Shopify Fall 2022 Data Science Intern Challenge

Upon examination of the data, I have concluded that the suspect average order value (AOV) of $3,415.13 is a result of outlier data in both total_items and order_amount columns. I have gathered that there were shoes with a cost of \$25,725 for a single pair as well as orders whose total items were 2000. Both of these findings when retained lead to a right skewed data distribution and subsequently an inaccurate AOV.

After scrutiny of the data, I chose to work with a subset of the original dataset. The subset I designated optimal contains 4928 rows or 98.56\% of the total dataset. This subset of data eliminated all rows containing the outlier price of $25,725 for a pair of shoes. Additionally, all rows where the total_items entry is greater than or equal to 6 were eliminated as these orders account for only 0.54\% of total orders and therefore deemed atypical. Lastly, in order to have a more accurate understanding of each order, this subset contains a new column entitled **cost_one_pair** (As each store only sells one pair of shoes, the cost of one pair is exact).

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

Subset of original dataset, hereafter referred to as dataset

The AOV for this dataset is $301.47 with the average cost for one pair of shoes at \$151.80.

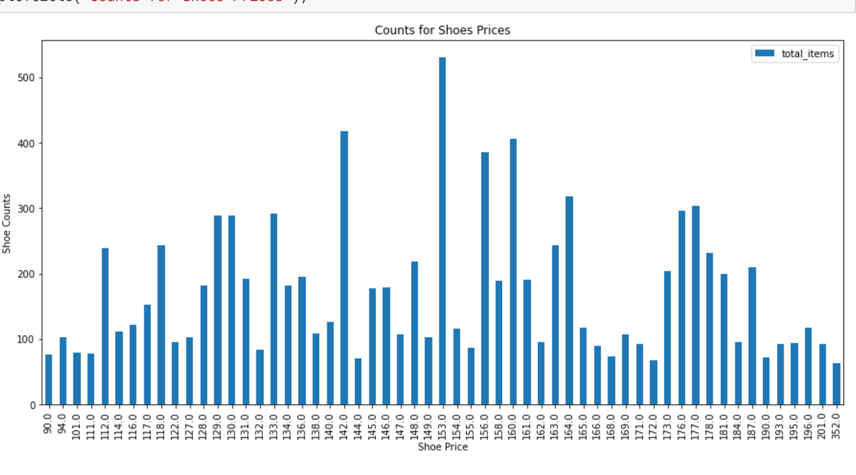
In [28]:
```
new_df = df.loc[(df.cost_one_pair <= 352) & (df.total_items < 6)]
new_df.describe()
```

Out[28]:

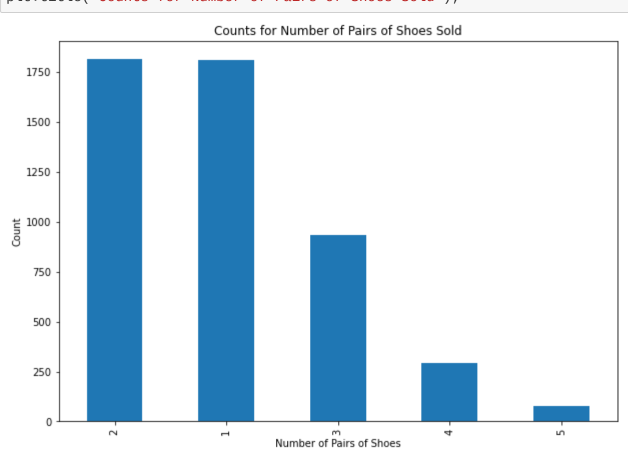
	order_id	shop_id	user_id	order_amount	total_items	cost_one_pair
count	4928.000000	4928.000000	4928.000000	4928.000000	4928.000000	4928.000000
mean	2497.955560	49.822646	849.810268	301.474432	1.987013	151.798295
std	1444.145469	29.065769	86.812494	158.775640	0.966579	29.050049
min	1.000000	1.000000	700.000000	90.000000	1.000000	90.000000
25%	1245.750000	24.000000	776.000000	163.000000	1.000000	132.000000
50%	2494.500000	50.000000	850.000000	284.000000	2.000000	153.000000
75%	3749.250000	74.000000	925.000000	387.000000	3.000000	166.000000
max	5000.000000	100.000000	999.000000	1760.000000	5.000000	352.000000

The dataset has acceptable mean and 50th percentile (median) values for both order_amount and cost_one_pair. While those values for both columns have a slighter higher mean than median, the difference is considered negligible. Overall, the data is symmetrical.

