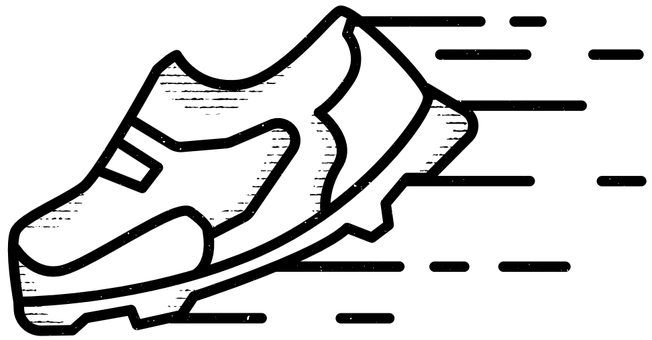
ARE YOU A SNEAKERHEAD?

SC1015 BCF2 GROUP 7:
XU MINTING (U2121554G)
TAN PUAY JUN, KLAUS (U2122365E)
SHERMAINE NG JINGLIN (U2121773F)

HELLO JACK



How does Jack know what kind of shoes to snag & resell?

PRACTICAL MOTIVATION

shoe brands, shoe models

REAL-WORLD PROBLEM

Concerns regarding what type of shoes and which shoe features appeal the most to the public, making resale of shoes most profitable.

shoe sizes, shoe colours

resale period, profitability

PROBLEM FORMULATION

DATA SCIENCE PROBLEM - Anomaly Detection

Analysing which shoe size and shoe colour will yield unusually high profits, through detecting price anomalies from the given data.

VARIABLES

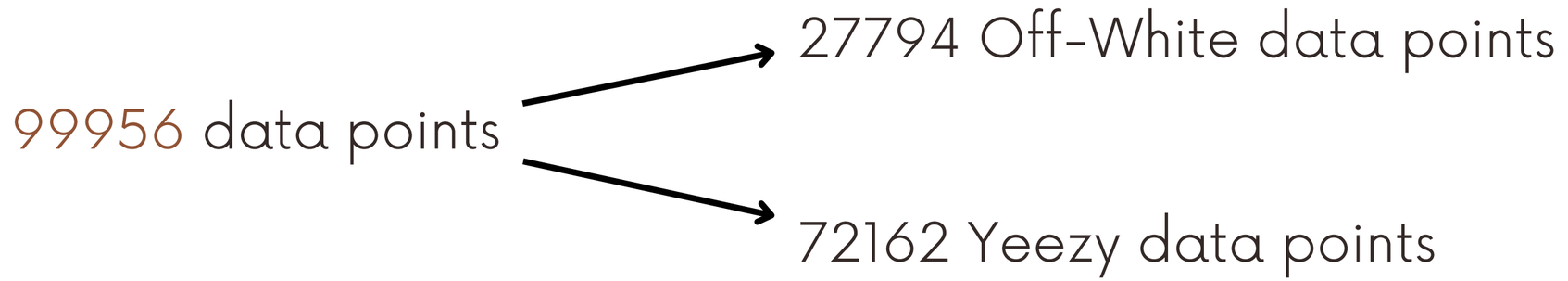
Size
Colour

ANOMALY

Deviation from the data set's norm

Data set was taken from StockX, an online marketplace and reseller.

Data set consists of all Off-White x Nike and Yeezy 350 sales from 1st September 2017 to 15th February 2019.



Data set consists of the following 8 variables:
Order Date, Brand, Sneaker Name, Sale Price, Retail Price, Release Date, Shoe Size, Buyer State.

DATA SET

DATA CLEANING

IMPORTING LIBRARIES

```
#importing the libraries
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
sb.set()
```

LOADING DATA SET

```
#loading raw data
sneakerData = pd.read_excel('StockX-Data-Contest-2019-3.xlsx',sheet_name=1)
sneakerData.info()
```

DATA RELATED TO PRICE AND PROFITS

Profit = Sale Price - Retail Price

```
#cleaning the price column
sneakerData['Profit'] = sneakerData['Sale Price'] - sneakerData['Retail Price']
sneakerData['Profit Ratio'] = (sneakerData['Profit'] / sneakerData['Retail Price']).round(2)
```

DATA CLEANING

DATA RELATED TO DATES

```
#cleaning the dates column
sneakerData['Order Date'] = sneakerData['Order Date'].astype('datetime64[ns]')
sneakerData['Release Date'] = sneakerData['Release Date'].astype('datetime64[ns]')
sneakerData['Turnover Days'] = sneakerData['Order Date'] - sneakerData['Release Date']
```

String type

Release Date
24/9/16
23/11/16
23/11/16
23/11/16
11/2/17
11/2/17
17/12/16
29/4/17

Datetime type

Release Date
2016-09-24
2016-11-23
2016-11-23
2016-11-23
2017-02-11
2017-02-11
2016-12-17
2017-04-29

DATA CLEANING

DATA RELATED TO MODELS

```
#breaking the Sneaker Name into Model Variants (categorical data)
sneakerData['Sneaker Name'] = sneakerData['Sneaker Name'].str.lower()

sneakerData["Model"] = sneakerData['Sneaker Name'].apply(
    lambda x : 'yeezy-boost-350' if 'yeezy' in x.split("-") else (
        'air-jordan-1-retro-high' if 'jordan' in x.split('-') else (
            'air-force-1' if 'force' in x.split('-') else(
                'air-max-90' if '90' in x.split('-') else (
                    'air-max-97' if '97' in x.split('-') else (
                        'air-presto' if 'presto' in x.split('-') else (
                            'air-vapormax' if 'vapormax' in x.split('-') else (
                                'blazer-mid' if 'blazer' in x.split('-') else (
                                    'react-hyperdunk-2017-flyknit' if 'hyperdunk' in x.split('-') else (
                                        'zoom-fly' if 'zoom' in x.split('-') else (np.nan)
                                    )
                                )
                            )
                        )
                    )
                )
            )
        )
    )
#check if all the Sneaker Names are categorised into Model Variants
uncategorised_model = pd.DataFrame()
uncategorised_model[sneakerData['Model'].isnull() == True]
print(uncategorised_model)
```

```
Empty DataFrame
Columns: []
Index: []
```

All sneakers have been categorised into model variants

DATA CLEANING

DATA RELATED TO COLOURS

```
#importing the colour information from stockX website, based on each sneaker name
colourway = pd.read_excel('supplemental_data_colorway.xlsx')
colourway['Style'] = colourway['Style'].str.lower()
colourway.head()

#merging the colourway DF with the sneakerData DF
sneakerData = pd.merge(sneakerData,colourway,how='left',left_on='Sneaker Name',right_on='Style')
```

After merging the two dataframes, irrelevant columns were dropped

```
#Dropping columns that are irrelevant to the analysis
parsed_data = sneakerData.drop(['Sale Price','Retail Price','Style','Number of Sales','Website','Product Line'],axis=1)
parsed_data['Turnover Days'] = parsed_data['Turnover Days'].dt.days
parsed_data['Turnover Weeks'] = (parsed_data['Turnover Days'] / 7).round(0)
parsed_data = parsed_data.drop(parsed_data[parsed_data['Turnover Weeks'] < 0].index)
parsed_data = parsed_data.drop(parsed_data[parsed_data['Turnover Weeks'] > 52].index)
```

Adding a new primary colour column

```
#consolidating the primary colors into 1 single column
def get_col(row):
    for color in parsed_data.columns[10:20]:
        if row[color] == 1:
            return color
parsed_data['color'] = parsed_data.apply(get_col, axis=1)
```

EXPLORATORY DATA ANALYSIS

UNIVARIATE ANALYSIS

NUMERICAL DATA

Narrowing down to numerical data only

```
#picking out numerical data
numericalvar=parsed_data[['Shoe Size','Turnover Days','Profit','Profit Ratio']]
```

- Violin Plot shows the variable's density, center and spread
- White dot -- Median
- Length of box plot -- Interquartile Range (IQR)

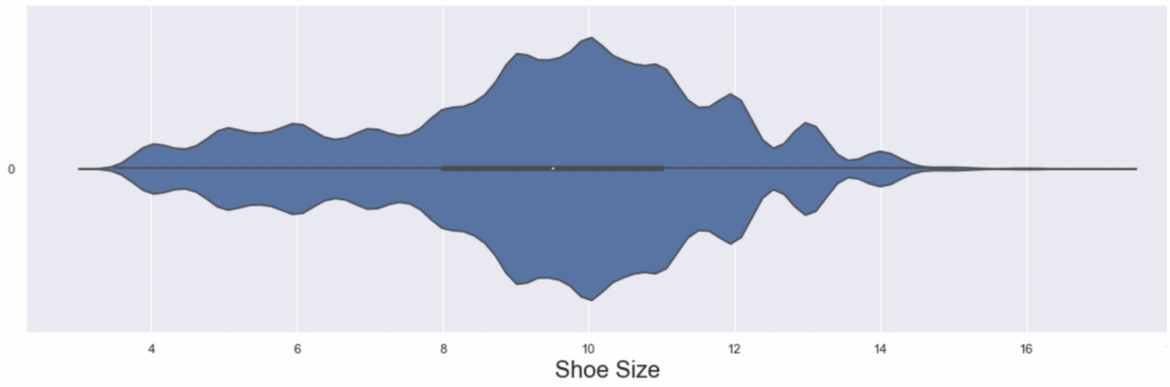
```
#plotting violin plots for numerical data
f, axes = plt.subplots(4, 1, figsize=(18, 24))

count = 0
for var in numericalvar:
    sb.violinplot(data = numericalvar[var], orient = "h", ax = axes[count])
    ax = axes[count]
    ax.set_xlabel(var,fontsize=20)
    count += 1
```

EXPLORATORY DATA ANALYSIS

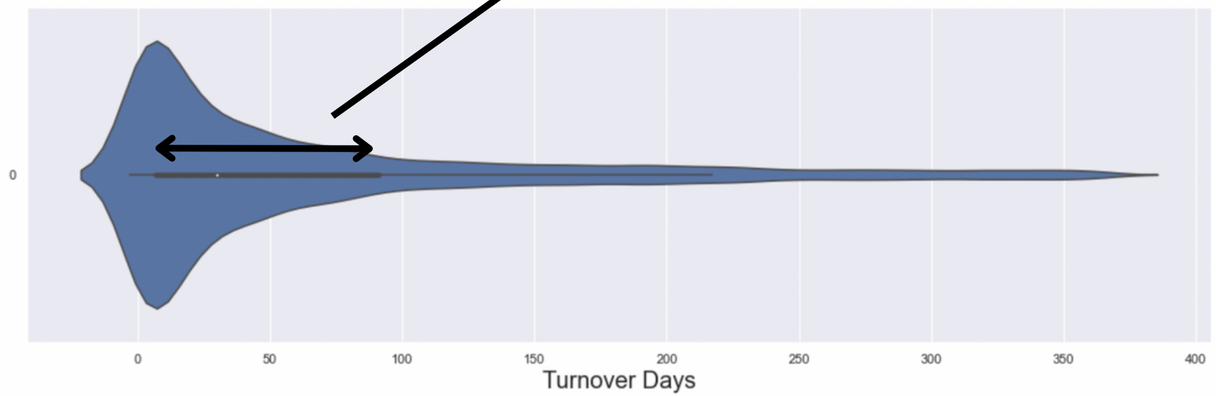
UNIVARIATE ANALYSIS

NUMERICAL DATA

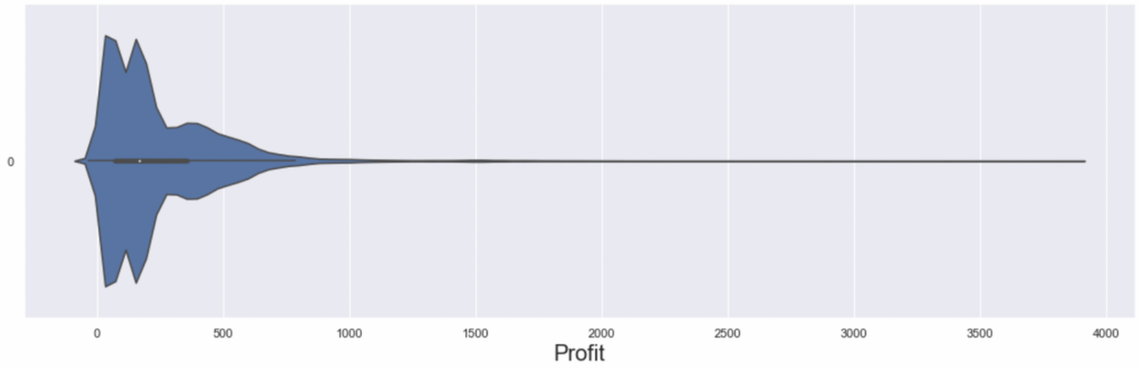
Shoe Sizes



Turnover Days



Profit



Profit Ratio

EXPLORATORY DATA ANALYSIS

CATEGORICAL DATA

UNIVARIATE ANALYSIS

Narrowing down to categorical data only

```
#picking out categorical data
categoricalvar=parsed_data[['Model','Buyer Region','color']]
```

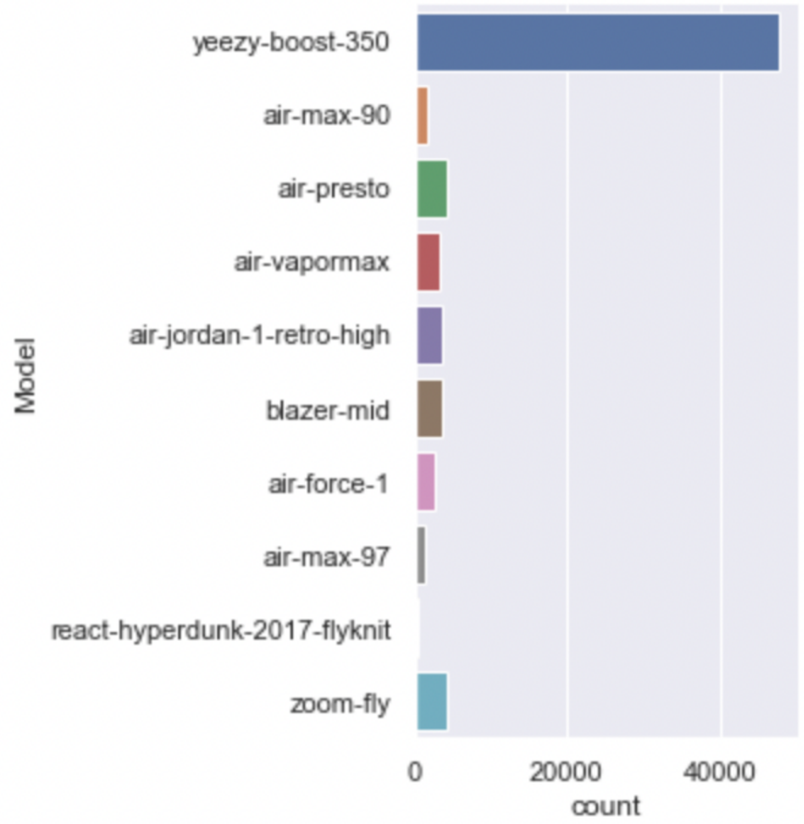
EXPLORATORY DATA ANALYSIS

UNIVARIATE ANALYSIS

CATEGORICAL DATA

Shoe Models

```
#plotting catplot for shoe models
sb.catplot(y = "Model", data = categoricalvar, kind = "count")
```



- Yeezy-Boost-350 is the best-selling model
- React-Hyperdunk-2017-Flyknit is the worst-selling model

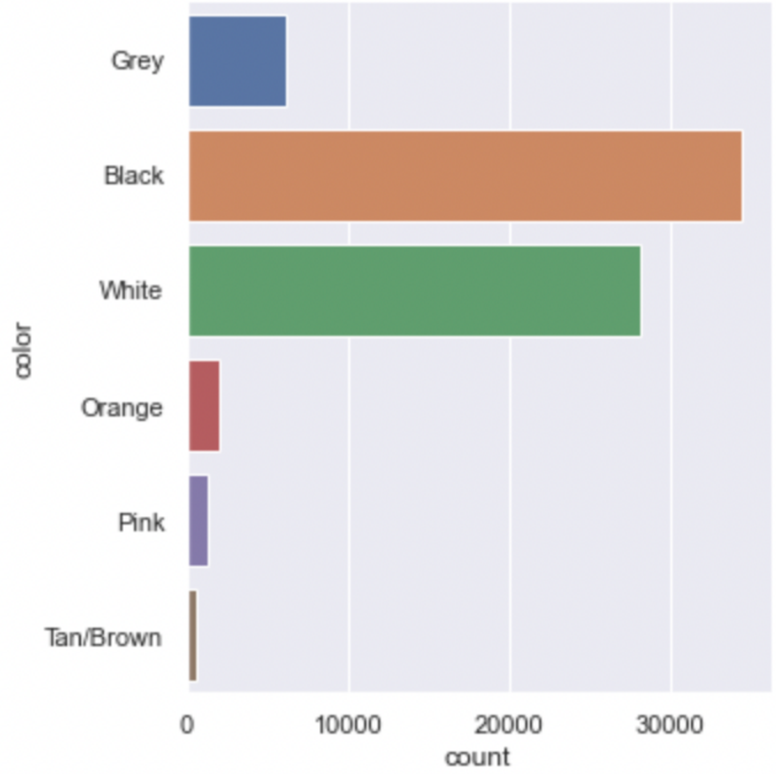
EXPLORATORY DATA ANALYSIS

UNIVARIATE ANALYSIS

CATEGORICAL DATA

Shoe Colours

```
#plotting catplot for shoe colours
sb.catplot(y = "color", data = categoricalvar, kind = "count")
```



- Black is the best-selling primary shoe colour, followed by White
- Tan/Brown is the worst-selling primary shoe colour

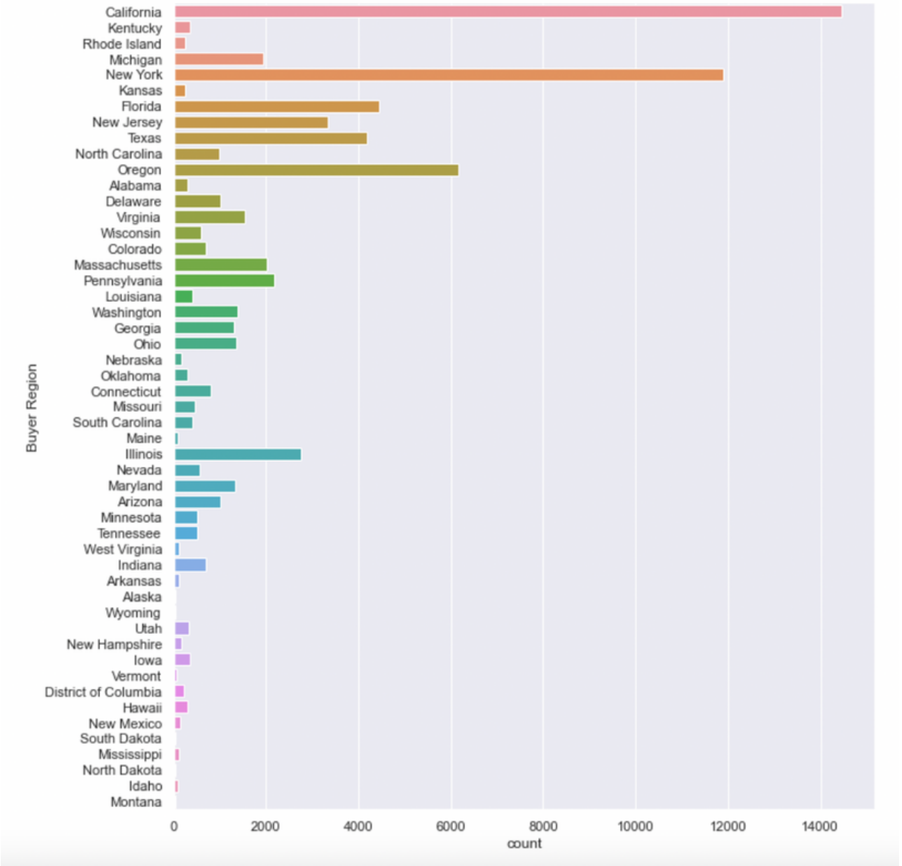
EXPLORATORY DATA ANALYSIS

UNIVARIATE ANALYSIS

CATEGORICAL DATA

Buyer Regions

```
#plotting catplot for buyer regions
plt.figure(figsize=(20,10))
sb.catplot(y = "Buyer Region", data = categoricalvar, kind = "count",height=10, aspect=1)
```



- California saw the highest sales, followed by New York
- Buyer regions may not be as relevant a variable since it limits our analysis to only US sales

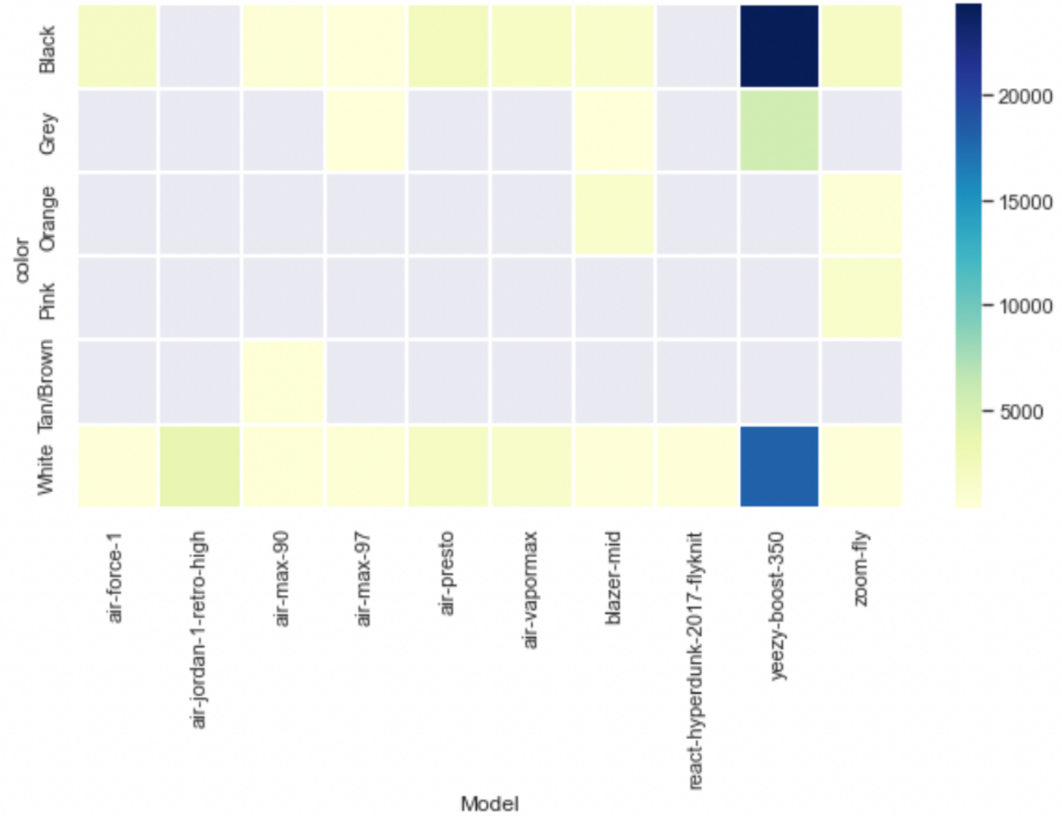
EXPLORATORY DATA ANALYSIS

DISTRIBUTION OF COLOURS AND MODELS

BIVARIATE ANALYSIS

```
#creating dataframe of colors and model
color_model_data = pd.DataFrame(parsed_data[['color','Model']])
color_model_data.head()

#distribution of colors and models
f = plt.figure(figsize=(10, 5))
sb.heatmap(color_model_data.groupby(['color','Model']).size().unstack(),linewidths=1, cmap = "YlGnBu")
```

- Create dataframe, followed by a heatplot
- The heatplot shows the number of data points for each colour corresponding to each model
- Black Yeezy-Boost-350 was the best-selling shoe



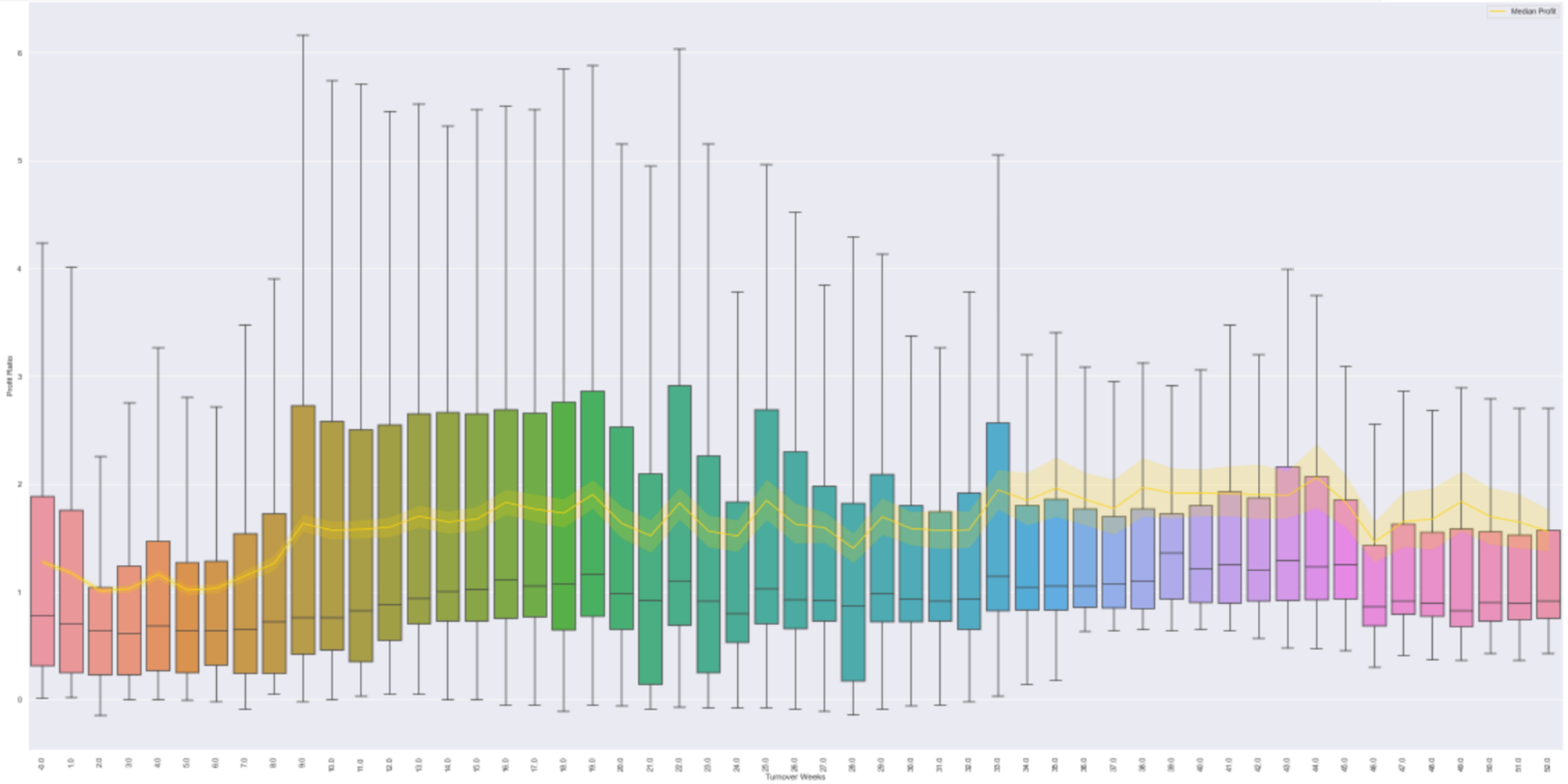
EXPLORATORY DATA ANALYSIS

PROFITABILITY ACROSS TURNOVER WEEKS

BIVARIATE ANALYSIS

```
#profitability across turnover weeks
f,axes = plt.subplots(figsize=(40,20))
plt.xticks(rotation=90)
sb.boxplot(x=parsed_data['Turnover Weeks'],y=parsed_data['Profit Ratio'],data=parsed_data,showfliers=False,ax=axes)
sb.lineplot(x=parsed_data['Turnover Weeks'],y=parsed_data['Profit Ratio'],data=parsed_data,color='gold',
            label='Median Profit',ax=axes)
```

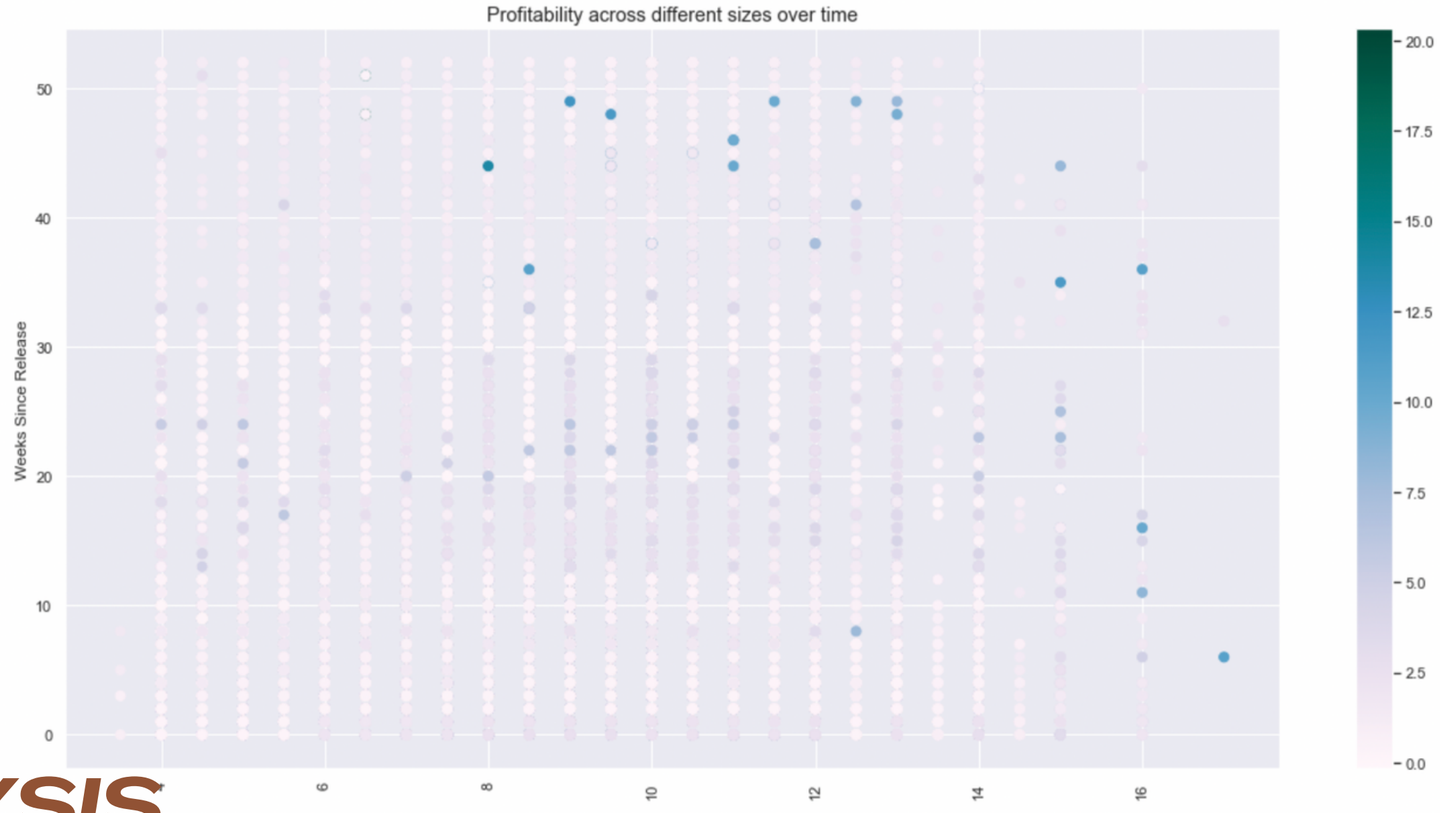
- Profit Ratio vs Turnover Weeks
- Week 33 - Week 44 saw a period of higher profits
- As the weeks go by, the profit range decreases as seen by the decreasing length of boxplot



EXPLORATORY DATA ANALYSIS

PROFITABILITY ACROSS SHOE SIZES OVER TIME

```
#plotting profit ratio, shoe sizes and turnover weeks
fig, axes = plt.subplots(figsize = (20,10))
plt.xticks(rotation=90)
plot = axes.scatter(x=parsed_data['Shoe Size'],y=parsed_data['Turnover Weeks'], c=parsed_data['Profit Ratio'],
                    cmap='PuBuGn',s=50)
fig.colorbar(plot, ax=axes)
plt.ylabel('Weeks Since Release')
plt.title('Profitability across different sizes over time', fontsize = 'large')
plt.show()
```



- Shoe sizes 4 to 7 had a stable spread of profits across the year
- Shoe sizes 8 to 13 had a spike in profits nearing the last few weeks of the year

MULTIVARIATE ANALYSIS

EXPLORATORY DATA ANALYSIS

PROFITABILITY ACROSS SHOE COLOURS OVER TIME

```
#plotting profit ratio, shoe colours and turnover weeks
fig, axes = plt.subplots(figsize = (20,10))
plt.xticks(rotation=90)
plot = axes.scatter(x=parsed_data['color'],y=parsed_data['Turnover Weeks'], c=parsed_data['Profit Ratio'],
                    cmap='PuBuGn',s=50)
fig.colorbar(plot, ax=axes)
plt.ylabel('Weeks Since Release')
plt.title('Price Premium on different colours over time', fontsize = 'large')
plt.show()
```

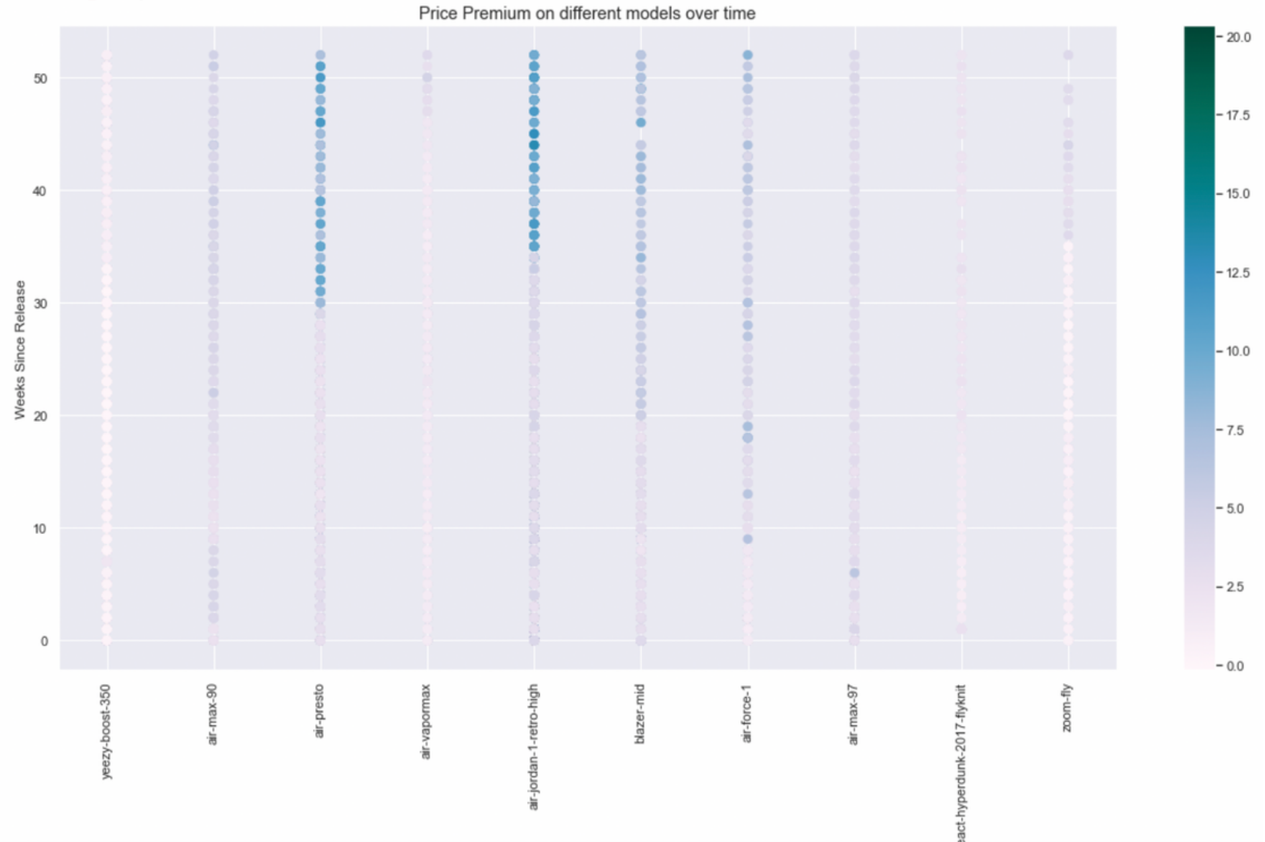
- White shoes saw the highest profits nearing the last few weeks of the year
- Orange shoes saw decreasing profits as the weeks went by

MULTIVARIATE ANALYSIS

EXPLORATORY DATA ANALYSIS

PROFITABILITY ACROSS SHOE MODELS OVER TIME

```
#plotting profit ratio, shoe models and turnover weeks
fig, axes = plt.subplots(figsize = (20,10))
plt.xticks(rotation=90)
plot = axes.scatter(x=parsed_data['Model'],y=parsed_data['Turnover Weeks'], c=parsed_data['Profit Ratio'],
                    cmap='PuBuGn',s=50)
fig.colorbar(plot, ax=axes)
plt.ylabel('Weeks Since Release')
plt.title('Price Premium on different models over time', fontsize = 'large')
plt.show()
```



- Air-Jordan-1-Retro-High and Air-Pesto were two models that showed increasing profits as the weeks went by

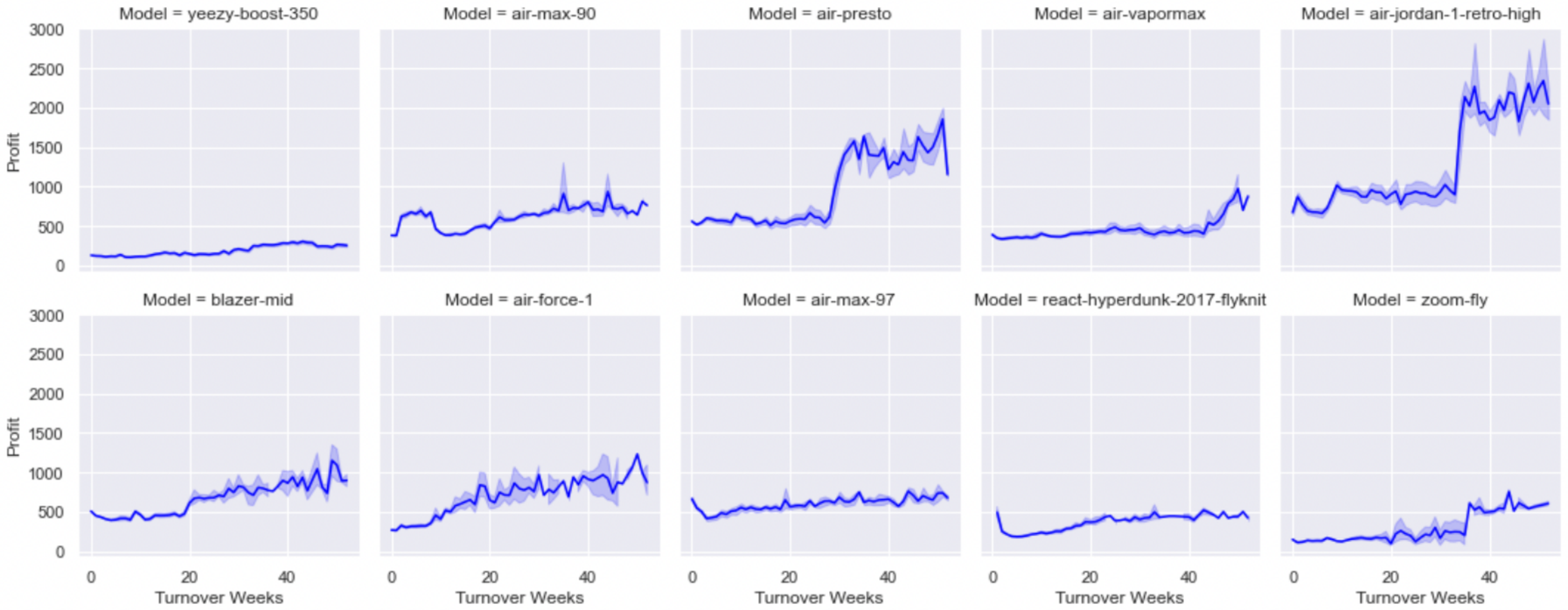
MULTIVARIATE ANALYSIS

EXPLORATORY DATA ANALYSIS

PROFITABILITY ACROSS SHOE MODELS OVER TIME

```
#lineplot of profit ratio, shoe models and turnover weeks
g = sb.FacetGrid(parsed_data, col='Model',col_wrap=5)
g.map(sb.lineplot, 'Turnover Weeks', 'Profit', color='blue')
```

- Air-Jordan-1-Retro-High and Air-Pesto were two models that showed abnormally high profits during the last few weeks of the year



MULTIVARIATE ANALYSIS

MACHINE LEARNING

Identifying Anomalies in Data

IDENTIFYING OUTLIERS
DBSCAN - Density-Based Spatial Clustering of Application with Noise

```
from sklearn.cluster import DBSCAN
base_dbscan=DBSCAN(eps=0.1,min_samples=10) #base model
base_dbscan.fit(parsed_data[['Profit Ratio']])
#plot the bdscan result
plt.figure(figsize=(20,10))
plt.scatter(parsed_data['Profit Ratio'],base_dbscan.labels_,c=base_dbscan.labels_)
```

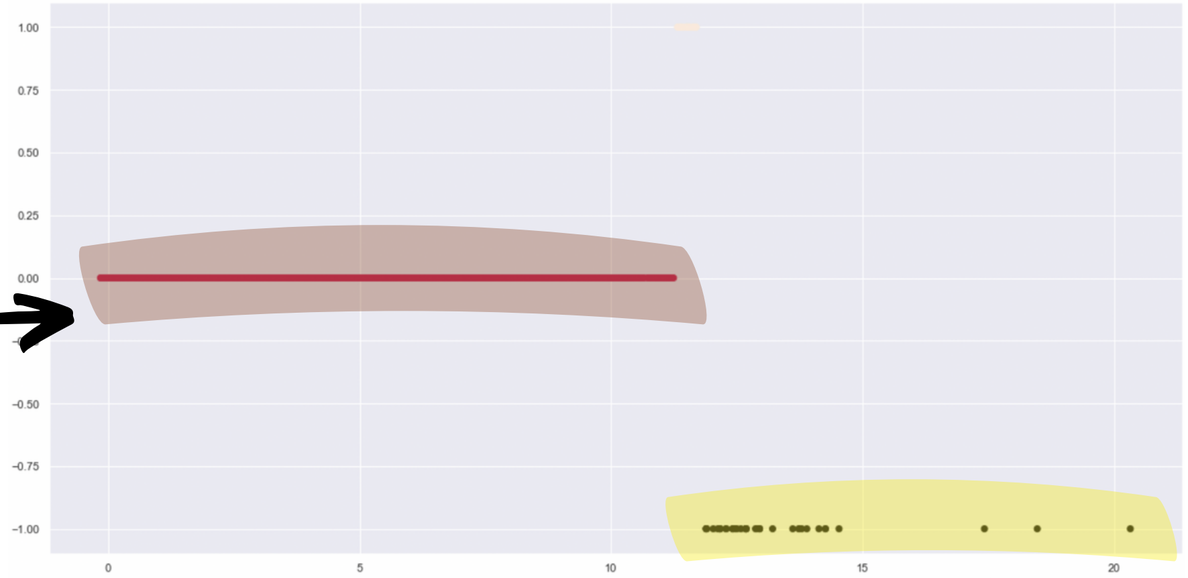
Parameters in DBSCAN:

- eps: Epsilon -- The radius of circle to be created around each data point to check for density
- min_samples: Minimum number of other data points to be inside 1 circle for the data point to be classified as a core point

MACHINE LEARNING

DBSCAN
Visualising the result of default model

```
from sklearn.cluster import DBSCAN
base_dbscan=DBSCAN(eps=0.1,min_samples=10) #base model
base_dbscan.fit(parsed_data[['Profit Ratio']])
#plot the bdscan result
plt.figure(figsize=(20,10))
plt.scatter(parsed_data['Profit Ratio'],base_dbscan.labels_,c=base_dbscan.labels_)
```



Outlier detection on 'Profit Ratio':

- Without optimising the model, most of the data points in parsed_data ['Profit Ratio'] are treated as core data points (with a label value greater or equal to 0)

- Data points with a label value of -1 are treated as noise, hence identfied as outliers

MACHINE LEARNING

DBSCAN
Optimising the DBSCAN model

MACHINE LEARNING

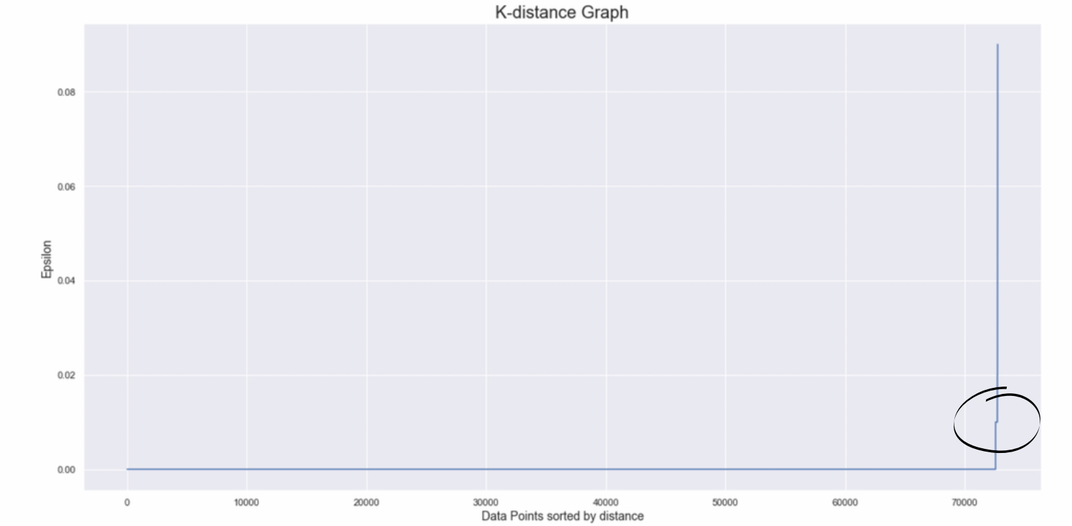
Determining the min_samples

- There is no computational way to determine the min_samples used in DBSCAN model
- Generally the min_samples should be greater or equal to the dimensions of the data used, for a multi-dimensional data like ours, the suggested min_samples = 2 * data dimensions
- Hence, min_samples = 2 * 23 = 46

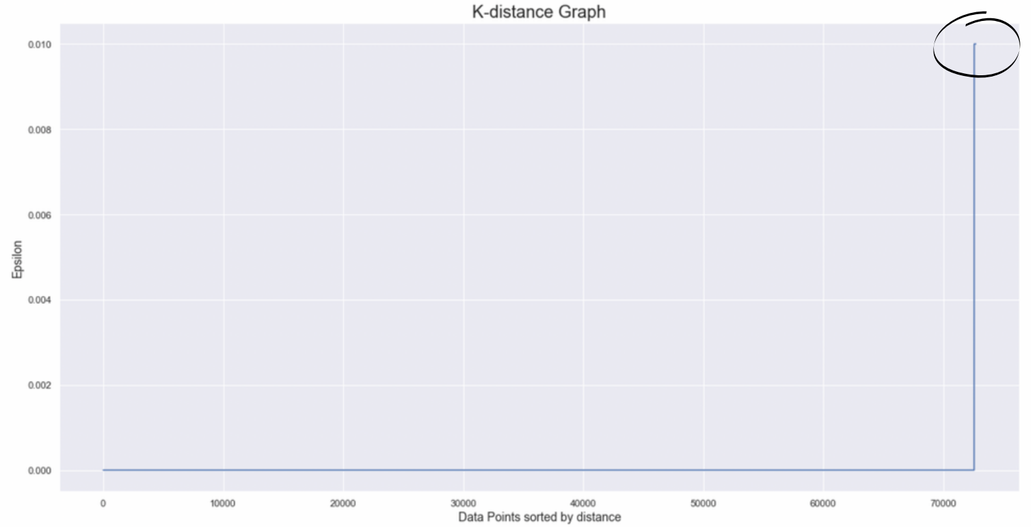
Determining the Epsilon

- One way to compute the optimal Epsilon is to plot a K-distance graph.
- This involves calculating the average distance between each data point and its K nearest neighbours, where K = min_samples
- The average K-points are then plotted in ascending order on a K-distance graph
- The optimal value of Epsilon is at the point where the graph has the maximum gradient

DBSCAN
Optimising the DBSCAN model




```python
#plotting a k-distance graph
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=46)
nbrs = neigh.fit(parsed_data[['Profit Ratio']])
distances,indices = nbrs.kneighbors(parsed_data[['Profit Ratio']])
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(20,10))
plt.plot(distances[distances<0.1]) #zoom in on epsilon < 0.10
plt.title('K-distance Graph',fontsize=20)
plt.xlabel('Data Points sorted by distance',fontsize=14)
plt.ylabel('Epsilon',fontsize=14)
plt.show()
```




```python
plt.figure(figsize=(20,10))
plt.plot(distances[distances<0.010]) #zoom in on epsilon < 0.01
plt.title('K-distance Graph',fontsize=20)
plt.xlabel('Data Points sorted by distance',fontsize=14)
plt.ylabel('Epsilon',fontsize=14)
plt.show()
```

Finding the Epsilon Value

- Upon plotting the K-distance graph, there are 2 ranges that return the maximum gradient, namely $0 < \varepsilon < 0.01$ and $0.01 < \varepsilon < 0.08$
- However an epsilon value of 0 is not useful to our data analysis, hence the second range with an epsilon value $0.01 < \varepsilon < 0.08$ is used.

MACHINE LEARNING

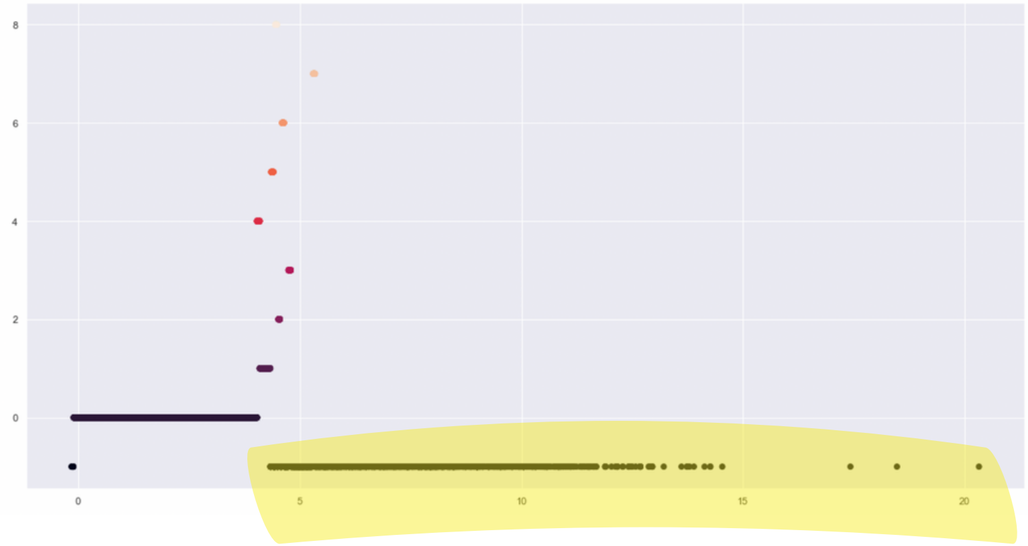
DBSCAN
Visualising the optimised model

MACHINE LEARNING

```
from sklearn.cluster import DBSCAN
optimised_dbscan=DBSCAN(eps=0.011,min_samples=46)
optimised_dbscan.fit(parsed_data[['Profit Ratio']])
#plot the bdscan result
plt.figure(figsize=(20,10))
plt.scatter(parsed_data['Profit Ratio'],optimised_dbscan.labels_,c=optimised_dbscan.labels_)
```

Outlier detection on 'Profit Ratio':

- After optimising the model, with an eps = 0.011 and min_samples = 46, it is able to identify more meaningful data points in parsed_data ['Profit Ratio'] which fall into the outliers category

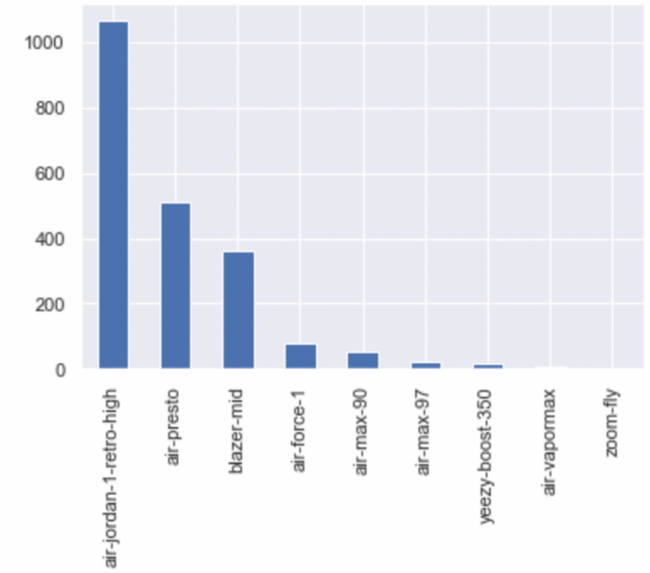


DBSCAN
Data Analysis on outliers

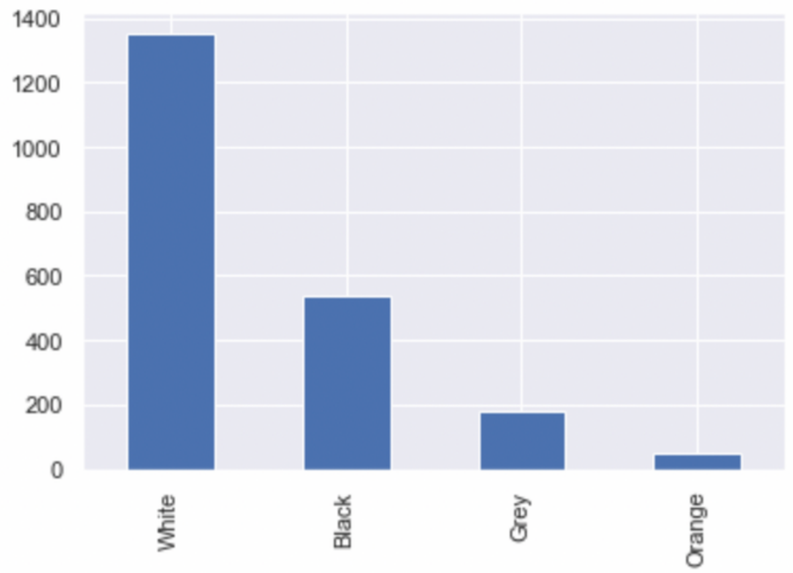
```
1 outliers = pd.DataFrame(parsed_data[optimised_dbscan.labels_ == -1])
2 outliers.to_csv('outliers.csv')
3 outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2113 entries, 127 to 72663
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Unnamed: 0      2113 non-null   int64
 1   Order Date      2113 non-null   object
 2   Brand           2113 non-null   object
 3   Sneaker Name    2113 non-null   object
 4   Release Date    2113 non-null   object
 5   Shoe Size       2113 non-null   float64
 6   Buyer Region    2113 non-null   object
 7   Turnover Days   2113 non-null   int64
 8   Profit          2113 non-null   float64
 9   Profit Ratio    2113 non-null   float64
 10  Model           2113 non-null   object
 11  Black           535 non-null    float64
 12  White           1835 non-null   float64
 13  Grey            182 non-null    float64
 14  Red             593 non-null    float64
 15  Green           0 non-null      float64
 16  Neo             5 non-null      float64
 17  Orange          48 non-null     float64
 18  Tan/Brown       52 non-null     float64
 19  Pink            193 non-null    float64
 20  Blue            221 non-null    float64
 21  Colorful        1061 non-null   float64
 22  Turnover Weeks  2113 non-null   float64
 23  color           2113 non-null   object
```

```
outliers.Model.value_counts().plot(kind='bar')
```




```
outliers.color.value_counts().plot(kind='bar')
```

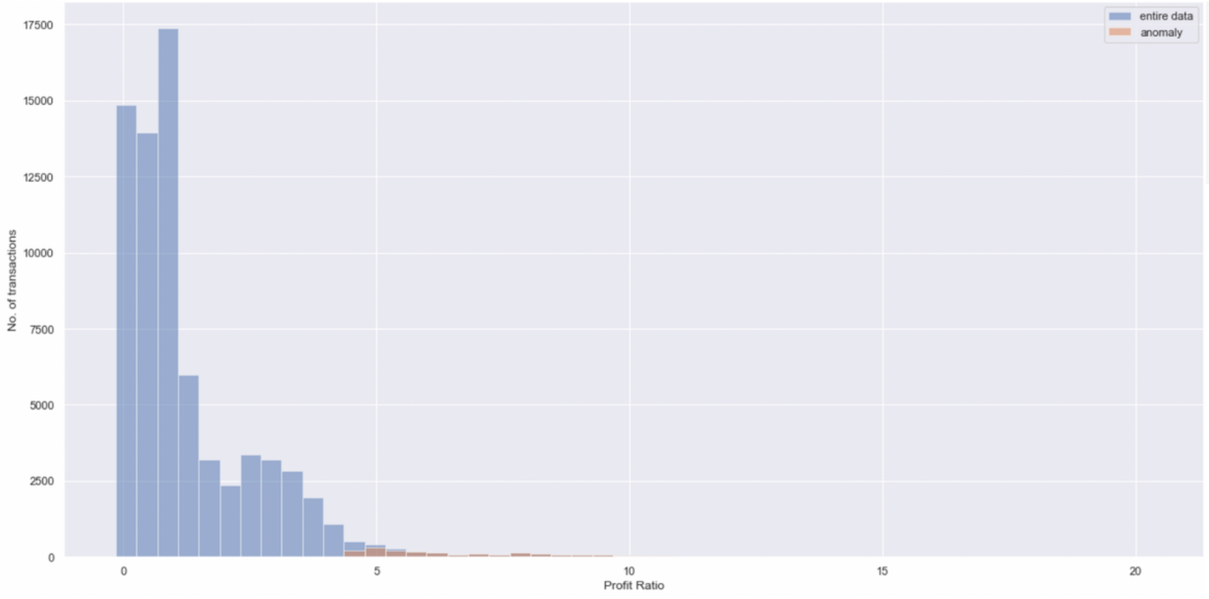


Upon consolidating the outliers into a separate DataFrame file

- We identified a total of 2113 anomalies data points in our data set, which are characterised by their unusually high profit ratio when compared to other data points.
- Overwhelming characteristics of these anomalies are 'White' and 'Air-Jordan-1-Retro-High'

MACHINE LEARNING

DBSCAN
Data Analysis on outliers




```python
plt.figure(figsize=(20,10))
plt.hist(parsed_data['Profit Ratio'], bins = 50, label = 'entire data', alpha = 0.5)
plt.hist(outliers['Profit Ratio'], bins = 50, label = 'anomaly', alpha = 0.5)
plt.legend(loc = "best")
plt.xlabel('Profit Ratio')
plt.ylabel('No. of transactions')
plt.show()
```

Plotting anomalies with the initial data

- The anomaly data points all lie in the right-most quadrant of the data distribution curve.
- This is in line with the logical assumptions that the low numbers of transactions with a high profit ratio is likely to be a rare (anomaly) case

```python
value = [1] * len(outliers)
outliers['$_$'] = value
outliers.head()
```

```python
parsed_data = pd.merge(parsed_data,outliers,how='left')
parsed_data.fillna(0,inplace=True)
parsed_data.to_csv('parsed_data.csv')
```

Merging the outlier data with the initial data

- In parsed_data, data points that are identified as outliers are denoted as '1' under the '$_$' column, where as non-outliers are identified as '0'.
- Parsed_data is now ready for predictive analysis on future anomaly transactions.

MACHINE LEARNING

MACHINE LEARNING

Predicting COLOURS of shoes most likely to sell for unusually high prices

SMOTE Oversampling & Edited Nearest Neighbours Undersampling

MACHINE LEARNING

Usage of a combination of oversampling & undersampling methods on TRAIN data:

1. SMOTE - selects a minority class instance randomly and finds k nearest minority neighbors & generating synthetic instances
2. Edited Nearest Neighbours - using k nearest neighbors to the data points that are misclassified and that are then removed before a K=1 classification rule is applied

```
from imblearn.combine import SMOTEENN
from imblearn.under_sampling import EditedNearestNeighbours
#from imblearn.under_sampling import TomekLinks

sm = SMOTEENN(random_state = 2,enn=EditedNearestNeighbours(sampling_strategy='majority'))
X_train_smote, y_train_smote = sm.fit_sample(X_train, y_train)
```

```
y_train.value_counts()
```

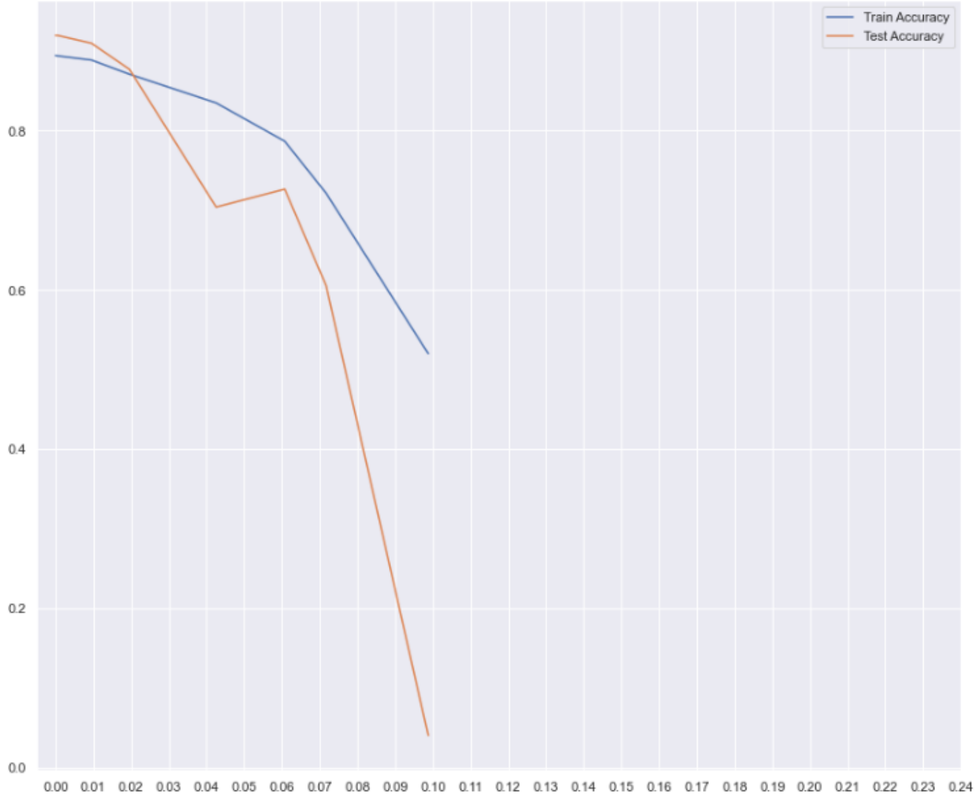
```
Outliers
0          56535
1           2283
dtype: int64
```

→

```
y_train_smote.value_counts()
```

```
Outliers
1          56535
0          52184
dtype: int64
```

Cost Complexity Pruning



Graph of Train and Test accuracies vs various alpha values

```
#Alpha value now set to approximately 0.06
dectree=DecisionTreeClassifier(ccp_alpha=0.062)
dectree.fit(X_train_smote,y_train_smote)
y_train_pred = dectree.predict(X_train_smote)
y_test_pred = dectree.predict(X_test)

# Check the Goodness of Fit (on Train Data)
print("Goodness of Fit of Model \tTrain Dataset")
print("Classification Accuracy \t:", dectree.score(X_train_smote, y_train_smote))
print()

# Check the Goodness of Fit (on Train Data)
print("Goodness of Fit of Model \tTest Dataset")
print("Classification Accuracy \t:", dectree.score(X_test, y_test))
print()
```

```
Goodness of Fit of Model        Train Dataset
Classification Accuracy         : 0.7869553619882449

Goodness of Fit of Model        Test Dataset
Classification Accuracy         : 0.7267596055763346
```

Usage of a cost complexity pruned decision tree to minimise fitting errors

Chosen alpha = 0.06 where accuracy differences are not too large

MACHINE LEARNING

K-Fold Cross Validation for model testing

Using K-Fold Cross Validation,

```
cv = StratifiedKFold(n_splits=10, random_state=1,shuffle=True)

scores = cross_val_score(dectree,predictors, upperoutlier, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Model Dataset Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Model Dataset Accuracy: 0.961 (0.000)
```

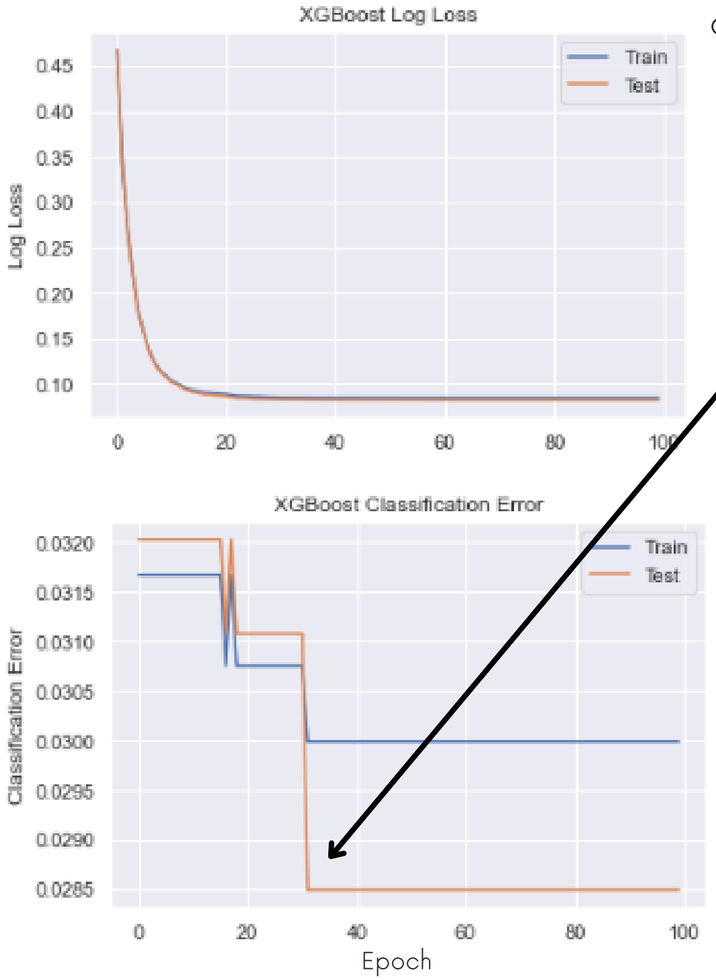
10- Folds of Stratified (for imbalanced classes in datasets) Cross Validation is used

Consistent mean score of 96.1% was achieved.

MACHINE LEARNING

XGBoost Classifier (Gradient Boosting)

MACHINE LEARNING



Initial fitting produced overfitting at approximately epoch = 30 as error seem to become constant

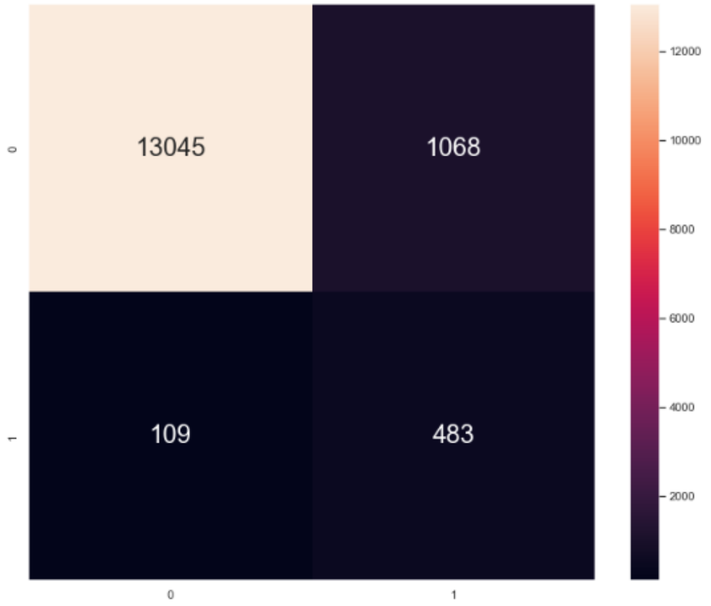
```
model = XGBClassifier(scale_pos_weight=24.35, early_stopping_rounds=10,max_depth=3)
model.fit(X_train, y_train,eval_metric=["error", "logloss"], eval_set=eval_set, verbose=True)

print(model)
```

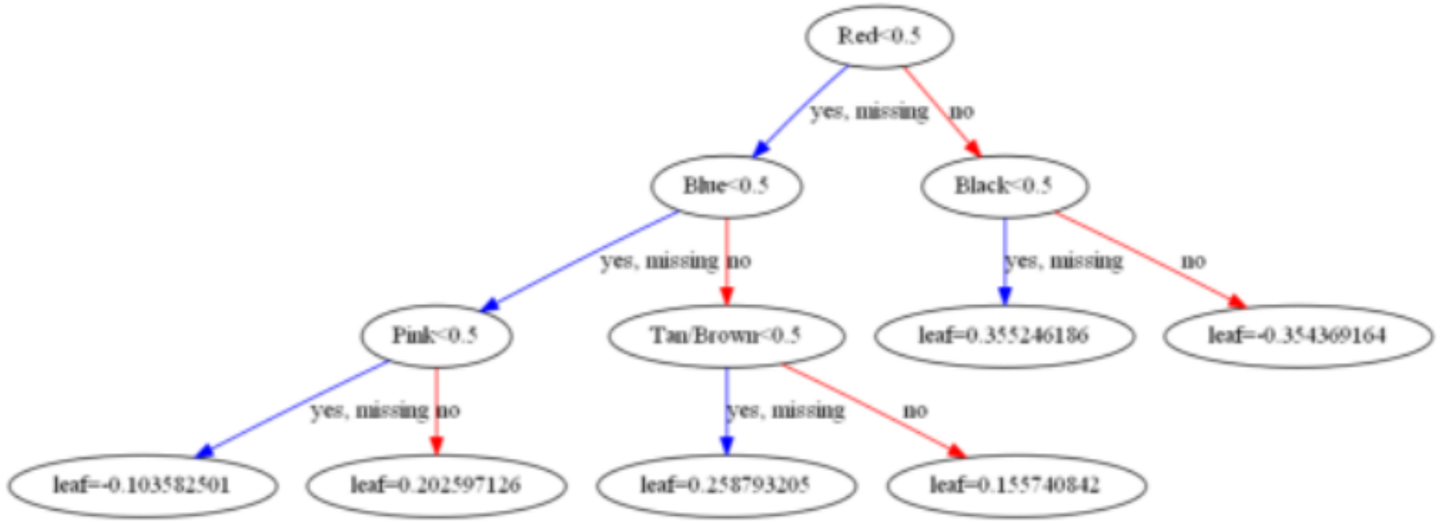
Improving on the model: Implemented weighted classification to account for imbalanced classes & early stopping after 10 epochs showing no improvement

```
]: from sklearn import metrics
   print(metrics.classification_report(y_test, y_pred_test))
```

	precision	recall	f1-score	support
0	0.99	0.92	0.96	14113
1	0.31	0.82	0.45	592
accuracy			0.92	14705
macro avg	0.65	0.87	0.70	14705
weighted avg	0.96	0.92	0.94	14705

XGBoost Classifier (Gradient Boosting)





High Accuracy on XGBoost is expected due to large dataset size as a gradient boosting ensemble model.

Tells us combinations that could give outliers: Shoes that are not red, blue, pink are better predicted to be outliers for example.

MACHINE LEARNING

BALANCED FOREST ON UNSAMPLED DATA

MACHINE LEARNING

Balanced Random Forest Classification:

```
from imblearn.ensemble import BalancedRandomForestClassifier
rf = BalancedRandomForestClassifier(n_estimators = 1000, random_state = 42, max_depth = 3)
rf.fit(X_train, y_train.values.ravel())
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)
```

```
# Check the Goodness of Fit (on Train Data)
print("Goodness of Fit of Model: Train Dataset")
print('Train Model accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_train_pred)))
print()

# Check the Goodness of Fit (on Test Data)
print("Goodness of Fit of Model: Test Dataset")
print('Test Model accuracy score:  {0:0.4f}'. format(accuracy_score(y_test, y_test_pred)))
print()
```

```
Goodness of Fit of Model: Train Dataset
Train Model accuracy score: 0.7129

Goodness of Fit of Model: Test Dataset
Test Model accuracy score:  0.7150
```

Using K-Fold Cross Validation,

```
cv = StratifiedKFold(n_splits=10, random_state=1,shuffle=True)

scores = cross_val_score(rf,predictors, upperoutlier, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Model Dataset Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Model Dataset Accuracy: 0.713 (0.003)
```

K-Folds mean score returns 71.3% accuracy which is not as desirable, possibly due to random sampling used in the model building.

RANDOM FOREST ON SAMPLED DATA

MACHINE LEARNING

Random Forest Classifier:

```
from sklearn.ensemble import RandomForestClassifier

#Use of balanced random forest, might not be necessarily as our data is oversampled
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42, max_depth = 3)
rf.fit(X_train_smote, y_train_smote.values.ravel())
y_train_pred = rf.predict(X_train_smote)
y_test_pred = rf.predict(X_test)
```

```
# Check the Goodness of Fit (on Train Data)
print("Goodness of Fit of Model: Train Dataset")
print('Train Model accuracy score: {0:0.4f}'. format(accuracy_score(y_train_smote, y_train_pred)))
print()

# Check the Goodness of Fit (on Test Data)
print("Goodness of Fit of Model: Test Dataset")
print('Test Model accuracy score:  {0:0.4f}'. format(accuracy_score(y_test, y_test_pred)))
print()
```

```
Goodness of Fit of Model: Train Dataset
Train Model accuracy score: 0.8751

Goodness of Fit of Model: Test Dataset
Test Model accuracy score:  0.7150
```

Using K-Fold Cross Validation,

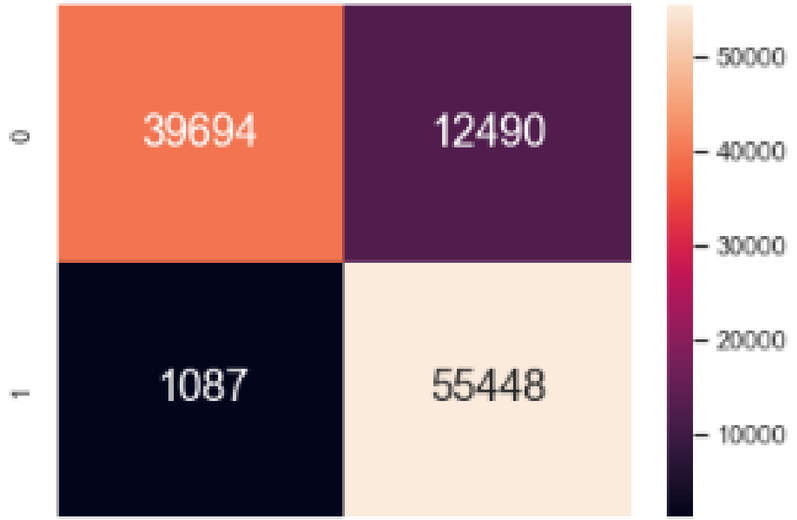
```
cv = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)

scores = cross_val_score(rf,predictors, upperoutlier, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Model Dataset Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Model Dataset Accuracy: 0.968 (0.001)
```

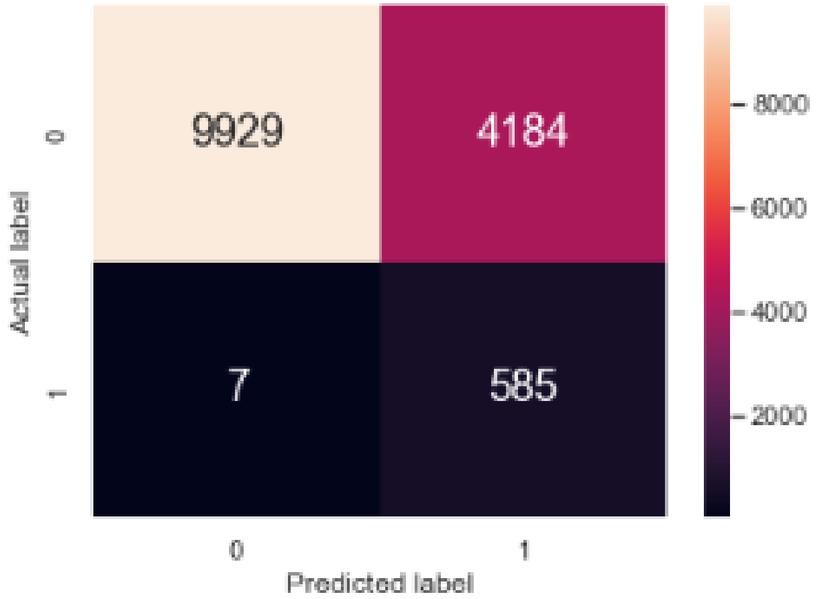
K-Folds mean score returns 96.8% using our sampled data which is a huge improvement from the previous. Test Accuracy of 71.5% is good for our model as outliers are difficult to predict.

EVALUATION

Since our models are tested on both the test data (untouched after splitting) & the whole dataset as a whole using K-Folds Cross Validation, we decided on the Random Forest Classifier to classify our data.



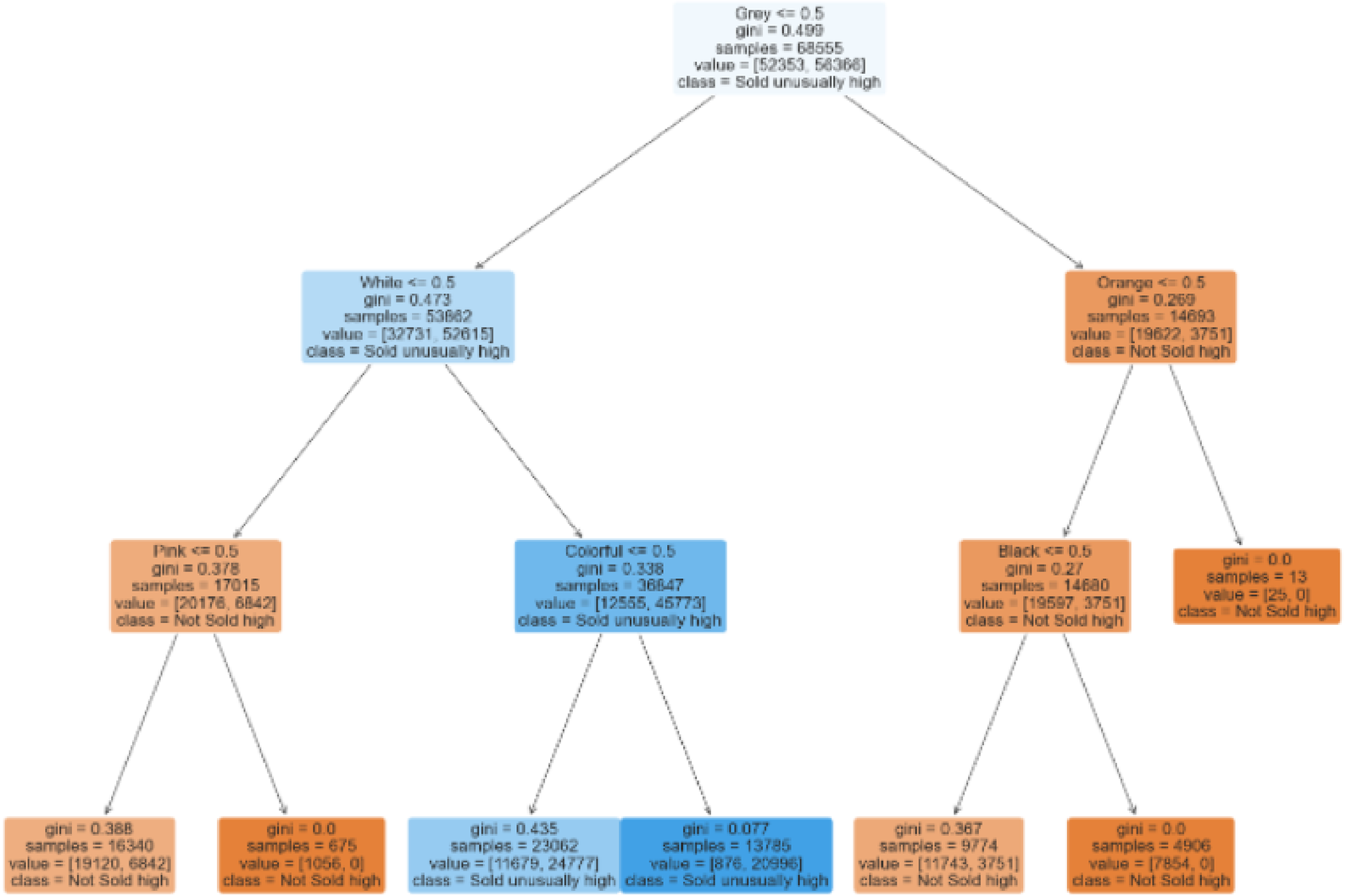
Train Data Confusion Matrix



Test Data Confusion Matrix

MACHINE LEARNING

DECISION TREE



1) Shoes not being grey is given as the most important predictor for unusually high price shoes

2) Shoes being white is the next most important predictor, followed by colourful shoes

MACHINE LEARNING

Recursive Feature Elimination (RFE)

Final Analysis using Recursive Feature Elimination (RFE)

```
from sklearn.feature_selection import RFE

rfe = RFE(estimator=RandomForestClassifier(n_estimators = 1000, random_state = 42, max_depth = 3), n_features_to_select=1)
rfe = rfe.fit(X_train_smote, y_train_smote.values.ravel())
print(rfe.support_)
print(rfe.ranking_)
```

```
[False  True False False False False False False False False False]
[ 4  1  2  5 11  9  7 10  8  6  3]
```

```
print(X_train.columns)
```

```
Index(['Black', 'White', 'Grey', 'Red', 'Green', 'Neo', 'Orange', 'Tan/Brown',
       'Pink', 'Blue', 'Colorful'],
      dtype='object')
```

Cross Comparing using RFE gives us the importance of each colour in predictions.

White, Grey and Colourful shoes are ranked as the 3 most important predictors, consistent with our random forest classifier.

MACHINE LEARNING

MACHINE LEARNING

Predicting SIZES of shoes most likely to sell for unusally high prices

SHOE SIZE PREDICTION

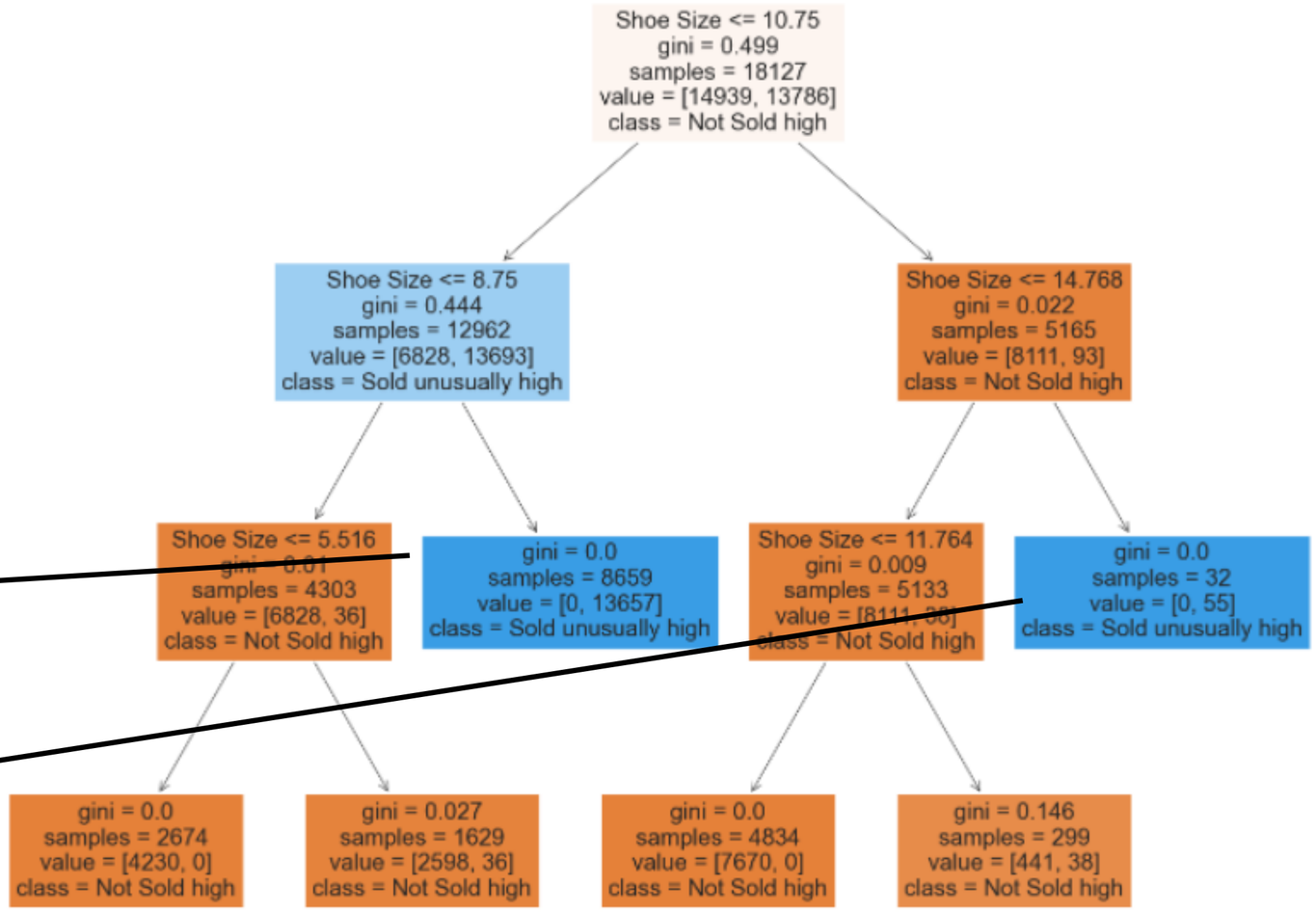
MACHINE LEARNING

Usage of similar SMOTEEN sampling, pruning & random forest techniques give us 96.1% accuracy on random forest modelling.

```
cv = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)

scores = cross_val_score(rf,predictors, upperoutlier, scoring='accuracy', cv=cv, n_
# report performance
print('Model Dataset Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Model Dataset Accuracy: 0.961 (0.000)
```

Shoe Sizes >8.75 but<= 10.75
or
Shoe Sizes > 14.768
are presented as predictions for outliers



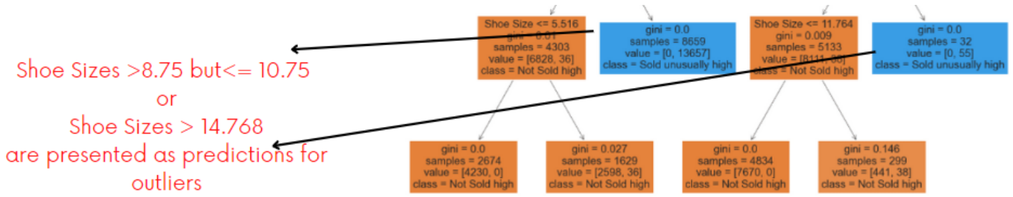
INSIGHTS & RECOMMENDATIONS

SO WHAT?

WE HAVE IDENTIFIED THESE COLOURS & SIZES MOST PREDICTABLE TO SELL UNUSUALLY HIGH

1) Shoes not being grey is given as the most important predictor for unusually high price shoes

2) Shoes being white is the next most important predictor, followed by colourful shoes



- Dataset consists of sneakers released from 2016 to 2019 (4 years) - unlikely to be periodic trends for these colours & sizes
- USAGE ON FUTURE NEW MODEL RELEASES to purchase for reselling to give better ideas on which ones to buy

TRENDS IN RESULTS

- SHOE SIZES? Sneaker sizes that are most likely to rise to abnormal prices for sale are that of >8.75 but<= 10.75, in line with the most popular sneaker sizes demanded at the 9.5-10.5 range.
    - Large sneaker sizes such as those >=15 too are predicted to be highly profitable.
- COLOURS? White & Colourful shoes seem to be the most popular colours as they are predicted to be able to sell for most profit usually compared to other colours, showing the fashion trend in this decade.
    - On the other hand, grey shoes are not popular seemingly from our model predictions

IS IT REALLY THAT PROFITABLE?

- What if turnover periods are abnormally high for a shoe predicted to rise up for unusually high profits?
  - Sneaker are generally appreciating asset in the long run, as proven in our EDA. After picking the pair based on our predictive analysis, it is important to be patient and 'diamond hand' until the optimal selling price has reached.
- How about certain models being popular only for a certain period, aka a mere 'hype'?
  - We used colours as a predictor instead of models, to reduce the influence of 'hype' over a specific model on our predictive analysis
  - We could look into convolutional neural network to improve our model by drawing similarities in design elements across different models that are sold for high prices

THANK YOU!
THANK YOU!