# Traffic prediction - machine learning models (regression)

Robert Kowalczyk 397108  
Jarosław Leski 411174

# 1. Prepare necessary libraries

import warnings  
warnings.filterwarnings("ignore")  
pd.set\_option("display.max\_columns",100)  
  
# Data wrangling  
import pandas as pd  
import numpy as np  
from datetime import datetime as dt   
  
# Visualizations  
import seaborn as sns  
import matplotlib.pyplot as plt   
%matplotlib inline  
  
# Modelling with scikit-learn  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler  
from sklearn.feature\_selection import VarianceThreshold  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_absolute\_percentage\_error, mean\_absolute\_error, r2\_score, make\_scorer  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV  
from sklearn.pipeline import Pipeline  
from sklearn.linear\_model import ElasticNet  
from sklearn.linear\_model import ElasticNetCV  
from sklearn.svm import SVR  
from sklearn.neighbors import KNeighborsRegressor  
from sklearn.linear\_model import TweedieRegressor  
from sklearn.ensemble import GradientBoostingRegressor  
from sklearn.inspection import permutation\_importance  
import optuna  
import sklearn

# 2. Loading data and cleaning

Your task is to apply various ML algorithms (see the rules below) to build a model **explaining the traffic** on one of the highways for one-hourly intervals based on the **training sample** and generate predictions for **all observations** from the **test sample**.

The dataset includes the following columns:

* *date\_time* – date and time (1 hourly interval)
* *weather\_general* – general short description of the current weather with the following levels: Clear, Clouds, Drizzle, Fog, Haze, Mist, Rain, Smoke, Snow, Squall, Thunderstorm
* *weather\_detailed* – more detailed description of the current weather with the following levels: broken clouds, drizzle, few clouds, fog, freezing rain, haze, heavy intensity drizzle, heavy intensity rain, heavy snow, light intensity drizzle, light intensity shower rain, light rain, light rain and snow, light shower snow, light snow, mist, moderate rain, overcast clouds, proximity shower rain, proximity thunderstorm, proximity thunderstorm with drizzle, proximity thunderstorm with rain, scattered clouds, shower drizzle, shower snow, sky is clear, sleet, smoke, snow, squalls, thunderstorm, thunderstorm with drizzle, thunderstorm with heavy rain, thunderstorm with light drizzle, thunderstorm with light rain, thunderstorm with rain, very heavy rain
* *clouds\_coverage* – percentage of sky covered by the clouds in the hourly interval
* *temperature* – average temperature in the hourly interval (in Celsius degrees)
* *rain\_mm* – amount of rain that occurred in the hourly interval (in mm)
* *snow\_mm* – amount of snow that occurred in the hourly interval (in mm)
* *traffic* – the amount of traffic in the hourly interval (outcome variable)

Let's load the data and look at the first five observations

traffic = pd.read\_csv('traffic\_train.csv')  
  
traffic.head()

date\_time weather\_general weather\_detailed clouds\_coverage\_pct \  
0 2014-10-01 00:00:00 Clear sky is clear 1   
1 2014-10-01 01:00:00 Clear sky is clear 1   
2 2014-10-01 02:00:00 Clear sky is clear 1   
3 2014-10-01 03:00:00 Clear sky is clear 1   
4 2014-10-01 04:00:00 Clear sky is clear 1   
  
 temperature rain\_mm snow\_mm traffic   
0 11.5 0.0 0.0 508   
1 10.3 0.0 0.0 323   
2 8.0 0.0 0.0 274   
3 7.9 0.0 0.0 372   
4 6.4 0.0 0.0 812

traffic.describe()

clouds\_coverage\_pct temperature rain\_mm snow\_mm \  
count 29701.000000 29701.000000 29701.000000 29701.000000   
mean 50.210027 7.500451 0.509914 0.000361   
std 38.657342 13.769115 57.058322 0.010403   
min 0.000000 -273.100000 0.000000 0.000000   
25% 1.000000 -1.300000 0.000000 0.000000   
50% 64.000000 8.300000 0.000000 0.000000   
75% 90.000000 18.300000 0.000000 0.000000   
max 100.000000 35.100000 9831.300000 0.510000   
  
 traffic   
count 29701.000000   
mean 3228.196761   
std 1989.059081   
min 0.000000   
25% 1159.000000   
50% 3309.000000   
75% 4918.000000   
max 7263.000000

In the training set we noticed two very interesting situations:

* *temperature* variable contains a minimum value of -273.1 degrees celsius, which we believe is an incorrect value. This value is equal to absolute zero, and there are no places on Earth with a similar temperature (possibly in the laboratory by laser cooling of molecules). This is rather an error in the reading by the sensor and we intend to replace for these observations the value of the average for the month in which this occurred,
* *rain\_mm* variable contains an outlier observation equal to 9831.3, which, compared to the mean of 0.51, can strongly influence the estimation of the models. To do this, we replace this observation with the second maximum value for this set.

traffic.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 29701 entries, 0 to 29700  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 date\_time 29701 non-null object   
 1 weather\_general 29701 non-null object   
 2 weather\_detailed 29701 non-null object   
 3 clouds\_coverage\_pct 29701 non-null int64   
 4 temperature 29701 non-null float64  
 5 rain\_mm 29701 non-null float64  
 6 snow\_mm 29701 non-null float64  
 7 traffic 29701 non-null int64   
dtypes: float64(3), int64(2), object(3)  
memory usage: 1.8+ MB

char\_cols = traffic.select\_dtypes(include=['object'])  
char\_cols = char\_cols.drop('date\_time', axis = 1)  
for col in char\_cols:  
 print(col, ':', char\_cols[col].value\_counts())

weather\_general : Clouds 10164  
Clear 8030  
Rain 3560  
Mist 3491  
Snow 1794  
Drizzle 920  
Haze 743  
Thunderstorm 497  
Fog 481  
Smoke 17  
Squall 4  
Name: weather\_general, dtype: int64  
weather\_detailed : sky is clear 8030  
mist 3491  
overcast clouds 3491  
broken clouds 3130  
light rain 2248  
scattered clouds 2044  
few clouds 1499  
light snow 1046  
moderate rain 966  
haze 743  
heavy snow 581  
light intensity drizzle 551  
fog 481  
drizzle 337  
proximity thunderstorm 311  
heavy intensity rain 287  
snow 157  
thunderstorm 58  
thunderstorm with heavy rain 42  
thunderstorm with light rain 32  
heavy intensity drizzle 31  
proximity shower rain 31  
thunderstorm with rain 23  
proximity thunderstorm with rain 19  
very heavy rain 18  
smoke 17  
light intensity shower rain 8  
proximity thunderstorm with drizzle 8  
light rain and snow 4  
squalls 4  
light shower snow 4  
freezing rain 2  
thunderstorm with light drizzle 2  
thunderstorm with drizzle 2  
shower snow 1  
shower drizzle 1  
sleet 1  
Name: weather\_detailed, dtype: int64

In our dataset, we do not notice any missing data in all columns, so we do not involve any algorithm to impute missing data

## Data Transformations

In our analysis of traffic jams in the city, we wanted to extract as much information as possible from the proposed variables. We decided to extract information from the primary variables as follows:

* *date\_time* -> From this variable we extracted information about the *year*, *month*, *day* of the week and *hour*. We decided not to decode each hour into a zero-one variable, but to make 6 hour intervals dividing our day into 4 equal parts. This is justified because for these 4 parts the traffic variable differs significantly due to sleeping hours and working and commuting hours. Additionally, we decided to add a variable informing whether it is currently the weekend.
* *weather\_detailed* -> For this variable, we considered to create an ordinal variable from the presented descriptions of the atmospheric phenomena observed on the road. We established 4 levels (from the worst conditions to sufficiently favourable conditions). Level 0 means fatal conditions, 1 poor, 2 moderate and 3 good.
* *weather\_general* -> For this nominal variable, we decoded all levels into zero-one variables and checked which weather types have a variance close to zero. For such variables we attached them to other equally thematically similar categories (Smoke -> Fog and Squall -> Rain)

def dayFromDate(traffic):  
 traffic['date\_time'] = pd.to\_datetime(traffic['date\_time'])  
 traffic['hour'] = traffic.date\_time.dt.hour  
 traffic['hour\_interval'] = pd.cut(traffic.hour, bins = [-1,6,12,18,24], labels = ['0-6','6-12', '12-18', '18-24'])  
   
   
   
 traffic['day'] = traffic.date\_time.dt.day\_of\_week   
   
 traffic['is\_weekend'] = np.where(traffic.day.isin([5, 6]), 1, 0)  
 traffic['month'] = traffic.date\_time.dt.month  
  
 traffic['tmp'] = range(0,len(traffic.hour))  
 traffic['year'] = traffic.date\_time.dt.year  
 t1 = pd.get\_dummies(traffic['month'])  
 t1['tmp'] = range(0,len(traffic.month))  
 traffic = pd.merge(traffic, t1, on = ['tmp'])  
  
 t1 = pd.get\_dummies(traffic['hour\_interval'])  
 t1['tmp'] = range(0,len(traffic.hour))  
 traffic = pd.merge(traffic, t1, on = ['tmp'])  
  
 t1 = pd.get\_dummies(traffic['hour'])  
 t1['tmp'] = range(0,len(traffic.hour))  
 traffic = pd.merge(traffic, t1, on = ['tmp'])  
  
 t1 = pd.get\_dummies(traffic['day'])  
 t1['tmp'] = range(0,len(traffic.hour))  
 traffic = pd.merge(traffic, t1, on = ['tmp']).drop(columns = ['tmp'])  
 return(traffic)  
  
def ordinal\_whether(traffic):  
 traffic['ordinal\_weather'] = np.where(traffic.weather\_detailed.isin(['fog', 'freezing rain', 'heavy intensity rain','heavy snow','proximity thunderstorm',  
 'proximity thunderstorm with drizzle', 'proximity thunderstorm with rain', 'sleet', 'squalls',  
 'thunderstorm', 'thunderstorm with drizzle', 'thunderstorm with heavy rain', 'thunderstorm with light drizzle',  
 'thunderstorm with light rain', 'thunderstorm with rain', 'very heavy rain']), 0, #fatal  
 np.where(traffic.weather\_detailed.isin(['drizzle','haze', 'heavy intensity drizzle','light intensity drizzle','light intensity shower rain',  
 'light rain', 'light rain and snow', 'light shower snow', 'light snow','mist', 'moderate rain', 'shower drizzle',  
 'shower snow', 'smoke', 'snow', ]), 1, #poor  
 np.where(traffic.weather\_detailed.isin(['broken clouds','few clouds', 'overcast clouds', 'proximity shower rain',  
 ]), 2, #moderate  
 np.where(traffic.weather\_detailed.isin(['scattered clouds', 'sky is clear', ]), 3, 99 )))) #good  
 return (traffic)  
   
def weather\_dummies(traffic):  
 t1 = pd.get\_dummies(traffic['weather\_general'])  
 t1['tmp'] = range(0,len(traffic.weather\_detailed))  
 traffic['tmp'] = range(0,len(traffic.weather\_detailed))  
 traffic = pd.merge(traffic, t1, on = ['tmp'])  
 traffic['Fog'] = traffic['Fog'] + traffic['Smoke']  
 traffic['Rain'] = traffic['Rain'] + traffic['Squall']  
 traffic = traffic.drop(columns = ['Smoke', 'Squall', 'tmp'])   
 return(traffic)  
  
def weather\_dummies\_test(traffic):  
 t1 = pd.get\_dummies(traffic['weather\_general'])  
 t1['tmp'] = range(0,len(traffic.weather\_detailed))  
 traffic['tmp'] = range(0,len(traffic.weather\_detailed))  
 traffic = pd.merge(traffic, t1, on = ['tmp'])  
 traffic['Fog'] = traffic['Fog'] + traffic['Smoke']  
 traffic['Rain'] = traffic['Rain'] #+ traffic['Squall']  
 traffic = traffic.drop(columns = ['Smoke', 'tmp']) # 'Squall',  
 return(traffic)  
  
def outliers\_correction(traffic):  
  
 traffic.temperature.loc[(traffic.temperature < -30)] = traffic.groupby(by = ['month']).mean().temperature[1]  
 traffic.rain\_mm.loc[traffic.rain\_mm > 1000] = traffic.rain\_mm.loc[traffic.rain\_mm < 1000].max()  
 return (traffic)

## Outliers detection

As we mentioned earlier for the variables temperature and rain\_mm we probably observed errors in the readings. As far as temperature is concerned, for these outlier observations we replaced their mean values for the month in which these readings occurred. For rain\_mm we replaced the outlier with the second largest observation in the set.

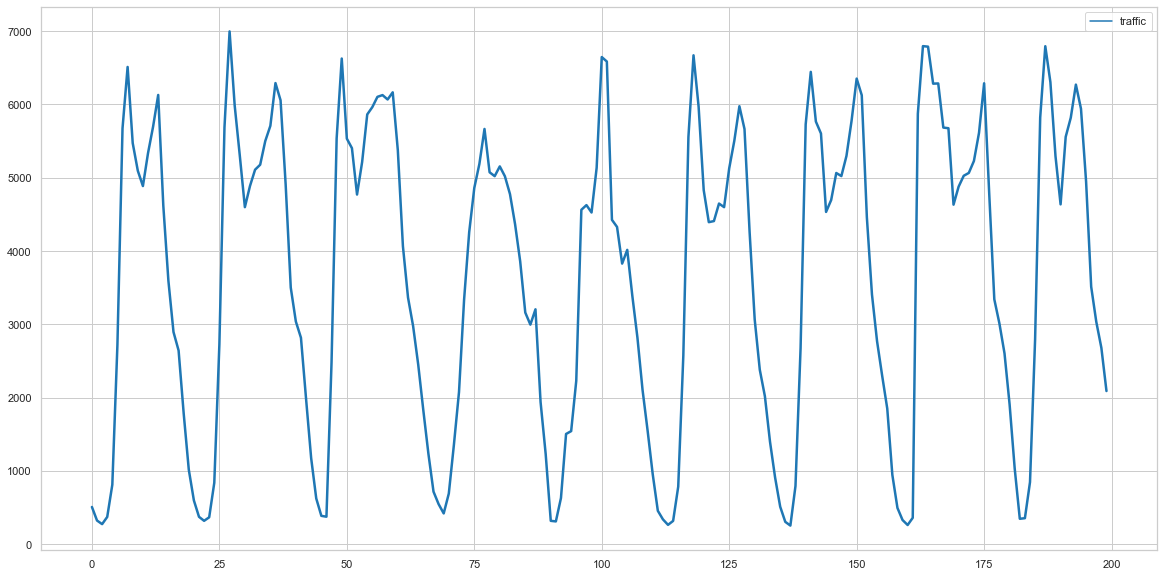
# 3. Exploanatory Data Analysis

Let's create a time series for the variable traffic for the first 200 observations.

