Mar-23 7.57.00 AM 7.30.00 AM Apr-23 7.09.00 AM 7.30.00 AM May-23 7.00.00 AM 7.30.00 AM Jun-23 7.01.00 AM 7.30.00 AM Jul-23 7.13.00 AM 7.30.00 AM Aug-23 7.01.00 AM 7.30.00 AM Sep-23 7.13.00 AM 7.30.00 AM Oct-23 7.40.00 AM 7.30.00 AM Nov-23 7.13.00 AM 7.30.00 AM Dec-23 7.57.00 AM 7.30.00 AM Jan-24 7.09.00 AM 7.30.00 AM Feb-24 7.01.00 AM 7.30.00 AM

Monnths File Timings Expected Time File Timings2 Expected Time2 Jan-23 7.40.00 AM 7.30.00 AM 7.57.00 AM Feb-23 7.13.00 AM 7.30.00 AM 7.09.00 AM 7.13.00 AM 7.57.00 AM 7.13.00 AM 7.31.00 AM 7.49.00 AM 7.13.00 AM 7.09.00 AM 7.01.00 AM 7.13.00 AM 7.13.00 AM 7.09.00 AM 7.01.00 AM Mar-24 7.13.00 AM 7.30.00 AM 7.13.00 AM Apr-24 7.31.00 AM 7.30.00 AM 7.09.00 AM May-24 7.49.00 AM 7.30.00 AM 7.01.00 AM Jun-24 7.13.00 AM 7.30.00 AM 7.31.00 AM Jul-24 7.01.00 AM 7.30.00 AM 7.49.00 AM 7.30.00 AM 7.49.00 AM Aug-24 7.13.00 AM Sep-24 7.31.00 AM 7.30.00 AM 7.13.00 AM

7.01.00 AM 7.30.00 AM

9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 7.01.00 AM File Timings2

Oct-24 7.49.00 AM Nov-24 7.13.00 AM 7.30.00 AM File Timings Expected Time 7.40.00 AM 7.30.00 AM 7.57.00 AM 9.30.00 AM 7.13.00 AM 7.30.00 AM 7.09.00 AM 9.30.00 AM 7.30.00 AM 7.13.00 AM 9.30.00 AM 7.09.00 AM 7.30.00 AM 7.57.00 AM 9.30.00 AM 7.30.00 AM 7.13.00 AM 9.30.00 AM 7.01.00 AM 7.30.00 AM 7.31.00 AM 9.30.00 AM 7.13.00 AM 7.30.00 AM 7.49.00 AM 9.30.00 AM

Monnths Jan-23 Feb-23 Mar-23 7.57.00 AM

Apr-23 May-23 7.00.00 AM Jun-23 Jul-23

7.30.00 AM 7.13.00 AM 9.30.00 AM

7.09.00 AM 7.13.00 AM 7.30.00 AM 9.30.00 AM 7.40.00 AM 7.30.00 AM 7.01.00 AM 9.30.00 AM 7.30.00 AM 7.13.00 AM 9.30.00 AM

9.30.00 AM 9.30.00 AM 9.30.00 AM 7.57.00 AM 7.30.00 AM 7.13.00 AM 9.30.00 AM 7.09.00 AM 7.30.00 AM 7.09.00 AM 9.30.00 AM

7.01.00 AM 7.30.00 AM 7.01.00 AM 9.30.00 AM

9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM 9.30.00 AM **Expected Time2** Aug-23 7.01.00 AM Sep-23 Oct-23 Nov-23 7.13.00 AM Dec-23 Jan-24 Feb-24

Mar-24 7.13.00 AM 7.30.00 AM 7.13.00 AM 9.30.00 AM 7.09.00 AM Apr-24 7.31.00 AM 7.30.00 AM 9.30.00 AM

May-24 7.49.00 AM 7.30.00 AM 7.01.00 AM 9.30.00 AM lun-24 7.13.00 AM 7.30.00 AM 7.31.00 AM 9.30.00 AM

Jul-24 7.30.00 AM 7.49.00 AM 7.01.00 AM 9.30.00 AM 7.30.00 AM 7.49.00 AM Aug-24 7.13.00 AM 9.30.00 AM Sep-24 7.31.00 AM 7.30.00 AM 7.13.00 AM 9.30.00 AM Oct-24 7.49.00 AM 7.30.00 AM 7.01.00 AM 9.30.00 AM Nov-24 7.13.00 AM 7.30.00 AM 7.01.00 AM 9.30.00 AM I have this data in excel I want to make a line graph in jyupter notebok which shows the timing on v-axis and months on x-axis excepted time should show as dotted lines for both files and separate both timings with different colors

