In [1]:

```
# Basic Libraries
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt # we only need pyplot
sb.set() # set the default Seaborn style for graphics
```

In [2]:

```
data = pd.read_csv('./files/Product_Survey_Results_Cleaned 18022021.csv', header = 0).set_index('Name')
data.head()
```

Out[2]:

	Brand	Type of toothbrush	Rate current price of product	How much you would pay	Rate the importance of Grip	Rate the importance of Weight	Rate the importance of On/Off	Rate the importance of Clean/Rinse	Rate the importance of Vibration	Rate the importance of Waterproof	...	Stand	He stora
Name													
Respondent 1	Oral-B	Rechargeable	4	10.0	4	3	5	5	4	5	...	1	
Respondent 2	Name brand	Manual	4	4.0	4	5	5	5	3	5	...	0	
Respondent 3	Generic	Manual	4	10.0	1	3	3	4	5	5	...	1	
Respondent 4	Name brand	Manual	4	8.0	4	3	4	5	4	5	...	1	
Respondent 1 - S	Oral-B	Manual	4	4.5	2	2	3	1	3	4	...	0	

5 rows × 60 columns

In [6]:

```
NumData = data.copy()
CatData = data.copy()
i = 0
for col in data.columns:
    if i < 3 or (i > 3 and i < 26) or (i <29 and i>26) or (i >44 and i<61):
        data[col] = data[col].astype('category')

    else:
        data[col]= data[col].astype('float64')
    i+=1

data.info()
NumData = NumData.select_dtypes(include=['float64'])
CatData = CatData.select_dtypes(include=['category'])
NumData.info()
CatData.info()

# Import LinearRegression model from Scikit-Learn
from sklearn.linear_model import LinearRegression

# Create a Linear Regression object
linreg = LinearRegression()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 21 entries, Respondent 1 to Respondent 8
Data columns (total 60 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
 0   Brand                                       21 non-null     category
 1   Type of toothbrush                          21 non-null     category
 2   Rate current price of product               21 non-null     category
 3   How much you would pay                      21 non-null     float64
 4   Rate the importance of Grip                 21 non-null     category
 5   Rate the importance of Weight               21 non-null     category
 6   Rate the importance of On/Off               21 non-null     category
 7   Rate the importance of Clean/Rinse          21 non-null     category
 8   Rate the importance of Vibration            21 non-null     category
 9   Rate the importance of Waterproof           21 non-null     category
 10  Rate the importance of Travel               21 non-null     category
 11  Rate the importance of Replace Battery      21 non-null     category
 12  Rate the importance of Replace Brush Head   21 non-null     category
 13  Rate the importance of $ Replacements       21 non-null     category
 14  Rate the importance of Avail in Area        21 non-null     category
 15  Rate the importance of Long Battery Life    21 non-null     category
 16  Rate the importance of Technology           21 non-null     category
 17  Rate the importance of Looks Cool           21 non-null     category
 18  Rate the importance of Distinguishable      21 non-null     category
 19  Rate the importance of Match Décor          21 non-null     category
 20  Rate the importance of Easy to Store        21 non-null     category
 21  Rate the importance of Small Space          21 non-null     category
 22  Rate the importance of Easy to hold         21 non-null     category
 23  Rate the importance of Toothbrush Sized     21 non-null     category
 24  Rate the importance of Packaging            21 non-null     category
 25  Rate the importance of Battery life from 1-3  21 non-null   category
 26  How much for rechargeable?                  21 non-null     float64
 27  Rate the Look of product from 1-3           21 non-null     category
 28  Would unique colors and patterns improve product?  21 non-null  category
 29  How much more would you pay for cool style?  21 non-null    float64
```

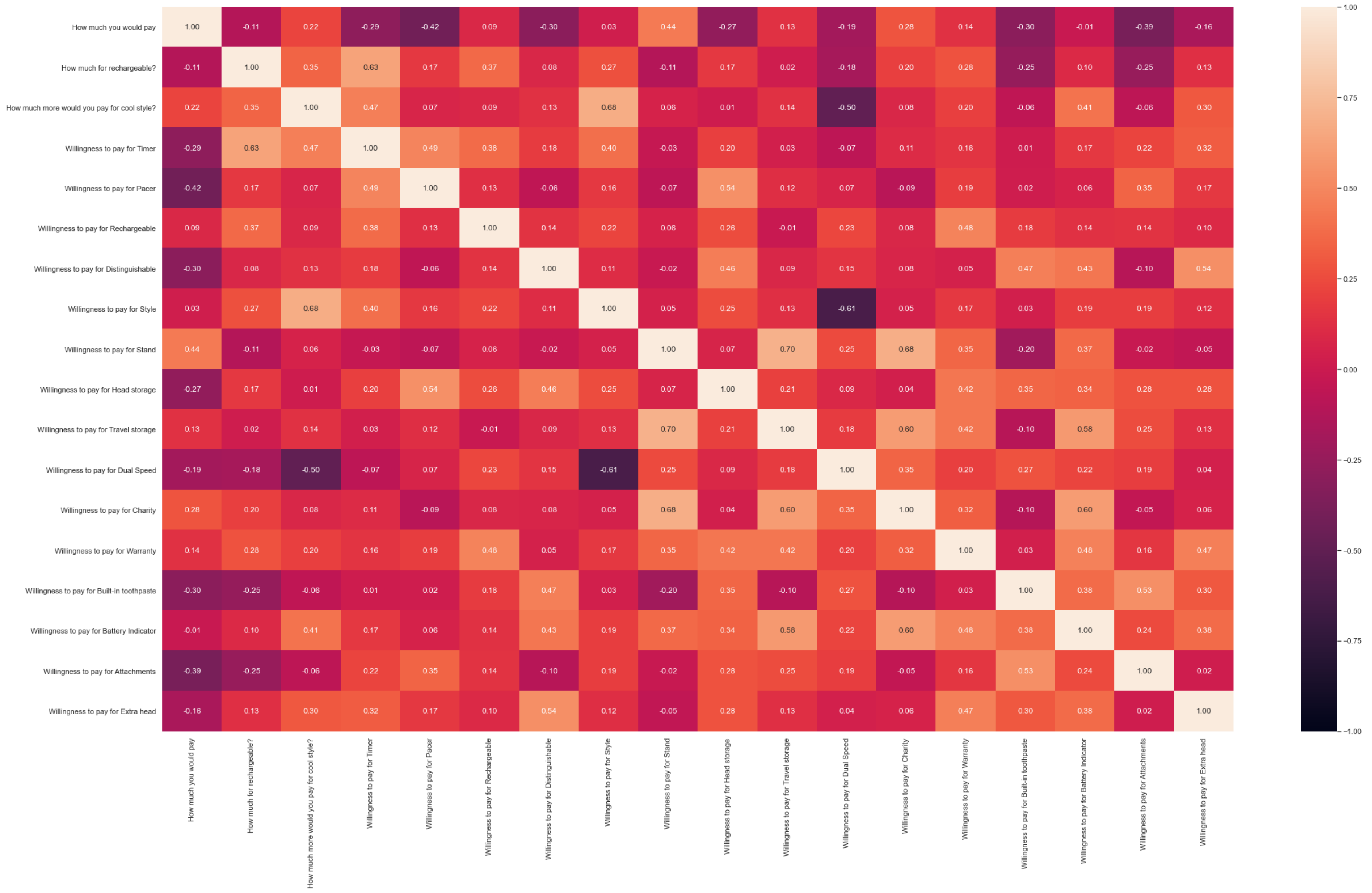
```
 30  Willingness to pay for Timer                      21 non-null     float64 
 31  Willingness to pay for Pacer                      21 non-null     float64 
 32  Willingness to pay for Rechargeable               21 non-null     float64 
 33  Willingness to pay for Distinguishable            21 non-null     float64 
 34  Willingness to pay for Style                      21 non-null     float64 
 35  Willingness to pay for Stand                      21 non-null     float64 
 36  Willingness to pay for Head storage               21 non-null     float64 
 37  Willingness to pay for Travel storage             21 non-null     float64 
 38  Willingness to pay for Dual Speed                 21 non-null     float64 
 39  Willingness to pay for Charity                    21 non-null     float64 
 40  Willingness to pay for Warranty                   21 non-null     float64 
 41  Willingness to pay for Built-in toothpaste        21 non-null     float64 
 42  Willingness to pay for Battery Indicator         21 non-null     float64 
 43  Willingness to pay for Attachments               21 non-null     float64 
 44  Willingness to pay for Extra head                21 non-null     float64 
 45  Timer                                             21 non-null     category
 46  Pacer                                             21 non-null     category
 47  Rechargeable                                      21 non-null     category
 48  Distinguishable                                   21 non-null     category
 49  Style                                             21 non-null     category
 50  Stand                                             21 non-null     category
 51  Head storage                                      21 non-null     category
 52  Travel storage                                    21 non-null     category
 53  Dual Speed                                        21 non-null     category
 54  Charity                                           21 non-null     category
 55  Warranty                                          21 non-null     category
 56  Built-in toothpaste                               21 non-null     category
 57  Battery Indicator                                 21 non-null     category
 58  Attachments                                       21 non-null     category
 59  Extra head                                        21 non-null     category
dtypes: category(42), float64(18)
memory usage: 11.2+ KB
<class 'pandas.core.frame.DataFrame'>
Index: 21 entries, Respondent 1 to Respondent 8
Data columns (total 18 columns):
 #   Column                                      Non-Null Count  Dtype  
---  ------                                      --------------  -----  
 0   How much you would pay                      21 non-null     float64
 1   How much for rechargeable?                  21 non-null     float64
 2   How much more would you pay for cool style?  21 non-null     float64
 3   Willingness to pay for Timer                21 non-null     float64
 4   Willingness to pay for Pacer                21 non-null     float64
 5   Willingness to pay for Rechargeable         21 non-null     float64
 6   Willingness to pay for Distinguishable      21 non-null     float64
 7   Willingness to pay for Style                21 non-null     float64
 8   Willingness to pay for Stand                21 non-null     float64
 9   Willingness to pay for Head storage         21 non-null     float64
 10  Willingness to pay for Travel storage       21 non-null     float64
 11  Willingness to pay for Dual Speed           21 non-null     float64
 12  Willingness to pay for Charity              21 non-null     float64
 13  Willingness to pay for Warranty             21 non-null     float64
 14  Willingness to pay for Built-in toothpaste  21 non-null     float64
 15  Willingness to pay for Battery Indicator    21 non-null     float64
 16  Willingness to pay for Attachments          21 non-null     float64
 17  Willingness to pay for Extra head           21 non-null     float64
dtypes: float64(18)
memory usage: 3.1+ KB
<class 'pandas.core.frame.DataFrame'>
Index: 21 entries, Respondent 1 to Respondent 8
Data columns (total 42 columns):
 #   Column                                            Non-Null Count  Dtype   
---  ------                                            --------------  -----   
 0   Brand                                             21 non-null     category
 1   Type of toothbrush                                21 non-null     category
 2   Rate current price of product                     21 non-null     category
 3   Rate the importance of Grip                       21 non-null     category
 4   Rate the importance of Weight                     21 non-null     category
 5   Rate the importance of On/Off                     21 non-null     category
 6   Rate the importance of Clean/Rinse                21 non-null     category
 7   Rate the importance of Vibration                  21 non-null     category
 8   Rate the importance of Waterproof                 21 non-null     category
 9   Rate the importance of Travel                     21 non-null     category
 10  Rate the importance of Replace Battery            21 non-null     category
 11  Rate the importance of Replace Brush Head         21 non-null     category
 12  Rate the importance of $ Replacements             21 non-null     category
 13  Rate the importance of Avail in Area              21 non-null     category
 14  Rate the importance of Long Battery Life          21 non-null     category
 15  Rate the importance of Technology                 21 non-null     category
 16  Rate the importance of Looks Cool                 21 non-null     category
 17  Rate the importance of Distinguishable            21 non-null     category
 18  Rate the importance of Match Décor                21 non-null     category
 19  Rate the importance of Easy to Store              21 non-null     category
 20  Rate the importance of Small Space                21 non-null     category
 21  Rate the importance of Easy to hold               21 non-null     category
 22  Rate the importance of Toothbrush Sized           21 non-null     category
 23  Rate the importance of Packaging                  21 non-null     category
 24  Rate the importance of Battery life from 1-3      21 non-null     category
 25  Rate the Look of product from 1-3                 21 non-null     category
 26  Would unique colors and patterns improve product?  21 non-null     category
 27  Timer                                             21 non-null     category
 28  Pacer                                             21 non-null     category
 29  Rechargeable                                      21 non-null     category
 30  Distinguishable                                   21 non-null     category
 31  Style                                             21 non-null     category
 32  Stand                                             21 non-null     category
 33  Head storage                                      21 non-null     category
 34  Travel storage                                    21 non-null     category
 35  Dual Speed                                        21 non-null     category
 36  Charity                                           21 non-null     category
 37  Warranty                                          21 non-null     category
```

```
 38  Built-in toothpaste                           21 non-null     category
 39  Battery Indicator                             21 non-null     category
 40  Attachments                                   21 non-null     category
 41  Extra head                                    21 non-null     category
dtypes: category(42)
memory usage: 8.3+ KB
```

In [7]:

```python
NumData.corr()
f = plt.figure(figsize=(36, 20))
sb.heatmap(NumData.corr(), vmin = -1, vmax = 1, annot = True, fmt=".2f")
```

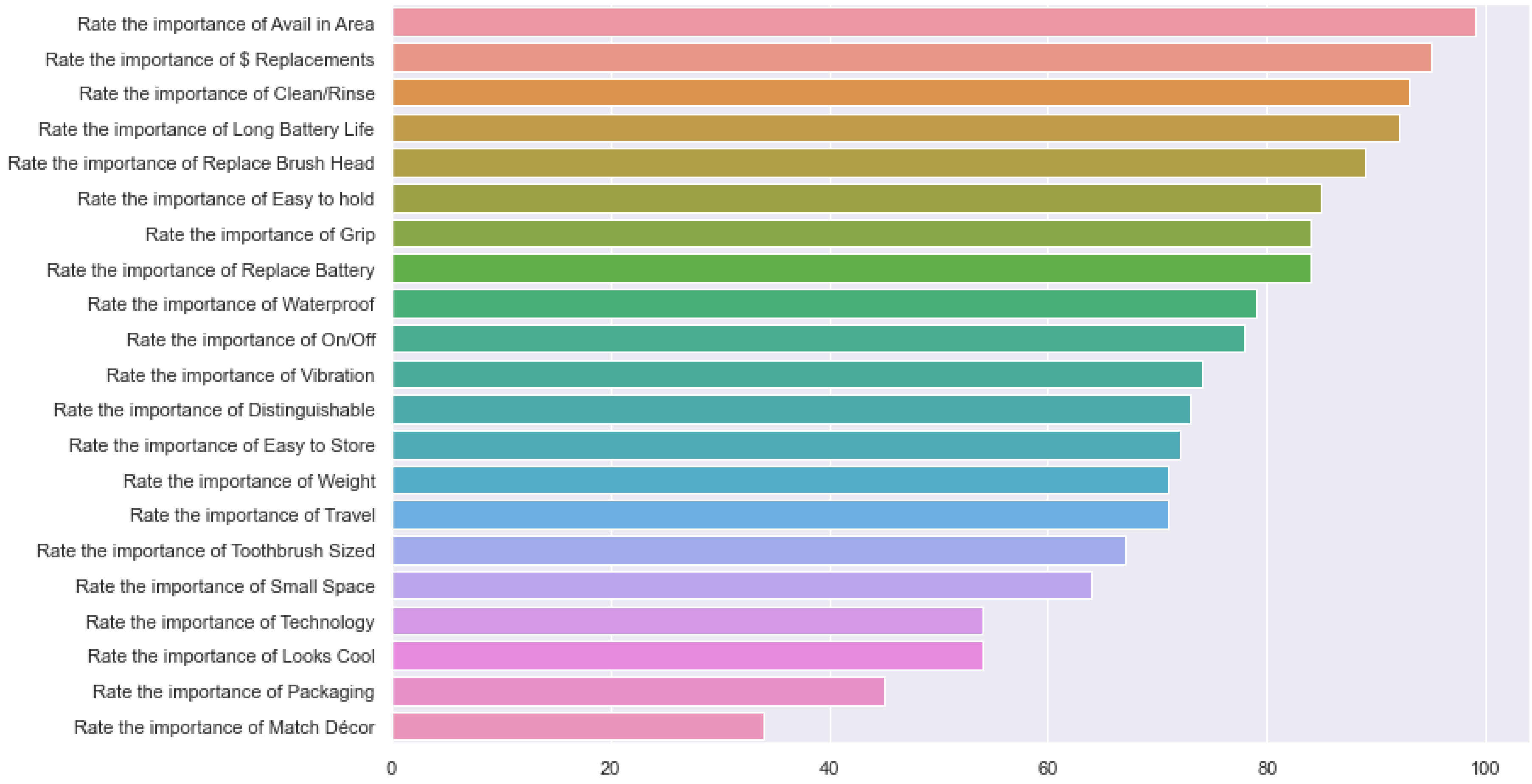
Out[7]: <AxesSubplot:>



In [106…

```python
Ratings = pd.DataFrame(CatData[['Rate the importance of Grip','Rate the importance of Weight','Rate the importance of O
                               'Rate the importance of Clean/Rinse','Rate the importance of Vibration','Rate the import
                               'Rate the importance of Travel','Rate the importance of Replace Battery','Rate the impor
                               'Rate the importance of $ Replacements','Rate the importance of Avail in Area',
                               'Rate the importance of Long Battery Life','Rate the importance of Technology',
                               'Rate the importance of Looks Cool','Rate the importance of Distinguishable',
                               'Rate the importance of Match Décor','Rate the importance of Easy to Store',
                               'Rate the importance of Small Space','Rate the importance of Easy to hold',
                               'Rate the importance of Toothbrush Sized','Rate the importance of Packaging']])
Ratings = Ratings.astype('int64')

rating_list = []
import operator
for col in Ratings.columns:
    rating_list.append(col)

rating_total_dict={}
for col in rating_list:
    total = Ratings[col].sum()
    rating_total_dict[col] = total

rating_total_dict = dict(sorted(rating_total_dict.items(),
                               key=operator.itemgetter(1),
                               reverse=True))

f = plt.figure(figsize = (12,8))
sb.barplot(y=list(rating_total_dict.keys()),x=list(rating_total_dict.values()),orient = 'h')
```

Out[106… <AxesSubplot:>

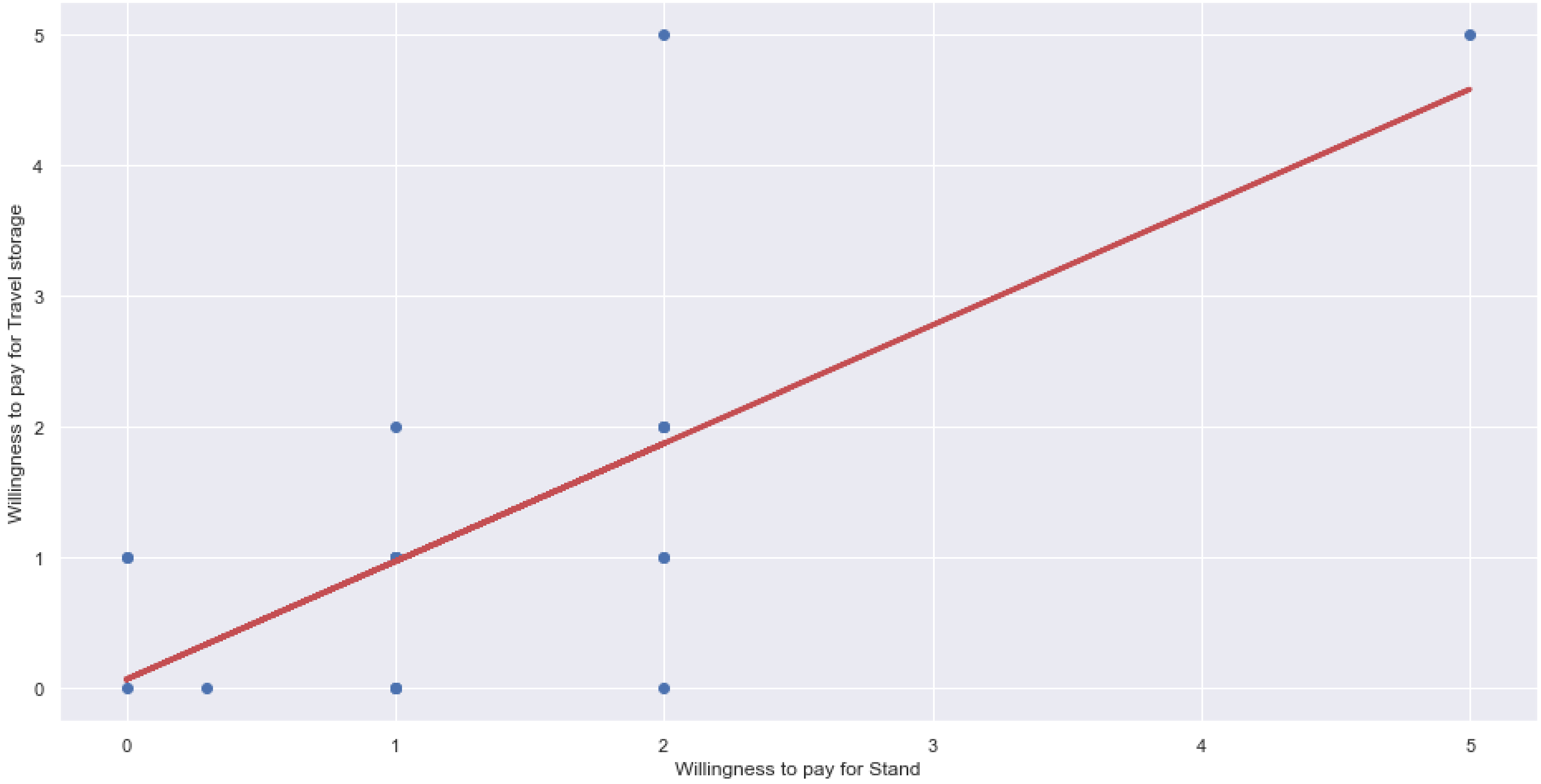


In [9]:

```
# Mean Squared Error (MSE)
def mean_sq_err(actual, predicted):
    '''Returns the Mean Squared Error of actual and predicted values'''
    return np.mean(np.square(np.array(actual) - np.array(predicted)))
```

In [10]:

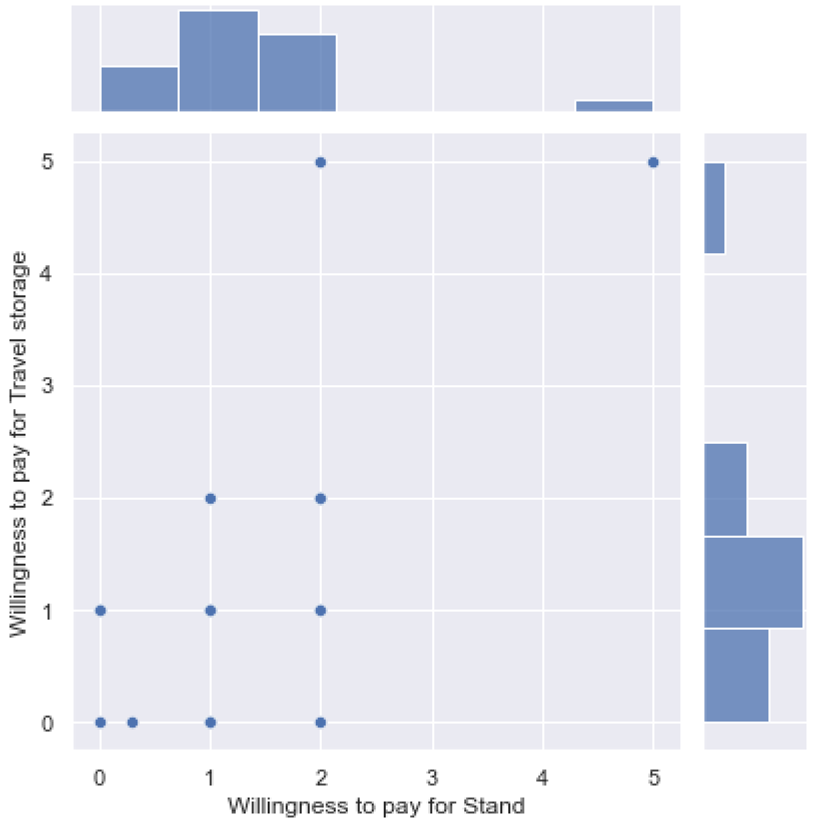
```
num1 = NumData[['Willingness to pay for Stand','Willingness to pay for Travel storage']]
W_stand = pd.DataFrame(NumData['Willingness to pay for Stand'])
W_travel_storage = pd.DataFrame(NumData['Willingness to pay for Travel storage'])

# Train the Linear Regression model
linreg.fit(W_stand,W_travel_storage)
# Formula for the Regression line
regline_x = W_stand
regline_y = linreg.intercept_ + linreg.coef_ * W_stand

# num1.info()
# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.plot(regline_x, regline_y, 'r-', linewidth = 3)
plt.scatter(W_stand, W_travel_storage)
plt.xlabel("Willingness to pay for Stand")
plt.ylabel("Willingness to pay for Travel storage")
plt.show()

# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(W_stand, W_travel_storage))

# Predict Total values corresponding to HP Train
W_travel_storage_pred = linreg.predict(W_stand)


mse = mean_sq_err(W_travel_storage, W_travel_storage_pred)
print("Mean Squared Error (MSE) \t:", mse)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

f = plt.figure(figsize=(10,6))
sb.jointplot(data = num1, x = "Willingness to pay for Stand", y = "Willingness to pay for Travel storage")
```




```
Explained Variance (R^2)         : 0.4841523201218223
Mean Squared Error (MSE)         : 0.9895853450724227
Root Mean Squared Error (RMSE)   : 0.994779043341999
```

Out[10]:
```
<seaborn.axisgrid.JointGrid at 0x182da81c0>
```

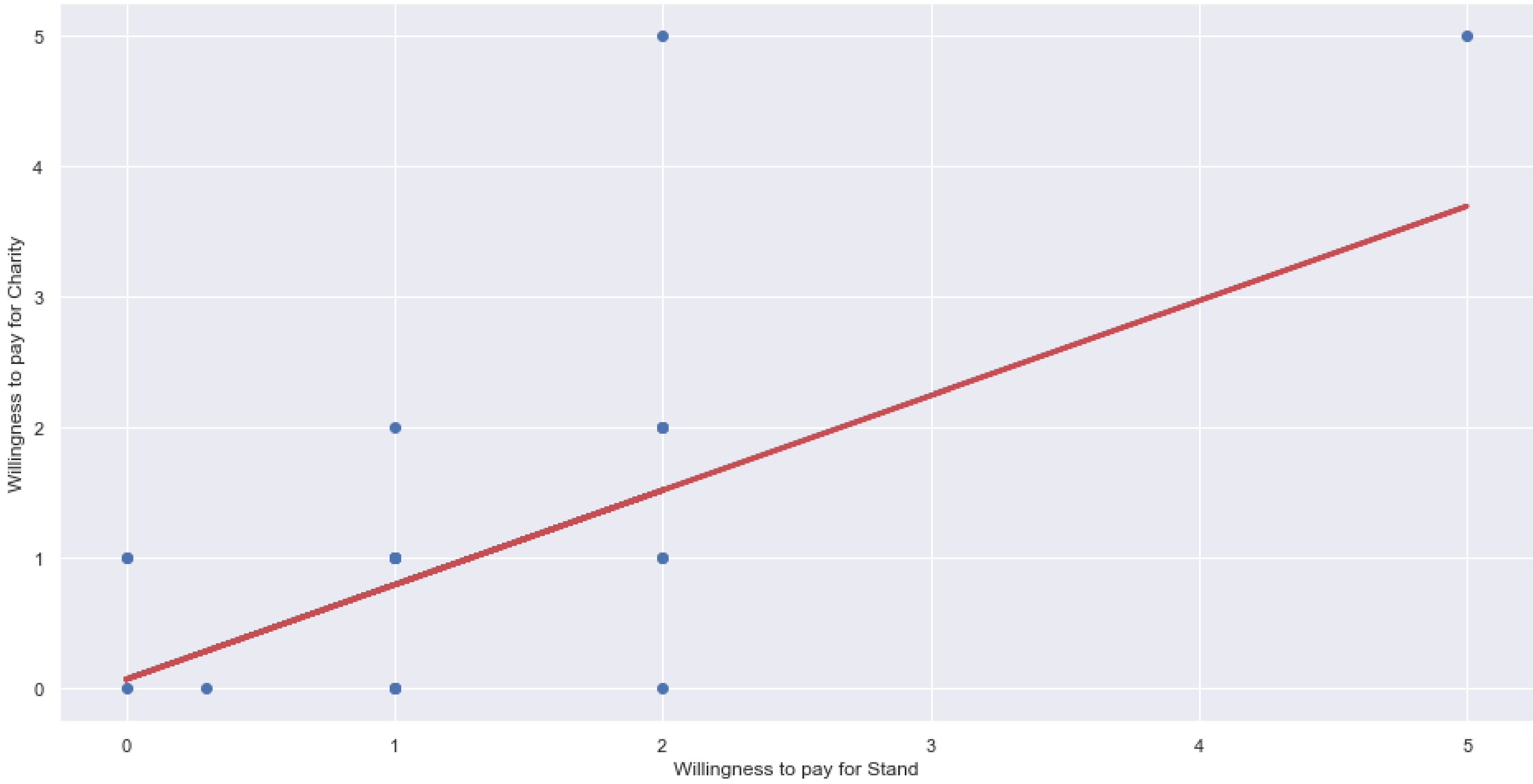
```
<Figure size 720x432 with 0 Axes>
```



More willing to pay for stand also more likely to pay for travel storage and vice-versa, positive correlation

In [11]:
```
num1 = NumData[['Willingness to pay for Stand','Willingness to pay for Charity']]
W_Charity = pd.DataFrame(NumData['Willingness to pay for Charity'])

# Train the Linear Regression model
linreg.fit(W_stand,W_Charity)
# Formula for the Regression line
regline_x = W_stand
regline_y = linreg.intercept_ + linreg.coef_ * W_stand

# num1.info()
# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.plot(regline_x, regline_y, 'r-', linewidth = 3)
plt.scatter(W_stand, W_travel_storage)
plt.xlabel("Willingness to pay for Stand")
plt.ylabel("Willingness to pay for Charity")
plt.show()

# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(W_stand, W_Charity))

# Predict Total values corresponding to HP Train
W_Charity_pred = linreg.predict(W_stand)


mse = mean_sq_err(W_Charity, W_Charity_pred)
print("Mean Squared Error (MSE) \t:", mse)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))
```

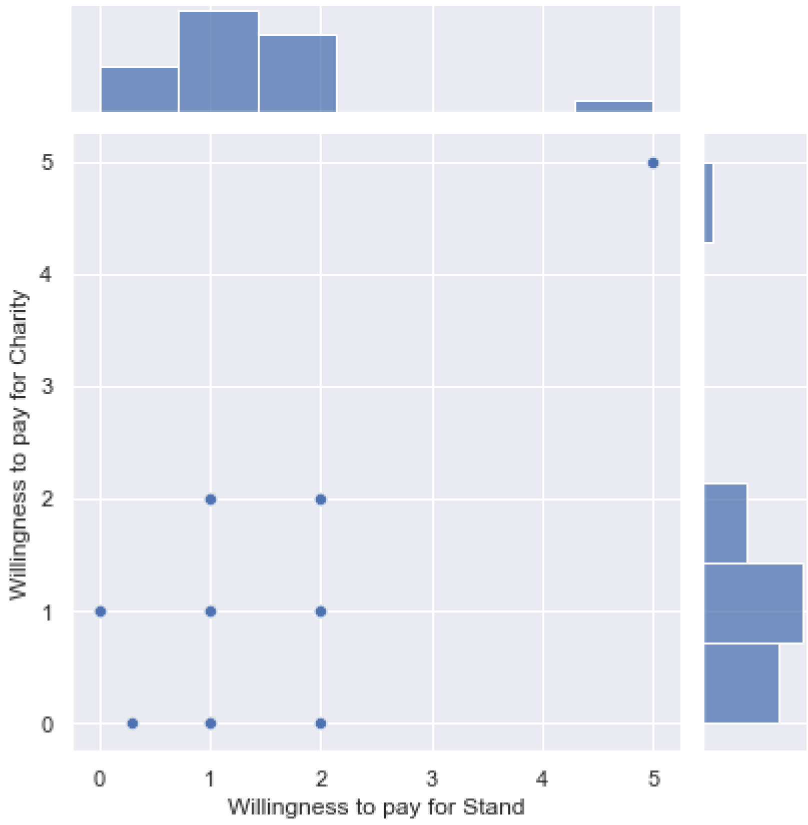
```
f = plt.figure(figsize=(10,6))
sb.jointplot(data = num1, x = "Willingness to pay for Stand", y = "Willingness to pay for Charity")
```




```
Explained Variance (R^2)        : 0.46641479743447434
Mean Squared Error (MSE)        : 0.6848281738142573
Root Mean Squared Error (RMSE)  : 0.8275434573569278
```

Out[11]: <seaborn.axisgrid.JointGrid at 0x182fd1ee0>

<Figure size 720x432 with 0 Axes>



num2 = NumData[['Willingness to pay for Style','How much more would you pay for cool style?']]

num2.info()

f = plt.figure(figsize=(10,6)) sb.jointplot(data = num2, x = "Willingness to pay for Style", y = "How much more would you pay for cool style?")

W_Style = pd.DataFrame(NumData["Willingness to pay for Style"]) W_C_Style = pd.DataFrame(NumData["How much more would you pay for cool style?"])

Train the Linear Regression model

linreg.fit(W_Style,W_C_Style)

Formula for the Regression line

regline_x = W_Style regline*y = linreg.intercept* + linreg.coef_ * W_Style

num1.info()

Plot the Linear Regression line

f = plt.figure(figsize=(16, 8)) plt.plot(regline_x, regline_y, 'r-', linewidth = 3) plt.scatter(W_Style, W_C_Style) plt.xlabel("Willingness to pay for Style") plt.ylabel("How much more would you pay for cool style?") plt.show()

Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg.score(W_Style, W_C_Style)) W_C_Style_pred = linreg.predict(W_Style) mse = mean_sq_err(W_C_Style, W_C_Style_pred) print("Mean Squared Error (MSE) \t:", mse) print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

num3 = NumData[['Willingness to pay for Charity','Willingness to pay for Travel storage']]

num3.info()

W_Charity = pd.DataFrame(NumData['Willingness to pay for Charity'])

f = plt.figure(figsize=(10,6)) sb.jointplot(data = num3, x = "Willingness to pay for Charity", y = "Willingness to pay for Travel storage")

Train the Linear Regression model

linreg.fit(W_Charity,W_travel_storage)

Formula for the Regression line

regline_x = W_Charity regline*y = linreg.intercept* + linreg.coef_ * W_Charity

Plot the Linear Regression line

f = plt.figure(figsize=(16, 8)) plt.plot(regline_x, regline_y, 'r-', linewidth = 3) plt.scatter(W_Charity, W_travel_storage) plt.show()

Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg.score(W_Charity, W_travel_storage)) W_travel_storage_pred_C = linreg.predict(W_Charity) mse = mean_sq_err(W_travel_storage, W_travel_storage_pred_C) print("Mean Squared Error (MSE) \t:", mse) print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

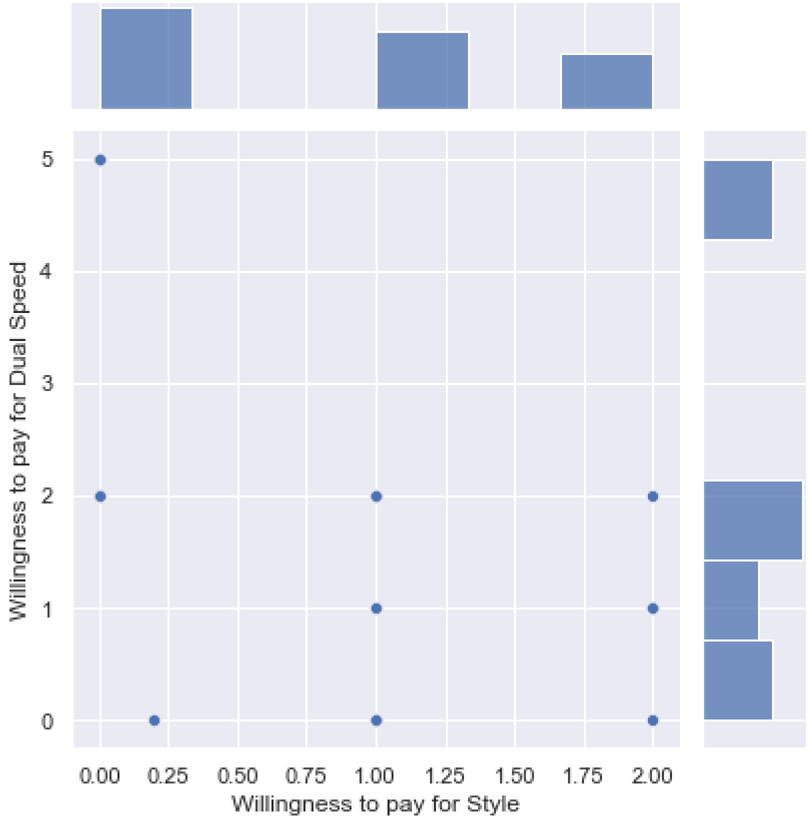
Speed is functionality while style is more aesthetics. Negatively correlated

In [12]:

```
num3 = NumData[['Willingness to pay for Style','Willingness to pay for Dual Speed']]
# num3.info()

f = plt.figure(figsize=(10,6))
sb.jointplot(data = num3, x = "Willingness to pay for Style", y = "Willingness to pay for Dual Speed")

W_Dual_Speed = pd.DataFrame(NumData['Willingness to pay for Dual Speed'])
# Train the Linear Regression model
linreg.fit(W_Style,W_Dual_Speed)
# Formula for the Regression line
regline_x = W_Style
regline_y = linreg.intercept_ + linreg.coef_ * W_Style

# Plot the Linear Regression line
f = plt.figure(figsize=(16, 8))
plt.plot(regline_x, regline_y, 'r-', linewidth = 3)
plt.xlabel("Willingness to pay for Style")
plt.ylabel("Willingness to pay for Dual Speed")
plt.scatter(W_Style, W_Dual_Speed)
plt.show()

# Explained Variance (R^2)
print("Explained Variance (R^2) \t:", linreg.score(W_Style, W_Dual_Speed))
W_Dual_Speed_pred = linreg.predict(W_Style)
mse = mean_sq_err(W_Dual_Speed, W_Dual_Speed_pred)
print("Mean Squared Error (MSE) \t:", mse)
print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

```
<ipython-input-12-aaf72a28cfbc> in <module>
      7 W_Dual_Speed = pd.DataFrame(NumData['Willingness to pay for Dual Speed'])
      8 # Train the Linear Regression model
----> 9 linreg.fit(W_Style,W_Dual_Speed)
     10 # Formula for the Regression line
     11 regline_x = W_Style

NameError: name 'W_Style' is not defined
<Figure size 720x432 with 0 Axes>
```



In [13]:

```
CatData.corr()
f = plt.figure(figsize=(30, 10))
sb.heatmap(CatData.groupby(['Rate the importance of Grip', 'Rate the importance of Weight']).size().unstack(),
           linewidths = 1, annot = True, annot_kws = {"size": 18}, cmap = "BuGn")
```

Out[13]: `<AxesSubplot:xlabel='Rate the importance of Weight', ylabel='Rate the importance of Grip'>`

In [73]:

```
# creating association rule for features wanted by participants
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
basket = data[['Timer','Pacer','Rechargeable','Distinguishable','Style','Stand','Head storage','Travel storage','Dual S
              ,'Built-in toothpaste','Battery Indicator','Attachments','Extra head']]#.set_index('Name')
frequent_itemsets = apriori(basket, min_support=0.05, use_colnames=True)
rules = association_rules(frequent_itemsets, metric = 'lift', min_threshold=1)
good_rules = rules[rules['confidence']> 0.4]
good_rules
# People who want a stand, would also want the battery to be rechargeable
# people who think style is important, would also want a stand
```

Out[73]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Distinguishable)	(Timer)	0.095238	0.285714	0.095238	1.000000	3.500000	0.068027	inf
5	(Style)	(Rechargeable)	0.238095	0.476190	0.142857	0.600000	1.260000	0.029478	1.309524
6	(Rechargeable)	(Stand)	0.476190	0.428571	0.285714	0.600000	1.400000	0.081633	1.428571
7	(Stand)	(Rechargeable)	0.428571	0.476190	0.285714	0.666667	1.400000	0.081633	1.571429
9	(Battery Indicator)	(Rechargeable)	0.285714	0.476190	0.142857	0.500000	1.050000	0.006803	1.047619
10	(Stand)	(Style)	0.428571	0.238095	0.190476	0.444444	1.866667	0.088435	1.371429

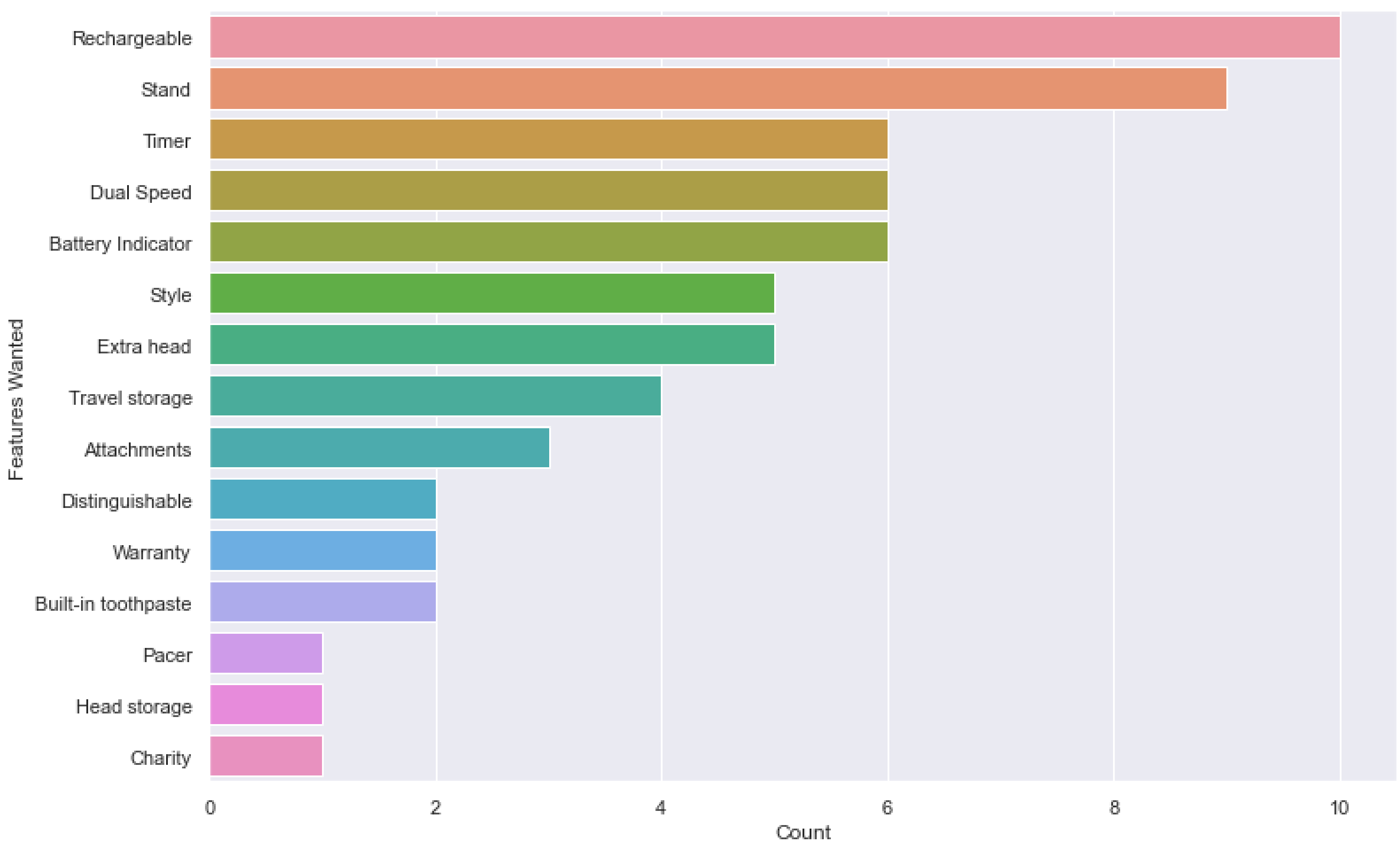
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
11	(Style)	(Stand)	0.238095	0.428571	0.190476	0.800000	1.866667	0.088435	2.857143
13	(Travel storage)	(Stand)	0.190476	0.428571	0.095238	0.500000	1.166667	0.013605	1.142857
15	(Battery Indicator)	(Stand)	0.285714	0.428571	0.142857	0.500000	1.166667	0.020408	1.142857
17	(Attachments)	(Dual Speed)	0.142857	0.285714	0.095238	0.666667	2.333333	0.054422	2.142857
18	(Warranty)	(Extra head)	0.095238	0.238095	0.095238	1.000000	4.200000	0.072562	inf
23	(Rechargeable, Style)	(Stand)	0.142857	0.428571	0.095238	0.666667	1.555556	0.034014	1.714286
24	(Stand, Style)	(Rechargeable)	0.190476	0.476190	0.095238	0.500000	1.050000	0.004535	1.047619
29	(Rechargeable, Battery Indicator)	(Stand)	0.142857	0.428571	0.095238	0.666667	1.555556	0.034014	1.714286
30	(Stand, Battery Indicator)	(Rechargeable)	0.142857	0.476190	0.095238	0.666667	1.400000	0.027211	1.571429

In [108…

```
features_wanted = basket.copy()
features_wanted = features_wanted.astype('int64')

features_list = []
import operator
for col in features_wanted.columns:
    features_list.append(col)

feature_total_dict={}
for col in features_list:
    total = features_wanted[col].sum()
    feature_total_dict[col] = total

feature_total_dict = dict(sorted(feature_total_dict.items(),
                                key=operator.itemgetter(1),
                                reverse=True))

f = plt.figure(figsize = (12,8))
plt.xlabel('Count')
plt.ylabel('Features Wanted')
sb.barplot(y=list(feature_total_dict.keys()),x=list(feature_total_dict.values()),orient = 'h')
```

Out[108…

```
<AxesSubplot:xlabel='Count', ylabel='Features Wanted'>
```



In [16]:

```
quest4 = pd.DataFrame(data[['Rate current price of product','How much you would pay']])
quest4["Rate current price of product"].value_counts()
print(quest4['How much you would pay'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "Rate current price of product", data = quest4, kind = "count")

f = plt.figure(figsize=(36,26))
sb.boxplot(x='Rate current price of product', y='How much you would pay', data=quest4)
```

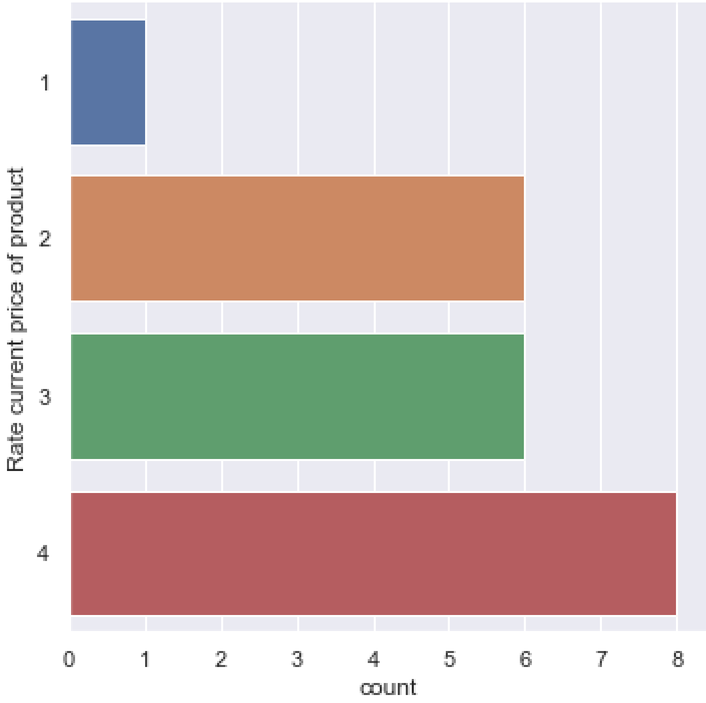
```
count    21.000000
mean      5.045714
std       2.433069
min       0.000000
25%       4.000000
50%       5.000000
75%       5.990000
```

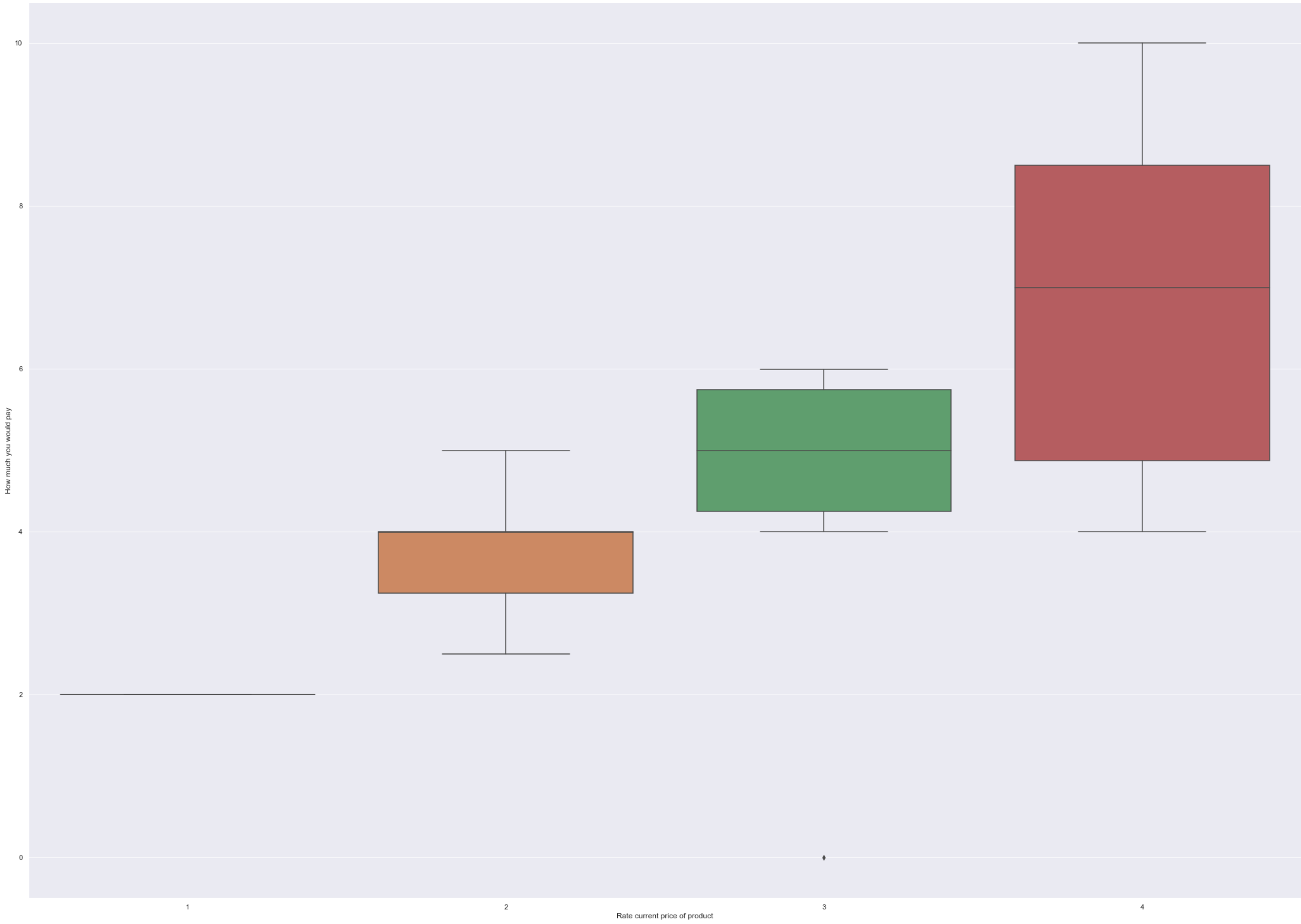
```
max      10.000000
Name: How much you would pay, dtype: float64
```

Out[16]:

```
<AxesSubplot:xlabel='Rate current price of product', ylabel='How much you would pay'>
```

```
<Figure size 2592x864 with 0 Axes>
```





In [17]:

```
quest6 = pd.DataFrame(data[['Rate the importance of Battery life from 1-3','How much for rechargeable?']])
quest6["Rate the importance of Battery life from 1-3"].value_counts()
print(quest6['How much for rechargeable?'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "Rate the importance of Battery life from 1-3", data = quest6, kind = "count")

f = plt.figure(figsize=(36,26))
sb.boxplot(x='Rate the importance of Battery life from 1-3', y='How much for rechargeable?', data=quest6)
```

```
count    21.00000
mean     13.13381
std       5.28406
min       6.99000
25%      10.99000
50%      10.99000
75%      10.99000
max      25.99000
Name: How much for rechargeable?, dtype: float64
```

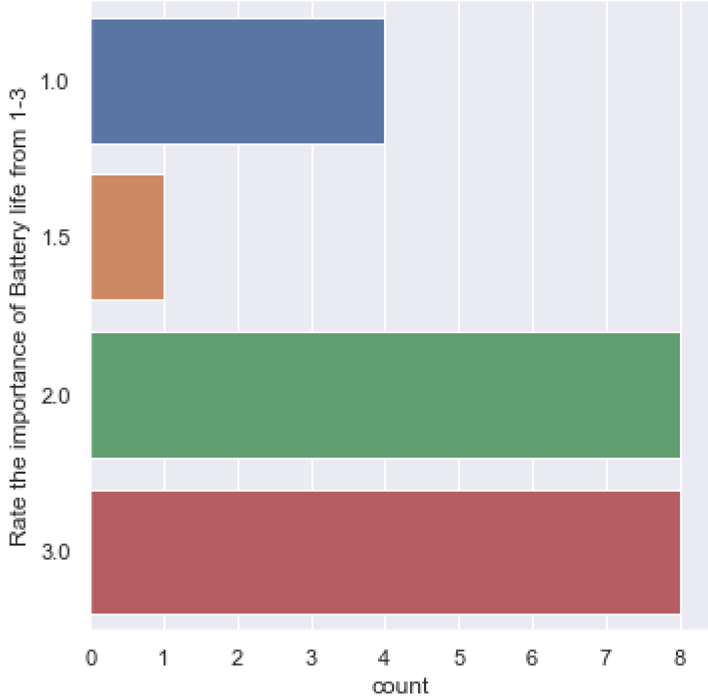
```
/usr/local/lib/python3.9/site-packages/pandas/io/formats/format.py:1405: FutureWarning: Index.ravel returning ndarray is deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
/usr/local/lib/python3.9/site-packages/pandas/io/formats/format.py:1405: FutureWarning: Index.ravel returning ndarray is
```

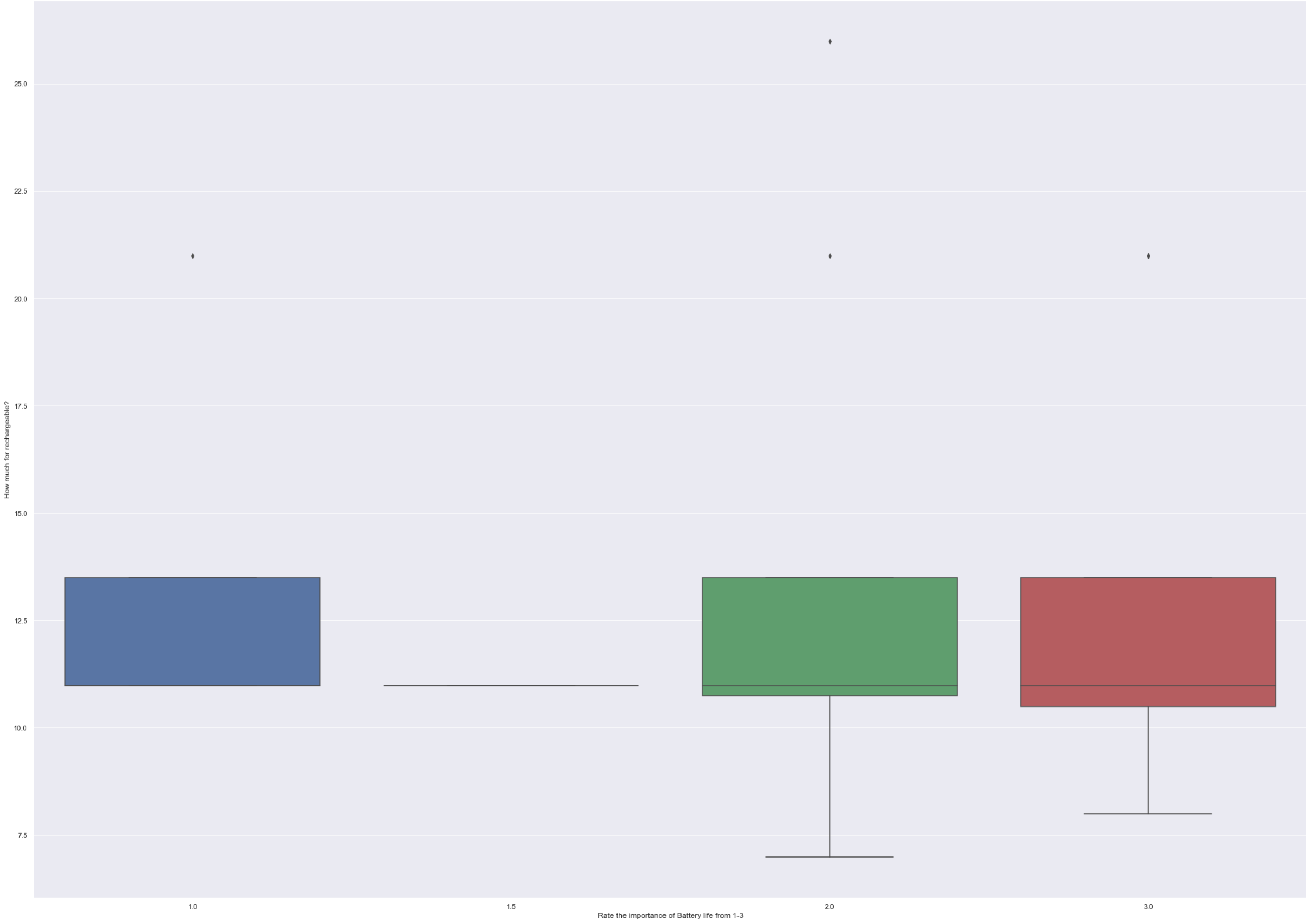
```
deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
/usr/local/lib/python3.9/site-packages/pandas/io/formats/format.py:1405: FutureWarning: Index.ravel returning ndarray is
deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```

Out[17]:

```
<AxesSubplot:xlabel='Rate the importance of Battery life from 1-3', ylabel='How much for rechargeable?'>
```

```
<Figure size 2592x864 with 0 Axes>
```





In [18]:

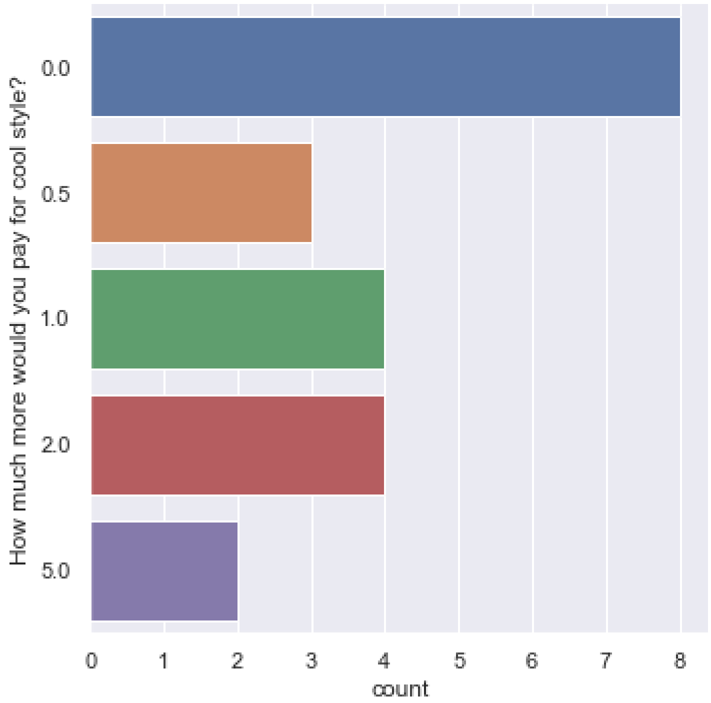
```python
quest7 = pd.DataFrame(data[['Rate the Look of product from 1-3','Would unique colors and patterns improve product?',
                            'How much more would you pay for cool style?']])
quest7_1 = quest7.drop(columns=['Would unique colors and patterns improve product?'])
quest7_1["Rate the Look of product from 1-3"].value_counts()
print(quest7['How much more would you pay for cool style?'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "How much more would you pay for cool style?", data = quest7_1, kind = "count")
f = plt.figure(figsize=(36,26))
sb.boxplot(x='Rate the Look of product from 1-3', y='How much more would you pay for cool style?', data=quest7_1)
# f = plt.figure(figsize=(36,26))
```

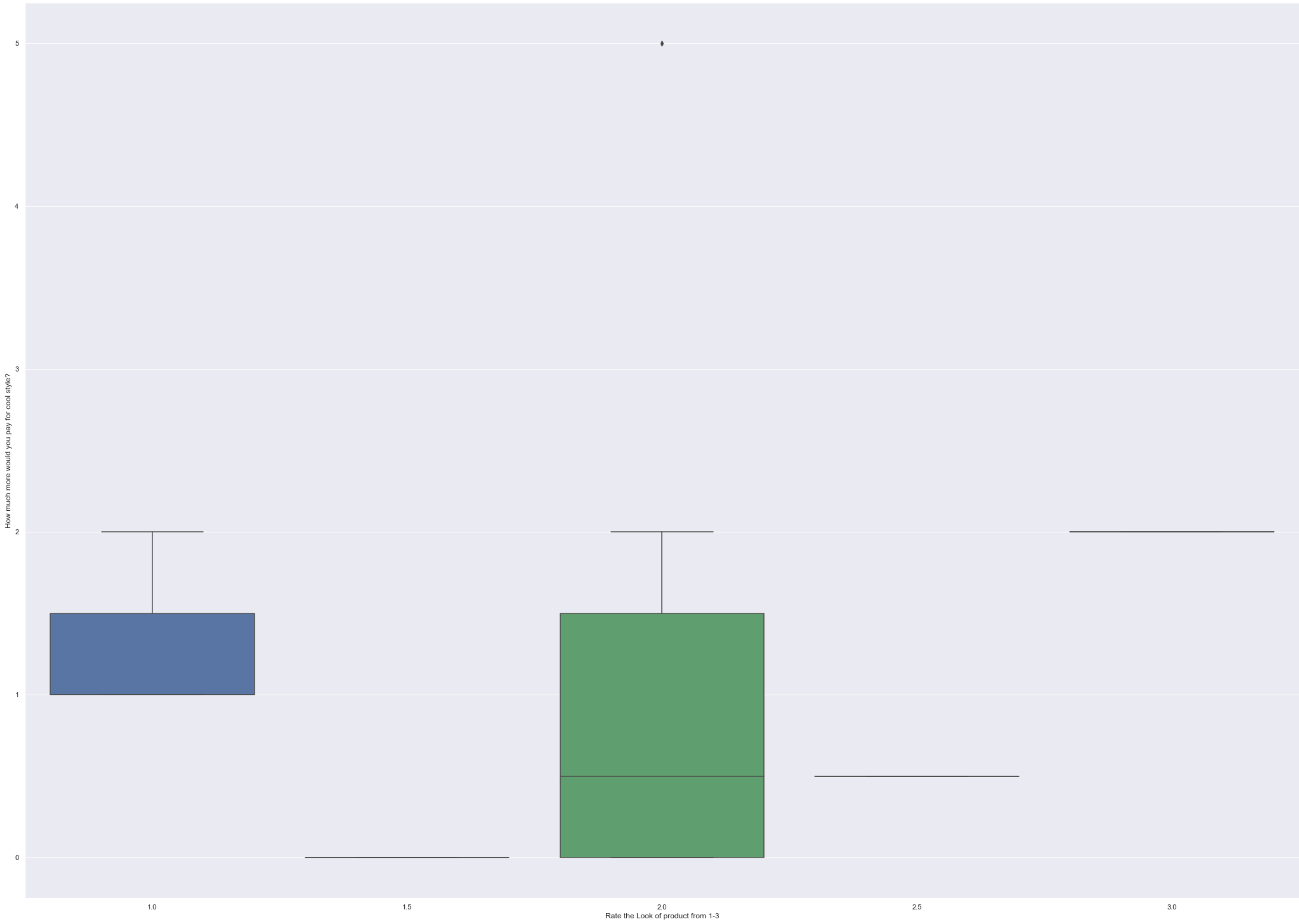
```
count    21.000000
mean      1.119048
std       1.490845
min       0.000000
25%       0.000000
50%       0.500000
75%       2.000000
max       5.000000
Name: How much more would you pay for cool style?, dtype: float64
```

```
/usr/local/lib/python3.9/site-packages/pandas/io/formats/format.py:1405: FutureWarning: Index.ravel returning ndarray is
deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```

Out[18]:
```
<AxesSubplot:xlabel='Rate the Look of product from 1-3', ylabel='How much more would you pay for cool style?'>
```

```
<Figure size 2592x864 with 0 Axes>
```

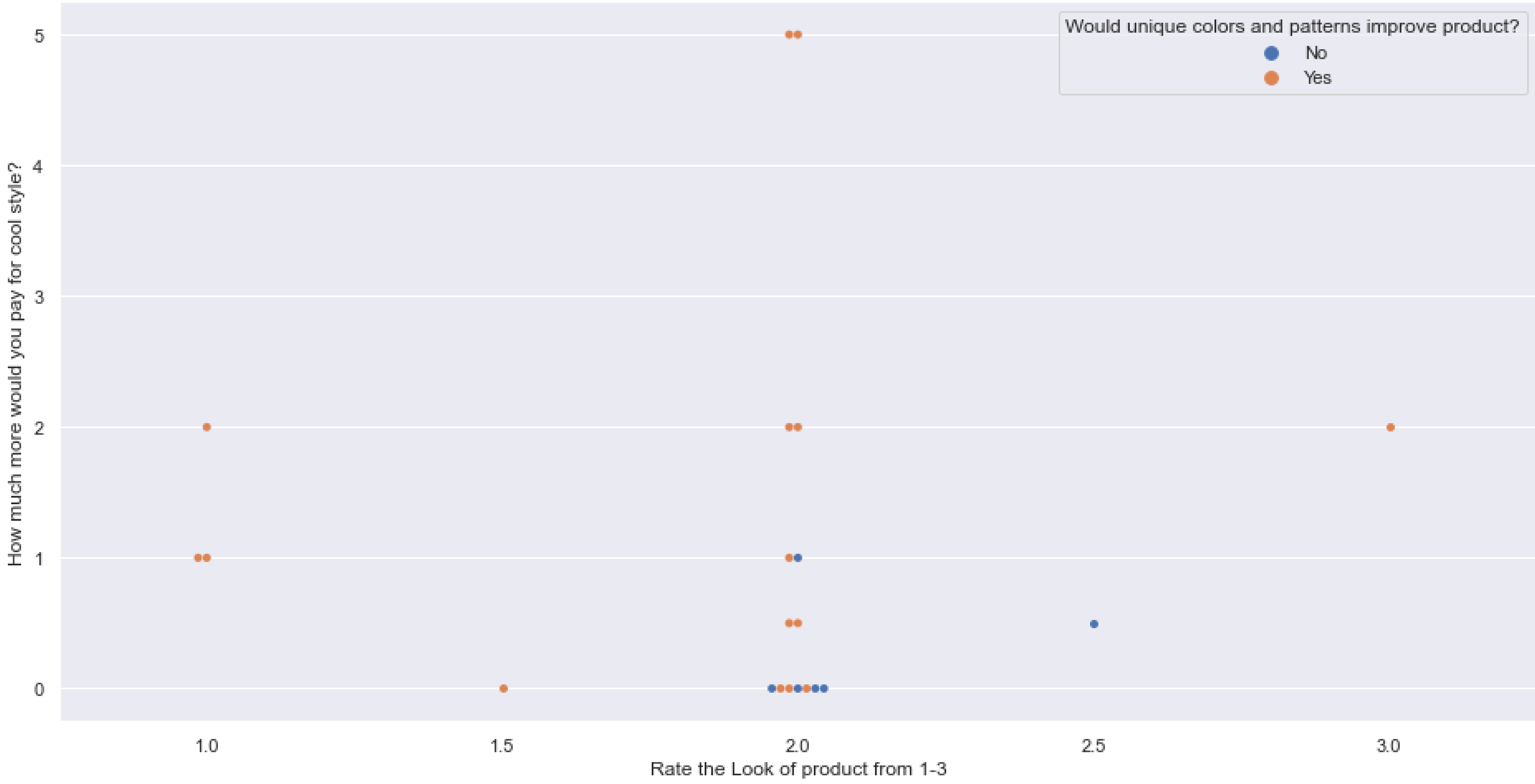




In [19]:
```
f = plt.figure(figsize=(16,8))
sb.swarmplot(x='Rate the Look of product from 1-3', y='How much more would you pay for cool style?', data=quest7,hue='Wo
```

```
/usr/local/lib/python3.9/site-packages/pandas/io/formats/format.py:1405: FutureWarning: Index.ravel returning ndarray is
deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
/usr/local/lib/python3.9/site-packages/pandas/io/formats/format.py:1405: FutureWarning: Index.ravel returning ndarray is
deprecated; in a future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```

Out[19]:
```
<AxesSubplot:xlabel='Rate the Look of product from 1-3', ylabel='How much more would you pay for cool style?'>
```



In [20]:

```
# creating association rule for Oral-B Cross Power Action Features Disliked by Participants
dislike = pd.read_csv('./files/Dislikes.csv').set_index('Name')
#dislike
frequent_dislike_itemsets = apriori(dislike, min_support=0.07, use_colnames=True)
dislike_rules = association_rules(frequent_dislike_itemsets, metric = 'lift', min_threshold=1)
dislike_rules

# People who want a stand, have a higher preference of battery convenience
```

Out[20]:

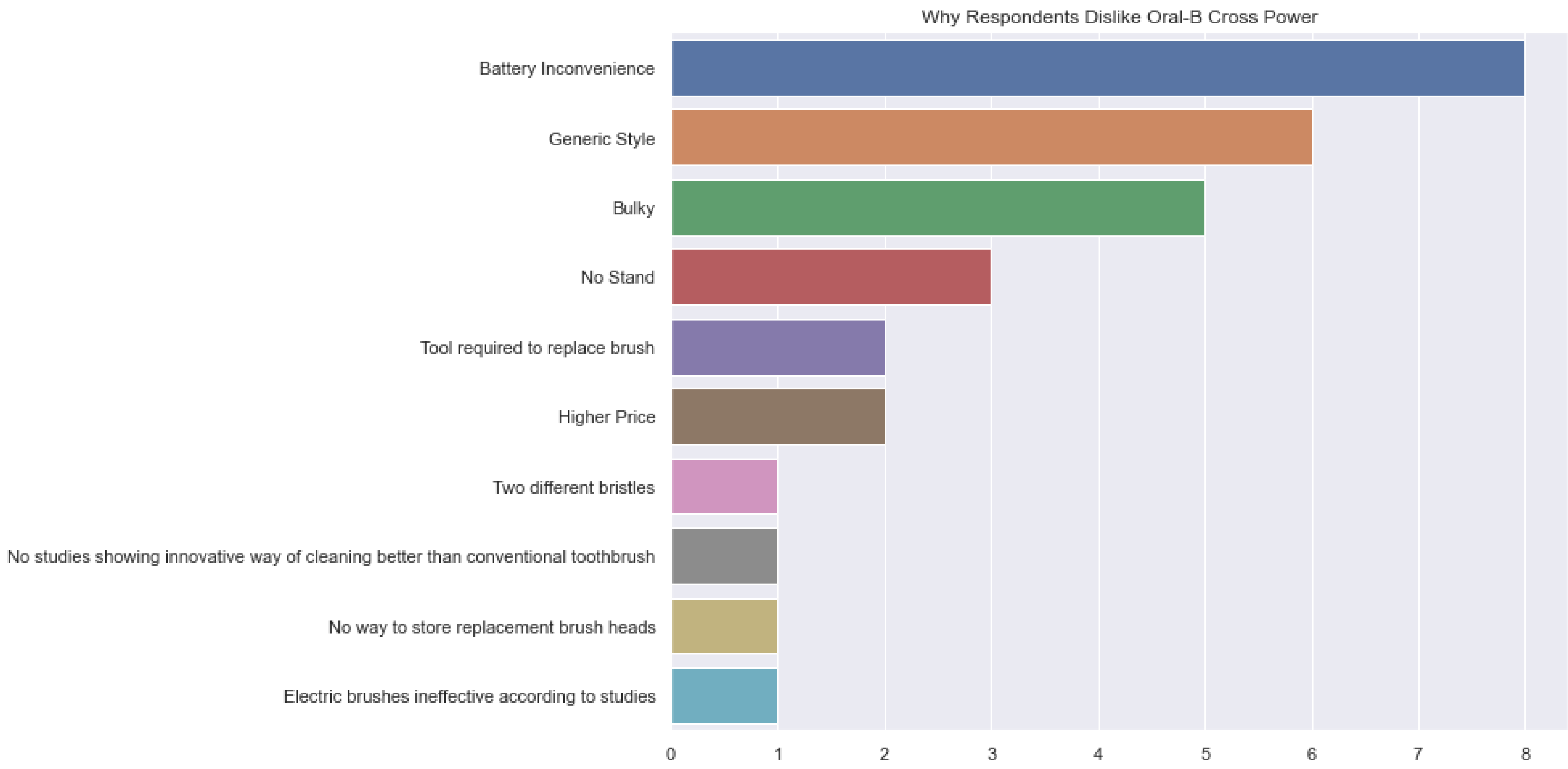
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Battery Inconvenience)	(No Stand)	0.380952	0.142857	0.095238	0.250000	1.75	0.040816	1.142857
1	(No Stand)	(Battery Inconvenience)	0.142857	0.380952	0.095238	0.666667	1.75	0.040816	1.857143

In [117…

```
current_dislike = dislike.copy()
current_dislike = current_dislike.astype('int64')

dislike_list = []
import operator
for col in current_dislike.columns:
    dislike_list.append(col)

dislike_total_dict={}
for col in dislike_list:
    total = current_dislike[col].sum()
    dislike_total_dict[col] = total

dislike_total_dict = dict(sorted(dislike_total_dict.items(),
                                 key=operator.itemgetter(1),
                                 reverse=True))

f = plt.figure(figsize = (10,8))
plt.title('Why Respondents Dislike Oral-B Cross Power')
# plt.ylabel('Current Dislikes')
sb.barplot(y=list(dislike_total_dict.keys()),x=list(dislike_total_dict.values()),orient = 'h')
```

Out[117…

```
<AxesSubplot:title={'center':'Why Respondents Dislike Oral-B Cross Power'}>
```



In [22]:

```
# creating association rule for Oral-B Cross Power Action Features Liked by Participants
likes = pd.read_csv('./files/Likes.csv').set_index('Name')
# likes.head()
# likes
frequent_like_itemsets = apriori(likes, min_support=0.1, use_colnames=True)
likes_rules = association_rules(frequent_like_itemsets, metric = 'lift', min_threshold=1)
likes_rules
```

Out[22]:

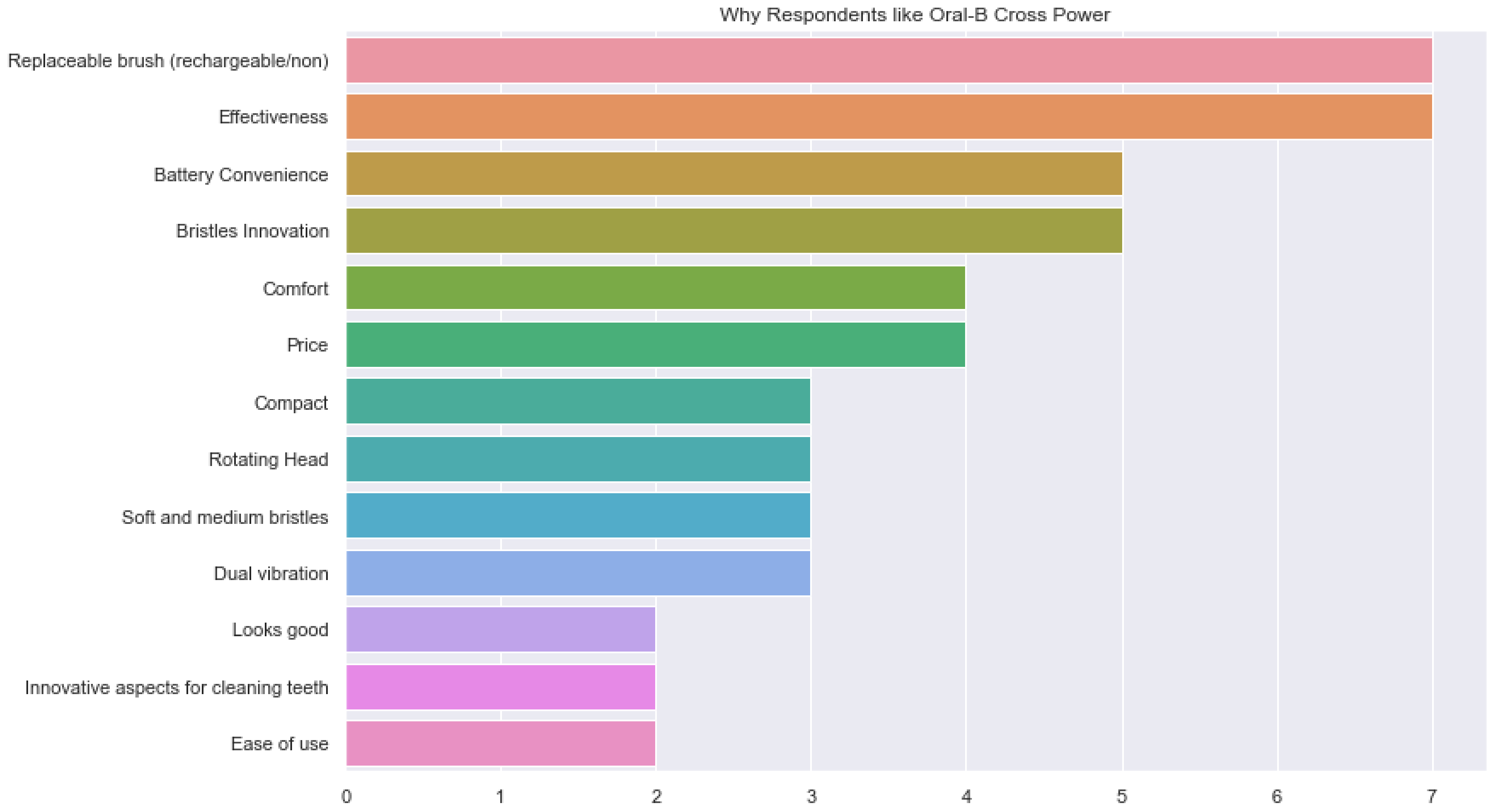
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Replaceable brush (rechargeable/non))	(Comfort)	0.333333	0.190476	0.142857	0.428571	2.250000	0.079365	1.416667
1	(Comfort)	(Replaceable brush (rechargeable/non))	0.190476	0.333333	0.142857	0.750000	2.250000	0.079365	2.666667
2	(Price)	(Battery Convenience)	0.190476	0.238095	0.142857	0.750000	3.150000	0.097506	3.047619
3	(Battery Convenience)	(Price)	0.238095	0.190476	0.142857	0.600000	3.150000	0.097506	2.023810
4	(Bristles Innovation)	(Battery Convenience)	0.238095	0.238095	0.142857	0.600000	2.520000	0.086168	1.904762
5	(Battery Convenience)	(Bristles Innovation)	0.238095	0.238095	0.142857	0.600000	2.520000	0.086168	1.904762
6	(Replaceable brush (rechargeable/non))	(Effectiveness)	0.333333	0.333333	0.142857	0.428571	1.285714	0.031746	1.166667
7	(Effectiveness)	(Replaceable brush (rechargeable/non))	0.333333	0.333333	0.142857	0.428571	1.285714	0.031746	1.166667

In [118…

```
current_likes = likes.copy()
current_likes = current_likes.astype('int64')

likes_list = []
import operator
for col in current_likes.columns:
    likes_list.append(col)

likes_total_dict={}
for col in likes_list:
    total = current_likes[col].sum()
    likes_total_dict[col] = total

likes_total_dict = dict(sorted(likes_total_dict.items(),
                               key=operator.itemgetter(1),
                               reverse=True))

f = plt.figure(figsize = (12,8))
plt.title('Why Respondents like Oral-B Cross Power')
# plt.ylabel('Current Dislikes')
sb.barplot(y=list(likes_total_dict.keys()),x=list(likes_total_dict.values()),orient = 'h')
```

Out[118…

```
<AxesSubplot:title={'center':'Why Respondents like Oral-B Cross Power'}>
```



In [24]:

```
# creating association rule for Brand and type of toothbrush Participants currently uses
brand = pd.read_csv('./files/Brand_and_type.csv').set_index('Name')
# brand
frequent_brand_type_itemsets = apriori(brand, min_support=0.07, use_colnames=True)
brand_rules = association_rules(frequent_brand_type_itemsets, metric = 'lift', min_threshold=1)
brand_rules
```

Out[24]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Rechargeable)	(Oral-B)	0.095238	0.380952	0.095238	1.000000	2.625000	0.058957	inf
1	(Oral-B)	(Rechargeable)	0.380952	0.095238	0.095238	0.250000	2.625000	0.058957	1.206349
2	(Oral-B)	(Manual)	0.380952	0.857143	0.333333	0.875000	1.020833	0.006803	1.142857
3	(Manual)	(Oral-B)	0.857143	0.380952	0.333333	0.388889	1.020833	0.006803	1.012987
4	(Manual)	(Name brand)	0.857143	0.095238	0.095238	0.111111	1.166667	0.013605	1.017857
5	(Name brand)	(Manual)	0.095238	0.857143	0.095238	1.000000	1.166667	0.013605	inf
6	(Colgate)	(Manual)	0.190476	0.857143	0.190476	1.000000	1.166667	0.027211	inf
7	(Manual)	(Colgate)	0.857143	0.190476	0.190476	0.222222	1.166667	0.027211	1.040816
8	(Battery Operated)	(Crest)	0.142857	0.142857	0.095238	0.666667	4.666667	0.074830	2.571429
9	(Crest)	(Battery Operated)	0.142857	0.142857	0.095238	0.666667	4.666667	0.074830	2.571429

In [25]:

```
# Creating Overall Association rule for Brand,type,like & disliked features, features wanted by Participants
combined = pd.concat([brand,likes,dislike,basket],axis=1)
# combined
frequent_combined_itemsets = apriori(combined, min_support=0.2, use_colnames=True)
combined_rules = association_rules(frequent_combined_itemsets, metric = 'lift', min_threshold=1)
combined_rules
```

Out[25]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Oral-B)	(Manual)	0.380952	0.857143	0.333333	0.875000	1.020833	0.006803	1.142857
1	(Manual)	(Oral-B)	0.857143	0.380952	0.333333	0.388889	1.020833	0.006803	1.012987
2	(Replaceable brush (rechargeable/non))	(Manual)	0.333333	0.857143	0.285714	0.857143	1.000000	0.000000	1.000000
3	(Manual)	(Replaceable brush (rechargeable/non))	0.857143	0.333333	0.285714	0.333333	1.000000	0.000000	1.000000
4	(Effectiveness)	(Manual)	0.333333	0.857143	0.333333	1.000000	1.166667	0.047619	inf
5	(Manual)	(Effectiveness)	0.857143	0.333333	0.333333	0.388889	1.166667	0.047619	1.090909
6	(Bristles Innovation)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
7	(Manual)	(Bristles Innovation)	0.857143	0.238095	0.238095	0.277778	1.166667	0.034014	1.054945
8	(Battery Inconvenience)	(Manual)	0.380952	0.857143	0.333333	0.875000	1.020833	0.006803	1.142857
9	(Manual)	(Battery Inconvenience)	0.857143	0.380952	0.333333	0.388889	1.020833	0.006803	1.012987
10	(Generic Style)	(Manual)	0.285714	0.857143	0.285714	1.000000	1.166667	0.040816	inf
11	(Manual)	(Generic Style)	0.857143	0.285714	0.285714	0.333333	1.166667	0.040816	1.071429
12	(Bulky)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
13	(Manual)	(Bulky)	0.857143	0.238095	0.238095	0.277778	1.166667	0.034014	1.054945
14	(Rechargeable)	(Manual)	0.476190	0.857143	0.476190	1.000000	1.166667	0.068027	inf
15	(Manual)	(Rechargeable)	0.857143	0.476190	0.476190	0.555556	1.166667	0.068027	1.178571
16	(Style)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
17	(Manual)	(Style)	0.857143	0.238095	0.238095	0.277778	1.166667	0.034014	1.054945
18	(Stand)	(Manual)	0.428571	0.857143	0.380952	0.888889	1.037037	0.013605	1.285714
19	(Manual)	(Stand)	0.857143	0.428571	0.380952	0.444444	1.037037	0.013605	1.028571
20	(Extra head)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
21	(Manual)	(Extra head)	0.857143	0.238095	0.238095	0.277778	1.166667	0.034014	1.054945
22	(Rechargeable)	(Battery Inconvenience)	0.476190	0.380952	0.238095	0.500000	1.312500	0.056689	1.238095
23	(Battery Inconvenience)	(Rechargeable)	0.380952	0.476190	0.238095	0.625000	1.312500	0.056689	1.396825
24	(Battery Inconvenience)	(Stand)	0.380952	0.428571	0.238095	0.625000	1.458333	0.074830	1.523810
25	(Stand)	(Battery Inconvenience)	0.428571	0.380952	0.238095	0.555556	1.458333	0.074830	1.392857
26	(Battery Inconvenience)	(Battery Indicator)	0.380952	0.285714	0.238095	0.625000	2.187500	0.129252	1.904762
27	(Battery Indicator)	(Battery Inconvenience)	0.285714	0.380952	0.238095	0.833333	2.187500	0.129252	3.714286
28	(Rechargeable)	(Generic Style)	0.476190	0.285714	0.238095	0.500000	1.750000	0.102041	1.428571
29	(Generic Style)	(Rechargeable)	0.285714	0.476190	0.238095	0.833333	1.750000	0.102041	3.142857
30	(Stand)	(Generic Style)	0.428571	0.285714	0.238095	0.555556	1.944444	0.115646	1.607143
31	(Generic Style)	(Stand)	0.285714	0.428571	0.238095	0.833333	1.944444	0.115646	3.428571
32	(Rechargeable)	(Stand)	0.476190	0.428571	0.285714	0.600000	1.400000	0.081633	1.428571
33	(Stand)	(Rechargeable)	0.428571	0.476190	0.285714	0.666667	1.400000	0.081633	1.571429
34	(Rechargeable, Battery Inconvenience)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
35	(Rechargeable, Manual)	(Battery Inconvenience)	0.476190	0.380952	0.238095	0.500000	1.312500	0.056689	1.238095
36	(Battery Inconvenience, Manual)	(Rechargeable)	0.333333	0.476190	0.238095	0.714286	1.500000	0.079365	1.833333
37	(Rechargeable)	(Battery Inconvenience, Manual)	0.476190	0.333333	0.238095	0.500000	1.500000	0.079365	1.333333
38	(Battery Inconvenience)	(Rechargeable, Manual)	0.380952	0.476190	0.238095	0.625000	1.312500	0.056689	1.396825
39	(Manual)	(Rechargeable, Battery Inconvenience)	0.857143	0.238095	0.238095	0.277778	1.166667	0.034014	1.054945
40	(Rechargeable, Generic Style)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
41	(Rechargeable, Manual)	(Generic Style)	0.476190	0.285714	0.238095	0.500000	1.750000	0.102041	1.428571
42	(Generic Style, Manual)	(Rechargeable)	0.285714	0.476190	0.238095	0.833333	1.750000	0.102041	3.142857
43	(Rechargeable)	(Generic Style, Manual)	0.476190	0.285714	0.238095	0.500000	1.750000	0.102041	1.428571
44	(Generic Style)	(Rechargeable, Manual)	0.285714	0.476190	0.238095	0.833333	1.750000	0.102041	3.142857
45	(Manual)	(Rechargeable, Generic Style)	0.857143	0.238095	0.238095	0.277778	1.166667	0.034014	1.054945
46	(Stand, Generic Style)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
47	(Stand, Manual)	(Generic Style)	0.380952	0.285714	0.238095	0.625000	2.187500	0.129252	1.904762
48	(Generic Style, Manual)	(Stand)	0.285714	0.428571	0.238095	0.833333	1.944444	0.115646	3.428571
49	(Stand)	(Generic Style, Manual)	0.428571	0.285714	0.238095	0.555556	1.944444	0.115646	1.607143
50	(Generic Style)	(Stand, Manual)	0.285714	0.380952	0.238095	0.833333	2.187500	0.129252	3.714286
51	(Manual)	(Stand, Generic Style)	0.857143	0.238095	0.238095	0.277778	1.166667	0.034014	1.054945
52	(Rechargeable, Stand)	(Manual)	0.285714	0.857143	0.285714	1.000000	1.166667	0.040816	inf
53	(Rechargeable, Manual)	(Stand)	0.476190	0.428571	0.285714	0.600000	1.400000	0.081633	1.428571
54	(Stand, Manual)	(Rechargeable)	0.380952	0.476190	0.285714	0.750000	1.575000	0.104308	2.095238
55	(Rechargeable)	(Stand, Manual)	0.476190	0.380952	0.285714	0.600000	1.575000	0.104308	1.547619
56	(Stand)	(Rechargeable, Manual)	0.428571	0.476190	0.285714	0.666667	1.400000	0.081633	1.571429
57	(Manual)	(Rechargeable, Stand)	0.857143	0.285714	0.285714	0.333333	1.166667	0.040816	1.071429

In [56]:

```
# creating association rule for features wanted by participants including Sex,brand and type
features_basket = pd.read_csv('./files/Features_wanted_M_F.csv', header = 0).set_index('Name')
features_itemsets = apriori(features_basket, min_support=0.15, use_colnames=True)
features_rules = association_rules(features_itemsets, metric = 'lift', min_threshold=1)
features_goodrules = features_rules[features_rules['confidence'] > 0.7]
features_goodrules
# People who want a stand, would also want the battery to be rechargeable
# people who think style is important, would also want a stand
```

Out[56]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Oral-B)	(Male)	0.380952	0.619048	0.333333	0.875000	1.413462	0.097506	3.047619
4	(Style)	(Male)	0.238095	0.619048	0.190476	0.800000	1.292308	0.043084	1.904762
8	(Female)	(Manual)	0.380952	0.857143	0.333333	0.875000	1.020833	0.006803	1.142857
14	(Oral-B)	(Manual)	0.380952	0.857143	0.333333	0.875000	1.020833	0.006803	1.142857
18	(Colgate)	(Manual)	0.190476	0.857143	0.190476	1.000000	1.166667	0.027211	inf
20	(Want Rechargeable)	(Manual)	0.476190	0.857143	0.476190	1.000000	1.166667	0.068027	inf
22	(Style)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
24	(Stand)	(Manual)	0.428571	0.857143	0.380952	0.888889	1.037037	0.013605	1.285714
26	(Extra head)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
31	(Style)	(Stand)	0.238095	0.428571	0.190476	0.800000	1.866667	0.088435	2.857143
32	(Oral-B , Male)	(Manual)	0.333333	0.857143	0.333333	1.000000	1.166667	0.047619	inf
33	(Oral-B , Manual)	(Male)	0.333333	0.619048	0.333333	1.000000	1.615385	0.126984	inf
35	(Oral-B)	(Male, Manual)	0.380952	0.523810	0.333333	0.875000	1.670455	0.133787	3.809524
38	(Male, Want Rechargeable)	(Manual)	0.285714	0.857143	0.285714	1.000000	1.166667	0.040816	inf
42	(Style, Male)	(Manual)	0.190476	0.857143	0.190476	1.000000	1.166667	0.027211	inf
43	(Style, Manual)	(Male)	0.238095	0.619048	0.190476	0.800000	1.292308	0.043084	1.904762
45	(Style)	(Male, Manual)	0.238095	0.523810	0.190476	0.800000	1.527273	0.065760	2.380952
48	(Stand, Male)	(Manual)	0.238095	0.857143	0.238095	1.000000	1.166667	0.034014	inf
54	(Female, Want Rechargeable)	(Manual)	0.190476	0.857143	0.190476	1.000000	1.166667	0.027211	inf
60	(Stand, Want Rechargeable)	(Manual)	0.285714	0.857143	0.285714	1.000000	1.166667	0.040816	inf
61	(Stand, Manual)	(Want Rechargeable)	0.380952	0.476190	0.285714	0.750000	1.575000	0.104308	2.095238
66	(Stand, Style)	(Manual)	0.190476	0.857143	0.190476	1.000000	1.166667	0.027211	inf
68	(Style, Manual)	(Stand)	0.238095	0.428571	0.190476	0.800000	1.866667	0.088435	2.857143
70	(Style)	(Stand, Manual)	0.238095	0.380952	0.190476	0.800000	2.100000	0.099773	3.095238

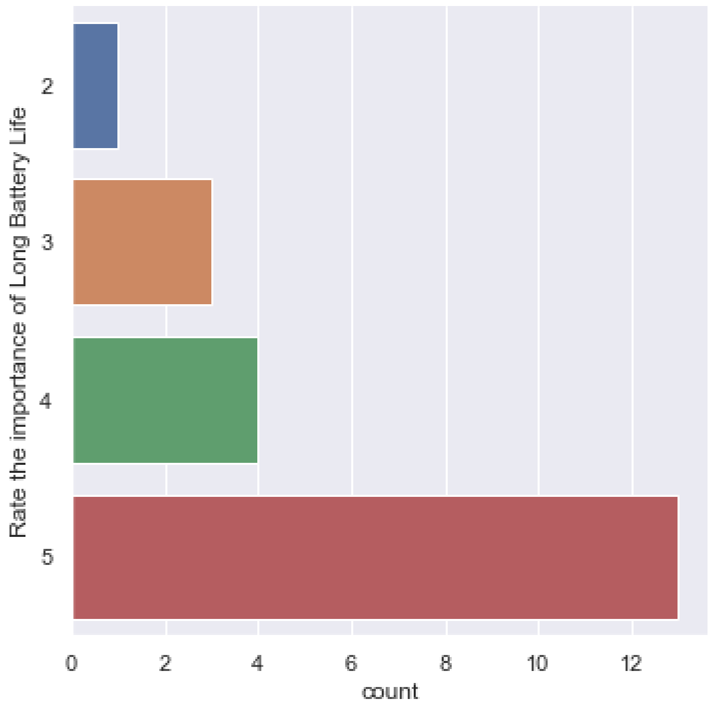
In [111…

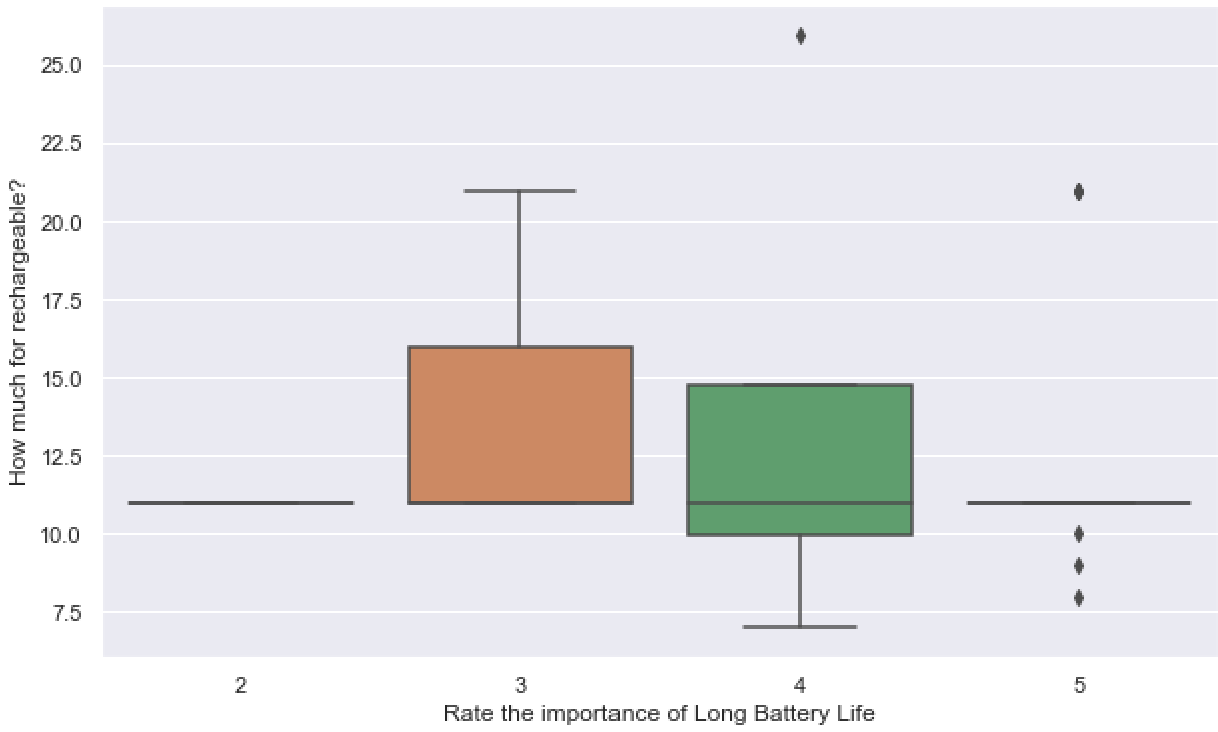
```
recharge_long_bat = data[['Rate the importance of Long Battery Life','How much for rechargeable?','Willingness to pay fo
recharge_long_bat = recharge_long_bat.rename(columns = {'Willingness to pay for Rechargeable' : 'Willingness to Pay Ext
# quest4 = pd.DataFrame(data[['Rate current price of product','How much you would pay']])
recharge_long_bat["Rate the importance of Long Battery Life"].value_counts()
print(recharge_long_bat['How much for rechargeable?'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "Rate the importance of Long Battery Life", data = recharge_long_bat, kind = "count")

f = plt.figure(figsize=(10,6))
sb.boxplot(x='Rate the importance of Long Battery Life', y='How much for rechargeable?', data=recharge_long_bat)

f = plt.figure(figsize=(10,6))
sb.boxplot(x='Rate the importance of Long Battery Life', y='Willingness to Pay Extra for Rechargeable', data=recharge_lo

# f = plt.figure(figsize=(18,9))
# sb.boxplot(x='How much for rechargeable?', y='Willingness to pay for Rechargeable', data=recharge_long_bat)

# # Plot the Linear Regression line
# f = plt.figure(figsize=(16, 8))
# how_much_recharge = data['How much for rechargeable?']
# will_recharge = data['Willingness to pay for Rechargeable']
# # plt.plot(regline_x, regline_y, 'r-', linewidth = 3)
# plt.xlabel("Willingness to pay for Style")
# plt.ylabel("Willingness to pay for Dual Speed")
# plt.scatter(will_recharge, how_much_recharge)
# plt.show()
```

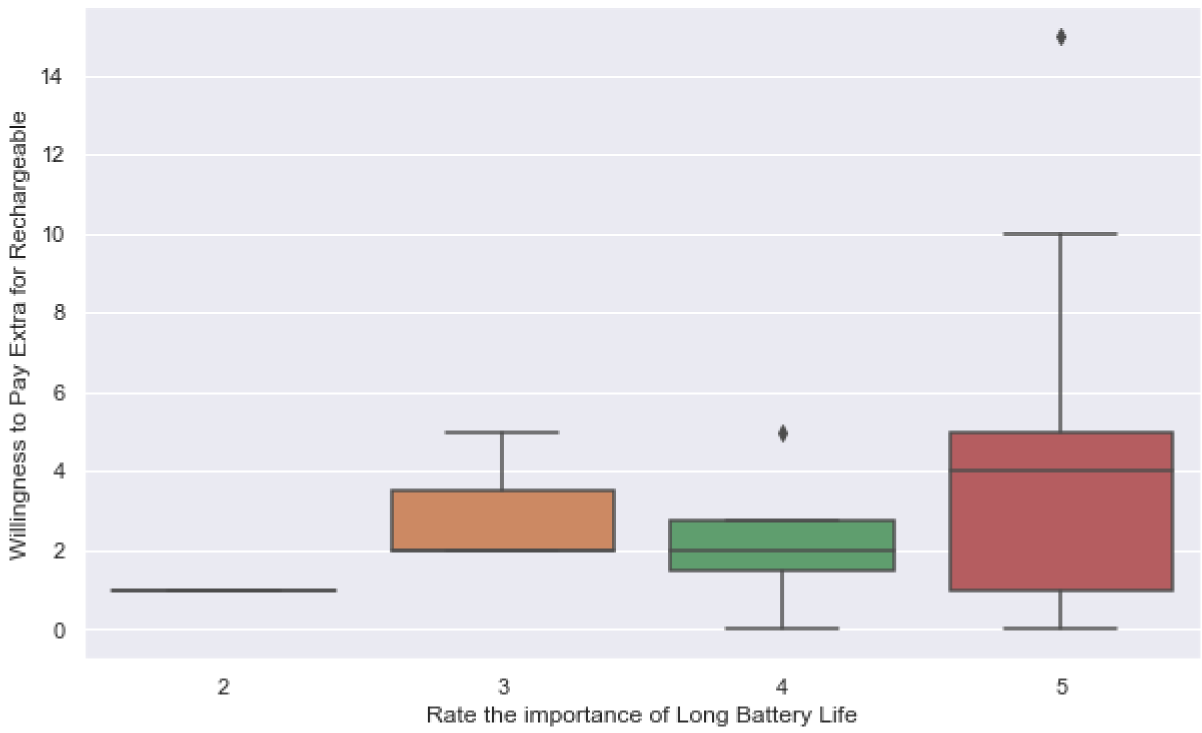
```
count    21.00000
mean     13.13381
std       5.28406
min       6.99000
25%      10.99000
50%      10.99000
75%      10.99000
max      25.99000
Name: How much for rechargeable?, dtype: float64
```

Out[111…

```
<AxesSubplot:xlabel='Rate the importance of Long Battery Life', ylabel='Willingness to Pay Extra for Rechargeable'>
<Figure size 2592x864 with 0 Axes>
```



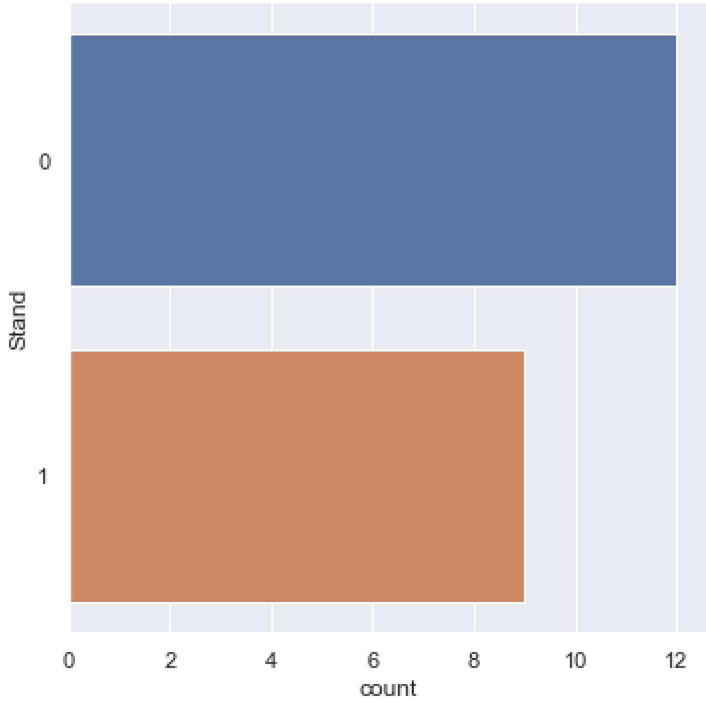


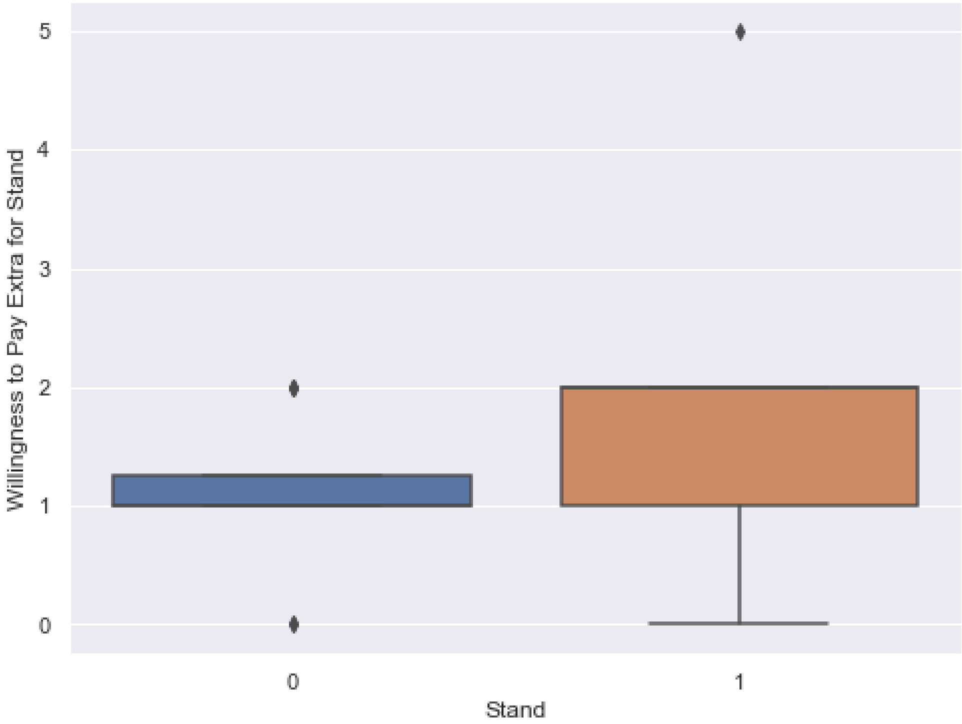


In [91]:
```
will_want_stand = data[['Willingness to pay for Stand','Stand']]
will_want_stand = will_want_stand.rename(columns={"Willingness to pay for Stand": "Willingness to Pay Extra for Stand"}
```

In [113…
```
will_want_stand["Stand"].value_counts()
# print(recharge_long_bat['How much for rechargeable?'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "Stand", data = will_want_stand, kind = "count")

f = plt.figure(figsize=(8,6))
sb.boxplot(x='Stand', y='Willingness to Pay Extra for Stand', data=will_want_stand)
```

Out[113…
```
<AxesSubplot:xlabel='Stand', ylabel='Willingness to Pay Extra for Stand'>
<Figure size 2592x864 with 0 Axes>
```





In [100…

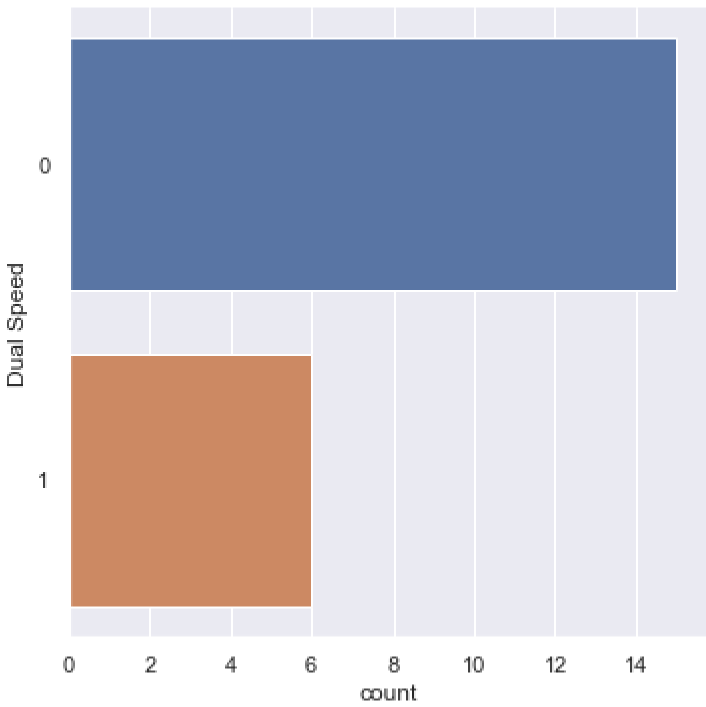
```
will_want_DualSpeed_Timer_BatteryIndicator = data[['Willingness to pay for Dual Speed','Dual Speed','Willingness to pay
will_want_DualSpeed_Timer_BatteryIndicator = will_want_DualSpeed_Timer_BatteryIndicator.rename(columns={"Willingness to
                                                                                                          "Willingness to p
                                                                                                          "Willingness to p

# will_want_DualSpeed_Timer_BatteryIndicator
```

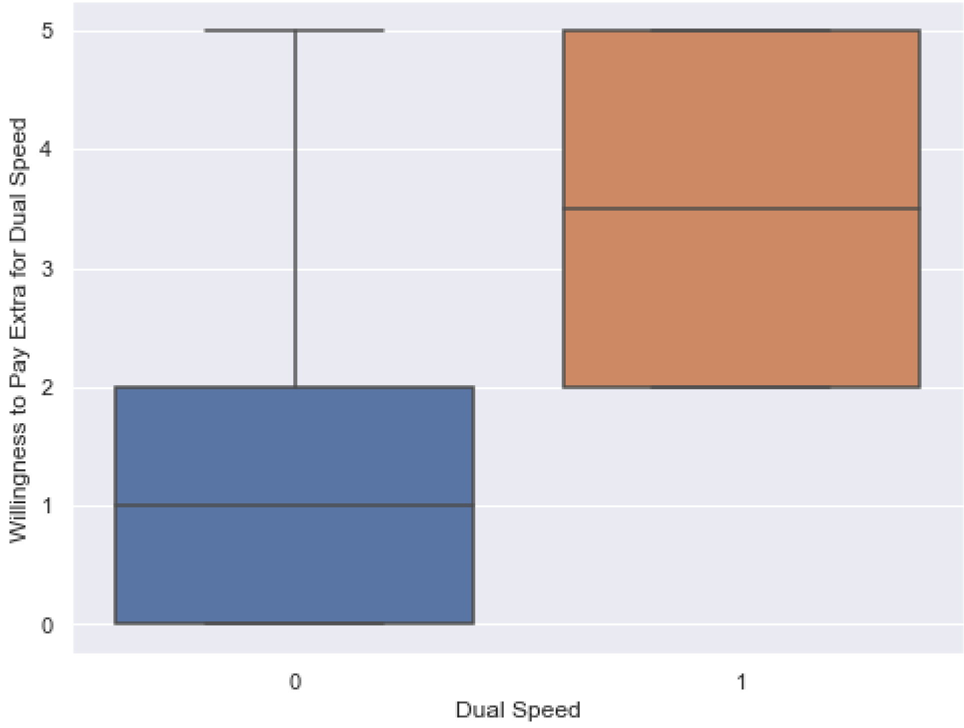
In [114…

```
will_want_DualSpeed_Timer_BatteryIndicator["Dual Speed"].value_counts()
# print(recharge_long_bat['How much for rechargeable?'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "Dual Speed", data = will_want_DualSpeed_Timer_BatteryIndicator, kind = "count")

f = plt.figure(figsize=(8,6))
sb.boxplot(x='Dual Speed', y='Willingness to Pay Extra for Dual Speed', data=will_want_DualSpeed_Timer_BatteryIndicator
```

Out[114…

```
<AxesSubplot:xlabel='Dual Speed', ylabel='Willingness to Pay Extra for Dual Speed'>
<Figure size 2592x864 with 0 Axes>
```



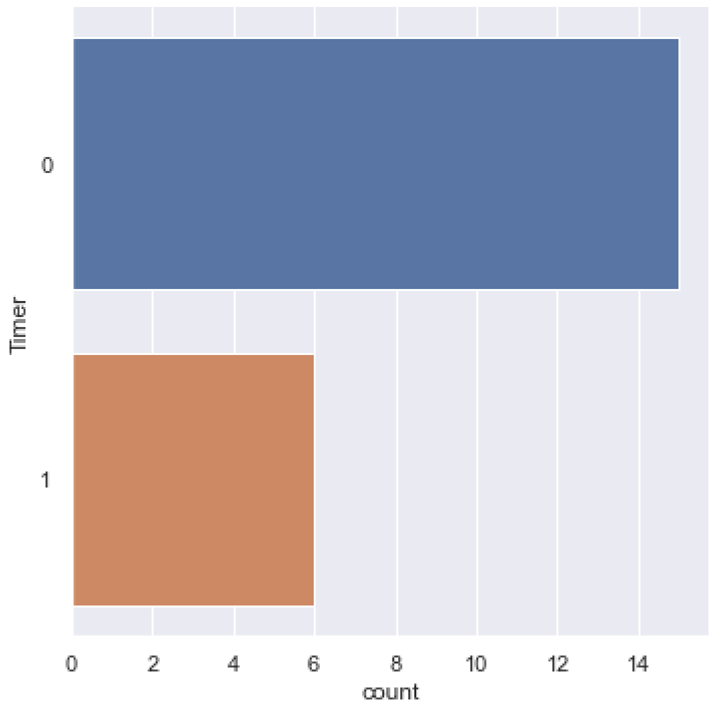


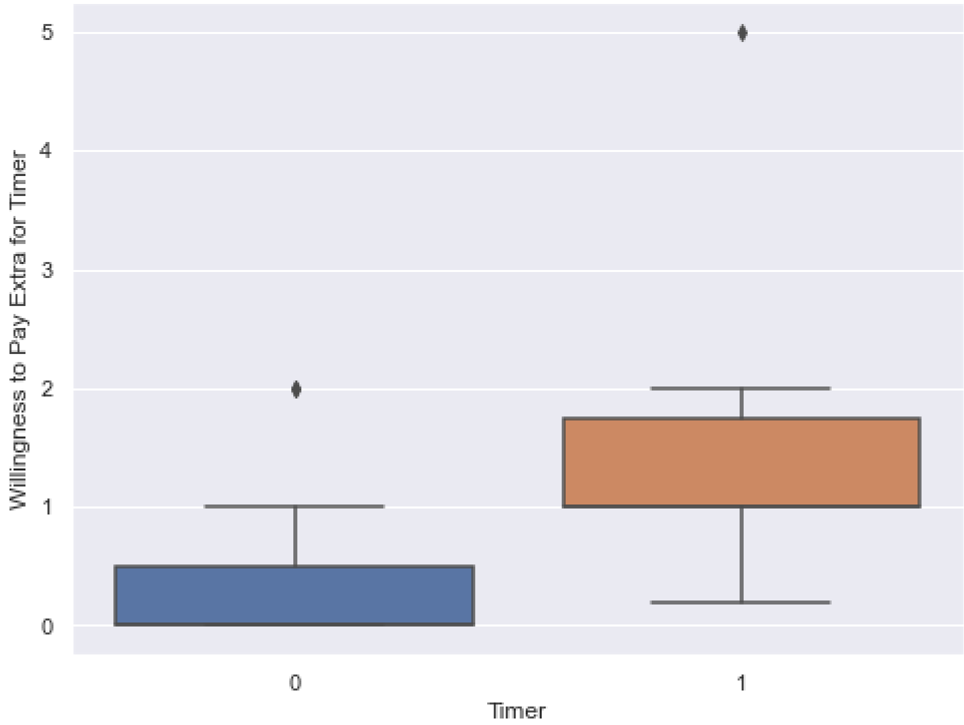
In [115…

```
will_want_DualSpeed_Timer_BatteryIndicator["Timer"].value_counts()
# print(recharge_long_bat['How much for rechargeable?'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "Timer", data = will_want_DualSpeed_Timer_BatteryIndicator, kind = "count")

f = plt.figure(figsize=(8,6))
sb.boxplot(x='Timer', y='Willingness to Pay Extra for Timer', data=will_want_DualSpeed_Timer_BatteryIndicator)
```

Out[115…

```
<AxesSubplot:xlabel='Timer', ylabel='Willingness to Pay Extra for Timer'>
<Figure size 2592x864 with 0 Axes>
```



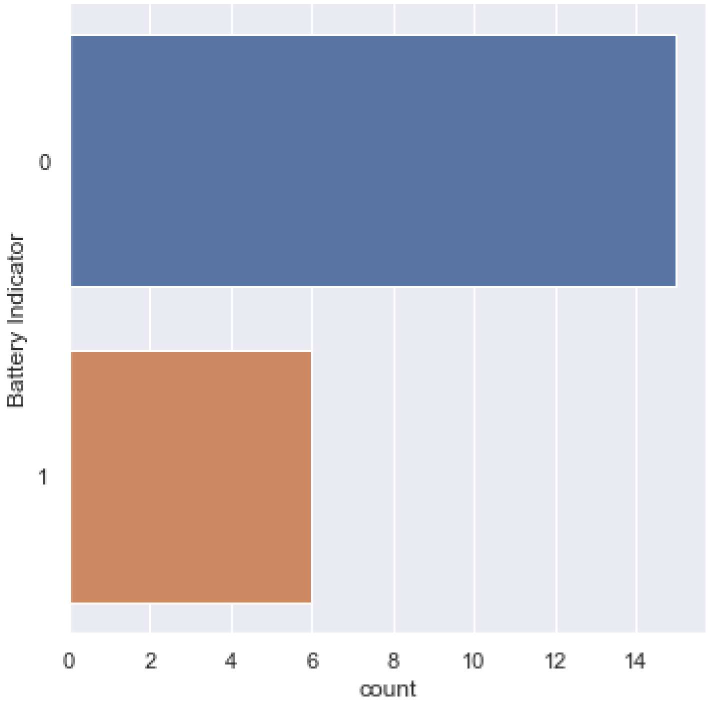


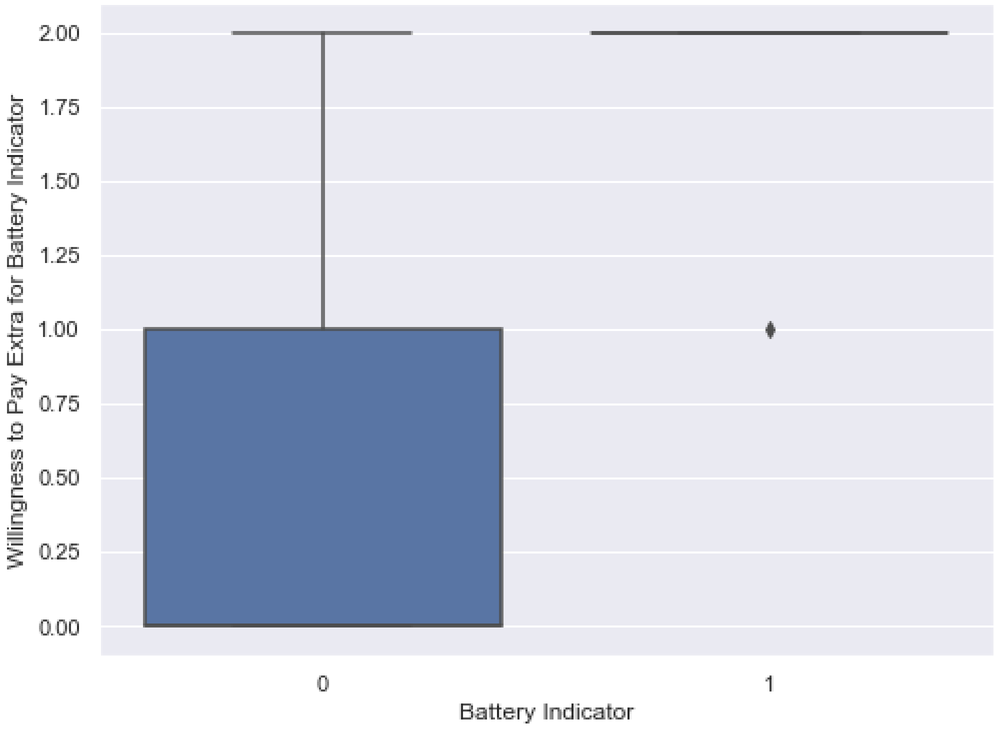
In [116…

```
will_want_DualSpeed_Timer_BatteryIndicator["Battery Indicator"].value_counts()
# print(recharge_long_bat['How much for rechargeable?'].describe())

f= plt.figure(figsize=(36,12))
sb.catplot(y = "Battery Indicator", data = will_want_DualSpeed_Timer_BatteryIndicator, kind = "count")

f = plt.figure(figsize=(8,6))
sb.boxplot(x='Battery Indicator', y='Willingness to Pay Extra for Battery Indicator', data=will_want_DualSpeed_Timer_Ba
```

Out[116…

```
<AxesSubplot:xlabel='Battery Indicator', ylabel='Willingness to Pay Extra for Battery Indicator'>
<Figure size 2592x864 with 0 Axes>
```





In [ ]: