**Brief Description of Project:**

**Prediction of Returns**

**By:**

**Naman Ratra**

**Aim:**

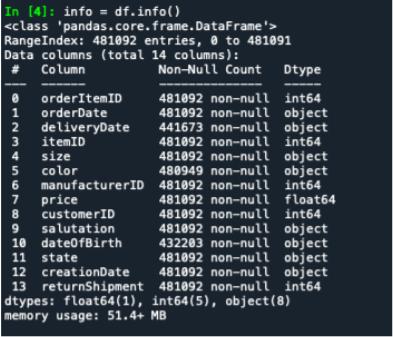
To predict If a person would return an ordered item or not, from an ecommerce website.

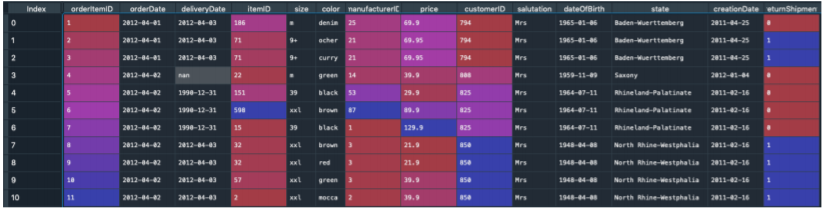
**Description of project:**

**Elementary data analysis of the data**:

Data Description:

Data had total of 14 features: orderItemID orderDate deliveryDate itemID size color manufacturerID price customerID salutation dateOfBirth state creationDate returnShipment





Total of 480,000 rows of data.

Total missing values: 39,419 missing values in DeliveryDate, 48,889 missing values in DateofBirth and 143 missing values in Color.

Particularly balanced data in terms of return of shipments of around 48%. Hence, ended I up using accuracy as a matric for the performance of the ML models.



**Challenges in the project:**

* Size of data
* a high number of unique values in some columns
* high number of missing data and outliers
* multiple data types.

**Data Preparation:**

* As a first step to make more use of current data I transformed deliveryDate and OrderDate into the column DeliveryTime. Which is just the difference between transform deliveryDate and OrderDate.
* Second step is to transform DateofBirth column into an age column, which displays as a float data type how old the customers were in the year they have ordered (orderDate).
* The third necessary transformation takes place in the Size column in which some sizes are displayed in different writing styles, e.g., ‘XXL’ and ‘xxl’. To reduce complexity of the unique values I align these sizes considering that they have the same meaning.

**Handling missing values:**

* After creating new variables, I needed to replace the missing values in these new variables, so there were DeliveryTime and age had 39,419 and 48,889 missing values respectively which is same number of missing variables as deliveryDate and DateofBirth because these new variables were created from those variables only.
* To handle missing variables in DeliveryTime, I imputed values of median DeliveryTime in a particular state as the missing value.
* To handle missing variables in age, I imputed values of median age in a particular state as the missing value.

**Outliers:**

* Dealing with outliers in DeliveryTime and age:

For age dropped values of age below 18 and above 100

For DeliveryTime dropped negative delivery times and delivery time greater than 60 days.

* Replacing nan values in 'color' feature to ‘No Color’

**Feature engineering:**

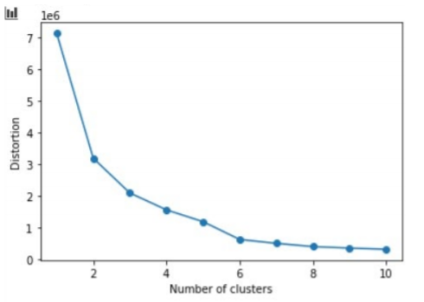
Created new variables using current variables:

