# CE1115 Mini Project

Done by :

Dhwani Patel
Kshitij Parashar
Chan Ray

# Data set chosen:

Brazilian E-Commerce Public Dataset by Olist From Kaggle

# Problem Statement :

## Predicting review score based on available variables

It is crucial for an e-commerce company to be able to predict whether a customer liked or disliked a product. This is so that the company can recommend more similar and related products as well as decide whether or not a product should be sold.

```python
cust_data = pd.read_csv('../input/brazilian-ecommerce/olist_customers_dataset.csv')
geo_data =  pd.read_csv('../input/brazilian-ecommerce/olist_geolocation_dataset.csv')
order_items =  pd.read_csv('../input/brazilian-ecommerce/olist_order_items_dataset.csv')
order_payments =  pd.read_csv('../input/brazilian-ecommerce/olist_order_payments_dataset.csv')
order_reviews =  pd.read_csv('../input/brazilian-ecommerce/olist_order_reviews_dataset.csv')
order_data =  pd.read_csv('../input/brazilian-ecommerce/olist_orders_dataset.csv')
products_data = pd.read_csv('../input/brazilian-ecommerce/olist_products_dataset.csv')
sellers_data = pd.read_csv('../input/brazilian-ecommerce/olist_sellers_dataset.csv')
product_category = pd.read_csv('../input/brazilian-ecommerce/product_category_name_translation.csv')
```

```python
#renaming all the zip_code_prefix so as to make the name common in all tables inorder to perform join

geo_data.rename(columns={'geolocation_zip_code_prefix':'zip_code_prefix'},inplace=True)
cust_data.rename(columns={'customer_zip_code_prefix':'zip_code_prefix'},inplace=True)
sellers_data.rename(columns={'seller_zip_code_prefix':'zip_code_prefix'},inplace=True)
```

```python
#keeping only the unique zip code prefix so that it can act as key to join tables
geo_data.drop_duplicates(subset='zip_code_prefix',inplace=True)
geo_data.shape
```

```
(19015, 5)
```

+ Code  + Markdown

```python
order_items.head()
```

| | order_id | order_item_id | product_id | seller_id | shipping_limit_date | price | freight_value |
|---|---|---|---|---|---|---|---|
| 0 | 00010242fe8c5a6d1ba2dd792cb16214 | 1 | 4244733e06e7ecb4970a6e2683c13e61 | 48436dade18ac8b2bce089ec2a041202 | 2017-09-19 09:45:35 | 58.90 | 13.29 |
| 1 | 00018f77f2f0320c557190d7a144bdd3 | 1 | e5f2d52b802189ee658865ca93d83a8f | dd7ddc04e1b6c2c614352b383efe2d36 | 2017-05-03 11:05:13 | 239.90 | 19.93 |
| 2 | 000229ec398224ef6ca0657da4fc703e | 1 | c777355d18b72b67abbeef9df44fd0fd | 5b51032eddd242adc84c38acab88f23d | 2018-01-18 14:48:30 | 199.00 | 17.87 |
| 3 | 00024acbcdf0a6daa1e931b038114c75 | 1 | 7634da152a4610f1595efa32f14722fc | 9d7a1d34a5052409006425275ba1c2b4 | 2018-08-15 10:10:18 | 12.99 | 12.79 |
| 4 | 00042b26cf59d7ce69dfabb4e55b4fd9 | 1 | ac6c3623068f30de03045865e4e10089 | df560393f3a51e74553ab94004ba5c87 | 2017-02-13 13:57:51 | 199.90 | 18.14 |

- We import all csv files
- Merging all the data from different csv files using the same columns

```python
#merging all customer related data

A = pd.merge(order_data,order_reviews,on='order_id')
A = pd.merge(A,order_payments,on='order_id')
A = pd.merge(A,cust_data,on='customer_id')
#peforming left outer join as we need every geo based address related to customer
A = pd.merge(A,geo_data,how='left',on='zip_code_prefix')
A.shape
```

```
(104485, 26)
```

+ Code + Markdown

```python
#merging all seller related data

B = pd.merge(order_items,products_data,on='product_id')
B = pd.merge(B,sellers_data,on='seller_id')
B = pd.merge(B,product_category,on='product_category_name')
#peforming left outer join as we need every geo based address related to seller
B = pd.merge(B,geo_data,how='left',on='zip_code_prefix')
B.shape
```

```
(111023, 23)
```

```python
#merging customer based data to the seller based data

data = pd.merge(A,B,on='order_id')
data.shape
```

```
(116581, 48)
```

- Merging customer and seller related data
- Having the final Data column as shown below

```python
#final data columns
data.columns
```

```
Index(['order_id', 'customer_id', 'order_status', 'order_purchase_timestamp',
       'order_approved_at', 'order_delivered_carrier_date',
       'order_delivered_customer_date', 'order_estimated_delivery_date',
       'review_id', 'review_score', 'review_comment_title',
       'review_comment_message', 'review_creation_date',
       'review_answer_timestamp', 'payment_sequential', 'payment_type',
       'payment_installments', 'payment_value', 'customer_unique_id',
       'zip_code_prefix_x', 'customer_city', 'customer_state',
       'geolocation_lat_x', 'geolocation_lng_x', 'geolocation_city_x',
       'geolocation_state_x', 'order_item_id', 'product_id', 'seller_id',
       'shipping_limit_date', 'price', 'freight_value',
       'product_category_name', 'product_name_lenght',
       'product_description_lenght', 'product_photos_qty', 'product_weight_g',
       'product_length_cm', 'product_height_cm', 'product_width_cm',
       'zip_code_prefix_y', 'seller_city', 'seller_state',
       'product_category_name_english', 'geolocation_lat_y',
       'geolocation_lng_y', 'geolocation_city_y', 'geolocation_state_y'],
      dtype='object')
```

# DATA CLEANING

- Removing useless features (cleaning up excessive data point that we deem not necessary)

```
#removing some mis filled data
data = data[data['geolocation_state_y'] == data['seller_state']]

#list of useless feature
useless_features = ['review_comment_title','review_comment_message','product_category_name','product_weight_g','review_creation_date',
                     'product_length_cm','product_height_cm','product_width_cm','seller_city','review_answer_timestamp',
                    'geolocation_lat_y','geolocation_lng_y','geolocation_city_y','geolocation_state_y','review_id','order_approved_at','order_status',
                    'order_id','customer_id','order_item_id','geolocation_lat_x',
                    'geolocation_lng_x','geolocation_city_x','geolocation_state_x']
print('Number of useless features as of now are : ',len(useless_features))

data.drop(useless_features,axis=1,inplace=True)

data.rename(columns = {'product_category_name_english':'product_category_name','zip_code_prefix_x':'zipCode_prefix_cust',
                       'zip_code_prefix_y':'zipCode_prefix_seller'},inplace=True)
```

```
Number of useless features as of now are :  24
```

# Why did we remove NULL rows instead of replacing NULL rows with the mean value?

# There is ENOUGH data!!

Before removing: 116,581 rows

After removing : 113,152 rows

With only 2.16% of the data removed, we still have vast amounts of data to work with

```
print('Only {}% of data got removed'.format(round(((prev_size - current_size)/prev_size)*100,2)))

(113152, 24)
Only 2.16% of data got removed
```

# Our Approach

## Regression

- ➔ Linear regression
- ➔ Random Forest Regression

## Classification

- ➔ Logistic Classification
- ➔ Random Forest Classification

# APPROACH 1 : LINEAR REGRESSION

# The Variables

- ➔ **Product Description Length**

  **- If** the description of the product is detailed and long , customer is most likely to know every crux of the product thus increasing the high rating probability.

- ➔ **Product Photo Quantity**

  **-**If it has many photos customer customer is well aware of how the product looks from all possible ways and is more certainly sure about going for it , thus increasing the chance of getting a positive rating.

