CPE 4040: Homework #5

1. Write your answer in the cell provided under each question.
2. **You must write comments to explain your thoughts and earn full credit**.
3. **Show your execution result**.
4. Do your own work. **Do not copy-and-paste other people's (or Generative AI's) codes.**

Submission:

- **Submit this notebook file and the pdf version**

In [278…
```
import pandas as pd
import numpy as np
```

Q1: Revisiting the Apple Store (25 Points)

It is time to revisit the AppleStore dataset that we worked on earlier in the semester.

Use what you learn from pandas, answer the following questions.

1. Import the dataset AppleStore.csv and assign a DataFrame called 'df_apps'.

How many rows and columns in the dataset? What are the row index and column labels?

Print the first 5 lines of the dataset.

In [279…
```
df_apps = pd.read_csv('AppleStore.csv')
```

In [280…
```
df_apps.head()
```

Out[280…

	id	track_name	size_bytes	currency	price	rating_count_tot	rating_count_ver
0	284882215	Facebook	389879808	USD	0.0	2974676	212
1	389801252	Instagram	113954816	USD	0.0	2161558	1289
2	529479190	Clash of Clans	116476928	USD	0.0	2130805	579
3	420009108	Temple Run	65921024	USD	0.0	1724546	3842
4	284035177	Pandora - Music & Radio	130242560	USD	0.0	1126879	3594

In [281…

```
print(f"Number of Columns: {df_apps.shape[1]}. Number of Rows: {df_apps.shape[0]}")
```

```
Number of Columns: 16. Number of Rows: 7197
```

2. Counting genres: how many different genres in "prime_genre"?

How many titles are in each genre? (Hint: use value_counts method)

In [282…

```
unique_genres = df_apps["prime_genre"].nunique()
unique_genres
```

Out[282… 23

In [283…

```
genre_counts = df_apps["prime_genre"].value_counts()
genre_counts
```

Out[283…
```
prime_genre
Games                3862
Entertainment         535
Education             453
Photo & Video         349
Utilities             248
Health & Fitness      180
Productivity          178
Social Networking     167
Lifestyle             144
Music                 138
Shopping              122
Sports                114
Book                  112
Finance               104
Travel                 81
News                   75
Weather                72
Reference              64
Food & Drink           63
Business               57
Navigation             46
Medical                23
Catalogs               10
Name: count, dtype: int64
```

3. Content rating: how many different categories of 'content_rating'?

What percentage of titles are in each category?

In [284…
```
unique_content_ratings = df_apps["cont_rating"].nunique()
unique_content_ratings
```

Out[284…
```
4
```

In [285…
```
content_rating_percentages = df_apps["cont_rating"].value_counts(normalize=True) *
content_rating_percentages
```

Out[285…
```
cont_rating
4+     61.595109
12+    16.048353
9+     13.714048
17+     8.642490
Name: proportion, dtype: float64
```

4. Solve a simiar problem as in Homework #3: Use GroupBy to find the distribution of rating in each genre.

a. How many apps in 'Games' are for age 17 and above?

b. How many apps in 'Education' are for kids 9 years and older?

In [286…
```
games_17_plus = df_apps[(df_apps['prime_genre'] == 'Games') & (df_apps['cont_rating

games_17_plus_count = games_17_plus.groupby(['prime_genre', 'cont_rating']).size().

print(games_17_plus_count)
```

```
  prime_genre cont_rating  count
0       Games         17+    177
```

In [287…
```
education_9_plus = df_apps[(df_apps['prime_genre'] == 'Education') & (df_apps['cont

education_9_plus_count = education_9_plus.shape[0]

print(education_9_plus_count)
```

```
21
```

5. How many apps are free? What is the title of the most expensive app?

In [288…
```
# Calculate the number of free apps
free_apps_count = df_apps[df_apps["price"] == 0.0].shape[0]

# Find the title of the most expensive app
most_expensive_app = df_apps.loc[df_apps["price"].idxmax()]

print(free_apps_count)
print(f"The most expensive app is: {most_expensive_app['track_name']} with a price
```

```
4056
The most expensive app is: LAMP Words For Life with a price of $299.99
```

Q2: Practicing GroupBy and Data Aggregation (25 Points)

1. Import the dataset from the URL. This is a set of statistics of age, gender and occupation.

How many different occupations are in the dataset?

In [289…
```
users = pd.read_table('https://raw.githubusercontent.com/justmarkham/DAT8/master/da
                      sep='|', index_col='user_id')
```

In [290…
```
unique_occupations = users["occupation"].nunique()
print(unique_occupations)
```

```
21
```

2.1 There is a column label, zip_code, which will not be used in this exercise.

Please remove the column from the DataFrame

In [291…
```
users = users.drop(columns=['zip_code'])
print(users.head())
```

```
         age gender  occupation
user_id
1         24      M  technician
2         53      F       other
3         23      M      writer
4         24      M  technician
5         33      F       other
```

2.2 As you may have noticed, there is an occupation called "retired".

Please remove the users whose occupation is "retired" from the DataFrame.

In [292…
```
users_filtered = users[users["occupation"] != "retired"]
print(users_filtered.head())
```

```
         age gender  occupation
user_id
1         24      M  technician
2         53      F       other
3         23      M      writer
4         24      M  technician
5         33      F       other
```

3. Calculate the average age for each occupation. What are the top 5 youngest occupation groups?

Hint: use sort_values() method.

In [293…
```
average_age_by_occupation = users_filtered.groupby("occupation")["age"].mean().sort
youngest_occupations = average_age_by_occupation.head(5)

print("Top 5 youngest occupation groups:")
print(youngest_occupations)
```

```
Top 5 youngest occupation groups:
occupation
student          22.081633
none             26.555556
entertainment    29.222222
artist           31.392857
homemaker        32.571429
Name: age, dtype: float64
```

4. Calculate the female ratio for each occupation and sort it in

ascending order.

In [294…
```
users_filtered["gender"] = users_filtered["gender"].str.strip()

occupation_counts = users_filtered.groupby("occupation")["gender"].value_counts().u
occupation_counts["total"] = occupation_counts.sum(axis=1)
occupation_counts["female_ratio"] = occupation_counts["F"] / occupation_counts["tot

sorted_female_ratio = occupation_counts["female_ratio"].sort_values()

print("Female ratio for each occupation (sorted in ascending order):")
print(sorted_female_ratio)
```

```
Female ratio for each occupation (sorted in ascending order):
occupation
doctor           0.000000
engineer         0.029851
technician       0.037037
programmer       0.090909
executive        0.093750
scientist        0.096774
entertainment    0.111111
lawyer           0.166667
salesman         0.250000
educator         0.273684
student          0.306122
other            0.342857
marketing        0.384615
writer           0.422222
none             0.444444
administrator    0.455696
artist           0.464286
librarian        0.568627
healthcare       0.687500
homemaker        0.857143
Name: female_ratio, dtype: float64
```

```
C:\Users\rbrin\AppData\Local\Temp\ipykernel_24304\2908118433.py:1: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  users_filtered["gender"] = users_filtered["gender"].str.strip()
```

5. For each occupation, calculate the minimum and maximum ages.

In [295…
```
age_stats_by_occupation = users_filtered.groupby("occupation")["age"].agg(['min', '

print("Minimum and maximum ages for each occupation:")
print(age_stats_by_occupation)
```

```
Minimum and maximum ages for each occupation:
               min  max
occupation
administrator   21   70
artist          19   48
doctor          28   64
educator        23   63
engineer        22   70
entertainment   15   50
executive       22   69
healthcare      22   62
homemaker       20   50
lawyer          21   53
librarian       23   69
marketing       24   55
none            11   55
other           13   64
programmer      20   63
salesman        18   66
scientist       23   55
student          7   42
technician      21   55
writer          18   60
```

6. For each combination of occupation and gender, calculate the mean age.

Hint: This will require you to pass two column labels to Groupby.

In [296…

```
mean_age_by_occupation_gender = users_filtered.groupby(['occupation', 'gender'])['a
print("Mean age for each combination of occupation and gender:")
print(mean_age_by_occupation_gender)
```

```
Mean age for each combination of occupation and gender:
       occupation gender        age
0   administrator      F  40.638889
1   administrator      M  37.162791
2          artist      F  30.307692
3          artist      M  32.333333
4          doctor      M  43.571429
5        educator      F  39.115385
6        educator      M  43.101449
7        engineer      F  29.500000
8        engineer      M  36.600000
9   entertainment      F  31.000000
10  entertainment      M  29.000000
11     executive      F  44.000000
12     executive      M  38.172414
13    healthcare      F  39.818182
14    healthcare      M  45.400000
15     homemaker      F  34.166667
16     homemaker      M  23.000000
17        lawyer      F  39.500000
18        lawyer      M  36.200000
19     librarian      F  40.000000
20     librarian      M  40.000000
21     marketing      F  37.200000
22     marketing      M  37.875000
23          none      F  36.500000
24          none      M  18.600000
25         other      F  35.472222
26         other      M  34.028986
27    programmer      F  32.166667
28    programmer      M  33.216667
29      salesman      F  27.000000
30      salesman      M  38.555556
31     scientist      F  28.333333
32     scientist      M  36.321429
33       student      F  20.750000
34       student      M  22.669118
35    technician      F  38.000000
36    technician      M  32.961538
37        writer      F  37.631579
38        writer      M  35.346154
```

Q3: Historical Stock Prices (50 Points)

In this exercise, you will download market data of a stock of your choice (or an index, such as Dow Jones or S&P500) and perform some exploratory data analysis (EDA).

You will follow the step-by-step guide in the companion Jupyter Notebook file, Homework 6 - Historical Stock data, to download the stock price data and clean up the data.

Please make sure to include the CSV file for the stock price data when submitting the homework.

NOTE: You cannot pick AAPL since it is used in the example.

To summarize, you will work on:

1. Data cleaning
2. Simple Exploratory Data Analysis
3. 52-Week Price Range
4. Yearly Performance Summary
5. Yearly Investment Results

Import the stock historical data of your choice.

In [297…
```
nvidia_df = pd.read_csv("nvidia_stock.csv")
```

Display the first 5 rows of data

In [298…
```
nvidia_df.head()
```

Out[298…

	Date	Close/Last	Volume	Open	High	Low
0	11/12/2024	$148.29	198634700	$146.78	$149.65	$146.01
1	11/11/2024	$145.26	182325600	$148.68	$148.85	$143.57
2	11/08/2024	$147.63	175665800	$148.77	$149.77	$146.26
3	11/07/2024	$148.88	207323300	$146.39	$148.93	$146.17
4	11/06/2024	$145.61	242043900	$142.96	$146.49	$141.96

1. Data Cleaning Before Data Analysis

Please follow the step-by-step guide below to clean up the data. You can refer to Step 4 in the Jupyter Notebook file, Homework 6 - Historical Stock data, to learn how to perform the task.

Step 1: Remove the dollar signs in the price columns and convert them to float

In [299…
```
price_columns = ["Close/Last", "Open", "High", "Low"]
for col in price_columns:
    nvidia_df[col] = nvidia_df[col].replace('[\$,]', '', regex=True).astype(float)

nvidia_df.head()
```

Out[299…

	Date	Close/Last	Volume	Open	High	Low
0	11/12/2024	148.29	198634700	146.78	149.65	146.01
1	11/11/2024	145.26	182325600	148.68	148.85	143.57
2	11/08/2024	147.63	175665800	148.77	149.77	146.26
3	11/07/2024	148.88	207323300	146.39	148.93	146.17
4	11/06/2024	145.61	242043900	142.96	146.49	141.96

Step 2: The Date column needs to be converted to datetime format first

In [300…

```
nvidia_df['Date'] = pd.to_datetime(nvidia_df['Date'], format='%m/%d/%Y')
nvidia_df.head()
```

Out[300…

	Date	Close/Last	Volume	Open	High	Low
0	2024-11-12	148.29	198634700	146.78	149.65	146.01
1	2024-11-11	145.26	182325600	148.68	148.85	143.57
2	2024-11-08	147.63	175665800	148.77	149.77	146.26
3	2024-11-07	148.88	207323300	146.39	148.93	146.17
4	2024-11-06	145.61	242043900	142.96	146.49	141.96

Rename the column 'Close/Last' to 'Close'

In [301…

```
nvidia_df.rename(columns = {'Close/Last':'Close'}, inplace=True)
nvidia_df.head()
```

Out[301…

	Date	Close	Volume	Open	High	Low
0	2024-11-12	148.29	198634700	146.78	149.65	146.01
1	2024-11-11	145.26	182325600	148.68	148.85	143.57
2	2024-11-08	147.63	175665800	148.77	149.77	146.26
3	2024-11-07	148.88	207323300	146.39	148.93	146.17
4	2024-11-06	145.61	242043900	142.96	146.49	141.96

2. Simple Exploratory Data Analysis

Examine the column data statistics using df.describe()

In [302...
```
nvidia_df.describe()
```

Out[302...

	Date	Close	Volume	Open	High	Lo
count	1258	1258.000000	1.258000e+03	1258.000000	1258.000000	1258.00000
mean	2022-05-14 17:26:13.926868224	34.971279	4.505747e+08	34.960976	35.616893	34.25686
min	2019-11-13 00:00:00	4.910000	9.788352e+07	5.002400	5.195500	4.51700
25%	2021-02-13 00:00:00	13.360625	3.156110e+08	13.381575	13.596925	13.11810
50%	2022-05-12 12:00:00	20.185500	4.254000e+08	20.021050	20.458650	19.79110
75%	2023-08-14 18:00:00	43.514250	5.474608e+08	43.575250	43.984000	42.68675
max	2024-11-12 00:00:00	148.880000	1.543911e+09	148.770000	149.770000	146.26000
std	NaN	34.815174	1.863226e+08	34.839766	35.466812	34.10708

Write a script to find the largest trading volumn in your dataset

In [303...
```
largest_volume = nvidia_df.loc[nvidia_df["Volume"].idxmax()]
largest_volume
```

Out[303...
```
Date      2023-05-25 00:00:00
Close                   37.98
Volume             1543911000
Open                   38.523
High                    39.48
Low                    36.635
Name: 369, dtype: object
```

Write a script to find the all-time high in closing price

In [304...
```
all_time_high = nvidia_df["Close"].max()
all_time_high
```

Out[304...
```
148.88
```

Which date(s) did the all-time high in closing price occur?

In [305…
```
all_time_high = nvidia_df["Close"].max()
dates_all_time_high = nvidia_df[nvidia_df["Close"] == all_time_high]["Date"]
print(dates_all_time_high)
```

```
3   2024-11-07
Name: Date, dtype: datetime64[ns]
```

3. The 52-Week Range

The 52-week range refers to the lowest and highest price at which a stock has traded during the previous 52 weeks. In this exercise, we consider the **intraday highs and lows**, that is, the 'High' and 'Low' columns, to find the answer.

Note: if the last date of your data is 2024-3-22, then the 52-week range starts at 2023-3-23 and ends at 2024-3-22.

Note: if your starting date falls on a non-trading day, then move the date up to the next trading day.

Write a script to show the 52-week range.

In [306…
```
max_close = nvidia_df["Close"].max()
min_close = nvidia_df["Close"].min()

print(f"52-week range: ${min_close:.2f} - ${max_close:.2f}")
```

```
52-week range: $4.91 - $148.88
```

4. Yearly Performance from 2014 to 2024

From the historical data, we can create a performance summary table for the past 10 years, such as the one shown below.

Year	Open	Close	Ave. Daily Volume	Price Change %
2014	19.2293	27.5950	230032228	43.50
2015	27.8475	26.3150	206585050	-5.50
2016	25.6525	28.9550	153070699	12.87
2017	28.9500	42.3075	108010805	46.14
2018	42.5400	39.4350	135073534	-7.30
2019	38.7225	73.4125	112242893	89.59
2020	74.0600	132.6900	157621403	79.17
2021	133.5200	177.5700	90540527	32.99
2022	177.8300	129.9300	87925014	-26.94
2023	130.2800	192.5300	59232595	47.78
2024	187.1500	172.2800	62188271	-7.95

Follow the steps below to create one for your stock.