* ordertotalcount: This can be seen as the cart size, so the value shows how many items have been ordered within the order that the item belongs to. The development of this variable is based on the assumption that customers probably order a variety of items within a single order but have the intention to keep only some of the items and return the rest.
* orderItemcount: The ‘orderItemcount’ represents the quantity of the specific item within its order regardless of its color and size. For Example, if there was the same shoe ordered in three different sizes and each size in three different colors within a single order, the orderItemcount would be 9. The idea behind this feature is that customers may order a specific item in various different colors and sizes because they do not know which fits them best or which color they like most.
* orderItemsizecount: The values of this feature indicate how many different sizes an item has been ordered within one order. As an example, if a customer ordered the same shoe in three different sizes and each size in three different colors within one order, the orderItemsizecount would be 3. This feature derives from a similar idea about customer behavior as the previous one, but here we are more specific. Maybe a customer orders an item in two sizes because he is unsure about which size, he/she needs. So, there is an upfront intention to only keep one item.
* Day of the month when the item was ordered ('orderDay' and 'ordermonthday'): Maybe at the beginning of the month, when people just received their salary, their tendency to return items is lower, because they are less likely to be short on cash.
* Orderhistory: Maybe there is some sort of learning process going on within customers, so that when they have previously ordered items, they know better what fits them. Or maybe if it's not their first order, they are satisfied with the products of the company and know what to expect.
* orderItemcolorcount: Maybe people order the same item in different colors with the intention to keep the one that they like most. Or maybe they order an item in different colors, because they really like that item and want it in multiple colors?
* orderItemsizecount: Maybe people order the same item in different sizes with the intention to keep the one that fits best.
* Frequency of Colors and Sizes in the data: Created colortype and sizetype based on the frequency (High, Moderate, Low) at which a particular size or color occur in a data.

Kmeans++

I used Unsupervised learning technique to extract more information from the data I had. So, I decided to create a feature cluster using the clustering technique Kmeans++

I came up with 6 clusters of the current data set. Using the elbow plot.

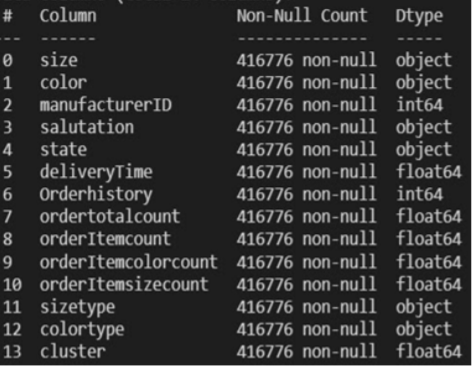


**Dropping of Unnecessary variables:**

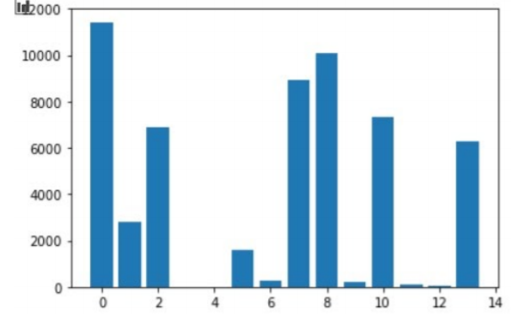
After all this data manipulation and creation of all these new columns I needed to find a way to reduce the size of data as the data from 14 features in beginning had gone to 26 features in total.

Of these features multiple were categorical, So to handle this I did a chi squared test on the **categorical data**:

Data in chi squared test:



Results of chi squared test:



Features that survive chi squared test: size, color, manufactureID, orderItemcount, ordertotalcount, orderItemsizecount and cluster.

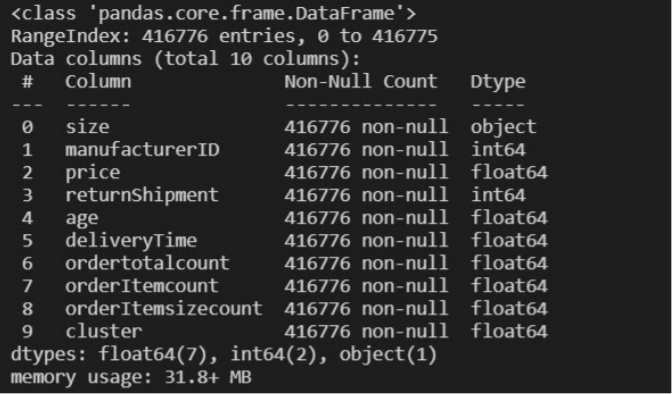
In addition to that dropped multiple redundant variables because of dummy variable trap.