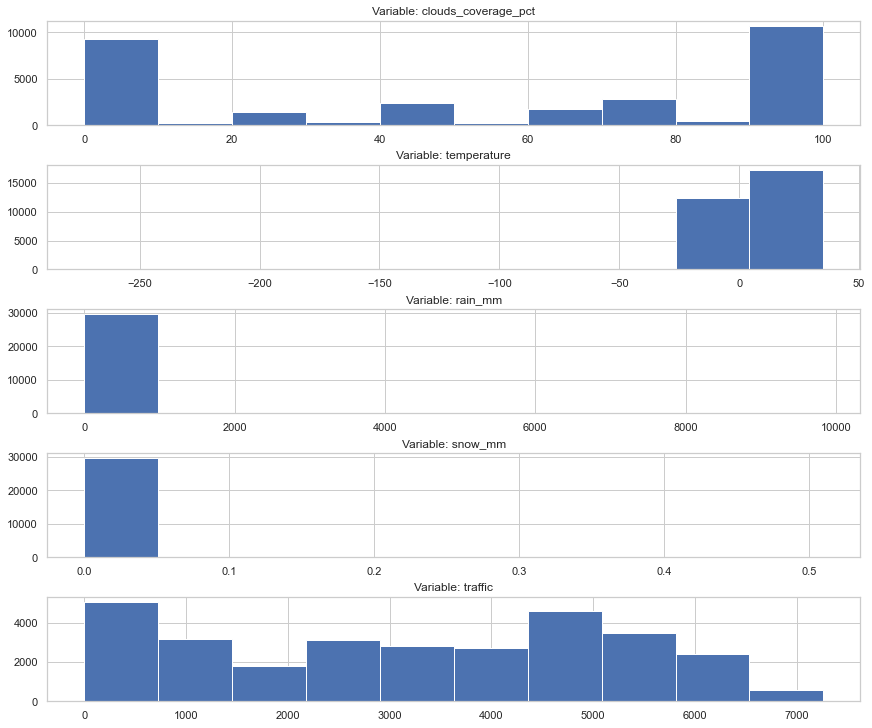
We can see that for this variable we have a very visible seasonality, which to a large extent certainly depends on the hour in which the traffic was recorded.

sns.set(rc = {'figure.figsize':(20,10)})  
sns.set\_theme(style="whitegrid")  
temp\_traffic = traffic[['date\_time', 'traffic']]  
temp\_traffic['date\_time'] = pd.to\_datetime(temp\_traffic['date\_time'])  
sns.lineplot(data=temp\_traffic[:200], palette="tab10", linewidth=2.5)

<AxesSubplot:>



numericalVar = ['clouds\_coverage\_pct', 'temperature', 'rain\_mm', 'snow\_mm', 'traffic']  
fig, axs = plt.subplots(ncols=1, nrows=5, figsize=(12, 10),  
 constrained\_layout=True)  
for idx, col in enumerate(numericalVar):  
 axs[idx].hist(traffic[f'{col}'])  
 axs[idx].set\_title(f'Variable: {col}')



Above you can see the histograms for the numerical variables before correction for outlier observations. Unfortunately, normal distributions are not visible at first glance. In addition, the variables rain\_mm and snow\_mm are highly concentrated around the value 0.

Let us now perform all the transformations of the variables and the correction of outlier observations.

# functions = [dayFromDate(traffic), ordinal\_whether(traffic), weather\_dummies(traffic), outliers\_correction(traffic)]  
  
traffic = dayFromDate(traffic)  
traffic = ordinal\_whether(traffic)  
traffic = weather\_dummies(traffic)  
traffic = outliers\_correction(traffic)

Let us now perform all the transformations of the variables and the correction of outlier observations. In addition, encoding variables spoils the naming of variables, so it is worth manually changing them to more understandable ones.

traffic.rename(columns={'1\_x':'January', '2\_x':'February', '3\_x':'March', '4\_x':'April',  
 '5\_x':'May','6\_x':'June','7\_x':'July', '8\_x':'August','9\_x':'September',  
 '10\_x':'November', '11\_x':'October', '12\_x':'December', '0\_x':'Hour\_0',  
 '1\_y':'Hour\_1','2\_y':'Hour\_2','3\_y':'Hour\_3','4\_y':'Hour\_4',  
 '5\_y':'Hour\_5','6\_y':'Hour\_6','7\_y':'Hour\_7','8\_y':'Hour\_8',  
 '9\_y':'Hour\_9','10\_y':'Hour\_10','11\_y':'Hour\_11','12\_y':'Hour\_12',  
 13:'Hour\_13',14:'Hour\_14',15:'Hour\_15',16:'Hour\_16',  
 17:'Hour\_17',18:'Hour\_18',19:'Hour\_19',20:'Hour\_20',  
 21:'Hour\_21',22:'Hour\_22',23:'Hour\_23','0\_y':'Monday',  
 1:'Tuesday', 2:'Wednesday', 3:'Thursday', 4:'Friday', 5:'Saturday', 6:'Sunday'}, inplace=True)

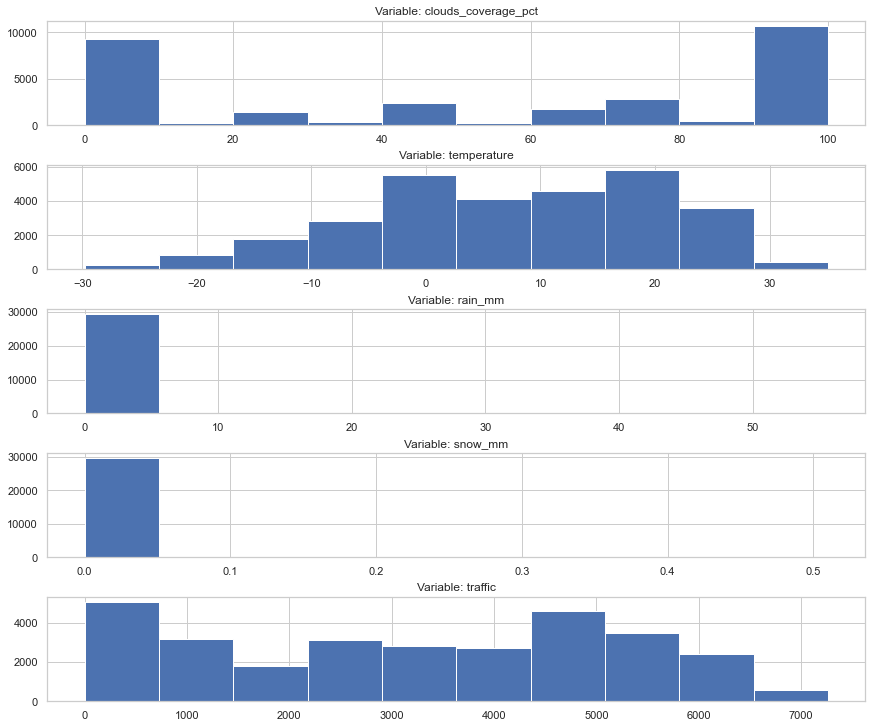
## Corrplot

Let's start by creating a correlation matrix.  
Only few of the variables are highly correlated with each other (assumed threshold >|0.7|).  
In particular, it is good information that the predicted variable has only very weak correlation coefficients.

corr = traffic.corr()  
mask = np.zeros\_like(corr, dtype=bool)  
mask[np.triu\_indices\_from(mask)] = True  
corr[mask] = np.nan  
(corr  
 .style  
 .background\_gradient(cmap='coolwarm', axis=None, vmin=-1, vmax=1)  
 .highlight\_null(null\_color='#f1f1f1')  
 .set\_precision(2))

<pandas.io.formats.style.Styler at 0x1c5e8a6a550>

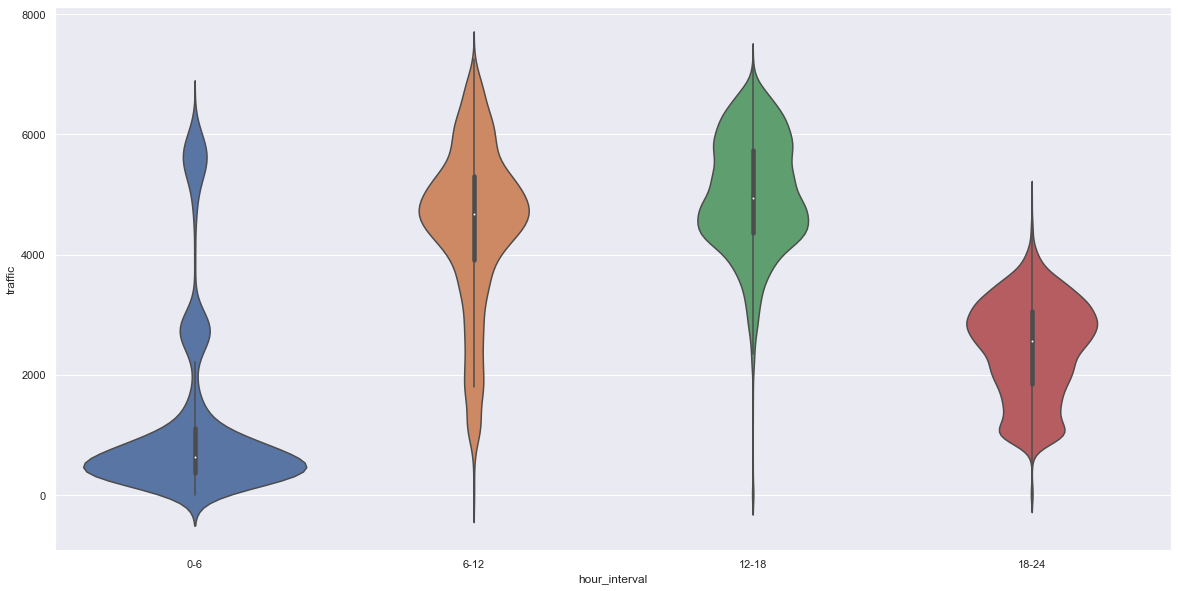
fig, axs = plt.subplots(ncols=1, nrows=5, figsize=(12, 10),  
 constrained\_layout=True)  
for idx, col in enumerate(numericalVar):  
 axs[idx].hist(traffic[f'{col}'])  
 axs[idx].set\_title(f'Variable: {col}')



After the applied changes on the data and the removal of outlier observations, the distributions of the analysed variables can be seen much better. However, still the variables rain\_mm and snow\_mm oscillate around zero, which means that they may not be significant variables.

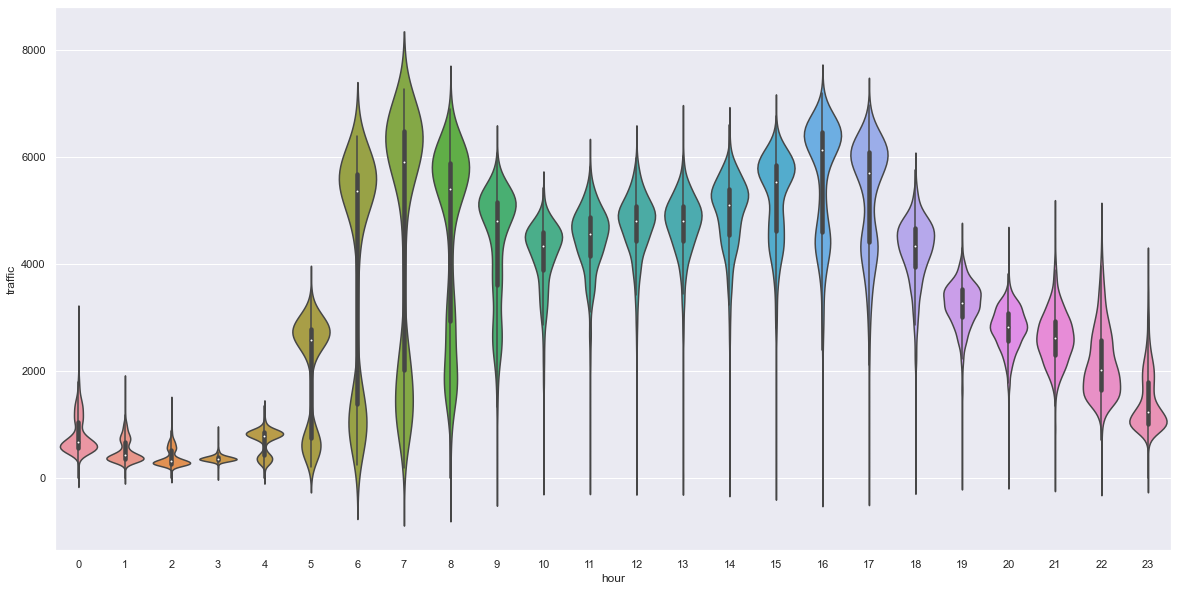
## Violin plots

sns.set\_theme(style="whitegrid")  
sns.set(rc = {'figure.figsize':(20,10)})  
ax = sns.violinplot(x = traffic.hour\_interval, y = traffic.traffic)



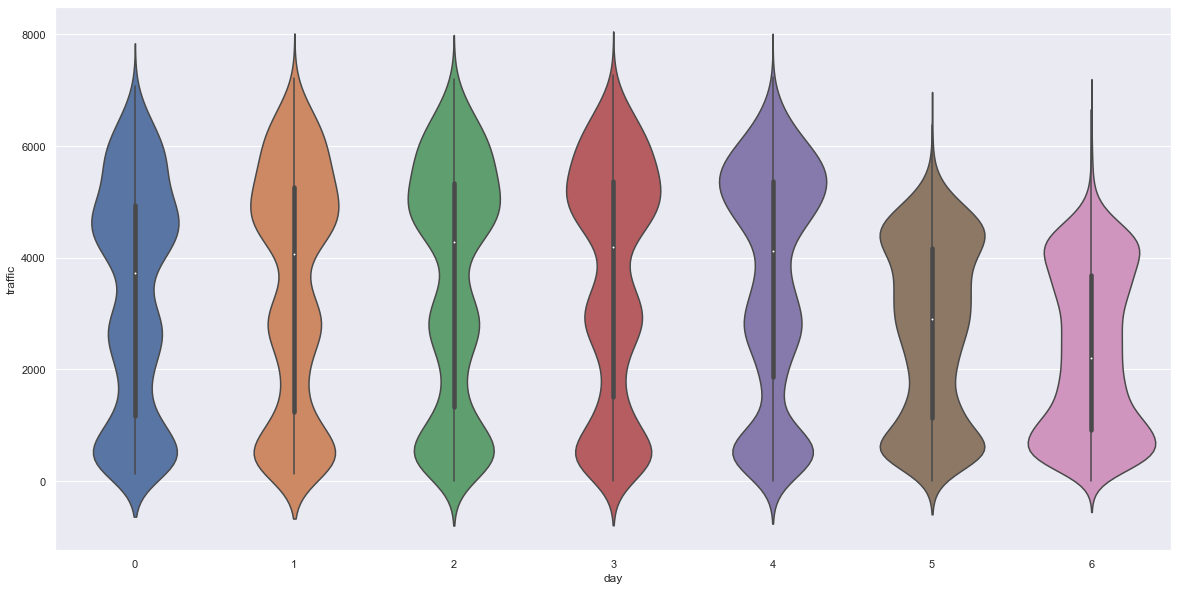
Let's look at the distribution of the traffic variable according to the hourly interval used. It is clear that there is much less traffic on the road at night and in the evening than during working hours. This variable can have a significant impact on the modelling.

ax = sns.violinplot(x = traffic.hour, y = traffic.traffic, scale = "width")



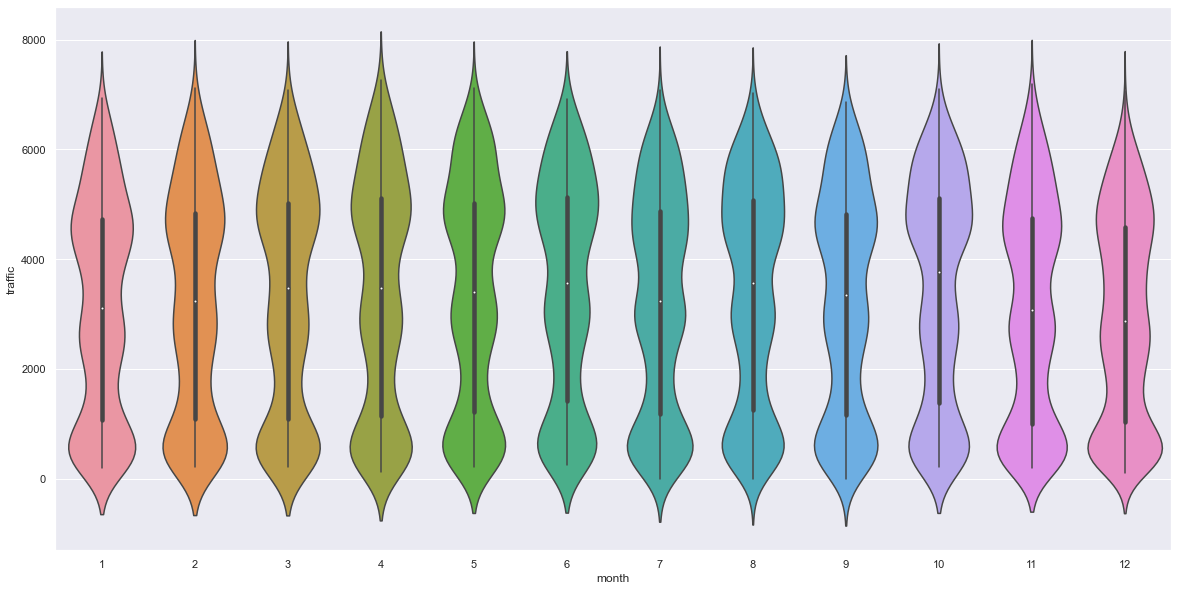
A more accurate view of the traffic variable can be seen for exact times. We can see a huge upward trend in the hours when work starts and when work ends. However, during the night hours traffic decreases accordingly.

ax = sns.violinplot(x = traffic.day, y = traffic.traffic)



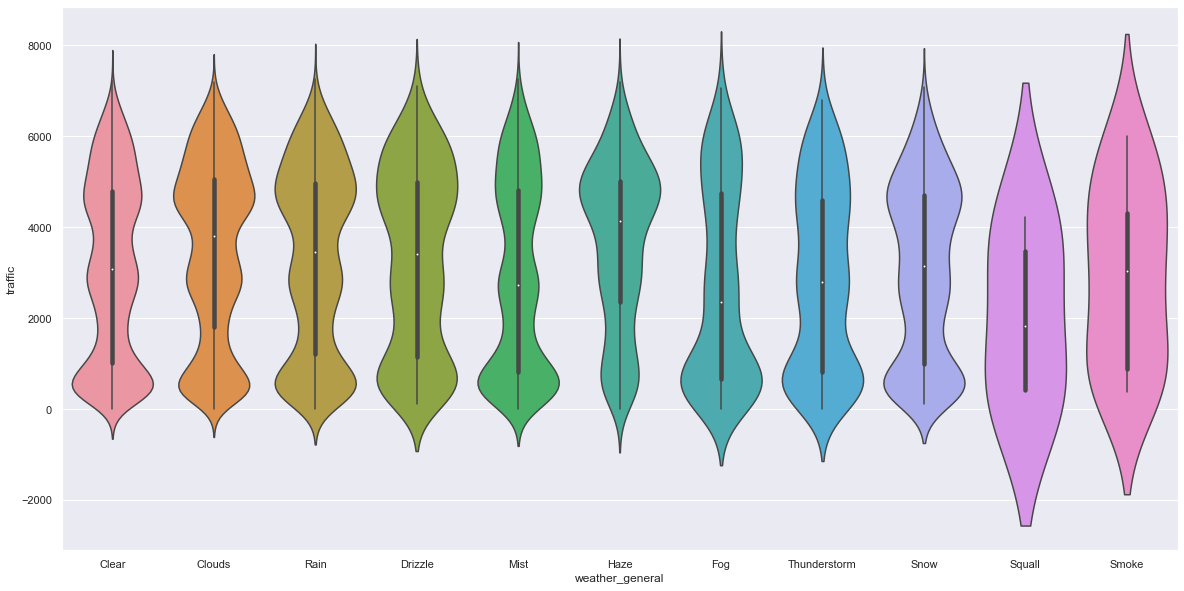
When it comes to comparing the distribution of traffic by day of the week, there is no noticeable difference between working days (Monday to Friday). However, a decrease in traffic is clearly visible on Saturdays and Sundays. This justifies the use of the variable informing whether it is the weekend.

ax = sns.violinplot(x = traffic.month, y = traffic.traffic)



Comparing the distribution of the traffic variable by each month of the year, we do not notice a significant change in traffic volume during the winter or summer months. This may be confirmed by the fact that people work all year round, and single anomalies related to the increase of snowfall, slippery roads, or the occurrence of storms in summer are impossible to detect.

ax = sns.violinplot(x = traffic.weather\_general, y = traffic.traffic, scale = "width")

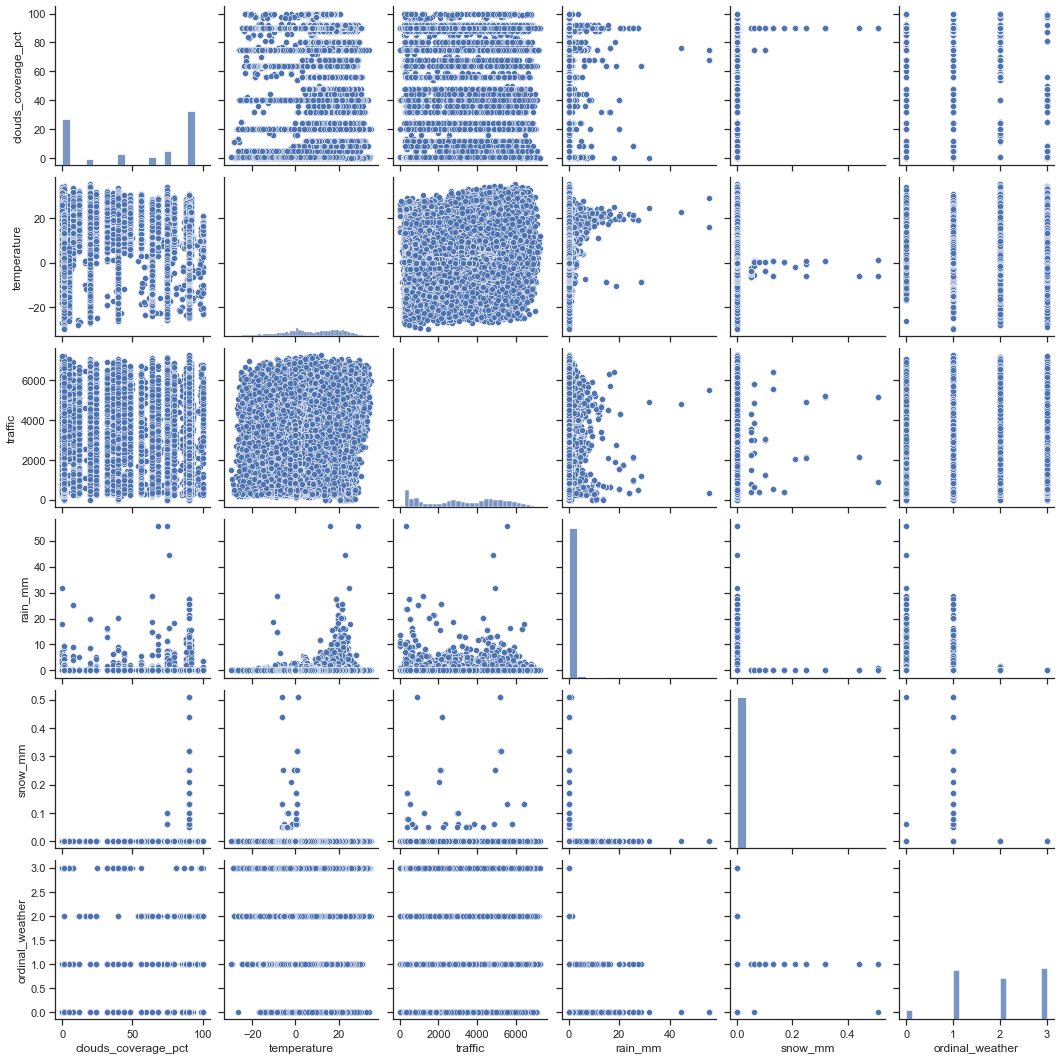


The last statement is the relationship of the variable traffic depending on weather conditions (weather\_general). Only Squall and Smoke differ from the other levels. However, as these two levels are the least numerous, they are included in Rain and Fog.

## Pairplot between all numerical variables

sns.set\_theme(style="ticks")  
sns.pairplot(traffic[['clouds\_coverage\_pct', 'temperature', 'traffic','rain\_mm','snow\_mm', 'ordinal\_weather']])

<seaborn.axisgrid.PairGrid at 0x1c5e8a645b0>



The combination of all numeric variables with continuous variables can be seen in the pairplot diagram.

## Deletion of observations equal to 0 for traffic and deletion of unnecessary variables

Because for regression our task is to estimate machine learning models with the smallest possible value of the cost function MAPE (Mean Absolute Percentage Error), unfortunately we have to remove observations that take the value 0 for the traffic variable. This is because the formula for MAPE has the value of the actual observation in the denominator, which leads to the situation that when we have a reading of 0 the MAPE cost function explodes to infinity.

traffic = traffic[traffic['traffic'] != 0]  
traffic = traffic[traffic['snow\_mm'] == 0]  
traffic = traffic[traffic['rain\_mm'] == 0]  
traffic = traffic.drop(columns = ['date\_time', 'weather\_general', 'weather\_detailed',   
 'month', 'year', 'day', 'hour', 'rain\_mm','snow\_mm','hour\_interval'])

## Division of the training set into training and test subsets

The breakdown of the training set is shown below. We divide our original training set into **85% training** observations and **15% test** observations to check the performance and effectiveness of the proposed models.

train\_x, test\_x, train\_y, test\_y = train\_test\_split(traffic.loc[:,traffic.columns!='traffic'], traffic.traffic, test\_size=0.15, random\_state=50)

After division, the training and test sets have the following sizes.

print('Shape of train\_x:', train\_x.shape, end = '\n')  
print('Shape of train\_y:', train\_y.shape, end = '\n')  
print('Shape of test\_x:', test\_x.shape, end = '\n')  
print('Shape of test\_y:', test\_y.shape)

Shape of train\_x: (22845, 60)  
Shape of train\_y: (22845,)  
Shape of test\_x: (4032, 60)  
Shape of test\_y: (4032,)

## Mutual info

Before we move on to the modelling section let's have a look at the relationship of a single variable to the traffic variable under study. For this purpose you can use a statistic called Mutual Info. This statistic measures the dependence of one variable on another. The value of this statistic is close to zero, then we can see no dependency (variables are independent), while the observed values are much higher than zero, then we can conclude that there is a dependency (dependent variables).

* High dependence variables: *clouds\_coverage\_pct*, *is\_weekend*, *0-6*, *6-12*, *12-18*, *18-24*, *ordinal\_weather*, *Clear*, *Clouds*,
* Low or none dependence variables: *January*, *February*, *March*, *April*, *May*, *June*, ..., *December*, *Hour\_0*, ..., *Hour\_23*.

from scipy import stats  
from sklearn import feature\_selection  
  
import warnings  
warnings.simplefilter(action='ignore', category=FutureWarning)  
  
minfos=[]  
  
for var in traffic.columns:  
 if var in ['temperature', 'traffic']:  
 continue  
 print("\n", var)  
 print("Pearson", np.round(stats.pearsonr(traffic.traffic, traffic[var]), 4))  
 print("Mutual info", np.round(feature\_selection.mutual\_info\_classif(traffic[var].values.reshape(-1,1),  
 traffic.traffic.values), 4))  
 minfos.append(feature\_selection.mutual\_info\_classif(traffic[var].values.reshape(-1,1),  
 traffic.traffic.values))  
 print("Chi2", np.round(feature\_selection.chi2(traffic[var].values.reshape(-1,1),  
 traffic.traffic.values), 4))  
 print("Anova", np.round(feature\_selection.f\_classif(traffic[var].values.reshape(-1,1),  
 traffic.traffic.values), 4))

clouds\_coverage\_pct  
Pearson [0.0357 0. ]  
Mutual info [0.285]  
Chi2 [[208532.4742]  
 [ 0. ]]  
Anova [[1.0486]  
 [0.0092]]  
  
 is\_weekend  
Pearson [-0.2222 0. ]  
Mutual info [0.4993]  
Chi2 [[6895.1787]  
 [ 0. ]]  
Anova [[1.8066]  
 [0. ]]  
  
 January  
Pearson [-0.018 0.0032]  
Mutual info [0.0084]  
Chi2 [[6.1088153e+03]  
 [9.9580000e-01]]  
Anova [[1.0415]  
 [0.0218]]  
  
 February  
Pearson [-0.0082 0.1764]  
Mutual info [0.0083]  
Chi2 [[6.2390547e+03]  
 [9.2720000e-01]]  
Anova [[1.0683e+00]  
 [5.0000e-04]]  
  
 March  
Pearson [-3.000e-04 9.565e-01]  
Mutual info [0.0138]  
Chi2 [[6.1971531e+03]  
 [9.6660000e-01]]  
Anova [[1.0576]  
 [0.0027]]  
  
 April  
Pearson [0.0209 0.0006]  
Mutual info [0.0023]  
Chi2 [[6.3758693e+03]  
 [5.9260000e-01]]  
Anova [[1.1046]  
 [0. ]]  
  
 May  
Pearson [0.0184 0.0025]  
Mutual info [0.0137]  
Chi2 [[6.1955385e+03]  
 [9.6770000e-01]]  
Anova [[1.0744e+00]  
 [2.0000e-04]]  
  
 June  
Pearson [0.027 0. ]  
Mutual info [0]  
Chi2 [[6.1635515e+03]  
 [9.8370000e-01]]  
Anova [[1.0342]  
 [0.0477]]  
  
 July  
Pearson [-0.0038 0.534 ]  
Mutual info [0.025]  
Chi2 [[6.0144626e+03]  
 [9.9980000e-01]]  
Anova [[1.0608]  
 [0.0017]]  
  
 August  
Pearson [0.0341 0. ]  
Mutual info [0.0077]  
Chi2 [[5.9929599e+03]  
 [9.9990000e-01]]  
Anova [[1.0176]  
 [0.1928]]  
  
 September  
Pearson [2.00e-04 9.69e-01]  
Mutual info [0.0048]  
Chi2 [[5.9245502e+03]  
 [1.0000000e+00]]  
Anova [[0.9858]  
 [0.7581]]  
  
 November  
Pearson [0.029 0. ]  
Mutual info [0.008]  
Chi2 [[6.0463464e+03]  
 [9.9930000e-01]]  
Anova [[1.05 ]  
 [0.0077]]  
  
 October  
Pearson [-0.0232 0.0001]  
Mutual info [0.0167]  
Chi2 [[5.6886926e+03]  
 [1.0000000e+00]]  
Anova [[0.995 ]  
 [0.5971]]  
  
 December  
Pearson [-0.0598 0. ]  
Mutual info [0.0266]  
Chi2 [[5.9912614e+03]  
 [9.9990000e-01]]  
Anova [[1.0805e+00]  
 [1.0000e-04]]  
  
 0-6  
Pearson [-0.6396 0. ]  
Mutual info [0.973]  
Chi2 [[14301.2673]  
 [ 0. ]]  
Anova [[9.9915]  
 [0. ]]  
  
 6-12  
Pearson [0.3623 0. ]  
Mutual info [0.4235]  
Chi2 [[8026.4728]  
 [ 0. ]]  
Anova [[2.1169]  
 [0. ]]  
  
 12-18  
Pearson [0.5063 0. ]  
Mutual info [0.6581]  
Chi2 [[9768.9878]  
 [ 0. ]]  
Anova [[2.9526]  
 [0. ]]  
  
 18-24  
Pearson [-0.2008 0. ]  
Mutual info [0.8123]  
Chi2 [[13426.6846]  
 [ 0. ]]  
Anova [[5.484]  
 [0. ]]  
  
 Hour\_0  
Pearson [-0.2573 0. ]  
Mutual info [0.0584]  
Chi2 [[11104.6449]  
 [ 0. ]]  
Anova [[2.4299]  
 [0. ]]  
  
 Hour\_1  
Pearson [-0.2876 0. ]  
Mutual info [0.0536]  
Chi2 [[7088.1408]  
 [ 0. ]]  
Anova [[1.2155]  
 [0. ]]  
  
 Hour\_2  
Pearson [-0.302 0. ]  
Mutual info [0.1058]  
Chi2 [[11715.8959]  
 [ 0. ]]  
Anova [[2.674]  
 [0. ]]  
  
 Hour\_3  
Pearson [-0.2961 0. ]  
Mutual info [0.0987]  
Chi2 [[10934.7577]  
 [ 0. ]]  
Anova [[2.3544]  
 [0. ]]  
  
 Hour\_4  
Pearson [-0.2732 0. ]  
Mutual info [0.0974]  
Chi2 [[10564.1039]  
 [ 0. ]]  
Anova [[2.2303]  
 [0. ]]  
  
 Hour\_5  
Pearson [-0.1276 0. ]  
Mutual info [0.039]  
Chi2 [[8613.1089]  
 [ 0. ]]  
Anova [[1.6087]  
 [0. ]]  
  
 Hour\_6  
Pearson [0.0943 0. ]  
Mutual info [0.0217]  
Chi2 [[8371.6873]  
 [ 0. ]]  
Anova [[1.5414]  
 [0. ]]  
  
 Hour\_7  
Pearson [0.153 0. ]  
Mutual info [0.0282]  
Chi2 [[11904.0111]  
 [ 0. ]]  
Anova [[2.7503]  
 [0. ]]  
  
 Hour\_8  
Pearson [0.1393 0. ]  
Mutual info [0.0147]  
Chi2 [[8586.6354]  
 [ 0. ]]  
Anova [[1.6026]  
 [0. ]]  
  
 Hour\_9  
Pearson [0.1136 0. ]  
Mutual info [0.0034]  
Chi2 [[7265.2295]  
 [ 0. ]]  
Anova [[1.2545]  
 [0. ]]  
  
 Hour\_10  
Pearson [0.0993 0. ]  
Mutual info [0.0201]  
Chi2 [[7900.3716]  
 [ 0. ]]  
Anova [[1.4177]  
 [0. ]]  
  
 Hour\_11  
Pearson [0.1266 0. ]  
Mutual info [0.0068]  
Chi2 [[7156.6174]  
 [ 0. ]]  
Anova [[1.2276]  
 [0. ]]  
  
 Hour\_12  
Pearson [0.1535 0. ]  
Mutual info [0.0136]  
Chi2 [[6876.0589]  
 [ 0. ]]  
Anova [[1.1633]  
 [0. ]]  
  
 Hour\_13  
Pearson [0.151 0. ]  
Mutual info [0.0148]  
Chi2 [[6995.4349]  
 [ 0. ]]  
Anova [[1.1879]  
 [0. ]]  
  
 Hour\_14  
Pearson [0.1796 0. ]  
Mutual info [0.0083]  
Chi2 [[7248.5936]  
 [ 0. ]]  
Anova [[1.2516]  
 [0. ]]  
  
 Hour\_15  
Pearson [0.2037 0. ]  
Mutual info [0.0122]  
Chi2 [[7856.11]  
 [ 0. ]]  
Anova [[1.3993]  
 [0. ]]  
  
 Hour\_16  
Pearson [0.2508 0. ]  
Mutual info [0.0444]  
Chi2 [[11655.4827]  
 [ 0. ]]  
Anova [[2.6424]  
 [0. ]]  
  
 Hour\_17  
Pearson [0.2104 0. ]  
Mutual info [0.0232]  
Chi2 [[8900.9123]  
 [ 0. ]]  
Anova [[1.6832]  
 [0. ]]  
  
 Hour\_18  
Pearson [0.106 0. ]  
Mutual info [0.0119]  
Chi2 [[7648.1157]  
 [ 0. ]]  
Anova [[1.3497]  
 [0. ]]  
  
 Hour\_19  
Pearson [0.0015 0.8099]  
Mutual info [0.0344]  
Chi2 [[10823.4319]  
 [ 0. ]]  
Anova [[2.3149]  
 [0. ]]  
  
 Hour\_20  
Pearson [-0.0443 0. ]  
Mutual info [0.0348]  
Chi2 [[9884.0882]  
 [ 0. ]]  
Anova [[1.9898]  
 [0. ]]  
  
 Hour\_21  
Pearson [-0.0599 0. ]  
Mutual info [0.0301]  
Chi2 [[9837.0011]  
 [ 0. ]]  
Anova [[1.9733]  
 [0. ]]  
  
 Hour\_22  
Pearson [-0.1104 0. ]  
Mutual info [0.0368]  
Chi2 [[11389.3616]  
 [ 0. ]]  
Anova [[2.5375]  
 [0. ]]  
  
 Hour\_23  
Pearson [-0.1914 0. ]  
Mutual info [0.0759]  
Chi2 [[12837.0276]  
 [ 0. ]]  
Anova [[3.1895]  
 [0. ]]  
  
 Monday  
Pearson [0.0145 0.0171]  
Mutual info [0.034]  
Chi2 [[5.9346785e+03]  
 [1.0000000e+00]]  
Anova [[1.1102]  
 [0. ]]  
  
 Tuesday  
Pearson [0.0547 0. ]  
Mutual info [0.0283]  
Chi2 [[5.7955672e+03]  
 [1.0000000e+00]]  
Anova [[1.0663e+00]  
 [7.0000e-04]]  
  
 Wednesday  
Pearson [0.0721 0. ]  
Mutual info [0.0512]  
Chi2 [[5.9077869e+03]  
 [1.0000000e+00]]  
Anova [[1.1102]  
 [0. ]]  
  
 Thursday  
Pearson [0.0722 0. ]  
Mutual info [0.0369]  
Chi2 [[5.9213331e+03]  
 [1.0000000e+00]]  
Anova [[1.0967]  
 [0. ]]  
  
 Friday  
Pearson [0.0751 0. ]  
Mutual info [0.0487]  
Chi2 [[5.8564969e+03]  
 [1.0000000e+00]]  
Anova [[1.0937]  
 [0. ]]  
  
 Saturday  
Pearson [-0.1041 0. ]  
Mutual info [0.0735]  
Chi2 [[6.6500634e+03]  
 [1.5300000e-02]]  
Anova [[1.2991]  
 [0. ]]  
  
 Sunday  
Pearson [-0.1822 0. ]  
Mutual info [0.1068]  
Chi2 [[7366.0102]  
 [ 0. ]]  
Anova [[1.5104]  
 [0. ]]  
  
 ordinal\_weather  
Pearson [0.0413 0. ]  
Mutual info [0.5483]  
Chi2 [[2.780652e+03]  
 [1.000000e+00]]  
Anova [[1.036 ]  
 [0.0396]]  
  
 Clear  
Pearson [-0.0445 0. ]  
Mutual info [0.2098]  
Chi2 [[4.6041426e+03]  
 [1.0000000e+00]]  
Anova [[1.0332]  
 [0.0524]]  
  
 Clouds  
Pearson [0.0991 0. ]  
Mutual info [0.5038]  
Chi2 [[4.2390055e+03]  
 [1.0000000e+00]]  
Anova [[1.0861]  
 [0. ]]  
  
 Drizzle  
Pearson [-0.0033 0.5839]  
Mutual info [0]  
Chi2 [[6.3804785e+03]  
 [5.7670000e-01]]  
Anova [[1.0311]  
 [0.0642]]  
  
 Fog  
Pearson [-0.0396 0. ]  
Mutual info [0.0029]  
Chi2 [[6.1009821e+03]  
 [9.9660000e-01]]  
Anova [[0.9601]  
 [0.9774]]  
  
 Haze  
Pearson [0.0316 0. ]  
Mutual info [0.006]  
Chi2 [[6.3810014e+03]  
 [5.7480000e-01]]  
Anova [[1.0314]  
 [0.0623]]  
  
 Mist  
Pearson [-0.0655 0. ]  
Mutual info [0.0051]  
Chi2 [[5.6000307e+03]  
 [1.0000000e+00]]  
Anova [[0.9779]  
 [0.8642]]  
  
 Rain  
Pearson [0.003 0.625]  
Mutual info [0.0038]  
Chi2 [[5.821013e+03]  
 [1.000000e+00]]  
Anova [[0.9714]  
 [0.923 ]]  
  
 Snow  
Pearson [-0.0243 0.0001]  
Mutual info [0.0024]  
Chi2 [[5.8479521e+03]  
 [1.0000000e+00]]  
Anova [[0.968 ]  
 [0.9449]]  
  
 Thunderstorm  
Pearson [-0.015 0.0137]  
Mutual info [0.0024]  
Chi2 [[5.7410875e+03]  
 [1.0000000e+00]]  
Anova [[0.8783]  
 [1. ]]

# 4. Modelling

In our analysis, we would like to present the following models and compare their effectiveness in predicting values for the variable traffic:

* Linear Regression
* Random Forest Regressor
* Elastic Net
* Support Vector Regressor
* K Nearest Neighbours Regressor
* Lasso
* Tweedie Regressor
* Gradient Boosting Regressor
* LightGBM with Optuna Optimizer

Most of the models were run through the so-called **Pipeline**. Before running the model learning itself, we also used the **RobustScaler()** functions to scale the data using the median and the interquartile deviation (IQR). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html>

Additionally, for some of the available algorithms, we applied the **GridSearchCV()** and **RandomizedSearchCV()** algorithms to tune the hyperparameters of the machine learning models. This allows us to select from among classes of models the one that achieves the lowest MAPE cost function value.

mapeOfModels = [] # list to store mape results of the models

## Linear Regression

For the linear regression model with normalised variables we obtained a MAPE of 61.65%, which for such a basic model gives satisfactory results. Below we can take a look at how our estimated values of the traffic variable behave in comparison with the actual readings. An R-square of 82.5% also explains 82.5% of the variation in the traffic variable to a greater extent.

lin\_reg = LinearRegression(normalize=True)  
lin\_reg.fit(train\_x, train\_y)  
  
y\_pred = lin\_reg.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_pred) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_pred)}')  
print(r2\_score(test\_y, y\_pred))  
print(y\_pred.reshape(-1,1))  
print(test\_y)  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, y\_pred) \* 100, 4),'LinearRegression'))

MAPE: 61.6471%  
MAE: 606.1984126984127  
0.8257185633078699  
[[3776.]  
 [4792.]  
 [4674.]  
 ...  
 [4564.]  
 [4900.]  
 [5189.]]  
16864 3586  
1018 5299  
24021 4563  
9878 771  
7962 4418  
 ...   
23375 5855  
18348 3118  
27379 4609  
6229 6189  
18195 2640  
Name: traffic, Length: 4032, dtype: int64

## Random Forest Regressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Source: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

Our random forest model achieved a MAPE score of 27.78% which gave a really satisfactory result. Unfortunately, this is not a method learned in class and, as we will see later, did not get the lowest MAPE score, so we do not use these estimations. The best random forest model through RandomizedSearchCV obtained parameters:

* max\_features = 12
* n\_estimators = 3

pip\_forest = Pipeline([("scaler", RobustScaler()),  
 ("classifier",RandomForestRegressor(random\_state=997))])  
  
param\_grid = [  
 {'classifier\_\_n\_estimators': [3, 10, 15, 20, 30], 'classifier\_\_max\_features': [2, 4, 6, 8,12]},  
 {'classifier\_\_bootstrap': [False], 'classifier\_\_n\_estimators': [3, 10, 15,20,25, 30], 'classifier\_\_max\_features': [2, 4, 6, 8,12]}  
 ]  
  
  
grid\_search = RandomizedSearchCV(pip\_forest, param\_grid, cv=3, n\_iter=100,  
 scoring = make\_scorer(mean\_absolute\_percentage\_error, greater\_is\_better=False),  
 return\_train\_score=True)  
grid\_search.fit(train\_x, train\_y)  
  
best\_forest = grid\_search.best\_estimator\_ #RandomForestRegressor(bootstrap=False, max\_features=3, n\_estimators=3, random\_state=42)  
best\_forest.fit(train\_x, train\_y)  
y\_forest = best\_forest.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_forest) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_forest)}')

MAPE: 27.784%  
MAE: 344.0429480820106

best\_forest = grid\_search.best\_estimator\_ #RandomForestRegressor(bootstrap=False, max\_features=12, n\_estimators=3, random\_state=997))  
best\_forest.fit(train\_x, train\_y)  
y\_forest = best\_forest.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_forest) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_forest)}')  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, y\_forest) \* 100, 4),'RandomForestRegressor'))

MAPE: 27.784%  
MAE: 344.0429480820106

## ElasticNet

The ElasticNet model is a simple linear regression model with regularisation through the L1 and L2 parameters. As we have learned about Lasso and Ridge regressions in this class, ElasticNet may also be one of the models used in this class. Unfortunately, the value of the MAPE cost function for this presented model was not much better than the usual linear regression and was equal to 61.26%.

pip\_elastic = Pipeline([("scaler", RobustScaler()),  
 ("classifier",ElasticNet(random\_state=997))])  
  
grid = dict()  
grid['classifier\_\_alpha'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.0, 1.0, 10.0, 100.0]  
grid['classifier\_\_l1\_ratio'] = np.arange(0, 1, 0.2)  
  
grid\_search\_2 = RandomizedSearchCV(pip\_elastic, grid, cv=3, n\_iter = 100,  
 scoring=make\_scorer(mean\_absolute\_percentage\_error, greater\_is\_better=False),  
 return\_train\_score=True)  
  
grid\_search\_2.fit(train\_x, train\_y)  
  
  
best\_elastic = grid\_search\_2.best\_estimator\_ #RandomForestRegressor(bootstrap=False, max\_features=3, n\_estimators=3, random\_state=42)  
best\_elastic.fit(train\_x, train\_y)  
  
y\_elastic = best\_elastic.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_elastic) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_elastic)}')

MAPE: 61.2677%  
MAE: 603.9216728702429

x = grid\_search\_2.best\_estimator\_.fit(train\_x, train\_y)  
y\_elastic = x.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_elastic) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_elastic)}')  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, y\_elastic) \* 100, 4),'ElasticNet'))

MAPE: 61.2677%  
MAE: 603.9216728702429

## Support Vector Regressor

In class, we used Support Vector Machines in correctly separating observations about whether the person being examined is currently insured. For regression this also works well enough and our own grid search found that for kernel = 'linear' the smallest MAPE value was obtained for a parameter value of C = 100 equal to 50.77%. As we will see later this is also not a satisfactory result compared to other models.

for c in [0.1, 1, 10, 100]:  
   
 svr = SVR(C=c, kernel='linear')  
 svr.fit(train\_x, train\_y)  
  
 y\_SVR = svr.predict(test\_x)  
 print('Value for C:',c)  
 print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_SVR) \* 100, 4)}%')  
 print(f'MAE: {mean\_absolute\_error(test\_y, y\_SVR)}')

Value for C: 0.1  
MAPE: 203.1887%  
MAE: 1333.982981712389  
Value for C: 1  
MAPE: 73.7624%  
MAE: 694.4092001059929  
Value for C: 10  
MAPE: 51.8253%  
MAE: 568.1317720119458  
Value for C: 100  
MAPE: 50.7774%  
MAE: 561.7810861649792

svr = SVR(C=100, kernel='linear')  
svr.fit(train\_x, train\_y)  
  
y\_SVR = svr.predict(test\_x)  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, y\_SVR) \* 100, 4),'SVR'))

## K Nearest Neighbours Regressor

Regression based on k-nearest neighbors.

The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set. Source: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>

As the name suggests, the algorithm decides into which group to classify the studied observation by giving it an appropriate traffic value based on similar characteristics. KNN was a very promising algorithm, obtaining a MAPE cost function value of 32.8%. The GridSearchCV used found that the smallest MAPE value this algorithm took for:

* n\_neighbors = 3

pip\_neigh = Pipeline([("scaler", RobustScaler()),  
 ("classifier", KNeighborsRegressor())])  
  
parameters = {"classifier\_\_n\_neighbors": range(2, 20)}  
  
gridsearch = GridSearchCV(pip\_neigh, parameters, cv=3,  
 scoring=make\_scorer(mean\_absolute\_percentage\_error, greater\_is\_better=False),  
 return\_train\_score=True)  
  
gridsearch.fit(train\_x, train\_y)  
  
knn\_predicted = gridsearch.best\_estimator\_.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, knn\_predicted) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, knn\_predicted)}')  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, knn\_predicted) \* 100, 4),'KNN'))

MAPE: 32.8019%  
MAE: 370.6324404761905

## Lasso

Linear Model trained with L1 prior as regularizer (aka the Lasso).

Like linear regression and ElasticNet before it, it creates a linear model with a regulariser who appropriately dictates the model for the use of insignificant variables. The Lasso model gave us a MAPE of 61.38%, which is comparable to linear regression and ElasticNet.

from sklearn import linear\_model  
pip\_Lasso = Pipeline([("scaler", RobustScaler()),  
 ("classifier", linear\_model.Lasso())])  
  
lasso\_alphas = np.linspace(0, 0.2, 21)  
grid = dict()  
grid['classifier\_\_alpha'] = lasso\_alphas  
  
grid\_search\_Lasso = GridSearchCV(pip\_Lasso, grid, cv=3,  
 scoring=make\_scorer(mean\_absolute\_percentage\_error, greater\_is\_better=False),  
 return\_train\_score=True)  
grid\_search\_Lasso.fit(train\_x, train\_y)  
y\_reg = grid\_search\_Lasso.best\_estimator\_.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_reg) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_reg)}')  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, y\_reg) \* 100, 4),'Lasso'))

MAPE: 61.3846%  
MAE: 604.1177804699292

## Tweedie Regressor

Generalized Linear Model with a Tweedie distribution.

This estimator can be used to model different GLMs depending on the power parameter, which determines the underlying distribution.

Our model achieved a moderate MAPE score of 45.88% (better than linear regression, ElasticNet or Lasso), but worse than KNN or Random Forest.

reg = TweedieRegressor(power=1, alpha=0.1, link='log')  
reg.fit(train\_x, train\_y)  
y\_reg = reg.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_reg) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_reg)}')  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, y\_reg) \* 100, 4),'TweedieRegressor'))

MAPE: 45.8844%  
MAE: 535.648143185541

## Gradient Boosting Regressor

Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is a generalization of boosting to arbitrary differentiable loss functions. Source: <https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting>

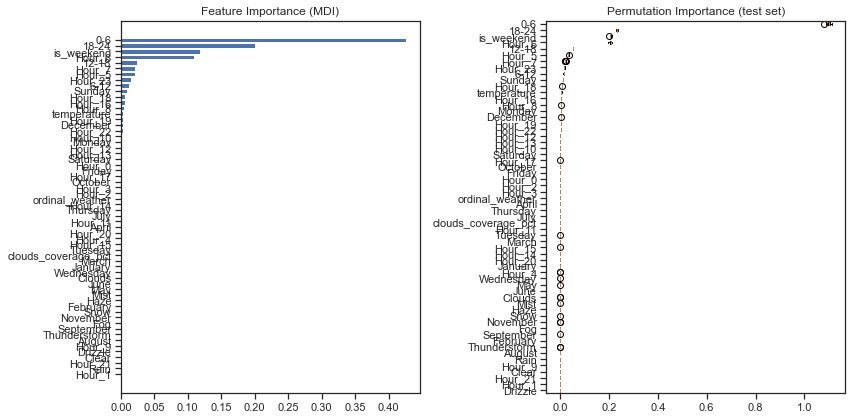
The algorithm seeks a better global minimum of the cost function by adjusting the available parameters:

* n\_estimators
* max\_depth
* min\_samples\_split
* learning\_rate

This model achieved a MAPE of 37.78%, which is also a moderately good result when compared to others. In addition, we note the Feature Importance below. From the graph we can see that some of the most significant variables were the hourly intervals from 24 to 6 and from 18 to 24, as well as whether it is currently the weekend.

params = {  
 "n\_estimators": 500,  
 "max\_depth": 4,  
 "min\_samples\_split": 5,  
 "learning\_rate": 0.01,  
 "loss": "squared\_error",  
}  
reg = GradientBoostingRegressor(\*\*params)  
reg.fit(train\_x, train\_y)  
  
y\_reg = reg.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, y\_reg) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, y\_reg)}')  
feature\_importance = reg.feature\_importances\_  
sorted\_idx = np.argsort(feature\_importance)  
pos = np.arange(sorted\_idx.shape[0]) + 0.5  
fig = plt.figure(figsize=(12, 6))  
plt.subplot(1, 2, 1)  
plt.barh(pos, feature\_importance[sorted\_idx], align="center")  
plt.yticks(pos, np.array(train\_x.columns)[sorted\_idx])  
plt.title("Feature Importance (MDI)")  
  
result = permutation\_importance(  
 reg, train\_x, train\_y, n\_repeats=10, random\_state=42, n\_jobs=2  
)  
sorted\_idx = result.importances\_mean.argsort()  
plt.subplot(1, 2, 2)  
plt.boxplot(  
 result.importances[sorted\_idx].T,  
 vert=False,  
 labels=np.array(train\_x.columns)[sorted\_idx],  
)  
plt.title("Permutation Importance (test set)")  
fig.tight\_layout()  
plt.show()  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, y\_reg) \* 100, 4),'GradientBoostingRegressor'))

MAPE: 37.7817%  
MAE: 386.62310716167445



## LightGBM

Similar to XGBoost, LightGBM (by Microsoft) is a distributed high-performance framework that uses decision trees for ranking, classification, and regression tasks.

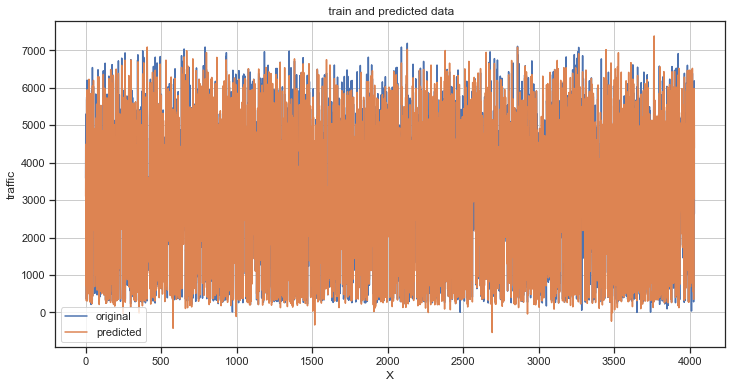
The advantages are as follows:

* Faster training speed and accuracy resulting from LightGBM being a histogram-based algorithm that performs bucketing of values (also requires lesser memory)
* Also compatible with large and complex datasets but is much faster during training
* Support for both parallel learning and GPU learning

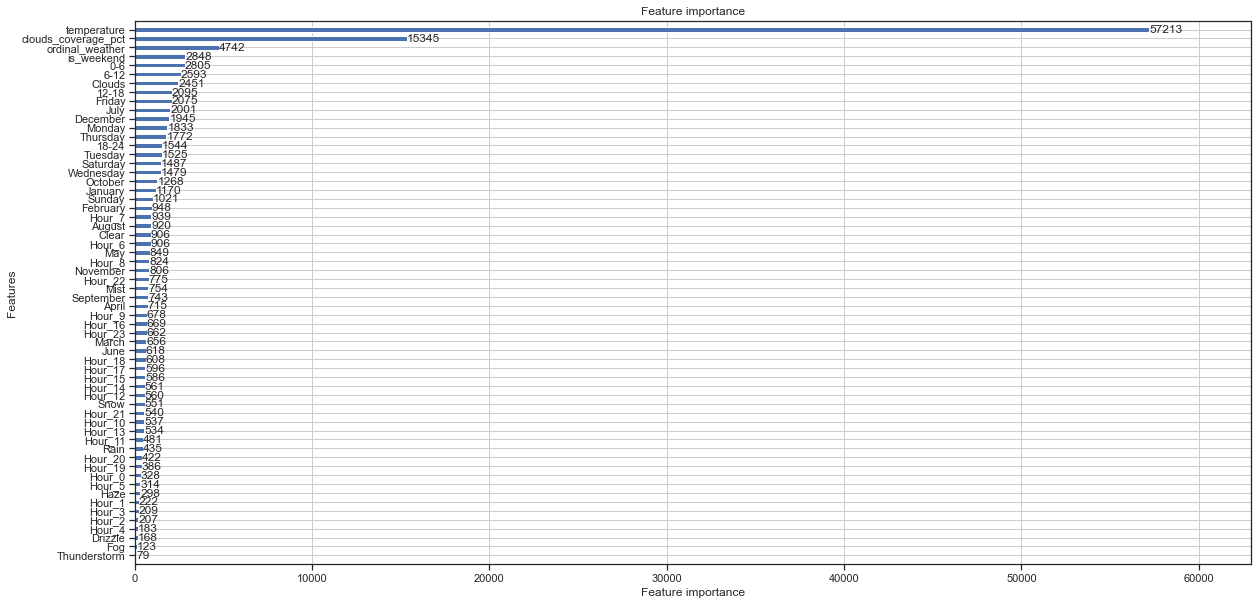
Source: <https://neptune.ai/blog/xgboost-vs-lightgbm> & <https://lightgbm.readthedocs.io/en/latest/>

import lightgbm as lgb  
from sklearn.metrics import mean\_absolute\_percentage\_error, mean\_absolute\_error, r2\_score  
  
params = {  
 'task': 'train',   
 'boosting': 'gbdt',  
 'n\_estimators': 900,  
 'learning\_rate': 0.3,  
 'objective': 'regression',  
 'num\_leaves': 4096,  
 'max\_depth' : 12,  
 'metric': {'l2','l1'},  
 'verbose': -1  
}  
  
lgb\_train = lgb.Dataset(train\_x, train\_y)  
  
  
model = lgb.train(params,  
 train\_set=lgb\_train)  
 #valid\_sets=y\_train,  
 #early\_stopping\_rounds=30)  
  
  
lgm\_pred = model.predict(test\_x)  
  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, lgm\_pred) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, lgm\_pred)}')  
  
x\_ax = range(len(test\_y))  
plt.figure(figsize=(12, 6))  
plt.plot(x\_ax, test\_y, label="original")  
plt.plot(x\_ax, lgm\_pred, label="predicted")  
plt.title(" train and predicted data")  
plt.xlabel('X')  
plt.ylabel('traffic')  
plt.legend(loc='best',fancybox=True, shadow=False)  
plt.grid(True)  
plt.show()   
  
lgb.plot\_importance(model, height=.5)

MAPE: 27.0738%  
MAE: 287.1861314765379



<AxesSubplot:title={'center':'Feature importance'}, xlabel='Feature importance', ylabel='Features'>



## LightGBM with Optuna Optimizer

Optuna is An open source hyperparameter optimization framework to automate hyperparameter search.

LightGBM together with Optuna is currently one of the fastest boosting algorithms. This is indicated by many benchmarks available on the internet in comparison with XGBoost. Optuna creates a study in which the algorithm finds the global minimum of the cost function in a very fast and economical way via a gradient function.

This combination of the two algorithms allowed us to obtain the lowest possible MAPE of 23.62%, and this learned model will be used further for prediction on the test set.

Looking at the Feature Importance for this model we can see that it was, however, the level of temperature, the level of cloud cover, the weather conditions and whether it was currently the weekend that most influenced the algorithm's decision making.

def objective(trial):  
 dtrain = lgb.Dataset(train\_x, label=train\_y)  
   
 param = {  
 'objective': 'regression',  
 'metric': {'l2','l1'},  
 'n\_estimators': trial.suggest\_int('n\_estimators', 100, 1000),  
 'learning\_rate': trial.suggest\_loguniform('learning\_rate', 0.01, 1),  
 'num\_leaves': trial.suggest\_int('num\_leaves', 2, 4096),  
 'max\_depth': trial.suggest\_int('max\_depth', 2, 8),  
 'task': 'train',   
 'boosting': 'gbdt',  
 'verbose': -1  
 }  
   
 gbm = lgb.train(param, dtrain)  
 preds = gbm.predict(test\_x)  
 mape = sklearn.metrics.mean\_absolute\_percentage\_error(test\_y, preds)  
 return mape  
   
study = optuna.create\_study(direction='minimize')  
study.optimize(objective, n\_trials=100)  
   
print('Number of finished trials:', len(study.trials))  
print('Best trial:', study.best\_trial.params)

[I 2022-06-03 10:36:14,929] A new study created in memory with name: no-name-c5a79431-fbb1-421c-973b-a4dce1120798  
[I 2022-06-03 10:36:16,029] Trial 0 finished with value: 0.28892860122044783 and parameters: {'n\_estimators': 590, 'learning\_rate': 0.9573212602687539, 'num\_leaves': 1849, 'max\_depth': 8}. Best is trial 0 with value: 0.28892860122044783.  
[I 2022-06-03 10:36:16,171] Trial 1 finished with value: 0.4120748099361683 and parameters: {'n\_estimators': 270, 'learning\_rate': 0.035762025271736324, 'num\_leaves': 3333, 'max\_depth': 3}. Best is trial 0 with value: 0.28892860122044783.  
[I 2022-06-03 10:36:16,339] Trial 2 finished with value: 0.6327079759522667 and parameters: {'n\_estimators': 478, 'learning\_rate': 0.012987429229243804, 'num\_leaves': 3969, 'max\_depth': 2}. Best is trial 0 with value: 0.28892860122044783.  
[I 2022-06-03 10:36:16,705] Trial 3 finished with value: 0.3060208613356925 and parameters: {'n\_estimators': 911, 'learning\_rate': 0.03578265733453882, 'num\_leaves': 3283, 'max\_depth': 3}. Best is trial 0 with value: 0.28892860122044783.  
[I 2022-06-03 10:36:17,183] Trial 4 finished with value: 0.2474829977567503 and parameters: {'n\_estimators': 273, 'learning\_rate': 0.15393056605666852, 'num\_leaves': 3889, 'max\_depth': 8}. Best is trial 4 with value: 0.2474829977567503.  
[I 2022-06-03 10:36:17,558] Trial 5 finished with value: 0.344929504491311 and parameters: {'n\_estimators': 946, 'learning\_rate': 0.022820940727540286, 'num\_leaves': 155, 'max\_depth': 3}. Best is trial 4 with value: 0.2474829977567503.  
[I 2022-06-03 10:36:17,822] Trial 6 finished with value: 0.2944929419801828 and parameters: {'n\_estimators': 306, 'learning\_rate': 0.025971034953190714, 'num\_leaves': 4078, 'max\_depth': 5}. Best is trial 4 with value: 0.2474829977567503.  
[I 2022-06-03 10:36:18,528] Trial 7 finished with value: 0.24605549139398153 and parameters: {'n\_estimators': 717, 'learning\_rate': 0.14773223813810504, 'num\_leaves': 1774, 'max\_depth': 6}. Best is trial 7 with value: 0.24605549139398153.  
[I 2022-06-03 10:36:19,487] Trial 8 finished with value: 0.24474402749492052 and parameters: {'n\_estimators': 543, 'learning\_rate': 0.0662144066734682, 'num\_leaves': 3905, 'max\_depth': 8}. Best is trial 8 with value: 0.24474402749492052.  
[I 2022-06-03 10:36:19,716] Trial 9 finished with value: 0.4365663275285835 and parameters: {'n\_estimators': 722, 'learning\_rate': 0.028056377363204356, 'num\_leaves': 2389, 'max\_depth': 2}. Best is trial 8 with value: 0.24474402749492052.  
[I 2022-06-03 10:36:19,879] Trial 10 finished with value: 0.2515127635689107 and parameters: {'n\_estimators': 119, 'learning\_rate': 0.472526380358355, 'num\_leaves': 661, 'max\_depth': 6}. Best is trial 8 with value: 0.24474402749492052.  
[I 2022-06-03 10:36:20,641] Trial 11 finished with value: 0.24731001508515724 and parameters: {'n\_estimators': 730, 'learning\_rate': 0.10995397476622899, 'num\_leaves': 1693, 'max\_depth': 6}. Best is trial 8 with value: 0.24474402749492052.  
[I 2022-06-03 10:36:21,377] Trial 12 finished with value: 0.24720768108125435 and parameters: {'n\_estimators': 594, 'learning\_rate': 0.19907899521848654, 'num\_leaves': 1271, 'max\_depth': 7}. Best is trial 8 with value: 0.24474402749492052.  
[I 2022-06-03 10:36:21,960] Trial 13 finished with value: 0.2515376677080685 and parameters: {'n\_estimators': 777, 'learning\_rate': 0.06775274889984072, 'num\_leaves': 2502, 'max\_depth': 5}. Best is trial 8 with value: 0.24474402749492052.  
[I 2022-06-03 10:36:22,554] Trial 14 finished with value: 0.2499003193472187 and parameters: {'n\_estimators': 467, 'learning\_rate': 0.2776601951581675, 'num\_leaves': 2867, 'max\_depth': 7}. Best is trial 8 with value: 0.24474402749492052.  
[I 2022-06-03 10:36:23,727] Trial 15 finished with value: 0.2431493293446067 and parameters: {'n\_estimators': 837, 'learning\_rate': 0.06643731931673084, 'num\_leaves': 1046, 'max\_depth': 7}. Best is trial 15 with value: 0.2431493293446067.  
[I 2022-06-03 10:36:25,219] Trial 16 finished with value: 0.24323951246953265 and parameters: {'n\_estimators': 847, 'learning\_rate': 0.06424054761187592, 'num\_leaves': 1121, 'max\_depth': 8}. Best is trial 15 with value: 0.2431493293446067.  
[I 2022-06-03 10:36:26,412] Trial 17 finished with value: 0.24193940875989547 and parameters: {'n\_estimators': 860, 'learning\_rate': 0.0636314493933737, 'num\_leaves': 935, 'max\_depth': 7}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:26,818] Trial 18 finished with value: 0.28369249225886617 and parameters: {'n\_estimators': 994, 'learning\_rate': 0.05021293646334947, 'num\_leaves': 6, 'max\_depth': 7}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:27,303] Trial 19 finished with value: 0.31732498569511236 and parameters: {'n\_estimators': 839, 'learning\_rate': 0.01445441853623153, 'num\_leaves': 644, 'max\_depth': 4}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:27,928] Trial 20 finished with value: 0.2524769901418894 and parameters: {'n\_estimators': 858, 'learning\_rate': 0.09496054488163175, 'num\_leaves': 1209, 'max\_depth': 5}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:29,027] Trial 21 finished with value: 0.24916714199185633 and parameters: {'n\_estimators': 821, 'learning\_rate': 0.06450012393547042, 'num\_leaves': 1043, 'max\_depth': 7}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:30,166] Trial 22 finished with value: 0.24911996497383798 and parameters: {'n\_estimators': 678, 'learning\_rate': 0.09856187823138897, 'num\_leaves': 744, 'max\_depth': 8}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:31,325] Trial 23 finished with value: 0.24445205662624764 and parameters: {'n\_estimators': 904, 'learning\_rate': 0.044784779618577755, 'num\_leaves': 1505, 'max\_depth': 7}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:32,278] Trial 24 finished with value: 0.2500332438174577 and parameters: {'n\_estimators': 992, 'learning\_rate': 0.08007311011352217, 'num\_leaves': 355, 'max\_depth': 6}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:33,941] Trial 25 finished with value: 0.25018617765467094 and parameters: {'n\_estimators': 652, 'learning\_rate': 0.017827330842612704, 'num\_leaves': 1075, 'max\_depth': 8}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:35,093] Trial 26 finished with value: 0.24371044960974117 and parameters: {'n\_estimators': 796, 'learning\_rate': 0.27398630481274866, 'num\_leaves': 865, 'max\_depth': 7}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:36,241] Trial 27 finished with value: 0.2511110723170001 and parameters: {'n\_estimators': 890, 'learning\_rate': 0.04788831643325428, 'num\_leaves': 2125, 'max\_depth': 6}. Best is trial 17 with value: 0.24193940875989547.  
[I 2022-06-03 10:36:37,661] Trial 28 finished with value: 0.23915842016476194 and parameters: {'n\_estimators': 773, 'learning\_rate': 0.12734940070132397, 'num\_leaves': 429, 'max\_depth': 8}. Best is trial 28 with value: 0.23915842016476194.  
[I 2022-06-03 10:36:38,903] Trial 29 finished with value: 0.27794680272719013 and parameters: {'n\_estimators': 628, 'learning\_rate': 0.8478640860678558, 'num\_leaves': 277, 'max\_depth': 8}. Best is trial 28 with value: 0.23915842016476194.  
[I 2022-06-03 10:36:40,001] Trial 30 finished with value: 0.24252471896100278 and parameters: {'n\_estimators': 764, 'learning\_rate': 0.1224371349017605, 'num\_leaves': 432, 'max\_depth': 7}. Best is trial 28 with value: 0.23915842016476194.  
[I 2022-06-03 10:36:41,013] Trial 31 finished with value: 0.2398142104999035 and parameters: {'n\_estimators': 765, 'learning\_rate': 0.136796541720543, 'num\_leaves': 457, 'max\_depth': 7}. Best is trial 28 with value: 0.23915842016476194.  
[I 2022-06-03 10:36:42,137] Trial 32 finished with value: 0.23627086271513884 and parameters: {'n\_estimators': 733, 'learning\_rate': 0.1579544387440169, 'num\_leaves': 468, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.

[I 2022-06-03 10:36:43,286] Trial 33 finished with value: 0.2573513765508389 and parameters: {'n\_estimators': 752, 'learning\_rate': 0.21265502181845963, 'num\_leaves': 519, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:44,039] Trial 34 finished with value: 0.24418890257140033 and parameters: {'n\_estimators': 499, 'learning\_rate': 0.2907586042048352, 'num\_leaves': 84, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:45,180] Trial 35 finished with value: 0.25866221769648257 and parameters: {'n\_estimators': 682, 'learning\_rate': 0.46158905529752303, 'num\_leaves': 874, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:45,729] Trial 36 finished with value: 0.24315778614708414 and parameters: {'n\_estimators': 415, 'learning\_rate': 0.1520128975261635, 'num\_leaves': 441, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:46,226] Trial 37 finished with value: 0.2599772723047272 and parameters: {'n\_estimators': 922, 'learning\_rate': 0.1910604221681323, 'num\_leaves': 1466, 'max\_depth': 4}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:47,238] Trial 38 finished with value: 0.24858985830256353 and parameters: {'n\_estimators': 622, 'learning\_rate': 0.40986893915823247, 'num\_leaves': 230, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:47,788] Trial 39 finished with value: 0.24779486561134578 and parameters: {'n\_estimators': 562, 'learning\_rate': 0.14825360970959317, 'num\_leaves': 568, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:49,469] Trial 40 finished with value: 0.24452628710715493 and parameters: {'n\_estimators': 954, 'learning\_rate': 0.034240857996317646, 'num\_leaves': 880, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:50,465] Trial 41 finished with value: 0.24051429426181414 and parameters: {'n\_estimators': 782, 'learning\_rate': 0.12465272246428721, 'num\_leaves': 431, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:51,409] Trial 42 finished with value: 0.2442373848938857 and parameters: {'n\_estimators': 791, 'learning\_rate': 0.12981698876384268, 'num\_leaves': 257, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:52,078] Trial 43 finished with value: 0.2518214627785657 and parameters: {'n\_estimators': 694, 'learning\_rate': 0.1861613155069533, 'num\_leaves': 761, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:53,168] Trial 44 finished with value: 0.24779526314875644 and parameters: {'n\_estimators': 880, 'learning\_rate': 0.08953449695559611, 'num\_leaves': 3466, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:54,166] Trial 45 finished with value: 0.2426971092518879 and parameters: {'n\_estimators': 745, 'learning\_rate': 0.12114548781518197, 'num\_leaves': 56, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:55,088] Trial 46 finished with value: 0.24286438442383346 and parameters: {'n\_estimators': 806, 'learning\_rate': 0.22474667659531472, 'num\_leaves': 521, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:56,165] Trial 47 finished with value: 0.2422910255022487 and parameters: {'n\_estimators': 706, 'learning\_rate': 0.16607274246329487, 'num\_leaves': 1372, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:56,976] Trial 48 finished with value: 0.2481742240121838 and parameters: {'n\_estimators': 961, 'learning\_rate': 0.3490421461857616, 'num\_leaves': 2052, 'max\_depth': 5}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:57,378] Trial 49 finished with value: 0.2508301407992834 and parameters: {'n\_estimators': 176, 'learning\_rate': 0.08328413664516236, 'num\_leaves': 200, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:57,517] Trial 50 finished with value: 0.3762234454589631 and parameters: {'n\_estimators': 347, 'learning\_rate': 0.10841399205244236, 'num\_leaves': 723, 'max\_depth': 2}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:58,395] Trial 51 finished with value: 0.24339768390136887 and parameters: {'n\_estimators': 697, 'learning\_rate': 0.17081946147612712, 'num\_leaves': 1358, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:36:59,256] Trial 52 finished with value: 0.2490071914420621 and parameters: {'n\_estimators': 719, 'learning\_rate': 0.24319330799361363, 'num\_leaves': 1639, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:00,209] Trial 53 finished with value: 0.24773142937555215 and parameters: {'n\_estimators': 783, 'learning\_rate': 0.14103013119071575, 'num\_leaves': 993, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:01,606] Trial 54 finished with value: 0.2446639199459005 and parameters: {'n\_estimators': 867, 'learning\_rate': 0.07575220291941821, 'num\_leaves': 621, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:02,235] Trial 55 finished with value: 0.24424651614545126 and parameters: {'n\_estimators': 659, 'learning\_rate': 0.1734594150620756, 'num\_leaves': 368, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:02,688] Trial 56 finished with value: 0.2625383808020475 and parameters: {'n\_estimators': 821, 'learning\_rate': 0.09732564549986496, 'num\_leaves': 1242, 'max\_depth': 4}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:03,550] Trial 57 finished with value: 0.24810980378635927 and parameters: {'n\_estimators': 577, 'learning\_rate': 0.05649869077212087, 'num\_leaves': 1904, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:04,462] Trial 58 finished with value: 0.24427862377356144 and parameters: {'n\_estimators': 732, 'learning\_rate': 0.11383093516233805, 'num\_leaves': 847, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:04,677] Trial 59 finished with value: 0.6688624318015521 and parameters: {'n\_estimators': 766, 'learning\_rate': 0.036770180904013995, 'num\_leaves': 2, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:05,206] Trial 60 finished with value: 0.24035768911947622 and parameters: {'n\_estimators': 531, 'learning\_rate': 0.3228527714696259, 'num\_leaves': 999, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:05,697] Trial 61 finished with value: 0.267295085339746 and parameters: {'n\_estimators': 499, 'learning\_rate': 0.6047523925265724, 'num\_leaves': 959, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:06,273] Trial 62 finished with value: 0.24885389009957637 and parameters: {'n\_estimators': 603, 'learning\_rate': 0.25113595071161715, 'num\_leaves': 668, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:06,859] Trial 63 finished with value: 0.24431469520136795 and parameters: {'n\_estimators': 451, 'learning\_rate': 0.1353192970635749, 'num\_leaves': 1149, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:07,620] Trial 64 finished with value: 0.25991092464875604 and parameters: {'n\_estimators': 531, 'learning\_rate': 0.6240389510873844, 'num\_leaves': 471, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:08,199] Trial 65 finished with value: 0.24990308828833482 and parameters: {'n\_estimators': 639, 'learning\_rate': 0.33596975183119543, 'num\_leaves': 1389, 'max\_depth': 5}. Best is trial 32 with value: 0.23627086271513884.

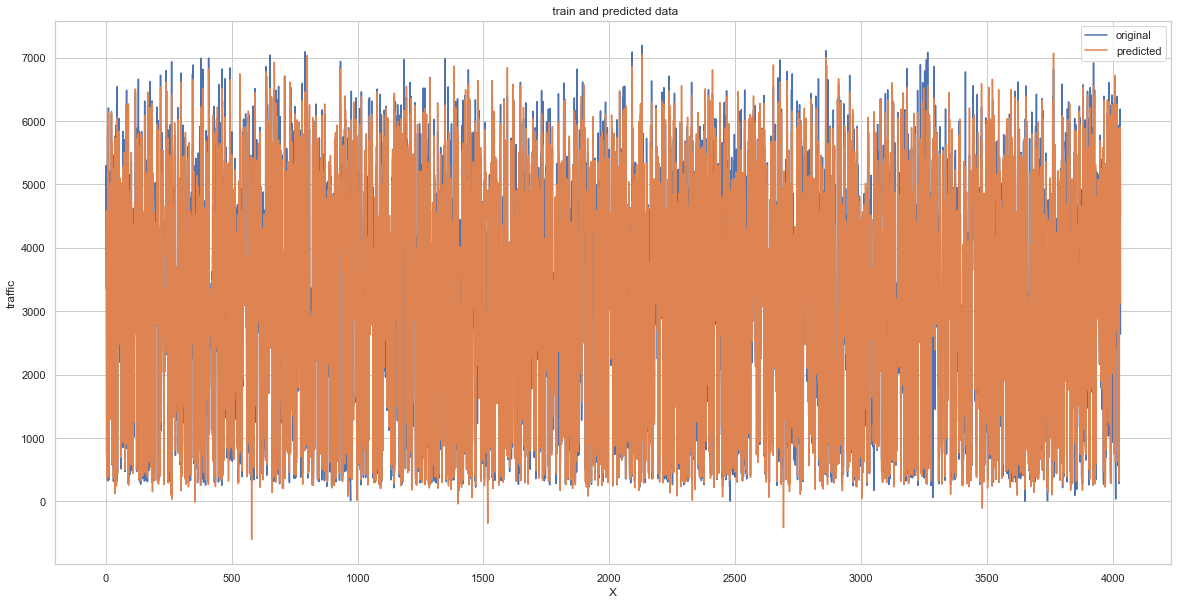
[I 2022-06-03 10:37:09,350] Trial 66 finished with value: 0.2412679115227688 and parameters: {'n\_estimators': 829, 'learning\_rate': 0.16601252163191482, 'num\_leaves': 2282, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:11,128] Trial 67 finished with value: 0.24848437355071626 and parameters: {'n\_estimators': 919, 'learning\_rate': 0.0753972798467698, 'num\_leaves': 2338, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:12,115] Trial 68 finished with value: 0.24898347180547528 and parameters: {'n\_estimators': 844, 'learning\_rate': 0.10598108812322653, 'num\_leaves': 2416, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:13,282] Trial 69 finished with value: 0.2390514584819428 and parameters: {'n\_estimators': 815, 'learning\_rate': 0.2109233670260973, 'num\_leaves': 345, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:14,514] Trial 70 finished with value: 0.24237123966605026 and parameters: {'n\_estimators': 753, 'learning\_rate': 0.33700060768077417, 'num\_leaves': 2728, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:15,538] Trial 71 finished with value: 0.24486229424136166 and parameters: {'n\_estimators': 811, 'learning\_rate': 0.20852462790334333, 'num\_leaves': 352, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:16,639] Trial 72 finished with value: 0.24535991217484363 and parameters: {'n\_estimators': 862, 'learning\_rate': 0.15574977408139595, 'num\_leaves': 163, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:17,237] Trial 73 finished with value: 0.24945362398292412 and parameters: {'n\_estimators': 825, 'learning\_rate': 0.19417113843044947, 'num\_leaves': 345, 'max\_depth': 5}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:18,425] Trial 74 finished with value: 0.24009663888762883 and parameters: {'n\_estimators': 890, 'learning\_rate': 0.056344354042328126, 'num\_leaves': 2259, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:20,646] Trial 75 finished with value: 0.25223337534609763 and parameters: {'n\_estimators': 894, 'learning\_rate': 0.01024854026804803, 'num\_leaves': 2861, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:21,796] Trial 76 finished with value: 0.24976787109550466 and parameters: {'n\_estimators': 934, 'learning\_rate': 0.3036985956490166, 'num\_leaves': 2631, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:22,511] Trial 77 finished with value: 0.24287138095753602 and parameters: {'n\_estimators': 775, 'learning\_rate': 0.2395247541897984, 'num\_leaves': 2178, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:23,712] Trial 78 finished with value: 0.24633702108449274 and parameters: {'n\_estimators': 971, 'learning\_rate': 0.12050849508748798, 'num\_leaves': 2233, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:25,114] Trial 79 finished with value: 0.24000279093036328 and parameters: {'n\_estimators': 799, 'learning\_rate': 0.13263299118673982, 'num\_leaves': 1882, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:26,464] Trial 80 finished with value: 0.24306406128018818 and parameters: {'n\_estimators': 669, 'learning\_rate': 0.057912065709003556, 'num\_leaves': 1835, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:27,726] Trial 81 finished with value: 0.24304571009201495 and parameters: {'n\_estimators': 834, 'learning\_rate': 0.142400439684015, 'num\_leaves': 1661, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:28,931] Trial 82 finished with value: 0.23667529415758176 and parameters: {'n\_estimators': 791, 'learning\_rate': 0.1808787657263937, 'num\_leaves': 3181, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:30,199] Trial 83 finished with value: 0.24303780597333294 and parameters: {'n\_estimators': 735, 'learning\_rate': 0.0905308594401789, 'num\_leaves': 3222, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:32,011] Trial 84 finished with value: 0.2438674915515849 and parameters: {'n\_estimators': 795, 'learning\_rate': 0.13004561242527124, 'num\_leaves': 3591, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:33,927] Trial 85 finished with value: 0.24316798161318215 and parameters: {'n\_estimators': 770, 'learning\_rate': 0.21178387819565697, 'num\_leaves': 3131, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:35,770] Trial 86 finished with value: 0.2590132264374649 and parameters: {'n\_estimators': 883, 'learning\_rate': 0.420326458342135, 'num\_leaves': 3043, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:37,742] Trial 87 finished with value: 0.24559506796377942 and parameters: {'n\_estimators': 799, 'learning\_rate': 0.022478651788333096, 'num\_leaves': 3790, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:39,139] Trial 88 finished with value: 0.24623927320464079 and parameters: {'n\_estimators': 747, 'learning\_rate': 0.185254883961714, 'num\_leaves': 4093, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:39,466] Trial 89 finished with value: 0.2696055165812379 and parameters: {'n\_estimators': 713, 'learning\_rate': 0.16173570559821737, 'num\_leaves': 134, 'max\_depth': 3}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:40,794] Trial 90 finished with value: 0.250643629735268 and parameters: {'n\_estimators': 536, 'learning\_rate': 0.10166904316883499, 'num\_leaves': 1962, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:42,449] Trial 91 finished with value: 0.23829446730432843 and parameters: {'n\_estimators': 865, 'learning\_rate': 0.1835142246961923, 'num\_leaves': 558, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:44,071] Trial 92 finished with value: 0.2622863132096334 and parameters: {'n\_estimators': 908, 'learning\_rate': 0.26600689614936596, 'num\_leaves': 547, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:45,554] Trial 93 finished with value: 0.25089341440578544 and parameters: {'n\_estimators': 856, 'learning\_rate': 0.22231061867992116, 'num\_leaves': 783, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:46,653] Trial 94 finished with value: 0.24498031507288778 and parameters: {'n\_estimators': 876, 'learning\_rate': 0.11841712024449962, 'num\_leaves': 588, 'max\_depth': 7}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:47,946] Trial 95 finished with value: 0.24370052708575377 and parameters: {'n\_estimators': 784, 'learning\_rate': 0.13646400497432218, 'num\_leaves': 431, 'max\_depth': 8}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:48,807] Trial 96 finished with value: 0.23866996569139348 and parameters: {'n\_estimators': 848, 'learning\_rate': 0.17788757825466117, 'num\_leaves': 311, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:49,802] Trial 97 finished with value: 0.24246283408990743 and parameters: {'n\_estimators': 803, 'learning\_rate': 0.18360262488059373, 'num\_leaves': 287, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.  
[I 2022-06-03 10:37:51,316] Trial 98 finished with value: 0.24081450292884485 and parameters: {'n\_estimators': 941, 'learning\_rate': 0.1488905920373234, 'num\_leaves': 692, 'max\_depth': 6}. Best is trial 32 with value: 0.23627086271513884.

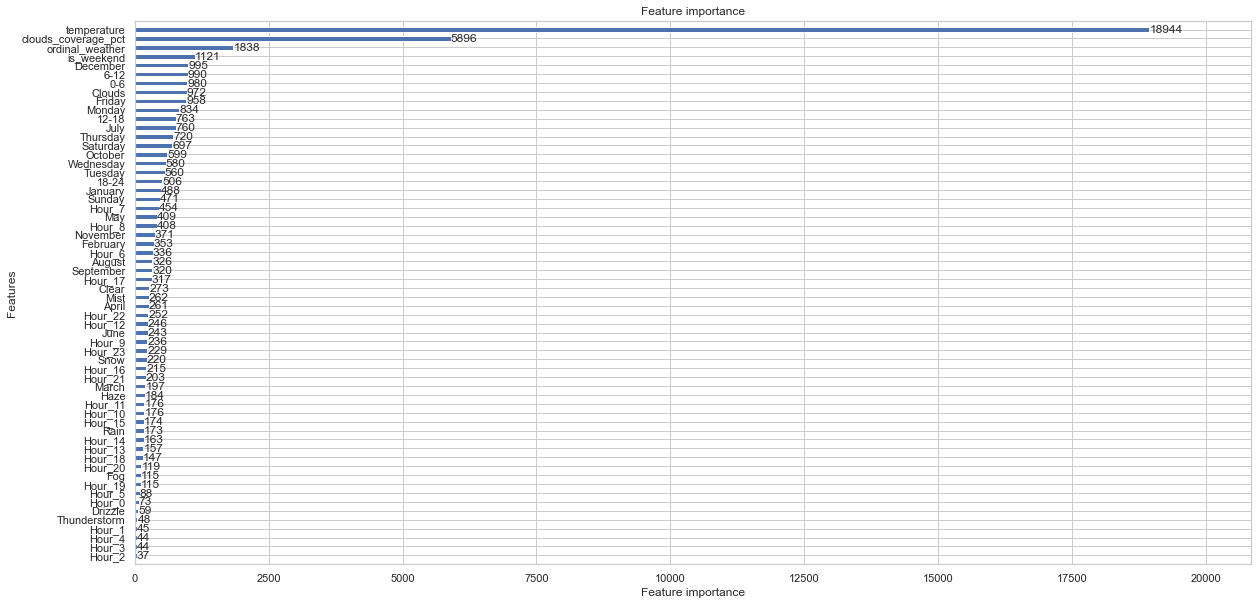
[I 2022-06-03 10:37:52,302] Trial 99 finished with value: 0.2446495016539613 and parameters: {'n\_estimators': 854, 'learning\_rate': 0.3075323578792076, 'num\_leaves': 102, 'max\_depth': 5}. Best is trial 32 with value: 0.23627086271513884.

Number of finished trials: 100  
Best trial: {'n\_estimators': 733, 'learning\_rate': 0.1579544387440169, 'num\_leaves': 468, 'max\_depth': 8}

import lightgbm as lgb  
from sklearn.metrics import mean\_absolute\_percentage\_error, mean\_absolute\_error, r2\_score  
  
params = {  
 'task': 'train',   
 'boosting': 'gbdt',  
 'n\_estimators': study.best\_trial.params['n\_estimators'],  
 'learning\_rate': study.best\_trial.params['learning\_rate'],  
 'objective': 'regression',  
 'num\_leaves': study.best\_trial.params['num\_leaves'],  
 'max\_depth' : study.best\_trial.params['max\_depth'],  
 'metric': {'l2','l1'},  
 'verbose': -1  
}  
  
lgb\_train = lgb.Dataset(train\_x, train\_y)  
  
  
model = lgb.train(params,  
 train\_set=lgb\_train)  
 #valid\_sets=y\_train,  
 #early\_stopping\_rounds=30)  
  
  
lgm\_pred = model.predict(test\_x)  
  
print(f'MAPE: {round(mean\_absolute\_percentage\_error(test\_y, lgm\_pred) \* 100, 4)}%')  
print(f'MAE: {mean\_absolute\_error(test\_y, lgm\_pred)}')  
  
x\_ax = range(len(test\_y))  
plt.figure(figsize=(20, 10))  
plt.plot(x\_ax, test\_y, label="original")  
plt.plot(x\_ax, lgm\_pred, label="predicted")  
plt.title(" train and predicted data")  
plt.xlabel('X')  
plt.ylabel('traffic')  
plt.legend(loc='best',fancybox=True, shadow=False)  
plt.grid(True)  
plt.show()   
  
lgb.plot\_importance(model, height=.5)  
  
mapeOfModels.append((round(mean\_absolute\_percentage\_error(test\_y, lgm\_pred) \* 100, 4),'lightgbmWithOptuna'))

MAPE: 23.6271%  
MAE: 270.9755703741944



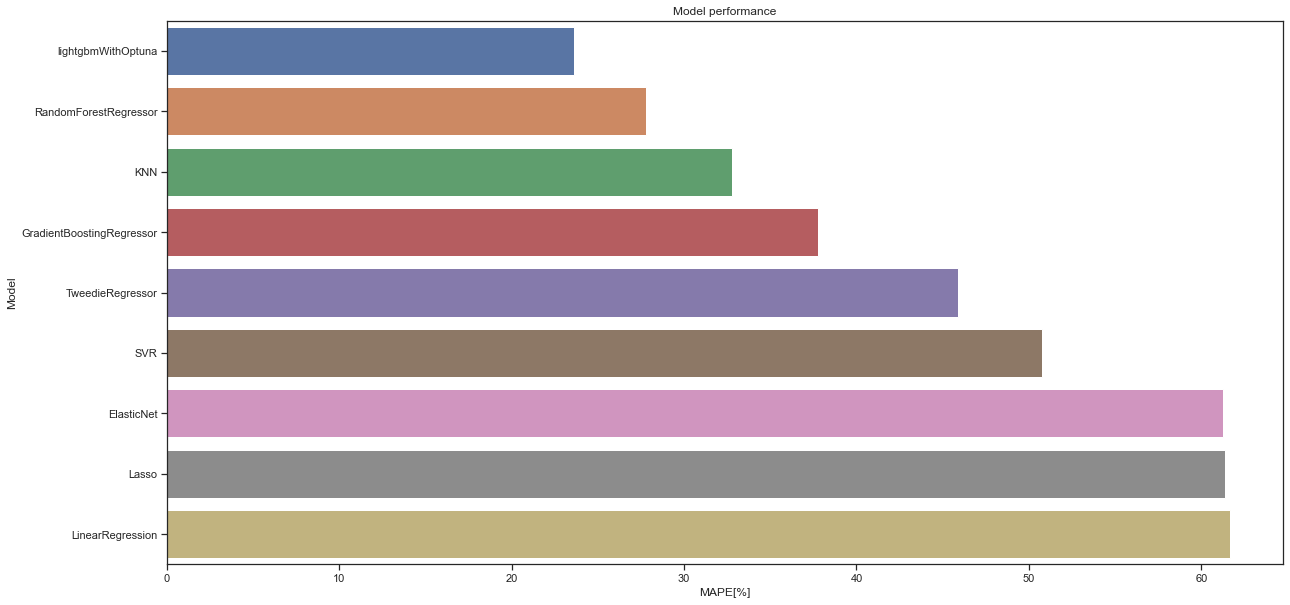


## Summary of the evaluation of the models

Below we can look at what effectiveness our proposed models have achieved. There is a clear advantage for algorithms such as **Random Forest** and **LightGBM with Optuna Optimizer**, which obtained **27.78%** and **23.62%** MAPE respectively. Finally, we will run our predictions on the additionally proposed model **LightGBM with Optuna Optimizer**, as it achieved the lowest MAPE cost function value.

mapeOfModels  
mapeOfModels.sort()

res = [[ i for i, j in mapeOfModels ],  
 [ j for i, j in mapeOfModels ]]  
ax = sns.barplot(x = res[0], y = res[1])  
ax.set(xlabel='MAPE[%]', ylabel='Model')  
plt.title("Model performance")  
plt.show()



In contrast, the machine learning models presented in class such as **Linear Regression**, **SVR**, **KNN** and **Lasso** scored moderately for the MAPE statistic, with the KNN model performing best with a score of **32.8%**.

# 5. Prediction on test dataset

traffic\_test = pd.read\_csv('traffic\_test.csv')  
  
traffic\_test = dayFromDate(traffic\_test)  
traffic\_test = ordinal\_whether(traffic\_test)  
traffic\_test = weather\_dummies\_test(traffic\_test)  
traffic\_test = outliers\_correction(traffic\_test)

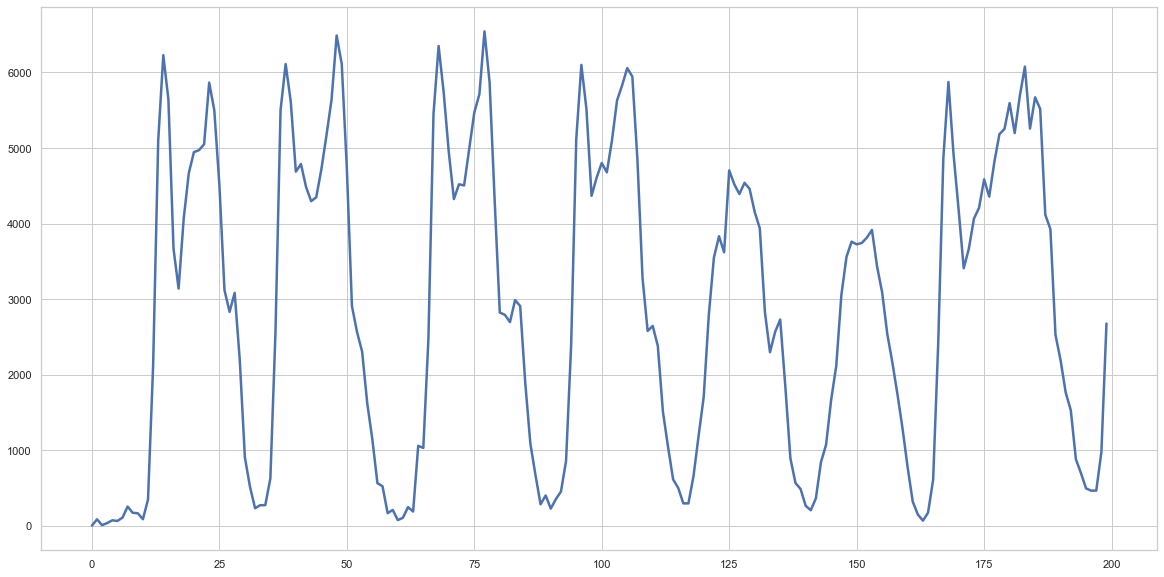
traffic\_test = traffic\_test.drop(columns = ['date\_time', 'weather\_general', 'weather\_detailed',   
 'month', 'year', 'day', 'hour', 'rain\_mm','snow\_mm','hour\_interval'])

traffic\_test.rename(columns={'1\_x':'January', '2\_x':'February', '3\_x':'March', '4\_x':'April',  
 '5\_x':'May','6\_x':'June','7\_x':'July', '8\_x':'August','9\_x':'September',  
 '10\_x':'November', '11\_x':'October', '12\_x':'December', '0\_x':'Hour\_0',  
 '1\_y':'Hour\_1','2\_y':'Hour\_2','3\_y':'Hour\_3','4\_y':'Hour\_4',  
 '5\_y':'Hour\_5','6\_y':'Hour\_6','7\_y':'Hour\_7','8\_y':'Hour\_8',  
 '9\_y':'Hour\_9','10\_y':'Hour\_10','11\_y':'Hour\_11','12\_y':'Hour\_12',  
 13:'Hour\_13',14:'Hour\_14',15:'Hour\_15',16:'Hour\_16',  
 17:'Hour\_17',18:'Hour\_18',19:'Hour\_19',20:'Hour\_20',  
 21:'Hour\_21',22:'Hour\_22',23:'Hour\_23','0\_y':'Monday',  
 1:'Tuesday', 2:'Wednesday', 3:'Thursday', 4:'Friday', 5:'Saturday', 6:'Sunday'}, inplace=True)

lgm\_pred = model.predict(traffic\_test)

result = [x if x > 0 else x \* (-1) for x in lgm\_pred]  
sns.set(rc = {'figure.figsize':(20,10)})  
sns.set\_theme(style="whitegrid")  
sns.lineplot(data=result[:200], palette="tab10", linewidth=2.5)

<AxesSubplot:>



# Save results  
submission = result  
pd.DataFrame(submission).to\_csv("final\_results\_reg.csv", header = None, index = None)