In [29]:
```
new_df[['cost_one_pair','total_items']].groupby(['cost_one_pair']).sum().plot.bar(figsize=(15,7));
plt.xlabel('Shoe Price')
plt.ylabel('Shoe Counts')
plt.title('Counts for Shoes Prices');
```



In [30]:
```
new_df.total_items.value_counts().plot.bar(figsize=(10,7))
plt.xlabel('Number of Pairs of Shoes')
plt.ylabel('Count')
plt.title('Counts for Number of Pairs of Shoes Sold');
```



Recomendations

For current as well as any future stores, it is recommended that a record of the shoe price be kept per store. Additionally, as 93.4% of orders were sales of either one, two, or three pairs of shoes (goes down to 74.3% for one or two pairs), an examination of the price range for these shoes would be informative and an avenue for higher profits for Shopify stores. Finally, an exploration of the user ids for stores that had orders with (the outlier) total items of 2000 is suggested as an avenue to procure - or at minimun advertise to - similar customers as well as to ensure continuous communication with those users.

Exploratory Data Analysis

In [2]:
```
df = pd.read_excel("Downloads/2019 Winter Data Science Intern Challenge Data Set.xlsx")
```

In [3]:
```
df
```

Out[3]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
0	1	53	746	224	2	cash	2017-03-13 12:36:56.190
1	2	92	925	90	1	cash	2017-03-03 17:38:51.999
2	3	44	861	144	1	cash	2017-03-14 04:23:55.595
3	4	18	935	156	1	credit_card	2017-03-26 12:43:36.649
4	5	18	883	156	1	credit_card	2017-03-01 04:35:10.773
...	...	...	...	...	...	...	...
4995	4996	73	993	330	2	debit	2017-03-30 13:47:16.597
4996	4997	48	789	234	2	cash	2017-03-16 20:36:16.389
4997	4998	56	867	351	3	cash	2017-03-19 05:42:42.228
4998	4999	60	825	354	2	credit_card	2017-03-16 14:51:18.188
4999	5000	44	734	288	2	debit	2017-03-18 15:48:18.205

5000 rows × 7 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   order_id        5000 non-null   int64
 1   shop_id         5000 non-null   int64
 2   user_id         5000 non-null   int64
 3   order_amount    5000 non-null   int64
 4   total_items     5000 non-null   int64
 5   payment_method  5000 non-null   object
 6   created_at      5000 non-null   datetime64[ns]
dtypes: datetime64[ns](1), int64(5), object(1)
memory usage: 273.6+ KB
```

Descriptive statistics for dataset. Note the discrepancy in the mean and 50th percentile (median) for the order_amount column. The mean being so much greater signifies the order_amount distribution is greatly skewed right and thus assymmetrical (i.e. there are outlying data points on the high end of order_amount entries).

In [5]:
```
df.describe()
```

Out[5]:

	order_id	shop_id	user_id	order_amount	total_items
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000
mean	2500.500000	50.078800	849.092400	3145.128000	8.78720
std	1443.520003	29.006118	87.798982	41282.539349	116.32032
min	1.000000	1.000000	607.000000	90.000000	1.00000
25%	1250.750000	24.000000	775.000000	163.000000	1.00000
50%	2500.500000	50.000000	849.000000	284.000000	2.00000
75%	3750.250000	75.000000	925.000000	390.000000	3.00000
max	5000.000000	100.000000	999.000000	704000.000000	2000.00000

Average order value when all orders and all shoes are retained is $3145.13 which is the mean of the order_amount column. This is the suspect AOV from the challenge.

In [6]:
```
df.payment_method.unique()
```

Out[6]:
```
array(['cash', 'credit_card', 'debit'], dtype=object)
```

Total pairs of shoes sold is 43,936

In [7]:
```
items_total = df.total_items.sum()
print(items_total)
```

```
43936
```

Total amount of all sales is $15,725,640

In [8]:
```
amount_total = df.order_amount.sum()
print(amount_total)
```

```
15725640
```

Average cost of a pair of shoes when all orders and all shoes are retained is $357.92

In [9]:
```
print(amount_total/items_total)
```

```
357.92152221412965
```

Examining unique amounts in the order_amount column to look for outliers.

In [10]:
```
df.order_amount.unique()
```

Out[10]:
```
array([   224,     90,    144,    156,    138,    149,    292,    266,
          146,    322,    111,    447, 704000,    176,    164,    258,
          408,    486,    704,    256,    495,    177,    145,    112,
          178,    310,    294,    465,    244,    316,    459,    148,
          352,    624,    284,    195,    652,    346,    438,    296,
          399,    276,    531,    692,    181,    561,    560,    187,
          260,    160,    480,    390,    184,    122,    130,    116,
          142,    127,    354,    132,    254,    584,    328,    234,
          118,    513,    193,    180,    780,    612,    532,    320,
          201,    414,    129,    136,    165,    262,    590,    374,
          153,    492,    280,    188,    312,    544,    765,    426,
          441,    290,    640,    222,    435,    173,    306,    330,
        25725,    228,    272,    489,    464,    468,    351,    147,
          342,    387,    648,    488,    362,    516,    326,    338,
          117,    163,    393,    268,    588,    236,    202,    133,
          336,    585,    570,    114,    360,    512,    324,    402,
          171,    128,    534,    386,    190,    131,    420,    282,
          368,    232,    432,    656,    161,    158,    519,    381,
          632,    166,    543,    356,    332,    508,    134,    392,
          568,    298,    154,    172,    169,    504,  51450,    308,
          712,    520,    670,    830,    396,    660,    579,    483,
          444,    644,    462,    710,    536,    456,    380,    748,
           94,    303, 154350,    348,    168,    772,    804,    507,
          333,    815,    645,    344,    376,    576,    528,    196,
          470,    498,    384,    672,    885,    552,   1056,    592,
          474,    676,    580,    450,    784,    140,    730,    101,
          162,    708,    288,    155,    935,  77175,    524,   1760,
         1408,    264,    616,    736,    448,    960,    800,    596,
          366,    270,    404,    865,    745,    664,    880,    920,
          774,    790,    603,    948,    724,    845,    760, 102900,
          965,    665,    805,    890,    980,    786,    655,    472,
         1086,    684,    816,    810,    740,    620,   1064,    735,
          725,    650])
```

Adding new column **cost_one_pair** to the dataframe to ascertain the average price of one pair of shoes for each order

In [11]:
```
df['cost_one_pair'] = df.apply(lambda row: row.order_amount/row.total_items, axis=1)
df
```

Out[11]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
0	1	53	746	224	2	cash	2017-03-13 12:36:56.190	112.0
1	2	92	925	90	1	cash	2017-03-03 17:38:51.999	90.0
2	3	44	861	144	1	cash	2017-03-14 04:23:55.595	144.0
3	4	18	935	156	1	credit_card	2017-03-26 12:43:36.649	156.0
4	5	18	883	156	1	credit_card	2017-03-01 04:35:10.773	156.0
...	...	...	...	...	...	...	...	...
4995	4996	73	993	330	2	debit	2017-03-30 13:47:16.597	165.0
4996	4997	48	789	234	2	cash	2017-03-16 20:36:16.389	117.0
4997	4998	56	867	351	3	cash	2017-03-19 05:42:42.228	117.0
4998	4999	60	825	354	2	credit_card	2017-03-16 14:51:18.188	177.0
4999	5000	44	734	288	2	debit	2017-03-18 15:48:18.205	144.0

5000 rows × 8 columns