Step 1: Reverse the order of the DataFrame

The dates in the DataFrame start from the most recent date and go back in time. Let's reverse the order so the dates are in chronological order.

In [307…
```
# There are several ways to reverse a DataFrame. However, slicing is still the most
# df is the name of your own DataFrame.

df_rev = nvidia_df.loc[::-1].copy()
```

Print the first 10 rows to confirm the results.

In [308…
```
df_rev.head(10)
```

Out[308...

	Date	Close	Volume	Open	High	Low
1257	2019-11-13	5.2142	302364960	5.2088	5.2365	5.1475
1256	2019-11-14	5.2447	527006800	5.2233	5.2470	5.1629
1255	2019-11-15	5.1048	1054216400	5.2425	5.2945	5.0159
1254	2019-11-18	5.3070	575318400	5.0972	5.3338	5.0943
1253	2019-11-19	5.1998	415380400	5.2758	5.2995	5.1413
1252	2019-11-20	5.2795	473017600	5.1768	5.3638	5.1641
1251	2019-11-21	5.2545	288389200	5.2750	5.3310	5.2328
1250	2019-11-22	5.2722	224098800	5.2773	5.3241	5.2485
1249	2019-11-25	5.5303	507347600	5.4010	5.5353	5.3880
1248	2019-11-26	5.4250	385680920	5.5125	5.5172	5.4185

Step 2: Extract the 'year' information from datetime data

Add a new column, 'Year', to the DataFrame. This is important for grouping by year.

In [309...

```
df_rev['Year'] = df_rev['Date'].dt.year
```

Print the first 10 rows of the DataFrame to confirm the results.

In [310...

```
df_rev.head(10)
```

Out[310...

	Date	Close	Volume	Open	High	Low	Year
1257	2019-11-13	5.2142	302364960	5.2088	5.2365	5.1475	2019
1256	2019-11-14	5.2447	527006800	5.2233	5.2470	5.1629	2019
1255	2019-11-15	5.1048	1054216400	5.2425	5.2945	5.0159	2019
1254	2019-11-18	5.3070	575318400	5.0972	5.3338	5.0943	2019
1253	2019-11-19	5.1998	415380400	5.2758	5.2995	5.1413	2019
1252	2019-11-20	5.2795	473017600	5.1768	5.3638	5.1641	2019
1251	2019-11-21	5.2545	288389200	5.2750	5.3310	5.2328	2019
1250	2019-11-22	5.2722	224098800	5.2773	5.3241	5.2485	2019
1249	2019-11-25	5.5303	507347600	5.4010	5.5353	5.3880	2019
1248	2019-11-26	5.4250	385680920	5.5125	5.5172	5.4185	2019

Step 3: Groupby the DataFrame by "Year"

In [311…
```
df_year = df_rev.groupby('Year')
df_year.groups.keys()
```

Out[311…
```
dict_keys([2019, 2020, 2021, 2022, 2023, 2024])
```

Please show the number of trading days for each year from 2013 to 2023.

In [312…
```
trading_days_per_year = df_rev[(df_rev['Year'] >= 2013) & (df_rev['Year'] <= 2023)]
print(trading_days_per_year)
```

```
Year
2019     33
2020    253
2021    252
2022    251
2023    250
Name: count, dtype: int64
```

Step 4: For each year, extract 'Open' price for the first trading day of the year and the 'Close' price for the last trading day of the year.

Note: this will require you to apply separate aggregate functions on 'Open' and 'Close' columns

In [313…
```
df_summary = df_year.agg({'Open': 'first', 'Close': 'last'})
```

Print the DataFrame, df_summary, to confirm the results.

In [314…
```
print(df_summary)
```

```
         Open     Close
Year
2019   5.2088    5.8825
2020   5.9688   13.0550
2021  13.1043   29.4110
2022  29.8150   14.6140
2023  14.8510   49.5220
2024  49.2440  148.2900
```

Step 5: Add a new column, "Average Daily Volume", to df_summary, by calculating the mean of 'Volume' for each year.

Note: the data type of the average daily volume is integer.

In [315…
```
df_summary['Average Daily Volume'] = df_year['Volume'].mean()
```

Print the DataFrame, df_summary, to confirm the results.

In [316…
```
print(df_summary)
```

```
         Open     Close  Average Daily Volume
Year
2019   5.2088    5.8825          3.540595e+08
2020   5.9688   13.0550          4.809963e+08
2021  13.1043   29.4110          3.595957e+08
2022  29.8150   14.6140          5.432221e+08
2023  14.8510   49.5220          4.736280e+08
2024  49.2440  148.2900          4.021603e+08
```

Step 6: Add another new column, "Price Change %", to df_summary, by calculating the percentage change of the stock price for each year.

% change = (Close - Open)/Open * 100

In [317…
```
df_summary['Price Change %'] = ((df_summary['Close'] - df_summary['Open']) / df_sum
```

Print the DataFrame, df_summary. This is your final performance table.

In [318…
```
print(df_summary)
```

```
         Open     Close  Average Daily Volume  Price Change %
Year
2019   5.2088    5.8825          3.540595e+08       12.933881
2020   5.9688   13.0550          4.809963e+08      118.720681
2021  13.1043   29.4110          3.595957e+08      124.437780
2022  29.8150   14.6140          5.432221e+08      -50.984404
2023  14.8510   49.5220          4.736280e+08      233.459026
2024  49.2440  148.2900          4.021603e+08      201.133133
```

5: Buy-and-Hold: Investment Results After 10 Years

Assume you invested $1,000 in the stock on the first trading date of your dataset (say, 2014-2-28) and you continued to hold the stock.

What are the investment values at the end of each year from 2014 to 2024?

Step 1: Calculate the yearly cumulative return

A cumulative return on an investment is the aggregate amount that the investment has gained or lost over time.

Note: cumulative return % = (Close - Open[2014])/Open[2014] * 100

Add an additional column, "Cumm Return %", to df_summary, that shows the yearly cumulative return of the stock.

In [319…
```
df_summary['Cumulative Return %'] = df_summary['Price Change %'].cumsum()
```

Print the DataFrame, df_summary, to show the result

In [320…
```
print(df_summary)
```

```
         Open     Close  Average Daily Volume  Price Change %  \
Year
2019   5.2088    5.8825          3.540595e+08       12.933881
2020   5.9688   13.0550          4.809963e+08      118.720681
2021  13.1043   29.4110          3.595957e+08      124.437780
2022  29.8150   14.6140          5.432221e+08      -50.984404
2023  14.8510   49.5220          4.736280e+08      233.459026
2024  49.2440  148.2900          4.021603e+08      201.133133

      Cumulative Return %
Year
2019            12.933881
2020           131.654562
2021           256.092342
2022           205.107938
2023           438.566964
2024           639.700097
```

Step 2: Calculate the investment values

Using the cumulative returns, add a column, 'Investment Value', by calculating the year-end value of the initial $1,000 investment.

In [321…
```
initial_investment = 1000
df_summary['Investment Value'] = initial_investment * (1 + df_summary['Cumulative R
```

Print the DataFrame, df_summary, to show the result

In [322…
```
print(df_summary)
```

```
         Open     Close  Average Daily Volume  Price Change %  \
Year
2019   5.2088    5.8825          3.540595e+08       12.933881
2020   5.9688   13.0550          4.809963e+08      118.720681
2021  13.1043   29.4110          3.595957e+08      124.437780
2022  29.8150   14.6140          5.432221e+08      -50.984404
2023  14.8510   49.5220          4.736280e+08      233.459026
2024  49.2440  148.2900          4.021603e+08      201.133133

      Cumulative Return %  Investment Value
Year
2019            12.933881      1.129339e+03
2020           131.654562      2.616165e+03
2021           256.092342      9.315963e+03
2022           205.107938      2.842374e+04
2023           438.566964      1.530809e+05
2024           639.700097      1.132339e+06
```