To generate a line graph in a Jupyter notebook with the data you provided, we can use libraries like

I'll show you the full code below assuming you've saved the data in a CSV format or have it available in the notebook. For now, I'll simulate this data in the code. You can replace the data part with the actual

Sample DataFrame creation for demonstration (replace this with your actual data loading

'File Timings': ['7:40:00 AM', '7:13:00 AM', '7:57:00 AM', '7:09:00 AM', '7:00:00 AM',

'7:09:00 AM', '7:01:00 AM', '7:31:00 AM', '7:49:00 AM', '7:13:00 AM', '7:01:00 AM',

'7:49:00 AM', '7:13:00 AM'],

'Expected Time': ['7:30:00 AM'] * 23,

'File Timings2': ['7:57:00 AM', '7:09:00 AM', '7:13:00 AM', '7:57:00 AM', '7:13:00

df['File Timings'] = pd.to_datetime(df['File Timings'], format='%I:%M:%S %p').dt.time
df['Expected Time'] = pd.to_datetime(df['Expected Time'], format='%I:%M:%S %p').dt.time
df['File Timings2'] = pd.to_datetime(df['File Timings2'], format='%I:%M:%S %p').dt.time
df['Expected Time2'] = pd.to_datetime(df['Expected Time2'], format='%I:%M:%S %p').dt.time

df['File Timings'] = df['File Timings'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['Expected Time'] = df['Expected Time'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['File Timings2'] = df['File Timings2'].apply(lambda x: x.hour * 3600 + x.minute * 60 + x.second)
df['Expected Time2'] = df['Expected Time2'].apply(lambda x: x.hour * 3600 + x.minute * 60

Plot the timings and expected times
plt.plot(df['Months'], df['File Timings'], label='File Timings', color='blue')
plt.plot(df['Months'], df['File Timings2'], label='File Timings2', color='green')

plt.plot(df['Months'], df['Expected Time'], label='Expected Time', color='blue',
linestyle='dotted')
plt.plot(df['Months'], df['Expected Time2'], label='Expected Time2', color='green',

2. **Time Conversion**: Convert the time strings into `datetime` objects for easier manipulation. 3. **Seconds Calculation**: Convert the times into seconds since midnight for simpler plotting.

4. **Plotting**: Use `matplotlib` to create the plot. Different colors represent different file timings, and

Replace the simulated data with your actual Excel data by loading it using `pd.read_excel()` if needed. Let me know if you need help loading the actual data into the notebook or any other modifications!

To customize the y-axis with specific time intervals, like showing time from `05:00 AM` to `11:00 AM`

Here's how you can modify the code to show time on the y-axis in a human-readable format (like

2. Use a list of times to represent the ticks on the y-axis, and format them accordingly.

Below is the updated version of the code with a variable to control the interval of y-axis labels:

Sample DataFrame creation for demonstration (replace this with your actual data loading

'File Timings': ['7:40:00 AM', '7:13:00 AM', '7:57:00 AM', '7:09:00 AM', '7:00:00 AM',

'Expected Time': ['7:30:00 AM'] * 23, 'File Timings2': ['7:57:00 AM', '7:09:00 AM', '7:13:00 AM', '7:57:00 AM', '7:13:00 AM', '7:31:00 AM', '7:49:00 AM',

df['File Timings'] = pd.to_datetime(df['File Timings'], format='%I:%M:%S %p').dt.time
df['Expected Time'] = pd.to_datetime(df['Expected Time'], format='%I:%M:%S %p').dt.time
df['File Timings2'] = pd.to_datetime(df['File Timings2'], format='%I:%M:%S %p').dt.time
df['Expected Time2'] = pd.to_datetime(df['Expected Time2'], format='%I:%M:%S %p').dt.time

Convert time to total seconds since midnight for plotting $df['File\ Timings'] = df['File\ Timings'].apply(lambda\ x:\ x.hour\ *\ 3600\ +\ x.minute\ *\ 60\ +\ Action 1.5000\ +\ Action 1.500$

df['Expected Time'] = df['Expected Time'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['File Timings2'] = df['File Timings2'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['Expected Time2'] = df['Expected Time2'].apply(lambda x: x.hour * 3600 + x.minute * 60

y_ticks_seconds = [(5 + i * interval_hours) * 3600 for i in range(4)] # Adjust range as

y_tick_labels = ['05:00 AM', '07:00 AM', '09:00 AM', '11:00 AM'] # Corresponding human-

plt.plot(df['Months'], df['File Timings'], label='File Timings', color='blue')
plt.plot(df['Months'], df['File Timings2'], label='File Timings2', color='green')

plt.plot(df['Months'], df['Expected Time'], label='Expected Time', color='blue', linestyle='dotted')
plt.plot(df['Months'], df['Expected Time2'], label='Expected Time2', color='green',
linestyle='dotted')

1. `interval hours` variable: Controls the interval of the y-axis labels. You can change it to any

2. 'y ticks seconds' and 'y tick labels': These are the y-axis tick positions (in seconds since midnight) and the corresponding human-readable labels (e.g., `05:00 AM`, `07:00 AM`, etc.).