Examine unique amounts in new column av_cost_one_pair to look for outliers. The highest prices for a pair of shoes are $352 and \$25,725

In [12]:
```
df.cost_one_pair.unique()
```

Out[12]:
```
array([  112.,    90.,   144.,   156.,   138.,   149.,   146.,   133.,
         161.,   111.,   352.,   176.,   164.,   129.,   136.,   162.,
         128.,   165.,   177.,   145.,   178.,   155.,   147.,   122.,
         158.,   153.,   148.,   142.,   195.,   163.,   173.,   181.,
         187.,   130.,   160.,   184.,   116.,   127.,   118.,   132.,
         117.,   171.,   193.,   201.,   131.,   140.,    94., 25725.,
         114.,   172.,   169.,   134.,   196.,   101.,   168.,   190.,
         166.,   154.])
```

Examining subset of dataset which contains only the two highest prices for a pair of shoes. This constraint yields a subset of 97 rows or 1.94% of total dataset.

In [13]:
```
df.loc[df.cost_one_pair > 201]
```

Out[13]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
15	16	42	607	704000	2000	credit_card	2017-03-07 04:00:00.000	352.0
40	41	42	793	352	1	credit_card	2017-03-24 14:15:40.649	352.0
60	61	42	607	704000	2000	credit_card	2017-03-04 04:00:00.000	352.0
160	161	78	990	25725	1	credit_card	2017-03-12 05:56:56.834	25725.0
308	309	42	770	352	1	credit_card	2017-03-11 18:14:38.774	352.0
...	...	...	...	...	...	...	...	...
4745	4746	42	872	352	1	debit	2017-03-24 00:57:24.130	352.0
4767	4768	42	720	704	2	credit_card	2017-03-14 10:26:08.027	352.0
4868	4869	42	607	704000	2000	credit_card	2017-03-22 04:00:00.000	352.0
4882	4883	42	607	704000	2000	credit_card	2017-03-25 04:00:00.000	352.0
4918	4919	78	823	25725	1	cash	2017-03-15 13:26:46.262	25725.0

97 rows × 8 columns

Examining subset of dataset which contains only the highest price ($25,725) for a pair of shoes. This constraint yields a subset of 46 rows or 0.92\% of total dataset. Note, with one pair of shoes being so exorbitant, the order_amount for these entries greatly skews the amount for total sales of all orders.

In [14]:
```
df.loc[df.cost_one_pair > 352]
```

Out[14]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
160	161	78	990	25725	1	credit_card	2017-03-12 05:56:56.834	25725.0
490	491	78	936	51450	2	debit	2017-03-26 17:08:18.911	25725.0
493	494	78	983	51450	2	cash	2017-03-16 21:39:35.400	25725.0
511	512	78	967	51450	2	cash	2017-03-09 07:23:13.640	25725.0
617	618	78	760	51450	2	cash	2017-03-18 11:18:41.848	25725.0
691	692	78	878	154350	6	debit	2017-03-27 22:51:43.203	25725.0
1056	1057	78	800	25725	1	debit	2017-03-15 10:16:44.830	25725.0
1193	1194	78	944	25725	1	debit	2017-03-16 16:38:25.551	25725.0
1204	1205	78	970	25725	1	credit_card	2017-03-17 22:32:21.438	25725.0
1259	1260	78	775	77175	3	credit_card	2017-03-27 09:27:19.843	25725.0
1384	1385	78	867	25725	1	cash	2017-03-17 16:38:06.279	25725.0
1419	1420	78	912	25725	1	cash	2017-03-30 12:23:42.551	25725.0
1452	1453	78	812	25725	1	credit_card	2017-03-17 18:09:54.089	25725.0
1529	1530	78	810	51450	2	cash	2017-03-29 07:12:01.466	25725.0
2270	2271	78	855	25725	1	credit_card	2017-03-14 23:58:21.635	25725.0
2452	2453	78	709	51450	2	cash	2017-03-27 11:04:04.363	25725.0
2492	2493	78	834	102900	4	debit	2017-03-04 04:37:33.848	25725.0
2495	2496	78	707	51450	2	cash	2017-03-26 04:38:52.497	25725.0
2512	2513	78	935	51450	2	debit	2017-03-18 18:57:13.421	25725.0
2548	2549	78	861	25725	1	cash	2017-03-17 19:35:59.663	25725.0
2564	2565	78	915	77175	3	debit	2017-03-25 01:19:35.410	25725.0
2690	2691	78	962	77175	3	debit	2017-03-22 07:33:25.104	25725.0
2773	2774	78	890	25725	1	cash	2017-03-26 10:36:43.445	25725.0
2818	2819	78	869	51450	2	debit	2017-03-17 06:25:50.921	25725.0
2821	2822	78	814	51450	2	cash	2017-03-02 17:13:25.271	25725.0
2906	2907	78	817	77175	3	debit	2017-03-16 03:45:46.089	25725.0
