Steam Marketplace Analysis

Jacob Knispel (Team Lead), Nithin Perumal, Alec Tiefenthal, Matt Buckner

Table of Contents

You need to provide more background for those unfamiliar with Team Fortress. What are "unusuals" and "keys" and "refined metals."

Executive Summary

Much clearer than before.

In this project, we seek to examine the data provided by **Valve Software** pertaining to their online digital economy of in-game items in their game **Team Fortress 2** in the attempt to locate statistically interesting anomalies or trends that may not be immediately obvious to the outside observer.

Introduction

Project Objectives:

- Find an interesting trend/story within Steam Market data
- Create visualizations for Steam Market data
- Form an interactive visualization of Steam Market data
- If time allows, enable users to dynamically retrieve current Steam Market data

Acquiring and Cleaning the Data:

The data was acquired with the backpack.tf api. Data was obtained by making calls to this api to obtain information regarding market prices as well as the price history of items. The api returned JSON responses for the calls. These JSON responses were quite tightly nested and therefore we had to unnest a many layered JSON file that was provided to us. After we managed to unnest this JSON file, the contents were poured into a CSV file as manipulating structured data in the CSV file is far easier to do.

No heavy cleaning was needed since the backpack.tf api returned datasets without too many errors.

Issues encountered during the Data Acquisition phase:

The JSON responses were nested heavily and it therefore took a lot of effort to unnest this JSON response and transfer it to a .csv file for the creation of visualizations as well as to run other algorithms on the data.

We were able to locate some interesting trends when comparing the attributes of a number of different items.

Data Preprocessing

In [5]:
```
import requests;                                 # import package used to downlaod
import json;
import pandas as pd;
import seaborn as sns;
import matplotlib.pyplot as plt;                 # import all tools used to examin
import matplotlib;
from bokeh.charts import Scatter, show;
from bokeh.io import output_notebook;
output_notebook();

from pandas.io.json import json_normalize;

matplotlib.style.use('ggplot');
%matplotlib inline
```

(http://bokeh.pydata.org) BokehJS successfully loaded.

The following lines utilize the API calls and key to download and store the data for examination

In [6]:
```
with open('API Keys.txt', 'r') as keyfile:                              # This key
    apiKeys=keyfile.read().split('\n')                                  # The spec

url = 'http://backpack.tf/api/IGetPrices/v4/?key=' + apiKeys[0];        # web addr

data_stream = requests.get(url,stream=True);                            # create a

rec = data_stream.iter_lines().next().strip()                           # Necessar
data = json.loads(rec)                                                  # convert
```

In [7]:
```
def byteify(input):
    if isinstance(input, dict):
        return {byteify(key): byteify(value)
                for key, value in input.iteritems()}
    elif isinstance(input, list):
        return [byteify(element) for element in input]
    elif isinstance(input, unicode):
        return input.encode('utf-8')
    else:
        return input

data2 = byteify(data)
```

In [8]:
```
df = json_normalize(data2['response'])
```

In [9]:
```
#Define values provided by Valve.

raw_usd_value = df['raw_usd_value']
current_time = df['current_time']
success = df['success']
usd_currency_index = df['usd_currency_index']
usd_currency = df['usd_currency']
```

In [10]:

```
#Import current item data

#THIS WILL TAKE >2 MINUTES TO RUN

dfItemFinal = pd.DataFrame(columns = ['Name', 'DefIndex', 'QualInt', 'Trade', 'Cra
df = json_normalize(data2['response'], 'items')
for itemName in df[0]:
    dfName = json_normalize(data2['response']['items'], [itemName])
    if(dfName.size != 2):
        raise Exception('More than 2 items for item' + itemName)
    if(dfName[0][0] != 'prices'):
        raise Exception('First row in %s is not prices' % (itemName))
    if(dfName[0][0] != 'prices'):
        raise Exception('Second row in %s is not defindex' % (itemName))
    defIndex = json_normalize(data2['response']['items'][itemName], ['defindex'])[
    dfPrices = json_normalize(data2['response']['items'][itemName], 'prices')
    for qualInt in dfPrices[0]:
        dfTrade = json_normalize(data2['response']['items'][itemName]['prices'], q

        for tradeable in dfTrade[0]:
            dfCraft = json_normalize(data2['response']['items'][itemName]['prices'

            for craftable in dfCraft[0]:
                dfPriceIndex = json_normalize(data2['response']['items'][itemName]

                for priceIndex in dfPriceIndex[0]:
                    dfFinalValues = json_normalize(data2['response']['items'][item

                    currency = dfFinalValues['currency'][0]
                    difference = dfFinalValues['difference'][0]
                    last_update = dfFinalValues['last_update'][0]
                    value = dfFinalValues['value'][0]
                    dfItemFinal.loc[len(dfItemFinal)]=[itemName, defIndex, qualInt
```

In [108]:

```
#Store the item data in an csv file for formatting and examinination

dfItemFinal.set_index(['Name', 'DefIndex', 'QualInt', 'Trade', 'Craft', 'PriceInde
#dfItemFinal = dfItemFinal.drop('Unnamed: 0', axis=1)
dfItemFinal.sort_index(inplace = True)
dfItemFinal.to_csv('itemOut.csv')
```

Once the data is stored in a .csv file, the following line prepares it for examination. If a set of data has already been prepared, one may skip directly to this point to begin working with the dataset.

In [109]:

```
#Move to our primary dataframe and access our item data

dfNew = pd.read_csv('itemOut.csv')
#dfNew = dfNew.set_index(['Name', 'DefIndex', 'QualInt', 'Trade', 'Craft', 'PriceI
dfNew = dfNew.drop('Unnamed: 0', axis=1)
dfNew.sort_index(inplace = True)
```

In [110]:

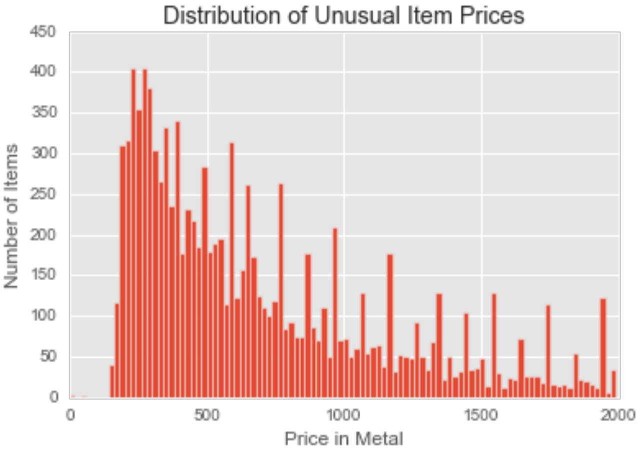
```
#Standardize the currency (there are 3 types: metal, keys, and usd. We want to tra
refinedMetalPerKey = dfNew[dfNew['Name'] == 'Mann Co. Supply Crate Key']['Value'].
dfNew.ix[dfNew['Currency'] == 'keys', 'Value'] = dfNew.ix[dfNew['Currency'] == 'ke

#Remove usd currency (this is usually pretty uninteresting)
dfNew = dfNew[dfNew['Currency'] != 'usd']
dfNew = dfNew.drop('Currency', axis=1)
```

Analysis

```
In [111]: dfAllUnusuals = dfNew[dfNew['QualInt'] == 5]
          dfUnusuals = dfAllUnusuals[dfAllUnusuals['Value'] < 2000] # Remove obvious outlier
          ax = dfUnusuals['Value'].plot(kind='hist', title='Distribution of Unusual Item Pri
          ax.set_xlabel("Price in Metal")
          ax.set_ylabel("Number of Items")
```

```
Out[111]: <matplotlib.text.Text at 0x26afed30>
```



It's obvious from this graph that unusual items rarely sell for less than 5 or 6 keys (~100 refined metal right now), but most of them are sold for around 10 keys (~200 refined metal). This phenomenon can be explained very simply -- in order to get an unusual, a player must actually use a key to open something called a crate. They have a 1.00% chance to get an unusual item in return.

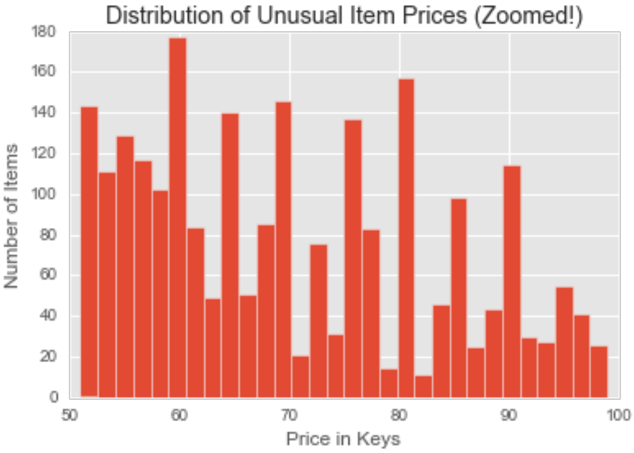
For this reason, it wouldn't make sense for unusual items to sell for anywhere close to 1 key, because all unusuals, no matter how little people like them, are rarer than keys, which can be bought from the store via microtransactions (for $2.50)¶

```
In [112]: dfUnusuals['Value'] = dfUnusuals['Value']/refinedMetalPerKey
          dfUnusuals = dfUnusuals[dfUnusuals['Value'] < 100] # Zoom
          dfUnusuals = dfUnusuals[dfUnusuals['Value'] > 50] # Zooom!
          ax = dfUnusuals['Value'].plot(kind='hist', title='Distribution of Unusual Item Pri
          ax.set_xlabel("Price in Keys")
          ax.set_ylabel("Number of Items")
```

```
C:\Users\knispeja\Anaconda2\lib\site-packages\ipykernel\__main__.py:1: SettingWit
hCopyWarning:
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: http://pandas.pydata.org/pandas-docs/stable
/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy)
  if __name__ == '__main__':
```

```
Out[112]: <matplotlib.text.Text at 0x2b916f98>
```



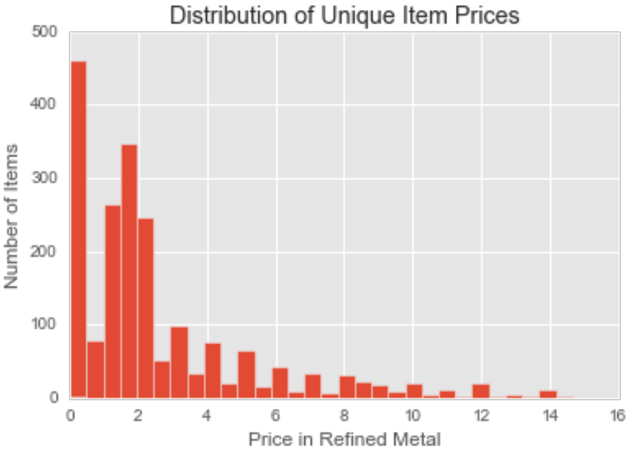
It's interesting to note that, because the prices are defined freely by humans (by however much people decide to sell their items for), there is a great tendency for prices to fall on or near values that are multiples of 5. For example, looking at the range 70-80 keys, there are spikes at 70 keys, 75 keys, and 80 keys.

Even the smaller spikes in-between the multiples of 5 can be explained -- these are at 73.33 and 76.66 from our range defined before.

Why? Items on the TF2 market are often priced at values like '2.33 keys' or '1.66 refined metal' because these are more convenient to split than the obscure fractions the 'true price' usually lands on.

```
In [113]: dfAllUniques = dfNew[dfNew['QualInt'] == 6]
          dfUniques = dfAllUniques[dfAllUniques['Value'] < 15] # Remove apparent outliers
          ax = dfUniques['Value'].plot(kind='hist', title='Distribution of Unique Item Price
          ax.set_xlabel("Price in Refined Metal")
          ax.set_ylabel("Number of Items")

Out[113]: <matplotlib.text.Text at 0x2c444b00>
```



Similar to the previous graph, we can clearly see that unique items tend to sell for easily-divisible values. However, refined metal can actually be split in-game, always into 3 parts. Therefore, prices are almost always multiples of three -- the most common price for a unique item is 0.33 refined metal, which is equivalent to a single piece of a split refined metal.

It's also interesting to note here how much cheaper unique items tend to be compared to unusuals. A unique item usually costs around 2 refined metal. In comparison, the median unusual item costs a whopping 657 refined metal, or about 33 keys, which amounts to $80!

Results

Should not just print out a table of values. Need to characterize your data. Seems you are still working on your project and it is incomplete in its current state, especially for a four member group.

In [116]:

```
print('Median unique item metal cost: %0.2f' % (dfAllUniques['Value'].median()))
print('Median unusual item metal cost: %0.2f' % (dfAllUnusuals['Value'].median()))

dfAllUnusuals = dfAllUnusuals.sort(['Value'])
dfAllUnusuals.tail(50)
```

```
Median unique item metal cost: 2.00
Median unusual item metal cost: 657.22
```

```
C:\Users\knispeja\Anaconda2\lib\site-packages\ipykernel\__main__.py:4: FutureWarn
ing: sort(columns=....) is deprecated, use sort_values(by=.....)
```

Out[116]:

	Name	DefIndex	QualInt	Trade	Craft	PriceIndex	Value	Last_Update	Differen
11594	Horrific Headsplitter	291	5	Tradable	Craftable	13	21649.60	1438035983	18293.2
4165	El Jefe	539	5	Tradable	Craftable	13	21649.60	1414121014	4072.65
866	Noh Mercy	361	5	Tradable	Craftable	17	22229.50	1435671765	-10003.8
9665	Crone's Dome	920	5	Tradable	Craftable	45	22229.50	1442996596	-7160.53
8039	Executioner	921	5	Tradable	Craftable	17	22229.50	1452697945	-4050.09
8842	Law	30362	5	Tradable	Craftable	17	22229.50	1450466486	-3481.50
446	Team Captain	378	5	Tradable	Craftable	72	22326.15	1406035139	990.867
871	Noh Mercy	361	5	Tradable	Craftable	30	23002.70	1411742030	8372.97
3538	Tipped Lid	30425	5	Tradable	Craftable	13	23196.00	1447437831	7838.40
12697	Brotherhood of Arms	30066	5	Tradable	Craftable	17	23196.00	1444676903	2408.25
1843	Villain's Veil	393	5	Tradable	Craftable	45	23196.00	1448882413	6127.50
13697	Polar Pullover	30329	5	Tradable	Craftable	45	23196.00	1453096023	20994.0
8936	Mask of the Shaman	514	5	Tradable	Craftable	38	24278.48	1413045068	3211.47
13705	Polar Pullover	30329	5	Tradable	Craftable	10	25129.00	1452795180	22743.5
8028	Executioner	921	5	Tradable	Craftable	13	25708.90	1409237558	1583.88
3369	Coffin Kit	938	5	Tradable	Craftable	44	25708.90	1454428084	10933.3
253	Virtual Viewfinder	30140	5	Tradable	Craftable	14	27062.00	1454596395	4253.85
450	Team Captain	378	5	Tradable	Craftable	14	27062.00	1445983777	5638.50
264	Virtual Viewfinder	30140	5	Tradable	Craftable	17	27062.00	1433496411	3811.00
3368	Coffin Kit	938	5	Tradable	Craftable	45	27177.98	1409506998	5461.59
865	Noh Mercy	361	5	Tradable	Craftable	14	28028.50	1421128338	2863.38
12696	Brotherhood of Arms	30066	5	Tradable	Craftable	14	28995.00	1453329542	5248.50

Links

API keys.txt (https://github.com/NithinPerumal/SteamMarketAnalysis/blob/master/IPython%20Notebook/API%20Keys.txt)

Sample CSV dataset (https://github.com/NithinPerumal/SteamMarketAnalysis/blob/master/IPython%20Notebook/itemOut.csv)

Known Quality Interger Meanings (https://github.com/NithinPerumal/SteamMarketAnalysis/blob/master/IPython%20Notebook/Known%20Quality%20Integer%20Meanings.txt) - List of mappings between integers and qualitative descriptions of items they correspond with

TF2 Item Qualities (https://wiki.teamfortress.com/wiki/Item_quality)

IGetPrices API (https://backpack.tf/api/prices) - Used for quick, current TF2 item data

IGetMarketPrices API (https://backpack.tf/api/market)

Disclaimer

All data examined has been freely provided by Valve Software via backpack.tf's APIs for the purpose of allowing users to examine this publicly available information.

Powered by Steam (http://steampowered.com)