This code will display the y-axis as 05:00 AM, 07:00 AM, 09:00 AM, and 11:00 AM, with an interval of 2 hours, and you can change the interval by modifying the `interval_hours` variable. Let me know if

3. `plt.yticks()`: Sets the custom y-ticks with the formatted time labels.

give variable through which i can adjust height and width of graph accordingly

You can use a variable to adjust the height and width of the graph by modifying the `figsize`

I've updated the code to allow you to set the graph's height and width using these variables:

parameter in the `plt.figure()` function. Here's how you can define two variables, `graph_width` and

Sample DataFrame creation for demonstration (replace this with your actual data loading

'Months': ['Jan-23', 'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23', 'Jul-23', 'Aug-'Sep-23', 'Oct-23',

'Sep-24', 'Oct-24', 'Nov-24'], 'File Timings': ['7:40:00 AM', '7:13:00 AM', '7:57:00 AM', '7:09:00 AM', '7:00:00 AM',

df['File Timings'] = pd.to_datetime(df['File Timings'], format='%I:%M:%S %p').dt.time
df['Expected Time'] = pd.to_datetime(df['Expected Time'], format='%I:%M:%S %p').dt.time
df['File Timings2'] = pd.to_datetime(df['File Timings2'], format='%I:%M:%S %p').dt.time
df['Expected Time2'] = pd.to_datetime(df['Expected Time2'], format='%I:%M:%S %p').dt.time

df['File Timings'] = df['File Timings'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['Expected Time'] = df['Expected Time'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['File Timings2'] = df['File Timings2'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['Expected Time2'] = df['Expected Time2'].apply(lambda x: x.hour * 3600 + x.minute * 60

y ticks_seconds = [(5 + i * interval_hours) * 3600 for i in range(4)] # Adjust range as

y_tick_labels = ['05:00 AM', '07:00 AM', '09:00 AM', '11:00 AM'] # Corresponding human-readable labels

plt.figure(figsize=(graph width, graph height)) # Use variables to set graph size

plt.plot(df['Months'], df['File Timings'], label='File Timings', color='blue')
plt.plot(df['Months'], df['File Timings2'], label='File Timings2', color='green')

plt.plot(df['Months'], df['Expected Time'], label='Expected Time', color='blue',

plt.plot(df['Months'], df['Expected Time2'], label='Expected Time2', color='green',

1. `graph_width` and `graph_height`: These variables control the size of the graph. You can change

2. `plt.figure(figsize=(graph_width, graph_height))`: This line uses the `figsize` parameter to set

You can change these values (`graph_width` and `graph_height`) to any size you like, and the graph will

I dont want to hard code y_tick_labels it should start from start from 05:00 AM and end on 11:00

To dynamically create the y-axis tick labels based on the user's desired interval, we can generate the

Sample DataFrame creation for demonstration (replace this with your actual data loading

'File Timings': ['7:40:00 AM', '7:13:00 AM', '7:57:00 AM', '7:09:00 AM', '7:00:00 AM',

'7:49:00 AM', '7:13:00 AM'],

'Expected Time': ['7:30:00 AM'] * 23,

'File Timings2': ['7:57:00 AM', '7:09:00 AM', '7:13:00 AM', '7:57:00 AM', '7:13:00

'7:01:00 AM'],

Convert time strings to datetime objects
df['File Timings'] = pd.to_datetime(df['File Timings'], format='%I:%M:%S %p').dt.time
df['Expected Time'] = pd.to_datetime(df['Expected Time'], format='%I:%M:%S %p').dt.time
df['File Timings2'] = pd.to_datetime(df['File Timings2'], format='%I:%M:%S %p').dt.time
df['Expected Time2'] = pd.to_datetime(df['Expected Time2'], format='%I:%M:%S %p').dt.time

Convert time to total seconds since midnight for plotting $df['File\ Timings'] = df['File\ Timings'].apply(lambda\ x:\ x.hour\ *\ 3600\ +\ x.minute\ *\ 60\ +\ Action{1}{c}$

df['Expected Time'] = df['Expected Time'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['File Timings2'] = df['File Timings2'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['Expected Time2'] = df['Expected Time2'].apply(lambda x: x.hour * 3600 + x.minute * 60

interval_hours = 2 # This variable controls the interval of the y-axis in hours

Generate the y-ticks (in seconds since midnight) and corresponding time labels $y_ticks = []$