2922	2923	78	740	25725	1	debit	2017-03-12 20:10:58.008	25725.0
3085	3086	78	910	25725	1	cash	2017-03-26 01:59:26.748	25725.0
3101	3102	78	855	51450	2	credit_card	2017-03-21 05:10:34.147	25725.0
3151	3152	78	745	25725	1	credit_card	2017-03-18 13:13:07.198	25725.0
3167	3168	78	927	51450	2	cash	2017-03-12 12:23:07.516	25725.0
3403	3404	78	928	77175	3	debit	2017-03-16 09:45:04.544	25725.0
3440	3441	78	982	25725	1	debit	2017-03-19 19:02:53.732	25725.0
3705	3706	78	828	51450	2	credit_card	2017-03-14 20:43:14.502	25725.0
3724	3725	78	766	77175	3	credit_card	2017-03-16 14:13:25.868	25725.0
3780	3781	78	889	25725	1	cash	2017-03-11 21:14:49.542	25725.0
4040	4041	78	852	25725	1	cash	2017-03-02 14:31:11.566	25725.0
4079	4080	78	946	51450	2	cash	2017-03-20 21:13:59.919	25725.0
4192	4193	78	787	77175	3	credit_card	2017-03-18 09:25:31.863	25725.0
4311	4312	78	960	51450	2	debit	2017-03-01 03:02:10.223	25725.0
4412	4413	78	756	51450	2	debit	2017-03-02 04:13:38.530	25725.0
4420	4421	78	969	77175	3	debit	2017-03-09 15:21:34.551	25725.0
4505	4506	78	866	25725	1	debit	2017-03-22 22:06:00.804	25725.0
4584	4585	78	997	25725	1	cash	2017-03-25 21:48:43.570	25725.0
4715	4716	78	818	77175	3	debit	2017-03-05 05:10:43.633	25725.0
4918	4919	78	823	25725	1	cash	2017-03-15 13:26:46.262	25725.0

In [15]:
```
df.loc[df.cost_one_pair > 352].count()
```

Out[15]:
```
order_id          46
shop_id           46
user_id           46
order_amount      46
total_items       46
payment_method    46
created_at        46
cost_one_pair     46
dtype: int64
```

Examining subset of dataset which contains only the second highest price ($325) for a pair of shoes. This constraint yields a subset of 51 rows or 1.02\% of total dataset. Note, there are 17 orders where the total_items entry is 2000. Thus, both the order_amount and the total_items for these entries greatly skews the amount for total sales of all orders.

In [16]:
```
df.loc[df.cost_one_pair == 352]
```

Out[16]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
15	16	42	607	704000	2000	credit_card	2017-03-07 04:00:00.000	352.0
40	41	42	793	352	1	credit_card	2017-03-24 14:15:40.649	352.0
60	61	42	607	704000	2000	credit_card	2017-03-04 04:00:00.000	352.0
308	309	42	770	352	1	credit_card	2017-03-11 18:14:38.774	352.0
409	410	42	904	704	2	credit_card	2017-03-04 14:32:57.621	352.0
520	521	42	607	704000	2000	credit_card	2017-03-02 04:00:00.000	352.0
834	835	42	792	352	1	cash	2017-03-25 21:31:24.596	352.0
835	836	42	819	704	2	cash	2017-03-09 14:15:15.136	352.0
938	939	42	808	1056	3	credit_card	2017-03-13 23:43:45.330	352.0
979	980	42	744	352	1	debit	2017-03-12 13:09:03.570	352.0
1104	1105	42	607	704000	2000	credit_card	2017-03-24 04:00:00.000	352.0
1362	1363	42	607	704000	2000	credit_card	2017-03-15 04:00:00.000	352.0
1364	1365	42	797	1760	5	cash	2017-03-10 06:28:21.039	352.0
1367	1368	42	926	1408	4	cash	2017-03-13 02:38:33.953	352.0
1436	1437	42	607	704000	2000	credit_card	2017-03-11 04:00:00.000	352.0
1471	1472	42	907	1408	4	debit	2017-03-12 23:00:21.521	352.0
1512	1513	42	946	352	1	debit	2017-03-24 13:35:04.115	352.0
1520	1521	42	756	704	2	debit	2017-03-22 13:10:31.448	352.0
1562	1563	42	607	704000	2000	credit_card	2017-03-19 04:00:00.000	352.0
1602	1603	42	607	704000	2000	credit_card	2017-03-17 04:00:00.000	352.0
1911	1912	42	739	704	2	cash	2017-03-07 05:42:52.154	352.0
1929	1930	42	770	352	1	credit_card	2017-03-17 08:11:13.184	352.0
2003	2004	42	934	704	2	cash	2017-03-26 09:21:25.916	352.0
2018	2019	42	739	352	1	debit	2017-03-01 12:42:25.759	352.0
2053	2054	42	951	352	1	debit	2017-03-19 11:49:12.071	352.0
2153	2154	42	607	704000	2000	credit_card	2017-03-12 04:00:00.000	352.0
2273	2274	42	747	704	2	debit	2017-03-27 20:48:19.095	352.0
