STAT 207 Homework 3 Solutions [50 points]

Data Frame Cleaning and Quantitative Variable Descriptive Analytics

Due: Friday, February 3 by noon (11:59 am) CST*

*No late submissions accepted

Case Study: Steam Game Play and Purchase Analysis

Steam is the world's most popular PC Gaming hub. With a massive collection that includes everything from AAA blockbusters to small indie titles, great discovery tools can be super valuable for Steam. What can we learn about Steam gaming behavior?

The dataset we will be analyzing is comprised of a random sample of 500 steam users and their game play and purchase behaviors. It has the following columns:

- user_id
- game_name,
- activity:
  - purchase: indicating that the user has *purchased* the corresponding game
  - play: indicating that the user has *played* the corresponding game (for at least some amount of time.)
- hours_played_if_play:
  - if the row corresponds to a 'play' activity, this number represents the number of hours the user has played the game
  - if the row corresponds to a 'purchase' activity, this number is always a 1 (and means nothing... it's a placeholder).

This is the same data that you worked with in Lab 2.

Package Imports

We'll import four Python packages that we've used so far and will need for this assignment. Those packages are pandas, numpy, matplotlib.pyplot, and seaborn.

Run the cell provided below to access the functions in these packages.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Note: you may have already completed Steps 1 & 2 below in Lab 2. Be sure to repeat this analysis here, showing the full process so that we can follow your logic and thought process.

1. Read in the Data [1 point]

Read in the steam_sample.csv data.

In [2]:
```
df_original = pd.read_csv('steam_sample.csv')
```

2. Cleaning the Data [5 points]

We plan to analyze the "hours_played_if_play" variable from the steam data.

Before beginning our analysis, we need to prepare the data. To do so, one step is to check for missing data. While doing so, be sure to pay careful attention to our variable of interest.

Complete the following steps to clean the data:

1. Identify values (if any) that have been encoded in the csv to represent a **missing value**.
2. Read in the csv file so that the missing values are written as NaN in the data.
3. Confirm that the "hours_played_if_play" is recognized as numerical values (either float64 or int64 type).
4. Delete any rows from your data that have missing values.
5. Report the number of observations dropped from the data.

First, let's use the **.dtypes** attribute as one way to *check* if there are missing values in our numerical variables in this dataframe. We know that the hours_played_if_play variable should be numerical, however dtypes returns **object** indicating that the types of values in this dataframe are not all of a numerical type of object (like int64 or float64 for instance).

In [3]:
```
df_original.dtypes
```

Out[3]:
```
user_id                  int64
game_name               object
activity                object
hours_played_if_play    object
dtype: object
```

Let's inspect the unique values of this column.

In [4]:
```
df_original['hours_played_if_play'].unique()
```

Out[4]:
```
array(['1', '0.6', '22', '1028', '1008', '148', '108', '72', '36', '35',
       '32', '21', '16', '15.8', '8.6', '7.8', '7.3', '3.1', '1.9', '1.7',
       '1.1', '0.4', '153', '63', '26', '1.4', '639', '479', '70', '65',
       '33', '30', '19.8', '16.2', '11.3', '4.2', '3.9', '2.3', '0.8',
       '0.7', '0.5', '0.3', '396', '227', '13.4', '12.6', '11.2', '10.1',
       '2.4', '210', '1.2', '0.2', '13.2', '48', '110', '0.1', '67',
       '429', '5.5', '61', '1.6', '18.3', '9.9', '4.7', '1714', '441',
       '197', '147', '117', '86', '73', '49', '46', '31', '24', '20',
       '19.7', '18.2', '14.2', '11.6', '9.8', '9.7', '8.4', '6.5', '4.8',
       '4.1', '3.7', '2.9', '2.7', '2.1', '222', 'unknown', '14.9',
       '14.1', '83', '11.1', '3.2', '6.9', '395', '251', '9.3', '7.4',
       '54', '34', '1.8', '99', '98', '96', '29', '27', '23', '19.1',
       '18.7', '17.5', '17', '16.6', '14.7', '13.3', '10.7', '10.2', '10',
       '9.6', '9.5', '8', '7.9', '7.6', '7', '6.6', '6', '5.8', '5.6',
       '5.3', '4.4', '4.3', '3.3', '3', '2.6', '2.5', '0.9', '6.2',
       '18.9', '2.8', '4814', '4', '494', '1.3', '184', '13.8', '11.4',
       '8.9', '6.7', '496', '1086', '525', '370', '270', '151', '85',
       '66', '56', '15.3', '15.1', '11.5', '7.2', '3.8', '19.6', '2',
       '274', '7.1', '4.9', '4.6', '3.6', '12.1', '103', '13.1', '60',
       '3309', '524', '431', '428', '262', '250', '180', '154', '137',
       '106', '100', '80', '68', '62', '50', '47', '44', '38', '37',
       '13.5', '12', '11.9', '11', '9', '8.2', '8.1', '7.5', '6.8', '5.1',
       '5', '2.2', '1.5', '57', '964', '41', '12.5', '10.4', '361', '88',
       '82', '17.6', '17.1', '16.3', '15.2', '14.6', '11.7', '8.8', '8.3',
       '5.9', '717', '303', '13.9', '10.6', '2179', '526', '28', '14.5',
       '12.4', '149', '10.8', '15', '19.9', '113', '76', '646', '25',
       '14', '546', '11.8', '381', '16.1', '215', '81', '19.5', '248',
       '59', '42', '13.7', '3626', '997', '665', '622', '522', '344',
       '224', '213', '208', '186', '95', '79', '78', '40', '18.1', '18',
       '17.8', '17.4', '15.9', '15.5', '14.8', '12.3', '3.5', '3.4',
       '116', '6.1', '393', '332', '238', '307', '157', '550', '52',
       '9.2', '1343', '13.6', '8.5', '15.7', '202', '530', '43', '6.3',
       '518', '232', '175', '58', '90', '15.4', '14.4', '193', '173',
       '12.2', '1339', '190', '121', '7.7', '4.5', '249', '77', '399',
       '278', '91', '6.4', '162', '89', '17.2', '9.1', '652', '517',
       '125', '105', '75', '55', '17.9', '10.9', '8.7', '10.3', '5.7',
       '5.4', '10.5', '118', '9.4', '284', '71', '19.3', '209', '5.2',
       '1605', '1540', '710', '617', '312', '644', '189', '191', '176',
       '136', '115', '39', '19', '69', '1912', '777', '686', '15.6',
       '18.4', '539', '1760', '165', '17.3', '112', '386', '720', '466',
       '124', '109', '87', '14.3', '676', '449', '236', '119', '53', '45',
       '3178', '1302', '795', '174', '123', '104', '101', '64', '51',
       '16.9', '16.8', '16.5', '1030', '145', '214', '11754', '144',
       '579', '263', '241', '160', '141', '120', '92', '13', '289',
       '1337', '17.7', '2258', '217', '135', '389', '228', '187', '111',
       '357', '923', '172', '1369', '1049', '488', '130', '126', '93',
       '18.8', '1064', '18.5', '12.9', '458', '418', '194', '272', '16.4',
       '419', '107', '290', '1499', '891', '693', '302', '273', '264',
       '158', '133', '1301', '485', '1754', '231', '12.8', '132', '733',
       '230', '127', '627', '994', '387', '487', '528', '138', '809',
       '511', '2055', '402', '451', '170', '527', '206', '164', '207',
       '196', '143', '1295', '520', '364', '1704'], dtype=object)
```

We see two problems here.

1. It looks like there is a value labeled 'unknown' in this column. The values with 'unknown' in the csv are what the csv creator used to represent a missing value in the data.
2. The number values in this column are represented as strings rather than integers or floats.

Let's use the technique we learned in class to fix both of these problems at once. Below, we read in the csv file again, this time indicating int he **pd.read_csv()** function that we should convert all entries with the 'unknown' into a NaN object.

In [5]:
```
df=pd.read_csv('steam_sample.csv', na_values=['unknown'])
df.head()
```

Out[5]:

	user_id	game_name	activity	hours_played_if_play
0	308653033	Unturned	purchase	1.0
1	308653033	Unturned	play	0.6
2	308653033	theHunter	purchase	1.0
3	144004384	Dota 2	purchase	1.0
4	144004384	Dota 2	play	22.0

Let's double check that this fixed the problem.

First, we can use the **.dtypes** attribute again and we now see that the 'hours_played_if_play' is comprised of all objects of float64 type. So this means that all of our values in this column will either be a 'float64' type or a NaN value.

In [6]:
```
df.dtypes
```

Out[6]:
```
user_id                    int64
game_name                 object
activity                  object
hours_played_if_play     float64
dtype: object
```

Using the **.isna()** function followed by the **.sum()** function we can see that Python detected 6 values in the hours_played_if_play column and automatically converted them into NaN values.

In [7]:
```
df.isna().sum()
```

Out[7]:
```
user_id                 0
game_name               0
activity                0
hours_played_if_play    6
dtype: int64
```

Finally, we will overwrite our dataframe by dropping all of the missing values.

In [8]:
```
df=df.dropna()
df.isna().sum()
```

Out[8]:
```
user_id                 0
game_name               0
activity                0
hours_played_if_play    0
dtype: int64
```

In [9]:
```
df_original.shape[0] - df.shape[0]
```

Out[9]:
```
6
```

We dropped 6 observations from our data due to missing values.

3. Purchases & Played Games [4 points]

a) Separate the Steam data into two different data frames:

- one for the user-game entries that have been played
- one for the user-game entries that represent purchases

In [10]:
```
df_play=df[df['activity']=='play']
df_play.head()
```

Out[10]:

	user_id	game_name	activity	hours_played_if_play
1	308653033	Unturned	play	0.6
4	144004384	Dota 2	play	22.0
6	54103616	Counter-Strike Global Offensive	play	1028.0
8	54103616	Counter-Strike	play	1008.0

	user_id	game_name	activity	hours_played_if_play
10	54103616	Left 4 Dead	play	148.0

In [11]:

```
df_purchase=df[df['activity']=='purchase']
df_purchase.head()
```

Out[11]:

	user_id	game_name	activity	hours_played_if_play
0	308653033	Unturned	purchase	1.0
2	308653033	theHunter	purchase	1.0
3	144004384	Dota 2	purchase	1.0
5	54103616	Counter-Strike Global Offensive	purchase	1.0
7	54103616	Counter-Strike	purchase	1.0

b) What proportion of games purchased have been played?

In [12]:

```
df_play.shape[0]/df_purchase.shape[0]
```

Out[12]:

```
0.5827922077922078
```

4. A High Game User [10 points]

a) Extract the row, which represents a user-game combination, that has the **sixth** highest amount of total hours played in the dataset.

To get full credit, you should do this in one line of code.

In [13]:

```
df_play.sort_values(by=['hours_played_if_play'], ascending=False).iloc[[5],:]
```

Out[13]:

	user_id	game_name	activity	hours_played_if_play
5463	39911920	Counter-Strike	play	2258.0

b) Filter the data to contain only the transactions (purchases and games played) for this user. Display this data frame.

In [14]:

```
df_user = df[df['user_id']==39911920]
df_user
```

Out[14]:

	user_id	game_name	activity	hours_played_if_play
5462	39911920	Counter-Strike	purchase	1.0
5463	39911920	Counter-Strike	play	2258.0
5464	39911920	Counter-Strike Condition Zero	purchase	1.0
5465	39911920	Counter-Strike Condition Zero	play	32.0
5466	39911920	Counter-Strike Condition Zero Deleted Scenes	purchase	1.0
5467	39911920	Day of Defeat	purchase	1.0
5468	39911920	Deathmatch Classic	purchase	1.0

	user_id	game_name	activity	hours_played_if_play
5469	39911920	Ricochet	purchase	1.0

c) Calculate the percentage of purchased games that this user actually played.

To get full credit, you should do this in one line of code using the dataframe created in part 4b.

In [15]:
```
df_user[df_user['activity'] == 'play'].shape[0] / df_user[df_user['activity'] == 'purchase
```

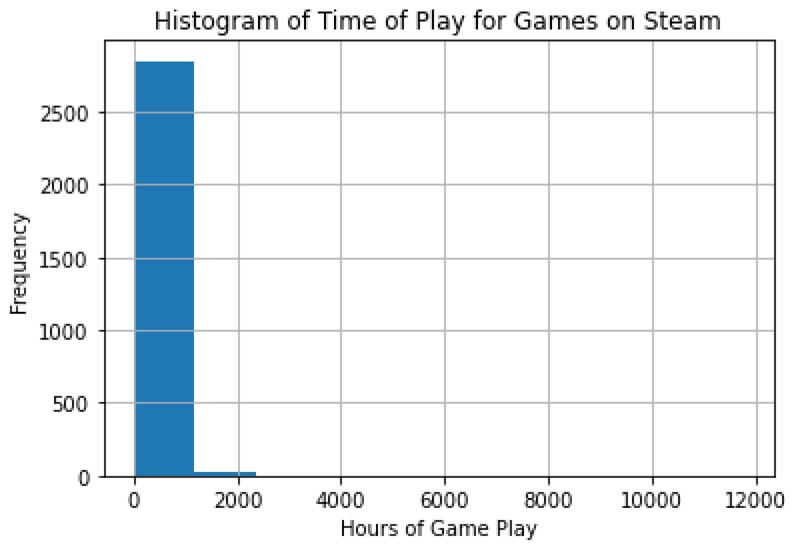
Out[15]:
```
0.3333333333333333
```

5. Distribution of Hours Played [9 points]

We will now focus on the time spent playing games.

Generate a **frequency** histogram for the time spent playing an individual game by an individual user. Make sure to include clear axes labels and titles on your graph. Then, describe the distribution.

In [16]:
```
df_play['hours_played_if_play'].hist()
plt.xlabel("Hours of Game Play")
plt.ylabel("Frequency")
plt.title("Histogram of Time of Play for Games on Steam")
plt.show()
```



This distribution of hours of game play for a user-game combination is unimodal and skewed right, with a peak around 500 hours, most games played by a user between about 0 and about 2200 hours, and a few apparent upper outliers.

6. Low and High Game Users [5 points]

a) Calculate the minimum time spent playing a game in our data. How many rows represent user-game combinations that were played for the minimum number of hours?

In [17]:
```
print(df_play['hours_played_if_play'].min())
print((df_play['hours_played_if_play'] == df_play['hours_played_if_play'].min()).sum())
```

```
0.1
```

```
67
```

The minimum time spent playing a game in our data is 0.1 hours. 67 users played a game for 0.1 hours.

b) Repeat the calculation from part a for the maximum time spent playing a game.

In [18]:
```
print(df_play['hours_played_if_play'].max())
print((df_play['hours_played_if_play'] == df_play['hours_played_if_play'].max()).sum())
```

```
11754.0
1
```

The maximum time spent playing a game in our data is 11,754 hours. Only 1 user played a game for 11,754 hours. This is not a surprising result.

7. Comparing Play Times of Two Games [16 points]

We want to compare the play times that users spent with two games of different genres:

- Counter-Strike is a first-person shooter game
- Sid Meier's Civilization V is a turn-based strategy game

a) First, filter the data to create two separate data frames that contain the play times for users of these two games. How many users played each of these two games?

Note: calculating the number of users with Python is sufficient; you do not need to report the number of users in a Markdown cell.

In [19]:
```
df_cs = df_play[df_play['game_name'] == 'Counter-Strike']
df_cs.shape
```

Out[19]:
```
(30, 4)
```

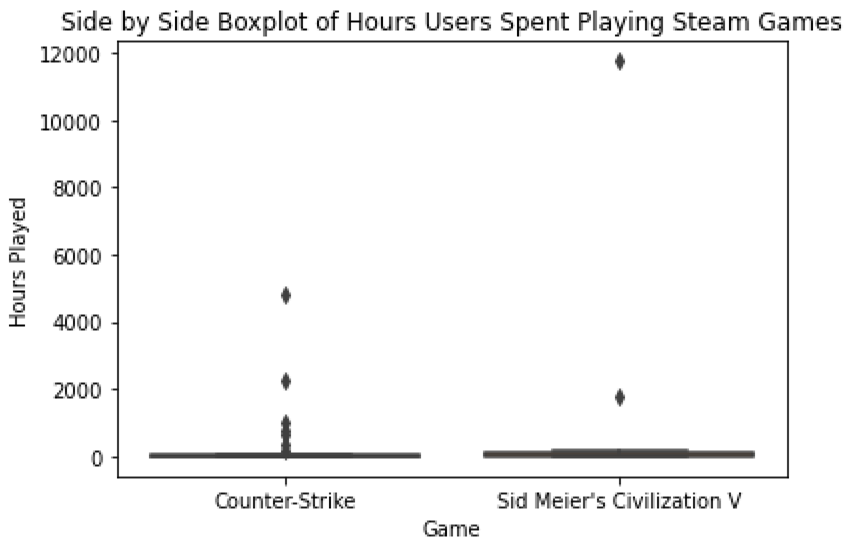
In [20]:
```
df_civ = df_play[df_play['game_name'] == "Sid Meier's Civilization V"]
df_civ.shape
```

Out[20]:
```
(20, 4)
```

b) Stack the separate data frames from part **a** into one data frame. Then, generate a side-by-side box plot of the play times based on the game played.

In [21]:
```
df_twogames = pd.concat([df_cs, df_civ], ignore_index = True)
```

In [22]:
```
sns.boxplot(x='game_name', y='hours_played_if_play', data=df_twogames)
plt.xlabel('Game')
plt.ylabel('Hours Played')
plt.title('Side by Side Boxplot of Hours Users Spent Playing Steam Games')
plt.show()
```



c) Calculate the five number summary, mean, and standard deviation of the play times for both of these games.

In [23]:
```
def mysummary(series):
    index = ['min', 'Q1', 'med', 'Q3', 'max', 'mean', 'std']
    value = [series.min(),
             series.quantile(q=0.25),
             series.median(),
             series.quantile(q=0.75),
             series.max(),
             series.mean(),
             series.std()]
    return pd.DataFrame({'value': value}, index=index)
```

In [24]:
```
print('Hours Spent Playing Counter-Strike on Steam Summary Statistics')
mysummary(df_cs['hours_played_if_play'])
```

```
Hours Spent Playing Counter-Strike on Steam Summary Statistics
```

Out[24]:

	value
min	0.200000
Q1	1.625000
med	7.750000
Q3	47.750000
max	4814.000000
mean	342.436667
std	964.242763

In [25]:
```
print('Hours Spent Playing Civilization on Steam Summary Statistics')
mysummary(df_civ['hours_played_if_play'])
```

```
Hours Spent Playing Civilization on Steam Summary Statistics
```

Out[25]:

	value
min	0.500000
Q1	10.875000
med	43.000000

	value
Q3	109.750000
max	11754.000000
mean	724.500000
std	2624.251107

d) Finally, compare the distributions of play times for these two games. Does there seem to be an association between play time and game for this sample of users?

We can see from the summary statistics that the time users spent playing Civilization is generally larger than the time users spent playing Counter-Strike. Each of the corresponding statistics is larger for Civilization than for Counter-Strike. Because there are a couple of large outliers for both games, the box plots are challenging to interpret. We can tell that there appear to be more upper outliers for Counter-Strike than Civilization. Of the top seven times spent playing either Counter-Strike or Civilization, it appears that 5 are from Counter-Strike. Other than the largest amount of time, the next two also come from Counter-Strike.

Because the statistics are always larger for Civilization, there does appear to be an association between play time and the game for this data.

Remember to keep all your cells and hit the save icon above periodically to checkpoint (save) your results on your local computer. Once you are satisified with your results restart the kernel and run all (Kernel -> Restart & Run All). **Make sure nothing has changed**. Checkpoint and exit (File -> Save and Checkpoint + File -> Close and Halt). Follow the instructions on the Homework 3 Canvas Assignment to submit your notebook to GitHub.