```
-- Question 1
-- SQL query to get Unique Players in the Data
-- DISTINCT gives unique event_user IDs, removing the repetitive ones present in the rest of the table
-- The DISTINCT user ID's are then counted in order to get the total number of unique players

SELECT
   COUNT(DISTINCT event_user)
FROM
   game_data;
```

Simple query, does not have much additional explanation.

Answer:

From pandas,
Unique players in the data: **1275**

```
-- Question 2 A
-- SQL query to get average number of slot machines per session
-- Used subquery: Subquery groups the game sessions together by using session_id (meaning from opening the game to closing) and groups them by session_token which corresponds to the number of times a slot machine is entered
-- Considered using unique slotmachine_id instead but since the term "number of slot machines" was vague, I took it to mean the number of times any slot machine is entered, hence the session_token
-- For the outside query: averages the number of slot machines per session and returns the value
SELECT
    AVG(number_of_slot_machines)
        AS average_number_of_slot_machines
FROM
    (SELECT
        session_id,
            COUNT(DISTINCT session_token)
                AS number_of_slot_machines
    FROM
        game_data
    GROUP BY
        session_id);
```

- subquery that groups session_id together to count the number of machines that were accessed by the player

(DISTINCT session_token) - DISTINCT was added in order to not mix spins in the same machine session

Answer:

From pandas,
Average number of slot machines a player plays in a session: **1.637902175934536**

– Question 2 B
-- SQL query to get the average number of spins per machine session
-- Similar structure to the question before
-- Used subquery to get the count of spins per machine session by grouping the session_tokens together and counting the total per token into a column
-- Outside query calculates the average of all the spins_count column and returns it

```
SELECT
    AVG(spins_count)
        AS average_spins_count
FROM
    (SELECT
        session_token, COUNT(*)
            AS spins_count
    FROM
        game_data
    GROUP BY
        session_token);
```

- subquery that groups by session_token and counts the rows that belong to each token in order to represent a machine session

Answer:

From pandas,
Average number of spins per machine session: **107.53668495597805**

```
-- Question 3
-- SQL query to get the probability of hitting the various win_types
-- Used inline query to count the total count of win_type which is the total number of rows
-- Divided count per win_type over total count to get probability
-- Can add "* 100" in order to make percentages
SELECT
   win_type,
   (COUNT(*) / (
      SELECT
         COUNT(*)
      FROM
         game_data))
            AS win_type_count
FROM
   game_data
GROUP BY
   win_type;
```

  - inline query that counts the total number of win_types by counting the rows

The SQL query was not accepted by pandasql so I converted the query into pandas code instead. Code should return the same results as the SQL query above.

```
# pandas code similar to the sql query above but does not have errors
# calculates the COUNT of each 'win_type'
win_type_counts = df['win_type'].value_counts()

# calculate the total number of rows in the DataFrame
total_count = len(df)

# calculate the 'win_type_count' as count of each 'win_type' divided by
the total count
result_df = pd.DataFrame({'win_type': win_type_counts.index,
'win_type_count': win_type_counts.values / total_count})

print("Table of win_types and their respective probabilities: ")
print(result_df)
```

Answer:

From pandas,

Table of win_types and their respective probabilities:

	win_type	win_type_count
0	none	0.562943
1	fake	0.273501
2	regular	0.118346
3	big	0.027120
4	fivekind	0.008833
5	mega	0.006301
6	ultra	0.002956

```
-- Question 4
-- SQL query to get the retention rate
-- Has 1 external query and 2 subqueries
-- subquery 1 returns the total number of unique players who returned after 24 hours have passed since install date
-- subquery 2 returns the total number of unique players
-- combined the two queries together in order to get a table where the two count values are side by side
-- dividing the two counts together and multiplying by 100 gives the percentage of players who returned
SELECT
   count_more_than_24_hours / total_count * 100
      AS retention_rate
FROM
   (SELECT
      COUNT(DISTINCT event_user)
         AS count_more_than_24_hours
   FROM
      game_data
   WHERE
      event_time > install_date + INTERVAL 24 HOUR)
         AS subquery1
      CROSS JOIN
   (SELECT
      COUNT(DISTINCT event_user)
         AS total_count
   FROM
      game_data)
         AS subquery2;
```

- subquery 1 which counts the unique returning players
- subquery 2 which counts the total unique players
- CROSS JOIN was used to combine the two tables (2 1x1 tables) into one (1 1x2 table)
- calculates the retention rate percentage

SQL query was not processed well by pandasql so I converted it into a similar pandas code instead. Code should return the same result as the SQL query.

```
# pandas code that returns the same result as the SQL query
# Calculate count_more_than_24_hours by filtering the data frame of those that have spun since the 24 hour mark
filtered_data = df[df['event_time'] > df['install_date'] + pd.Timedelta(hours=24)]
# only take unique users
```

```
count_more_than_24_hours = filtered_data['event_user'].nunique()

# calculate unique users
total_count = df['event_user'].nunique()

# Calculate retention_rate
retention_rate = (count_more_than_24_hours / total_count) * 100

print("The retention rate of the game is: " + retention_rate)
```

Answer:
From pandas,

The retention rate of the game is: **36.549019607843135**

```
-- Question 5
-- SQL query to get the average RTP
-- nested query
-- summary query is the innermost query which calculates RTP per spin
-- RTP is then averaged together according to slotmachine_id
SELECT
    slotmachine_id, AVG(rtp) AS avg_rtp
FROM
    (SELECT
        slotmachine_id,
            amount / total_bet_amount AS rtp
    FROM
        game_data
            ) AS summary
GROUP BY
            slotmachine_id
```

- subquery which calculates the rtp per row

The SQL query had errors in the pandasql module so I converted the SQL query into a similar pandas code instead. The code in pandas is as follows:

```
df['rtp'] = df['amount'] / df['total_bet_amount']

# Group by 'slotmachine_id' and calculates the average 'rtp' per
slotmachine_id
summary_df = df.groupby('slotmachine_id')['rtp'].mean().reset_index()

# Rename the 'rtp' column to 'avg_rtp'
summary_df.rename(columns={'rtp': 'avg_rtp'}, inplace=True)

print(summary_df)
```

Answer:
From pandas,

	slotmachine_id	avg_rtp
0	**AdventuresOfAlice**	**0.892479**
1	**ArcaneReels**	**0.715103**
2	**AstroBlitz**	**0.687863**
3	**BigBucksBarbeque**	**0.806378**

4	BigWinBuffalo	0.678838
5	BubbleCubes	0.691478
6	CasinoCats	0.684488
7	CasinoClassicSevens	368.750000
8	Cleopatra	0.886940
9	ClockworkChronicles	8.664632
10	DiaDeMuertos	0.796922
11	DiamondDeluxe	0.783850
12	Diner	0.746502
13	DoubleWinClassic	4.017158
14	DragonCubes	0.782849
15	Easter	6.683764
16	FairyTale	0.566664
17	FarmCubes	7.086228
18	Freya	0.614902
19	GalleonsOfGlory	0.758251
20	GoldenConquest	0.755877
21	GoldenDragon	0.703146
22	HackCity	0.683897
23	ImmortalFortunes	1.605245
24	JuiceCubes	0.780718
25	JungleCubes	0.784546
26	KingdomOfGold	0.761619
27	KongsQuest	0.694621
28	LuckyClassic777	0.518239
29	MysticWolf	1.632389
30	PiratePlunder	0.832638
31	PotOfGold	1.189712
32	ReelsOfSpeed	0.712102
33	RichesOfAtlantis	0.404248
34	RichesOfOsiris	10.481468
35	SantaExtravaganza	0.769127
36	SaveTheEarth	0.703766
37	SciFi	0.957634
38	SherwoodBounty	0.692348
39	StarshipSquadron	0.749665
40	SteamyHotJackpots	0.706160
41	SweetStakes	0.748697
42	TemptationIsland	0.848238
43	TheOutsiders	1.016247
44	TitanDefenders	0.906801
45	VegasRoyale	0.695511
46	WildWest	0.904452
47	WorldFeast	0.807976

48	**goldennights**	**0.964556**