SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya School of Engineering
(formerly K J Somaiya College of Engineering)

K. J. Somaiya School of Engineering, Mumbai-77
(Somaiya Vidyavihar University)
Department of Computer Engineering

Somaiya TRUST

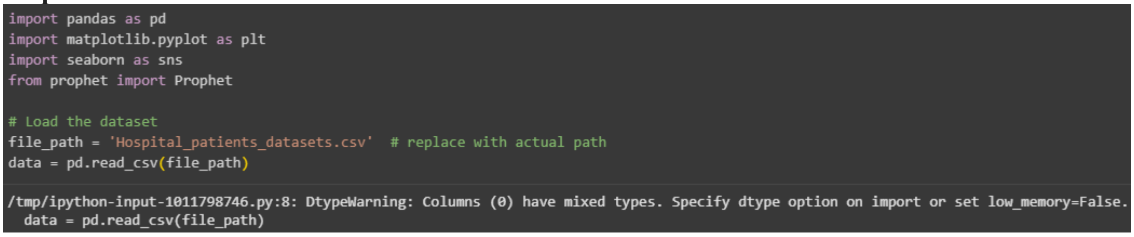
Course Name:	**Data Analysis Laboratory (216H03L501)**	**Semester:**	**V**
Date of Performance:	**13/10/2025**	**DIV/ Batch No:**	**HDA2**
Student Name:	**Aaryan Sharma**	**Roll No:**	**16010123012**

TITLE: Perform a time series analysis on health care (AR/MA/ARIMA)

AIM: To perform forecasting using time series analysis
Expected OUTCOME of Experiment:
CO4: Perform Time series Analytics and forecasting

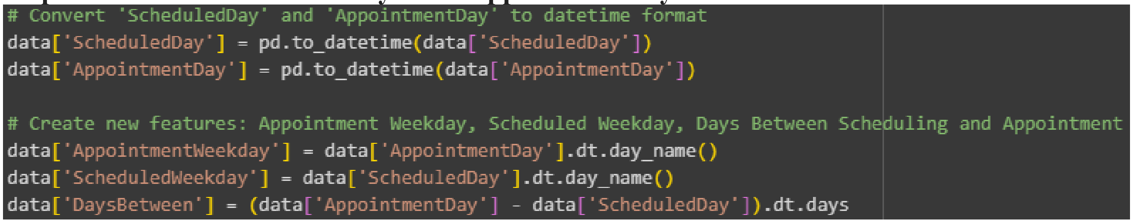
Books/ Journals/ Websites referred:
https://www.geeksforgeeks.org/machine-learning/time-series-analysis-and-forecasting/

Pre Lab/ Prior Concepts:
Students should have a basic understanding of: Time series Analytics and forecasting
Procedure:
Data set Used: Hospital_patients_datasets
Step1: Select and Load the dataset



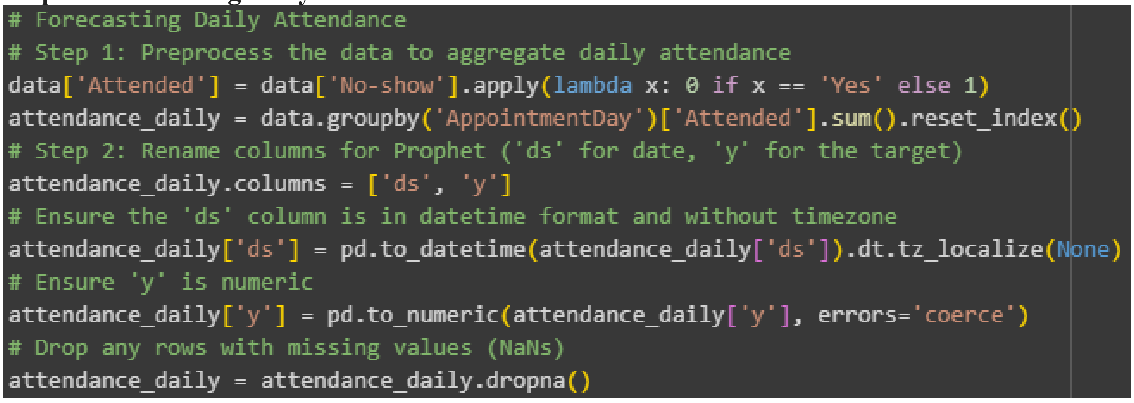
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from prophet import Prophet

# Load the dataset
file_path = 'Hospital_patients_datasets.csv'  # replace with actual path
data = pd.read_csv(file_path)

/tmp/ipython-input-1011798746.py:8: DtypeWarning: Columns (0) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv(file_path)
```

Step 2: Convert 'ScheduledDay' and 'AppointmentDay' to datetime format



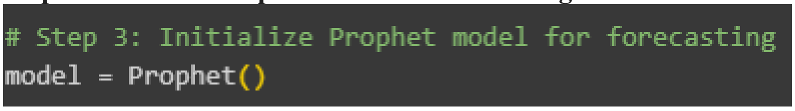
```
# Convert 'ScheduledDay' and 'AppointmentDay' to datetime format
data['ScheduledDay'] = pd.to_datetime(data['ScheduledDay'])
data['AppointmentDay'] = pd.to_datetime(data['AppointmentDay'])