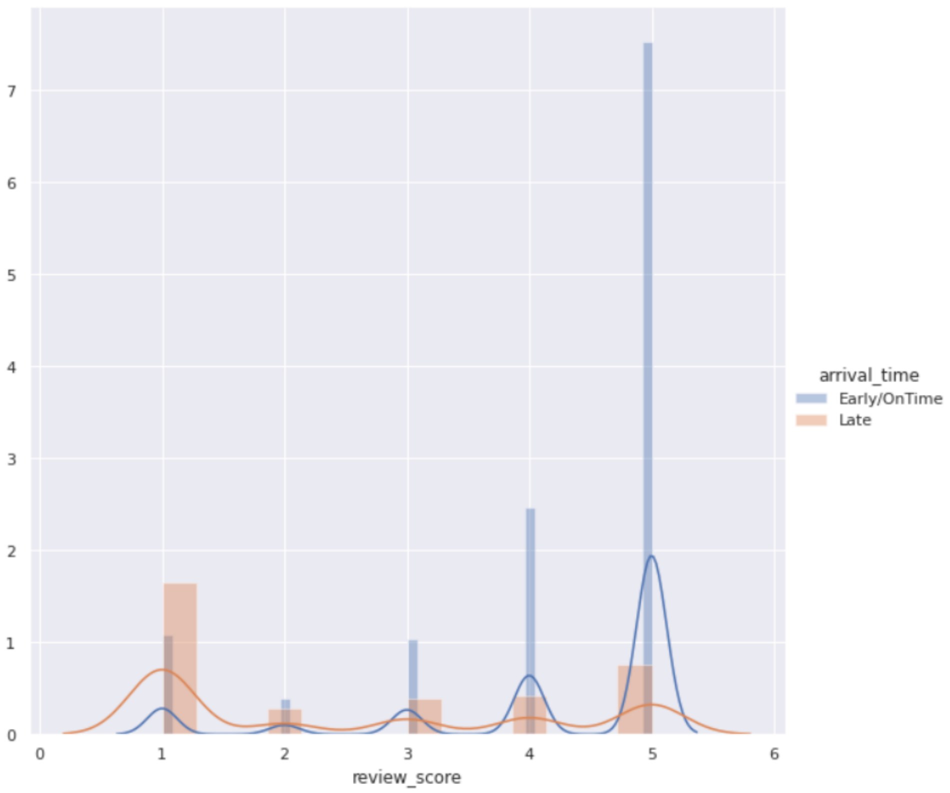
- ➔ **Delivery Days**

  **-** With the emphasis on instant gratification we would expect faster delivery times to correlate to a better experience

# Analysis of relation between review score and arrival time

**Observations**

1. Clearly from the plot the customers are more likely to give an 4-5 rating if the product either arrives early or arrive on time.
2. Hence, delivery time impacts a lot to the customer rating and is a variable we've considered



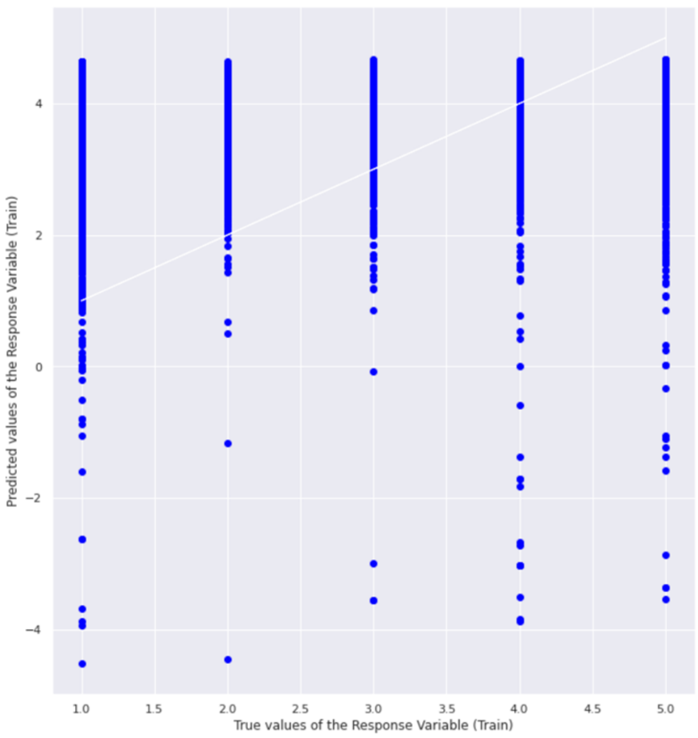
# Linear Regression Analysis

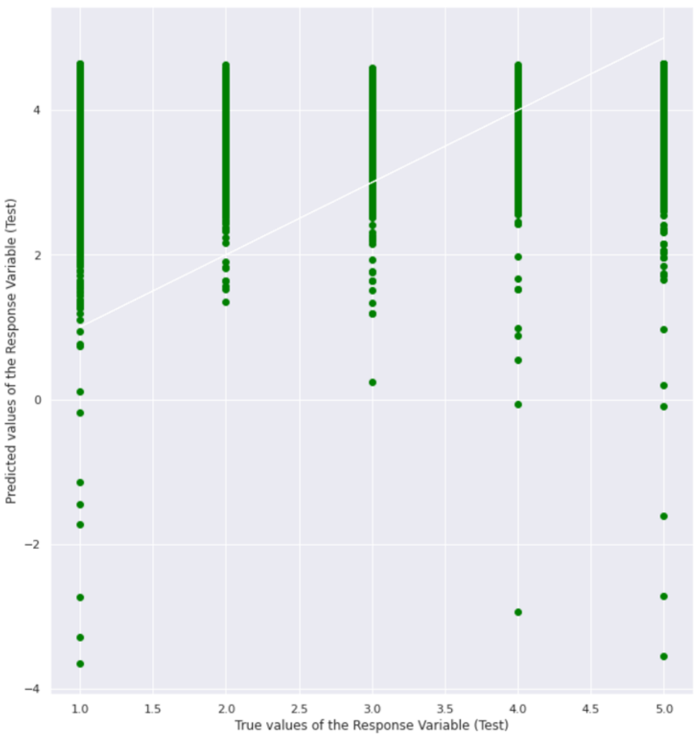
```
Intercept of Regression        : b =  [4.56854311]
Coefficients of Regression     : a =  [[ 3.04186076e-05  7.17395134e-03 -4.34934188e-02]]

                    Predictors  Coefficients
0  product_description_lenght      0.000030
1          product_photos_qty      0.007174
2               delivery_days     -0.043493
```

```
Goodness of Fit of Model         Train Dataset
Explained Variance (R^2)         : 0.09225683216334288
Mean Squared Error (MSE)         : 1.6766275418544327
Root Mean Squared Error (RMSE)   : 1.2948465321629559

Goodness of Fit of Model         Test Dataset
Explained Variance (R^2)         : 0.09541800507659837
Mean Squared Error (MSE)         : 1.66594212258128
Root Mean Squared Error (RMSE)   : 1.2948465321629559
```



# Why was Linear Regression analysis not optimal?

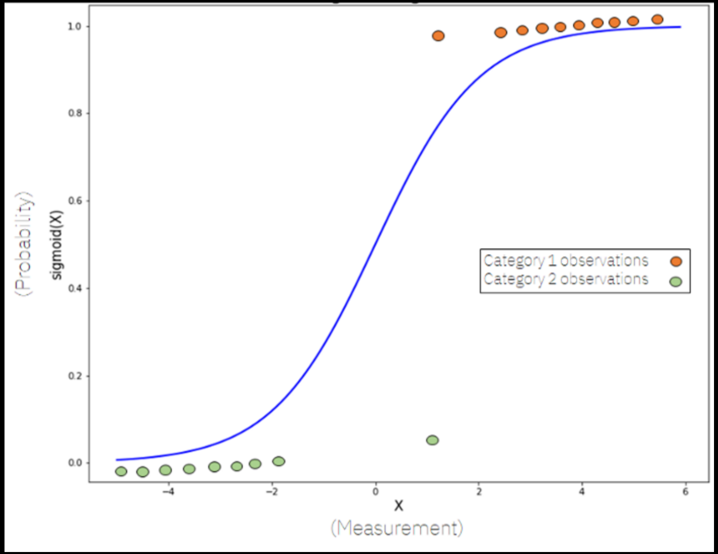
The Data type for Review is categorical, i.e, numbers in the set {1, 2, 3, 4, 5}.

But, the predicted values are real numbers instead of categorical values which leads to a drop in the $R^2$ value.

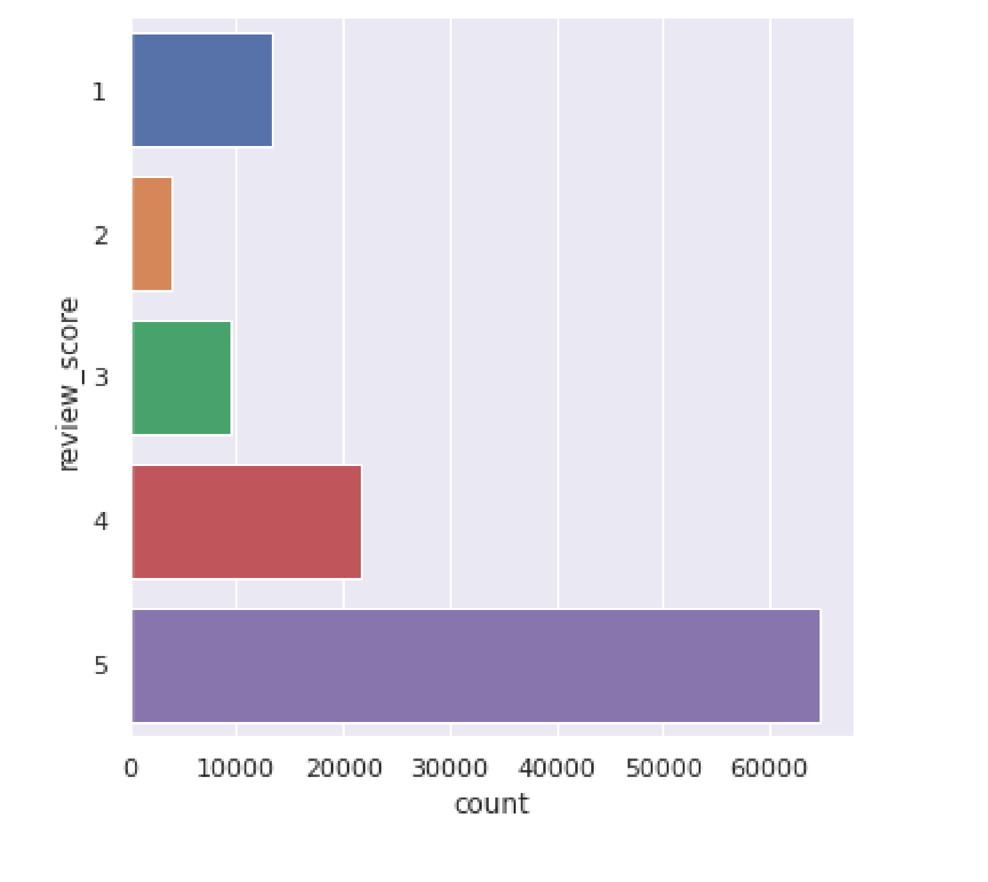
# The new things we tried

# APPROACH 2 : LOGISTIC REGRESSION

# What is Logistic Regression?




- A machine learning algorithm for classification problems
- It is a parametric classification model
- Fitting a Sigmoid Curve
- Sigmoid vs Softmax
- Different solvers: lbfgs, newton-cg, sag, saga



1
2
3
4
5
review_score
0
10000
20000
30000
40000
50000
60000
count

```python
# Training multi-classification model with logistic regression
lbfgs = linear_model.LogisticRegression(multi_class='multinomial', solver='lbfgs')
lbfgs.fit(x_train, y_train)

#Predicting values corresponding to the columns
lbfgs_train_pred = lbfgs.predict(x_train)
lbfgs_test_pred = lbfgs.predict(x_test)

#Calculating Accuracies
print ("Logistic regression Train Accuracy ::", metrics.accuracy_score(y_train, lbfgs.predict(x_train)))
print ("Logistic regression Test Accuracy :: ", metrics.accuracy_score(y_test, lbfgs.predict(x_test)))
```

```
Logistic regression Train Accuracy :: 0.572330379011691
Logistic regression Test Accuracy ::  0.5722029105049196
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

# Example: lbgs solver

# APPROACH 3 : RANDOM FOREST REGRESSION

# Random Forest Regression

The $R^2$ value of the Train dataset is observed to be more than that of the Test dataset. This means that the model is **Overfitting**.

```
The Explained Variance R^2 on Train set is: 0.8270095937238293
The Explained Variance R^2 on Test set is: 0.1494036810651O127
Mean Squared Error: 1.5951938373858188
```

Overfitting can be prevented in Random Forest by:

Using **Cross-Validation** which involves selecting the best_estimator_ based on the scoring parameter of accuracy. Then, using **Hyper-Parameter Tuning** that governs the number of features that are randomly chosen to grow each tree from the bootstrapped data.

Hyper-Parameter tuning works by running multiple trials in a single training job. Each trial is a complete execution of a training application with values for our chosen hyper-parameters, set within the limits which we specify.

# APPROACH 4 : RANDOM FOREST CLASSIFICATION

# Random Forest Classification

Observing the $R^2$ values obtained on the Train and Test data set, we conclude that this model provides quite optimal results.

Furthermore, this model has a high accuracy of 0.9374.

Again, the accuracy and Explained Variance have further scope of improvement using methods like:

- Feature Engineering
- Hyper-Parameter Tuning
- Treating Outliers (eg. Transformation)

```
r2_train=explained_variance_score(y_train,y_clf_train_pred)
r2_test=explained_variance_score(y_test,y_clf_test_pred)

print("The Explained Variance R^2 on Train set is:",r2_train)
print("The Explained Variance R^2 on Test set is:",r2_test)
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_clf_test_pred))
```

```
The Explained Variance R^2 on Train set is: 0.8350984495130678
The Explained Variance R^2 on Test set is: 0.8384882138449854
Mean Squared Error: 0.3064573145584163
```

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

```
Accuracy: 0.9374005773876156
```

# Comparing the different types of analysis we have done.

## Linear Regression

Not Optimal as review_score has categorical numbers as its values.

## Logistic Regression

This model has the accuracy of **0.5916**
This is not optimal as it would mean the model is only correct **59%** of the time

## Random Forest Regression

The $R^2$ value of the Train dataset is observed to be more than that of the Test dataset. This means that the model is **Overfitting**.

## Random Forest Classification

This model has the highest accuracy of **0.9374**.

# Allocation of work

Chan Ray - Data Cleaning, Linear Regression

Dhwani Patel - Logistic Regression, video editing

Kshitij Parashar - Random Forest Regression and Random Forest Classification

# Sources

- Walk-through Notebook on NTU Learn
- https://www.datacamp.com/community/tutorials/random-forests-classifier-python
- https://towardsdatascience.com/logistic-regression-explained-9ee73cede081
- https://www.jeremyjordan.me/hyperparameter-tuning/
- https://machinelearningmastery.com/calculate-feature-importance-with-python/
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://towardsdatascience.com/classification-with-random-forests-in-python-29b8381680ed

# THANK YOU!