# Data Ingest

import pandas as pd  
  
FILE\_LOCATION = '/content/daily-website-visitors.csv'  
  
whole\_dataset = pd.read\_csv(FILE\_LOCATION,  
 index\_col='Date',  
 thousands=',')  
whole\_dataset.index = pd.to\_datetime(whole\_dataset.index)  
whole\_dataset

Row Day Day.Of.Week Page.Loads Unique.Visits \  
Date   
2014-09-14 1 Sunday 1 2146 1582   
2014-09-15 2 Monday 2 3621 2528   
2014-09-16 3 Tuesday 3 3698 2630   
2014-09-17 4 Wednesday 4 3667 2614   
2014-09-18 5 Thursday 5 3316 2366   
... ... ... ... ... ...   
2020-08-15 2163 Saturday 7 2221 1696   
2020-08-16 2164 Sunday 1 2724 2037   
2020-08-17 2165 Monday 2 3456 2638   
2020-08-18 2166 Tuesday 3 3581 2683   
2020-08-19 2167 Wednesday 4 2064 1564   
  
 First.Time.Visits Returning.Visits   
Date   
2014-09-14 1430 152   
2014-09-15 2297 231   
2014-09-16 2352 278   
2014-09-17 2327 287   
2014-09-18 2130 236   
... ... ...   
2020-08-15 1373 323   
2020-08-16 1686 351   
2020-08-17 2181 457   
2020-08-18 2184 499   
2020-08-19 1297 267   
  
[2167 rows x 7 columns]

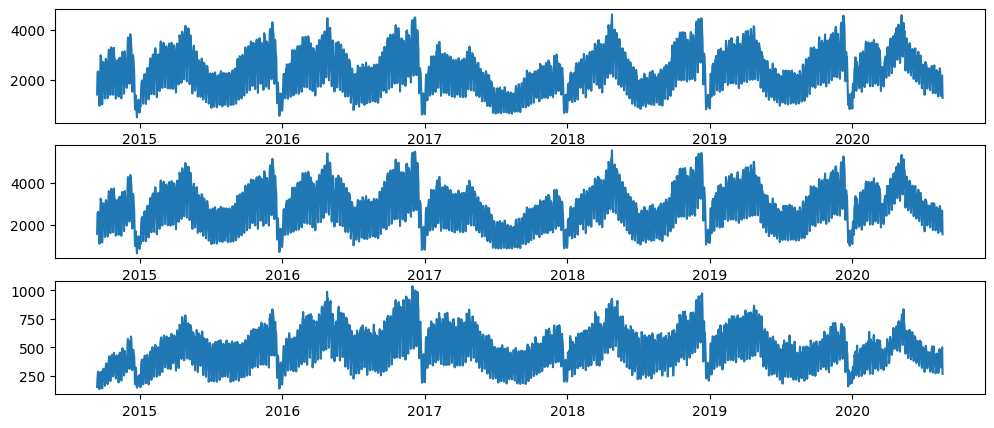
whole\_dataset.info()

<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 2167 entries, 2014-09-14 to 2020-08-19  
Data columns (total 7 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Row 2167 non-null int64   
 1 Day 2167 non-null object  
 2 Day.Of.Week 2167 non-null int64   
 3 Page.Loads 2167 non-null int64   
 4 Unique.Visits 2167 non-null int64   
 5 First.Time.Visits 2167 non-null int64   
 6 Returning.Visits 2167 non-null int64   
dtypes: int64(6), object(1)  
memory usage: 135.4+ KB

whole\_dataset.describe()

Row Day.Of.Week Page.Loads Unique.Visits \  
count 2167.000000 2167.000000 2167.000000 2167.000000   
mean 1084.000000 3.997231 4116.989386 2943.646516   
std 625.703338 2.000229 1350.977843 977.886472   
min 1.000000 1.000000 1002.000000 667.000000   
25% 542.500000 2.000000 3114.500000 2226.000000   
50% 1084.000000 4.000000 4106.000000 2914.000000   
75% 1625.500000 6.000000 5020.500000 3667.500000   
max 2167.000000 7.000000 7984.000000 5541.000000   
  
 First.Time.Visits Returning.Visits   
count 2167.000000 2167.000000   
mean 2431.824181 511.822335   
std 828.704688 168.736370   
min 522.000000 133.000000   
25% 1830.000000 388.500000   
50% 2400.000000 509.000000   
75% 3038.000000 626.500000   
max 4616.000000 1036.000000

import matplotlib.pyplot as plt  
  
fig, axs = plt.subplots(3, figsize=(12, 5))  
  
axs[0].plot(whole\_dataset['First.Time.Visits'])  
axs[1].plot(whole\_dataset['Unique.Visits'])  
axs[2].plot(whole\_dataset['Returning.Visits'])  
plt.show()



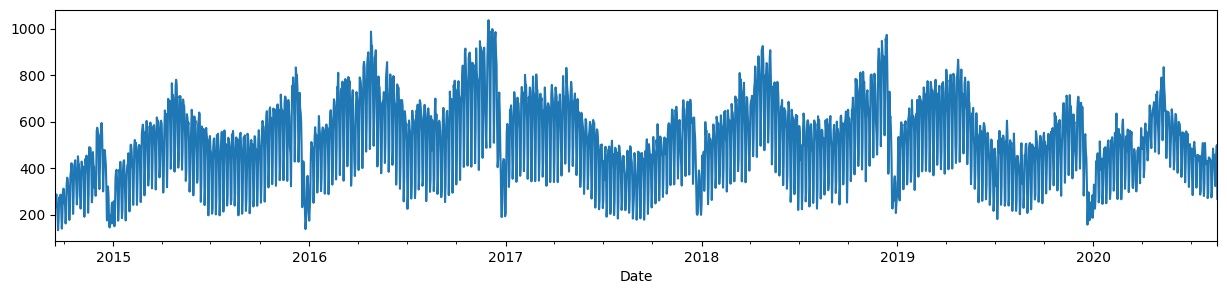
# Preprocessing the data

* Target Attribute: **Returning.Visits** We shall predict the **Returning.Visits** given past data.

target\_column = whole\_dataset['Returning.Visits']  
target\_column

Date  
2014-09-14 152  
2014-09-15 231  
2014-09-16 278  
2014-09-17 287  
2014-09-18 236  
 ...   
2020-08-15 323  
2020-08-16 351  
2020-08-17 457  
2020-08-18 499  
2020-08-19 267  
Name: Returning.Visits, Length: 2167, dtype: int64

target\_column.plot(figsize=(15, 3))  
plt.show()



# Compute Train and Test Data Boundaries

len(target\_column)

2167

TEST\_DATA\_PERCENTAGE = 0.1  
  
TEST\_DATA\_BOUNDARY\_INDEX = int((1 - TEST\_DATA\_PERCENTAGE) \* len(target\_column))  
print(f"Train data:\tReturning Visits [:{TEST\_DATA\_BOUNDARY\_INDEX}] ({TEST\_DATA\_BOUNDARY\_INDEX + 1})")  
print(f"Test data:\tReturning Visits [{TEST\_DATA\_BOUNDARY\_INDEX}:] ({len(target\_column) - TEST\_DATA\_BOUNDARY\_INDEX})")  
print(f"\nLast target on train data: {target\_column[TEST\_DATA\_BOUNDARY\_INDEX]}")

Train data: Returning Visits [:1950] (1951)  
Test data: Returning Visits [1950:] (217)  
  
Last target on train data: 441

print(f"Train dataset ending values: {target\_column[TEST\_DATA\_BOUNDARY\_INDEX - 10: TEST\_DATA\_BOUNDARY\_INDEX].values}")  
print(f"Test dataset starting values: {target\_column[TEST\_DATA\_BOUNDARY\_INDEX: TEST\_DATA\_BOUNDARY\_INDEX + 10].values}")

Train dataset ending values: [429 423 442 464 372 253 277 515 434 394]  
Test dataset starting values: [441 413 246 314 443 484 473 490 353 249]

## Window-ize the dataset

from tensorflow.keras.utils import timeseries\_dataset\_from\_array  
  
WINDOW\_SIZE = 3  
train\_dataset = timeseries\_dataset\_from\_array(target\_column[:-WINDOW\_SIZE],  
 target\_column[WINDOW\_SIZE:],  
 sequence\_length=WINDOW\_SIZE,  
 end\_index=TEST\_DATA\_BOUNDARY\_INDEX - 1)  
len(train\_dataset), len(list(train\_dataset.unbatch()))

(16, 1947)

target\_column[TEST\_DATA\_BOUNDARY\_INDEX-10:TEST\_DATA\_BOUNDARY\_INDEX+10].values, (list(train\_dataset)[-1][0][-1].numpy(), list(train\_dataset)[-1][1][-1].numpy())

(array([429, 423, 442, 464, 372, 253, 277, 515, 434, 394, 441, 413, 246,  
 314, 443, 484, 473, 490, 353, 249]),  
 (array([277, 515, 434]), 394))

test\_dataset = timeseries\_dataset\_from\_array(target\_column[TEST\_DATA\_BOUNDARY\_INDEX - WINDOW\_SIZE:],  
 target\_column[TEST\_DATA\_BOUNDARY\_INDEX:],  
 sequence\_length=WINDOW\_SIZE  
 )  
len(test\_dataset), len(list(test\_dataset.unbatch()))

(2, 217)

target\_column[TEST\_DATA\_BOUNDARY\_INDEX-10:TEST\_DATA\_BOUNDARY\_INDEX+10].values, list(test\_dataset)[0][0][0].numpy(), list(test\_dataset)[0][1][0].numpy()

(array([429, 423, 442, 464, 372, 253, 277, 515, 434, 394, 441, 413, 246,  
 314, 443, 484, 473, 490, 353, 249]),  
 array([515, 434, 394]),  
 441)

# First point in test dataset  
list(test\_dataset)[0][0][0].numpy(), list(test\_dataset)[0][1][0].numpy()

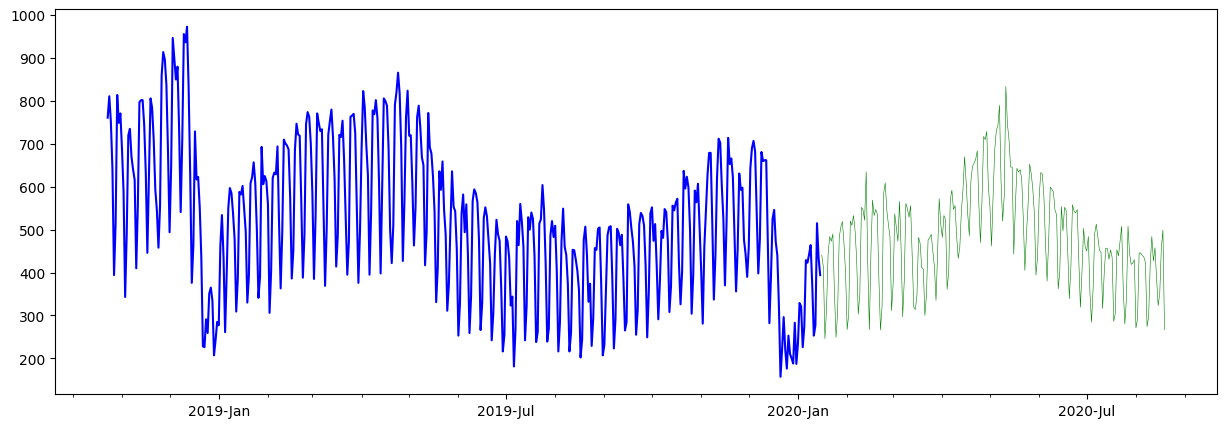
(array([515, 434, 394]), 441)

# Last point in test dataset  
list(test\_dataset)[-1][0][-1].numpy(), list(test\_dataset)[-1][1][-1].numpy()

(array([351, 457, 499]), 267)

## Plot the train and test datasets

import numpy as np  
import matplotlib.dates as mdates  
  
def plot\_time\_series(predictions = None, start\_index=1500):  
 timesteps = pd.to\_datetime(target\_column.index)  
  
 fig,ax = plt.subplots(1,figsize=(15,5))  
 ax.xaxis.set\_major\_locator(mdates.MonthLocator(bymonth=(1, 7)))  
 ax.xaxis.set\_minor\_locator(mdates.MonthLocator())  
 ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%b'))  
  
 # Plot train dataset  
 plt.plot(timesteps[start\_index:TEST\_DATA\_BOUNDARY\_INDEX], target\_column[start\_index:TEST\_DATA\_BOUNDARY\_INDEX],  
 color='blue')  
 # Plot test dataset  
 plt.plot(timesteps[TEST\_DATA\_BOUNDARY\_INDEX:], target\_column[TEST\_DATA\_BOUNDARY\_INDEX:],  
 color='green', linewidth=0.4)  
  
 if predictions is not None:  
 pred\_timesteps = timesteps[TEST\_DATA\_BOUNDARY\_INDEX:]  
 plt.plot(pred\_timesteps, predictions, linewidth=0.4, color='red')  
 plt.scatter(pred\_timesteps, predictions, s=0.4, color='red')  
  
  
plot\_time\_series()



# Model 0: Baseline model

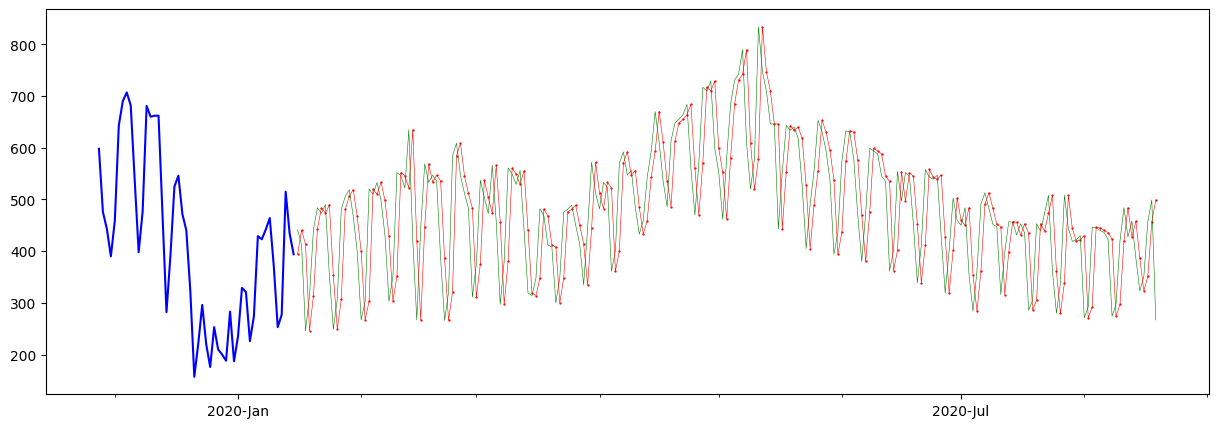
import tensorflow as tf  
from tensorflow.keras.layers import Layer  
from tensorflow.keras import Model  
  
class NaiveForecastLayer(Model):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 def call(self, inputs):  
 result = inputs[:, -1]  
 return result[:, tf.newaxis]

baseline\_model = NaiveForecastLayer()  
baseline\_model.\_name = 'model\_0'  
  
baseline\_model.compile(metrics=[tf.keras.metrics.MeanAbsoluteError()])

baseline\_predictions = baseline\_model.predict(test\_dataset)

2/2 [==============================] - 0s 16ms/step

plot\_time\_series(baseline\_predictions.ravel(), start\_index=1900)



y\_true = target\_column[TEST\_DATA\_BOUNDARY\_INDEX : ]  
  
len(y\_true), y\_true

(217,  
 Date  
 2020-01-16 441  
 2020-01-17 413  
 2020-01-18 246  
 2020-01-19 314  
 2020-01-20 443  
 ...   
 2020-08-15 323  
 2020-08-16 351  
 2020-08-17 457  
 2020-08-18 499  
 2020-08-19 267  
 Name: Returning.Visits, Length: 217, dtype: int64)

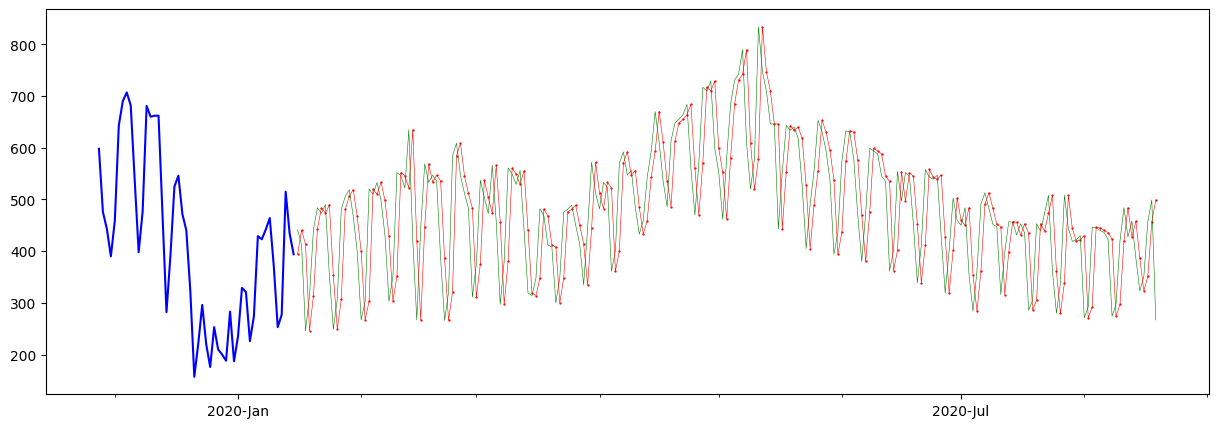
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_absolute\_percentage\_error  
  
def evaluate\_predictions(y\_true, y\_preds):  
 mae = mean\_absolute\_error(y\_true, y\_preds)  
 mse = mean\_squared\_error(y\_true, y\_preds)  
 rmse = np.sqrt(mse)  
 mape = mean\_absolute\_percentage\_error(y\_true, y\_preds)  
  
 return {  
 'mae': mae,  
 'mse': mse,  
 "rmse": rmse,  
 "mape": mape  
 }  
  
evaluate\_predictions(y\_true, baseline\_predictions)

{'mae': 72.19815668202764,  
 'mse': 8508.622119815669,  
 'rmse': 92.24219273096054,  
 'mape': 0.16713927858326993}

MODEL\_METRICS = pd.DataFrame(columns=['mae', 'mse', 'rmse', 'mape'])  
  
def evaluate\_model(model):  
 predictions = model.predict(test\_dataset, verbose=0)  
 metrics = evaluate\_predictions(y\_true, predictions)  
  
 MODEL\_METRICS.loc[model.name] = metrics  
 plot\_time\_series(predictions.ravel(), start\_index=1900)  
 return metrics

evaluate\_model(baseline\_model)

{'mae': 72.19815668202764,  
 'mse': 8508.622119815669,  
 'rmse': 92.24219273096054,  
 'mape': 0.16713927858326993}



MODEL\_METRICS

mae mse rmse mape  
model\_0 72.198157 8508.62212 92.242193 0.167139

# Model 1: Recurrent Network Model (GRU)

from tensorflow.keras.layers import GRU, Dense, Input, Lambda  
from tensorflow.keras import Sequential  
  
tf.random.set\_seed(42)  
model\_1 = Sequential([  
 Input(shape=(WINDOW\_SIZE,)),  
 Lambda(lambda x: tf.expand\_dims(x, axis=1)),  
 GRU(128, activation="relu"),  
 Dense(1)  
], name='model\_1')  
  
model\_1.compile(  
 loss=tf.keras.losses.MeanAbsoluteError(),  
 optimizer=tf.keras.optimizers.Adam()  
)  
  
model\_1.summary()

Model: "model\_1"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 lambda (Lambda) (None, 1, 3) 0   
   
 gru (GRU) (None, 128) 51072   
   
 dense (Dense) (None, 1) 129   
   
=================================================================  
Total params: 51201 (200.00 KB)  
Trainable params: 51201 (200.00 KB)  
Non-trainable params: 0 (0.00 Byte)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

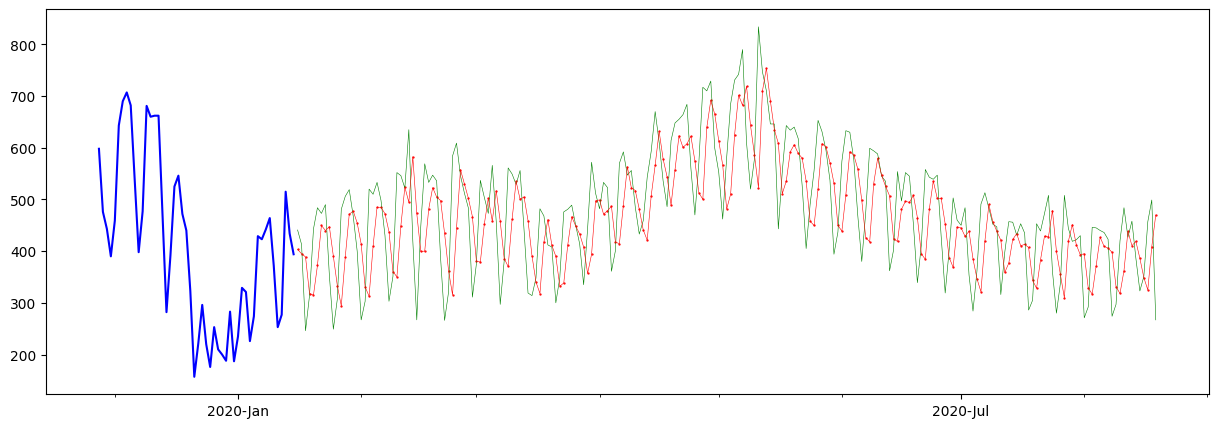
from tensorflow.keras.callbacks import ModelCheckpoint  
import os  
  
def create\_checkpoint\_callback(model):  
 filepath = os.path.join('models', model.name)  
 return ModelCheckpoint(filepath, monitor='loss', save\_weights\_only=True, save\_best\_only=True)  
  
model\_1.fit(train\_dataset, epochs=5, callbacks=[ create\_checkpoint\_callback(model\_1) ])

Epoch 1/5  
16/16 [==============================] - 4s 64ms/step - loss: 432.0301  
Epoch 2/5  
16/16 [==============================] - 1s 31ms/step - loss: 233.0275  
Epoch 3/5  
16/16 [==============================] - 0s 30ms/step - loss: 134.8151  
Epoch 4/5  
16/16 [==============================] - 1s 31ms/step - loss: 121.6276  
Epoch 5/5  
16/16 [==============================] - 0s 19ms/step - loss: 111.4610

<keras.src.callbacks.History at 0x7c8ef9e12650>

evaluate\_model(model\_1)

{'mae': 74.44923084452405,  
 'mse': 9023.324910312058,  
 'rmse': 94.99118332935988,  
 'mape': 0.1666447405608733}



MODEL\_METRICS

mae mse rmse mape  
model\_0 72.198157 8508.62212 92.242193 0.167139  
model\_1 74.449231 9023.32491 94.991183 0.166645

# Model 3: Multi-input Model

unbatched\_train\_dataset = whole\_dataset[:TEST\_DATA\_BOUNDARY\_INDEX + 1].copy()  
unbatched\_train\_dataset

Row Day Day.Of.Week Page.Loads Unique.Visits \  
Date   
2014-09-14 1 Sunday 1 2146 1582   
2014-09-15 2 Monday 2 3621 2528   
2014-09-16 3 Tuesday 3 3698 2630   
2014-09-17 4 Wednesday 4 3667 2614   
2014-09-18 5 Thursday 5 3316 2366   
... ... ... ... ... ...   
2020-01-12 1947 Sunday 1 2762 2238   
2020-01-13 1948 Monday 2 4298 3242   
2020-01-14 1949 Tuesday 3 3838 2884   
2020-01-15 1950 Wednesday 4 3754 2864   
2020-01-16 1951 Thursday 5 3817 2951   
  
 First.Time.Visits Returning.Visits   
Date   
2014-09-14 1430 152   
2014-09-15 2297 231   
2014-09-16 2352 278   
2014-09-17 2327 287   
2014-09-18 2130 236   
... ... ...   
2020-01-12 1961 277   
2020-01-13 2727 515   
2020-01-14 2450 434   
2020-01-15 2470 394   
2020-01-16 2510 441   
  
[1951 rows x 7 columns]

DA: Any significant different per day, per month, per year?

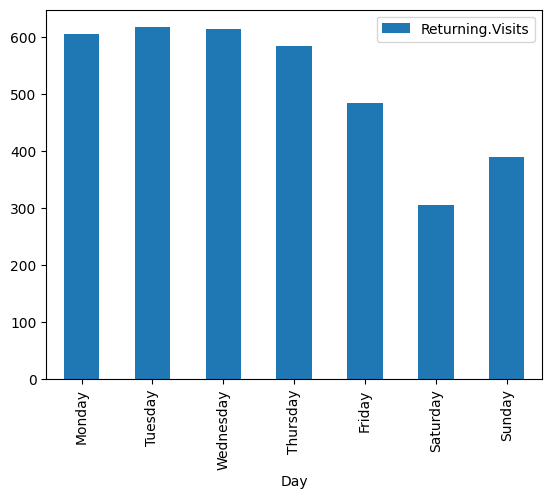
## Per Day of Week grouping

dataset\_by\_day = unbatched\_train\_dataset.groupby(by=['Day'])  
dataset\_by\_day['Returning.Visits'].mean()

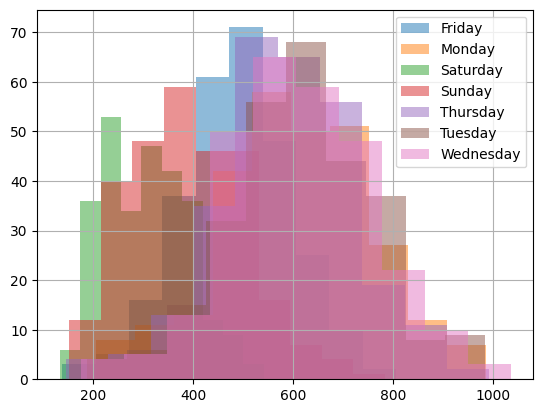
Day  
Friday 484.697842  
Monday 606.512545  
Saturday 306.071942  
Sunday 390.573477  
Thursday 584.627240  
Tuesday 617.888889  
Wednesday 614.369176  
Name: Returning.Visits, dtype: float64

DAYS\_OF\_WEEK = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']  
pd.DataFrame(dataset\_by\_day['Returning.Visits'].mean()).loc[DAYS\_OF\_WEEK].plot(kind='bar')

<Axes: xlabel='Day'>



dataset\_by\_day['Returning.Visits'].hist(legend=True, alpha=0.5)  
plt.show()



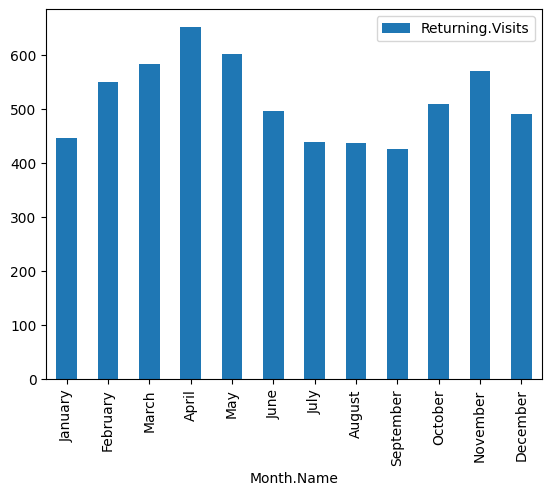
import calendar  
  
train\_dataset\_with\_months = unbatched\_train\_dataset.copy()  
train\_dataset\_with\_months['Month.Name'] = pd.Series(train\_dataset\_with\_months.index,  
 index=train\_dataset\_with\_months.index)\  
 .apply(lambda x: calendar.month\_name[x.month])  
train\_dataset\_with\_months

Row Day Day.Of.Week Page.Loads Unique.Visits \  
Date   
2014-09-14 1 Sunday 1 2146 1582   
2014-09-15 2 Monday 2 3621 2528   
2014-09-16 3 Tuesday 3 3698 2630   
2014-09-17 4 Wednesday 4 3667 2614   
2014-09-18 5 Thursday 5 3316 2366   
... ... ... ... ... ...   
2020-01-12 1947 Sunday 1 2762 2238   
2020-01-13 1948 Monday 2 4298 3242   
2020-01-14 1949 Tuesday 3 3838 2884   
2020-01-15 1950 Wednesday 4 3754 2864   
2020-01-16 1951 Thursday 5 3817 2951   
  
 First.Time.Visits Returning.Visits Month.Name   
Date   
2014-09-14 1430 152 September   
2014-09-15 2297 231 September   
2014-09-16 2352 278 September   
2014-09-17 2327 287 September   
2014-09-18 2130 236 September   
... ... ... ...   
2020-01-12 1961 277 January   
2020-01-13 2727 515 January   
2020-01-14 2450 434 January   
2020-01-15 2470 394 January   
2020-01-16 2510 441 January   
  
[1951 rows x 8 columns]

MONTH\_NAMES = list(calendar.month\_name)[1:]  
dataset\_group\_by\_month = train\_dataset\_with\_months.groupby(by='Month.Name')  
dataset\_group\_by\_month['Returning.Visits'].mean().loc[MONTH\_NAMES]

Month.Name  
January 445.976608  
February 549.354610  
March 583.470968  
April 651.740000  
May 601.135484  
June 496.180000  
July 438.509677  
August 437.522581  
September 426.173653  
October 509.209677  
November 569.716667  
December 490.274194  
Name: Returning.Visits, dtype: float64

pd.DataFrame(dataset\_group\_by\_month['Returning.Visits'].mean()).loc[MONTH\_NAMES].plot(kind='bar')  
plt.show()



## Prepare the dataset

train\_dataset\_with\_months

dataset2 = train\_dataset\_with\_months.copy()[['Day', 'Month.Name', 'Returning.Visits']]  
dataset2

Day Month.Name Returning.Visits  
Date   
2014-09-14 Sunday September 152  
2014-09-15 Monday September 231  
2014-09-16 Tuesday September 278  
2014-09-17 Wednesday September 287  
2014-09-18 Thursday September 236  
... ... ... ...  
2020-01-12 Sunday January 277  
2020-01-13 Monday January 515  
2020-01-14 Tuesday January 434  
2020-01-15 Wednesday January 394  
2020-01-16 Thursday January 441  
  
[1951 rows x 3 columns]

def windowize\_dataset(dataset):  
 for i in range(WINDOW\_SIZE):  
 dataset[f'Returning.Visits[t-{i+1}]'] = dataset['Returning.Visits'].shift(periods=i+1)  
 return dataset  
  
dataset2 = windowize\_dataset(dataset2.copy())  
dataset2

Day Month.Name Returning.Visits Returning.Visits[t-1] \  
Date   
2014-09-14 Sunday September 152 NaN   
2014-09-15 Monday September 231 152.0   
2014-09-16 Tuesday September 278 231.0   
2014-09-17 Wednesday September 287 278.0   
2014-09-18 Thursday September 236 287.0   
... ... ... ... ...   
2020-01-12 Sunday January 277 253.0   
2020-01-13 Monday January 515 277.0   
2020-01-14 Tuesday January 434 515.0   
2020-01-15 Wednesday January 394 434.0   
2020-01-16 Thursday January 441 394.0   
  
 Returning.Visits[t-2] Returning.Visits[t-3]   
Date   
2014-09-14 NaN NaN   
2014-09-15 NaN NaN   
2014-09-16 152.0 NaN   
2014-09-17 231.0 152.0   
2014-09-18 278.0 231.0   
... ... ...   
2020-01-12 372.0 464.0   
2020-01-13 253.0 372.0   
2020-01-14 277.0 253.0   
2020-01-15 515.0 277.0   
2020-01-16 434.0 515.0   
  
[1951 rows x 6 columns]

dataset2 = dataset2.dropna()  
dataset2

Day Month.Name Returning.Visits Returning.Visits[t-1] \  
Date   
2014-09-17 Wednesday September 287 278.0   
2014-09-18 Thursday September 236 287.0   
2014-09-19 Friday September 241 236.0   
2014-09-20 Saturday September 133 241.0   
2014-09-21 Sunday September 175 133.0   
... ... ... ... ...   
2020-01-12 Sunday January 277 253.0   
2020-01-13 Monday January 515 277.0   
2020-01-14 Tuesday January 434 515.0   
2020-01-15 Wednesday January 394 434.0   
2020-01-16 Thursday January 441 394.0   
  
 Returning.Visits[t-2] Returning.Visits[t-3]   
Date   
2014-09-17 231.0 152.0   
2014-09-18 278.0 231.0   
2014-09-19 287.0 278.0   
2014-09-20 236.0 287.0   
2014-09-21 241.0 236.0   
... ... ...   
2020-01-12 372.0 464.0   
2020-01-13 253.0 372.0   
2020-01-14 277.0 253.0   
2020-01-15 515.0 277.0   
2020-01-16 434.0 515.0   
  
[1948 rows x 6 columns]

rv\_cols = [f"Returning.Visits[t-{i+1}]" for i in range(WINDOW\_SIZE)]  
  
dataset2\_rv\_history\_features = dataset2[rv\_cols]  
dataset2\_rv\_history\_features

Returning.Visits[t-1] Returning.Visits[t-2] \  
Date   
2014-09-17 278.0 231.0   
2014-09-18 287.0 278.0   
2014-09-19 236.0 287.0   
2014-09-20 241.0 236.0   
2014-09-21 133.0 241.0   
... ... ...   
2020-01-12 253.0 372.0   
2020-01-13 277.0 253.0   
2020-01-14 515.0 277.0   
2020-01-15 434.0 515.0   
2020-01-16 394.0 434.0   
  
 Returning.Visits[t-3]   
Date   
2014-09-17 152.0   
2014-09-18 231.0   
2014-09-19 278.0   
2014-09-20 287.0   
2014-09-21 236.0   
... ...   
2020-01-12 464.0   
2020-01-13 372.0   
2020-01-14 253.0   
2020-01-15 277.0   
2020-01-16 515.0   
  
[1948 rows x 3 columns]

dataset2\_cat\_features = dataset2[['Day', 'Month.Name']]  
dataset2\_cat\_features

Day Month.Name  
Date   
2014-09-17 Wednesday September  
2014-09-18 Thursday September  
2014-09-19 Friday September  
2014-09-20 Saturday September  
2014-09-21 Sunday September  
... ... ...  
2020-01-12 Sunday January  
2020-01-13 Monday January  
2020-01-14 Tuesday January  
2020-01-15 Wednesday January  
2020-01-16 Thursday January  
  
[1948 rows x 2 columns]

train\_dataset2 = dataset2['Returning.Visits']  
train\_dataset2

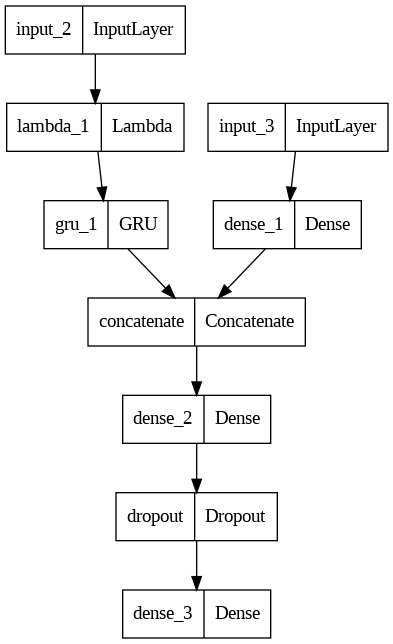
Date  
2014-09-17 287  
2014-09-18 236  
2014-09-19 241  
2014-09-20 133  
2014-09-21 175  
 ...   
2020-01-12 277  
2020-01-13 515  
2020-01-14 434  
2020-01-15 394  
2020-01-16 441  
Name: Returning.Visits, Length: 1948, dtype: int64

## Building the model

from tensorflow.keras.layers import Concatenate, Dropout  
  
tf.random.set\_seed(42)  
def build\_model\_3():  
 seq\_input = Input(shape=(WINDOW\_SIZE,))  
 lambda\_layer = Lambda(lambda x: x[:, tf.newaxis])(seq\_input)  
 rnn\_layer = GRU(64, activation='relu')(lambda\_layer)  
  
 cat\_input = Input(shape=(2,))  
 cat\_dense\_layer = Dense(32, activation='relu')(cat\_input)  
  
 concat\_layer = Concatenate()([rnn\_layer, cat\_dense\_layer])  
 dense\_layer1 = Dense(128, activation='relu')(concat\_layer)  
 dropout\_layer = Dropout(0.5)(dense\_layer1)  
 output\_layer = Dense(1, activation='linear')(dropout\_layer)  
  
 return Model(inputs=[seq\_input, cat\_input], outputs=output\_layer, name="model\_3")  
  
model\_3 = build\_model\_3()  
model\_3.compile(  
 loss=tf.keras.losses.MeanAbsoluteError(),  
 optimizer=tf.keras.optimizers.Adam()  
)  
  
model\_3.summary()

Model: "model\_3"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param # Connected to   
==================================================================================================  
 input\_2 (InputLayer) [(None, 3)] 0 []   
   
 lambda\_1 (Lambda) (None, 1, 3) 0 ['input\_2[0][0]']   
   
 input\_3 (InputLayer) [(None, 2)] 0 []   
   
 gru\_1 (GRU) (None, 64) 13248 ['lambda\_1[0][0]']   
   
 dense\_1 (Dense) (None, 32) 96 ['input\_3[0][0]']   
   
 concatenate (Concatenate) (None, 96) 0 ['gru\_1[0][0]',   
 'dense\_1[0][0]']   
   
 dense\_2 (Dense) (None, 128) 12416 ['concatenate[0][0]']   
   
 dropout (Dropout) (None, 128) 0 ['dense\_2[0][0]']   
   
 dense\_3 (Dense) (None, 1) 129 ['dropout[0][0]']   
   
==================================================================================================  
Total params: 25889 (101.13 KB)  
Trainable params: 25889 (101.13 KB)  
Non-trainable params: 0 (0.00 Byte)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tf.keras.utils.plot\_model(model\_3)



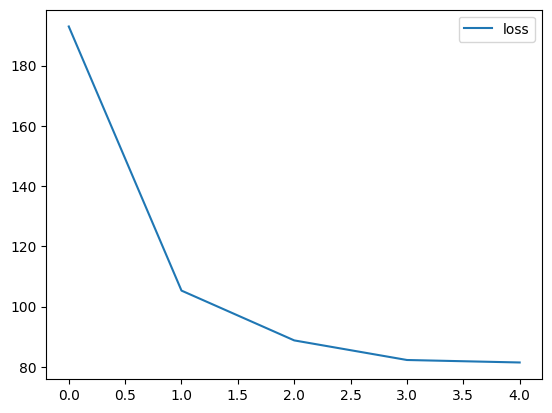
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder  
X\_cat\_encoder = OrdinalEncoder(categories = [DAYS\_OF\_WEEK, MONTH\_NAMES])  
X\_cat\_encoded = X\_cat\_encoder.fit\_transform(dataset2\_cat\_features)  
X\_cat\_encoded, X\_cat\_encoder.categories\_

(array([[2., 8.],  
 [3., 8.],  
 [4., 8.],  
 ...,  
 [1., 0.],  
 [2., 0.],  
 [3., 0.]]),  
 [array(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',  
 'Sunday'], dtype=object),  
 array(['January', 'February', 'March', 'April', 'May', 'June', 'July',  
 'August', 'September', 'October', 'November', 'December'],  
 dtype=object)])

from tensorflow.data import Dataset  
  
model3\_history = model\_3.fit(x=[dataset2\_rv\_history\_features, X\_cat\_encoded], y=train\_dataset2, epochs=5)  
pd.DataFrame(model3\_history.history).plot()

Epoch 1/5  
61/61 [==============================] - 2s 3ms/step - loss: 192.9761  
Epoch 2/5  
61/61 [==============================] - 0s 3ms/step - loss: 105.3255  
Epoch 3/5  
61/61 [==============================] - 0s 3ms/step - loss: 88.7999  
Epoch 4/5  
61/61 [==============================] - 0s 3ms/step - loss: 82.2974  
Epoch 5/5  
61/61 [==============================] - 0s 3ms/step - loss: 81.4735

<Axes: >



test\_dataset2 = windowize\_dataset(whole\_dataset[TEST\_DATA\_BOUNDARY\_INDEX-WINDOW\_SIZE:].copy())  
test\_dataset2['Month.Name'] = pd.Series(test\_dataset2.index, index=test\_dataset2.index)\  
 .apply(lambda x: calendar.month\_name[x.month])  
test\_dataset2 = test\_dataset2.dropna()  
test\_dataset2

Row Day Day.Of.Week Page.Loads Unique.Visits \  
Date   
2020-01-16 1951 Thursday 5 3817 2951   
2020-01-17 1952 Friday 6 3175 2419   
2020-01-18 1953 Saturday 7 2336 1927   
2020-01-19 1954 Sunday 1 2597 2031   
2020-01-20 1955 Monday 2 3715 2948   
... ... ... ... ... ...   
2020-08-15 2163 Saturday 7 2221 1696   
2020-08-16 2164 Sunday 1 2724 2037   
2020-08-17 2165 Monday 2 3456 2638   
2020-08-18 2166 Tuesday 3 3581 2683   
2020-08-19 2167 Wednesday 4 2064 1564   
  
 First.Time.Visits Returning.Visits Returning.Visits[t-1] \  
Date   
2020-01-16 2510 441 394.0   
2020-01-17 2006 413 441.0   
2020-01-18 1681 246 413.0   
2020-01-19 1717 314 246.0   
2020-01-20 2505 443 314.0   
... ... ... ...   
2020-08-15 1373 323 386.0   
2020-08-16 1686 351 323.0   
2020-08-17 2181 457 351.0   
2020-08-18 2184 499 457.0   
2020-08-19 1297 267 499.0   
  
 Returning.Visits[t-2] Returning.Visits[t-3] Month.Name   
Date   
2020-01-16 434.0 515.0 January   
2020-01-17 394.0 434.0 January   
2020-01-18 441.0 394.0 January   
2020-01-19 413.0 441.0 January   
2020-01-20 246.0 413.0 January   
... ... ... ...   
2020-08-15 458.0 427.0 August   
2020-08-16 386.0 458.0 August   
2020-08-17 323.0 386.0 August   
2020-08-18 351.0 323.0 August   
2020-08-19 457.0 351.0 August   
  
[217 rows x 11 columns]

X\_test\_rv\_history\_input = test\_dataset2[rv\_cols]  
X\_test\_rv\_history\_input

Returning.Visits[t-1] Returning.Visits[t-2] \  
Date   
2020-01-16 394.0 434.0   
2020-01-17 441.0 394.0   
2020-01-18 413.0 441.0   
2020-01-19 246.0 413.0   
2020-01-20 314.0 246.0   
... ... ...   
2020-08-15 386.0 458.0   
2020-08-16 323.0 386.0   
2020-08-17 351.0 323.0   
2020-08-18 457.0 351.0   
2020-08-19 499.0 457.0   
  
 Returning.Visits[t-3]   
Date   
2020-01-16 515.0   
2020-01-17 434.0   
2020-01-18 394.0   
2020-01-19 441.0   
2020-01-20 413.0   
... ...   
2020-08-15 427.0   
2020-08-16 458.0   
2020-08-17 386.0   
2020-08-18 323.0   
2020-08-19 351.0   
  
[217 rows x 3 columns]

X\_test\_cat\_input = test\_dataset2[['Day', 'Month.Name']]  
X\_test\_cat\_input = X\_cat\_encoder.transform(X\_test\_cat\_input)  
X\_test\_cat\_input.shape, X\_test\_cat\_input[:5]

((217, 2),  
 array([[3., 0.],  
 [4., 0.],  
 [5., 0.],  
 [6., 0.],  
 [0., 0.]]))

model\_3\_preds = model\_3.predict([X\_test\_rv\_history\_input, X\_test\_cat\_input])  
model\_3\_preds[:15]

7/7 [==============================] - 0s 2ms/step

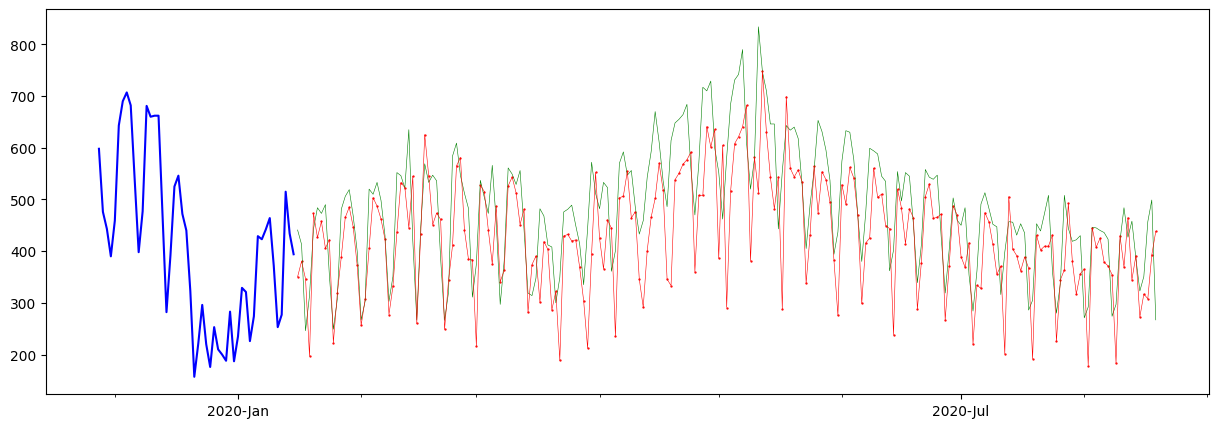
array([[350.29953],  
 [380.3174 ],  
 [346.55417],  
 [196.7649 ],  
 [473.16367],  
 [426.6182 ],  
 [457.4037 ],  
 [405.64368],  
 [422.29507],  
 [222.63646],  
 [318.1331 ],  
 [388.89676],  
 [465.1659 ],  
 [485.11026],  
 [446.89838]], dtype=float32)

y\_dataset = test\_dataset2['Returning.Visits']  
y\_dataset

Date  
2020-01-16 441  
2020-01-17 413  
2020-01-18 246  
2020-01-19 314  
2020-01-20 443  
 ...   
2020-08-15 323  
2020-08-16 351  
2020-08-17 457  
2020-08-18 499  
2020-08-19 267  
Name: Returning.Visits, Length: 217, dtype: int64

def evaluate\_model\_predictions(y\_true, predictions, model\_name):  
 metrics = evaluate\_predictions(y\_true, predictions)  
  
 MODEL\_METRICS.loc[model\_name] = metrics  
 plot\_time\_series(predictions.ravel(), start\_index=1900)  
 return metrics  
  
evaluate\_model\_predictions(y\_dataset, model\_3\_preds, 'model\_3 (multi-input)')

{'mae': 74.8837408953548,  
 'mse': 9136.574399241705,  
 'rmse': 95.58542984807728,  
 'mape': 0.15742602603819877}



MODEL\_METRICS

mae mse rmse mape  
model\_0 72.198157 8508.622120 92.242193 0.167139  
model\_1 74.449231 9023.324910 94.991183 0.166645  
model\_3 (multi-input) 74.883741 9136.574399 95.585430 0.157426

# Model 4: Ensemble methods

def build\_model\_5(n\_models, loss\_fns):  
 models = []  
 for loss\_fn in loss\_fns:  
 print(f"Training {n\_models} models for {loss\_fn} loss...")  
 for i in range(n\_models):  
 model = Sequential([  
 Input(shape=(WINDOW\_SIZE,)),  
 Lambda(lambda x: tf.expand\_dims(x, axis=1)),  
 GRU(128, activation='relu'),  
 Dense(1, activation='linear')  
 ])  
  
 model.compile(loss=loss\_fn, optimizer=tf.keras.optimizers.Adam())  
 models.append(model)  
  
  
 return models  
  
  
model\_5 = build\_model\_5(n\_models=5, loss\_fns=['mae', 'mse', 'mape'])  
model\_5

Training 5 models for mae loss...  
Training 5 models for mse loss...  
Training 5 models for mape loss...

[<keras.src.engine.sequential.Sequential at 0x7c8ef90f1e70>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef910d990>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef9159e40>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef915a9e0>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef9152e90>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef8fb2d10>,  
 <keras.src.engine.sequential.Sequential at 0x7c8efa87c2e0>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef8ffb2b0>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef918f790>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef918c0a0>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef8ff9d80>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef9052e30>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef908b1f0>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef9158100>,  
 <keras.src.engine.sequential.Sequential at 0x7c8ef8ecb250>]

model\_5[0].summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 lambda\_2 (Lambda) (None, 1, 3) 0   
   
 gru\_2 (GRU) (None, 128) 51072   
   
 dense\_4 (Dense) (None, 1) 129   
   
=================================================================  
Total params: 51201 (200.00 KB)  
Trainable params: 51201 (200.00 KB)  
Non-trainable params: 0 (0.00 Byte)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for i, model in enumerate(model\_5):  
 print(f"Training model {i+1} out of {len(model\_5)} models")  
 model.fit(train\_dataset, epochs=5, verbose=0)

Training model 1 out of 15 models  
Training model 2 out of 15 models  
Training model 3 out of 15 models  
Training model 4 out of 15 models  
Training model 5 out of 15 models  
Training model 6 out of 15 models  
Training model 7 out of 15 models  
Training model 8 out of 15 models  
Training model 9 out of 15 models  
Training model 10 out of 15 models  
Training model 11 out of 15 models  
Training model 12 out of 15 models  
Training model 13 out of 15 models  
Training model 14 out of 15 models  
Training model 15 out of 15 models

def ensemble\_prediction(models):  
 predictions = []  
 for model in models:  
 pred = model.predict(test\_dataset, verbose=0)  
 predictions.append(pred)  
  
 return np.array(predictions)  
  
model\_5\_all\_preds = ensemble\_prediction(model\_5)  
model\_5\_all\_preds.shape

WARNING:tensorflow:5 out of the last 14 calls to <function Model.make\_predict\_function.<locals>.predict\_function at 0x7c8ef63175b0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce\_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling\_retracing and https://www.tensorflow.org/api\_docs/python/tf/function for more details.  
WARNING:tensorflow:6 out of the last 16 calls to <function Model.make\_predict\_function.<locals>.predict\_function at 0x7c8ef6317d00> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce\_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling\_retracing and https://www.tensorflow.org/api\_docs/python/tf/function for more details.

(15, 217, 1)

model\_5\_all\_preds.shape

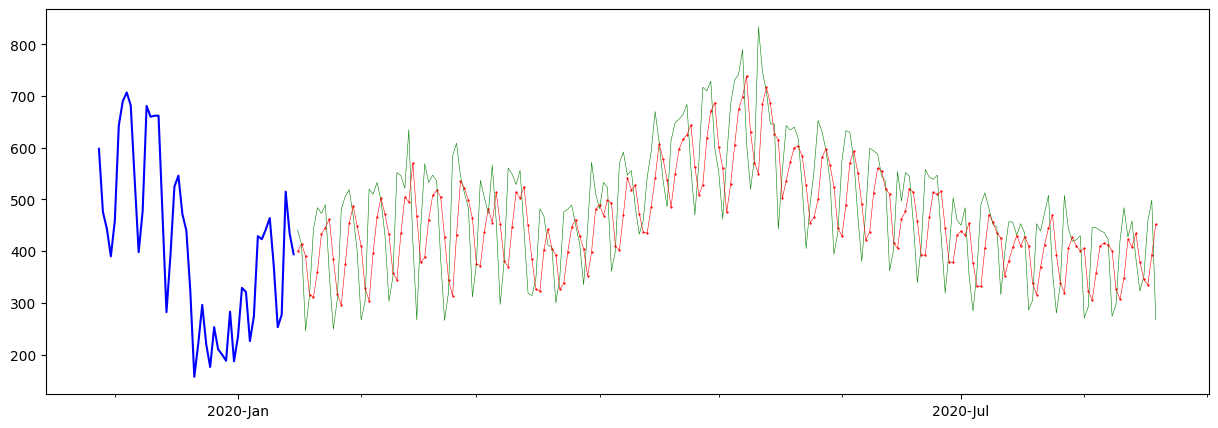
(15, 217, 1)

def aggregate\_ensemble\_predictions(predictions):  
 return tf.reduce\_mean(predictions, axis=0).numpy()  
  
model\_5\_preds = aggregate\_ensemble\_predictions(model\_5\_all\_preds)  
model\_5\_preds.shape

(217, 1)

evaluate\_model\_predictions(y\_true, model\_5\_preds, 'model\_5 (ensemble)')

{'mae': 76.62611452445456,  
 'mse': 9265.398361454503,  
 'rmse': 96.25693928987407,  
 'mape': 0.16981836578744697}



MODEL\_METRICS

mae mse rmse mape  
model\_0 72.198157 8508.622120 92.242193 0.167139  
model\_1 74.449231 9023.324910 94.991183 0.166645  
model\_3 (multi-input) 74.883741 9136.574399 95.585430 0.157426  
model\_5 (ensemble) 76.626115 9265.398361 96.256939 0.169818