# Create new features: Appointment Weekday, Scheduled Weekday, Days Between Scheduling and Appointment
data['AppointmentWeekday'] = data['AppointmentDay'].dt.day_name()
data['ScheduledWeekday'] = data['ScheduledDay'].dt.day_name()
data['DaysBetween'] = (data['AppointmentDay'] - data['ScheduledDay']).dt.days
```

Step 3: Forecasting Daily Attendance



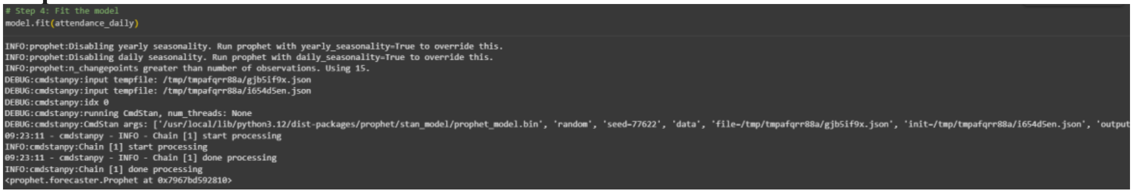
```
# Forecasting Daily Attendance
# Step 1: Preprocess the data to aggregate daily attendance
data['Attended'] = data['No-show'].apply(lambda x: 0 if x == 'Yes' else 1)
attendance_daily = data.groupby('AppointmentDay')['Attended'].sum().reset_index()
# Step 2: Rename columns for Prophet ('ds' for date, 'y' for the target)
attendance_daily.columns = ['ds', 'y']
# Ensure the 'ds' column is in datetime format and without timezone
attendance_daily['ds'] = pd.to_datetime(attendance_daily['ds']).dt.tz_localize(None)
# Ensure 'y' is numeric
attendance_daily['y'] = pd.to_numeric(attendance_daily['y'], errors='coerce')
# Drop any rows with missing values (NaNs)
attendance_daily = attendance_daily.dropna()
```

Step 4: Initialize Prophet model for forecasting



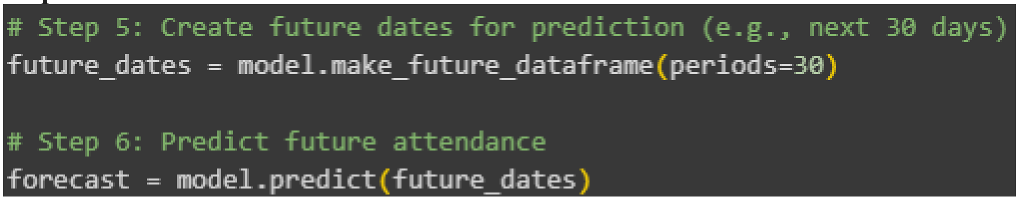
```
# Step 3: Initialize Prophet model for forecasting
model = Prophet()
```

Step 5: Fit the model



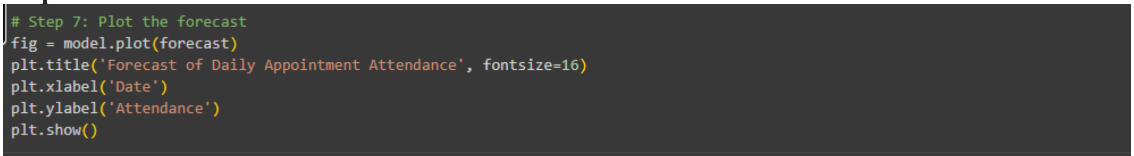
```
# Step 4: Fit the model
model.fit(attendance_daily)

INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:prophet:n_changepoints greater than number of observations. Using 15.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpafqrr88a/gjb5if9x.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpafqrr88a/i654d5en.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.12/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=77622', 'data', 'file=/tmp/tmpafqrr88a/gjb5if9x.json', 'init=/tmp/tmpafqrr88a/i654d5en.json', 'output
09:23:11 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
09:23:11 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x7967bd592810>
```

Step 6: Predict future attendance

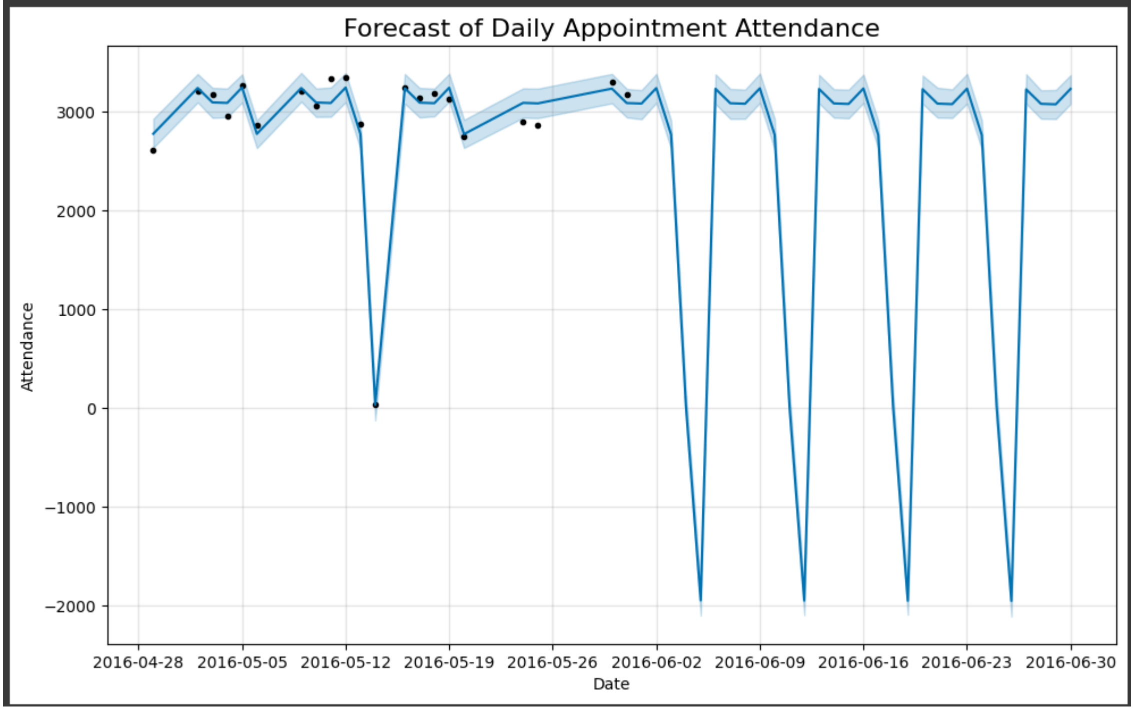


```
# Step 5: Create future dates for prediction (e.g., next 30 days)
future_dates = model.make_future_dataframe(periods=30)

