1. Data Retrieval (API Call)

- **Cell [50]:**
    - from requests import ..., import json: Imports the requests library to make API calls (HTTP requests) and the json library to handle the JSON data received from the API.
    - url = ..., parameters = ..., headers = ...: Sets up the necessary components for the API call.
        - url: The API endpoint (internet address) to get data from.
        - parameters: Tells the API what data we want (in this case: data for 15 coins, with prices in USD).
        - headers: Includes the "key" (API Key) required to use the API.
    - session = Session(): Creates a persistent connection (session) with the API.
    - try...except...: This block attempts to make the API call (session.get).
    - data = json.loads(response.text): If the API call is successful, it takes the response (JSON text) and parses it into a Python dictionary named data.

2. Data Preprocessing

- **Cell [51]:**
    - type(data): Checks the data type of the data variable. (Output: dict - meaning it's a dictionary).
- **Cell [52]:**
    - import pandas as pd: Imports the 'pandas' library, which is the main tool for data analysis in Python.
    - pd.set_option(...): Changes pandas settings to ensure it displays all columns and rows when printing a DataFrame, rather than truncating them with "...".
- **Cell [54]:**
    - df = pd.json_normalize(data['data']): This is a key step. It takes the nested JSON data (located inside the 'data' key of the data dictionary) and flattens it into a simple table, known as a DataFrame. df becomes our main data table.

- df['timestamp'] = pd.to_datetime('now'): Adds a new column named timestamp to the DataFrame and fills it with the exact time the data was fetched.
- df.head(): Displays the first 5 rows of the DataFrame.

- **Cell [55]:**
  - df.shape: Shows the dimensions of the DataFrame (rows, columns). The output (15, 37) means it has data for 15 coins and 37 columns of information for each.

3. Data Saving and Automation

- **Cell [56]:**
  - import os: Imports the os (Operating System) library, used here to check for files.
  - def api_runner(): ...: Defines a function named api_runner. This function bundles the API call (from Cell 50) and the data normalization (from Cell 54) into one reusable command.
  - if not os.path.isfile(...): ... else: ...: This logic saves the data to a API.csv file.
    - os.path.isfile(...): Checks if a file named API.csv already exists.
    - **If not (if not...):** It creates a new API.csv file (df.to_csv(...)) and saves the data, including the column headers.
    - **If it exists (else...):** It appends (mode='a') the new data to the bottom of the existing file *without* adding the headers again (header=False).
  - df = api_runner(): Runs the api_runner function one time to update the df variable with fresh data.

- **Cell [57]:**
  - """ ... """: The triple quotes (""") mark this entire cell as a comment.
  - from time import sleep: This line (if it were active) would import the sleep function.
  - for i in range(5): ... sleep(60): This commented-out code shows how you would automate the process. It's a for loop that would run the api_runner() 5 times, waiting for one minute (sleep(60)) between each run.
  - **Note:** Because this cell is a comment, this automation code did not actually run.

4. Data Analysis

- **Cell [58]:**
    - df = pd.read_csv(...): Reads the data from the saved API.csv file back into the df DataFrame.
- **Cell [59]:**
    - pd.set_option('display.float_format', ...): Changes a pandas setting to format all float numbers (decimals) to 5 decimal places for cleaner display.
- **Cell [61]:**
    - df3 = df.groupby('name', sort=False)[...].mean(): This is an important analysis step.
        - df.groupby('name'): Groups all the rows by the coin's name (e.g., all "Bitcoin" entries together, all "Ethereum" entries together).
        - [...]: Selects only the columns that show percentage changes in price (1h, 24h, 7d, etc.).
        - .mean(): Calculates the average (mean) of those columns for each coin.
- **Cell [62]:**
    - df4 = df3.stack(): Reshapes the data. It takes the columns (1h, 24h, 7d...) and "stacks" them into rows. This creates a pandas Series.
- **Cell [64] - [68]:**
    - df5 = df4.to_frame(name='values'): Converts the Series (df4) back into a DataFrame (df5).
    - df6 = df5.reset_index(): Converts the stacked index (like 'Bitcoin', '1h') into regular columns named name and level_1.
- **Cell [69] - [70]:**
    - df7 = df6.rename(columns=...): Renames the level_1 column to percent_change.
    - df7['percent_change'] = df7['percent_change'].replace(...): Cleans the data by replacing the long, technical names (e.g., quote.USD.percent_change_1h) with short, simple names (e.g., 1h).

5. Data Visualization

- **Cell [71]:**

    - import seaborn as sns, import matplotlib.pyplot as plt: Imports the seaborn and matplotlib libraries, which are used for creating plots and charts.

- **Cell [72]:**
    - sns.catplot(..., kind='point'): Creates a "point plot".
        - x='percent_change': Sets the x-axis (1h, 24h, 7d...).
        - y='values': Sets the y-axis (the average % change).
        - hue='name': Creates a different colored line for each unique coin name.
    - **Purpose:** To visually compare the average price change trends of all coins over different time periods in a single chart.
- **Cell [73]:**
    - df10 = df[['name','quote.USD.price','Timestamp']]: Creates a new, simple DataFrame df10 by selecting only the name, price, and Timestamp columns from the main df.
    - df10 = df10.query("name == 'Bitcoin'"): Filters this new DataFrame to keep *only* the rows where the name is "Bitcoin".
- **Cell [74]:**
    - sns.lineplot(x='Timestamp', y='quote.USD.price', data=df10): Creates a line plot.
    - **Purpose:** To show how the price of "Bitcoin" has changed over time.
    - plt.xticks(rotation=90): Rotates the labels on the x-axis (the timestamps) by 90 degrees to make them easier to read.

Python Concepts and Libraries Used

As you requested, here is a separate explanation of the Python tools used in this notebook.

Libraries

1. **requests:**
    - **Use:** Used for sending HTTP requests over the internet (e.g., to call the API).
    - **Key Features:**
        - Session: Creates a persistent connection for making multiple requests.

        - session.get(url, params, headers): Sends a GET request to the url to fetch data, passing along the params (what data you want) and headers (your API key).

2. **json:**
    - **Use:** For working with the JSON (JavaScript Object Notation) data format, which is the standard for APIs.
    - **Key Features:**
        - json.loads(text): ("load string") Parses a JSON text string from the API and converts it into a Python dictionary or list.
3. **pandas (pd):**
    - **Use:** The most powerful and common library for data analysis and manipulation.
    - **Key Features:**
        - DataFrame: The primary data structure, like a table with rows and columns.
        - pd.json_normalize(): Converts complex, nested JSON/dictionaries into a flat DataFrame.
        - pd.to_datetime(): Converts text strings of dates/times into proper datetime objects.
        - df.head(): Shows the first 5 rows.
        - df.shape: Shows the number of rows and columns.
        - df.to_csv('file.csv'): Saves a DataFrame to a CSV file.
        - df.read_csv('file.csv'): Reads a CSV file into a DataFrame.
        - df.groupby('column'): Groups data based on the values in a column.
        - .mean(): Calculates the average.
        - .stack(), .reset_index(): Used for reshaping the DataFrame.
        - .rename(), .replace(): Used for cleaning column names and values.
        - df.query('...'): An easy way to filter the DataFrame (e.g., name == 'Bitcoin').

4. **os:**
    - **Use:** Provides a way to use operating system-dependent functionality.
    - **Key Features:**
        - os.path.isfile('file.csv'): Checks if a file with this name already exists (returns True or False).
5. **time:**
    - **Use:** For time-related tasks.
    - **Key Features:**
        - sleep(seconds): Pauses the execution of the code for a specified number of seconds.
6. **matplotlib.pyplot (plt) and seaborn (sns):**
    - **Use:** For data visualization (plotting). seaborn is built on top of matplotlib and provides a simpler interface for creating attractive statistical plots.
    - **Key Features:**
        - sns.catplot(): A "categorical" plot, used here with kind='point'.
        - sns.lineplot(): A line chart, perfect for showing trends over time.
        - plt.figure(figsize=...): Sets the size (width, height) of the plot.
        - plt.xticks(rotation=...): Rotates the labels on the x-axis.

General Python Concepts

- **def ... (): (Function Definition):** Creates a reusable block of code that can be "called" by its name (e.g., api_runner()).
- **if...else... (Conditional Logic):** Allows your code to make decisions. It runs one block of code if a condition is true, and another block else if it is false.
- **try...except... (Error Handling):** A safety measure. It "tries" to run code that might fail (like an API call). If an error occurs, it "excepts" the error and runs a different block of code instead of crashing the program.
- **for ... in range()... (For Loop):** A way to repeat an action a specific number of times (e.g., range(5) repeats the loop 5 times).