AirBnb Data Analysis

Cameron Smith, Austin Nocero, Sravani Yerramaneni

Data

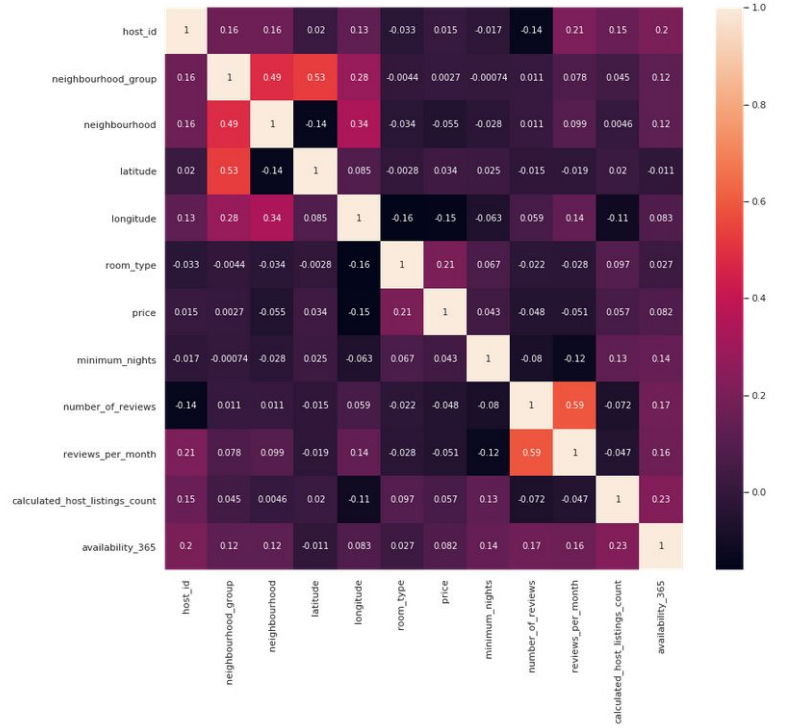
We got our data from kaggle. Our data described airbnb listings in select neighbourhoods of New York City. It has 16 features and almost 49000 rows of data.

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability_365
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	1	9	2018-10-19	0.21	6	365

Descriptive Statistics

We can find out a lot about our dataset by using descriptive statistics. Our dataset had a lot of rows of information so it was hard to understand how all of the data worked and related to one another. First we wanted to find the correlation between the data.

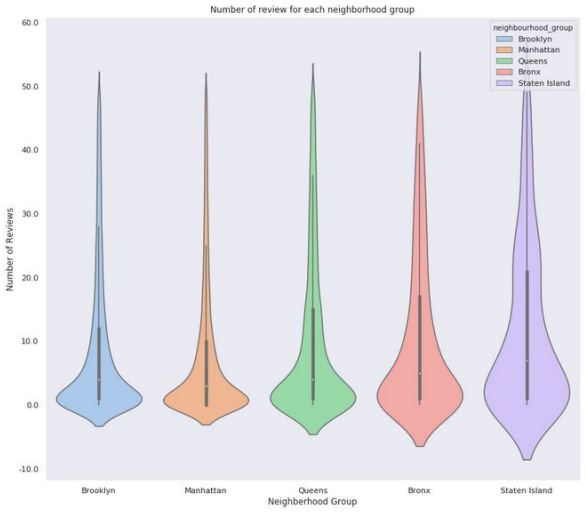
```
plt.figure(figsize=(12,12))
ax = sns.heatmap(df.corr(), annot=True)
```



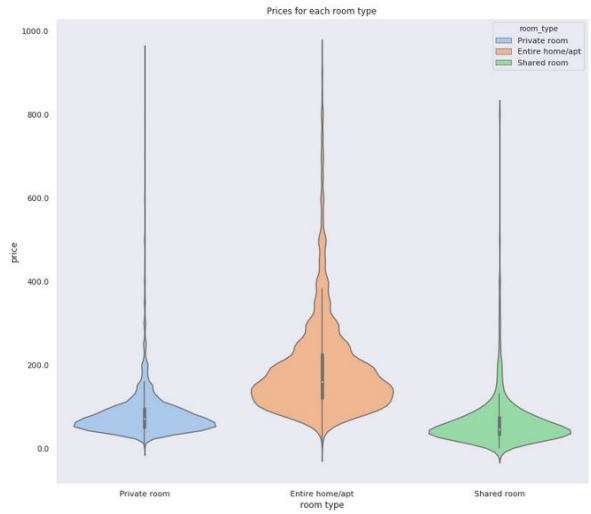
This shows how the data did not have much correlation between each other, besides latitude and neighbourhood_group because of the relation between the locations, and correlation between number_of_reviews and reviews_per_month since reviews_per_month comes from the number of reviews. Now that we know that the data is not all that correlated we can make some assumptions when we are going to be applying different algorithms and models to the data. There is still more to learn from our data as we compared different elements of the data against each other.

To attempt to visualize our data better we created more graphs.

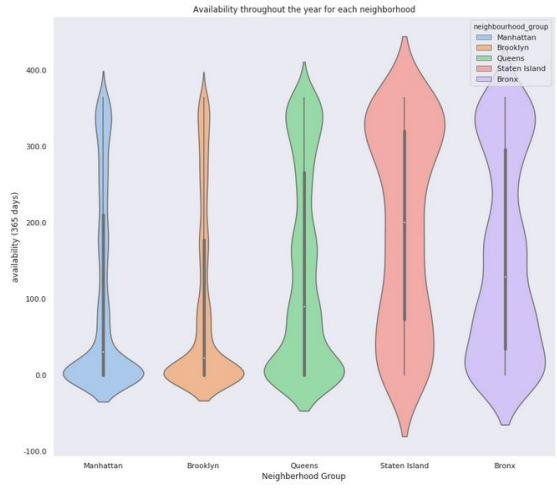
```
sns.set(style="dark")
max_review = 50

reviewbnb = data[data.number_of_reviews < max_review]

fig, ax = plt.subplots(figsize=(12, 12))
density_neigh_price_plot = sns.violinplot(ax=ax, x="neighbourhood_group", y="number_of_reviews",
                                          hue="neighbourhood_group", data=reviewbnb,
                                          palette="pastel", dodge=False)
density_neigh_price_plot.set(xlabel='Neighberhood Group', ylabel='Number of Reviews',
                             title='Number of review for each neighborhood group')
ylabels = ['{}'.format(x) for x in density_neigh_price_plot.get_yticks()]
density_neigh_price_plot.set_yticklabels(ylabels)
plt.show()
```




```
sns.set(style="dark")
price = 950

priceType = data[data.price < price]

fig, ax = plt.subplots(figsize=(12, 12))
density_neigh_price_plot = sns.violinplot(ax=ax, x="room_type", y="price",
                                          hue="room_type", data=priceType,
                                          palette="pastel", dodge=False)
density_neigh_price_plot.set(xlabel='room type', ylabel='price',
                             title='Prices for each room type')
ylabels = ['{}'.format(x) for x in density_neigh_price_plot.get_yticks()]
density_neigh_price_plot.set_yticklabels(ylabels)
plt.show()
```



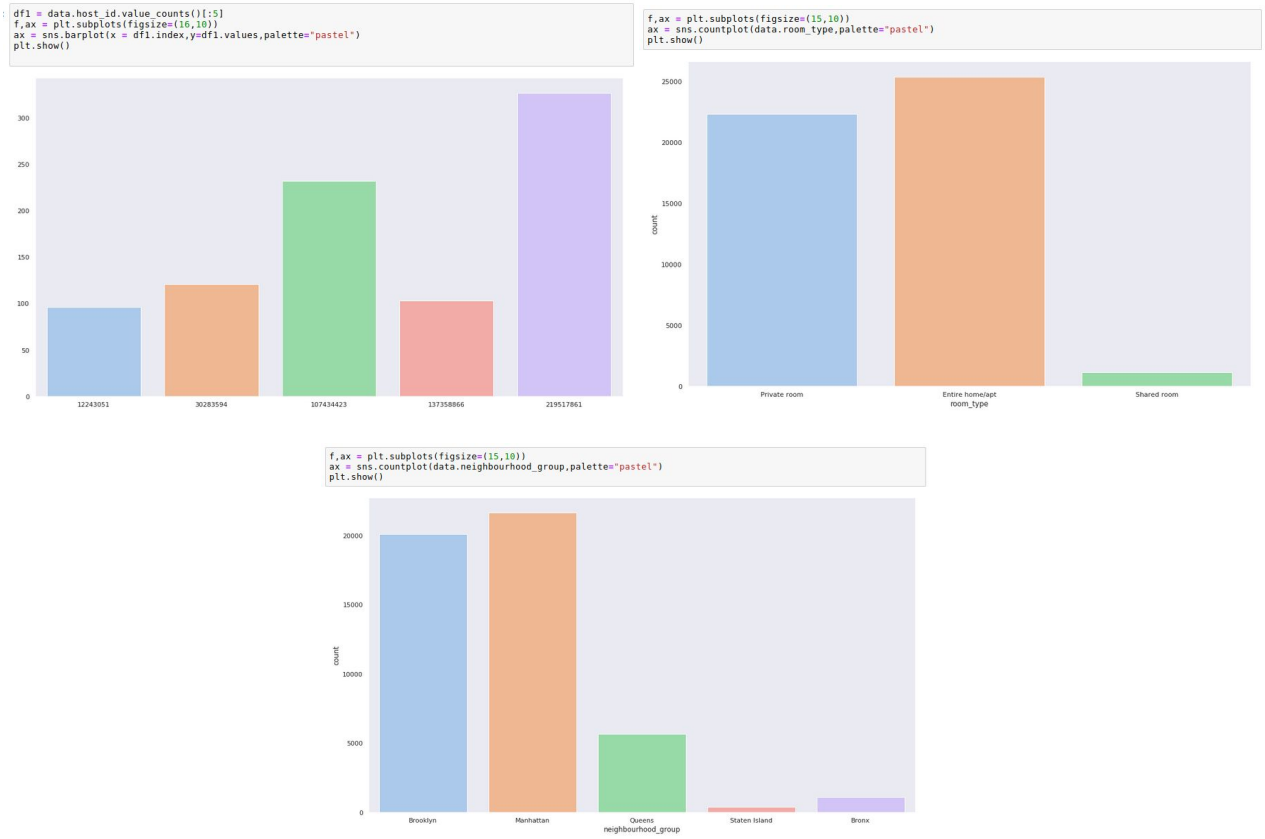

```
sns.set(style="dark")
days = 365

avl = data[data.availability_365 < days]

fig, ax = plt.subplots(figsize=(12, 12))
density_neigh_price_plot = sns.violinplot(ax=ax, x="neighbourhood_group", y="availability_365",
                                          hue="neighbourhood_group", data=avl,
                                          palette="pastel", dodge=False)
density_neigh_price_plot.set(xlabel='Neighberhood Group', ylabel='availability (365 days)',
                             title='Availability throughout the year for each neighborhood')
ylabels = ['{}'.format(x) for x in density_neigh_price_plot.get_yticks()]
density_neigh_price_plot.set_yticklabels(ylabels)
plt.show()
```



These three graphs that we plotted show different conclusions about our data. To start all three of the graphs are plotted in a violin graph format to better show where the majority of the data points lie in relation to the different axises. The first graph shows the relation between the number of reviews and the five distinct neighbourhood groups and how they relate to one another. The graph shows that on average each one of the groups has around the same

number of reviews between one and ten, but the Bronx and Staten Island both have airbnb's with more reviews, around 20 to 30, than the other three neighbourhoods. The second graph shows the average price and how it compares to each room type, the three being private room, shared room, and entire home/apartment. With this graph we can see that private and shared rooms are pretty much the same and then the entire home on average is almost double the price of the other two rooms and there are more airbnbs that are the entire home that are in the $600+ range. The final violin graph shows the five neighbourhood groups and the availability throughout the year in each one of the groups. We can see from the graph that the airbnbs in Manhattan and Brooklyn are mostly available between 0 to 50 days throughout the year while the other neighbourhood groups are mostly available year round with Staten Island even having most of its data points between the days 300 to 365, or year round availability.

We also used bar graphs as a way to visualize more of the data.



These three graphs were used to show us how the data was organized in the dataframe. The first bar graph shows the number of listings for the top five users on airbnb in the New York area. This shows us that there are users that represent a lot of the market with the total listings reaching over 300. We got these numbers by counting the repeating user_id's in the dataset. The second bar graph shows the amount of listing for each of the three room types. Private and the entire home/apartment both represent most of the data with both of them with over 20000 data points, while shared room is closer to about 2000 data points. The final bar graph shows

the amount of listings in each neighbourhood group. We can see by the graph that Manhattan and Brooklyn both have the majority of the listings throughout all of the New York neighbourhood groups

Preprocessing

When we first looked at our data we noticed that last_review and reviews per month had a large amount of null values and were also dates which made it hard to fill those values as you can't take an average of a date. We therefore decided to simply drop these features from the data. We also dropped Id, name, and host_name from the data as these were mostly unique values.

We created functions to change the categorical data into numbers to be compatible with the algorithms

```
def replace_room_type (val):
    if val == "Private room":
        return 1
    elif val == "Entire home/apt":
        return 2
    elif val == "Shared room":
        return 3

def replace_neighbourhood_group (val):
    if val == "Brooklyn":
        return 1
    elif val == "Manhattan":
        return 2
    elif val == "Queens":
        return 3
    elif val == "Staten Island":
        return 4
    elif val == "Bronx":
        return 5
```

```
t 'room_type' and 'neighbourhood_group' to numerical values
_type"] = df["room_type"].apply(replace_room_type, 'Private room')
hbourhood_group"] = df["neighbourhood_group"].apply(replace_neighbourhood_group, 'Brooklyn')
)
t 'neighbourhood' to numerical values
hbourhood"] = df["neighbourhood"].astype('category')
hbourhood"] = df["neighbourhood"].cat.codes
hbourhood"] = pd.factorize(df['neighbourhood'])[0] + 1
25)
```

We initially used the Interquartile Range to determine the outliers but that method checked every feature for outliers, even the categorical ones. This ended up removing over half the data so this was not a method that could be used. We ended up writing a function that would check for outliers and change those values to nan so that they could all be dropped together with the dropna function.

```
def detect_outlier(df):

    threshold=3
    mean_1 = np.mean(df)
    std_1 =np.std(df)


    for y in df:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            df.replace(y, np.nan, inplace=True)
```

```
detect_outlier(df.price)
detect_outlier(df.number_of_reviews)
detect_outlier(df.minimum_nights)
detect_outlier(df.availability_365)
dropdata = df.dropna()
```

Algorithms

We used the random forest, k-means, and linear regression algorithms to measure our data.

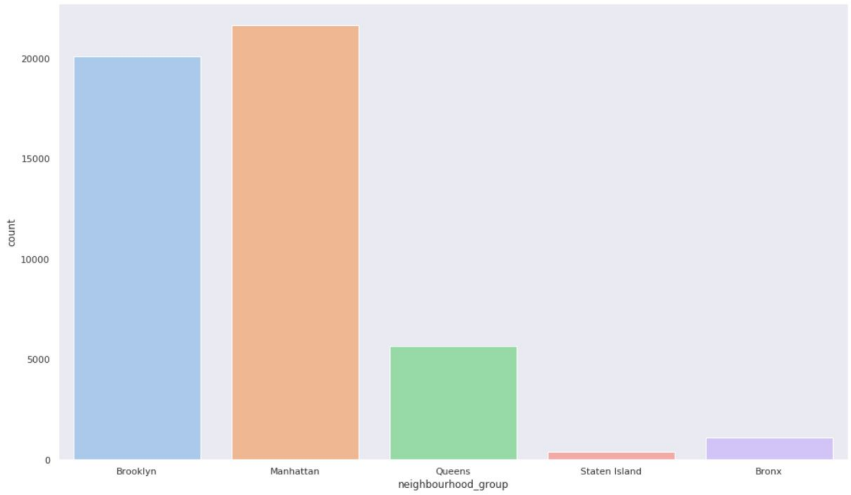
Random Forest

For our random forest classifier, we aimed to classify the neighbourhood group which was one of five values. Ordered from the highest number of occurrences to the lowest number of occurrences, the neighbourhood groups are Manhattan, Brooklyn, Queens, Bronx, Staten Island.

```
## Random Forest Classifier

import pandas as pd
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import numpy as mp


x = df.drop(columns=['neighbourhood_group'])
y = df['neighbourhood_group']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33)

clf = RandomForestClassifier(n_estimators=20)
clf.fit(x_train, y_train)
pred_train = clf.predict(x_train)
pred_test = clf.predict(x_test)


from sklearn.metrics import accuracy_score
print("Accuracy: ", accuracy_score(y_test, pred_test))
```

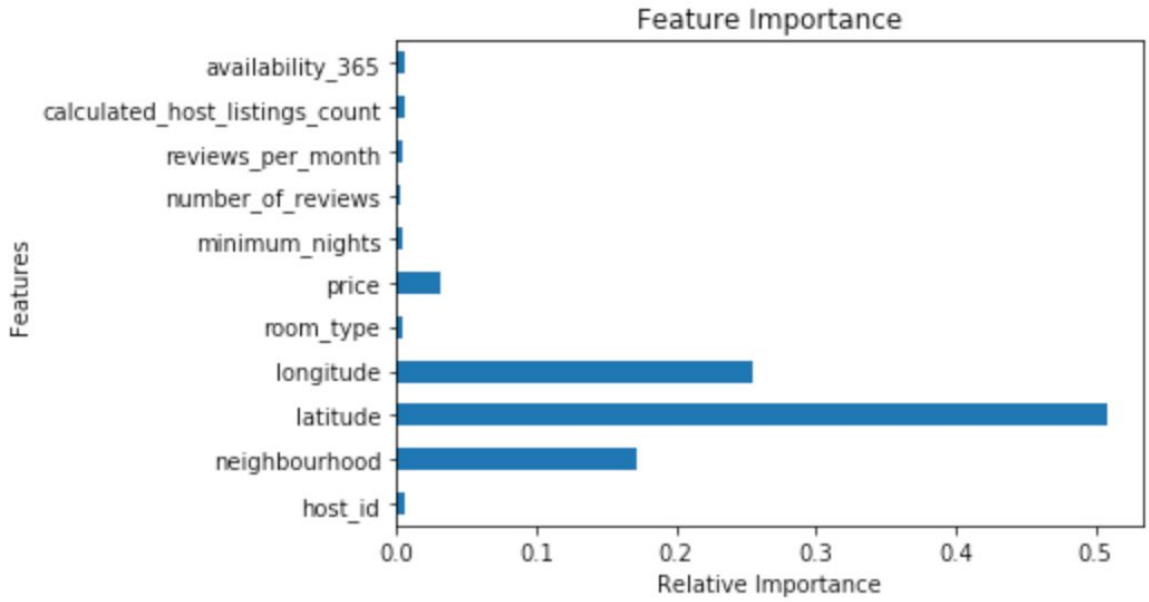
```
Accuracy:  0.9995661874070402
```

Our random forest classifier had an accuracy rate of 99.9%. The reason for this can be seen in this countplot of the occurrences of each neighbourhood group.



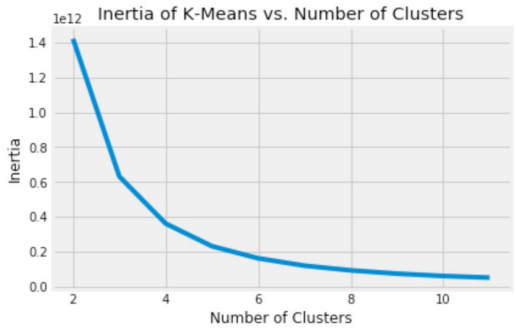
Manhattan takes up almost 50% of the data. If our random forest classifier classified every data point it encountered as Manhattan, it would achieve a 50% accuracy rate.

The misleading high accuracy rate can be fixed by upsampling the minority neighbourhood groups (i.e. Queens, Staten Island, and the Bronx) or downsampling the majority classes (Manhattan and Brooklyn), or a combination of the two.



From the chart above, we can see which features were the most important in deciding which neighbourhood group a data point belonged to.

K-means




```
## Choosing a value for k

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns

# K-Means clusters data in n groups of equal variance by minimizing
# inertia, also known as, within-cluster sum-of-squares

scores = [10]
for (i in range(10)):
    scores[i]= KMeans(n_clusters=i+2).fit(df).inertia_


sns.lineplot(np.arange(2, 12), scores)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title("Inertia of K-Means vs. Number of Clusters")
```

Before implementing K-Means, we first plotted a graph to help us choose how many clusters we want to use in our K-Means algorithm.

Inertia is a measure of how internally coherent clusters are and K-Means seeks to minimize this. So in the graph, we want to find the spot of diminishing returns where the decrease in inertia slopes out gradually. In this case, the ideal number of clusters is 5.

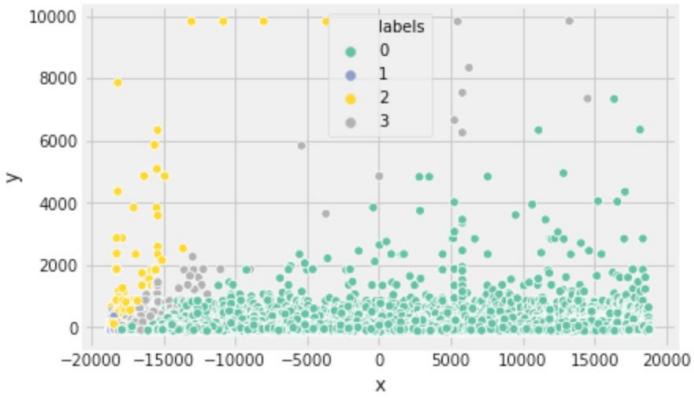
```
## Error associated with each clustering algorithm

from sklearn import preprocessing
from sklearn.metrics import silhouette_score

# K-Means
kmeans = KMeans(n_clusters=5).fit(df)

# Cosine K-Means
normalized_vectors = preprocessing.normalize(df)
normalized_kmeans = KMeans(n_clusters=4).fit(normalized_vectors)

# Show associated silhouette score (ranges from -1 to 1; 1 is ideal)
print('kmeans: ', silhouette_score(df, kmeans.labels_, metric='euclidean')
print('cosine kmeans: ', silhouette_score(normalized_vectors, normalized_k
```

```
kmeans:  0.556040767094206
cosine kmeans:  0.9300384637314381
```

The picture above shows the silhouette scores of the K-Means algorithm using the Euclidean and Cosine distances.

We used the silhouette score to judge accuracy. The silhouette value is a measure of how similar an object is to its own cluster, where a high value indicates that the object is well matched to its own cluster. This indicates the clustering configuration is appropriate.

Cosine distance usually works better when there are a lot of features because at high dimensionality, euclidean distances have less meaning since they are often pretty close together.




```
#PCA showing K-Means

axes = ['x', 'y', 'z']
PCA_df = pd.DataFrame(PCA(3).fit_transform(df))
for (i in range(3)):
    PCA_df.rename({i:axes[i], axis=1, inplace=True})
    PCA_df['labels'] = normalized_kmeans.labels_

sns.scatterplot(x=pca_df.x, y=pca_df.y, hue=pca_df.labels, palette="Set2")
```

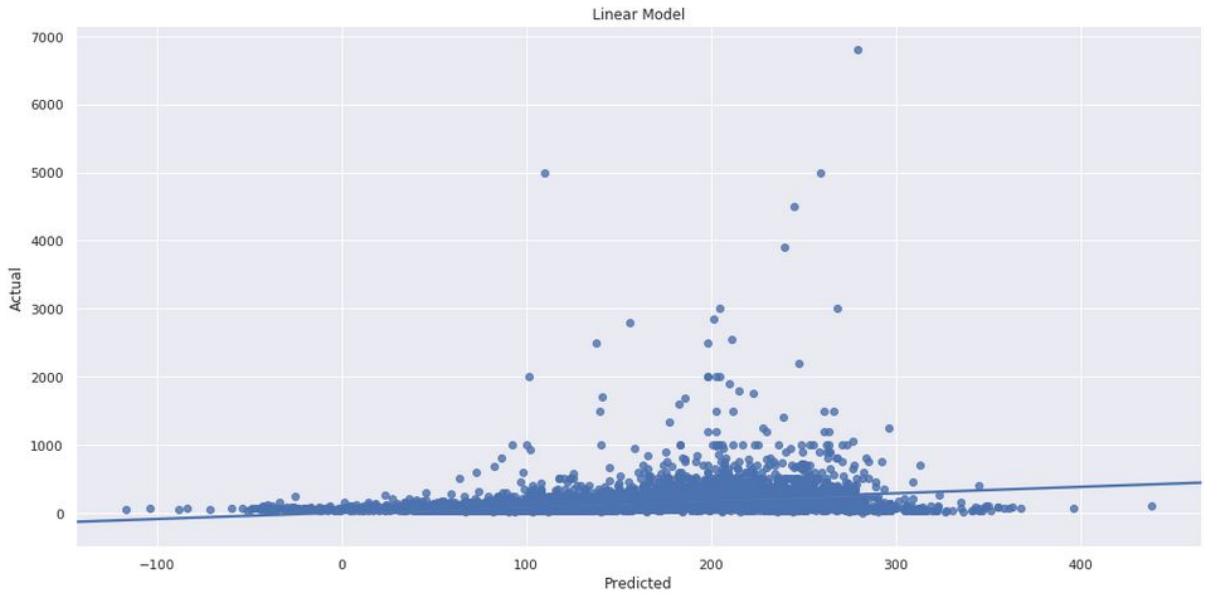
We used PCA for dimensionality reduction to be able to visualize the K-Means clustering. However, many of the 3D points cluster near each other and overlap in 2D space. The x- and y-axes also seemingly don't measure anything valuable for analysis either.

Linear Regression

For linear regression we started by applying linear regression to the original dataset before applying any kind of algorithms to remove the outliers of the data.

```
Mean Squared Error: 181.810009264
R2 Score: 9.81971075263
```

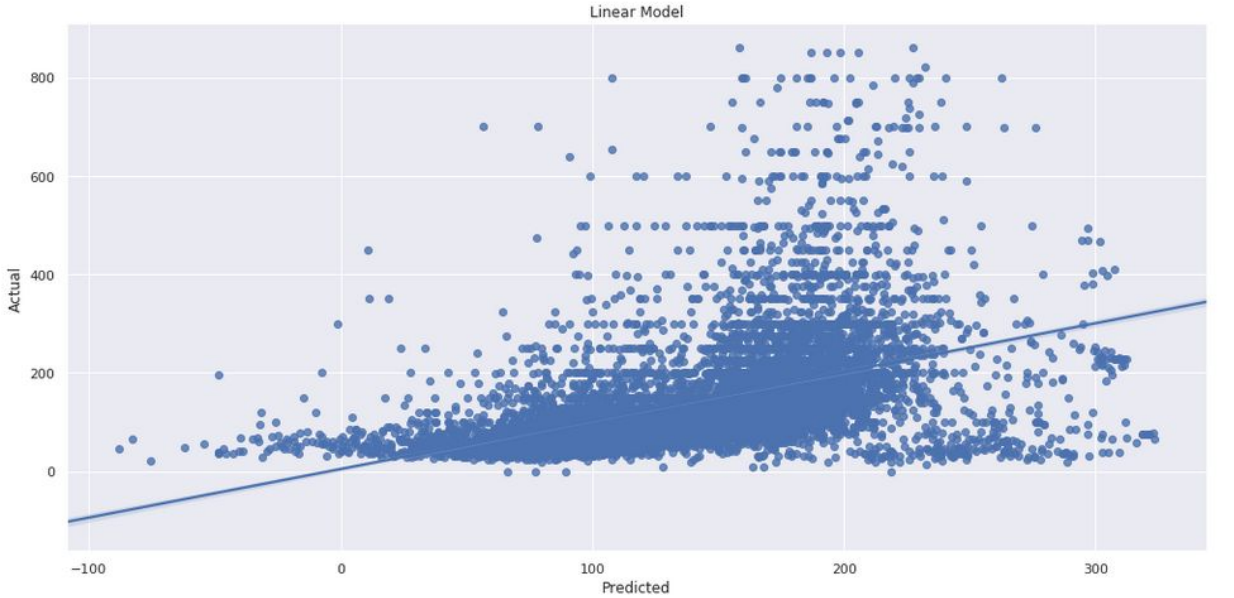
```
plt.figure(figsize=(15,8))
sns.regplot(predicts, y_test)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Linear Model")
plt.grid(True)
plt.show()
```



Our result was not great, we had a high mean squared error and a low R squared score. This was an expected result due to how low the correlation between our data was when we measured it. We still wanted to get better results so we removed the outliers in our data by using z-score and then we applied linear regression again to the modified dataset.

```
Mean Squared Error: 97.2393246153
R2 Score: 23.7025898533
```

```
plt.figure(figsize=(15,8))
sns.regplot(predicts, y_test)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Linear Model")
plt.grid(True)
plt.show()
```



After removing some of the outliers we can see an increase in our R squared score and a decrease in our mean squared error. Removing outliers using z-score gave us increased accuracy to our model. We wanted to try and increase the accuracy more but we were having problems with other methods. One of those methods being IQR to remove more outliers. We applied IQR to the dataset and we saw an increase to our linear regression model, but by using IQR we removed too much data. We were left with about 26000 rows, which is almost half of the data that we started with gone. We decided that this was too much data loss to continue using IQR even if we got better accuracy with our model.