Convert current time to seconds since midnight
seconds_since_midnight = current_time.hour * 3600 + current_time.minute * 60
y_ticks.append(seconds_since_midnight)

plt.figure(figsize=(graph_width, graph_height)) # Use variables to set graph size

plt.plot(df['Months'], df['File Timings'], label='File Timings', color='blue')
plt.plot(df['Months'], df['File Timings2'], label='File Timings2', color='green')

Plot expected times as dotted lines
plt.plot(df['Months'], df['Expected Time'], label='Expected Time', color='blue',

plt.plot(df['Months'], df['Expected Time2'], label='Expected Time2', color='green',

1. `start time` and `end time`: You can adjust these to set the range of times shown on the y-axis.

2. `interval_hours`: Controls the time intervals on the y-axis. You can change this value to set how

3. `y_ticks` and `y_tick_labels`: These are generated dynamically based on the start time, end time, and the interval you define. The labels are formatted to appear as times like `05:00 AM`,

4. `timedelta(hours=interval_hours)`: This is used to increment the time by the given interval.

With this approach, you can easily adjust the y-axis time range and interval without hardcoding any

Sample DataFrame creation for demonstration (replace this with your actual data loading step)

'Months': ['Jan-23', 'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23', 'Jul-23', 'Aug-23', 'Sep-23', 'Oct-

'File Timings': ['7:40:00 AM', '7:13:00 AM', '7:57:00 AM', '7:09:00 AM', '7:00:00 AM', '7:01:00 AM',

'7:01:00 AM', '7:13:00 AM', '7:40:00 AM', '7:13:00 AM', '7:57:00 AM', '7:09:00 AM',

'7:13:00 AM', '7:31:00 AM', '7:49:00 AM', '7:13:00 AM', '7:01:00 AM', '7:13:00 AM',

'File Timings2': ['7:57:00 AM', '7:09:00 AM', '7:13:00 AM', '7:57:00 AM', '7:13:00 AM', '7:31:00 AM',

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:13:00 AM', '7:13:00 AM', '7:09:00 AM',

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:31:00 AM', '7:49:00 AM', '7:49:00 AM',

df['File Timings'] = pd.to_datetime(df['File Timings'], format='%I:%M:%S %p').dt.time

df['Expected Time'] = pd.to_datetime(df['Expected Time'], format='%I:%M:%S %p').dt.time df['File Timings2'] = pd.to_datetime(df['File Timings2'], format='%I:%M:%S %p').dt.time

df['Expected Time2'] = pd.to_datetime(df['Expected Time2'], format='%I:%M:%S %p').dt.time

df['File Timings'] = df['File Timings'].apply(lambda x: x.hour * 3600 + x.minute * 60 + x.second) df['Expected Time'] = df['Expected Time'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['File Timings2'] = df['File Timings2'].apply(lambda x: x.hour * 3600 + x.minute * 60 + x.second) df['Expected Time2'] = df['Expected Time2'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

'Nov-23', 'Dec-23', 'Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-24', 'Jul-24', 'Aug-24',

They are defined in the `"%I:%M %p"` format (e.g., `05:00 AM` to `11:00 AM`).

frequently the tick marks appear (e.g., every 1 hour or every 2 hours).

y_tick_labels.append(current_time.strftime('%I:%M %p')) # Convert time back to 'HH:MM
AM/PM' format

Define the start and end time for the y-axis in 'HH:MM' format start_time = "05:00 AM"

Convert start and end times to datetime objects

start_time_dt = datetime.strptime(start_time, '%I:%M %p')
end_time_dt = datetime.strptime(end_time, '%I:%M %p')

Increment the current time by the interval
current_time += timedelta(hours=interval_hours)

'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23', 'Jul-23', 'Aug-

'Dec-23', 'Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-

'7:01:00 AM', '7:13:00 AM', '7:40:00 AM', '7:13:00 AM', '7:57:00 AM',

'7:13:00 AM', '7:31:00 AM', '7:49:00 AM', '7:13:00 AM', '7:01:00 AM',

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:13:00 AM', '7:13:00

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:31:00 AM', '7:49:00

time range programmatically instead of hardcoding the values. Here's how to do it:

2. Define the interval (in hours) to determine the spacing between y-axis labels. 3. Dynamically generate the tick labels and positions based on the interval.

'Oct-24', 'Nov-24']

1. Define a start and end time (e.g., `05:00 AM` to `11:00 AM`).

the graph's dimensions based on the `graph_width` and `graph_height` variables.

'Dec-23', 'Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-

'7:01:00 AM', '7:13:00 AM', '7:40:00 AM', '7:13:00 AM', '7:57:00 AM',

'7:13:00 AM', '7:31:00 AM', '7:49:00 AM', '7:13:00 AM', '7:01:00 AM',

'7:49:00 AM', '7:13:00 AM'], ['7:30:00 AM'] * 23, ['7:57:00 AM', '7:09:00 AM', '7:13:00 AM', '7:57:00 AM', '7:13:00

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:13:00 AM', '7:13:00

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:31:00 AM', '7:49:00

interval hours = 2 # This variable controls the interval of the y-axis

Create list of times from 5:00 AM to 11:00 AM in seconds since midnight

'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23', 'Jul-23', 'Aug-'Dec-23', 'Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-

'7:01:00 AM', '7:13:00 AM', '7:40:00 AM', '7:13:00 AM', '7:57:00 AM',

'7:13:00 AM', '7:31:00 AM', '7:49:00 AM', '7:13:00 AM', '7:01:00 AM',

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:13:00 AM', '7:13:00

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:31:00 AM', '7:49:00

1. We will first convert the times into seconds since midnight for plotting, but we'll display the ticks

I wanted to show the timings on y-axis from 05:00 AM to 11:00 AM with interval of 2

'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23', 'Jul-23', 'Aug-

'Dec-23', 'Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-

'7:01:00 AM', '7:13:00 AM', '7:40:00 AM', '7:13:00 AM', '7:57:00 AM',

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:13:00 AM', '7:13:00

'7:13:00 AM', '7:09:00 AM', '7:01:00 AM', '7:31:00 AM', '7:49:00

`matplotlib` and `pandas` in Python. Below is the code you can use to create the graph:

1. We'll first need to import the data from Excel into a pandas DataFrame.

3. Plot the graph with the timings as lines and expected times as dotted lines.

'Oct-24', 'Nov-24'],

2. Convert the times to a proper format.

DataFrame.

python

step) $data = {$

}

x.second)

x.second)

+ x.second)

Plotting

plt.figure(figsize=(10, 6))

linestyle='dotted')

plt.xticks(rotation=45)

like on y-axis it should show

`11:00 AM`, `09:00 AM`, etc.):

import numpy as np

24', 'Jul-24',

on the y-axis as times.

import pandas as pd
import matplotlib.pyplot as plt

'Months': ['Jan-23', 'Sep-23', 'Oct-23', 'Months . ['Oct-23', 23', 'Sep-23', 'Nov-23', 24'

'7:01:00 AM', '7:13:00 AM'

'7:09:00 AM', '7:01:00 AM'

'7:13:00 AM', '7:31:00 AM'

AM', '7:09:00 AM', '7:01:00 AM'

AM', '7:49:00 AM', '7:13:00 AM'

df = pd.DataFrame(data)

x.second)

x.second)

needed

readable labels

plt.figure(figsize=(10, 6))

Plotting

Formatting

plt.legend() plt.grid(True)

plt.show()

Key Changes:

python

step) $data = {$

+ x.second)

11:00 AM 09:00 AM 07:00 AM 05:00 AM

python

step) $data = {$

Formatting

plt.legend() plt.grid(True)

Explanation:

plt.tight_layout()
plt.show()

Plot expected times as dotted lines

plt.xlabel('Months')
plt.ylabel('Time (Seconds since Midnight)')
plt.title('File Timings vs Expected Time')

dotted lines represent the expected times.

and I can change its interval with any variable

with 2-hour intervals, we need to adjust the `y-ticks` in `matplotlib`.

'Aug-24', 'Sep-24', 'Oct-24', 'Nov-24'],

'7:01:00 AM', '7:01:00 AM'],
'Expected Time2': ['9:30:00 AM'] * 23

Convert time strings to datetime objects

Define the y-axis interval (in hours)

Plot the timings and expected times

Plot expected times as dotted lines

plt.xlabel('Months')
plt.ylabel('Time of Day')
plt.title('File Timings vs Expected Time')

plt.yticks(y_ticks_seconds, y_tick_labels)

interval (e.g., 1 for hourly intervals).

`graph height`, to control the dimensions of the graph.

import pandas as pd
import matplotlib.pyplot as plt

'Nov-23' 'Aug-24'

import numpy as np

23', 'Sep-23',

24', 'Jul-24',

'7:01:00 AM', '7:13:00 AM'

'7:09:00 AM', '7:01:00 AM',

'7:13:00 AM', '7:31:00 AM'

Expected Time': File Timings2':

df = pd.DataFrame(data)

x.second)

x.second)

x.second)

needed

+ x.second)

'7:31:00 AM', '7:49:00 AM'

Convert time strings to datetime objects

Define the y-axis interval (in hours)

Variables to adjust the graph's size graph_width = 12 # Adjust the width here
graph_height = 8 # Adjust the height here

Plot the timings and expected times

Plot expected times as dotted lines

plt.xlabel('Months')
plt.ylabel('Time of Day')
plt.title('File Timings vs Expected Time')

plt.yticks(y_ticks_seconds, y_tick_labels)

their values to adjust the width and height accordingly.

Set y-axis ticks and labels

plt.xticks(rotation=45)

plt.tight_layout()

adjust its size accordingly.

AM and user can adjust the interval

Here's the modified code to achieve this:

import matplotlib.pyplot as plt

'Months': ['Jan-23', 'Sep-23', 'Oct-23',

'7:01:00 AM', '7:13:00 AM'

'7:09:00 AM', '7:01:00 AM'

'7:13:00 AM', '7:31:00 AM'

'File Timings2': ['7:57:00 AM', '7:31:00 AM', '7:49:00 AM'

AM', '7:09:00 AM', '7:01:00 AM

AM', '7:49:00 AM', '7:13:00 AM'

df = pd.DataFrame(data)

'7:01:00 AM',
'Expected Time2': ['9:30:00 AM']

from datetime import datetime, timedelta

'Nov-23'

'Aug-24' 'Sep-24',

import pandas as pd

import numpy as np

24', 'Jul-24',

linestyle='dotted')

linestyle='dotted')

Formatting

plt.legend() plt.grid(True)

plt.show()

Explanation:

(S)

python

step) $data = {$

}

x.second)

x.second)

+ x.second)

 $end_{time} = "11:00 AM"$

y_tick_labels = []

Plotting

linestyle='dotted')

linestyle='dotted')

Formatting
plt.xlabel('Months')
plt.ylabel('Time of Day')

plt.xticks(rotation=45)

plt.legend() plt.grid(True)

plt.show()

Key Changes:

plt.tight_layout()

`07:00 AM`, etc.

import pandas as pd

import numpy as np

data = {

'7:13:00 AM',

'7:01:00 AM',

'7:31:00 AM',

'7:49:00 AM',

'7:01:00 AM',

'7:13:00 AM',

x.second)

x.second)

y_ticks = []

y_tick_labels = []

AM/PM' format

Plotting

Formatting

plt.xlabel('Months') plt.ylabel('Time of Day')

Set y-axis ticks and labels plt.yticks(y_ticks, y_tick_labels)

plt.xticks(rotation=45)

plt.legend() plt.grid(True)

plt.show()

so on

1. Imports

import pandas as pd

specific code.

python

python

python

x.second)

x.second)

x.second)

For example:

python

Here, we define:

python

y_ticks = []
y_tick_labels = []

current_time = start_time_dt

while current_time <= end_time_dt:</pre>

each iteration of the loop.

graph_width = 12 # Width of the graph
graph_height = 8 # Height of the graph

7. Adjustable Graph Size

8. Plotting the Data

linestyle='dotted')

plt.xlabel('Months')
plt.ylabel('Time of Day')

AM`) for the y-axis.

10. Display the Plot

python

Summary:

9. Formatting

python

python

python

+ x.second)

 $data = {$

dates and times.

df = pd.DataFrame(data)

import matplotlib.pyplot as plt

Here we import the necessary libraries:

2. Creating a Sample DataFrame

import numpy as np
from datetime import datetime, timedelta

work with the data in a table-like structure called a DataFrame.

interactive visualizations like the line plot in our case.

python

plt.tight_layout()

current_time = start_time_dt

while current_time <= end_time_dt:

Variables to adjust the graph's size graph_width = 12 # Adjust the width here graph_height = 8 # Adjust the height here

Plot the timings and expected times

Plot expected times as dotted lines

plt.title('File Timings vs Expected Time')

logic used, with beginner-friendly explanations:

start_time = "06:30 AM" end_time = "09:30 AM"

df = pd.DataFrame(data)

}

23',

import matplotlib.pyplot as plt

from datetime import datetime, timedelta

'Sep-24', 'Oct-24', 'Nov-24'],

'7:49:00 AM', '7:13:00 AM'],

'7:01:00 AM', '7:01:00 AM'],

Convert time to total seconds since midnight for plotting

Define the start and end time for the y-axis in 'HH:MM' format

Convert start and end times to datetime objects

start_time_dt = datetime.strptime(start_time, '%I:%M %p') end_time_dt = datetime.strptime(end_time, '%I:%M %p')

Convert current time to seconds since midnight

y_ticks.append(seconds_since_midnight)

Increment the current time by the interval current_time += timedelta(hours=interval_hours)

interval_hours = 0.5 # This variable controls the interval of the y-axis in hours

Generate the y-ticks (in seconds since midnight) and corresponding time labels

seconds_since_midnight = current_time.hour * 3600 + current_time.minute * 60

plt.figure(figsize=(graph_width, graph_height)) # Use variables to set graph size

plt.plot(df['Months'], df['File Timings'], label='File Timings', color='blue') plt.plot(df['Months'], df['File Timings2'], label='File Timings2', color='green')

y_tick_labels.append(current_time.strftime('%I:%M %p')) # Convert time back to 'HH:MM

plt.plot(df['Months'], df['Expected Time'], label='Expected Time', color='blue', linestyle='dotted') plt.plot(df['Months'], df['Expected Time2'], label='Expected Time2', color='green', linestyle='dotted')

explain each part and function used with examples as i am beginner in this why we use that and

Certainly! Let me walk you through each part of the code step-by-step, explaining the functions and

`pandas` (`pd`): A powerful library for data manipulation and analysis. We're using it to store and

• `matplotlib.pyplot` (`plt`): A plotting library for Python. It's used to create static, animated, and

• `numpy` (`np`): A library for numerical operations. While it's imported here, we don't use it in this

• `datetime` and `timedelta`: These are part of Python's standard library. They help us work with

'Months': ['Jan-23', 'Feb-23', 'Mar-23', ... , 'Nov-24'],
'File Timings': ['7:40:00 AM', '7:13:00 AM', '7:57:00 AM', ... , '7:13:00 AM'],
'Expected Time': ['7:30:00 AM'] * 23,
'File Timings2': ['7:57:00 AM', '7:09:00 AM', '7:13:00 AM', ... , '7:01:00 AM'],
'Expected Time2': ['9:30:00 AM'] * 23

• `data`: This is a dictionary where keys (e.g., `'Months'`, `'File Timings'`) represent column

• `pd.DataFrame(data)`: This converts the dictionary `data` into a pandas DataFrame, a tabular structure similar to an Excel sheet or SQL table, which we can easily manipulate and visualize.

df['File Timings'] = pd.to_datetime(df['File Timings'], format='%I:%M:%S %p').dt.time
df['Expected Time'] = pd.to_datetime(df['Expected Time'], format='%I:%M:%S %p').dt.time
df['File Timings2'] = pd.to_datetime(df['File Timings2'], format='%I:%M:%S %p').dt.time
df['Expected Time2'] = pd.to_datetime(df['Expected Time2'], format='%I:%M:%S %p').dt.time

• `pd.to_datetime()`: Converts a string representation of a date/time (like `'7:40:00 AM'`) into a

names, and values are lists representing the data in each column.

`datetime` object. This allows us to perform operations on time data. `format='%I:%M:%S %p'`: This specifies the format of the time string:

• `.dt.time`: Extracts just the time part from the datetime object (e.g., `07:40:00`).

df['File Timings'] = df['File Timings'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['Expected Time'] = df['Expected Time'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['File Timings2'] = df['File Timings2'].apply(lambda x: x.hour * 3600 + x.minute * 60 +

df['Expected Time2'] = df['Expected Time2'].apply(lambda x: x.hour * 3600 + x.minute * 60

...)`: This applies a function to each

○ `(7 * 3600) + (40 * 60) + 0 = 25200 + 2400 = 27600 seconds`.

interval_hours = 0.5 # This variable controls the interval of the y-axis in hours

seconds_since_midnight = current_time.hour * 3600 + current_time.minute * 60
y_ticks.append(seconds_since_midnight)
y_tick_labels.append(current_time.strftime('%I:%M %p')) # 'HH:MM AM/PM'

• `timedelta(hours=interval_hours)`: Adds the interval (e.g., 0.5 hours) to the current time in

• `strptime()`: Converts the time string (like `'06:30 AM'`) into a `datetime` object. `while` loop: This loop generates y-tick values (in seconds since midnight) and their

plt.figure(figsize=(graph_width, graph_height)) # Use variables to set graph size

values of `graph_width` and `graph_height` to customize the size of your plot.

plt.plot(df['Months'], df['File Timings'], label='File Timings', color='blue')
plt.plot(df['Months'], df['File Timings2'], label='File Timings2', color='green')

plt.plot(df['Months'], df['Expected Time'], label='Expected Time', color='blue',
linestyle='dotted') plt.plot(df['Months'], df['Expected Time2'], label='Expected Time2', color='green',

different colors and line styles ('dotted' for expected times).

• `linestyle='dotted'`: Makes the line dotted.

• `color`: Specifies the line color.

plt.title('File Timings vs Expected Time')

plt.legend() # Display the legend plt.grid(True) # Add a grid to the plot

`title()`: Sets the title of the graph.

plt.show() # Displays the plot

titles fit within the figure.

• `label='...'`: Adds a label to each line (used in the legend).

plt.yticks(y_ticks, y_tick_labels) # Set y-axis ticks and labels

• `xlabel()` and `ylabel()`: Set labels for the x-axis and y-axis.

• `grid(True)`: Adds a grid to the graph to improve readability.

plt.tight_layout() # Adjusts the plot so everything fits nicely

• `show()`: Finally, this command renders the plot and shows it in the output.

plots the data on a graph with readable labels (like `'07:00 AM'`).

Printed using <u>Save ChatGPT as PDF</u>, powered by PDFCrowd <u>HTML to PDF API</u>.

• You can adjust the graph size, colors, and labels as needed!

plt.xticks(rotation=45) # Rotate x-axis labels for better readability

• `plt.plot()`: This creates a line plot. We plot both sets of file timings and expected times, using

• `yticks()`: Sets the tick positions (in seconds since midnight) and labels (e.g., `07:00 AM`, `09:30

• `tight_layout()`: Automatically adjusts the padding around the plot to ensure that labels and

• The code visualizes how file timings (actual) compare to expected timings over different months. • It converts time strings into a format that can be plotted (total seconds since midnight) and then

ChatGPT can make mistakes. Check important info.

1/1

• `xticks(rotation=45)`: Rotates the month labels by 45 degrees for better readability. • `legend()`: Displays the legend that shows which line corresponds to which data.

`plt.figure(figsize=(width, height))`: This adjusts the size of the graph. You can change the

• `interval_hours`: This sets the interval between y-ticks. In this case, it's 0.5 hours (or 30 minutes).

calculate the total number of seconds since midnight: • `x.hour * 3600`: Convert hours to seconds. • `x.minute * 60`: Convert minutes to seconds. o `x.second`: Keep the seconds as they are.

• **7:40:00 AM** would be calculated as:

start_time = "06:30 AM" end_time = "09:30 AM"

Defining Start and End Time for Y-Axis

• `start_time`: The start of the y-axis (6:30 AM). • `end_time`: The end of the y-axis (9:30 AM).

start_time_dt = datetime.strptime(start_time, '%I:%M %p') end_time_dt = datetime.strptime(end_time, '%I:%M %p')

current_time += timedelta(hours=interval_hours)

corresponding labels (like `'06:30 AM'`, `'07:00 AM'`, etc.).

Generating Y-Ticks and Labels

3. Converting Time Strings to `datetime` Objects

o `%**I**`: 12-hour format (e.g., 01, 02, ..., 12)

4. Convert Time to Total Seconds Since Midnight

`%M`: Minutes (00-59) `%**s**`: Seconds (00-59)

° '%p': AM/PM

'Expected Time': ['7:30:00 AM'] * 23,

'Expected Time2': ['9:30:00 AM'] * 23

Convert time strings to datetime objects

Set y-axis ticks and labels plt.yticks(y_ticks, y_tick_labels)

current_time = start_time_dt

while current_time <= end_time_dt:</pre>

Variables to adjust the graph's size graph_width = 12 # Adjust the width here
graph_height = 8 # Adjust the height here

Plot the timings and expected times

plt.title('File Timings vs Expected Time')

values. Let me know if you want further adjustments!

'7:01:00 AM', '7:01:00 AM'],
'Expected Time2': ['9:30:00 AM'] * 23

Convert time to total seconds since midnight for plotting

interval hours = 2 # This variable controls the interval of the y-axis

Create list of times from 5:00 AM to 11:00 AM in seconds since midnight

AM', '7:09:00 AM', '7:01:00 AM'

AM', '7:49:00 AM', '7:13:00 AM'

you'd like to adjust anything further!

Set y-axis ticks and labels

plt.xticks(rotation=45)

plt.tight layout()

'7:49:00 AM', '7:13:00 AM'],

1. **Data**: We create a pandas DataFrame to represent the data.

import pandas as pd

24', 'Jul-24',

'Months': ['Jan-23', 23', 'Sep-23', 'Oct-23',

'7:01:00 AM', '7:13:00 AM'

'7:31:00 AM',

df = pd.DataFrame(data)

AM', '7:09:00 AM', '7:01:00 AM'

AM', '7:49:00 AM', '7:13:00 AM'

import matplotlib.pyplot as plt

'Nov-23'

'Aug-24' 'Sep-24',

'7:49:00 AM'

Convert time strings to datetime objects

'7:01:00 AM', '7:01:00 AM'],
'Expected Time2': ['9:30:00 AM'] * 23

Convert time to total seconds since midnight for plotting