In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [3]:

```
!pip install polars
!pip install lets-plot
```

```
Collecting polars
  Downloading polars-0.16.17-cp37-abi3-win_amd64.whl (17.6 MB)
Requirement already satisfied: typing_extensions>=4.0.1 in c:\users\shre
ya\anaconda3\lib\site-packages (from polars) (4.5.0)
Installing collected packages: polars
Successfully installed polars-0.16.17
Collecting lets-plot
  Downloading lets_plot-3.1.0-cp38-cp38-win_amd64.whl (3.5 MB)
Collecting palettable
  Downloading palettable-3.3.0-py2.py3-none-any.whl (111 kB)
Collecting pypng
  Downloading pypng-0.20220715.0-py3-none-any.whl (58 kB)
Installing collected packages: pypng, palettable, lets-plot
Successfully installed lets-plot-3.1.0 palettable-3.3.0 pypng-0.2022071
5.0
```

In [6]:

```
pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.7.5-py3-none-win_amd64.whl (70.9 MB)
Requirement already satisfied: numpy in c:\users\shreya\anaconda3\lib\si
te-packages (from xgboost) (1.20.1)
Requirement already satisfied: scipy in c:\users\shreya\anaconda3\lib\si
te-packages (from xgboost) (1.6.2)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.5
Note: you may need to restart the kernel to use updated packages.
```

In [8]:

```
pip install optuna
```

```
Collecting optuna
  Downloading optuna-3.1.0-py3-none-any.whl (365 kB)
Requirement already satisfied: tqdm in c:\users\shreya\anaconda3\lib\sit
e-packages (from optuna) (4.59.0)
Collecting cmaes>=0.9.1
Note: you may need to restart the kernel to use updated packages.
  Downloading cmaes-0.9.1-py3-none-any.whl (21 kB)
Collecting colorlog
  Downloading colorlog-6.7.0-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: sqlalchemy>=1.3.0 in c:\users\shreya\anac
onda3\lib\site-packages (from optuna) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\shreya\anacon
da3\lib\site-packages (from optuna) (20.9)
Requirement already satisfied: numpy in c:\users\shreya\anaconda3\lib\si
te-packages (from optuna) (1.20.1)
Collecting alembic>=1.5.0
  Downloading alembic-1.10.2-py3-none-any.whl (212 kB)
Requirement already satisfied: PyYAML in c:\users\shreya\anaconda3\lib\s
ite-packages (from optuna) (5.4.1)
Requirement already satisfied: typing-extensions>=4 in c:\users\shreya\a
naconda3\lib\site-packages (from alembic>=1.5.0->optuna) (4.5.0)
Collecting importlib-resources
  Downloading importlib_resources-5.12.0-py3-none-any.whl (36 kB)
Requirement already satisfied: importlib-metadata in c:\users\shreya\ana
conda3\lib\site-packages (from alembic>=1.5.0->optuna) (3.10.0)
Collecting Mako
  Downloading Mako-1.2.4-py3-none-any.whl (78 kB)
Requirement already satisfied: pyparsing>=2.0.2 in c:\users\shreya\anaco
nda3\lib\site-packages (from packaging>=20.0->optuna) (2.4.7)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\shreya\anaco
nda3\lib\site-packages (from sqlalchemy>=1.3.0->optuna) (1.0.0)
Requirement already satisfied: colorama in c:\users\shreya\anaconda3\lib
\site-packages (from colorlog->optuna) (0.4.4)
Requirement already satisfied: zipp>=0.5 in c:\users\shreya\anaconda3\li
b\site-packages (from importlib-metadata->alembic>=1.5.0->optuna) (3.4.
1)
Requirement already satisfied: MarkupSafe>=0.9.2 in c:\users\shreya\anac
onda3\lib\site-packages (from Mako->alembic>=1.5.0->optuna) (1.1.1)
Installing collected packages: Mako, importlib-resources, colorlog, cmae
s, alembic, optuna
Successfully installed Mako-1.2.4 alembic-1.10.2 cmaes-0.9.1 colorlog-6.
7.0 importlib-resources-5.12.0 optuna-3.1.0
```

In [10]:

```
pip install plotly
```

```
Collecting plotly
  Downloading plotly-5.14.0-py2.py3-none-any.whl (15.3 MB)
Requirement already satisfied: packaging in c:\users\shreya\anaconda3\li
b\site-packages (from plotly) (20.9)
Collecting tenacity>=6.2.0
  Downloading tenacity-8.2.2-py3-none-any.whl (24 kB)
Requirement already satisfied: pyparsing>=2.0.2 in c:\users\shreya\anaco
nda3\lib\site-packages (from packaging->plotly) (2.4.7)
Installing collected packages: tenacity, plotly
Successfully installed plotly-5.14.0 tenacity-8.2.2
Note: you may need to restart the kernel to use updated packages.
```

In [11]:

```
import polars as pl
import xgboost as xgb
import numpy as np
import optuna
import math
import statistics as stat

from lets_plot import *
from lets_plot.mapping import as_discrete
from sklearn import model_selection
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
import plotly
import plotly.figure_factory as ff

LetsPlot.setup_html()
plotly.offline.init_notebook_mode(connected = True)
```

In [12]:

```
df = pl.read_csv("traffic.csv", parse_dates = True).drop("ID")
df = df.filter(pl.col("Junction") == 1) # Take only the first junction
df = df.with_row_count(name = "Time_index", offset = 0) # add time index column
df.head()
```

```
<ipython-input-12-04c3432364fd>:1: DeprecationWarning:

`parse_dates` is deprecated as an argument to `read_csv`; use `try_parse
_dates` instead.
```

Out[12]:

shape: (5, 4)

Time_index	DateTime	Junction	Vehicles
u32	datetime[μs]	i64	i64
0	2015-11-01 00:00:00	1	15
1	2015-11-01 01:00:00	1	13
2	2015-11-01 02:00:00	1	10
3	2015-11-01 03:00:00	1	7
4	2015-11-01 04:00:00	1	9

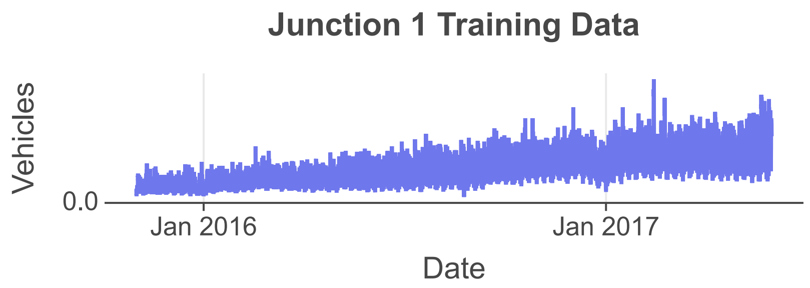
In [13]:

```
# Split into training and validation sets
df_train = df.filter(pl.col("DateTime") < pl.datetime(2017, 6, 1))
df_valid = df.filter(pl.col("DateTime") >= pl.datetime(2017, 6, 1))
```

In [14]:

```
j1_color = "#6f77ec"
linesize = 1

# Time series plot for each junction's training set
plt_ts = \
    ggplot(df_train)+\
    geom_line(aes(x = "DateTime", y = "Vehicles"),
              color = j1_color, sampling = "none", size = linesize)+\
    scale_x_datetime(format = "%b %Y")+\
    theme_minimal2()+\
    theme(plot_title = element_text(hjust = 0.5, face = 'bold'))+\
    labs(x = "Date", y = "Vehicles", title = "Junction 1 Training Data")

ts_bunch = GGBunch()
ts_bunch.add_plot(plt_ts, 0, 0, 850, 300)
ts_bunch
```

Out[14]:



In [16]:

```
# Pre-process data for the regression model
xtrain = df_train.get_column("Time_index").to_numpy()
xvalid = df_valid.get_column("Time_index").to_numpy()

xtrain = xtrain.reshape(-1,1)
xvalid = xvalid.reshape(-1,1)

ytrain = df_train.get_column("Vehicles").to_numpy()
yvalid = df_valid.get_column("Vehicles").to_numpy()

# Training
trend_model = LinearRegression().fit(xtrain, ytrain)

# Predicting
trend_preds_valid = trend_model.predict(xvalid)

# Getting the RMSE
print
print("Trend model validation set RMSE:", math.sqrt(mean_squared_error(yvalid, trend_pr
print
```

```
Trend model validation set RMSE: 27.24957995600769
```

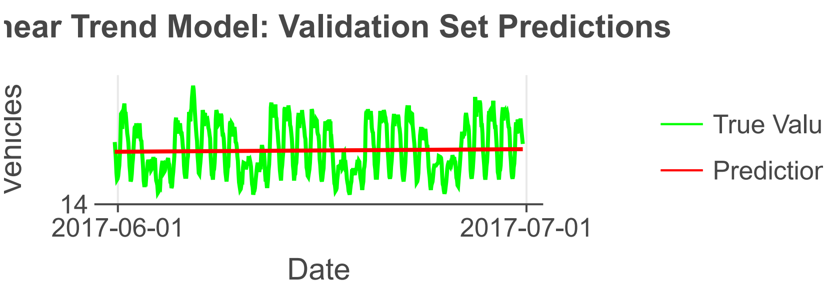
Out[16]:

```
<function print>
```

In [17]:

```
#Initializing a color
true_values_color = "Green"
trend_model_color = '#FF0000'

# Getting the validation data for the plot
df_labels = pl.DataFrame(
    {'DateTime': df_valid.get_column("DateTime"),
     'Vehicles': df_valid.get_column("Vehicles"),
     'Group': ["Label"]*len(df_valid)}
)

df_preds = pl.DataFrame(
    {'DateTime_preds': df_valid.get_column("DateTime"),
     'Vehicles_preds': trend_preds_valid,
     'Group_preds': ["Predictions"]*len(df_valid)}
)

df_trend_results_valid = (
    pl.concat([df_labels, df_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)

# Plotting the predictions on the validation set
plt_reg_valid = \
    ggplot(df_trend_results_valid)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = linesize, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = linesize, show_legend = True)+\
    scale_color_manual(values = [true_values_color, trend_model_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 145])+\
    theme_minimal2()+\
    theme(plot_title = element_text(hjust = 0.5, face = 'bold'),
         legend_title = element_blank())+\
    labs(x = "Date", y = "Vehicles", title = "Linear Trend Model: Validation Set Predic

reg_bunch = GGBunch()
reg_bunch.add_plot(plt_reg_valid, 0, 0, 850, 300)
reg_bunch
```

Out[17]:



In [18]:

```
# Features on the training set
df_train = df_train.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_train = df_train.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_train = df_train.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_train = df_train.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_train = df_train.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))

# Features on the validation set
df_valid = df_valid.with_columns(pl.col("DateTime").dt.year().alias("Year"))
df_valid = df_valid.with_columns(pl.col("DateTime").dt.month().alias("Month"))
df_valid = df_valid.with_columns(pl.col("DateTime").dt.day().alias("Day_month"))
df_valid = df_valid.with_columns(pl.col("DateTime").dt.weekday().alias("Day_week"))
df_valid = df_valid.with_columns(pl.col("DateTime").dt.hour().alias("Hour"))

print(df_train.head())
print(df_valid.head())
```

```
shape: (5, 9)
┌─────────────┬─────────────────────┬──────────┬──────────┬───┬───────┬──
───────────┬──────────┬───────┐
│ Time_index ┆ DateTime            ┆ Junction ┆ Vehicles ┆ … ┆ Month ┆ D
ay_month ┆ Day_week ┆ Hour │
│ ---        ┆ ---                 ┆ ---      ┆ ---      ┆   ┆ ---   ┆ -
--       ┆ ---      ┆ ---  │
│ u32        ┆ datetime[μs]        ┆ i64      ┆ i64      ┆   ┆ u32   ┆ u
32       ┆ u32      ┆ u32  │
╞═════════════╪═════════════════════╪══════════╪══════════╪═══╪═══════╪══
═══════════╪══════════╪═══════╡
│ 0          ┆ 2015-11-01 00:00:00 ┆ 1        ┆ 15       ┆ … ┆ 11    ┆ 1
┆ 7          ┆ 0    │
│ 1          ┆ 2015-11-01 01:00:00 ┆ 1        ┆ 13       ┆ … ┆ 11    ┆ 1
┆ 7          ┆ 1    │
│ 2          ┆ 2015-11-01 02:00:00 ┆ 1        ┆ 10       ┆ … ┆ 11    ┆ 1
┆ 7          ┆ 2    │
│ 3          ┆ 2015-11-01 03:00:00 ┆ 1        ┆ 7        ┆ … ┆ 11    ┆ 1
┆ 7          ┆ 3    │
│ 4          ┆ 2015-11-01 04:00:00 ┆ 1        ┆ 9        ┆ … ┆ 11    ┆ 1
┆ 7          ┆ 4    │
└─────────────┴─────────────────────┴──────────┴──────────┴───┴───────┴──
───────────┴──────────┴───────┘
```

```
shape: (5, 9)
┌─────────────┬─────────────────────┬──────────┬──────────┬───┬───────┬──
───────────┬──────────┬───────┐
│ Time_index ┆ DateTime            ┆ Junction ┆ Vehicles ┆ … ┆ Month ┆ D
ay_month ┆ Day_week ┆ Hour │
│ ---        ┆ ---                 ┆ ---      ┆ ---      ┆   ┆ ---   ┆ -
--       ┆ ---      ┆ ---  │
│ u32        ┆ datetime[μs]        ┆ i64      ┆ i64      ┆   ┆ u32   ┆ u
32       ┆ u32      ┆ u32  │
╞═════════════╪═════════════════════╪══════════╪══════════╪═══╪═══════╪══
═══════════╪══════════╪═══════╡
│ 13872      ┆ 2017-06-01 00:00:00 ┆ 1        ┆ 80       ┆ … ┆ 6     ┆ 1
┆ 4          ┆ 0    │
│ 13873      ┆ 2017-06-01 01:00:00 ┆ 1        ┆ 71       ┆ … ┆ 6     ┆ 1
┆ 4          ┆ 1    │
│ 13874      ┆ 2017-06-01 02:00:00 ┆ 1        ┆ 61       ┆ … ┆ 6     ┆ 1
┆ 4          ┆ 2    │
│ 13875      ┆ 2017-06-01 03:00:00 ┆ 1        ┆ 47       ┆ … ┆ 6     ┆ 1
┆ 4          ┆ 3    │
│ 13876      ┆ 2017-06-01 04:00:00 ┆ 1        ┆ 40       ┆ … ┆ 6     ┆ 1
┆ 4          ┆ 4    │
└─────────────┴─────────────────────┴──────────┴──────────┴───┴───────┴──
───────────┴──────────┴───────┘
```

In [20]:

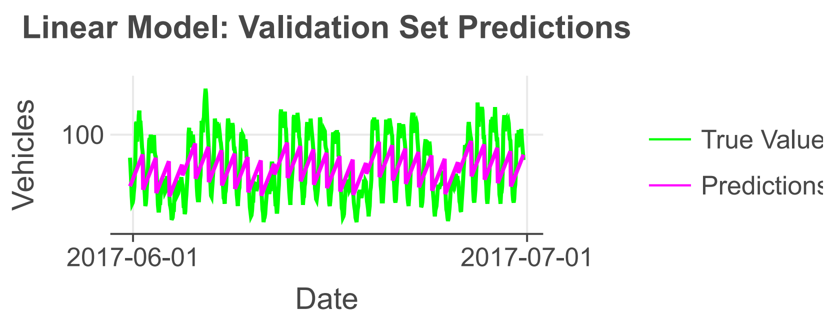
```
# Pre-process data for the regression model
xtrain = df_train.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()
xvalid = df_valid.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()

ytrain = df_train.get_column("Vehicles").to_numpy()
yvalid = df_valid.get_column("Vehicles").to_numpy()

# Training
reg_model = LinearRegression().fit(xtrain, ytrain)

# Predicting
reg_preds_valid = reg_model.predict(xvalid)

# Getting the RMSE
print("Linear model + more features validation set RMSE:",
      math.sqrt(mean_squared_error(yvalid, reg_preds_valid)))
```

```
Linear model + more features validation set RMSE: 21.36780895655701
```

In [21]:

```
# Initializing a color
reg_model_color = '#FF00FF'

# Getting the validation data for the plot
df_preds = pl.DataFrame(
    {'DateTime_preds': df_valid.get_column("DateTime"),
     'Vehicles_preds': reg_preds_valid,
     'Group_preds': ["Predictions"]*len(df_valid)}
)

df_reg_results_valid = (
    pl.concat([df_labels, df_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)
# Plotting the predictions on the validation set
plt_reg_valid = \
    ggplot(df_reg_results_valid)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = linesize, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = linesize, show_legend = True)+\
    scale_color_manual(values = [true_values_color, reg_model_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 145])+\
    theme_minimal2()+\
    theme(plot_title = element_text(hjust = 0.5, face = 'bold'),
         legend_title = element_blank())+\
    labs(x = "Date", y = "Vehicles", title = "Linear Model: Validation Set Predictions'

reg_bunch = GGBunch()
reg_bunch.add_plot(plt_reg_valid, 0, 0, 900, 350)
reg_bunch
```

Out[21]:



In [33]:

```
# Suppress optuna log messages
optuna.logging.set_verbosity(optuna.logging.WARNING)

# Optuna objective function
def objective_xgb(trial):
    """
    Optuna objective function. Returns
    the RMSE for an XGBoost model

    Assumes the training data are
    polars data frames
    """
    # Get data for the XGBoost model
    xtrain = df_train.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()
    xvalid = df_valid.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()

    ytrain = df_train.get_column("Vehicles").to_numpy()
    yvalid = df_valid.get_column("Vehicles").to_numpy()

    dmat_train = xgb.DMatrix(xtrain, label = ytrain)
    dmat_valid = xgb.DMatrix(xvalid, label = yvalid)

    # Suggest hyperparameters for XGBoost
    params = {'objective': 'reg:squarederror',
              'eval_metric': 'rmse',
              'seed': 19970507,
              'eta': trial.suggest_float("eta", 1e-2, 0.25, log = True),
              'max_depth': trial.suggest_int("max_depth", 1, 7),
              'lambda': trial.suggest_float("lambda", 1e-8, 100.0, log = True),
              'alpha': trial.suggest_float("alpha", 1e-8, 100.0, log = True),
             }

    # To evaluate training progress (set verbose_eval = True)
    watchlist = [(dmat_train, 'train'), (dmat_valid, 'eval')]

    # Train the XGBoost model
    xgb_model = xgb.train(params,
                          dtrain = dmat_train,
                          num_boost_round = trial.suggest_int("num_boost_round", 20, 3
                          evals = watchlist,
                          verbose_eval = False)

    xgb_preds_valid = xgb_model.predict(dmat_valid)

    # Return the RMSE
    return math.sqrt(mean_squared_error(yvalid, xgb_preds_valid))


# Set up and run the Optuna study
study_xgb = optuna.create_study(direction = 'minimize')
study_xgb.optimize(objective_xgb, n_trials = 10)

# Create a table showing the best parameters
xgb_table = [["Parameter", "Optimal Value from Optuna"],
            ["Iterations (num_boost_rounds)", study_xgb.best_params['num_boost_round']
            ['Learning Rate (eta)', round(study_xgb.best_params['eta'], 3)],
            ['Max Depth (max_depth)', round(study_xgb.best_params['max_depth'], 3)],
            ['Lambda (lambda)', round(study_xgb.best_params['lambda'], 3)],
            ['Alpha (alpha)', round(study_xgb.best_params['alpha'], 3)]]
```

```
ff.create_table(xgb_table)
```

Parameter	Optimal Value from O
Iterations (num_boost_rounds)	513
Learning Rate (eta)	0.034
Max Depth (max_depth)	5

In [34]:

```
# Taking the model with the best hyperparameters and testing it
xtrain = df_train.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()
xvalid = df_valid.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()

ytrain = df_train.get_column("Vehicles").to_numpy()
yvalid = df_valid.get_column("Vehicles").to_numpy()

dmat_train = xgb.DMatrix(xtrain, label = ytrain)
dmat_valid = xgb.DMatrix(xvalid, label = yvalid)

best_params = {'objective': 'reg:squarederror',
                'eval_metric': 'rmse',
                'seed': 19970507,
                'eta': study_xgb.best_params['eta'],
                'max_depth': study_xgb.best_params['max_depth'],
                'lambda': study_xgb.best_params['lambda'],
                'alpha': study_xgb.best_params['alpha'],
                 }

xgb_model = xgb.train(best_params,
                      dtrain = dmat_train,
                      num_boost_round = study_xgb.best_params['num_boost_round'],
                      verbose_eval = False)

xgb_preds_valid = xgb_model.predict(dmat_valid)

print('----------------------------------------------------------')
print('XGBoost validation set RMSE:', math.sqrt(mean_squared_error(yvalid, xgb_preds_va
print('----------------------------------------------------------')
```

```
----------------------------------------------------------
XGBoost validation set RMSE: 7.483966089515467
----------------------------------------------------------
```

In [35]:

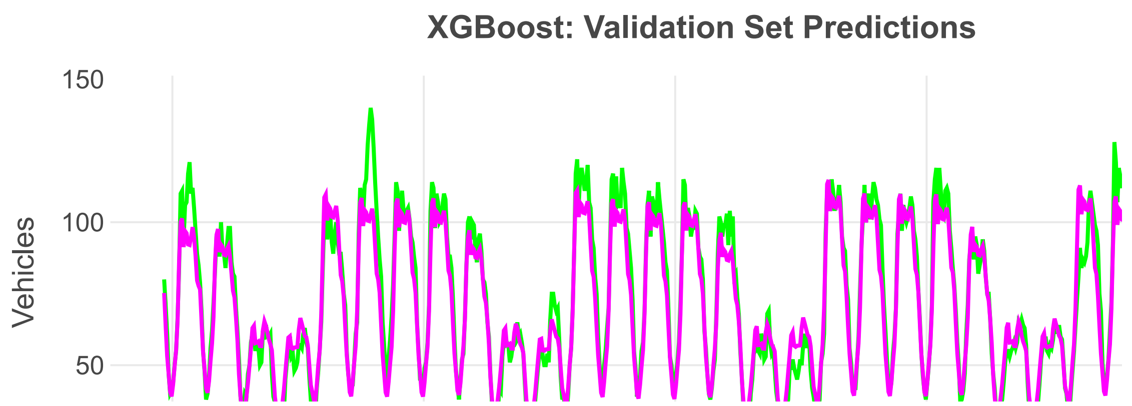
```
import sklearn.metrics as sm
print("Mean absolute error =", round(sm.mean_absolute_error(yvalid, xgb_preds_valid), 2
print("Mean squared error =", round(sm.mean_squared_error(yvalid, xgb_preds_valid), 2))
print("Median absolute error =", round(sm.median_absolute_error(yvalid, xgb_preds_valid
print("Explain variance score =", round(sm.explained_variance_score(yvalid, xgb_preds_v
print("R2 score =", round(sm.r2_score(yvalid, xgb_preds_valid), 2))
```

```
Mean absolute error = 5.43
Mean squared error = 56.01
Median absolute error = 3.91
Explain variance score = 0.93
R2 score = 0.92
```

In [36]:

```python
# Initializing a color
xgb_color = '#FF00FF'

# Getting the validation data for the plot
df_preds = pl.DataFrame(
    {'DateTime_preds': df_valid.get_column("DateTime"),
     'Vehicles_preds': xgb_preds_valid,
     'Group_preds': ["Predictions"]*len(df_valid)}
)

df_xgb_results_valid = (
    pl.concat([df_labels, df_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)

# Plotting the predictions on the validation set
plt_xgb_valid = \
    ggplot(df_xgb_results_valid)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = linesize, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = linesize, show_legend = True)+\
    scale_color_manual(values = [true_values_color, xgb_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 145])+\
    theme_minimal2()+\
    theme(plot_title = element_text(hjust = 0.5, face = 'bold'),
          legend_title = element_blank())+\
    labs(x = "Date", y = "Vehicles", title = "XGBoost: Validation Set Predictions")

xgb_bunch = GGBunch()
xgb_bunch.add_plot(plt_xgb_valid, 0, 0, 850, 300)
xgb_bunch
```

Out[36]:



In [37]:

```
# Suppress optuna log messages
optuna.logging.set_verbosity(optuna.logging.WARNING)

# Optuna objective function
def objective_trend_xgb(trial):

    # Get data for the trend model
    xtrain_reg = df_train.get_column("Time_index").to_numpy()
    xvalid_reg = df_valid.get_column("Time_index").to_numpy()

    xtrain_reg = xtrain_reg.reshape(-1,1)
    xvalid_reg = xvalid_reg.reshape(-1,1)

    ytrain = df_train.get_column("Vehicles").to_numpy()
    yvalid = df_valid.get_column("Vehicles").to_numpy()

    # Train and predict w/ the trend model
    reg_model = LinearRegression().fit(xtrain_reg, ytrain)

    # Predicting
    reg_preds_train = reg_model.predict(xtrain_reg)
    reg_preds_valid = reg_model.predict(xvalid_reg)

    # Calculate the residuals
    reg_resids_train = (ytrain - reg_preds_train)
    reg_resids_valid = (yvalid - reg_preds_valid)

    # Get the data for the XGB model
    xtrain_xgb = df_train.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()
    xvalid_xgb = df_valid.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()

    dmat_train = xgb.DMatrix(xtrain_xgb, label = reg_resids_train)
    dmat_valid = xgb.DMatrix(xvalid_xgb, label = reg_resids_valid)

    # Suggest hyperparameters
    params = {'objective': 'reg:squarederror',
              'eval_metric': 'rmse',
              'seed': 19970507,
              'eta': trial.suggest_float("eta", 1e-2, 0.25, log = True),
              'max_depth': trial.suggest_int("max_depth", 1, 7),
              'lambda': trial.suggest_float("lambda", 1e-8, 100.0, log = True),
              'alpha': trial.suggest_float("alpha", 1e-8, 100.0, log = True),
             }

    # To evaluate training progress (set verbose_eval = True)
    watchlist = [(dmat_train, 'train'), (dmat_valid, 'eval')]

    # Train and predict w/ the XGBoost model
    xgb_model = xgb.train(params,
                          dtrain = dmat_train,
                          num_boost_round = trial.suggest_int("num_boost_round", 20, 3(
                          evals = watchlist,
                          verbose_eval = False)

    xgb_preds_valid = xgb_model.predict(dmat_valid)

    # Sum the final predictions
    trend_xgb_preds_valid = (reg_preds_valid + xgb_preds_valid)
```

```python
    # Return the RMSE
    return math.sqrt(mean_squared_error(yvalid,  trend_xgb_preds_valid))


# Set up and run the Optuna study
study_trend_xgb = optuna.create_study(direction = 'minimize')
study_trend_xgb.optimize(objective_trend_xgb, n_trials = 10)

# Create a table showing the best parameters
trend_xgb_table = [["Parameter", "Optimal Value from Optuna"],
                  ["Iterations (num_boost_rounds)", study_trend_xgb.best_params['num_bo
                  ['Learning Rate (eta)', round(study_trend_xgb.best_params['eta'], 3)]
                  ['Max Depth (max_depth)', round(study_trend_xgb.best_params['max_dept
                  ['Lambda', round(study_trend_xgb.best_params['lambda'], 3)],
                  ['Alpha', round(study_trend_xgb.best_params['alpha'], 3)]]

ff.create_table(trend_xgb_table)
```

Parameter	Optimal Value from O
Iterations (num_boost_rounds)	687
Learning Rate (eta)	0.013
Max Depth (max_depth)	6

In [38]:

```
# Taking the model with the best hyperparameters and testing it

# Get data for the trend model
xtrain_reg = df_train.get_column("Time_index").to_numpy()
xvalid_reg = df_valid.get_column("Time_index").to_numpy()

xtrain_reg = xtrain_reg.reshape(-1,1)
xvalid_reg = xvalid_reg.reshape(-1,1)

ytrain = df_train.get_column("Vehicles").to_numpy()
yvalid = df_valid.get_column("Vehicles").to_numpy()

# Train and predict w/ the trend model
reg_model = LinearRegression().fit(xtrain_reg, ytrain)

# Predicting
reg_preds_train = reg_model.predict(xtrain_reg)
reg_preds_valid = reg_model.predict(xvalid_reg)

# Calculate the residuals
reg_resids_train = (ytrain - reg_preds_train)
reg_resids_valid = (yvalid - reg_preds_valid)

# Get the data for the XGB model
xtrain_xgb = df_train.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()
xvalid_xgb = df_valid.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()

dmat_train = xgb.DMatrix(xtrain_xgb, label = reg_resids_train)
dmat_valid = xgb.DMatrix(xvalid_xgb, label = reg_resids_valid)

best_params = {'objective': 'reg:squarederror',
                'eval_metric': 'rmse',
                'seed': 19970507,
                'eta': study_trend_xgb.best_params['eta'],
                'max_depth': study_trend_xgb.best_params['max_depth'],
                'lambda': study_trend_xgb.best_params['lambda'],
                'alpha': study_trend_xgb.best_params['alpha'],
                 }

xgb_model = xgb.train(best_params,
                      dtrain = dmat_train,
                      num_boost_round = study_trend_xgb.best_params['num_boost_round'],
                      verbose_eval = False)

xgb_preds_valid = xgb_model.predict(dmat_valid)

# Sum the final predictions
trend_xgb_preds_valid = (reg_preds_valid + xgb_preds_valid)

print('-----------------------------------------------------------')
print('Trend model + XGBoost validation set RMSE:', math.sqrt(mean_squared_error(yvalid
print('-----------------------------------------------------------')
```

```
----------------------------------------------------------
Trend model + XGBoost validation set RMSE: 6.958519492196214
----------------------------------------------------------
```

In [39]:

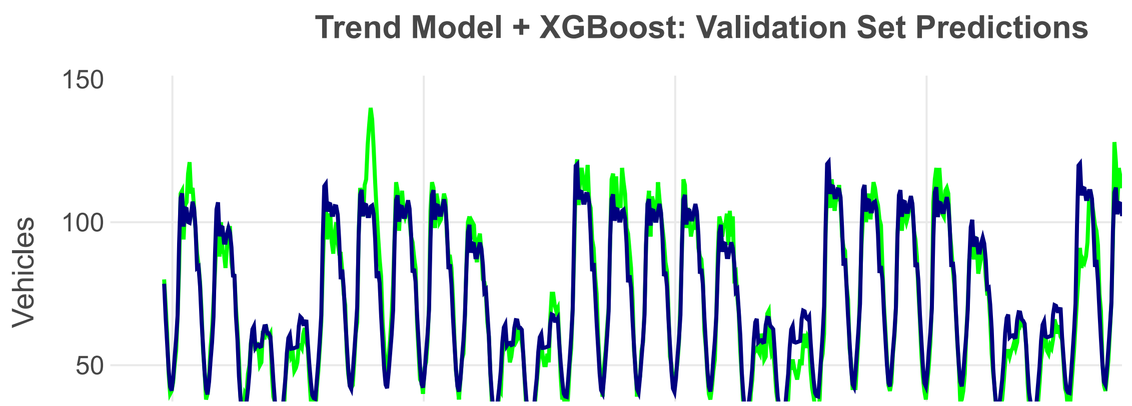
```
import sklearn.metrics as sm
print("Mean absolute error =", round(sm.mean_absolute_error(yvalid, trend_xgb_preds_val
print("Mean squared error =", round(sm.mean_squared_error(yvalid, trend_xgb_preds_valic
print("Median absolute error =", round(sm.median_absolute_error(yvalid, trend_xgb_preds
print("Explain variance score =", round(sm.explained_variance_score(yvalid, trend_xgb_p
print("R2 score =", round(sm.r2_score(yvalid, trend_xgb_preds_valid), 2))
```

```
Mean absolute error = 5.01
Mean squared error = 48.42
Median absolute error = 3.99
Explain variance score = 0.93
R2 score = 0.93
```

In [40]:

```
# Initializing a color
trend_xgb_color = '#000080'

# Getting the validation data for the plot
df_preds = pl.DataFrame(
    {'DateTime_preds': df_valid.get_column("DateTime"),
     'Vehicles_preds': trend_xgb_preds_valid,
     'Group_preds': ["Predictions"]*len(df_valid)}
)

df_trend_xgb_results_valid = (
    pl.concat([df_labels, df_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)

# Plotting the predictions on the validation set
plt_trend_xgb_valid = \
    ggplot(df_trend_xgb_results_valid)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = linesize, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = linesize, show_legend = True)+\
    scale_color_manual(values = [true_values_color, trend_xgb_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 145])+\
    theme_minimal2()+\
    theme(plot_title = element_text(hjust = 0.5, face = 'bold'),
          legend_title = element_blank())+\
    labs(x = "Date", y = "Vehicles", title = "Trend Model + XGBoost: Validation Set Pre

trend_xgb_bunch = GGBunch()
trend_xgb_bunch.add_plot(plt_trend_xgb_valid, 0, 0, 850, 300)
trend_xgb_bunch
```

Out[40]:



In [41]:

```
# Lagged Features
print(pl.concat([df_train.select(pl.col("Vehicles")),
                 df_train.select([pl.col("Vehicles").shift(1).alias("Lag 1")]),
                 df_train.select([pl.col("Vehicles").shift(2).alias("Lag 2")]),
                 df_train.select([pl.col("Vehicles").shift(3).alias("Lag 3")]),
                 df_train.select([pl.col("Vehicles").shift(4).alias("Lag 4")])], how =

# Adding two lagged features to the model
one_month = 24*30 # To match validation set length month = 30 days
one_year = 24*364

df_train = df_train.with_columns([
    (pl.col("Vehicles").shift(one_month).alias("Lag_month")),
    (pl.col("Vehicles").shift(one_year).alias("Lag_year"))
])
```

shape: (5, 5)

Vehicles --- i64	Lag 1 --- i64	Lag 2 --- i64	Lag 3 --- i64	Lag 4 --- i64
15	null	null	null	null
13	15	null	null	null
10	13	15	null	null
7	10	13	15	null
9	7	10	13	15

In [42]:

```
# Now the validaiton set needs lags
# Note the validation set length is one month
df_valid = df_valid.with_columns([
    (df_train.get_column("Vehicles")[-one_month:].alias("Lag_month")),
    (df_train.get_column("Vehicles")[-one_year:(-one_year + one_month)].alias("Lag_year
])

# Set up and run the Optuna study
study_xgb_lag = optuna.create_study(direction = 'minimize')
study_xgb_lag.optimize(objective_xgb, n_trials = 10)

# Create a table showing the best parameters
xgb_lag_table = [["Parameter", "Optimal Value from Optuna"],
                 ["Iterations (num_boost_rounds)", study_xgb_lag.best_params['num_boost
                 ['Learning Rate (eta)', round(study_xgb_lag.best_params['eta'], 3)],
                 ['Max Depth (max_depth)', round(study_xgb_lag.best_params['max_depth']
                 ['Lambda (lambda)', round(study_xgb_lag.best_params['lambda'], 3)],
                 ['Alpha (alpha)', round(study_xgb_lag.best_params['alpha'], 3)]]

ff.create_table(xgb_lag_table)
```

Parameter	Optimal Value from O
Iterations (num_boost_rounds)	1858
Learning Rate (eta)	0.121
Max Depth (max_depth)	3

In [43]:

```
# Taking the model with the best hyperparameters and testing it
xtrain = df_train.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()
xvalid = df_valid.drop(["DateTime", "Junction", "Vehicles"]).to_numpy()

ytrain = df_train.get_column("Vehicles").to_numpy()
yvalid = df_valid.get_column("Vehicles").to_numpy()

dmat_train = xgb.DMatrix(xtrain, label = ytrain)
dmat_valid = xgb.DMatrix(xvalid, label = yvalid)

best_params = {'objective': 'reg:squarederror',
                'eval_metric': 'rmse',
                'seed': 19970507,
                'eta': study_xgb_lag.best_params['eta'],
                'max_depth': study_xgb_lag.best_params['max_depth'],
                'lambda': study_xgb_lag.best_params['lambda'],
                'alpha': study_xgb_lag.best_params['alpha'],
                 }

xgb_lag_model = xgb.train(best_params,
                          dtrain = dmat_train,
                          num_boost_round = study_xgb_lag.best_params['num_boost_round']
                          verbose_eval = False)

xgb_lag_preds_valid = xgb_lag_model.predict(dmat_valid)

print('----------------------------------------------------------')
print('XGBoost (w/ Lagging) validation set RMSE:', math.sqrt(mean_squared_error(yvalid,
print('----------------------------------------------------------')
```

```
----------------------------------------------------------
XGBoost (w/ Lagging) validation set RMSE: 7.160714638937514
----------------------------------------------------------
```

In [44]:

```
# Initializing a color
xgb_lag_color = '#00FF00'

# Getting the validation data for the plot
df_preds = pl.DataFrame(
    {'DateTime_preds': df_valid.get_column("DateTime"),
     'Vehicles_preds': xgb_lag_preds_valid,
     'Group_preds': ["Predictions"]*len(df_valid)}
)

df_xgb_lag_results_valid = (
    pl.concat([df_labels, df_preds], how = 'horizontal')
    .with_columns(
        (pl.lit("True Values").alias("Group_label")),
        (pl.lit("Predictions").alias("Group_pred")))
)

# Plotting the predictions on the validation set
plt_xgb_lag_valid = \
    ggplot(df_xgb_lag_results_valid)+\
    geom_line(aes(x = "DateTime", y = "Vehicles", color = "Group_label"),
              sampling = "none", size = linesize, show_legend = True)+\
    geom_line(aes(x = "DateTime", y = "Vehicles_preds", color = "Group_pred"),
              sampling = "none", size = linesize, show_legend = True)+\
    scale_color_manual(values = [true_values_color, xgb_color])+\
    scale_x_datetime(format = "%Y-%m-%d")+\
    scale_y_continuous(limits = [20, 145])+\
    theme_minimal2()+\
    theme(plot_title = element_text(hjust = 0.5, face = 'bold'),
          legend_title = element_blank())+\
    labs(x = "Date", y = "Vehicles", title = "XGBoost (w/ Lagging): Validation Set Prec

xgb_lag_bunch = GGBunch()
xgb_lag_bunch.add_plot(plt_xgb_lag_valid, 0, 0, 850, 300)
xgb_lag_bunch
```

Out[44]: