Indicium Data Scientist Challenge - Behavior of the urban traffic of the city of Sao Paulo in Brazil

Introduction

This project has the objective of analyzing a dataset containing information from urban traffic of the city of São Paulo in Brazil and walk through the process of build a machine learning model to predict the traffic slowness percentage.

The original dataset can be accessed via the link below
https://archive.ics.uci.edu/ml/datasets/Behavior+of+the+urban+traffic+of+the+city+of+Sao+Paulo+in+Brazil

The processes contained in this document consist of:

- Exploratory Data Analysis
- Feature Engineering
- Modeling
- Model Evaluation
- Deploy

Index

1.Methodology and Requirements

(Back to top)

All documentation and the initial steps of Exploratory Data Analysis(EDA), feature engineering and modeling are written on this document. The final code is built using kedro framework and is stored in a github repository.

EDA will analyze the data structures, types, distribution and behavior. Feature engineering will transform the data to be best suited for modelling, including selecting, creating and renaming features, treating null values and so on. Modelling is composed of presenting and testing some techniques, detailing how they work, deciding the best model based on chosen metrics. Validation and performance metrics are assessed using traditional error calculation methods such as RMSE and MAE. the result will be presented in the form of the error behavior.

All requirements for running the kedro project are listed in the requirements.txt and at the README.md of the kedro repository.

The deployment strategy will be using an API and will be explained in the last section of this report.

2.Exploratory Data Analysis

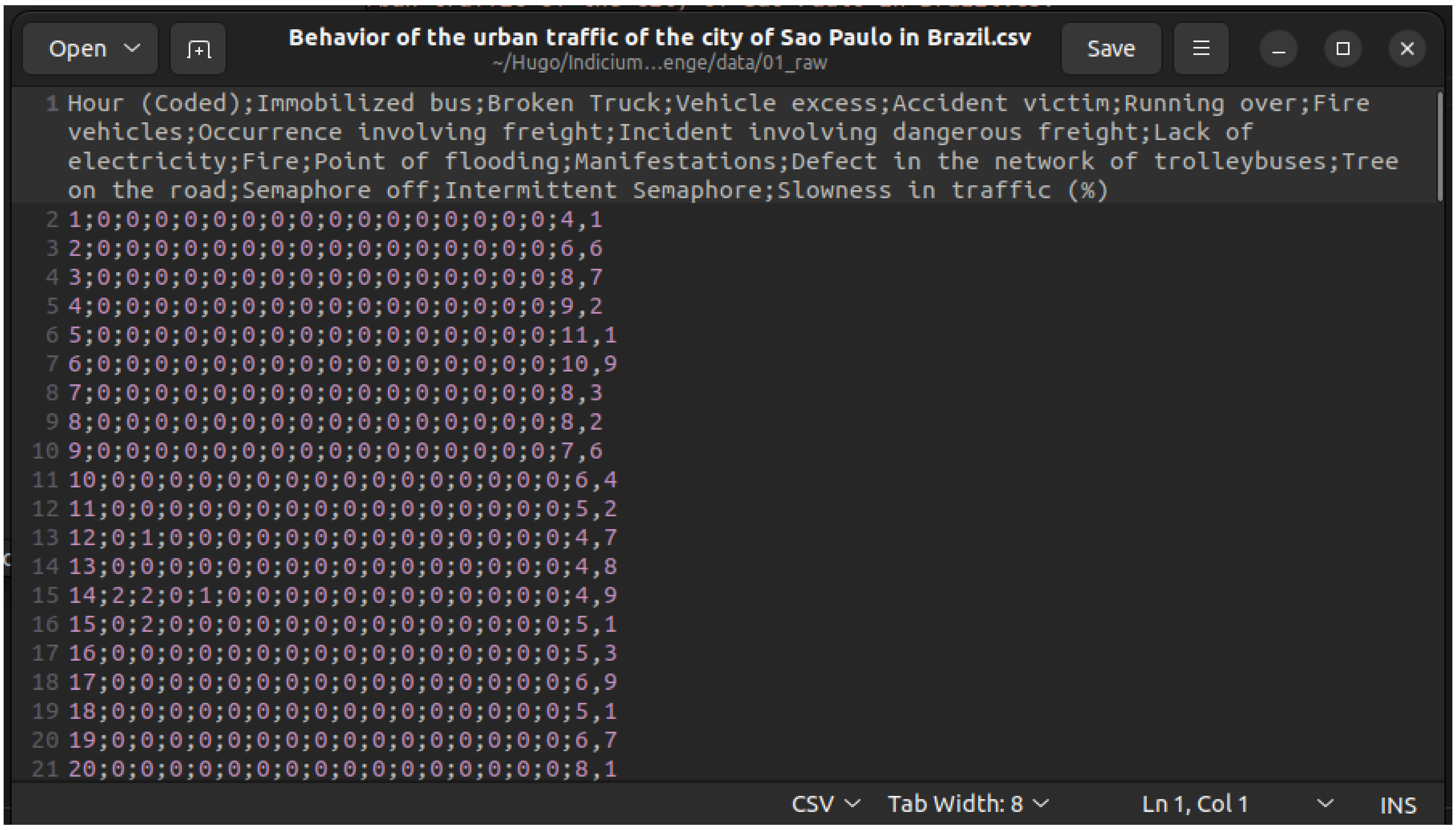
(Back to top)

Structuring a good solution begins with a good understanding and exploration of all elements avaliable and its constraints. Knowing your problem as deep as possible can help making the best decisions and avoid paliative measures and incomplete solutions.

Exploring the data involves listing all data avaliable and its integrity and assessing the relevance of each data in the solution's construction

Therefore, the first step is to verify the data integrity, checking if all data was correctly loaded into the work environment, verifying missing and/or corrupted data, confirming all data types are correct and so on.

Opening the csv file, the first issue encountered is the separators used. Traditionally, the column separator is a comma(,) and the decimal separator is a dot(.). In this file, the column separator is a semicolon(;) and the decimal separator is a dot(.). This could result in data corruption but it will be treated directly into the kedro catalog configuration so the data load function will work without problems.


Open
Behavior of the urban traffic of the city of Sao Paulo in Brazil.csv
~/Hugo/Indicium...enge/data/01_raw
Save

```
Hour (Coded);Immobilized bus;Broken Truck;Vehicle excess;Accident victim;Running over;Fire vehicles;Occurrence involving freight;Incident involving dangerous freight;Lack of electricity;Fire;Point of flooding;Manifestations;Defect in the network of trolleybuses;Tree on the road;Semaphore off;Intermittent Semaphore;Slowness in traffic (%)
1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;4,1
2;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;6,6
3;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;8,7
4;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;9,2
5;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;11,1
6;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;10,9
7;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;8,3
8;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;8,2
9;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;7,6
10;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;6,4
11;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;5,2
12;0;1;0;0;0;0;0;0;0;0;0;0;0;0;0;0;4,7
13;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;4,8
14;2;2;0;1;0;0;0;0;0;0;0;0;0;0;0;0;4,9
15;0;2;0;0;0;0;0;0;0;0;0;0;0;0;0;0;5,1
16;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;5,3
17;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;6,9
18;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;5,1
19;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;6,7
20;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;8,1
```

CSV Tab Width: 8 Ln 1, Col 1 INS

Figure-1: Raw CSV file

With the data poperly configured on kedro catalog, the next step is to load the data into the jupyter notebook and verify its integrity:

In [ ]:
```
%reload_kedro

# basic  imports used in data manipulation and visualization
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np


# Imports related to the interactive plots in the EDA section
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
```

```python
from bokeh.models import ColumnDataSource, HoverTool, ColorBar, LinearColorMapper
import bokeh.io
from bokeh.palettes import Plasma10


# Imports related to the separation of data into train and test parts
from sklearn.model_selection import train_test_split
import re


# Imports of the Regression modelling functions
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
import xgboost as xgb


# Import of the main metrics for the models performance
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from xgboost import plot_importance


# Import related to the tuning of the XGBoost model
from sklearn.model_selection import GridSearchCV

bokeh.io.output_notebook()
```

```
                     INFO     Resolved project path as: /home/hbeltrao/Hugo/Indicium/LightHouse/Data __init__.py:132
                              Science Track/kedro_tutorial/kedro_project_1/lighthouse-ds-challenge.
                              To set a different path, run '%reload_kedro <project_root>'
[02/20/23 18:32:40] INFO     Kedro project Lighthouse_ds_challenge                                  __init__.py:101
                     INFO     Defined global variable 'context', 'session', 'catalog' and            __init__.py:102
                              'pipelines'
                     WARNING  /home/hbeltrao/Hugo/Indicium/LightHouse/Data Science                   warnings.py:109
                              Track/kedro_tutorial/kedro_project_1/lighthouse-ds-challenge/.venv/lib
                              /python3.8/site-packages/bokeh/core/property/primitive.py:37:
                              DeprecationWarning: `np.bool8` is a deprecated alias for `np.bool_`.
                              (Deprecated NumPy 1.24)
                                bokeh_bool_types = (bool, np.bool8)
```

BokehJS 3.0.3 successfully loaded.

In [ ]:
```python
# verifying if all files were loaded into the framework
catalog.list()
```

```
[
    'MLPredictor',
    'raw_traffic_data',
    'train_features',
    'train_targets',
    'test_features',
    'test_targets',
    'fitted_regressor',
    'model_predictions',
    'parameters',
    'params:features',
    'params:target'
]
```

In [ ]:
```python
# Loading our dataset as raw data to start exploring
raw_traffic_data = catalog.load('raw_traffic_data')
raw_traffic_data.reset_index(inplace=True)

raw_traffic_data.head()
```

```
[02/20/23 18:32:41] INFO     Loading data from 'raw_traffic_data' (CSVDataSet)...           data_catalog.py:343
```

Out[ ]:

	index	Hour (Coded)	Immobilized bus	Broken Truck	Vehicle excess	Accident victim	Running over	Fire vehicles	Occurrence involving freight	Incident involving dangerous freight	Lack of electricity	Fire	Point of flooding	Manifestations	Defect in the network of trolleybuses	Tr t ro
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	4	5	0	0	0	0	0	0	0	0	0	0	0	0	0	

In [ ]:
```python
# Checking if all data was loaded corrected by counting the total rows and columns
# and comparing with the original data
raw_traffic_data.shape
```

```
(135, 19)
```

In [ ]:
```python
# Checking all column types and null values
raw_traffic_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135 entries, 0 to 134
Data columns (total 19 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   index                                  135 non-null    int64  
 1   Hour (Coded)                           135 non-null    int64  
 2   Immobilized bus                        135 non-null    int64  
 3   Broken Truck                           135 non-null    int64  
 4   Vehicle excess                         135 non-null    int64  
 5   Accident victim                        135 non-null    int64  
 6   Running over                           135 non-null    int64  
 7   Fire vehicles                          135 non-null    int64  
 8   Occurrence involving freight           135 non-null    int64  
 9   Incident involving dangerous freight   135 non-null    int64  
 10  Lack of electricity                    135 non-null    int64  
 11  Fire                                   135 non-null    int64  
 12  Point of flooding                      135 non-null    int64  
 13  Manifestations                         135 non-null    int64  
 14  Defect in the network of trolleybuses  135 non-null    int64  
 15  Tree on the road                       135 non-null    int64  
 16  Semaphore off                          135 non-null    int64  
 17  Intermittent Semaphore                 135 non-null    int64  
 18  Slowness in traffic (%)                135 non-null    float64
dtypes: float64(1), int64(18)
memory usage: 20.2 KB
```

In [ ]:
```python
# Printing the percentage of null values of each column
raw_traffic_data.isnull().sum()*100/len(raw_traffic_data)
```

```
index                                    0.0
Hour (Coded)                             0.0
Immobilized bus                          0.0
Broken Truck                             0.0
Vehicle excess                           0.0
Accident victim                          0.0
Running over                             0.0
Fire vehicles                            0.0
Occurrence involving freight             0.0
Incident involving dangerous freight     0.0
Lack of electricity                      0.0
Fire                                     0.0
Point of flooding                        0.0
Manifestations                           0.0
Defect in the network of trolleybuses    0.0
Tree on the road                         0.0
Semaphore off                            0.0
Intermittent Semaphore                   0.0
Slowness in traffic (%)                  0.0
dtype: float64
```

In [ ]:
```python
# Checking the distribution of the data in each column
raw_traffic_data.describe()
```

Out[ ]:

	index	Hour (Coded)	Immobilized bus	Broken Truck	Vehicle excess	Accident victim	Running over	Fire vehicles	Occurrence involving freight	Incident involving dangerous freight	Lack of electricity	Fire	Poir flooc
count	135.000000	135.00000	135.000000	135.000000	135.000000	135.000000	135.000000	135.000000	135.000000	135.000000	135.000000	135.000000	135.000
mean	67.000000	14.00000	0.340741	0.874074	0.029630	0.422222	0.118519	0.007407	0.007407	0.007407	0.118519	0.007407	0.118
std	39.115214	7.81789	0.659749	1.102437	0.170195	0.696116	0.346665	0.086066	0.086066	0.086066	0.504485	0.086066	0.712
min	0.000000	1.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	33.500000	7.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	67.000000	14.00000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
75%	100.500000	21.00000	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
max	134.000000	27.00000	4.000000	5.000000	1.000000	3.000000	2.000000	1.000000	1.000000	1.000000	4.000000	1.000000	7.000

The first analysis round showed that all data was loaded correctly (all rows and columns), is numeric and have no null values.

One possible issue is in the "Hour (Coded)" column, wich have values from 1 to 27 but a day only have 24 unique hours. Since the column is tagged as coded but no information about how it was efectivelly coded is available, those values cannot be counted as errors.

Next, will be shown the unique values distribution of each column data:

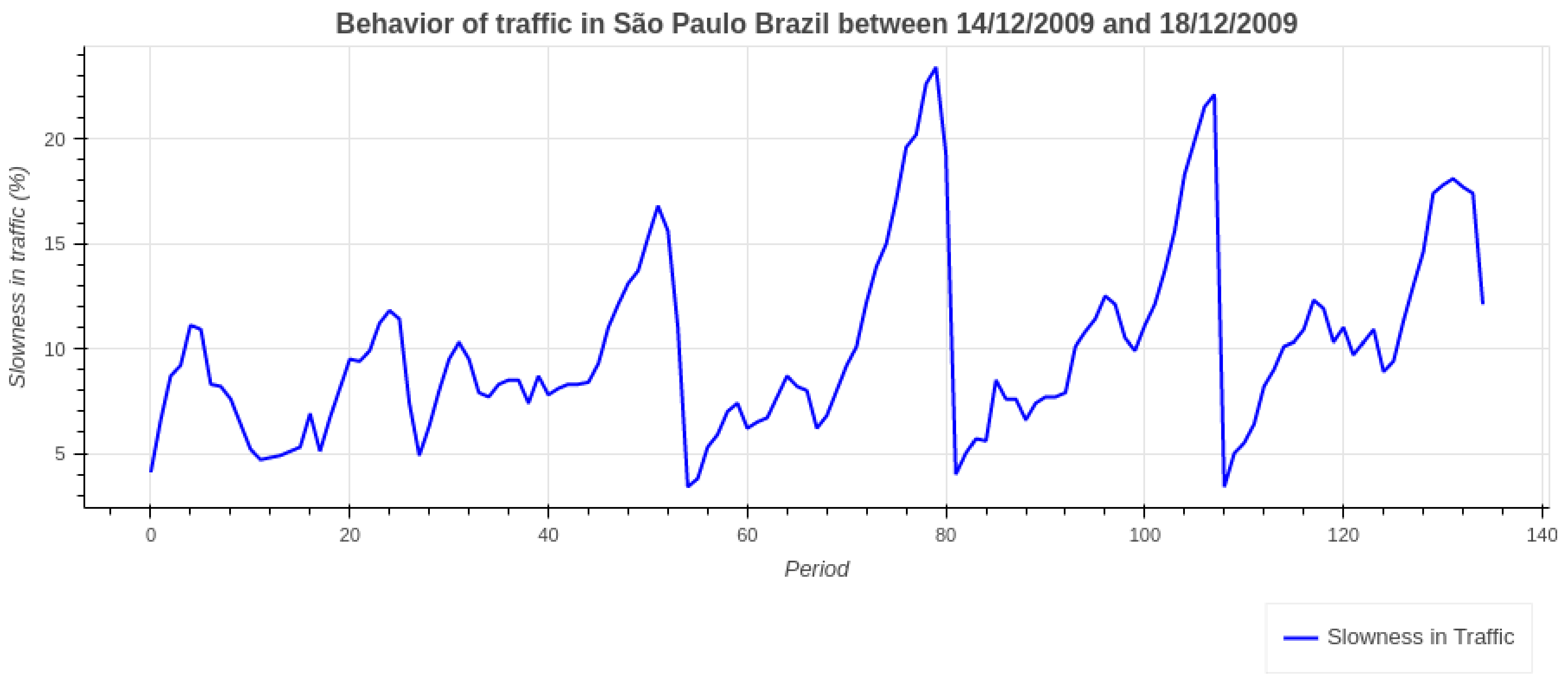
In [ ]:
```python
# Checking the ammount of unique values in each column
for col in raw_traffic_data.columns.values:
    print(col, " : ", len(raw_traffic_data[col].unique()))
```

```
index  :  135
Hour (Coded)  :  27
Immobilized bus  :  4
Broken Truck  :  6
Vehicle excess  :  2
Accident victim  :  4
Running over  :  3
Fire vehicles  :  2
Occurrence involving freight  :  2
Incident involving dangerous freight  :  2
Lack of electricity  :  5
Fire  :  2
Point of flooding  :  4
Manifestations  :  2
Defect in the network of trolleybuses  :  5
Tree on the road  :  2
Semaphore off  :  4
Intermittent Semaphore  :  2
Slowness in traffic (%)  :  83
```

Since all data is numeric and no other information was given for the columns metadata, it is not possible to interpret any column as purely categorical, counting some specific event or represent some mathematical or physical greatness. Some columns are booleans but coding them as booleans does not improve the modelling process, so they will be kept as numeric.

Assuming the data is ordered cronologically by indexes, plotting a line will show the behavior of the traffic slowness on a time passage base and might highlight some sazonal behavior:

In [ ]:
```
# Converting the dataset to be used with bokeh

data_source = ColumnDataSource(raw_traffic_data)

# Creating and configuring the plotting figure
p = figure(title="Behavior of traffic in São Paulo Brazil between 14/12/2009 and 18/12/2009"
           , x_axis_label="Period", y_axis_label="Slowness in traffic (%)"
           , width=900, height=400, tools='', toolbar_location=None)


# Making axis and title adjustments
p.title.align = 'center'
p.title.text_font_size='12pt'


# Lineplot creation
a=p.line(x='index', y='Slowness in traffic (%)', source = data_source,
        color= 'blue', line_width=2, legend_label="Slowness in Traffic")


# Adding the hovertool
p.add_tools(HoverTool(
    tooltips=[
        ( 'index',   '$index'),
        ( 'Hour Coded',   '@{Hour (Coded)}'),
        ( 'Slowness in traffic (%)', '@{Slowness in traffic (%)}{0.0}')
    ],

    mode='vline',
    renderers=[a]
))


# Adding legend outside the plot
p.add_layout(p.legend[0], 'below')

show(p)
```

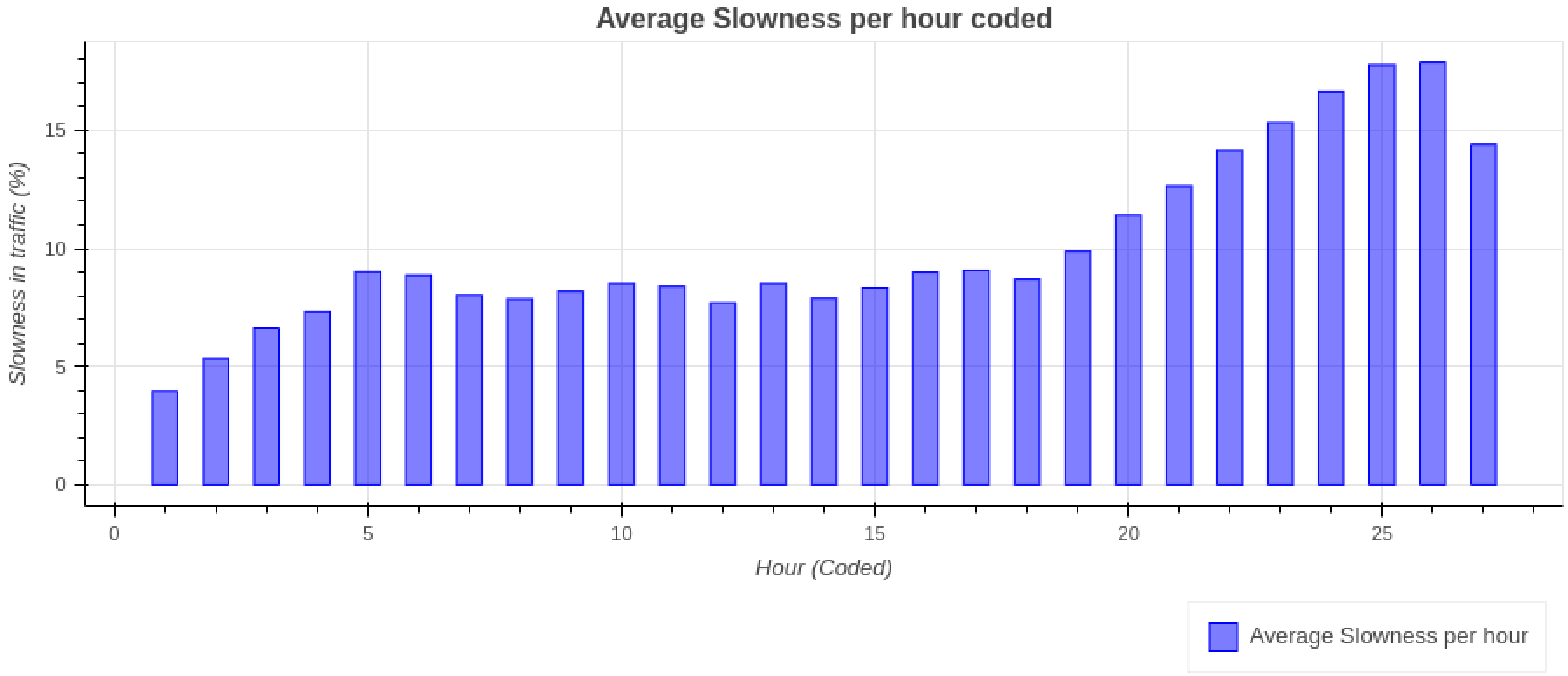


From the time series, the behavior of the traffic slowness is quite related to the *Hour (Coded)* column and some spikes can bem identified.

another way to view this relationship is to plot the average slowness in each hour:

In [ ]:
```
# Getting the average slowness per hour coded, to be plotted in a bar chart
data = raw_traffic_data.groupby(['Hour (Coded)'])['Slowness in traffic (%)'].mean().to_frame()
```

In [ ]:
```
data_source = ColumnDataSource(data)

p = figure(title="Average Slowness per hour coded", x_axis_label="Hour (Coded)",
           y_axis_label="Slowness in traffic (%)", width=900, height=400, tools='',
           toolbar_location=None)


# Making axis and title adjustments
p.title.align = 'center'
p.title.text_font_size='12pt'


# Vertical bar creation
a=p.vbar(x='Hour (Coded)', top='Slowness in traffic (%)', source = data_source,
                color= 'blue', width=0.5, fill_alpha=0.5, legend_label="Average Slowness per hour")


# Adding the hovertool
p.add_tools(HoverTool(
    tooltips=[
        ( 'Hour Coded',   '@{Hour (Coded)}'),
        ( 'Average Slowness (%)', '@{Slowness in traffic (%)}{0.0}')
    ],

    mode='vline',
    renderers=[a]
))



# Adding legend outside the plot
p.add_layout(p.legend[0], 'below')

show(p)
```



With this relationship explored, the next step is to explore the rest of the features present in the dataset.

This analysis will be performed in the feature engineering section, in order to select the most fit features to be used in the modeling phase.

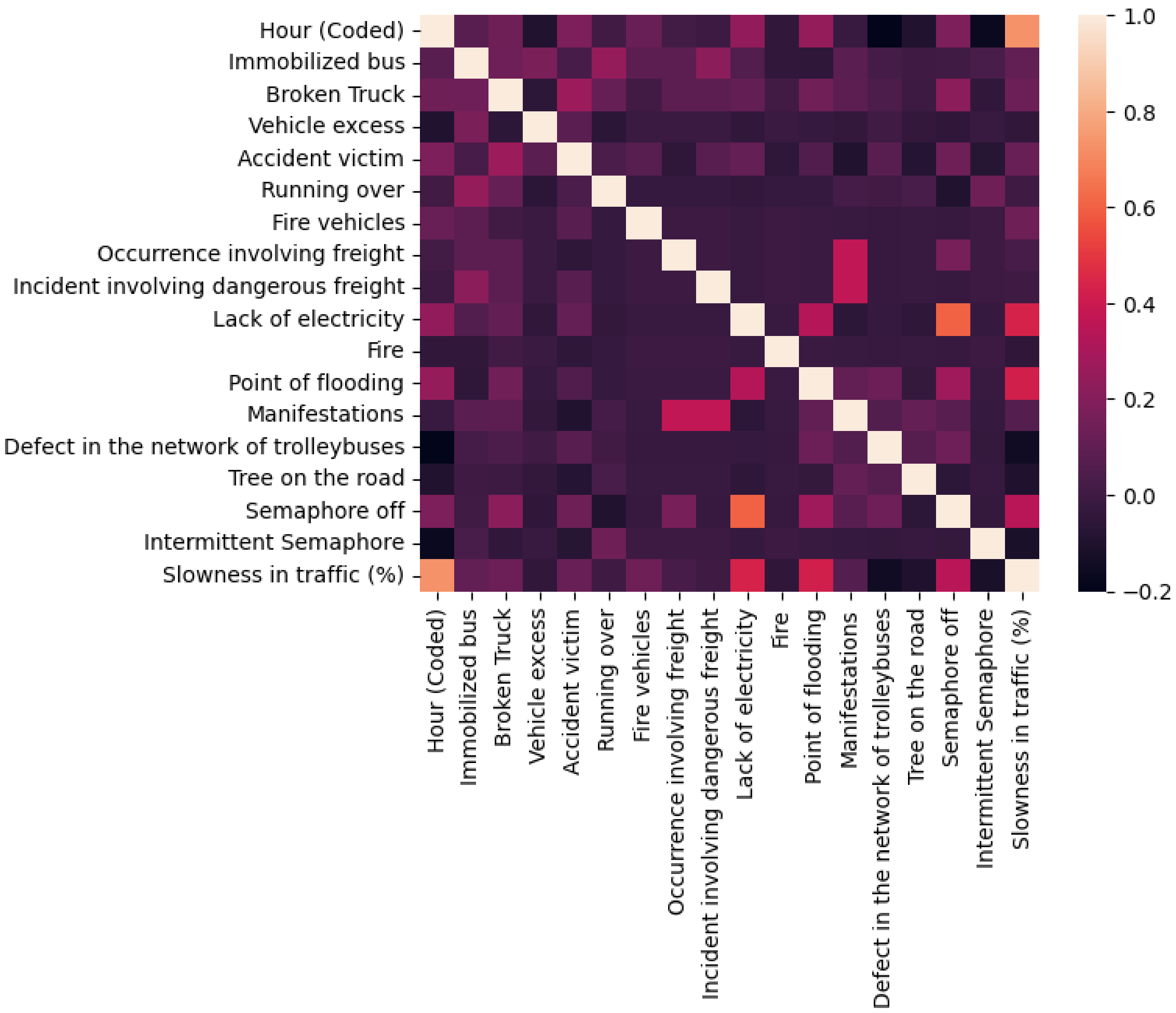
3.Feature Engineering

(Back to top)

It would be advised to perform some level of analysis in all features, to verify patterns and peculiarities. Since there are 17 features, a good way to optimize this work is to verify the correlation between each feature and the target, in this case *slowness in traffic (%)*.

An quick way to visualize the correlation between features is to plot the correlation matrix as a heatmap, as shown below:

In [ ]:
```
sns.heatmap(raw_traffic_data.drop(['index'], axis=1).corr())
```



Although a heatmap can help visualize the correlation between features, it can be inefficient to identify the most impactful features. A faster way to get this information is to extract only the correlation between all features and the target. Below will be shown a list with this correlations and the most relevant ones will be separated to be analyzed and eventually used in the modelling phase:

In [ ]:

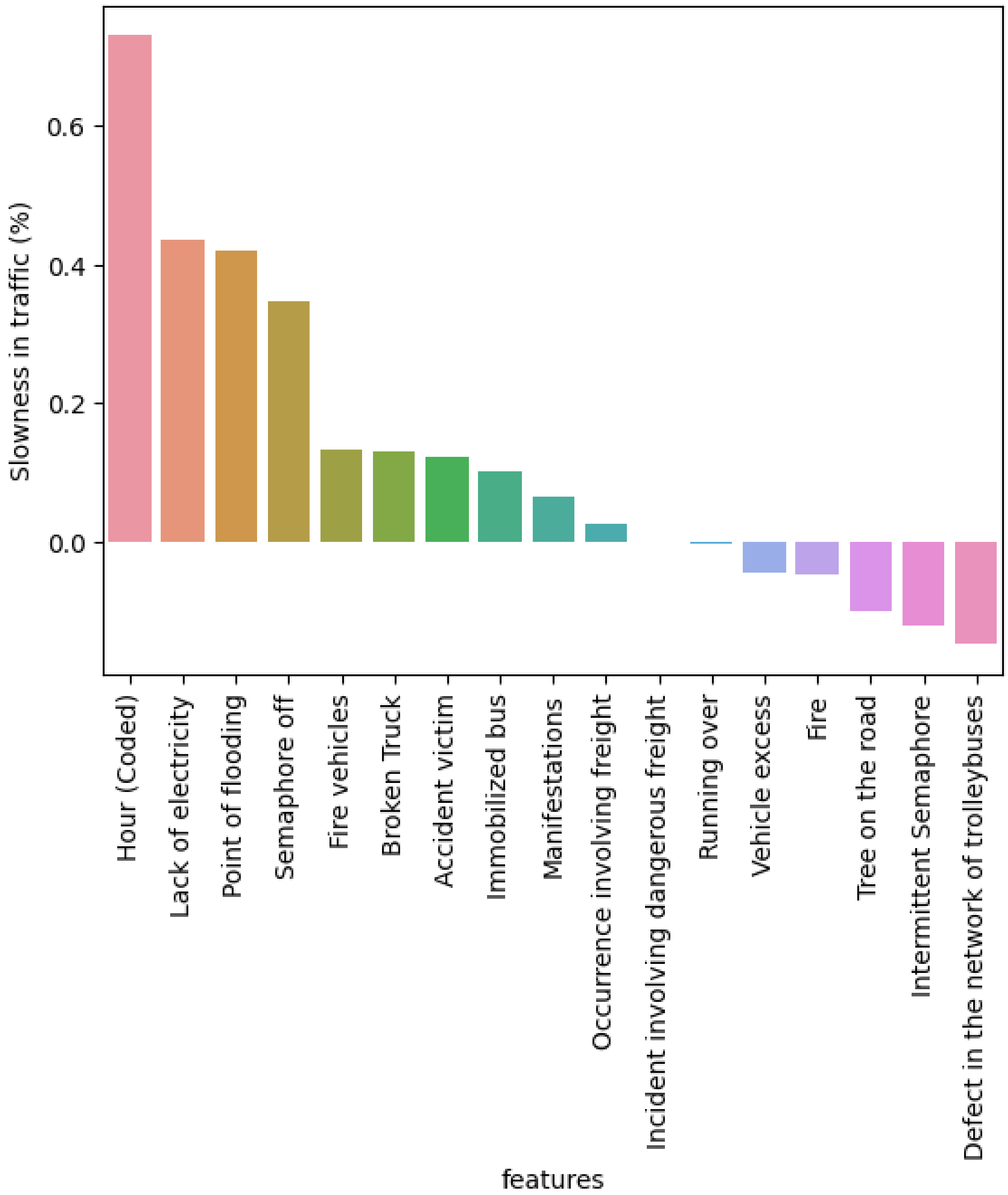
```
correlation_matrix = raw_traffic_data.drop(['index'], axis=1).corr()
corr = (correlation_matrix['Slowness in traffic (%)']
        .to_frame()
        .reset_index(names='features')
        .sort_values(by='Slowness in traffic (%)', ascending=False))
corr.drop(index=17, axis=0, inplace=True)
corr
```

Out[ ]:

	features	Slowness in traffic (%)
0	Hour (Coded)	0.729962
9	Lack of electricity	0.436569
11	Point of flooding	0.420016
15	Semaphore off	0.347242
6	Fire vehicles	0.134103
2	Broken Truck	0.131998
4	Accident victim	0.121730
1	Immobilized bus	0.101143
12	Manifestations	0.066377
7	Occurrence involving freight	0.026791
8	Incident involving dangerous freight	0.000957
5	Running over	-0.001133
3	Vehicle excess	-0.045297
10	Fire	-0.046737
14	Tree on the road	-0.098489
16	Intermittent Semaphore	-0.119942
13	Defect in the network of trolleybuses	-0.147035

In [ ]:

```
sns.barplot(data=corr, x='features', y='Slowness in traffic (%)')
plt.xticks(rotation=90)
plt.show()
```



The graphic shows four features with significant inpact to the slowness and five features that actually reduce slowness when present, although not very impactfully.

Before rule out the less impactfull features, it is good to verify the incidence rate of each feature

In [ ]:
```
# Calculating the incidence rate of each feature

incidence_x_correlation = {}
for col in raw_traffic_data.drop(['index'], axis=1).columns.values:
    incidence_rate = round(sum(raw_traffic_data[col] > 0)/len(raw_traffic_data[col])*100 ,2)
    print(col, incidence_rate)
    incidence_x_correlation[col] = {}
    incidence_x_correlation[col]['incidence_rate'] = incidence_rate
    incidence_x_correlation[col]['slowness_in_traffic'] = (corr[corr['features']==col]['Slowness in traffic (%)']
                                                           .values)
```

```
Hour (Coded) 100.0
Immobilized bus 25.93
Broken Truck 52.59
Vehicle excess 2.96
Accident victim 31.85
Running over 11.11
Fire vehicles 0.74
Occurrence involving freight 0.74
Incident involving dangerous freight 0.74
Lack of electricity 7.41
Fire 0.74
Point of flooding 4.44
Manifestations 5.19
Defect in the network of trolleybuses 14.81
Tree on the road 4.44
Semaphore off 9.63
Intermittent Semaphore 1.48
Slowness in traffic (%) 100.0
```

In [ ]:
```
# Comparing the incidence_rate to the correlation index to verify possible dropable features
incidence_x_correlation_df = pd.DataFrame(incidence_x_correlation)
_incidence_x_correlation_df = incidence_x_correlation_df.transpose().reset_index(names='feature')
_incidence_x_correlation_df.sort_values(by='slowness_in_traffic', ascending=False)
```

```
[02/20/23 18:32:42] WARNING  /home/hbeltrao/Hugo/Indicium/LightHouse/Data Science           warnings.py:109
                             Track/kedro_tutorial/kedro_project_1/lighthouse-ds-challenge/.venv/lib
                             /python3.8/site-packages/pandas/core/sorting.py:438:
                             DeprecationWarning: The truth value of an empty array is ambiguous.
                             Returning False, but in future this will result in an error. Use
                             `array.size > 0` to check that an array is not empty.
                               indexer = non_nan_idx[non_nans.argsort(kind=kind)]
```

Out[ ]:

	feature	incidence_rate	slowness_in_traffic
0	Hour (Coded)	100.0	[0.7299622611542895]
9	Lack of electricity	7.41	[0.4365694933961246]
11	Point of flooding	4.44	[0.4200156746993659]
15	Semaphore off	9.63	[0.3472417166131551]
6	Fire vehicles	0.74	[0.13410263355364693]
2	Broken Truck	52.59	[0.1319979021813809]
4	Accident victim	31.85	[0.12173047795731425]
1	Immobilized bus	25.93	[0.10114325874482982]
12	Manifestations	5.19	[0.06637673323323363]
7	Occurrence involving freight	0.74	[0.026791085956930555]
8	Incident involving dangerous freight	0.74	[0.0009568244984617961]
5	Running over	11.11	[-0.0011329304587513783]
3	Vehicle excess	2.96	[-0.04529666918256658]
10	Fire	0.74	[-0.046737196655634354]
14	Tree on the road	4.44	[-0.09848881067172303]
16	Intermittent Semaphore	1.48	[-0.11994230722260359]
13	Defect in the network of trolleybuses	14.81	[-0.1470352468035566]
17	Slowness in traffic (%)	100.0	[]

From the incidence_rate x slowness_in_traffic, it is possible to identify some columns with both loow incidence rate and impact on the traffic slowness, so those features could be removed from the model with less risk of harming the model.

With this strategy, the features removed are:

feature	incidence_rate	slowness_in_traffic
Fire vehicles	0.74	0.1341
Manifestations	5.19	0.0663
Occurrence involving freight	0.74	0.0267
Incident involving dangerous freight	0.74	0.0009
Running over	11.11	-0.0011
Vehicle excess	2.96	-0.0452
Fire	0.74	-0.0467
Tree on the road	4.44	-0.0984
Intermittent Semaphore	1.48	-0.1199

With this, the final feature list to be used in the modeling phase will be :

- Hour (Coded)
- Immobilized bus
- Broken Truck
- Accident victim
- Lack of electricity
- Point of flooding
- Defect in the network of trolleybuses
- Semaphore off

4.Modelling

(Back to top)

With all data analyzed and the proper features selectec. It is time to create a preditcion model to effectivelly predict the slowness in traffic in São Paulo city.

Since the problem is resumed to estimate a numeric value based on other numeric values, a regression model might be the best fit to perform this task.

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable and one or more independent variables.

4.1. Regression Models

There are several types of regression techniques, suited to different data nature. To choose the most suited to our dataset, a brief introduction to some regression models is appreciated.

4.1.1. Linear Regression

Linear Regression assume a linear relation between the features and the target variable. The model consists in a simple line of equation:

$$Y_i = m * X_i + C + e_i$$

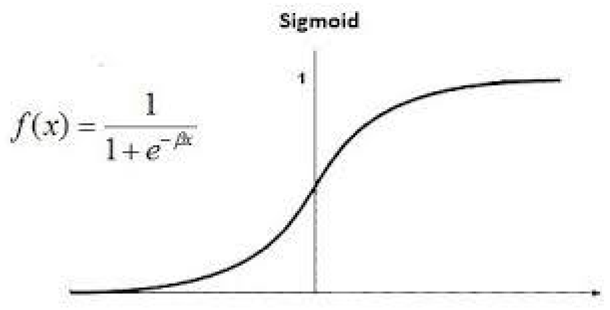
Where:

Y = variable to be predicted
X = feature value
C = intercept
m = slope of the curve
e = error function

It can be used with one or more independent variables (features). Due to its linear nature, evidently this type of regression is not suited to be used to model non-linear relationships.

4.1.2. Logistic Regression

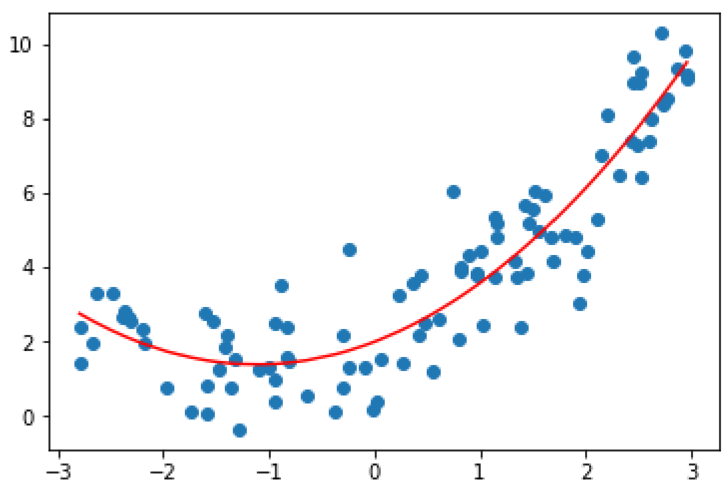
Logistic Regression is a technique used to predict discreet values (i.e. categorical or boolean). In essence, it models a Sigmoid curve in wich the target value will be setted to 0 or 1 depending on its value and the threshold value:



The main downside of this technique is it is limited to discreet features and target. Another limitation is that it assume linearity between the target and the features.

4.1.3. Polynomial Regression

Is similar to the Linear Regression, but the resulting line is curved and better represents non-linear relationships:



4.1.4 Ridge Regression

Is a technique used when there is multicolinearity (high correlation) between the features. It intentionally bias the regression estimates by applying a penalty function.

4.2. Preparing the data for modelling

with all analysis and transformations done and all important features selected, it is time to structure the final dataset to be modelled.

This way, the original dataset will be sliced to contain only the selected features and then will be separated into training and test parts.

From then, the model will be fitted based on the training dataset and after that, validated with the test dataset.

4.2.1. Organizing and separating data for modelling

Below is the code to filter the selected features and split the dataset into train and test parts:

In [ ]:
```
# Listing all features. the target column can be ommited here because it will be put into a separated variable.
features = [ 'Hour (Coded)', 'Immobilized bus', 'Broken Truck'
            , 'Accident victim', 'Lack of electricity',
            'Point of flooding', 'Defect in the network of trolleybuses'
            , 'Semaphore off']


# Creating the dataset with all features data
feature_selected_traffic_data = raw_traffic_data[features]
```

```
# Dataset containing the data to be predicted
target_data = raw_traffic_data["Slowness in traffic (%)"]
```

In [ ]:
```
# Separating both datasets into train and test parts

X_train, X_test, y_train, y_test = train_test_split(feature_selected_traffic_data
                                                    , target_data
                                                    , test_size=0.2
                                                    , random_state=33)
```

4.2.2. Training and comparing some models

A practical way to choose the best modelling technique for this problem is to test and verify the precision of some techniques.

In this session, some models will be trained to be compared with each other.

The models that will be trained are Linear, Polynomial, Ridge and a XGBoost regressor wich is a gradient boost regressor that usually is very efficient in general purpose problems.

The metric used to compare the models is a combination of Mean Absolute Error (MAE) and Rooted Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

The RMSE metric is good to penalyze bigger erros (such as outliers) and a combination with the MAE can help diagnose the excessive presence of those outliers.

4.2.2.1. Linear Regression Code

In [ ]:
```
# Declaring the regressor element
linear_regressor = LinearRegression()

# Fitting the regressor with the train data
fitted_lin_reg = linear_regressor.fit(X_train, y_train)

# Calculating the score for the train and test datasets
lin_reg_score_train = fitted_lin_reg.score(X_train, y_train)
lin_reg_score_test = fitted_lin_reg.score(X_test, y_test)
print("Linear Regression Train Score:", lin_reg_score_train
      ,"\n"
      , "Linear Regression Test Score:", lin_reg_score_test)
```

```
Linear Regression Train Score: 0.6716778409881007
 Linear Regression Test Score: 0.4850416449090481
```

In [ ]:
```
linear_predict = fitted_lin_reg.predict(X_test)
rmse = np.sqrt(MSE(y_test, linear_predict))
mae = MAE(y_test, linear_predict)
print(rmse, "\n", mae)
```

```
2.629481732399216
 2.0836367218565304
```

4.2.2.2. Polynomial Regression Code

In [ ]:
```
# Creating the polynomial features to be used to generate the polynomial curve
# and adjusting the datasets to the same format
poly = PolynomialFeatures(degree = 4)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.fit_transform(X_test)

# Fitting the model
polynomial_regressor = LinearRegression()
fitted_polynomial_regressor = polynomial_regressor.fit(X_train_poly, y_train)

# Calculating the score for the train and test datasets
poly_reg_score_train = fitted_polynomial_regressor.score(X_train_poly, y_train)
poly_reg_score_test = fitted_polynomial_regressor.score(X_test_poly, y_test)
```

In [ ]:
```
print("Polynomial Regression Train Score:", poly_reg_score_train
      ,"\n"
      , "Polynomial Regression Test Score:", poly_reg_score_test)
```

```
Polynomial Regression Train Score: 0.9409161930936613
 Polynomial Regression Test Score: -2205.9674638592974
```

In [ ]:
```
poly_predict = fitted_polynomial_regressor.predict(X_test_poly)
rmse = np.sqrt(MSE(y_test, poly_predict))
mae = MAE(y_test, poly_predict)
print(rmse, "\n", mae)
```

```
172.14010712619387
 72.80118255164638
```

4.2.2.3. Ridge Regression Code

In [ ]:
```
# Creating the Ridge classifier
ridge_regressor = Ridge(alpha=1.0)

# Fitting the model
fitted_ridge_regressor = ridge_regressor.fit(X_train, y_train)

# Calculating the scores
ridge_reg_score_train = fitted_ridge_regressor.score(X_train, y_train)
ridge_reg_score_test = fitted_ridge_regressor.score(X_test, y_test)
print("Ridge Regression Train Score:", ridge_reg_score_train
      ,"\n"
      , "Ridge Regression Test Score:", ridge_reg_score_test)
```

```
Ridge Regression Train Score: 0.6716331076194091
 Ridge Regression Test Score: 0.4866007712119762
```

In [ ]:
```
ridge_predict = fitted_ridge_regressor.predict(X_test)
rmse = np.sqrt(MSE(y_test, ridge_predict))
mae = MAE(y_test, ridge_predict)
print(rmse, "\n", mae)
```

```
2.6254981073808707
 2.07638864425551
```

4.2.2.4. XGBoost Regression Code

In [ ]:
```
# Creating the xgboost regressor
xgb_regressor = xgb.XGBRegressor(objective='reg:squarederror', seed=123)

fitted_xgb_regressor = xgb_regressor.fit(X_train, y_train)

xgb_predict = fitted_xgb_regressor.predict(X_test)
rmse = np.sqrt(MSE(y_test, xgb_predict))
mae = MAE(y_test, xgb_predict)
print(rmse, "\n", mae)
```

```
2.7547446307777044
 2.208144795453107
```

4.3. Choosing and building the final model

Judging all models tested before, although the linear, ridge and xgboost models presented similar root mean squared errors (RMSE), there are some important characteristics that need to be accounted for choosing the final model.
First, linear models assume that all relationships betweeen featuires and target are linear, wich is not necessarily true. It is also very simple and usually have a broad error band.
The Ridge regression can help with overfitting but it also inputs bias to the result wich can milead the result.
The XGBoost model is based on decision tree, wich is more precise than linear regression and it is still very fast. In this scenario with limited data samples, the XGBoost model could be overfitted, but with long usage, the dataset could grow enough to benefit this technique. Also XGBoost still have some parameter tuning that can be made in order to improve its performance.

With that in mind, the final model will be a tuned version of the XGBoost model, that will be adjusted in the following steps:

In [ ]:
```
# Logic to test and identify the best parameters for the regressor
param_grid = {"max_depth":    [2, 4, 6],
              "n_estimators": [100, 300, 500],
              "learning_rate": [0.01, 0.015]}

search = GridSearchCV(xgb_regressor, param_grid, cv=5).fit(X_train, y_train)
```

In [ ]:
```
# Input the best parameters into the regressor and fit it
tuned_xgb_regressor=xgb.XGBRegressor(learning_rate = search.best_params_["learning_rate"],
                           n_estimators  = search.best_params_["n_estimators"],
                           max_depth     = search.best_params_["max_depth"])

tuned_xgb_regressor.fit(X_train, y_train)
```

In [ ]:
```
# Calculating the RMSE of the tuned xgb model using the test records
tuned_xgb_predict = tuned_xgb_regressor.predict(X_test)
rmse = np.sqrt(MSE(y_test, tuned_xgb_predict))
mae = MAE(y_test, tuned_xgb_predict)
print(rmse, "\n", mae)
```

The final model have a RMSE of 2.3593, wich is 11 % lower than the second best model(ridge model).

One last piece of information about the model is the importance of each feature in the prediction algorithm:

In [ ]:
```
plt.style.use('fivethirtyeight')
plt.rcParams.update({'font.size': 16})
```

```
fig, ax = plt.subplots(figsize=(12,6))
plot_importance(tuned_xgb_regressor, max_num_features=12, ax=ax)
plt.show()
```

Clearlym the *Hour (Coded)* was the most impactful feature for the model.

5. Model Evaluation

(Back to top)

One way to show the model performance is to plot the predictions and the real values from the test samples, alongside with the residue distribution:

In [ ]:
```
# Creatint the table with predictions and real values to be plotted with the tested data
tuned_xgb_predict_df = pd.DataFrame(tuned_xgb_predict).rename(columns={0:'Predictions'})
predictions_x_real_test = pd.DataFrame(y_test).reset_index()

predictions_x_real_test["Predictions"] = tuned_xgb_predict_df
predictions_x_real_test["Residues"] = (predictions_x_real_test["Slowness in traffic (%)"]
                                        - predictions_x_real_test["Predictions"])
predictions_x_real_test.rename(columns={"index":"original_index"}, inplace=True)
predictions_x_real_test
```

In [ ]:
```
# Creatint the table with predictions and real values to be plotted with the training data
training_predictions = tuned_xgb_regressor.predict(X_train)

training_predictions_df = pd.DataFrame(training_predictions).rename(columns={0:'Predictions'})

predictions_x_real_train = pd.DataFrame(y_train).reset_index()
predictions_x_real_train["Predictions"] = training_predictions_df
predictions_x_real_train["Residues"] = (predictions_x_real_train["Slowness in traffic (%)"]
                                        - predictions_x_real_train["Predictions"])
predictions_x_real_train.rename(columns={"index":"original_index"}, inplace=True)
predictions_x_real_train
```

In [ ]:
```
data_source = ColumnDataSource(predictions_x_real_train)

# Creating and configuring the plotting figure
p = figure(title="Predictions x Real - Train Data", x_axis_label="index",
           y_axis_label="Slowness in traffic (%)", width=990, height=400, tools='',
           toolbar_location=None)


# Making axis and title adjustments
p.title.align = 'center'
p.title.text_font_size='12pt'



# Lineplot creation
a=p.line(x='index', y='Slowness in traffic (%)', source = data_source,
        color= 'blue', line_width=2, legend_label="real")

b=p.line(x='index', y='Predictions', source=data_source, color="red",
        line_width=1, legend_label="predictions", line_dash='dashed')


# Adding the hovertool
p.add_tools(HoverTool(
    tooltips=[
        ( 'index',   '$index'),
        ( 'Real Value', '@{Slowness in traffic (%)}{0.0}'),
        ( 'Prediction', '@Predictions')
    ],

    mode='vline',
    renderers=[a]
))



# Adding legend outside the plot
p.add_layout(p.legend[0], 'below')

show(p)
```

In [ ]:
```
data_source = ColumnDataSource(predictions_x_real_test)

# Creating and configuring the plotting figure
p = figure(title="Predictions x Real - Test Data", x_axis_label="index",
           y_axis_label="Slowness in traffic (%)", width=990, height=400, tools='',
           toolbar_location=None)


# Making axis and title adjustments
p.title.align = 'center'
```

```
p.title.text_font_size='12pt'



# Lineplot creation
a=p.line(x='index', y='Slowness in traffic (%)', source = data_source,
        color= 'blue', line_width=2, legend_label="real")

b=p.line(x='index', y='Predictions', source=data_source, color="red",
        line_width=1, legend_label="predictions", line_dash='dashed')


# Adding the hovertool
p.add_tools(HoverTool(
    tooltips=[
        ( 'index',   '$index'),
        ( 'Real Value', '@{Slowness in traffic (%)}{0.0}'),
        ( 'Prediction', '@Predictions')
    ],

    mode='vline',
    renderers=[a]
))



# Adding legend outside the plot
p.add_layout(p.legend[0], 'below')

show(p)
```

In [ ]:

```
sns.histplot(data=predictions_x_real_train, x="Residues", bins=30)
plt.title("Residue Distribution for training data")
plt.show()
```

In [ ]:

```
sns.histplot(data=predictions_x_real_test, x="Residues", bins=30)
plt.title("Residue Distribution for testing data")
plt.show()
```

The prediction line have a similar shape as the real values but with some outliers, wich could indicate some level of overfitting.

The residues distribution for linear relationships is a bell shape, wich is the result obtained in the training data. Since there is too few samples for testing, it's shape is not well defined.

Looking at the RMSE value, its interpretation is that with a RMSE of 2.35, it is expected for the prediction value to be 2.35 above or below the real observation value.
Since the average value of the slowness in traffic in the sample is 10.05, the RMSE represents a deviation of 23% of the real value, wich can be significant.

With all preparations and analysis done, it is time to structure the final model into the kedro framework and prepare the deployment strategy.

6. Deployment

(Back to top)

Finally, the last part of the project is to structure it in a way that is easier to maintain and to be productized.

The first step is to build the project on the kedro framework, since it is easy to structure and deploy.

The last step is to build a way for the model to be consumed. This project will not implement the deploy strategy but will suggest some solutions.

6.1. Building the whole project on the kedro framework

The last step to productize the machine learning model is to build it on the kedro framework, in order to facilitate maintenance, documentation and deployment.

All files and documentation referred to the kedro implementation of this project are on the project repository (accessed via link below)

https://github.com/hbeltrao/lighthouse_ds_challenge

Instructions on how to clone and run the project are on the repository README.md file.

6.2. Deploy strategies

The best way to consume the data generated from the model would be an API. In this form, the model could be hosted in a web interface and the user could input
the data manually in some form interface and export the result to a file and download it.

The API solution could also be accessed using a script to generate periodic predictions and save them into a cloud provider.

The kedro fast-api module is an easy method to implement that.

7.Conclusions

(Back to top)

Summarizing everything, the initial data represent some characteristics that impact the traffic flow but some context and metadata were ommited, wich could help improve the solution performance.

The temporal window of the data comprises of only 4 days wich can limit significantly the model coverage, therefore the data have bias torwards some seasonality or be completely blind to it, among other limitations.

The initial dataset possess 18 features to analyze but in the end only 8 were used and even between those 8, the *Hour (Coded)* have significant more impact than the rest of the features.

Also, even after model optimzation, the error ended up with significant value.

With all limitations above, the model still prove to be efficient in traffic slowness prediction and the deployment strategy suggested is flexible enough to make the solution viable.

In [ ]:

```

```