2297	2298	42	607	704000	2000	credit_card	2017-03-07 04:00:00.000	352.0
2491	2492	42	868	704	2	debit	2017-03-01 18:33:33.377	352.0
2609	2610	42	868	704	2	debit	2017-03-23 18:10:13.529	352.0
2766	2767	42	970	704	2	credit_card	2017-03-05 10:45:41.577	352.0
2835	2836	42	607	704000	2000	credit_card	2017-03-28 04:00:00.000	352.0
2969	2970	42	607	704000	2000	credit_card	2017-03-28 04:00:00.000	352.0
2987	2988	42	819	1056	3	cash	2017-03-03 09:09:25.327	352.0
3332	3333	42	607	704000	2000	credit_card	2017-03-24 04:00:00.000	352.0
3513	3514	42	726	1056	3	debit	2017-03-24 17:51:04.520	352.0
3651	3652	42	830	352	1	credit_card	2017-03-24 22:26:57.542	352.0
3697	3698	42	839	352	1	debit	2017-03-12 02:45:08.932	352.0
3903	3904	42	975	352	1	debit	2017-03-12 01:28:30.852	352.0
3998	3999	42	886	352	1	debit	2017-03-09 20:10:40.505	352.0
4056	4057	42	607	704000	2000	credit_card	2017-03-28 04:00:00.000	352.0
4231	4232	42	962	352	1	cash	2017-03-04 00:01:19.133	352.0
4294	4295	42	859	704	2	cash	2017-03-24 20:50:39.607	352.0
4326	4327	42	788	704	2	debit	2017-03-16 23:37:56.791	352.0
4421	4422	42	736	704	2	credit_card	2017-03-01 12:19:49.175	352.0
4625	4626	42	809	352	1	credit_card	2017-03-11 08:21:26.204	352.0
4646	4647	42	607	704000	2000	credit_card	2017-03-02 04:00:00.000	352.0
4745	4746	42	872	352	1	debit	2017-03-24 00:57:24.130	352.0
4767	4768	42	720	704	2	credit_card	2017-03-14 10:26:08.027	352.0
4868	4869	42	607	704000	2000	credit_card	2017-03-22 04:00:00.000	352.0
4882	4883	42	607	704000	2000	credit_card	2017-03-25 04:00:00.000	352.0

In [17]:
```
df.loc[df.cost_one_pair == 352].count()
```

Out[17]:
```
order_id          51
shop_id           51
user_id           51
order_amount      51
total_items       51
payment_method    51
created_at        51
cost_one_pair     51
dtype: int64
```

Average spent on a pair of shoes with outlier costs of $352 and \$25,725 excluded. Doing so reduces total number of orders to 4903 or 98.06% of total dataset.

In [18]:
```
av_one = ((df['cost_one_pair'].loc[df.cost_one_pair <= 201]).sum())/4903
av_one
```

Out[18]:
```
150.40016316540894
```

Average spent on a pair of shoes with outlier cost of $25,725 excluded. Doing so reduces total number of orders to 4954 or 99.08% of total dataset.

In [19]:
```
av_two = ((df['cost_one_pair'].loc[df.cost_one_pair <= 352]).sum())/4954
av_two
```

Out[19]:
```
152.47557529269278
```

Retaining the second highest cost ($352) for a pair of shoes as opposed to eliminating two highest costs (\$352 and $25,725) yields a difference of $2.08 in average spent on a pair of shoes. This difference is deemed negligible.

Examine unique entries in the total_items column. Subsequently established the count for each unique entry. Orders for six pairs of shoes account for 0.18% of total orders. Orders for eight pairs of shoes account for 0.02% of total orders. Orders for 2,000 pairs of shoes account for 0.34% of total orders. All together, orders that have these entries for total_items account for 0.54% of total orders.

In [20]:
```
df.total_items.unique()
```

Out[20]:
```
array([   2,    1,    3, 2000,    4,    5,    6,    8])
```

In [21]:
```
one_pair = df.total_items.loc[df.total_items == 1].count()
two_pair = df.total_items.loc[df.total_items == 2].count()
three_pair = df.total_items.loc[df.total_items == 3].count()
four_pair = df.total_items.loc[df.total_items == 4].count()
five_pair = df.total_items.loc[df.total_items == 5].count()
six_pair = df.total_items.loc[df.total_items == 6].count()
eight_pair = df.total_items.loc[df.total_items == 8].count()
twothou_pair = df.total_items.loc[df.total_items == 2000].count()
print(one_pair, two_pair, three_pair, four_pair, five_pair, six_pair, eight_pair, twothou_pair)
```

