```
X_values.shape
```

Out[221]:
```
(2663940, 62)
```

In [222]:
```
a
```

Out[222]:
```
0.03533605123108019
```

In [225]:
```
b
```

Out[225]:
```
0.22887685074063102
```

2. Neural Network

In [131]:
```
import tensorflow as tf
from tensorflow.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras import Sequential
from tensorflow.python.keras import optimizers
```

In [137]:
```
model=Sequential()
model.add(Dense(units=100, activation='tanh', input_dim=62))
model.add(Dense(units=150, activation='tanh'))
model.add(Dense(units=50, activation='tanh'))
model.add(Dense(units=2, activation='linear'))

model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=['mse'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 100)               6300
_________________________________________________________________
dense_6 (Dense)              (None, 150)               15150
_________________________________________________________________
dense_7 (Dense)              (None, 50)                7550
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 102
=================================================================
Total params: 29,102
Trainable params: 29,102
Non-trainable params: 0
_________________________________________________________________
```

In [138]:
```
y_test.shape
```

Out[138]:
```
(799182, 2)
```

In [139]:
```
X_train.shape
```

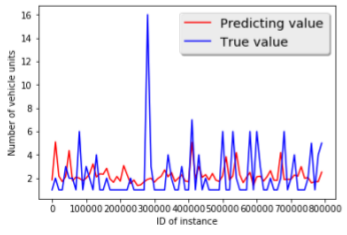
Out[139]:
```
(1864758, 62)
```

In [140]:
```
model.fit(X_train, y_train,epochs=100,batch_size=50)
```

```
Epoch 1/100
1864758/1864758 [==============================] - 157s 84us/step - loss: 3.5822 - mean_squared_error: 3.5822
Epoch 2/100
1864758/1864758 [==============================] - 178s 96us/step - loss: 3.5690 - mean_squared_error: 3.5690
Epoch 3/100
1864758/1864758 [==============================] - 128s 69us/step - loss: 3.5660 - mean_squared_error: 3.5660
```

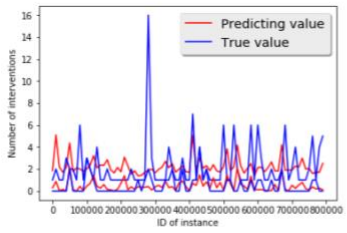
```
In [140]: model.fit(X_train, y_train,epochs=100,batch_size=50)

Epoch 1/100
1864758/1864758 [==============================] - 157s 84us/step - loss: 3.5822 - mean_squared_error: 3.5822
Epoch 2/100
1864758/1864758 [==============================] - 178s 96us/step - loss: 3.5690 - mean_squared_error: 3.5690
Epoch 3/100
1864758/1864758 [==============================] - 128s 69us/step - loss: 3.5660 - mean_squared_error: 3.5660
Epoch 4/100
1864758/1864758 [==============================] - 93s 50us/step - loss: 3.5638 - mean_squared_error: 3.5638
Epoch 5/100
1864758/1864758 [==============================] - 87s 47us/step - loss: 3.5621 - mean_squared_error: 3.5621
Epoch 6/100
1864758/1864758 [==============================] - 95s 51us/step - loss: 3.5606 - mean_squared_error: 3.5606
Epoch 7/100
1864758/1864758 [==============================] - 93s 50us/step - loss: 3.5594 - mean_squared_error: 3.5594
Epoch 8/100
1864758/1864758 [==============================] - 94s 50us/step - loss: 3.5582 - mean_squared_error: 3.5582
Epoch 9/100
1864758/1864758 [==============================] - 96s 51us/step - loss: 3.5574 - mean_squared_error: 3.5574
Epoch 10/100
1864758/1864758 [==============================] - 94s 51us/step - loss: 3.5565 - mean_squared_error: 3.5565
Epoch 11/100
1864758/1864758 [==============================] - 95s 51us/step - loss: 3.5558 - mean_squared_error: 3.5558
Epoch 12/100
1864758/1864758 [==============================] - 94s 50us/step - loss: 3.5548 - mean_squared_error: 3.5548
Epoch 13/100
1864758/1864758 [==============================] - 94s 50us/step - loss: 3.5543 - mean_squared_error: 3.5543
Epoch 14/100
1864758/1864758 [==============================] - 98s 53us/step - loss: 3.5538 - mean_squared_error: 3.5538
Epoch 15/100
1864758/1864758 [==============================] - 96s 52us/step - loss: 3.5533 - mean_squared_error: 3.5533
Epoch 16/100
1864758/1864758 [==============================] - 98s 52us/step - loss: 3.5524 - mean_squared_error: 3.5524
Epoch 17/100
1864758/1864758 [==============================] - 99s 53us/step - loss: 3.5519 - mean_squared_error: 3.5519
Epoch 18/100
1864758/1864758 [==============================] - 98s 53us/step - loss: 3.5518 - mean_squared_error: 3.5518
Epoch 19/100
1864758/1864758 [==============================] - ETA: 0s - loss: 3.5508 - mean_squared_error: 3.55 - 99s 53us/step - loss: 3.
5514 - mean_squared_error: 3.5514
Epoch 20/100
1864758/1864758 [==============================] - 92s 49us/step - loss: 3.5509 - mean_squared_error: 3.5509
Epoch 21/100
1864758/1864758 [==============================] - 89s 48us/step - loss: 3.5506 - mean_squared_error: 3.5506
Epoch 22/100
1864758/1864758 [==============================] - 90s 48us/step - loss: 3.5501 - mean_squared_error: 3.5501
Epoch 23/100
1864758/1864758 [==============================] - 93s 50us/step - loss: 3.5499 - mean_squared_error: 3.5499
Epoch 24/100
1864758/1864758 [==============================] - 92s 49us/step - loss: 3.5494 - mean_squared_error: 3.5494
Epoch 25/100
```

```
Out[140]: <tensorflow.python.keras.callbacks.History at 0x23d0cf99f98>

In [146]: y_hat=model.predict(X_test)

In [142]: y_hat.shape

Out[142]: (799182, 2)

In [143]: MSE(y_hat[0],y_test[0])

Out[143]: 0.40983181576010175

In [144]: MSE(y_hat[1],y_test[1])

Out[144]: 0.07357071037036178

In [191]: fig, ax = plt.subplots()
ax.plot(range(0,len(y_test),10000),y_hat[:,0][range(0,len(y_test),10000)],'r-', label='Predicting value')
ax.plot(range(0,len(y_test),10000),y_test[:,0][range(0,len(y_test),10000)],'b-', label='True value')
# plt.plot(range(0,len(y_test),10000),y_hat[:,0][range(0,len(y_test),10000)],'r-',range(0,len(y_test),10000),y_test[:,0][range
(0,len(y_test),10000)],'b-')
legend = ax.legend(shadow=True, fontsize='x-large')
plt.xlabel("ID of instance")
plt.ylabel("Number of vehicle units")

Out[191]: <matplotlib.text.Text at 0x23d558c0d30>
```



In [192]:
```
fig, ax = plt.subplots()
ax.plot(range(0,len(y_test),10000),y_hat[:,0][range(0,len(y_test),10000)],'r-', label='Predicting value')
ax.plot(range(0,len(y_test),10000),y_test[:,0][range(0,len(y_test),10000)],'b-', label='True value')
# plt.plot(range(0,len(y_test),10000),y_hat[:,0][range(0,len(y_test),10000)],'r-',range(0,len(y_test),10000),y_test[:,0][range
(0,len(y_test),10000)],'b-')
legend = ax.legend(shadow=True, fontsize='x-large')
plt.xlabel("ID of instance")
plt.ylabel("Number of interventions")

# plt.plot(range(0,len(y_test),10000),y_hat[:,1][range(0,len(y_test),10000)],'r-',range(0,len(y_test),10000),y_test[:,1][range
(0,len(y_test),10000)],'b-')
```

Out[192]:
```
[<matplotlib.lines.Line2D at 0x23d559e1fd0>,
 <matplotlib.lines.Line2D at 0x23d559f1a20>]
```



In [ ]: