PREDICTING FUTURE TRAFFIC TRENDS AND USER BEHAVIOR PATTERNS USING MACHINE LEARNING.

1. DATA LOADING AND PREPROCESSING:

- The code begins by loading historical website traffic data from CSV file using the Pandas library.
- The data typically contains daily metrics related to website visitors, such as the number of visitors, first-time visits, unique visits, and returning visits.
- The 'Date' column is set as the index of the Data Frame, and it handles thousands of separators in numbers during data loading.

PYTHON CODE

```python
import pandas as pd


df = pd.read_csv("/content/daily-website-visitors.csv",
                             index_col='Date',
                             thousands=',')
df.index = pd.to_datetime(whole_dataset.index)
df.info()
```

OUTPUT

Row	Day	Day.Of.Week	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits	
Date							
2014-09-14	1	Sunday	1	2146	1582	1430	152
2014-09-15	2	Monday	2	3621	2528	2297	231
2014-09-16	3	Tuesday	3	3698	2630	2352	278
2014-09-17	4	Wednesday	4	3667	2614	2327	287
2014-09-18	5	Thursday	5	3316	2366	2130	236
...	...	...	...	...	...	...	...
2020-08-15	2163	Saturday	7	2221	1696	1373	323
2020-08-16	2164	Sunday	1	2724	2037	1686	351

Row	Day	Day.Of.Week	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits	
Date							
2020-08-17	2165	Monday	2	3456	2638	2181	457
2020-08-18	2166	Tuesday	3	3581	2683	2184	499
2020-08-19	2167	Wednesday	4	2064	1564	1297	267

```
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)
```

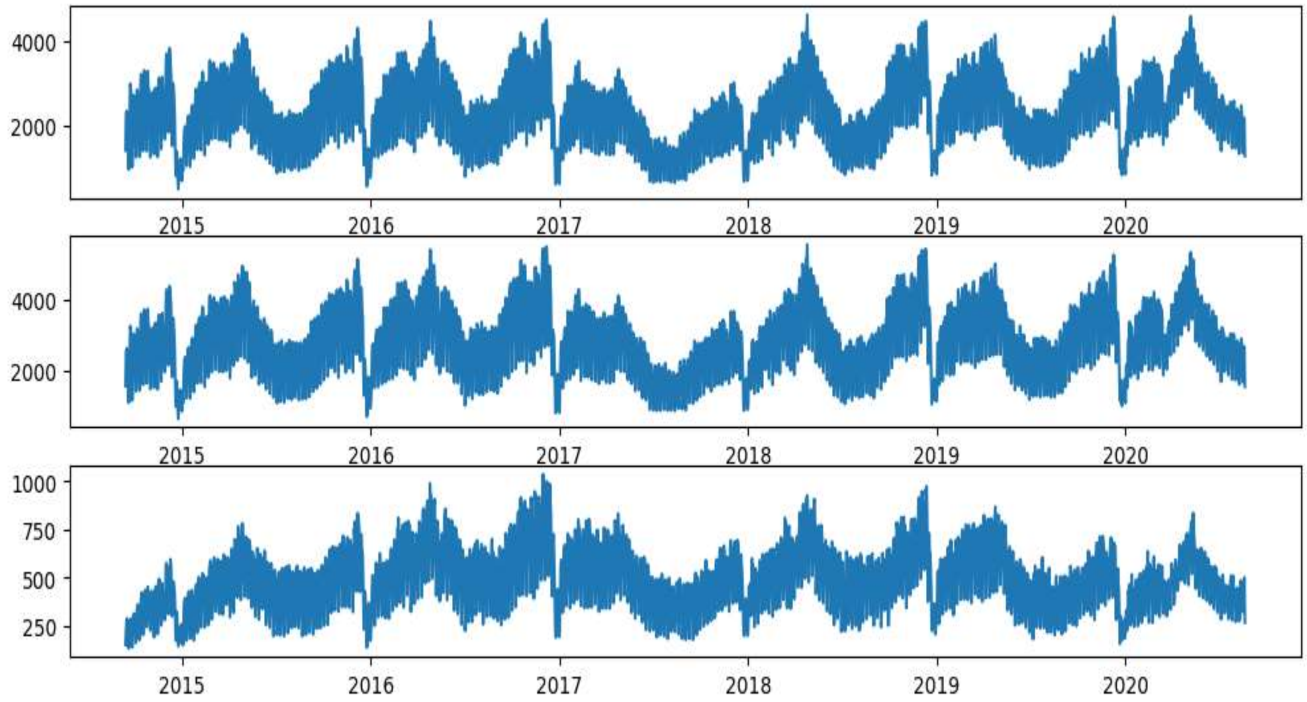
2. DATA VISUALIZATION:

- To gain a better understanding of the historical data, the code uses Matplotlib to create visualizations.
- It plots three key metrics: 'First.Time.Visits,' 'Unique.Visits,' and 'Returning.Visits.'
- These visualizations help identify trends and patterns in website traffic, such as daily or seasonal variations.

PYTHON CODE

```
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()
target_column = whole_dataset['Returning.Visits']
target_column
target_column.plot(figsize=(15, 3))
plt.show()
```

OUTPUT



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

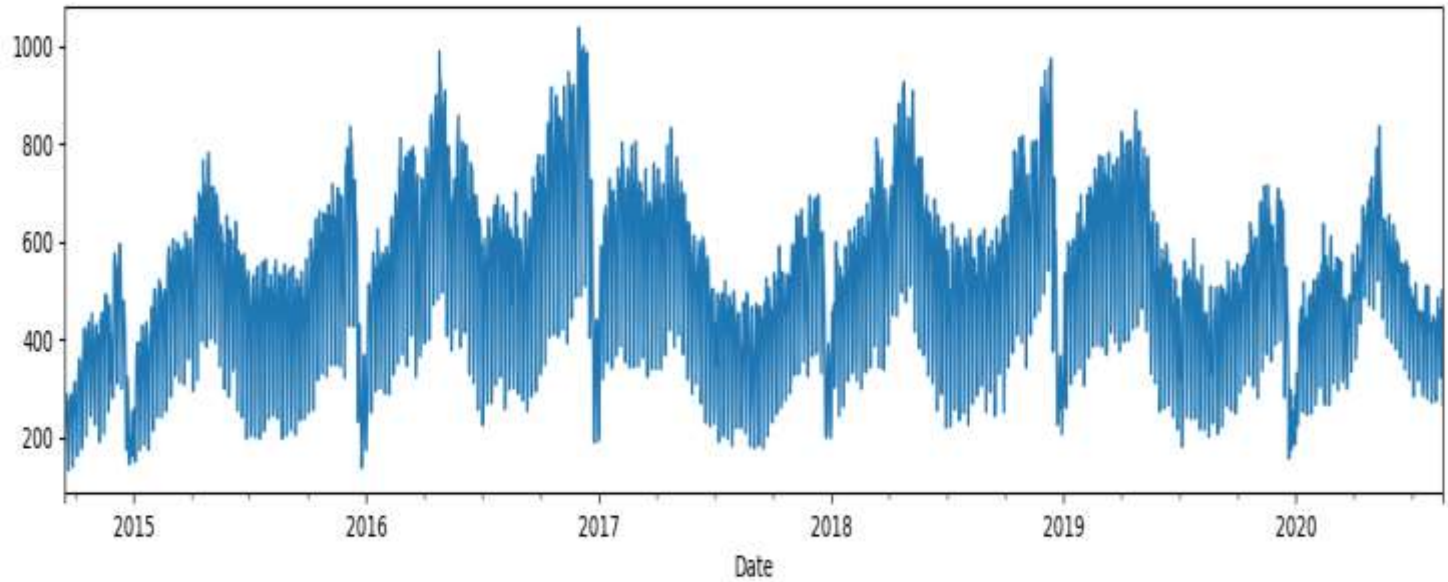
2020-08-16   351

2020-08-17   457

2020-08-18   499

2020-08-19   267

Name: Returning.Visits, Length: 2167, dtype: int64



3. DATA SPLITTING:

- The dataset is divided into training and testing data.
- A portion of the data (10%) is reserved for testing, and the boundary index is calculated.
- This separation is crucial for assessing the model's ability to make predictions on unseen data.

PYTHON CODE

```
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}")
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}")
```

OUTPUT

Train data: Returning Visits [:1950] (1951)

Test data: Returning Visits [1950:] (217)

Last target on train data: 441

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]

4. TIME SERIES DATASET CREATION:

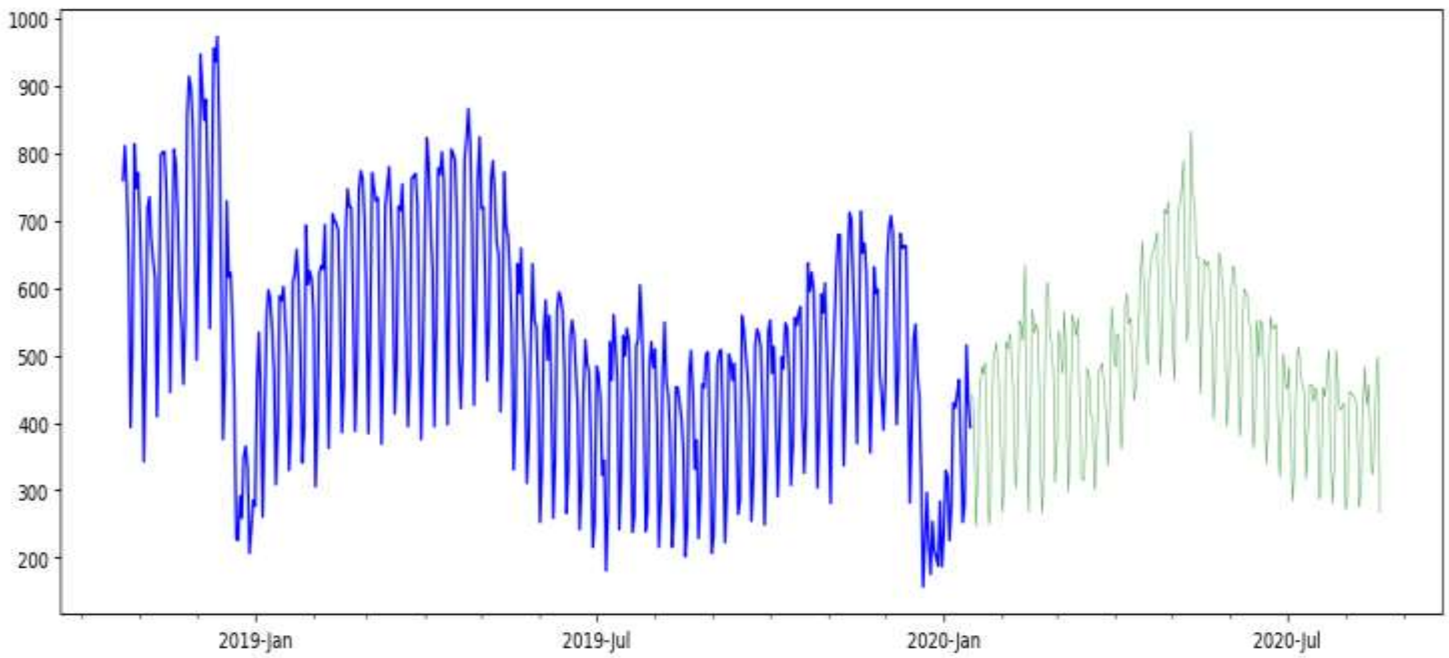
- Time series datasets are created to train machine learning models.
- The data is windowed with a specified window size (`WINDOW_SIZE`).
- This windowed data is used to predict future values based on historical sequences.

PYTHON CODE

```
 test_dataset =

timeseries_dataset_from_array(target_column[TEST_DATA_BOUNDARY_INDEX -
WINDOW_SIZE:],

                                                  target_column[TEST_DATA_BOUND
ARY_INDEX:],
                                                  sequence_length=WINDOW_SIZE
                                             )
len(test_dataset), len(list(test_dataset.unbatch()))
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()
```

OUTPUT



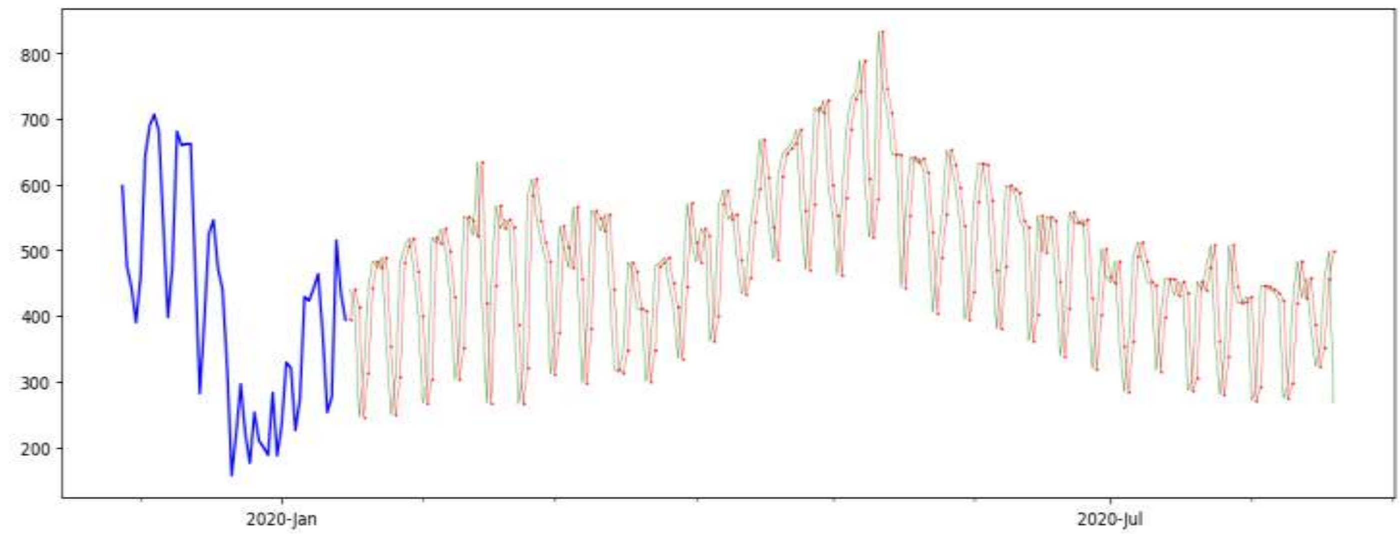
5. BASELINE MODEL:

- The code defines a simple baseline model named `NaiveForecastLayer`.
- This model makes predictions based on the last observed value in the time series.
- It serves as a reference point for comparing the performance of more advanced models.

PYTHON CODE

```
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()])
plot_time_series(baseline_predictions.ravel(), start_index=1900)
```

OUTPUT



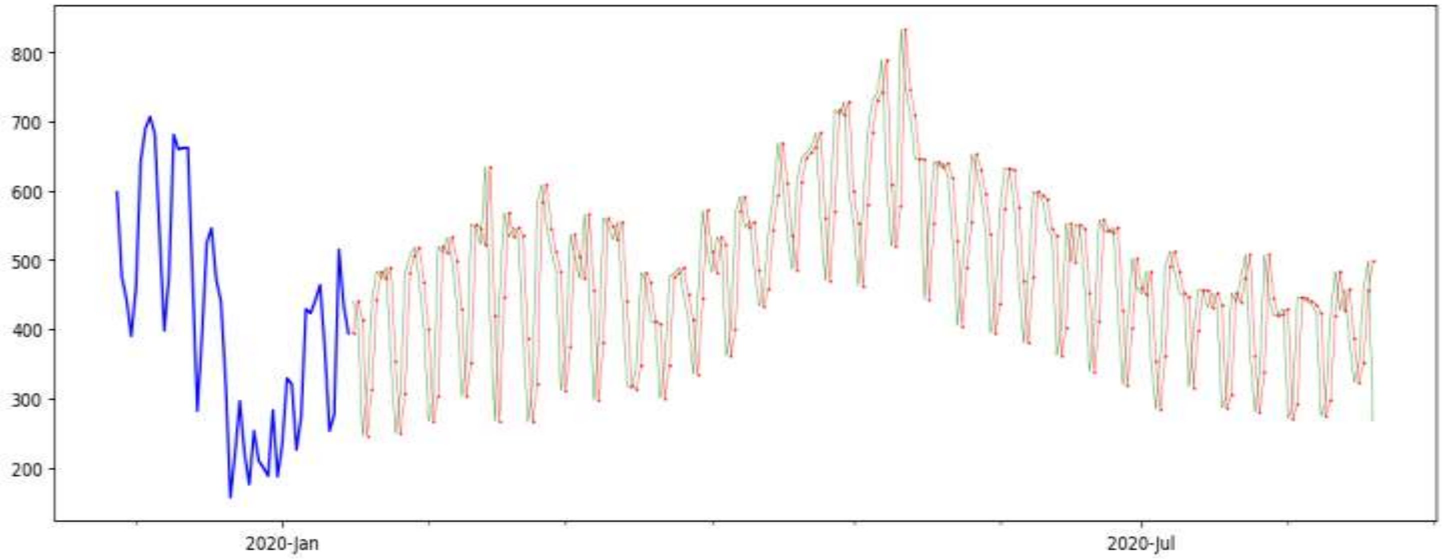
6. MODEL EVALUATION FUNCTIONS:

- Functions for evaluating the performance of machine learning models are defined.
- These functions calculate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).
- These metrics are used to assess how well the models are predicting future traffic.

PYTHON CODE

```
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)

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)
```

OUTPUT



7. RECURRENT NETWORK MODEL (GRU):

- The code introduces a more sophisticated machine learning model, a Gated Recurrent Unit (GRU).
- GRU is a type of recurrent neural network (RNN) designed to capture sequential patterns in time series data.
- This model is trained to make predictions based on historical traffic data.

PYTHON CODE

```
   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()
```

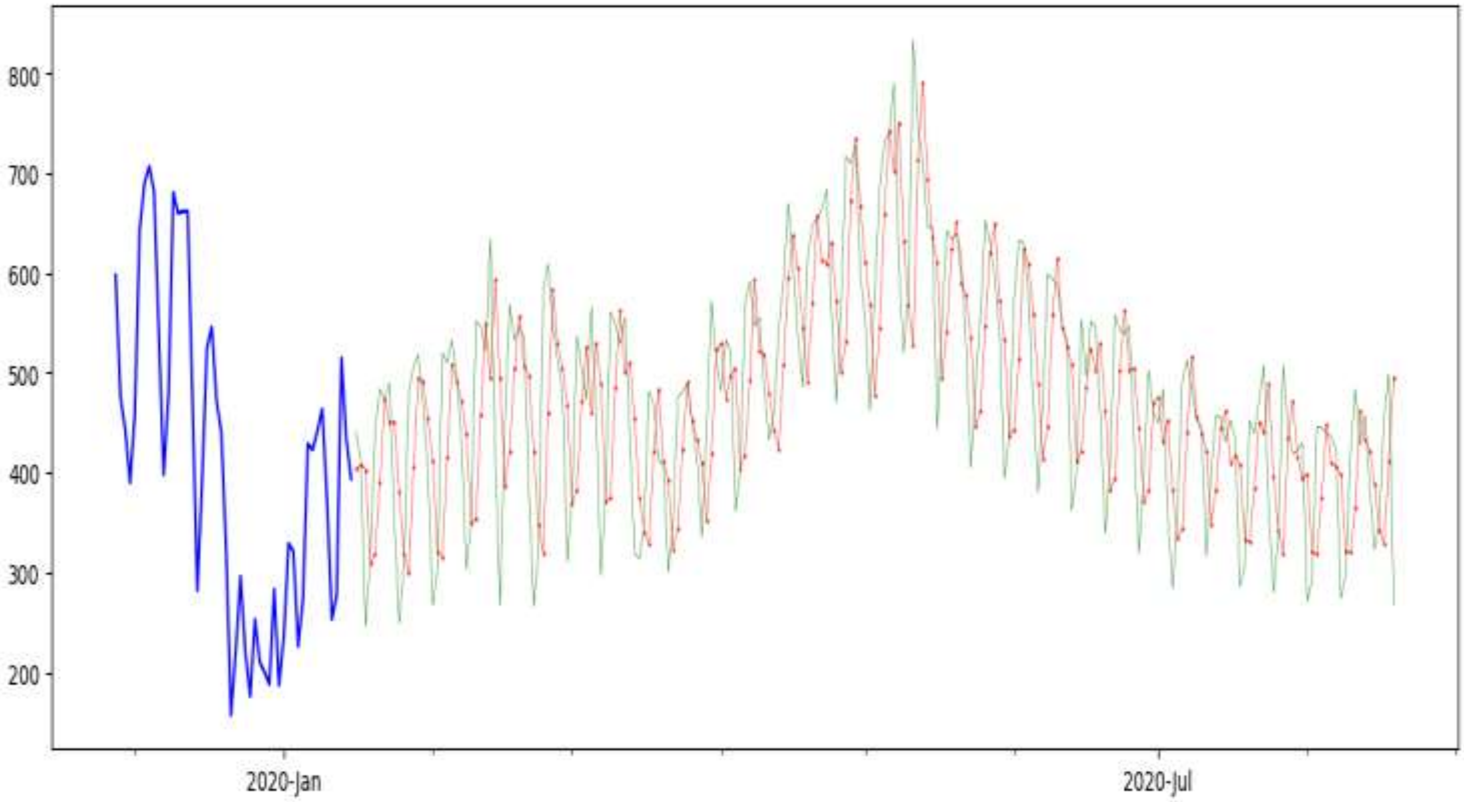
8. CHECKPOINT CALLBACK:

- A checkpoint callback is created to save the best model weights during training.
- This allows for the restoration of the model's state in case of interruptions or for later use.

PYTHON CODE

```
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) ])
evaluate_model(model_1)
```

OUTPUT



9. MULTI-INPUT MODEL:

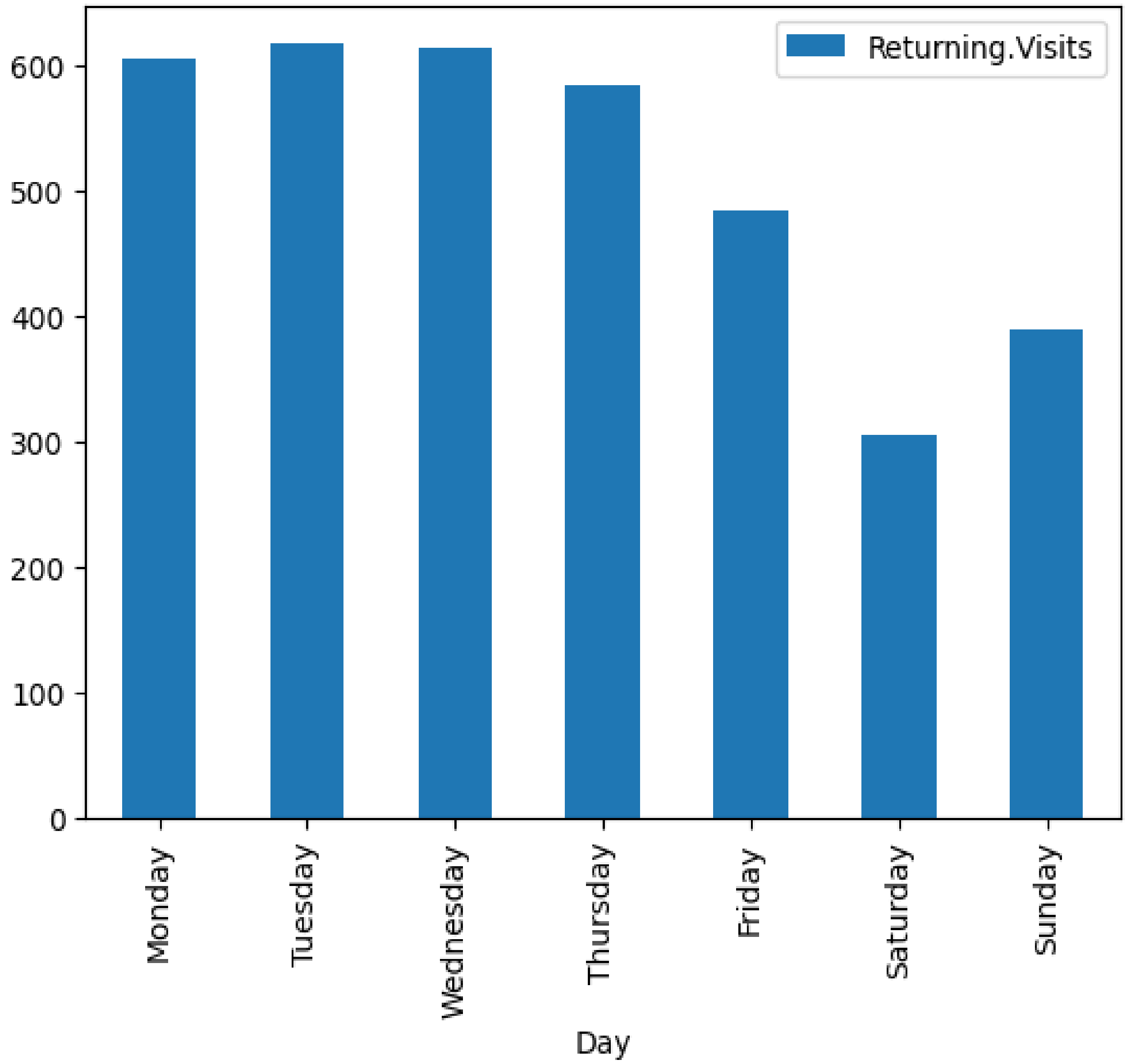
- In addition to the GRU model, the code defines a multi-input model that considers both historical traffic data and categorical features.
- Categorical features include the day of the week and the month.
- This model aims to capture more complex patterns by incorporating additional information.

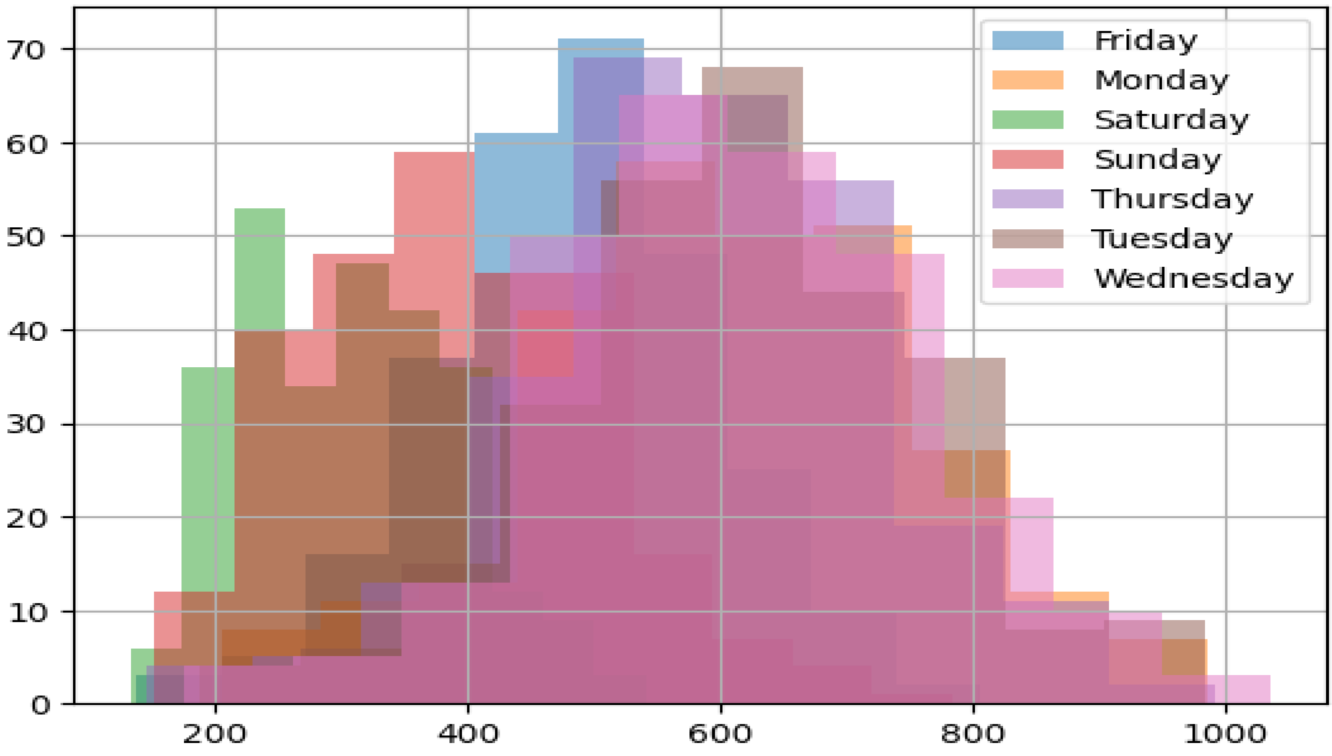
PYTHON CODE

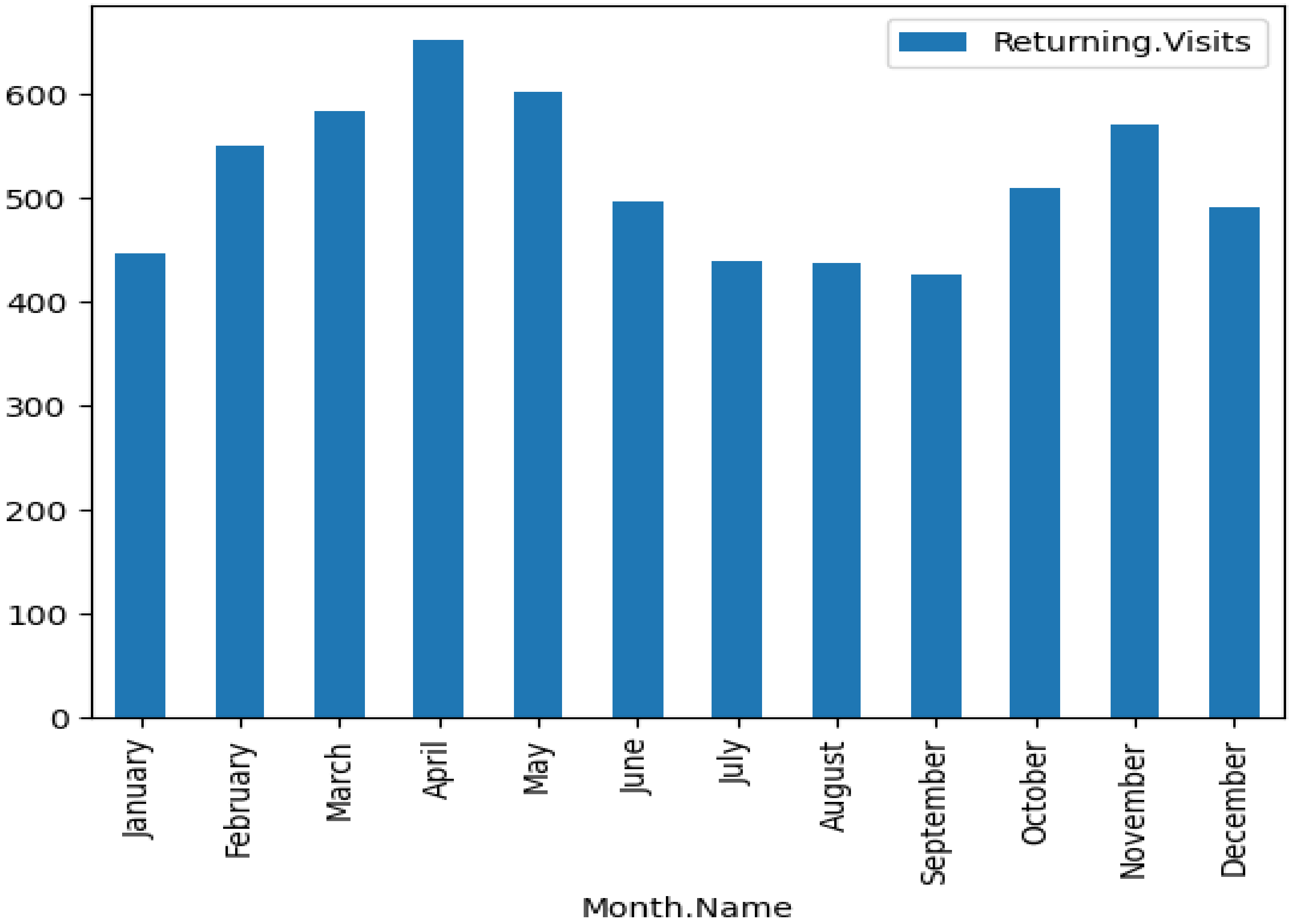
```
unbatched_train_dataset = whole_dataset[:TEST_DATA_BOUNDARY_INDEX + 1].copy()

unbatched_train_dataset
# Per Day of Week grouping
dataset_by_day = unbatched_train_dataset.groupby(by=['Day'])
dataset_by_day['Returning.Visits'].mean()
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')
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_m
onths.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]
pd.DataFrame(dataset_group_by_month['Returning.Visits'].mean()).loc[MONTH_NAME
S].plot(kind='bar')
plt.show()
```

OUTPUT

```
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
```




Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
0
10
20
30
40
50
60
70
200
400
600
800
1000


Returning.Visits
0
100
200
300
400
500
600
January
February
March
April
May
June
July
August
September
October
November
December
Month.Name

10. DATA ENCODING AND PREPROCESSING:

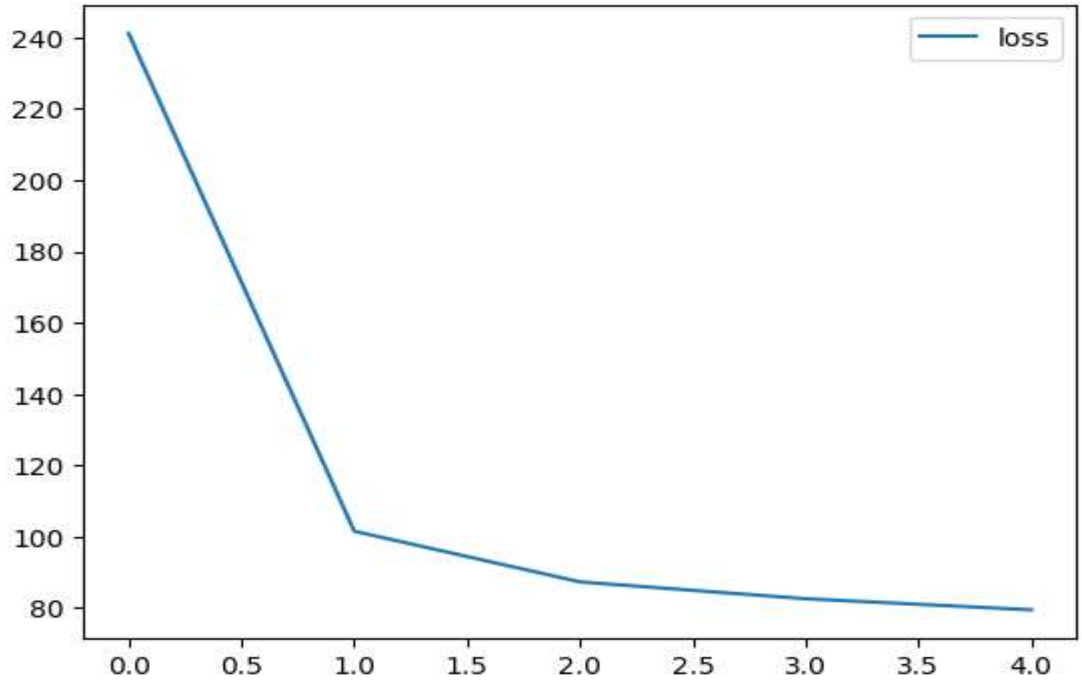
- To use categorical features in the multi-input model, the code performs data encoding.
- Categorical values like days of the week and months are converted into numerical representations using ordinal encoding.
- This allows the model to work with these features.

PYTHON CODE

```
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 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()
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
X_test_rv_history_input = test_dataset2[rv_cols]
X_test_rv_history_input
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]
model_3_preds = model_3.predict([X_test_rv_history_input, X_test_cat_input])
model_3_preds[:15]
y_dataset = test_dataset2['Returning.Visits']
y_dataset
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)')
```

OUTPUT

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




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

11. ENSEMBLE MODEL (MODEL 5):

- The code goes a step further by creating an ensemble of multiple GRU-based models.
- Each model in the ensemble uses a different loss function.
- An ensemble can capture diverse patterns in the data and potentially improve prediction accuracy.

PYTHON CODE

```
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
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)
```

OUTPUT

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

12. ENSEMBLE PREDICTION AND AGGREGATION:

- The ensemble of models is used to make predictions on the test dataset.
- The code aggregates these predictions by calculating the mean.
- Ensemble models often yield robust predictions by combining the outputs of multiple models.

PYTHON CODE

```
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
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
```

OUTPUT

(15, 217, 1)

(217, 1)

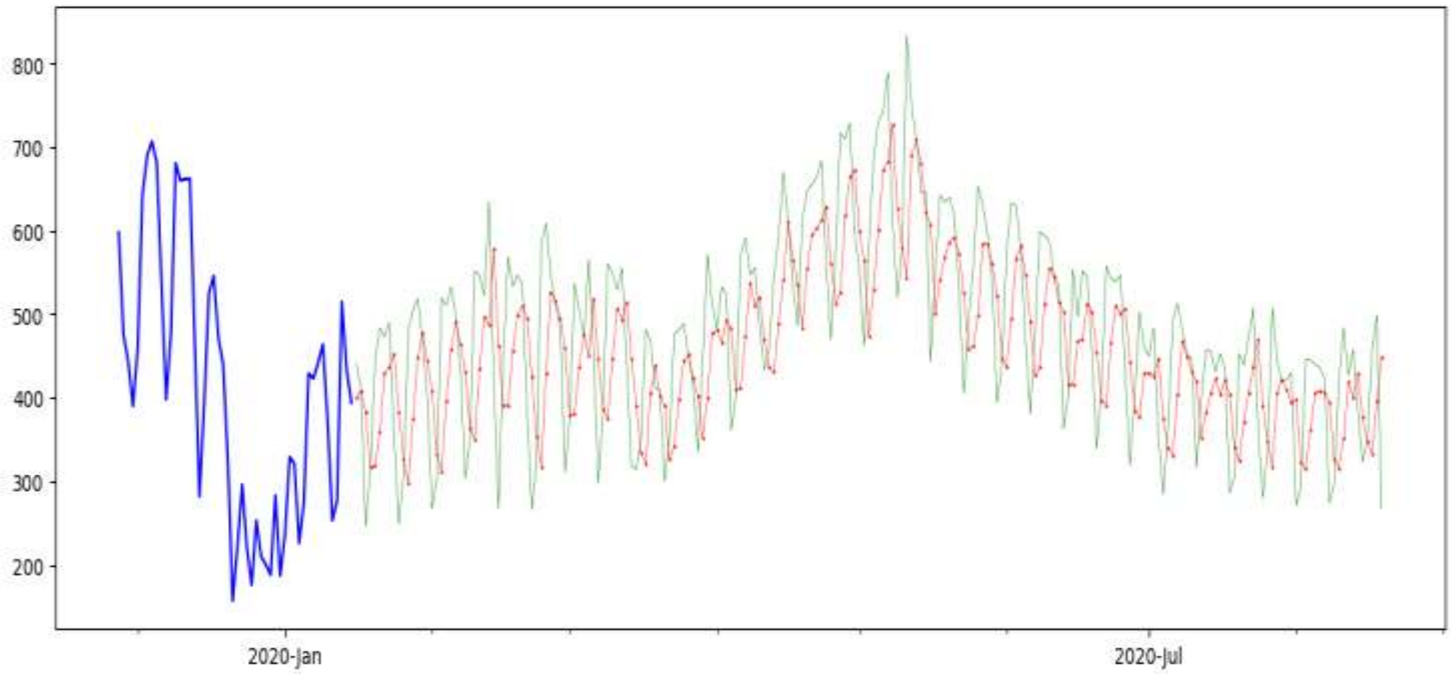
13. MODEL EVALUATION AND METRICS:

- After making predictions with each model, the code evaluates their performance using the evaluation functions defined earlier.
- Metrics such as MAE, MSE, RMSE, and MAPE are calculated.
- The code also visualizes the model's predictions alongside the actual traffic data, making it easier to understand the model's strengths and weaknesses.

PYTHON CODE

```
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
 evaluate_model_predictions(y_true, model_5_preds, 'model_5 (ensemble)')
```

OUTPUT



Conclusion

The above code demonstrates a comprehensive workflow for predicting future traffic trends and user behaviour patterns using machine learning:

- It loads and preprocesses historical website traffic data.
- It visualizes the data to understand trends and patterns.
- It splits the data into training and testing sets.
- It creates time series datasets for training and testing.
- It defines and trains various models, including a baseline model, a GRU model, a multi-input model, and an ensemble model.
- It evaluates each model's performance using a variety of metrics.
- It visualizes model predictions alongside the actual data, facilitating interpretation.

This approach allows businesses and website operators to make data-driven decisions, optimize their websites for peak traffic periods, and gain insights into user behaviour patterns.