Warehouse

September 7, 2018

1 Warehouse cargo analysis

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

In [2]:
```
columns = ['package', 'height', 'date', 'destination_agent', 'shipper']
df = pd.DataFrame(pd.read_csv('dataset.csv', low_memory=False), columns=columns)
df['height'] = df['height'].str.replace(',','.')
df['height'] = pd.to_numeric(df['height'])
df['date'] = pd.to_datetime(df['date'])
```

Cleaning dataframe (rows without height are useless)

In [3]:
```
df = df[df.height != 0.00]
df = df.fillna('')
df.head(10)
```

Out[3]:

	package	height	date	destination_agent	shipper
0	Box	7.5	2018-08-30 00:00:00		Midwest Truck & Auto Parts, Inc.
1	Box	7.4	2018-08-30 00:00:00		Midwest Truck & Auto Parts, Inc.
2	Box	16.7	2018-08-30 00:00:00		Ligowave Networks Inc
3	Box	10.8	2018-08-30 00:00:00		Ligowave Networks Inc
4	Box	10.6	2018-08-30 00:00:00		Ligowave Networks Inc
5	PALLET	57.0			
6	Box	19.0			
7	Box	57.0			
8	PALLET	39.0	2018-08-30 00:00:00		
9	PALLET	38.0			

```
        7
        8  BIGREEN INTERNATIONAL PRODUCTS CORP
        9
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 322861 entries, 0 to 339149
Data columns (total 5 columns):
package              322861 non-null object
height               322861 non-null float64
date                 322861 non-null object
destination_agent    322861 non-null object
shipper              322861 non-null object
dtypes: float64(1), object(4)
memory usage: 14.8+ MB
```

Detecting and cleaning outliers

```
In [5]: df = df[df.height < 110.00]
        df = df[df.height > 10.00]
```

Max, min and avg values

```
In [6]: max_height = df['height'].max()
        print(max_height)
```

```
109.0
```

```
In [7]: min_height = df['height'].min()
        print(min_height)
```

```
10.08
```

```
In [8]: avg_height = df['height'].mean()
        print(avg_height)
```

```
37.65439892632642
```

```
In [9]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 261532 entries, 2 to 339148
Data columns (total 5 columns):
package              261532 non-null object
height               261532 non-null float64
```

```
date                 261532 non-null object
destination_agent    261532 non-null object
shipper              261532 non-null object
dtypes: float64(1), object(4)
memory usage: 12.0+ MB
```

Defining categories

```
Category 1 = between 0' and 25'
Category 2 = between 25' and 50'
Category 3 = between 50' and 75'
Category 4 = between 75' and 100'
Category 5 = 100' and higher values
```

```
In [10]: df['category'] = None
         for i, row in df.iterrows():
             if row['height'] > 0 and row['height'] < 25:
                 df.at[i, 'category'] = 1
             if row['height'] > 25 and row['height'] < 50:
                 df.at[i, 'category'] = 2
             if row['height'] > 50 and row['height'] < 75:
                 df.at[i, 'category'] = 3
             if row['height'] > 75 and row['height'] < 100:
                 df.at[i, 'category'] = 4
             if row['height'] > 100:
                 df.at[i, 'category'] = 5
```

```
In [11]: df.head(3)
```

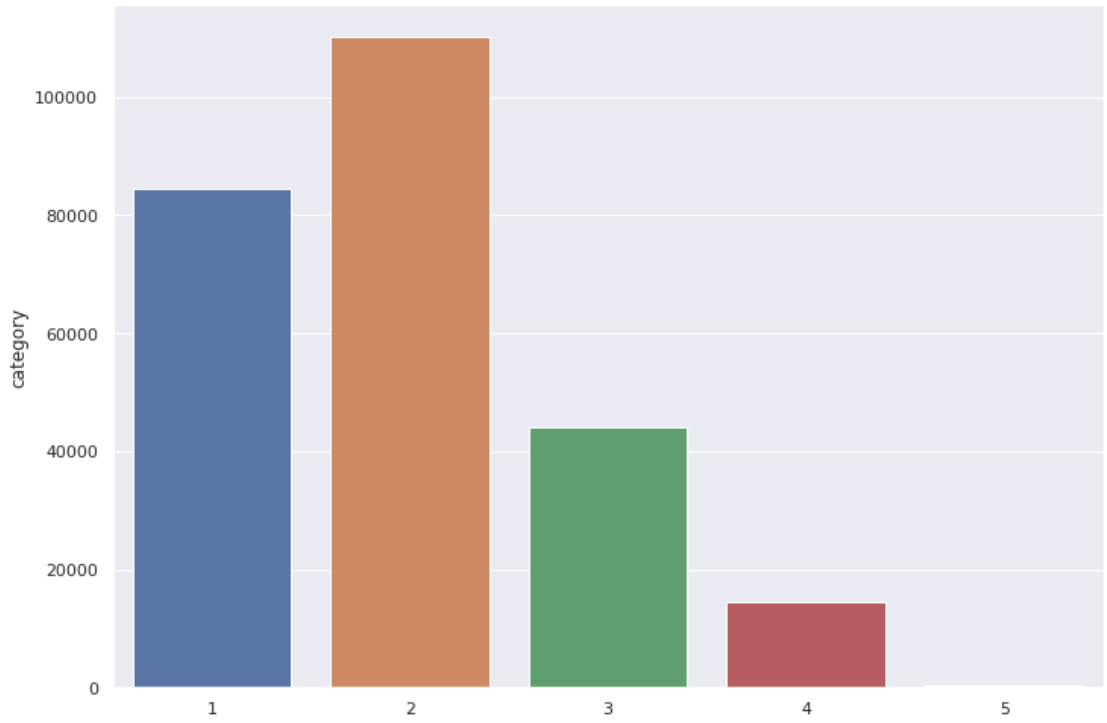
```
Out[11]:   package  height                 date destination_agent  \
         2     Box    16.7  2018-08-30 00:00:00
         3     Box    10.8  2018-08-30 00:00:00
         4     Box    10.6  2018-08-30 00:00:00

                         shipper category
         2  Ligowave Networks Inc        1
         3  Ligowave Networks Inc        1
         4  Ligowave Networks Inc        1
```

Data representation

```
In [12]: sns.set(rc={'figure.figsize':(11.7,8.27)})
         sns.barplot(x=df.category.value_counts().index, y=df.category.value_counts())
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8859823a50>
```



Percentage Values

```
In [13]: category_1 = float((df.loc[df['category'] == 1].shape)[0]) / float(261532) * 100
         print(category_1)
```

```
32.3061805056
```

```
In [14]: category_2 = float((df.loc[df['category'] == 2].shape)[0]) / float(261532) * 100
         print(category_2)
```

```
42.1355704082
```

```
In [15]: category_3 = float((df.loc[df['category'] == 3].shape)[0]) / float(261532) * 100
         print(category_3)
```

```
16.8465044431
```

```
In [16]: category_4 = float((df.loc[df['category'] == 4].shape)[0]) / float(261532) * 100
         print(category_4)
```

```
5.52972485203
```

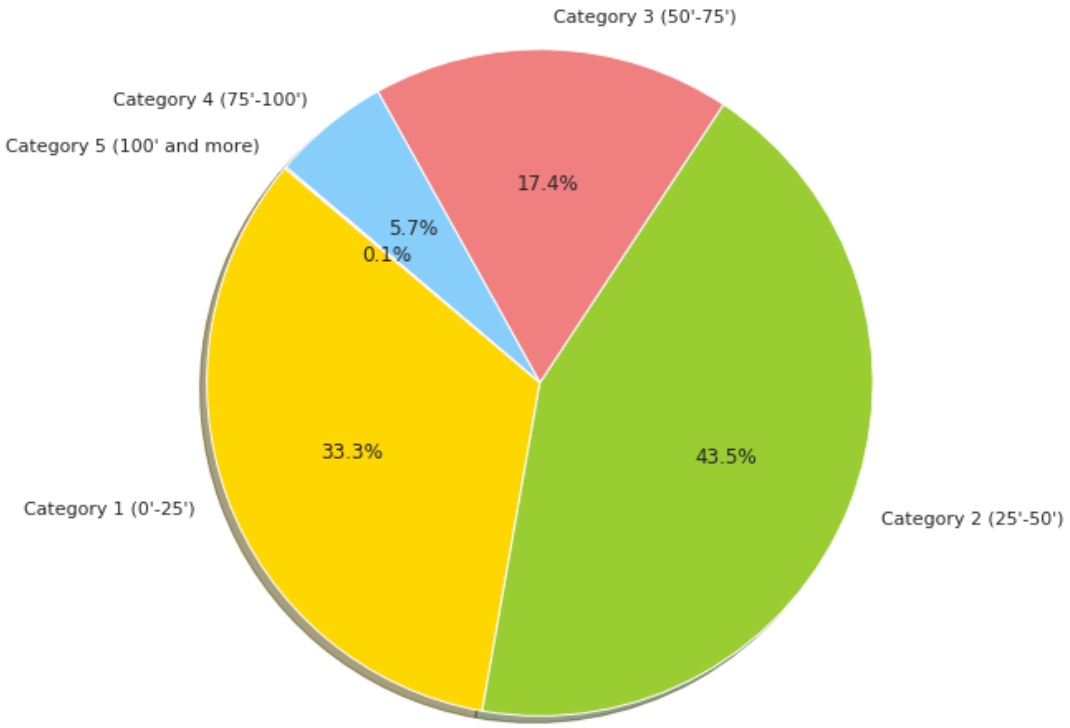
```
In [17]: category_5 = float((df.loc[df['category'] == 5].shape)[0]) / float(261532) * 100
         print(category_5)
```

```
0.0848844500864
```

```
In [18]: labels = "Category 1 (0'-25')", "Category 2 (25'-50')", "Category 3 (50'-75')","Categor
         sizes = [category_1, category_2, category_3, category_4, category_5]
         colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'red']

         # Plot
         plt.pie(sizes, labels=labels, colors=colors,
                 autopct='%1.1f%%', shadow=True, startangle=140)

         plt.axis('equal')
         plt.show()
```



Since category 5 is so small, it is a good idea to mix it with category 4

```
In [19]: for i, row in df.iterrows():
             if row['category'] == 5 :
                 df.at[i, 'category'] = 4
```

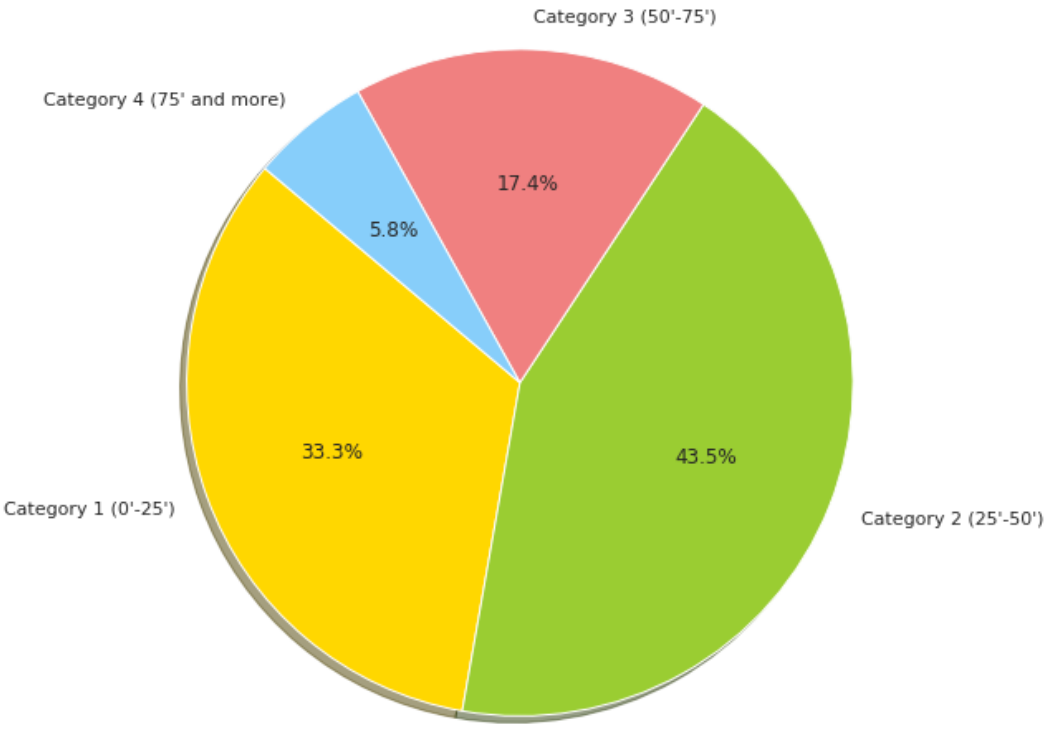
```
In [20]: category_4 = float((df.loc[df['category'] == 4].shape)[0]) / float(261532) * 100
         print(category_4)
```

```
5.61460930211
```

```
In [21]: labels = "Category 1 (0'-25')", "Category 2 (25'-50')", "Category 3 (50'-75')","Categor
         sizes = [category_1, category_2, category_3, category_4]
         colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']

         # Plot
         plt.pie(sizes, labels=labels, colors=colors,
                 autopct='%1.1f%%', shadow=True, startangle=140)

         plt.axis('equal')
         plt.show()
```



Distributing pallet positions

```
In [22]: total_positions = 198
         total_height = 169
         margin_of_error = 5
         rack_positions_by_floor = 2
```

```
In [23]: positions_for_category_1 = int(total_positions * 0.333)
         positions_for_category_2 = int(total_positions * 0.435)
         positions_for_category_3 = int(total_positions * 0.174)
```

```
        positions_for_category_4 = int(total_positions * 0.058)
        print(positions_for_category_1, positions_for_category_2, positions_for_category_3, pos
```

```
(65, 86, 34, 11)
```

Results for category 1 (25′ and bellow)

```
In [24]: floors_by_rack_category_1 = total_height / (margin_of_error + 25)
         print floors_by_rack_category_1
```

```
5
```

```
In [25]: positions_by_rack_category_1 = floors_by_rack_category_1 * 2
         print positions_by_rack_category_1
```

```
10
```

Results for category 2 (25′ - 50′)

```
In [26]: floors_by_rack_category_2 = total_height / (margin_of_error + 50)
         print floors_by_rack_category_2
```

```
3
```

```
In [27]: positions_by_rack_category_2 = floors_by_rack_category_2 * 2
         print positions_by_rack_category_2
```

```
6
```

Results for category 3 and 4 (50′-75′ and 75 and more)

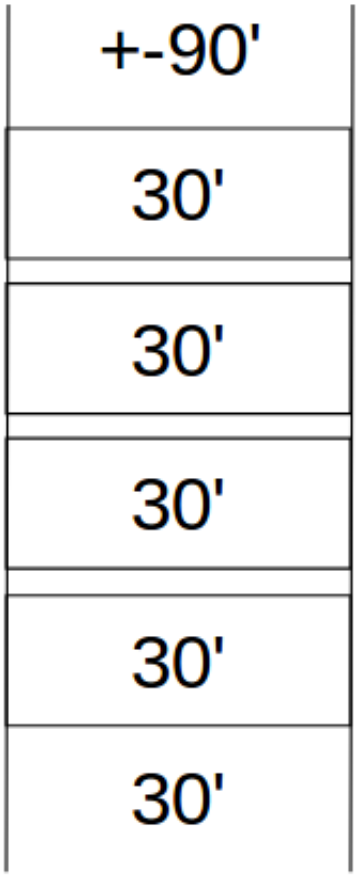
```
In [28]: floors_by_rack_category_3 = total_height / (margin_of_error + 50)
         print floors_by_rack_category_3
```

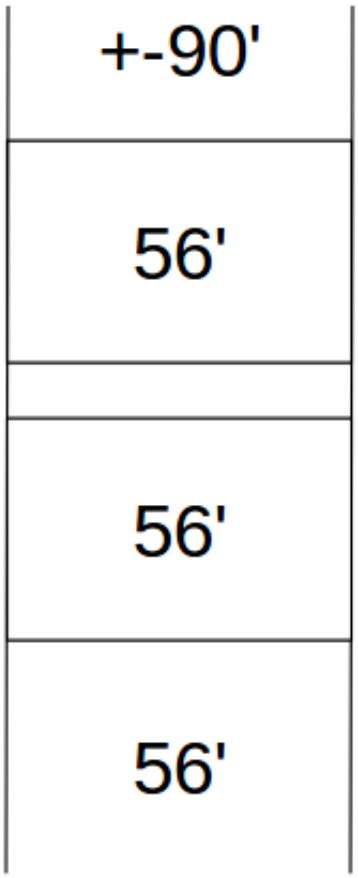
```
3
```

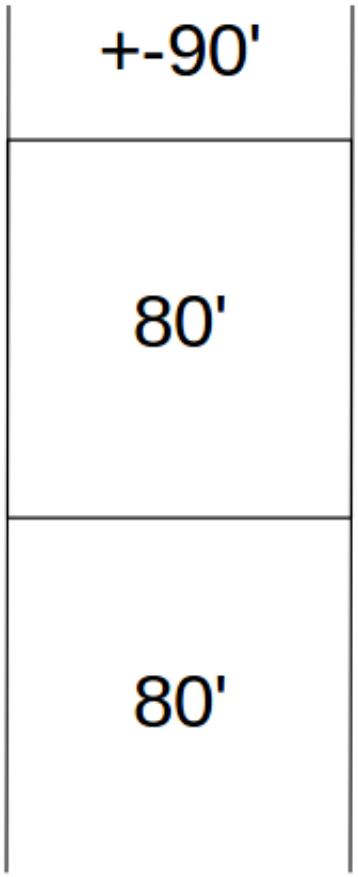
```
In [29]: positions_by_rack_category_3 = floors_by_rack_category_3 * 2
         print positions_by_rack_category_3
```

```
6
```

Final recommendation
This is based on a 198 racks setup (what we currently have in our warehouse)

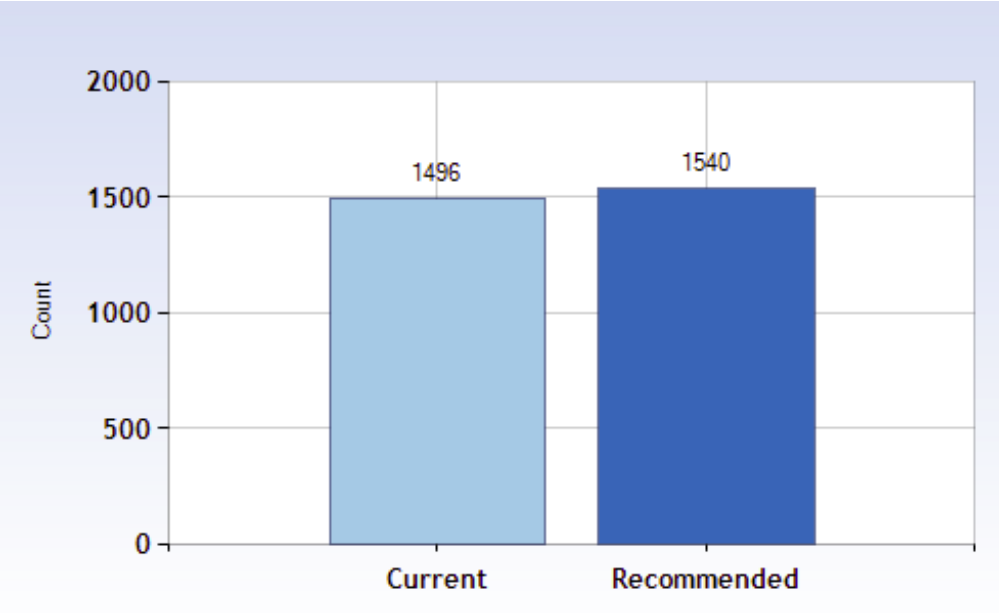

+-90'
30'
30'
30'
30'
30'


+-90'
56'
56'
56'


+-90'
80'
80'

Height Range	Category	Number of racks	Number of positions	Statistic Recomendation
0′ - 25′	Category 1	40	440	396
25′ - 50′	Category 2	48	528	528
50′ - 75′	Category 3	24	264	220
75′ and more	Category 4	0	396	396

Current distribution vs recommendation



- There is an increase in the amount of available positions.
- Even though the increase in number of positions isn't very relevant, the recommended distribution is more likely to match the heights of the actual cargo, taking better adventage of the available spaces.
- The ammount of spaces for category 4 (top of the racks), is way higher to the needed spaces, this gives us a good capacity for handling the margin of error.

In [ ]: