**DAC\_Phase 3**

import pandas as pd

FILE\_LOCATION = '/content/daily-website-visitors.csv'

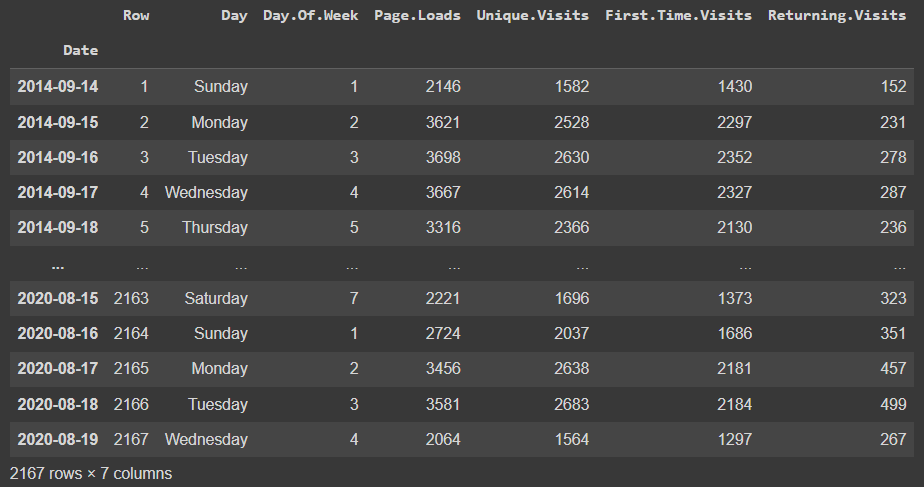
whole\_dataset = pd.read\_csv(FILE\_LOCATION,

index\_col='Date',

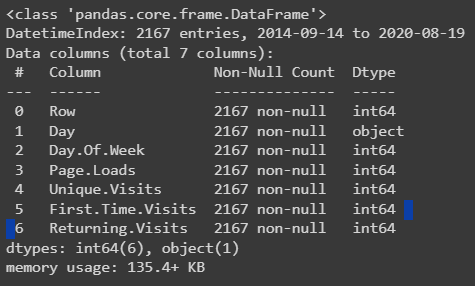
thousands=',')

whole\_dataset.index = pd.to\_datetime(whole\_dataset.index)

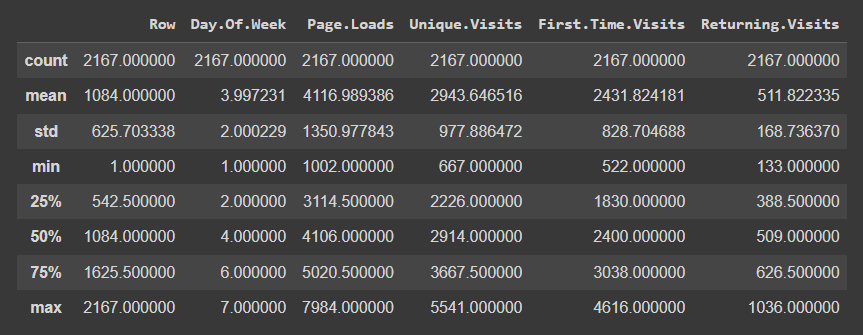
whole\_dataset



whole\_dataset.info()

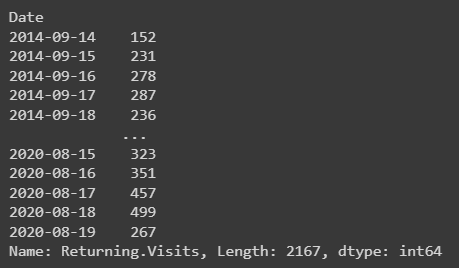


whole\_dataset.describe()



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

target\_column



len(target\_column)



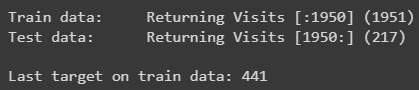
TEST\_DATA\_PERCENTAGE = 0.1

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]}")



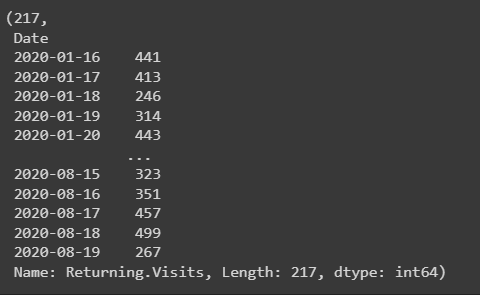
print(f"Train dataset ending values: {target\_column[TEST\_DATA\_BOUNDARY\_INDEX - 10: TEST\_DATA\_BOUNDARY\_INDEX].values}")

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



y\_true = target\_column[TEST\_DATA\_BOUNDARY\_INDEX : ]

len(y\_true), y\_true



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,

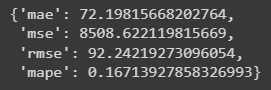
'mse': mse,

"rmse": rmse,

"mape": mape

}

evaluate\_predictions(y\_true, baseline\_predictions)



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')

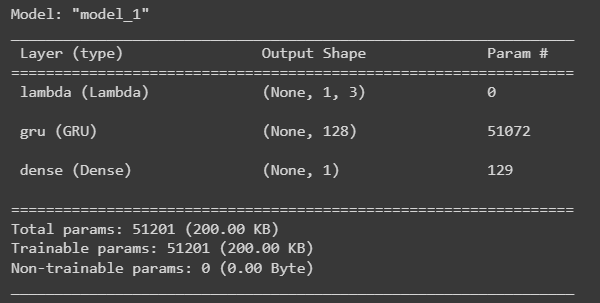
model\_1.compile(

loss=tf.keras.losses.MeanAbsoluteError(),

optimizer=tf.keras.optimizers.Adam()

)

model\_1.summary()



from tensorflow.keras.callbacks import ModelCheckpoint

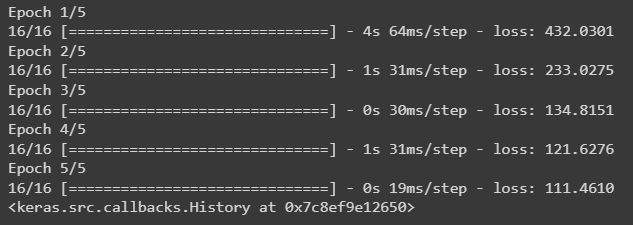
import os

def create\_checkpoint\_callback(model):

filepath = os.path.join('models', model.name)

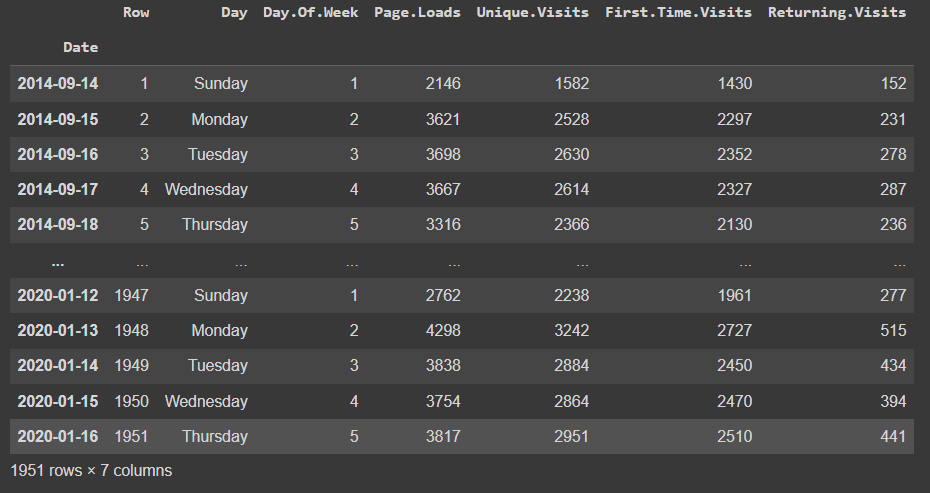
return ModelCheckpoint(filepath, monitor='loss', save\_weights\_only=True, save\_best\_only=True)

model\_1.fit(train\_dataset, epochs=5, callbacks=[ create\_checkpoint\_callback(model\_1) ])



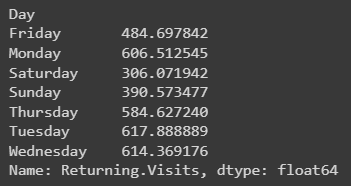
unbatched\_train\_dataset = whole\_dataset[:TEST\_DATA\_BOUNDARY\_INDEX + 1].copy()

unbatched\_train\_dataset



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

dataset\_by\_day['Returning.Visits'].mean()



import calendar

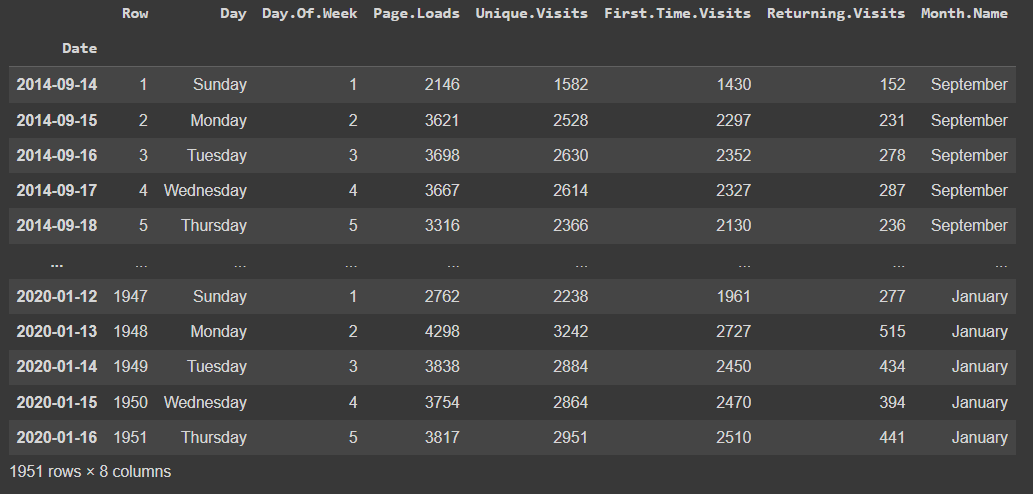
train\_dataset\_with\_months = unbatched\_train\_dataset.copy()

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])

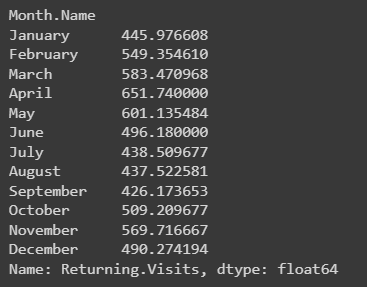
train\_dataset\_with\_months



MONTH\_NAMES = list(calendar.month\_name)[1:]

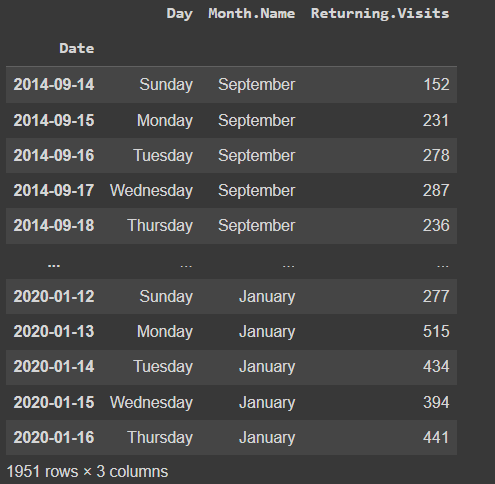
dataset\_group\_by\_month = train\_dataset\_with\_months.groupby(by='Month.Name')

dataset\_group\_by\_month['Returning.Visits'].mean().loc[MONTH\_NAMES]



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

dataset2



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