# Step 6: Predict future attendance
forecast = model.predict(future_dates)
```

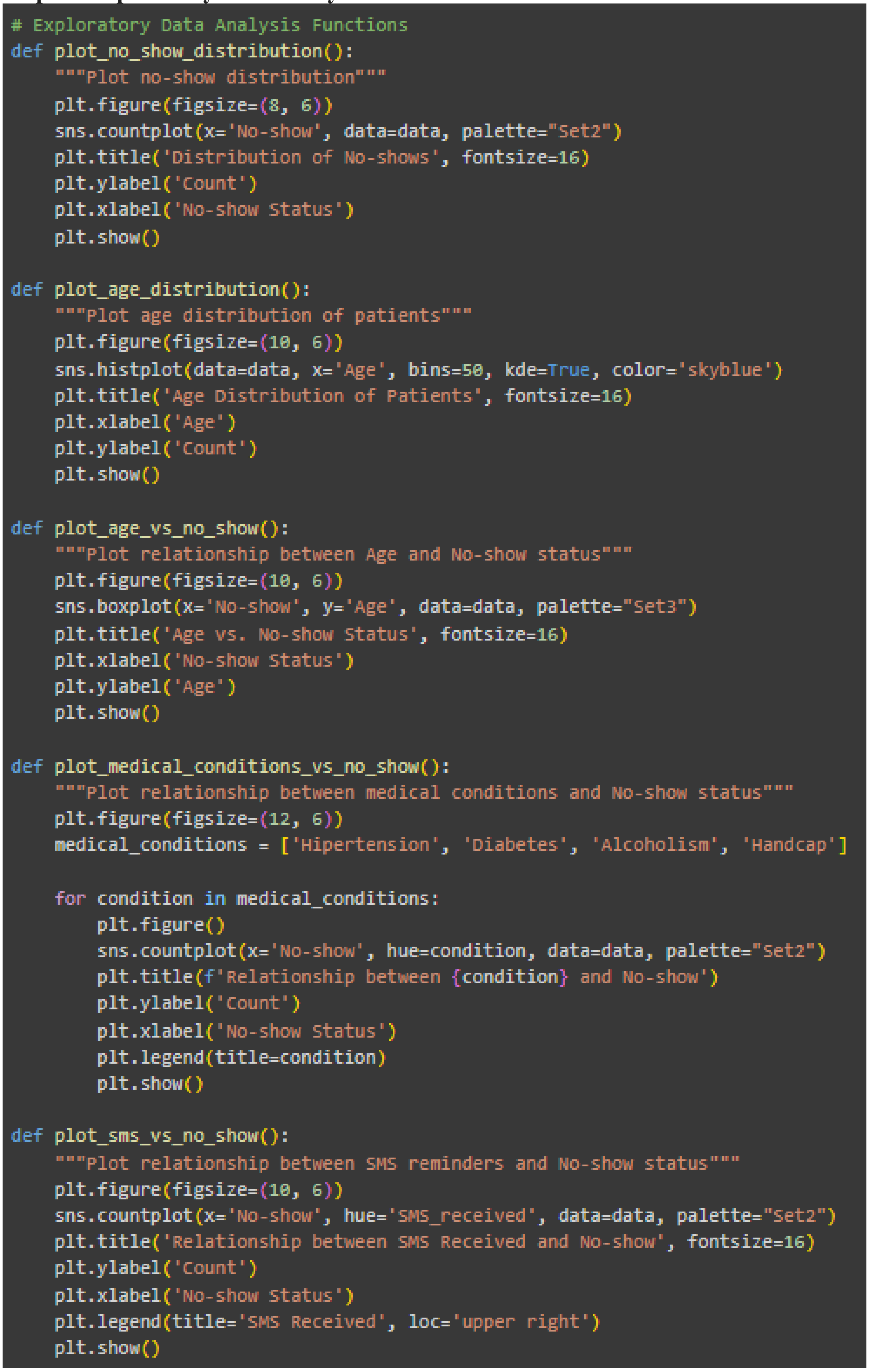
Step 7: Plot the forecast



```
# Step 7: Plot the forecast
fig = model.plot(forecast)
plt.title('Forecast of Daily Appointment Attendance', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Attendance')
plt.show()
```

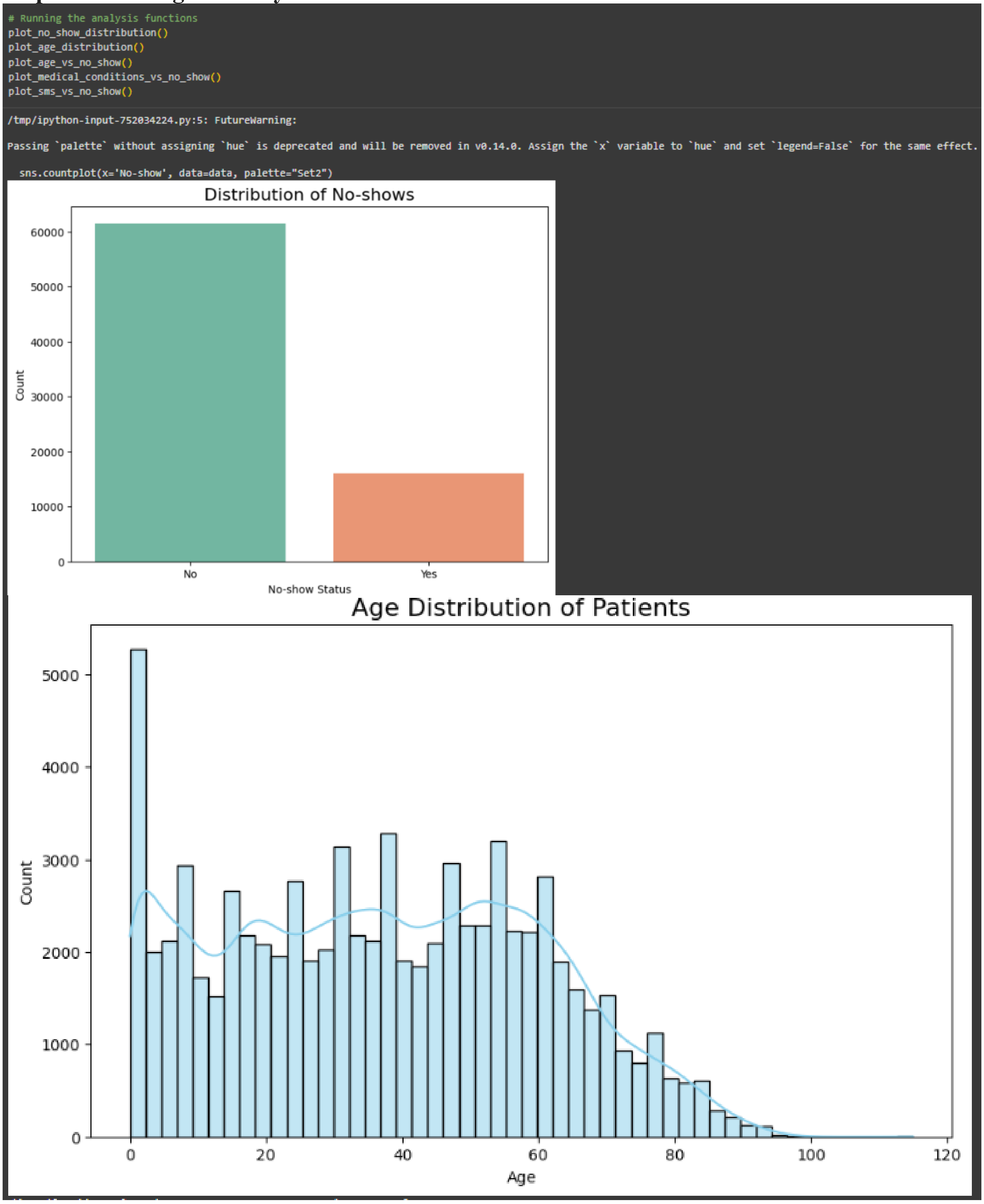


Step 8: Exploratory Data Analysis Functions



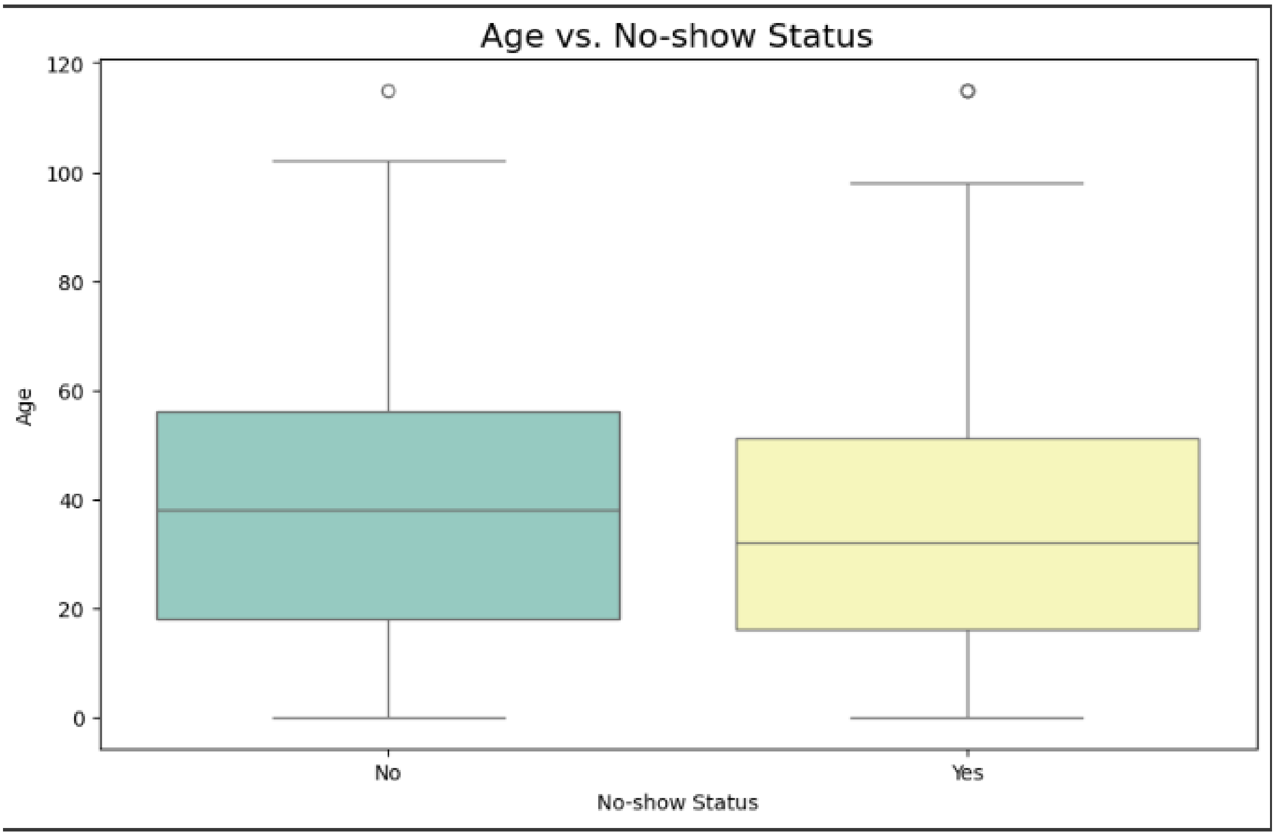
```python
# Exploratory Data Analysis Functions
def plot_no_show_distribution():
    """Plot no-show distribution"""
    plt.figure(figsize=(8, 6))
    sns.countplot(x='No-show', data=data, palette="Set2")
    plt.title('Distribution of No-shows', fontsize=16)
    plt.ylabel('Count')
    plt.xlabel('No-show Status')
    plt.show()

def plot_age_distribution():
    """Plot age distribution of patients"""
    plt.figure(figsize=(10, 6))
    sns.histplot(data=data, x='Age', bins=50, kde=True, color='skyblue')
    plt.title('Age Distribution of Patients', fontsize=16)
    plt.xlabel('Age')
    plt.ylabel('Count')
    plt.show()

def plot_age_vs_no_show():
    """Plot relationship between Age and No-show status"""
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='No-show', y='Age', data=data, palette="Set3")
    plt.title('Age vs. No-show Status', fontsize=16)
    plt.xlabel('No-show Status')
    plt.ylabel('Age')
    plt.show()

def plot_medical_conditions_vs_no_show():
    """Plot relationship between medical conditions and No-show status"""
    plt.figure(figsize=(12, 6))
    medical_conditions = ['Hipertension', 'Diabetes', 'Alcoholism', 'Handcap']

    for condition in medical_conditions:
        plt.figure()
        sns.countplot(x='No-show', hue=condition, data=data, palette="Set2")
        plt.title(f'Relationship between {condition} and No-show')
        plt.ylabel('Count')
        plt.xlabel('No-show Status')
        plt.legend(title=condition)
        plt.show()

def plot_sms_vs_no_show():
    """Plot relationship between SMS reminders and No-show status"""
    plt.figure(figsize=(10, 6))
    sns.countplot(x='No-show', hue='SMS_received', data=data, palette="Set2")
    plt.title('Relationship between SMS Received and No-show', fontsize=16)
    plt.ylabel('Count')
    plt.xlabel('No-show Status')
    plt.legend(title='SMS Received', loc='upper right')
    plt.show()
```

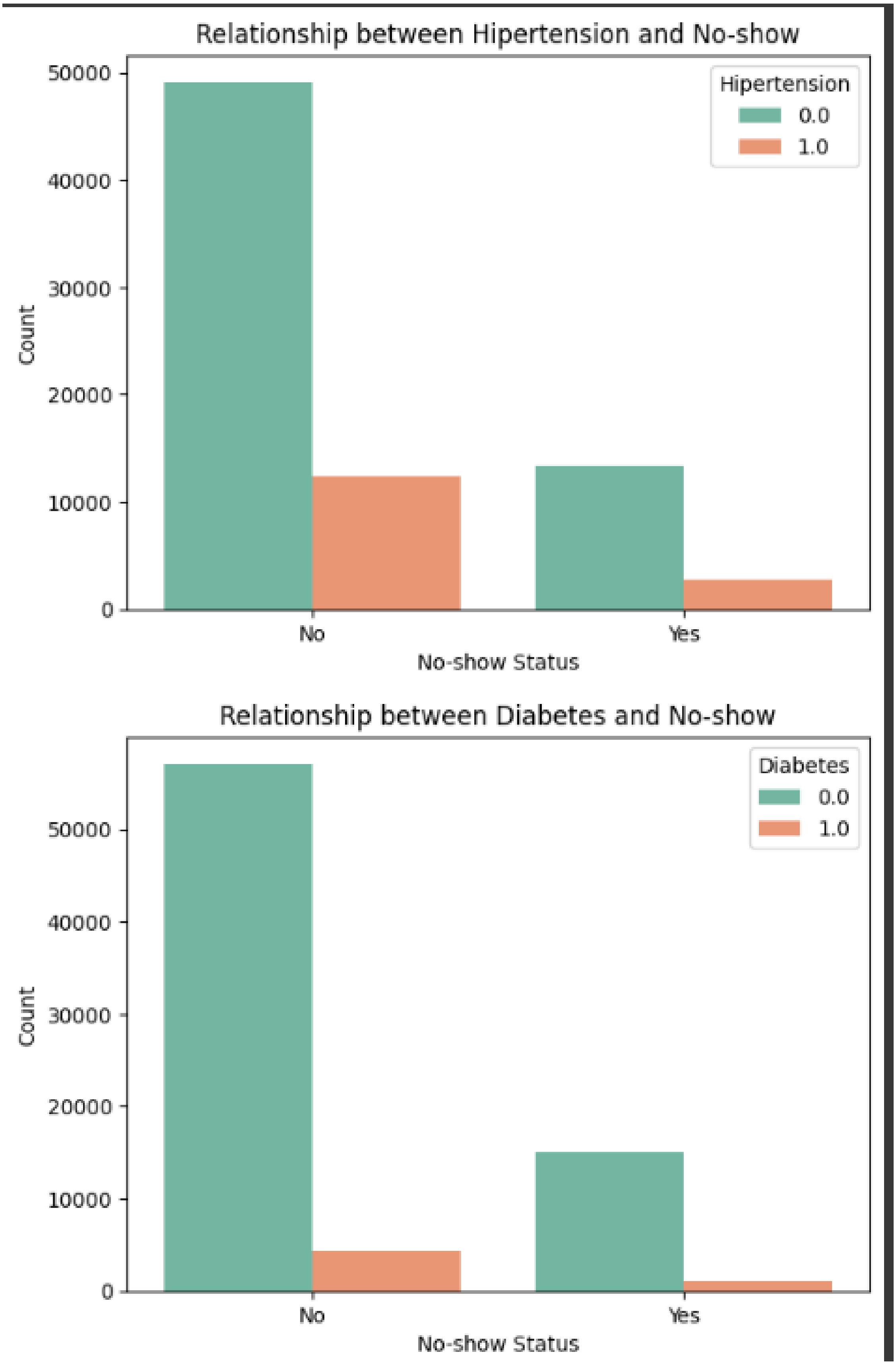
Step 9: Running the analysis functions

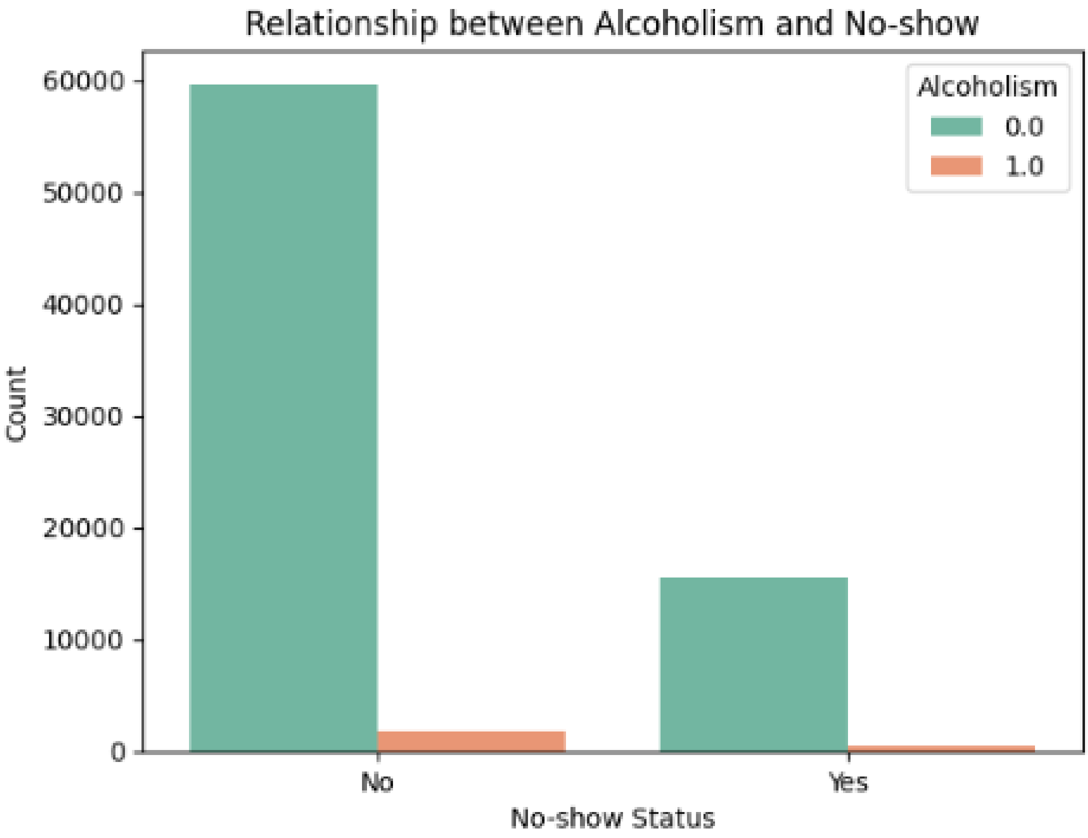


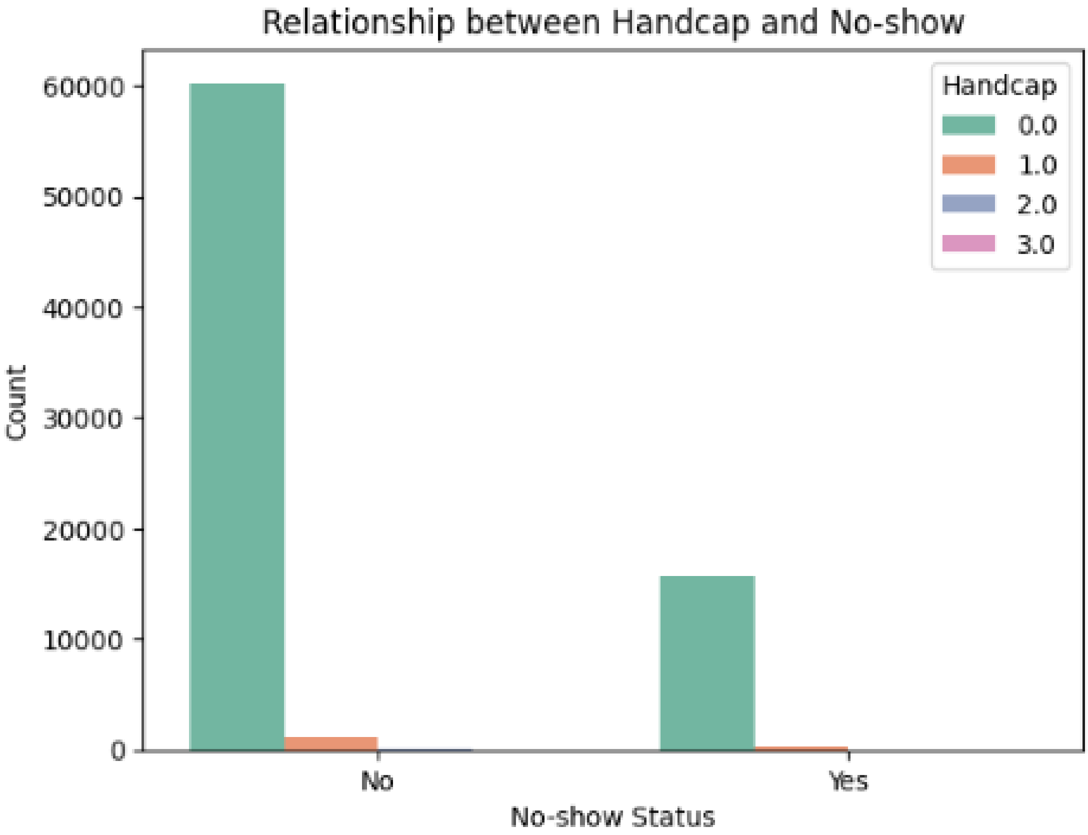
```
# Running the analysis functions
plot_no_show_distribution()
plot_age_distribution()
plot_age_vs_no_show()
plot_medical_conditions_vs_no_show()
plot_sms_vs_no_show()
```

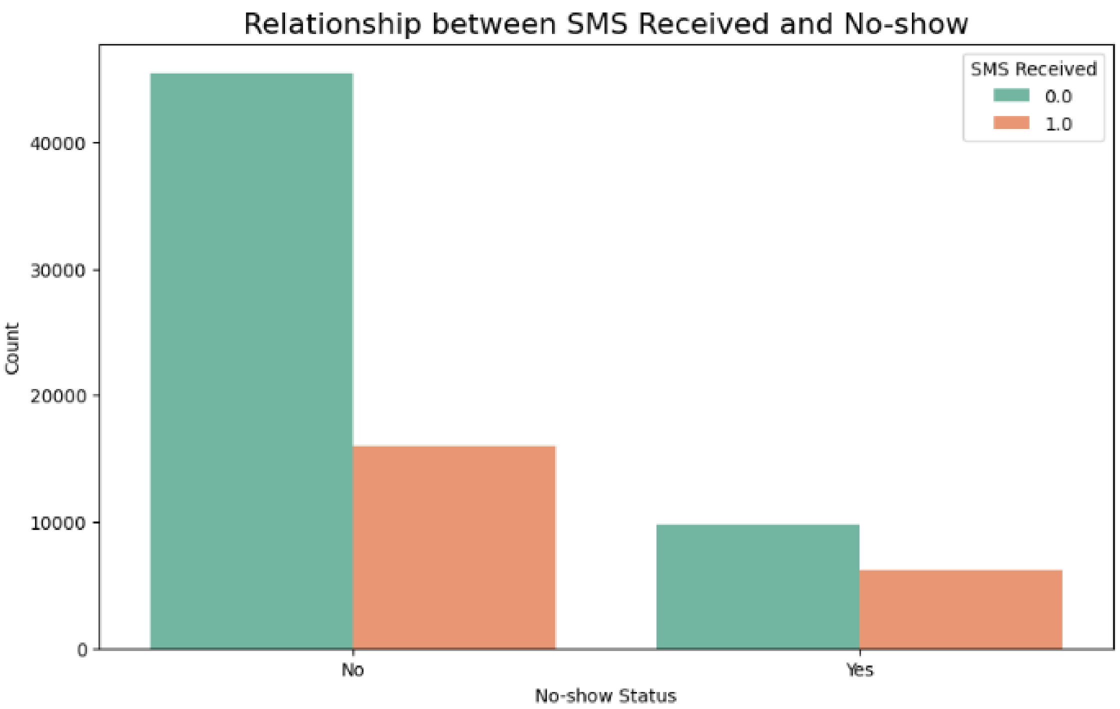
```
/tmp/ipython-input-752034224.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='No-show', data=data, palette="Set2")
```


Age vs. No-show Status
Age
120
100
80
60
40
20
0
No
Yes
No-show Status


Relationship between Hipertension and No-show
Count
50000
40000
30000
20000
10000
0
Hipertension
0.0
1.0
No
Yes
No-show Status
Relationship between Diabetes and No-show
Count
50000
40000
30000
20000
10000
0
Diabetes
0.0
1.0
No
Yes
No-show Status


Relationship between Alcoholism and No-show
Alcoholism
0.0
1.0
Count
60000
50000
40000
30000
20000
10000
0
No
Yes
No-show Status


Relationship between Handcap and No-show
Handcap
0.0
1.0
2.0
3.0
Count
60000
50000
40000
30000
20000
10000
0
No
Yes
No-show Status

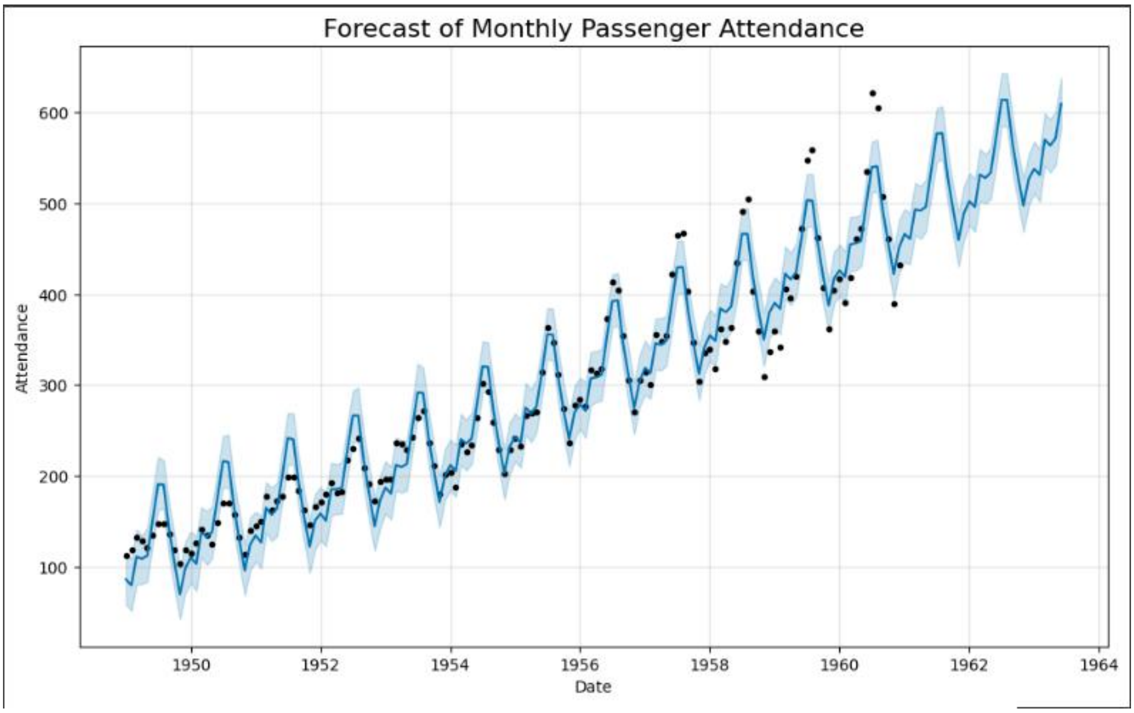


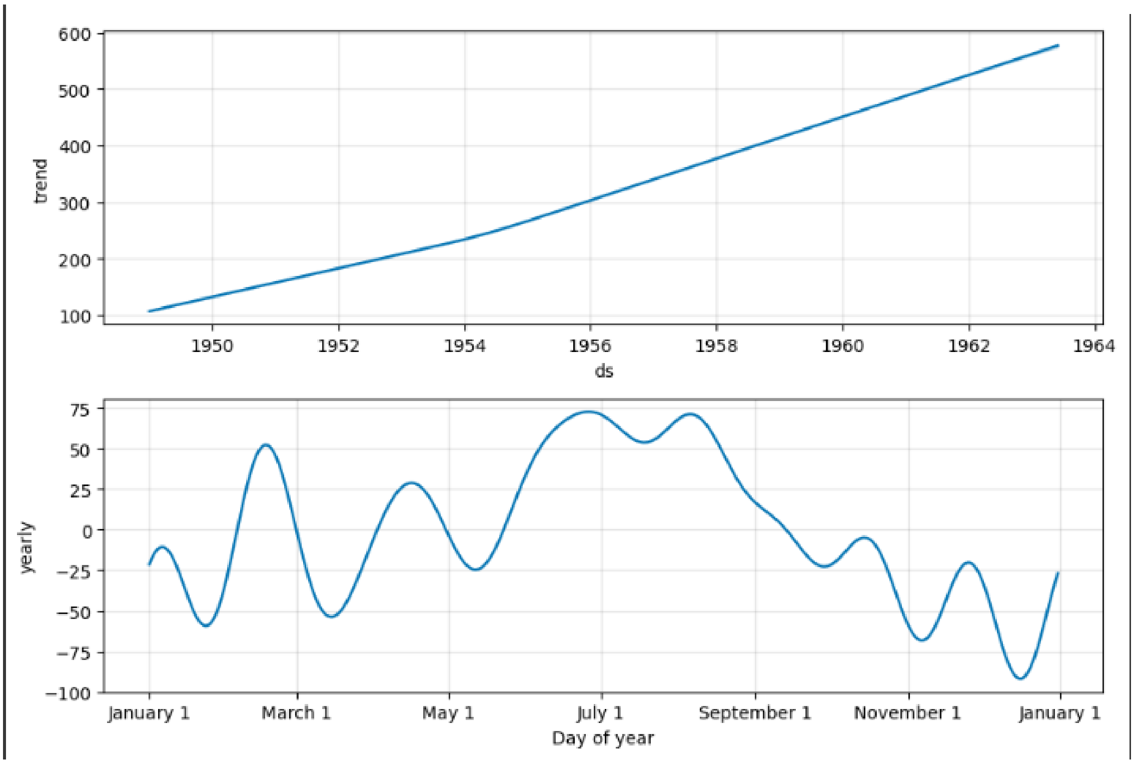
Implementation details:

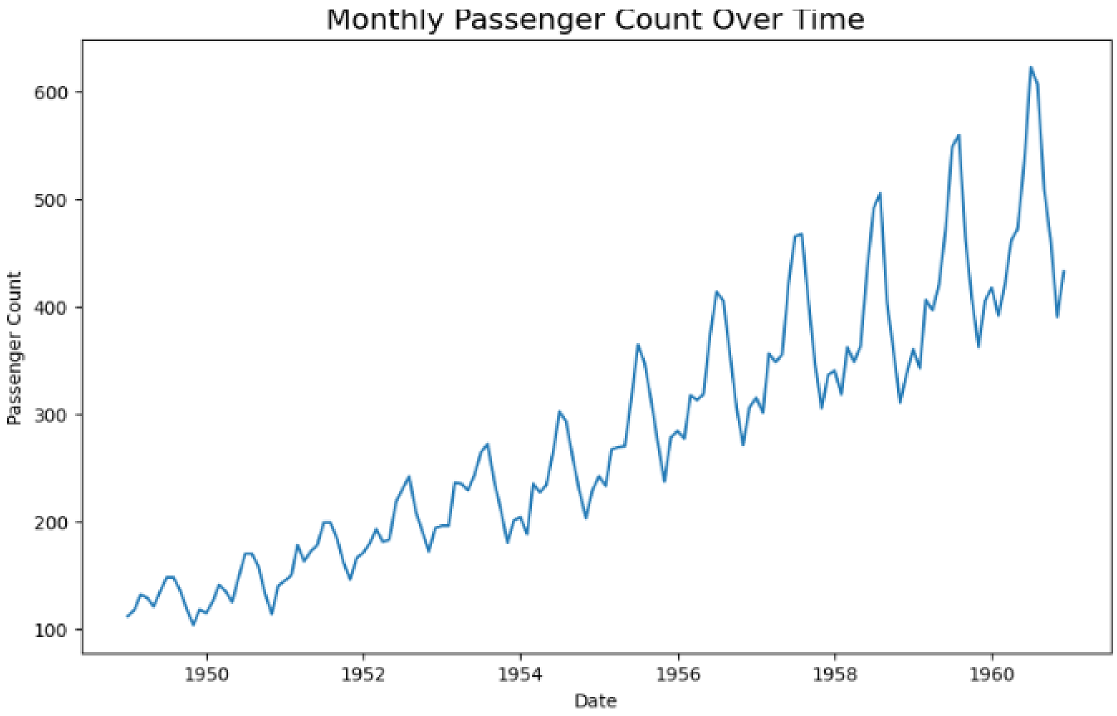
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from prophet import Prophet
import kagglehub
from kagglehub import KaggleDatasetAdapter

file_path = "AirPassengers.csv"

df_local = kagglehub.dataset_load(
  KaggleDatasetAdapter.PANDAS,
  "ashfakyeafi/air-passenger-data-for-time-series-analysis",
  file_path
)

df_local = df_local.rename(columns={'#Passengers': 'Passengers'})

attendance_daily = df_local[['Month', 'Passengers']].copy()
attendance_daily.columns = ['ds', 'y']

attendance_daily['ds'] = pd.to_datetime(attendance_daily['ds']).dt.tz_localize(None)

attendance_daily['y'] = pd.to_numeric(attendance_daily['y'], errors='coerce')

attendance_daily = attendance_daily.dropna()
```

```
model = Prophet()

model.fit(attendance_daily)

future_dates = model.make_future_dataframe(periods=30, freq='MS')

forecast = model.predict(future_dates)

fig = model.plot(forecast)
plt.title('Forecast of Monthly Passenger Attendance', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Attendance')
plt.show()

fig2 = model.plot_components(forecast)
plt.show()

def plot_passenger_time_series():
    """Plot monthly passenger count over time"""
    plt.figure(figsize=(10, 6))
    sns.lineplot(x='ds', y='y', data=attendance_daily)
    plt.title('Monthly Passenger Count Over Time', fontsize=16)
    plt.xlabel('Date')
    plt.ylabel('Passenger Count')
    plt.show()

plot_passenger_time_series()
```

Output:


Forecast of Monthly Passenger Attendance
Attendance
100
200
300
400
500
600
1950
1952
1954
1956
1958
1960
1962
1964
Date


trend
100
200
300
400
500
600
1950
1952
1954
1956
1958
1960
1962
1964
ds
yearly
75
50
25
0
−25
−50
−75
−100
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year



Date: 13/10/2025 **Signature of faculty in-charge**

<u>Post Lab Descriptive Questions:</u>

1. **Explain the components of time series?**
   A time series is made up of four main components: the trend, which shows the overall long-term direction of the data; seasonality, which reflects regular and repeating patterns over fixed intervals like months or quarters; cyclic patterns, which are longer-term fluctuations influenced by economic or business cycles but are irregular in length; and irregular or noise, which represents random, unpredictable variations in the data.

2. **How do you handle seasonality in time series data? What methods or transformations can you apply?**
   Seasonality can be managed by decomposing the series to isolate and remove the seasonal component using techniques like STL decomposition or classical additive/multiplicative methods. Another approach is seasonal differencing, where values from the same season in previous cycles are subtracted to stabilize the data. You can also use seasonal dummy variables in models or apply seasonal ARIMA (SARIMA) models that explicitly account for seasonality.

3. **What are some common metrics for evaluating forecasting models (e.g., MAE, RMSE, and MAPE)?**
   To evaluate forecasting accuracy, metrics like MAE (Mean Absolute Error) measure the average magnitude of errors without considering direction. RMSE

(Root Mean Squared Error) penalizes larger errors more strongly, making it sensitive to outliers. MAPE (Mean Absolute Percentage Error) expresses errors as percentages, which is intuitive but can be misleading when actual values are near zero.