```
1830 1832 941 293 77 9 1 17
```

In [22]:
```
df.loc[df.total_items == 6]
```

Out[22]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
691	692	78	878	154350	6	debit	2017-03-27 22:51:43.203	25725.0
1563	1564	91	934	960	6	debit	2017-03-23 08:25:48.948	160.0
2127	2128	83	745	774	6	cash	2017-03-27 06:59:46.440	129.0
2307	2308	61	723	948	6	credit_card	2017-03-26 11:29:37.333	158.0
3252	3253	67	706	786	6	credit_card	2017-03-29 16:05:41.357	131.0
3538	3539	43	830	1086	6	debit	2017-03-17 19:56:29.343	181.0
3865	3866	68	815	816	6	debit	2017-03-11 09:31:49.831	136.0
4711	4712	86	883	780	6	cash	2017-03-18 14:18:18.549	130.0
4847	4848	13	993	960	6	cash	2017-03-27 11:00:45.292	160.0

In [23]:
```
df.loc[df.total_items == 8]
```

Out[23]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
4141	4142	54	733	1064	8	debit	2017-03-07 17:05:17.910	133.0

In [24]:
```
df.loc[df.total_items == 2000]
```

Out[24]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
15	16	42	607	704000	2000	credit_card	2017-03-07 04:00:00	352.0
60	61	42	607	704000	2000	credit_card	2017-03-04 04:00:00	352.0
520	521	42	607	704000	2000	credit_card	2017-03-02 04:00:00	352.0
1104	1105	42	607	704000	2000	credit_card	2017-03-24 04:00:00	352.0
1362	1363	42	607	704000	2000	credit_card	2017-03-15 04:00:00	352.0
1436	1437	42	607	704000	2000	credit_card	2017-03-11 04:00:00	352.0
1562	1563	42	607	704000	2000	credit_card	2017-03-19 04:00:00	352.0
1602	1603	42	607	704000	2000	credit_card	2017-03-17 04:00:00	352.0
2153	2154	42	607	704000	2000	credit_card	2017-03-12 04:00:00	352.0
2297	2298	42	607	704000	2000	credit_card	2017-03-07 04:00:00	352.0
2835	2836	42	607	704000	2000	credit_card	2017-03-28 04:00:00	352.0
2969	2970	42	607	704000	2000	credit_card	2017-03-28 04:00:00	352.0
3332	3333	42	607	704000	2000	credit_card	2017-03-24 04:00:00	352.0
4056	4057	42	607	704000	2000	credit_card	2017-03-28 04:00:00	352.0
4646	4647	42	607	704000	2000	credit_card	2017-03-02 04:00:00	352.0
4868	4869	42	607	704000	2000	credit_card	2017-03-22 04:00:00	352.0
4882	4883	42	607	704000	2000	credit_card	2017-03-25 04:00:00	352.0

Subset of data with all rows containing the outlier price of $25,725 for a pair of shoes eliminated. Additionally, all rows where the total_items entry is greater than or equal to 6 were eliminated. Consequently, this subset contains 4928 rows or 98.56\% of total dataset.

In [25]:
```
new_df = df.loc[(df.cost_one_pair <= 352) & (df.total_items < 6)]
new_df
```

Out[25]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	cost_one_pair
0	1	53	746	224	2	cash	2017-03-13 12:36:56.190	112.0
1	2	92	925	90	1	cash	2017-03-03 17:38:51.999	90.0
2	3	44	861	144	1	cash	2017-03-14 04:23:55.595	144.0
3	4	18	935	156	1	credit_card	2017-03-26 12:43:36.649	156.0
4	5	18	883	156	1	credit_card	2017-03-01 04:35:10.773	156.0
...	...	...	...	...	...	...	...	...
4995	4996	73	993	330	2	debit	2017-03-30 13:47:16.597	165.0
4996	4997	48	789	234	2	cash	2017-03-16 20:36:16.389	117.0
4997	4998	56	867	351	3	cash	2017-03-19 05:42:42.228	117.0
4998	4999	60	825	354	2	credit_card	2017-03-16 14:51:18.188	177.0
4999	5000	44	734	288	2	debit	2017-03-18 15:48:18.205	144.0

4928 rows × 8 columns

Descriptive statistics for subset of data.

In [26]:
```
new_df.describe()
```

Out[26]:

	order_id	shop_id	user_id	order_amount	total_items	cost_one_pair
count	4928.000000	4928.000000	4928.000000	4928.000000	4928.000000	4928.000000
mean	2497.955560	49.822646	849.810268	301.474432	1.987013	151.798295
std	1444.145469	29.065769	86.812494	158.775640	0.966579	29.050049
min	1.000000	1.000000	700.000000	90.000000	1.000000	90.000000
25%	1245.750000	24.000000	776.000000	163.000000	1.000000	132.000000
50%	2494.500000	50.000000	850.000000	284.000000	2.000000	153.000000
75%	3749.250000	74.000000	925.000000	387.000000	3.000000	166.000000
max	5000.000000	100.000000	999.000000	1760.000000	5.000000	352.000000

An examination of mean and 50th percentile (median) for order_amount and cost_one_pair with second highest cost ($352) for one pair of shoes eliminated. The change is negligible so the choice is keep as much of the original data as possible.

In [27]:
```
try_df = df.loc[(df.cost_one_pair < 352) & (df.total_items < 6)]
try_df.describe()
```

Out[27]:

	order_id	shop_id	user_id	order_amount	total_items	cost_one_pair
count	4894.000000	4894.000000	4894.000000	4894.000000	4894.000000	4894.000000
mean	2497.977728	49.876992	849.917450	299.037597	1.987944	150.407438
std	1444.298056	29.159238	86.859966	153.806187	0.966249	23.860542
min	1.000000	1.000000	700.000000	90.000000	1.000000	90.000000
25%	1244.250000	24.000000	776.000000	163.000000	1.000000	132.000000
50%	2497.500000	50.000000	850.000000	283.000000	2.000000	153.000000
75%	3748.750000	74.000000	925.000000	386.000000	3.000000	166.000000
max	5000.000000	100.000000	999.000000	980.000000	5.000000	201.000000

In [ ]: