UGLY CHRISTMAS PARTY

Nataly Valenzuela

November 2021

Introduction

Ugly Christmas Party (UCP) is a seasonal small business dedicated to selling Christmas sweaters online shipped through Amazon and their own warehouse.

Dataset

The data was taking from programs (Channel Advisor and SKU Vault) UCP uses to keep track of their products. The final dataset used to apply models was a combination of UCP's inventory, sales, and pricing datasets.

Sources:

- uglychristmasparty.com
- channeladvisor.com
- skuvault.com

Purpose of the analysis

UCP loses thousands of dollars every year when they do not order enough and sell out of a Christmas sweater or when they overestimate the number of sweaters that they will sell. Once the sweaters are ordered they cannot order more or return them. Seasonal product companies like UCP lose money every year in inventory costs whether it's in house or in the Amazon warehouses. Storing and housing a large amount of sweaters that are not sold during the Christmas season cost the business a lot of money. At the end of 2020 UCP had 11,843 sweaters left over = $110,000 approximately of unsold merchandise. The company needs an accurate model to help them determine how many sweaters they will sell for each design created.

Research Question

How do Decision Tree, Random Forest, K-nearest neighbor, and Linear Regression compare with each other in predicting how many Ugly Christmas sweaters will be sold based on the sweater's characteristics?

Variables

Output variable:

- Sold Quantity: Number of sweaters sold.

Input variables:

- SKU: unique sweater ID.
- Buy It Now Price: $14-$50.
- Category: Cute, Pop culture, Religious, Comedy, Costume, Alcohol, Naughty, and 3D Pop out.
- Change Price: has the price changed during the season or not?
- Character: Reindeer, Elf, Snowman, Jesus, Cat, Tv Show, Tree, Dog, Mrs. Claus, Gingerbread man, Santa, Llama, Fireplace, Giraffe, Godzilla, Internet, Polar bear, Menorah, Dinosaur, Sleigh, Penguins, Skull, Tuxedo, Wine, and Yeti.
- Color: black, green, blue, orange, red, grey, white, yellow, multicolor, turquoise, and purple.
- Image Count: 1-9.
- Lights: does the sweater have lights or not?
- Hood: does the sweater have a hood or not?
- Size: X-Small - 4X-Large.
- Classification: Men's Sweater, Women's Sweater, and Dog Sweater.

(See Appendix A for descriptive statistics, Appendix B frequency tables and Appendix C graphs)

Data Preprocessing, Data Partitioning, and Feature Selection

The sales, inventory, and pricing datasets were analyzed, and the data was very dirty. Irrelevant, duplicate, and over 400 null values columns were dropped. The dataset was scaled appropriately based on the type of data (categorical or continuous). Sold Quantity had a lot of outliers, so anything with the y variable above 200 was dropped. The data was cleaned thoroughly and merged into a final dataset containing 430 observations and 11 variables. The data was split into 70% training data and 30% testing data.

The model was performing poorly, so feature importance using Decision trees and field knowledge was used to select the most important variables. Variables that looked important in the graphs (Appendix C) were included, and then were left out because they were not adding significantly to the model's performance.

Important Variables Selected for Models		
Size	Change Price?	Lights
Image Count	Category: Pop culture and Comedy	Character: Elf, Dog, Reindeer, and Santa
Buy It Now Price	Color: blue and grey.	Hood

(See appendix D for feature importance graph.)

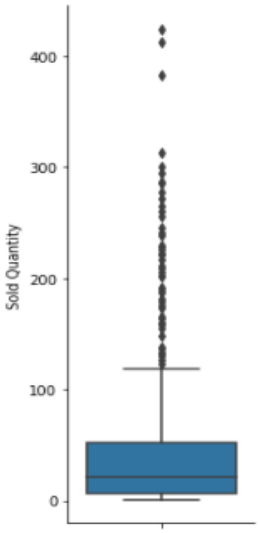
Algorithms

A Decision Tree, Random Forest, K-nearest neighbor, and Linear Regression models were applied. The training data was used to fit the model. In an intent to improve the results the output variable was made categorical using bins of 5 for the number of sweaters, but this did not improve the model significantly. Each model was hyper tuned using a grid search method, and the best result was the Random Forest model. This model had the least number of sweaters off by of 35 which implies the model is possibly not predicting well. (See Appendix E for models and Appendix F for hyperparameter tuning)

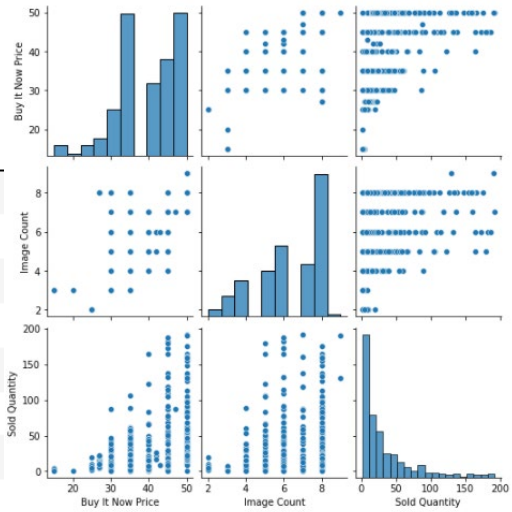
Conclusion

In 2020 UCP had an excess of 12,000 sweaters or $115,000 approximately, and the best model would be off by 14,000 sweaters or $195,000 by average not including storage costs. The datasets were not created to collect relevant data to solve the issue, and UCP does not have the infrastructure to collect the data needed to solve the problem. Although a lot was done to address the overfitting, this resulted in underfitting and to be off by 3 more sweaters (38 total per SKU). From the results of all the models applied it is clear important variables that explain how many sweaters will sell are missing. In the future text analysis of the reviews and social media promotions done in 2021 should help improve the model.

Appendix A - Descriptive Statistics



	Buy It Now Price	Image Count	Sold Quantity
count	430.000000	430.000000	430.000000
mean	40.150465	6.365116	32.609302
std	8.271321	1.688903	39.531120
min	14.990000	2.000000	1.000000
25%	34.990000	5.000000	5.000000
50%	39.990000	7.000000	18.000000
75%	49.990000	8.000000	43.000000
max	49.990000	9.000000	192.000000



Appendix B - Frequency Tables

Change Price?	
FALSE	300
TRUE	130

Category	
Pop culture	162
Comedy	63
Cute	58
Costume	47
Religious	40
Alcohol	37
Naughty	14
3D Pop out	9

Size	
Small	85
Medium	75
Large	74
XX-Large	61
X-Large	59
3X-Large	35
4X-Large	34
X-Small	6
2X Tall	1

Color	
black	169
red	96
blue	48
green	42
grey	23
multicolor	13
yellow	12
white	11
turquoise	6
purple	5
orange	5

Classification	
Men's Sweater	361
Women's Sweater	61
Dog Sweater	8

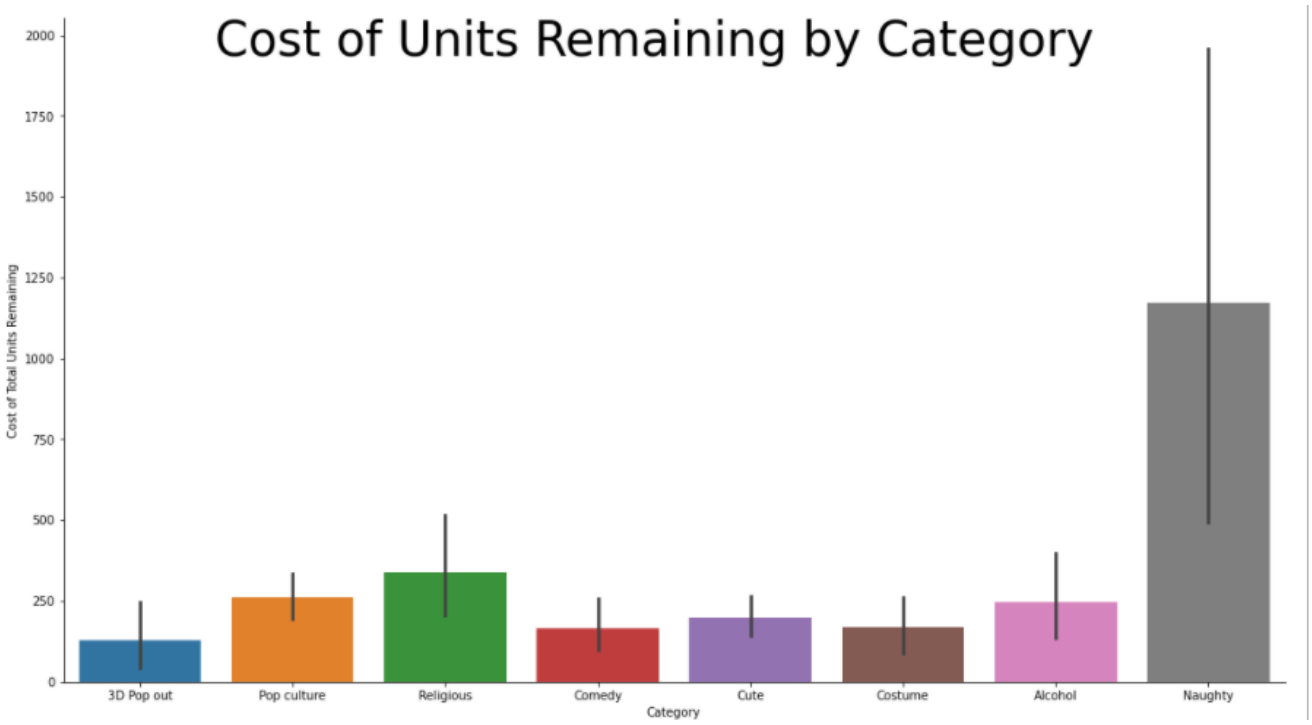
Hood	
0	416
1	14

Lights	
0	333
1	97

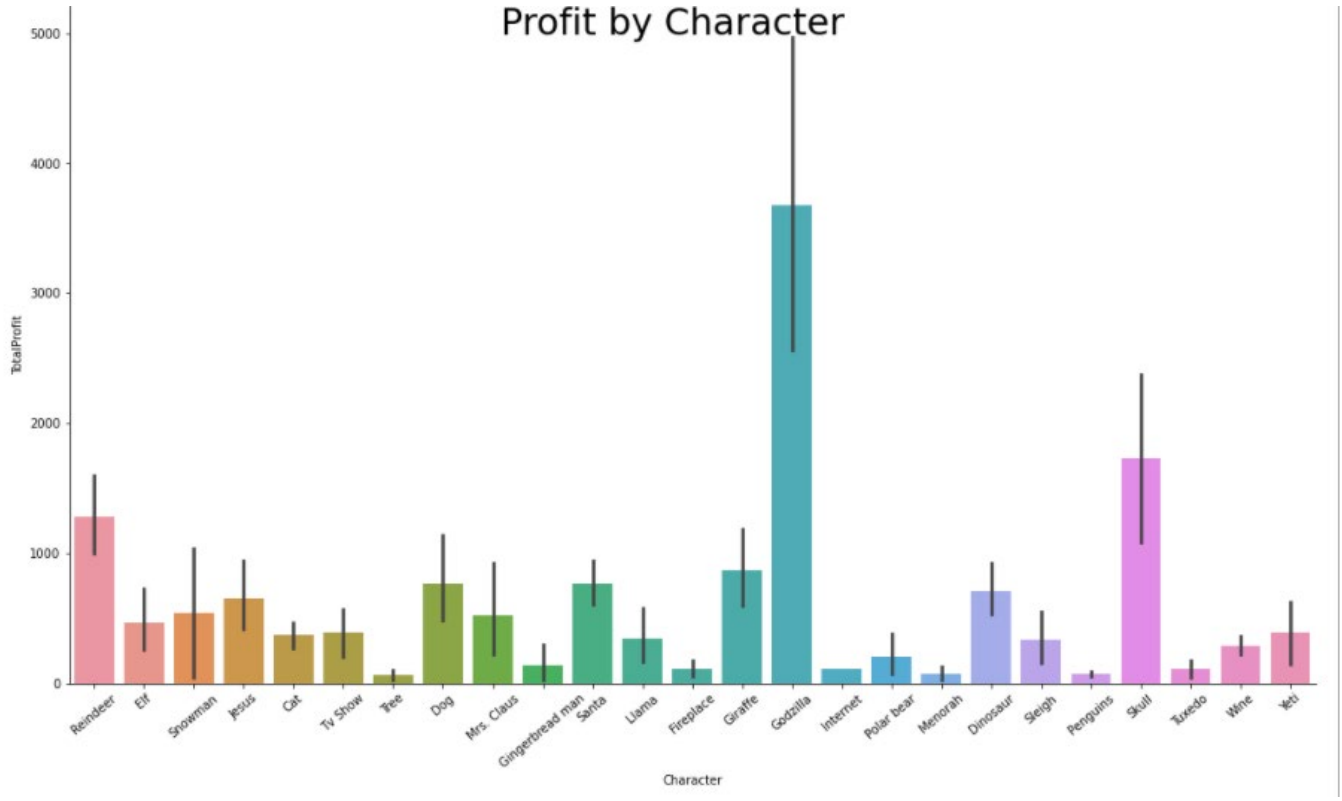
Character	
Santa	118
Reindeer	66
Elf	43
Jesus	33
Dog	29
Cat	16
Polar bear	14
Skull	12
Mrs. Claus	10
Penguins	9
Gingerbread man	8
Sleigh	7
Menorah	7
Giraffe	7
Tv Show	7
Dinosaur	7
Snowman	6
Tree	6
Fireplace	5
Llama	5
Godzilla	5
Wine	5
Yeti	2
Tuxedo	2
Internet	1

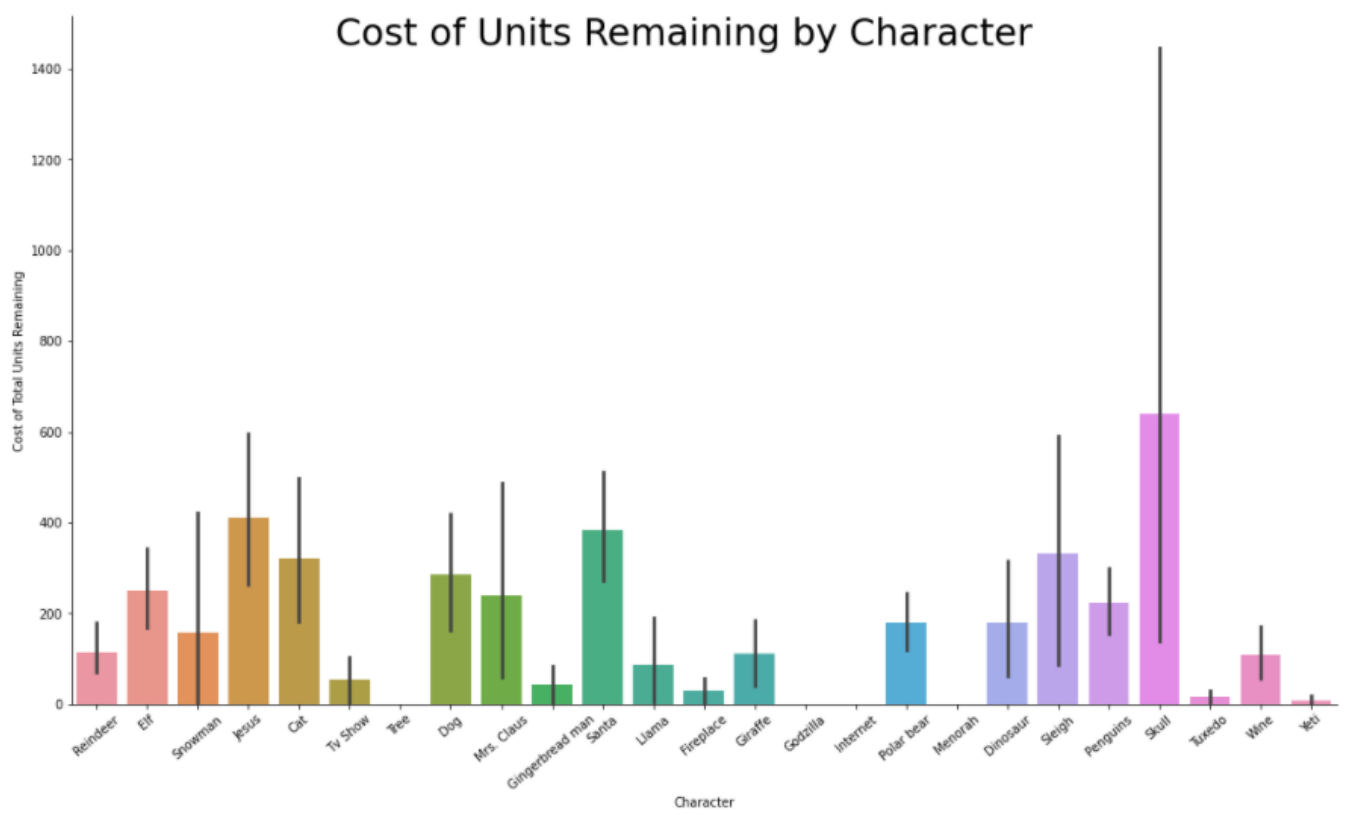
Appendix C - Graphs

- **Category**

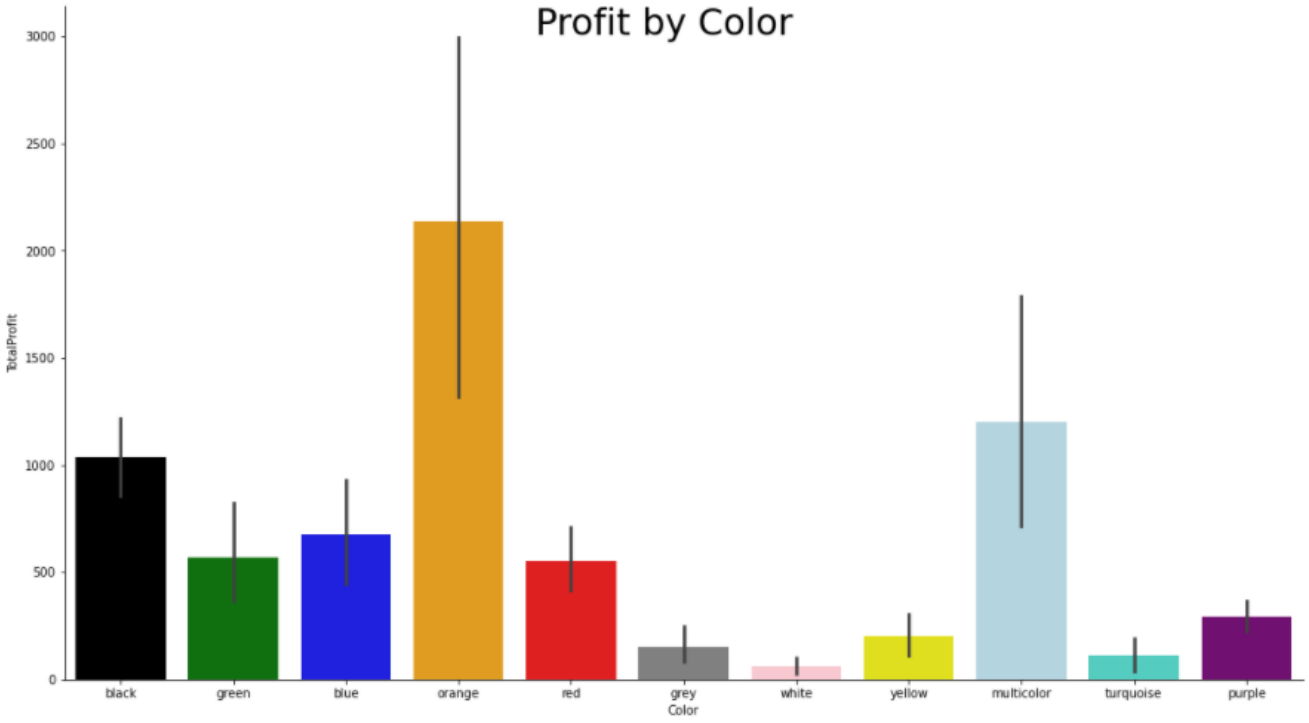


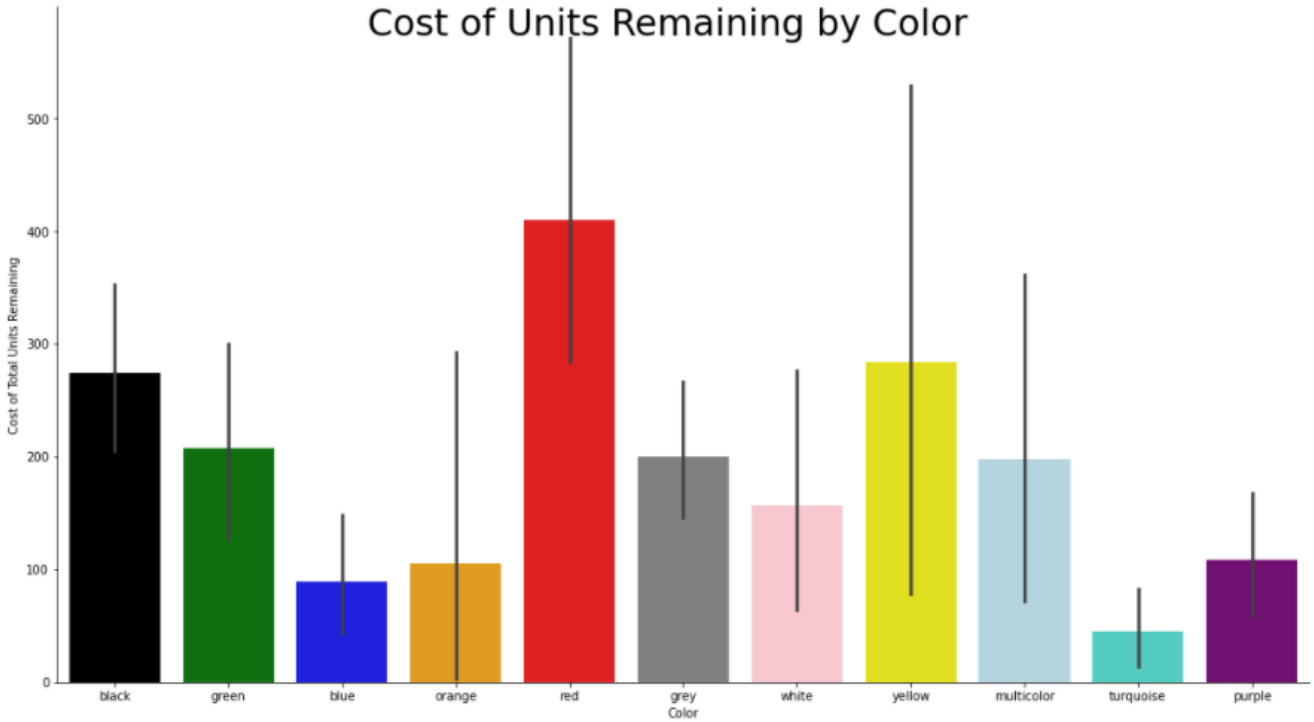
- **Character**


Profit by Character
TotalProfit
5000
4000
3000
2000
1000
0
Reindeer
Elf
Snowman
Jesus
Cat
Tv Show
Tree
Dog
Mrs. Claus
Gingerbread man
Santa
Llama
Fireplace
Giraffe
Godzilla
Internet
Polar bear
Menorah
Dinosaur
Sleigh
Penguins
Skull
Tuxedo
Wine
Yeti
Character

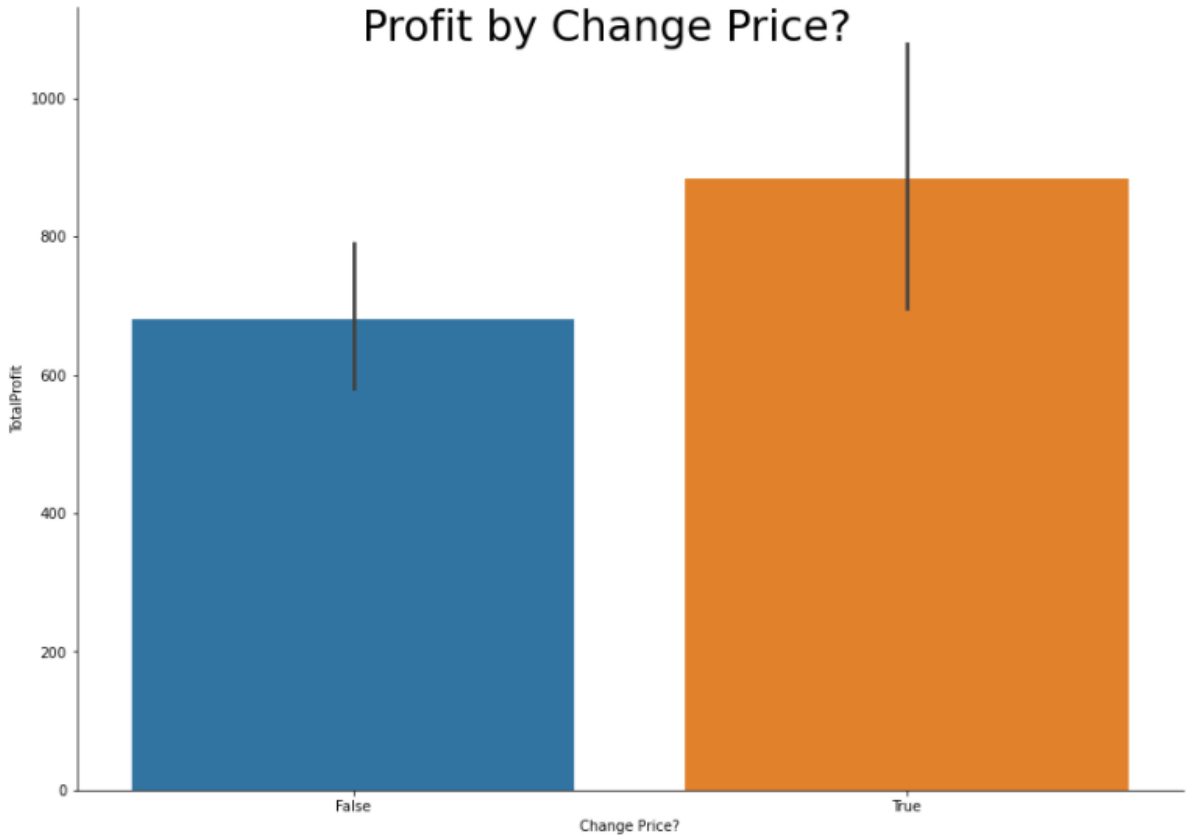

Cost of Units Remaining by Character
Cost of Total Units Remaining
1400
1200
1000
800
600
400
200
0
Reindeer
Elf
Snowman
Jesus
Cat
Tv Show
Tree
Dog
Mrs. Claus
Gingerbread man
Santa
Llama
Fireplace
Giraffe
Godzilla
Internet
Polar bear
Menorah
Dinosaur
Sleigh
Penguins
Skull
Tuxedo
Wine
Yeti
Character

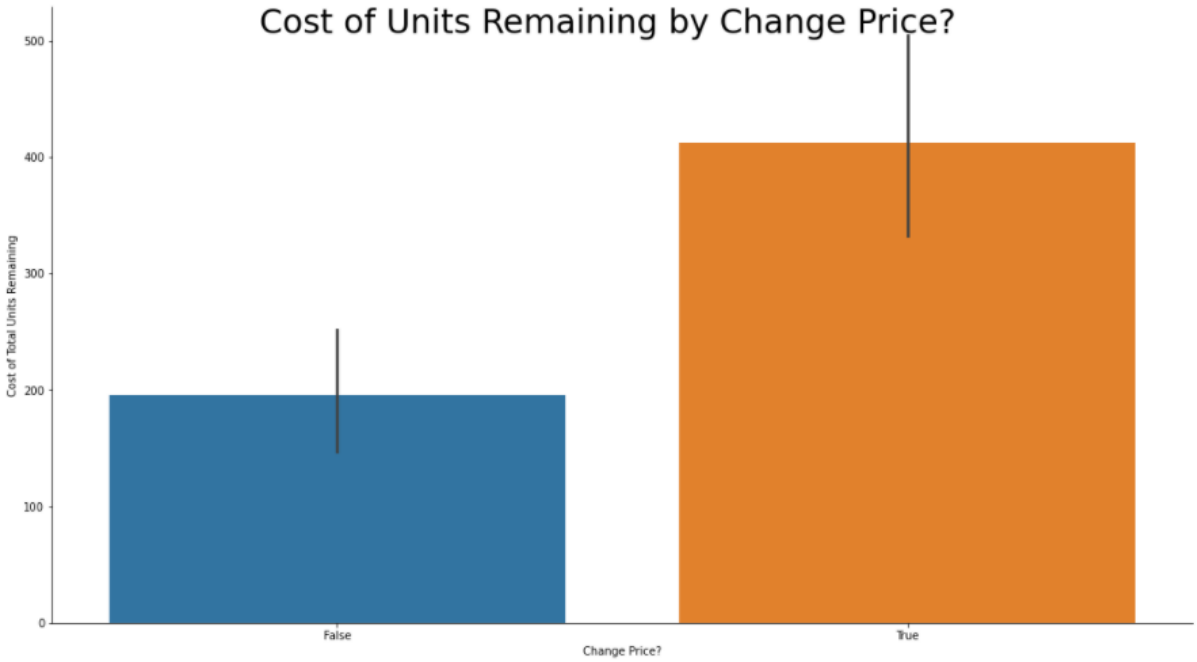
- **Color**


Profit by Color
TotalProfit
3000
2500
2000
1500
1000
500
0
black
green
blue
orange
red
grey
white
yellow
multicolor
turquoise
purple
Color

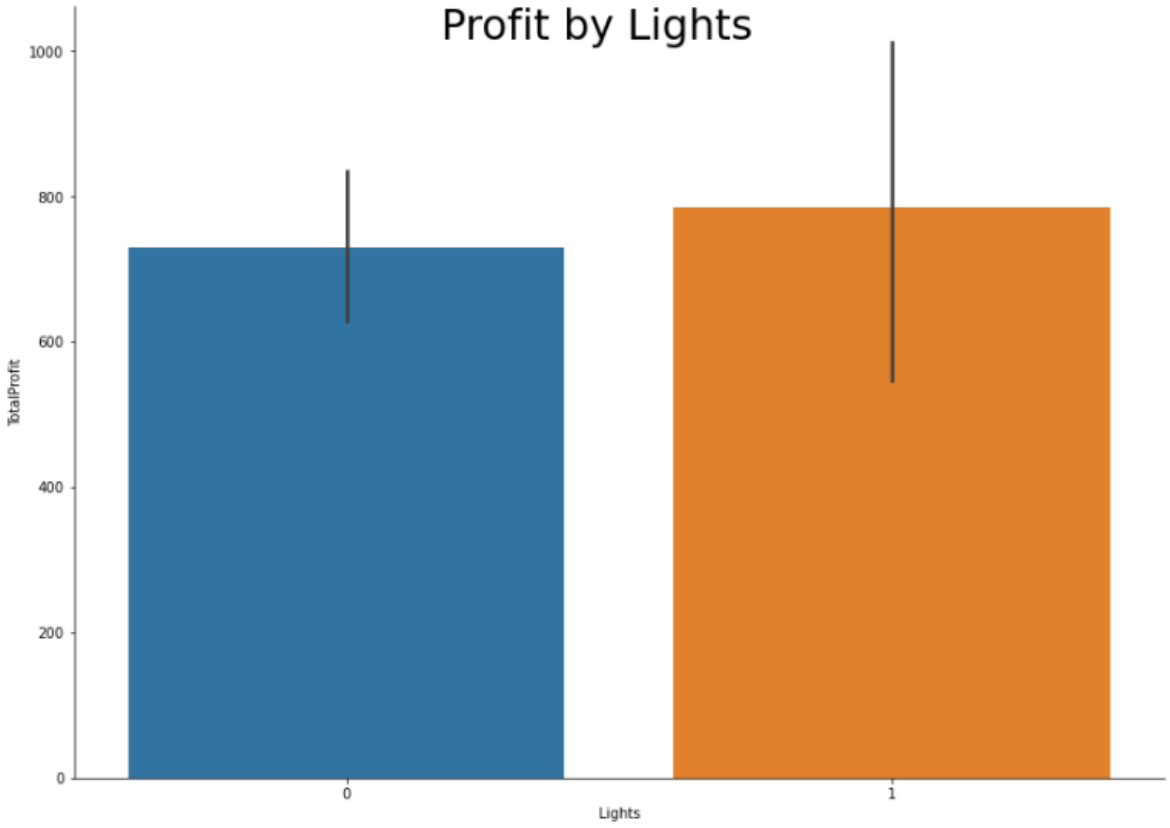

Cost of Units Remaining by Color
Cost of Total Units Remaining
500
400
300
200
100
0
black
green
blue
orange
red
grey
white
yellow
multicolor
turquoise
purple
Color

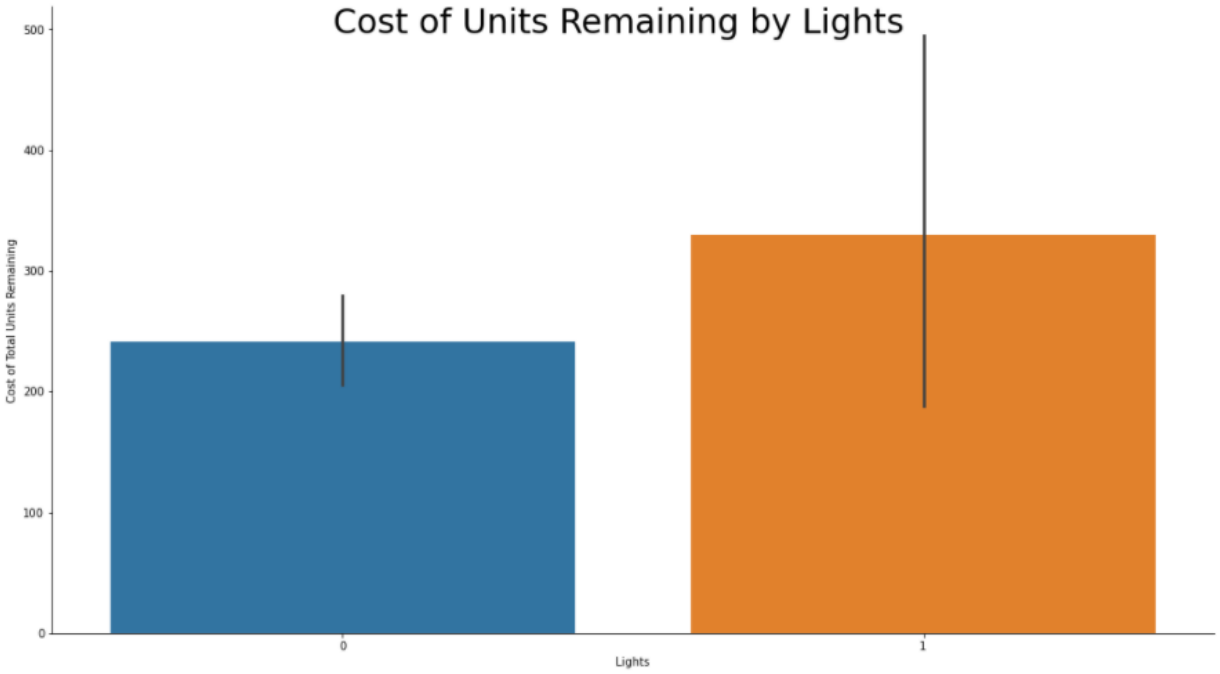
- **Change Price?**



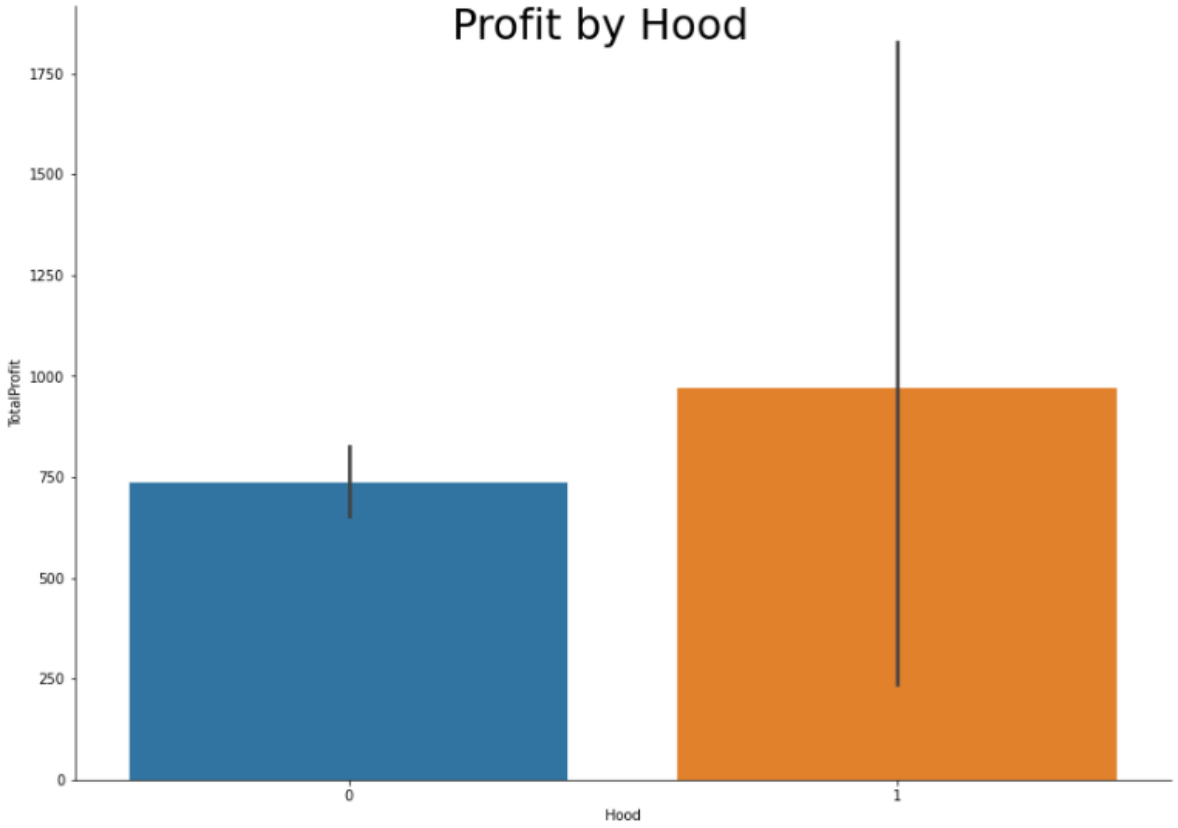


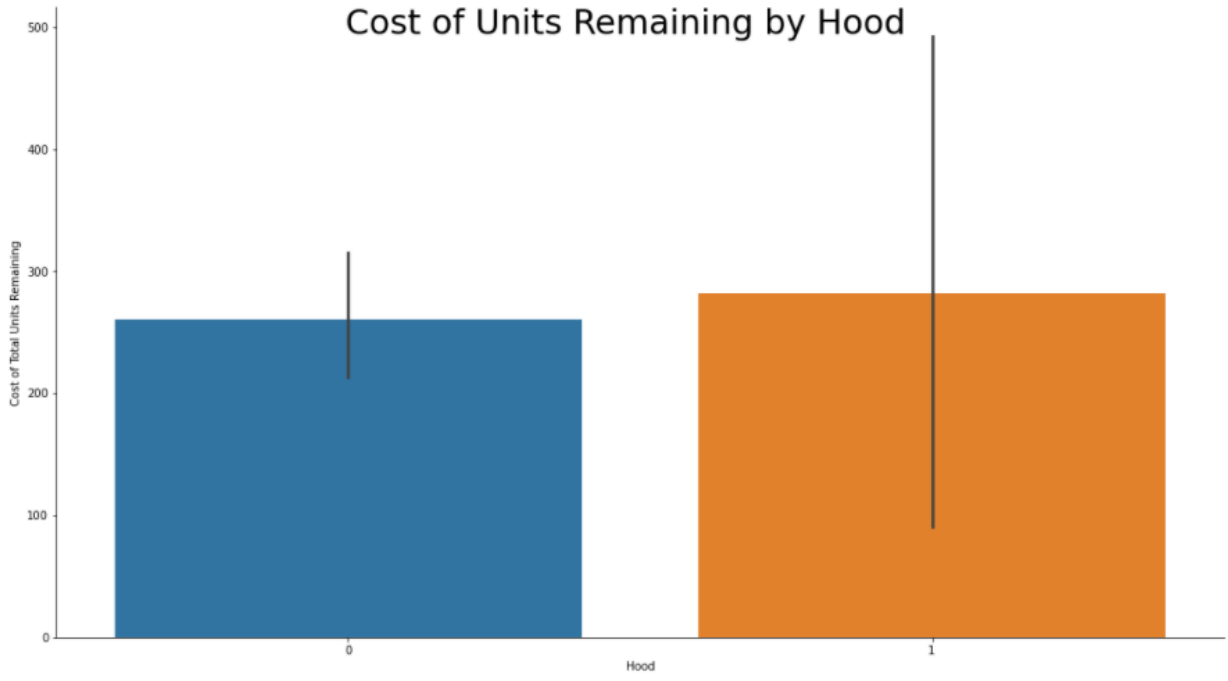
- **Lights**


Profit by Lights
1000
800
600
400
200
0
TotalProfit
0
1
Lights

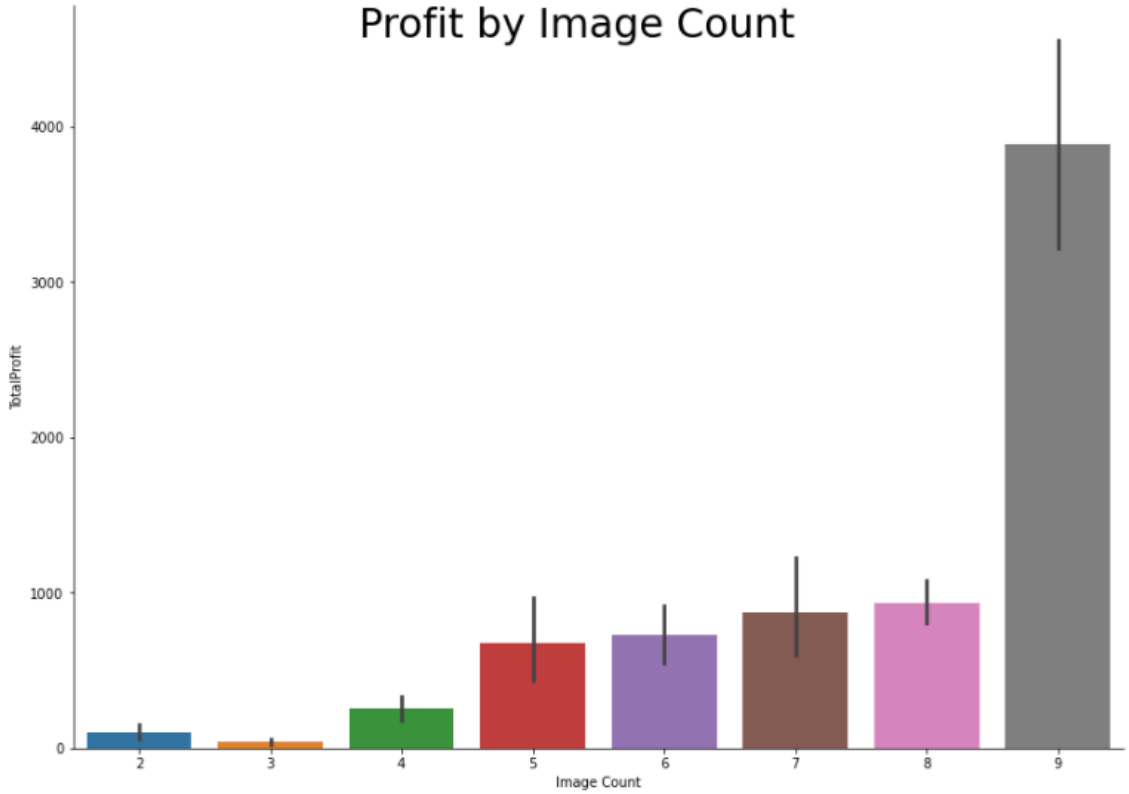

Cost of Units Remaining by Lights
500
400
300
200
100
0
Cost of Total Units Remaining
0
1
Lights

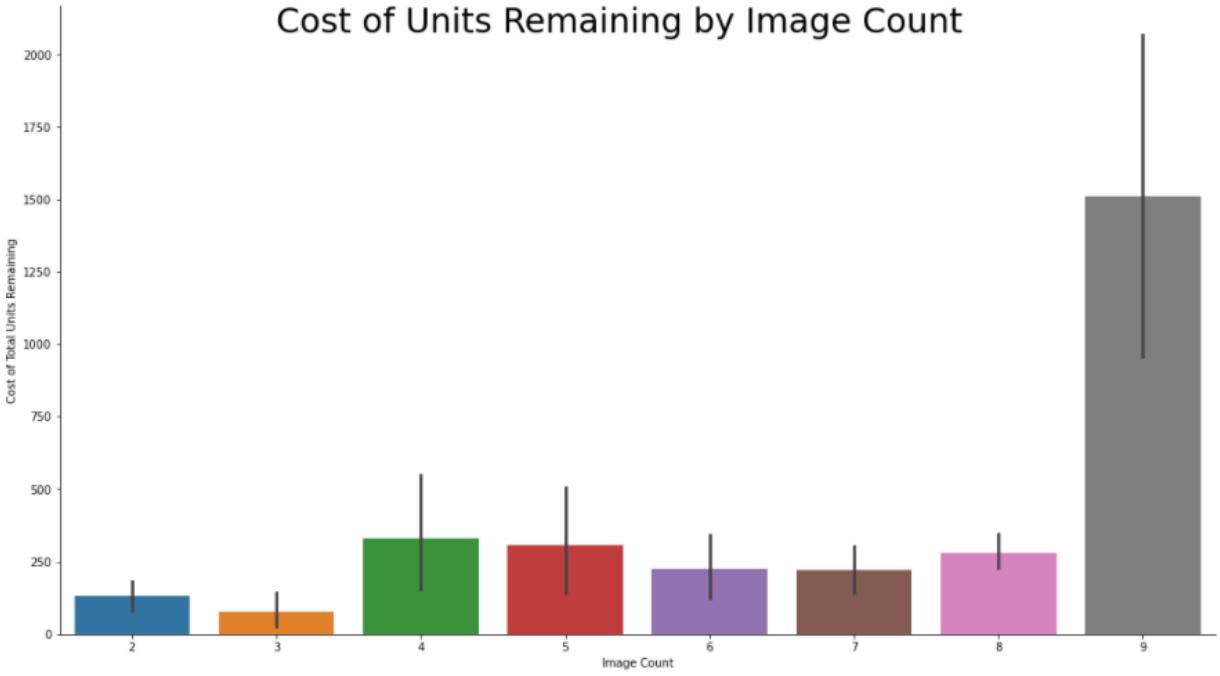
- **Hood**


Profit by Hood
TotalProfit
1750
1500
1250
1000
750
500
250
0
0
1
Hood

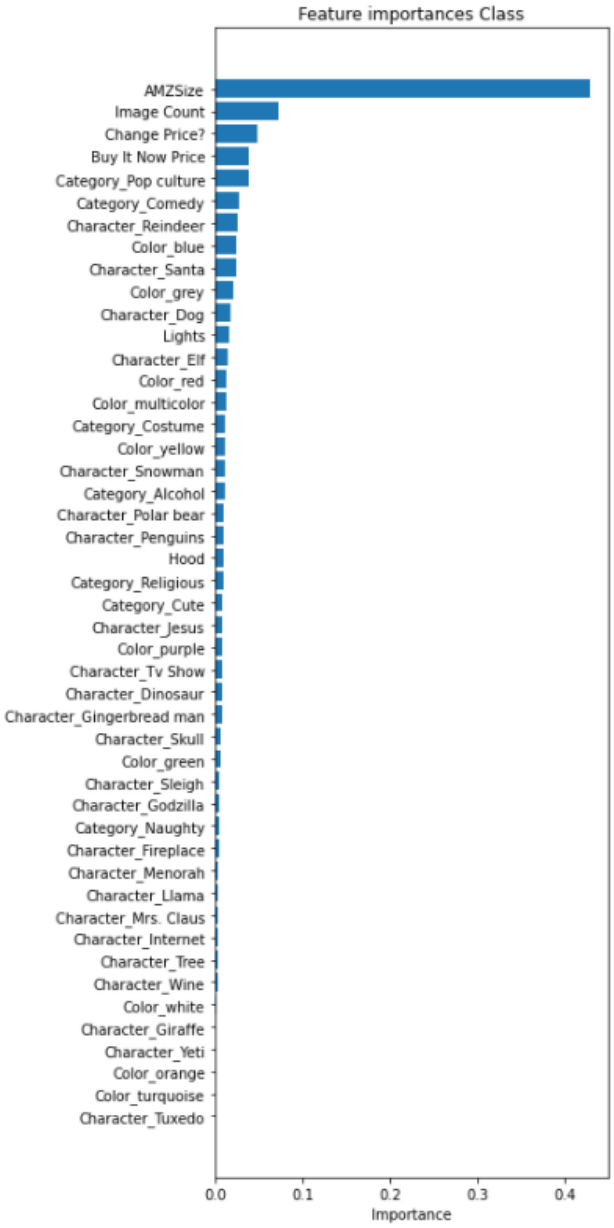

Cost of Units Remaining by Hood
Cost of Total Units Remaining
500
400
300
200
100
0
0
1
Hood

- **Image Count**


Profit by Image Count
TotalProfit
0
1000
2000
3000
4000
2
3
4
5
6
7
8
9
Image Count


Cost of Units Remaining by Image Count
Cost of Total Units Remaining
0
250
500
750
1000
1250
1500
1750
2000
2
3
4
5
6
7
8
9
Image Count

Appendix D Feature Importance Graph



Appendix E – Models

Decision Tree

```
from sklearn import tree
clf=tree.DecisionTreeRegressor(random_state=42)
clf=clf.fit(X_train, y_train)
y_train_pred=clf.predict(X_train)
y_test_pred=clf.predict(X_test)
print(clf.score(X_train, y_train))
print(clf.score(X_test, y_test))

print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
0.988338256360634
-0.042761938760697005
Training RMSE:  3.9111816875071184
Test RMSE:  46.633134408982215
```

Random Forest

```
clf_rf = RandomForestRegressor(random_state=42)
clf_rf.fit(X_train, y_train)
rf_y_train_pred=clf_rf.predict(X_train)
rf_y_test_pred=clf_rf.predict(X_test)
print(clf_rf.score(X_train, y_train))
print(clf_rf.score(X_test, y_test))


print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, rf_y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test,rf_y_test_pred)))
```

```
0.9184678135274456
0.365007466476005
Training RMSE:  10.341672736701419
Test RMSE:  36.390358253967044
```

Linear Regression

```
lr=LinearRegression()
lr=lr.fit(X_train, y_train)
y_train_pred=lr.predict(X_train)
y_test_pred=lr.predict(X_test)

print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))

print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
0.2855092164554113
0.21684658145525337
Training RMSE:  30.614316048256367
Test RMSE:  40.41340524577581
```

KNN

```
knn=KNeighborsRegressor()
knn.fit(X_train, y_train)
y_train_prediction=knn.predict(X_train)
y_prediction=knn.predict(X_test)
print(knn.score(X_train, y_train))
print(knn.score(X_test, y_test))


print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train,y_train_prediction)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test,y_prediction)))
```

```
0.41797198649515654
0.27982134324856556
Training RMSE:  27.63110097089487
Test RMSE:  38.75450061385719
```

Appendix F - Hyperparameter Tuning

Decision Tree

```
maxDepth=clf.tree_.max_depth
param_grid = {'max_depth':range(1, maxDepth+1),
              'max_features':[0.2,0.4,0.6,0.8],
              'max_leaf_nodes':[20,30,40,50],
             'min_samples_leaf': [1,2,5,10]}
clf = tree.DecisionTreeRegressor()
grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
grid_search.best_score_
```

```
{'max_depth': 16, 'max_features': 0.6, 'max_leaf_nodes': 20, 'min_samples_leaf': 1}

0.41483769073224136
```

```
results=pd.DataFrame(grid_search.cv_results_)
results.sort_values(by=['rank_test_score'], inplace=True)
results.head(10)
#order by rank test score
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_features	param_max_leaf_nodes	param_min_samples_leaf
992	0.003334	4.706415e-04	0.002333	4.711456e-04	16	0.6	20	1
1189	0.002999	1.655632e-06	0.002332	4.739557e-04	19	0.6	30	2
792	0.003668	4.709240e-04	0.002332	4.720467e-04	13	0.4	40	1
549	0.002666	4.714266e-04	0.002000	5.619580e-07	9	0.6	30	2
976	0.003000	8.171187e-04	0.002668	4.702466e-04	16	0.4	20	1
1248	0.003001	4.495664e-07	0.002667	9.440894e-04	20	0.6	20	1

```
#BEST
clf=tree.DecisionTreeRegressor(max_depth=6, max_features= 0.4, random_state=42)
clf=clf.fit(X_train, y_train)
y_train_pred=clf.predict(X_train)
y_test_pred=clf.predict(X_test)
print(clf.score(X_train, y_train))
print(clf.score(X_test, y_test))

print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
0.4975822711038823
0.30047392294196773
Training RMSE:  25.671942055023905
Test RMSE:  38.19477677461694
```

Random Forest

```
param_grid = {'max_depth':np.arange(4, 10),
              'max_features':[0.2,0.4,0.6,0.8],
              'n_estimators': [10,50,100,200,300,500,1000]}
grid_search = GridSearchCV(clf_rf, param_grid, cv=5)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
grid_search.best_score_
```

```
RandomForestRegressor(max_depth=9, max_features=0.4, n_estimators=200,
                      random_state=42)

0.4835204395671754
```

```
results=pd.DataFrame(grid_search.cv_results_)
results.sort_values(by=['rank_test_score'], inplace=True)
results.head(30)
#order by rank test score
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_features	param_n_estimators	params
150	0.374330	0.031983	0.022601	0.003877	9	0.4	200	{'max_depth': 9, 'max_features': 0.4, 'n_estim...
148	0.101607	0.008429	0.007401	0.000490	9	0.4	50	{'max_depth': 9, 'max_features': 0.4, 'n_estim...
151	0.569442	0.086988	0.030403	0.006087	9	0.4	300	{'max_depth': 9, 'max_features': 0.4, 'n_estim...
153	1.993651	0.121234	0.084606	0.009521	9	0.4	1000	{'max_depth': 9, 'max_features': 0.4, 'n_estim...

```
#BEST
clf_rf = RandomForestRegressor(max_depth=9, max_features= 0.4, n_estimators=5000,random_state=42)
clf_rf.fit(X_train, y_train)
rf_y_train_pred=clf_rf.predict(X_train)
rf_y_test_pred=clf_rf.predict(X_test)

print(clf_rf.score(X_train, y_train))
print(clf_rf.score(X_test, y_test))


print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, rf_y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test,rf_y_test_pred)))
# in order to address the overfit I would have to drastially underfit it
```

```
0.8410060180364674
0.3978976564111224
Training RMSE:  14.441642467989873
Test RMSE:  35.435387032881124
```

```
maxDepth=clf.tree_.max_depth
param_grid = {'alpha': np.arange(0,0.1,0.001),
              'random_state':[42]}
lasso= Lasso()
grid_search = GridSearchCV(lasso, param_grid, cv=5)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
grid_search.best_score_
```

```
{'alpha': 0.099, 'random_state': 42}
```

```
0.210295969602524
```

```
results=pd.DataFrame(grid_search.cv_results_)
results.sort_values(by=['rank_test_score'], inplace=True)
results.head(10)
#order by rank test score
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_random_state	params	split0_test_score	split1_test_score
99	0.004800	0.000746	0.002201	0.000400	0.099	42	{'alpha': 0.099, 'random_state': 42}	0.194880	0.227773
98	0.003801	0.000400	0.002600	0.000489	0.098	42	{'alpha': 0.098, 'random_state': 42}	0.194795	0.227879
97	0.004803	0.001163	0.002400	0.000485	0.097	42	{'alpha': 0.097, 'random_state': 42}	0.194709	0.227986
96	0.004802	0.000980	0.003199	0.000398	0.096	42	{'alpha': 0.096, 'random_state': 42}	0.194623	0.228096
95	0.004400	0.000489	0.002401	0.000490	0.095	42	{'alpha': 0.095, 'random_state': 42}	0.194537	0.228202
94	0.003600	0.000490	0.002400	0.000490	0.094	42	{'alpha': 0.094, 'random_state': 42}	0.194450	0.228308

```
lasso= Lasso(alpha=0)
lasso=lasso.fit(X_train, y_train)
y_train_pred=lasso.predict(X_train)
y_test_pred=lasso.predict(X_test)
print(lasso.score(X_train, y_train))
print(lasso.score(X_test, y_test))

print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
0.2855092164554113
0.21684658145525337
Training RMSE:  30.614316048256367
Test RMSE:  40.41340524577581
```

```
lasso= Lasso(alpha=0.099)
lasso=lasso.fit(X_train, y_train)
y_train_pred=lasso.predict(X_train)
y_test_pred=lasso.predict(X_test)
print(lasso.score(X_train, y_train))
print(lasso.score(X_test, y_test))

print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
0.2839507098864915
0.21684725410543482
Training RMSE:  30.647687103155658
Test RMSE:  40.413387890242674
```

KNN

Tuning

```
param_grid= {'n_neighbors': np.arange(1,40),
             'p':[1,2]}
grid_search= GridSearchCV(knn, param_grid, cv=3)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
grid_search.best_score_
```

```
KNeighborsRegressor(n_neighbors=10, p=1)

0.22819335079381073
```

```
results=pd.DataFrame(grid_search.cv_results_)
results.sort_values(by=['rank_test_score'], inplace=True)
results.head(30)
#order by rank test score
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	param_p	params	split0_test_score	split1_test_score	s
18	0.002667	4.717640e-04	0.002667	4.711459e-04	10	1	{'n_neighbors': 10, 'p': 1}	0.229380	0.246877	
16	0.003000	8.778064e-07	0.002000	5.150430e-07	9	1	{'n_neighbors': 9, 'p': 1}	0.244063	0.205266	
20	0.002667	4.710899e-04	0.002667	4.713704e-04	11	1	{'n_neighbors': 11, 'p': 1}	0.207859	0.220736	
22	0.002333	4.711457e-04	0.002667	4.713171e-04	12	1	{'n_neighbors': 12, 'p': 1}	0.218679	0.181953	
24	0.003001	8.485379e-07	0.002666	4.709228e-04	13	1	{'n_neighbors': 13, 'p': 1}	0.212236	0.182456	
12	0.003004	6.072267e-06	0.002997	5.886359e-06	7	1	{'n_neighbors': 7, 'p': 1}	0.260543	0.164098	

```
#BEST
knn=KNeighborsRegressor(n_neighbors=5, p=1)
knn.fit(X_train, y_train)
y_train_prediction=knn.predict(X_train)
y_prediction=knn.predict(X_test)
print(knn.score(X_train, y_train))
print(knn.score(X_test, y_test))


print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train,y_train_prediction)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test,y_prediction)))
```

```
0.4510330360444692
0.3133711503417126
Training RMSE:  26.83486104756014
Test RMSE:  37.84103838488727
```

Appendix G – Model Selection

	Score		RMSE	
Model	**Train**	**Test**	**Train**	**Test**
Linear	0.29	0.22	31	40
Decision Tree	0.5	0.3	26	38
Random Forest	0.84	0.4	14	35
KNN	0.45	0.31	27	38

```
#BEST
clf_rf = RandomForestRegressor(max_depth=9, max_features= 0.4, n_estimators=1000,random_state=42)
clf_rf.fit(X_train, y_train)
rf_y_train_pred=clf_rf.predict(X_train)
rf_y_test_pred=clf_rf.predict(X_test)

print(clf_rf.score(X_train, y_train))
print(clf_rf.score(X_test, y_test))


print("Training RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, rf_y_train_pred)))
print("Test RMSE: ", np.sqrt(metrics.mean_squared_error(y_test,rf_y_test_pred)))
```

```
0.8403700499832222
0.39657321604835716
Training RMSE:  14.4704965703458
Test RMSE:  35.47433911193609
```