Final feature dropping code:

data = data.drop(['orderItemID','itemID','customerID','orderDay','orderItemID','ordermonthday'], axis = 1)

data = data.drop(['state','salutation','Orderhistory','orderItemcolorcount','sizetype','colortype'], axis = 1)

**Final data set:**



**Encoding and PCA:**

Used label binarizer for categorical data, Ordinal encoder for ordinal data, and max min scalar for continuous data

Performed PCA:

n components = 385. To n components = 30

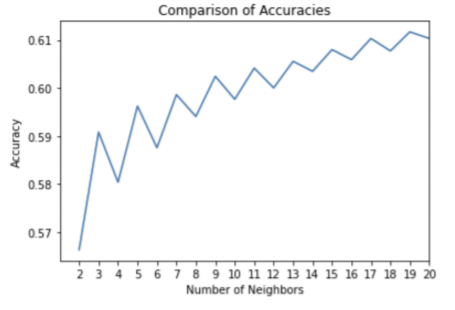
**Data Split:**

Train: 75%

Test: 25%

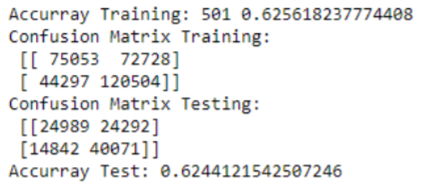
**Results of ML models:**

KNN: Ran KNN from 2 neighbours to 20

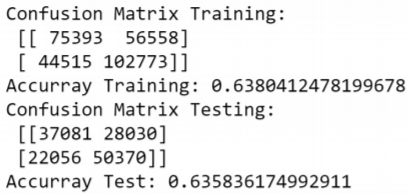


Its quite obvious that the accuracy falls for even numvers because of the even number trap for KNN

Best accuracy in neighbours = 19

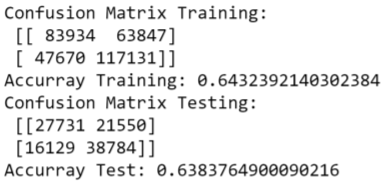


Logistic Regression:



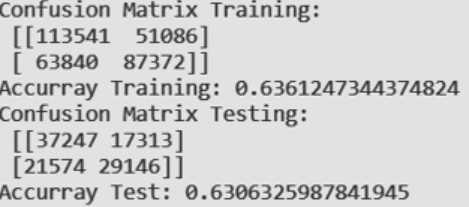
Neural Networks:

I got the best results at solver='lbfgs' and hidden layers = 8



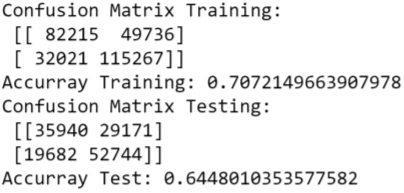
Decision Tree:

Best model after grid search, criterion='entropy',random\_state=0,max\_depth=9



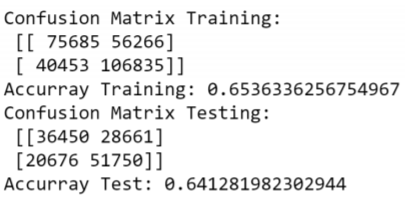
Random Forest:

Random forest after the grid search the best accuracy was.



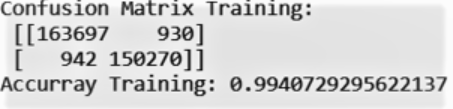
Gradient Boost Trees:

gbmodel = GradientBoostingClassifier(random\_state=0,max\_depth=12, learning\_rate=0.13)

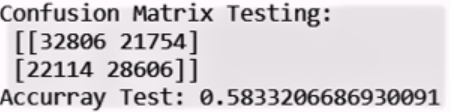


ADA Boost:

AdaBoostClassifier(DecisionTreeClassifier(max\_depth=15), n\_estimators=180, algorithm="SAMME.R", learning\_rate=0.5)



Overfitting in this classifier



**Cross Validation:**

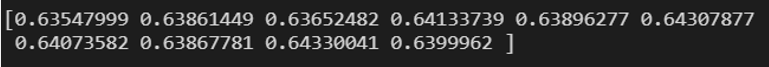
Used Random forest classifier for it

rfmodel = RandomForestClassifier(random\_state=0, max\_depth = 15, n\_estimators= 180)

accuracies = cross\_val\_score(rfmodel, X\_train\_pca, y\_train,

scoring='accuracy', cv = 10)

Outputs





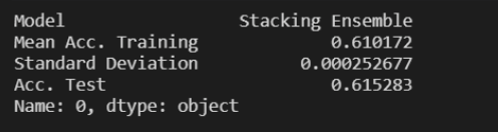
**Ensemble Stacking:**

Classifiers: KNN and Random forest

Meta Classifier: Logistic Regression.

stens1model = StackingClassifier(classifiers=[knnmodel,rfmodel], use\_probas=True,average\_probas=False, meta\_classifier=lr\_ensemble)

Result



**Conclusion:**

Overall, I can conclude that random forest seems to be the best classifier among all of the classifiers mentioned.