Problem Statement:
The aim of the assignment is to build a model that predicts whether a person purchases an item after it has been added to the cart or not. Being a classification problem, you are expected to use your understanding of all the three models covered till now. You must select the most robust model and provide a solution that predicts the churn in the most suitable manner.

For this assignment, you are provided the data associated with an e-commerce company for the month of October 2019. Your task is to first analyze the data, and then perform multiple steps towards the model building process.

As part of the assignment, you are expected to complete the following tasks:

Task 1 - Data Exploration
Task 2 - Feature Engineering
Task 3 - Model Selection
Task 4 - Model Inference

Dataset Reference:
Note that the dataset stores the activity on the e-commerce platform for the month of October 2019. It is stored in the following path:

https://mlc-c5-assignment.s3.amazonaws.com/2019-Oct.csv
S3 Bucket: mlc-c5-assignment
Filename: 2019-Oct.csv

Data description:
The dataset stores the information of a customer session on the e-commerce platform. It records the activity and the associated parameters with it.

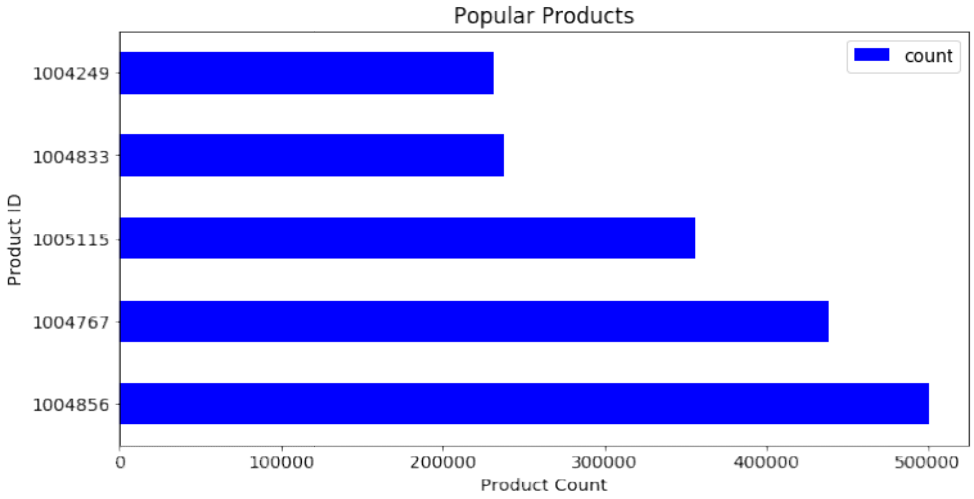
event_time: Date and time when user accesses the platform
event_type: Action performed by the customer
- View
- Cart
- Purchase
- Remove from cart

product_id: Unique number to identify the product in the event
category_id: Unique number to identify the category of the product
category_code: Stores primary and secondary categories of the product
brand: Brand associated with the product
price: Price of the product
user_id: Unique ID for a customer
user_session: Session ID for a user

Task 1 - Data Exploration:

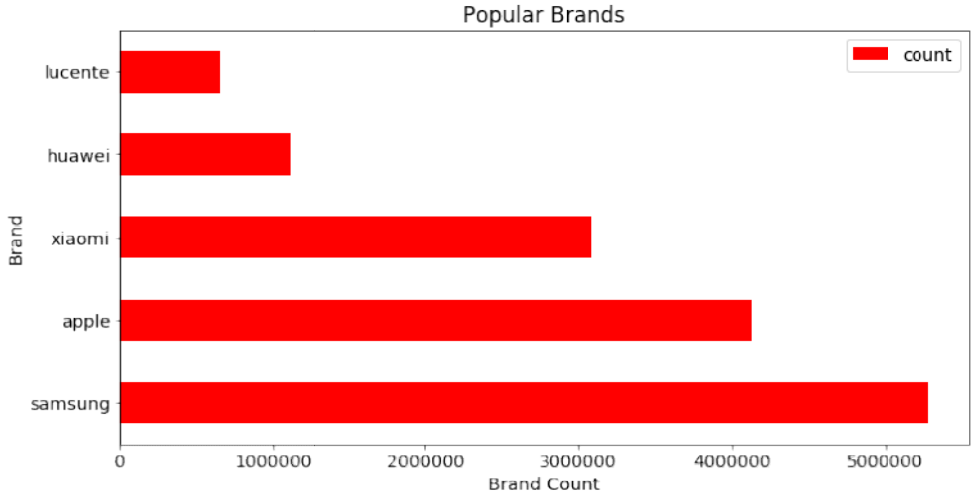
Question_1) Find the 5 most popular Products sold by the e-commerce company in the month.

➢ *As per the assignment guidelines - "Popularity is measured by the frequency of all the activities recorded for the product - view, cart, purchase and remove". We will use the **groupBy aggregate** function over feature 'product_id' with **count** and **orderBy** descending order by count to identify the 5 most popular products and then plot them as a bar graph –*



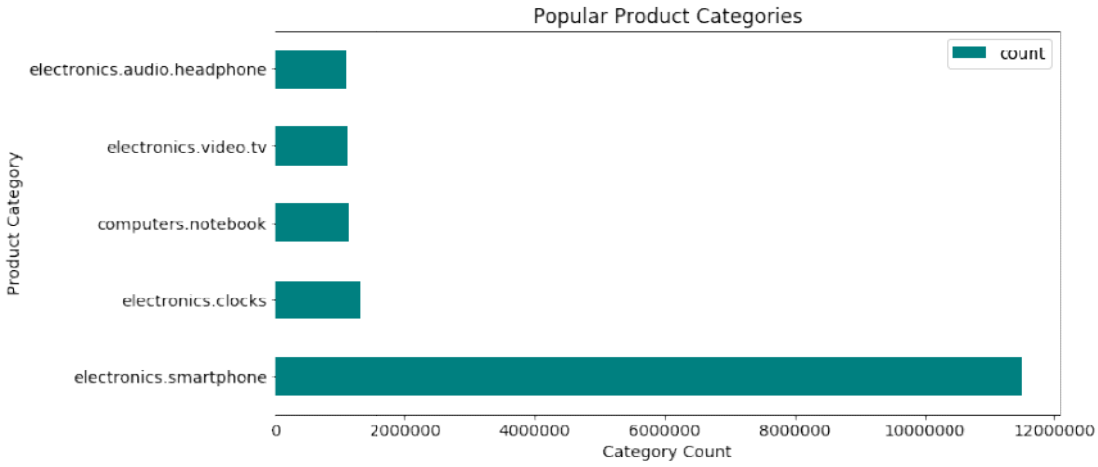
Question_2) Find the 5 most popular Brands on the platform.

➢ *We will use the **groupBy aggregate** function over feature 'brand' and **count** and **orderBy** in descending order by count to identify the 5 most popular brands and then plot them as a bar graph –*



Question_3) Find the 5 most popular Products categories.

➢ *We will first **filter** by non-null values, followed by the **groupBy aggregate** function over feature 'category_code' and **count** and **orderBy** in descending order by count to identify the 5 most popular categories and then plot them as a bar graph –*



Question_4) Find the number of unique users and the most active user on the platform.

➢ *We will use the **groupBy aggregate** function over feature 'user_id' and **count** and **orderBy** in descending order by count to arrive at the full users list in most active first order. The count of the full users list yields the number of unique users, while the topmost entry yields the most active user on the platform –*

Number of unique users = **3022290**

Most active user on the platform is: **UserID = 512475445 with activity count = 7436**

Question_5) Find the average and maximum price for smartphones purchased by the customers.

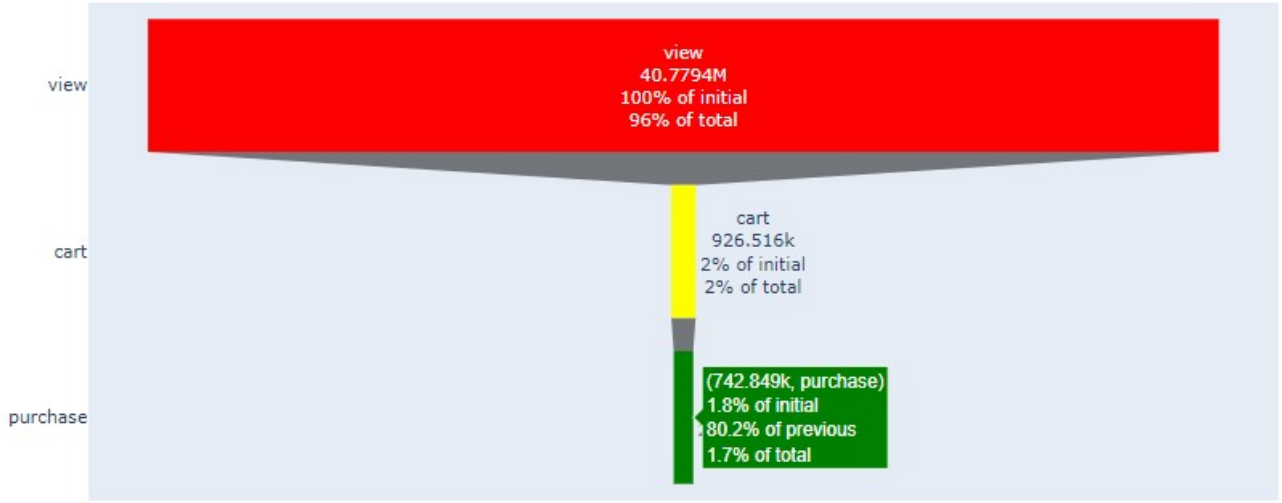
➢ *We will **filter** the dataset by condition as [('category_code' == 'electronics.smartphone') & ('event_type' == 'purchase')] and then **aggregate** using the **mean** and **max** functions over feature 'price' to yield the desired numbers –*

Average price for smartphones purchased by the customers **= 464.62**

Maximum price for smartphones purchased by the customers **= 2110.45**

Question_6) Plot the Event-type funnel distribution in e-commerce shopping journey.

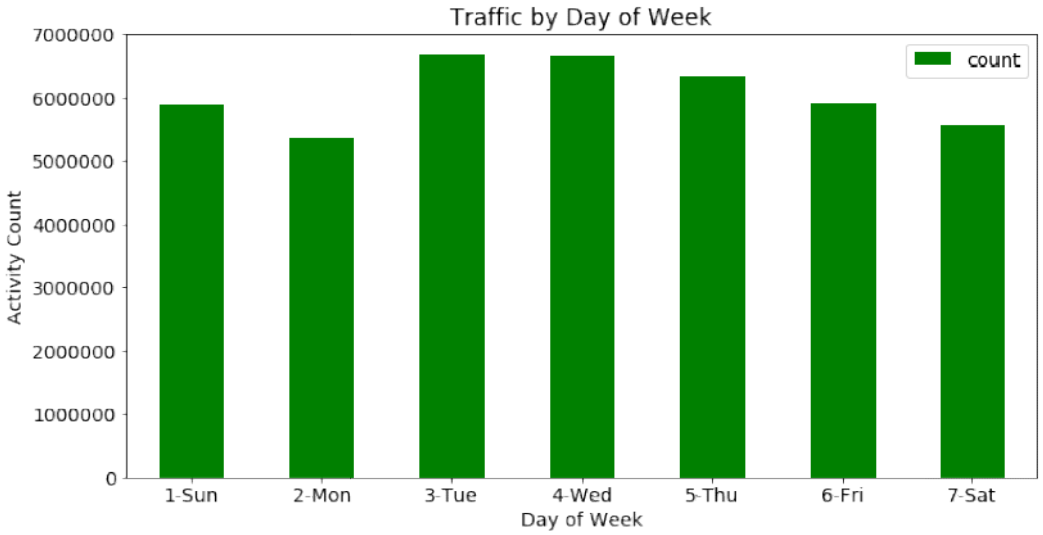
➢ *We will use the **groupBy aggregate** function over feature 'event_type' and **count** and **orderBy** in descending order by count to yield the aggregate counts. We will then use the **plotly.graph_objects** library to plot the event-type funnel distribution - -*

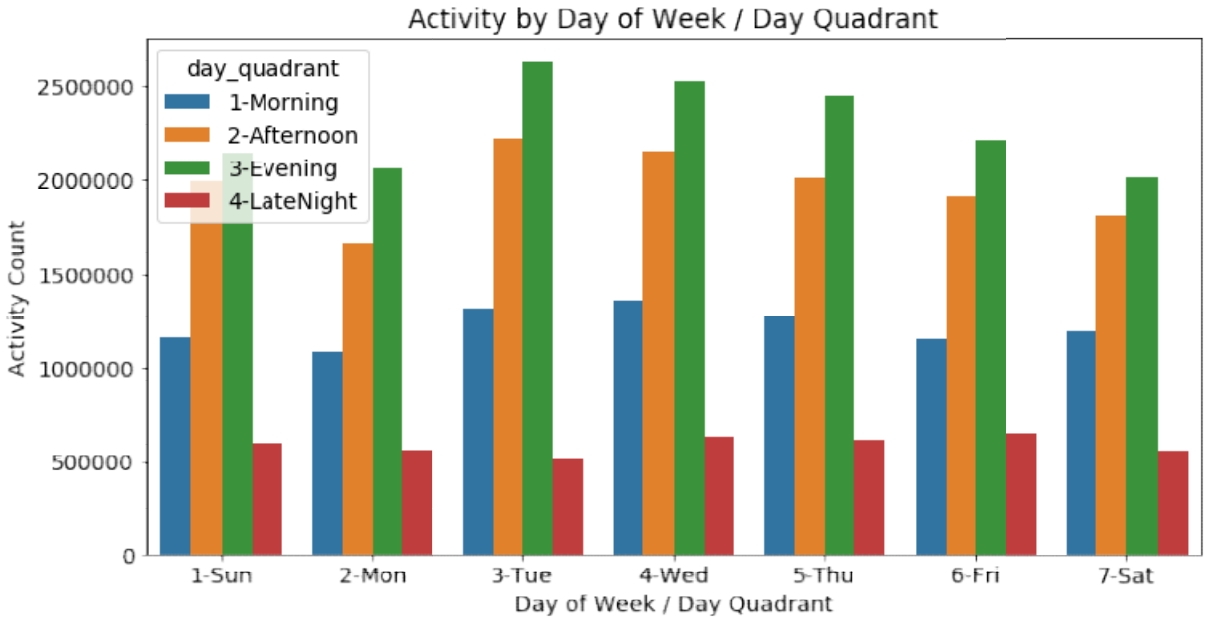


Question_7) Plot the Traffic on different days of the week

➢ *We will first derive a new feature 'day_of_week' from the 'event_time' feature using the **to_timestamp**, **date_format**, **dayofweek** and **hour** functions. We will then use the **groupBy aggregate** function over new feature 'day_of_week' and **count** and **orderBy** in ascending order by count to yield the aggregate counts to be plotted as a bar graph.*

➢ *We will also derive another new feature 'day_quadrant' by binning the hour component from the 'event_time' feature into four buckets. We will then use the **groupBy aggregate** function together over the new features 'day_of_week' and 'day_quadrant' and **count** and **orderBy** in ascending order by count to yield the two-level aggregate counts to be plotted as a separate bar graph -*





Task 2 - Feature Engineering:

Handle missing values (provide justification for approach)

➢ *We will use the **select, count, when** and **isnull** functions to arrive at the missing value counts -*

```
df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in df.columns]).show()
```

```
+----------+----------+-----------+-------------+-------+-----+-------+------------+-----------+----+
|event_type|product_id|category_id|category_code|  brand|price|user_id|user_session|day_of_week|hour|
+----------+----------+-----------+-------------+-------+-----+-------+------------+-----------+----+
|         0|         0|          0|     13515609|6113008|    0|      0|           2|          0|   0|
+----------+----------+-----------+-------------+-------+-----+-------+------------+-----------+----+
```

➢ *The 'category_code' and 'brand' attributes have a significant number of missing values and hence will be replaced with literal value 'unknown' without dropping the rows.*

➢ *However, the 'user_session' column has a minuscule number of missing values and hence those rows will be entirely dropped from the dataset.*

Generate the category code at 2 levels (Split into 2 columns)

➢ *We will use the **split** function over feature 'category_code' to split into 2 columns. Spark treats '.' as special character and so need to escape it as '\.' -*

```
df = df.withColumn('category', F.split(F.col('category_code'),'\.').getItem(0)) \
       .withColumn('sub_category', F.split(F.col('category_code'),'\.').getItem(1))
```

Capture user activity in different columns

1. Total activities (view/cart/etc.) in the session
2. Affinity towards a particular product (Product count for user)
3. Affinity towards a category (Secondary category count for user)
4. Average shopping expense for a product category (secondary)
5. Number of user sessions

➢ *We will use the Spark **Window** with **partitionBy** along with the **count, avg**, and **approx_count_distinct** aggregate functions over the dataset features to engineer new features to capture the user activity counts and average shopping expense numbers -*

```
windowSpec = Window.partitionBy('user_session')
df.withColumn('user_session_activity_count', F.count(F.col('user_session')).over(windowSpec))

windowSpec = Window.partitionBy('user_id', 'product_id')
df = df.withColumn('product_count_for_user', F.count(F.col('product_id')).over(windowSpec))

windowSpec = Window.partitionBy('user_id', 'sub_category')
df.withColumn('sub_category_count_for_user', F.count(F.col('sub_category')).over(windowSpec))

windowSpec = Window.partitionBy('user_id', 'sub_category')
df = df.withColumn('avg_expense_for_sub_category', F.avg(F.col('price')).over(windowSpec))

windowSpec = Window.partitionBy('user_id')
df = df.withColumn('user_sessions_count',
F.approx_count_distinct(F.col('user_session')).over(windowSpec))
```

Impact of time: Day and Hour (Binning hours into 4 buckets)

➢ *We have already derived new feature 'day_quadrant' by binning the hour component from the 'event_time' feature into four buckets labeled as [1-Morning, 2-Afternoon, 3-Evening, 4-LateNight] and plotted the Day of Week / Day Quadrant two-level aggregate traffic counts as a bar chart.*

Reduction in brands for analysis: Top 20 + 'others'

➢ *We will derive the top 20 brands from earlier aggregation and then consolidate all remaining brand values not in this list under the 'others' bucket -*

```
top_20_brands = list(brands_df.head(20).brand)
df = df.withColumn('brand', F.when(F.col('brand').isin(top_20_brands),
F.col('brand')).otherwise(F.lit('others')))
```

Target variable generation: is_purchased

➢ *As per the assignment guidelines: "Every purchase will involve multiple steps. The customer will first view the product, then, add it to the cart, and finally, make the decision of buying or removing it from the cart. Since the model is based on the items that were bought from his/her cart, you must generate the*

column 'is_purchased', which reflects that the user has purchased the item that was present in his/her cart. Note that once the column is generated, you must drop all the additional rows associated with that item to remove redundancy in the dataset. For example, if you have purchased a particular item, you need to remove the multiple entries of that product for that user by taking a look at the column 'event'."

- *From the dataset analysis, we see that there are **4 event_type workflows** :*
  a) Product **'view' => no further action** (e.g. user_session = '0403a55c-7775-457d-af9e-4be364c47c53', product_id = '26300086')
  b) Product **'view' => 'cart' => no further action** (e.g. user_session = '041777b7-b2d3-4929-a7dc-c5dc8dc0be09', product_id = '1004767')
  c) Product **'view' => 'purchase' DIRECTLY** (e.g. user_session = '0403a55c-7775-457d-af9e-4be364c47c53', product_id = '26500314')
  d) Product **'view' => 'cart' => 'purchase'** (e.g. user_session = '04223ea8-7220-41df-9b56-e23c068822d9', product_id = '1005169')

- *Accordingly we will take the following actions for each event_type workflow :*
  - All 'view' event_type rows will be dropped since their contribution is already captured in the newly engineered user activity columns
  - The remaining event_types ('cart' and 'purchase') for each product_id will be collected as a set in another 'all_event_types' engineered column after grouping by Window.partitionBy('user_session', 'product_id')
  - The target variable 'is_purchased' will be set to 1 if the new column 'all_event_types' contains the 'purchase' event, otherwise it will be set to 0
  - Finally the new 'all_event_types' column as well as the 'event_type' column will be dropped before invoking the df.dropDuplicates() function

---

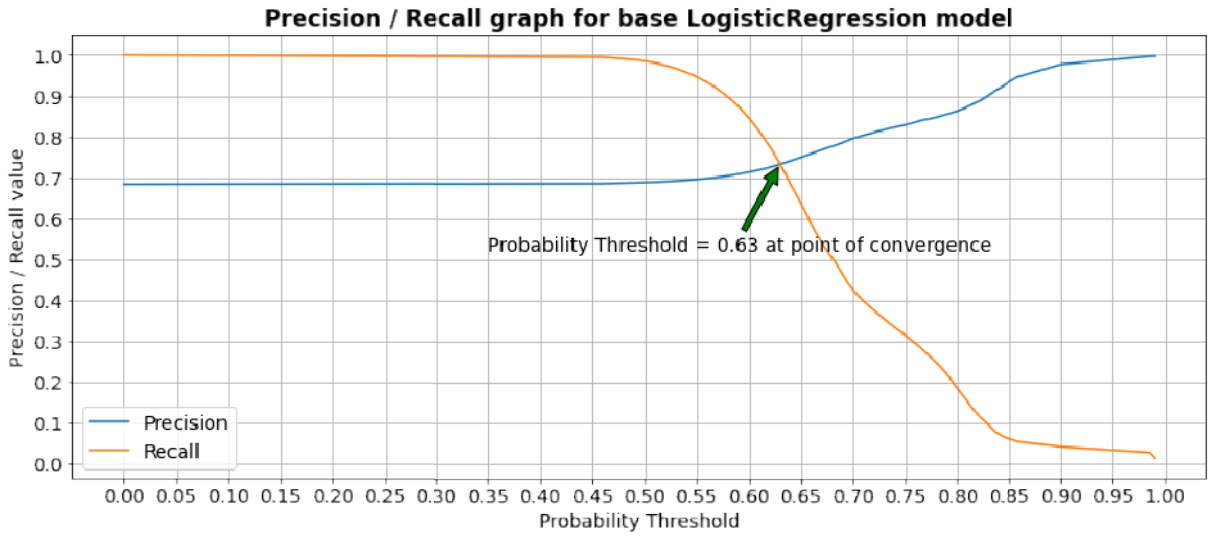
Task 3 - Model Selection:

Logistic Regression

- *Feature selection will be done based on distinct value counts of categorical columns and high-count features will be eliminated accordingly.*
- *The categorical columns will be encoded as numerical using the Spark **StringIndexer** and **OneHotEncoderEstimator** classes in stage 1 and 2 respectively.*
- *In stage 3, the encoded categorical features will be combined with the numeric continuous features and assembled into the 'features' vector using the **VectorAssembler.***
- *All the 3 stages will be combined into a Pipeline and fitted together on the input dataset.*
- *The resulting transformed dataset will be persisted to local storage for future use.*
- *The transformed dataset including the 'features' vector is then split into random Train and Test portions in a 70:30 ratio with seed=42 for consistent splits over multiple iterations.*
- *The model is then built using the **LogisticRegression** class and fitted over the Train dataset.*
- *The fitted model is then evaluated against the Test dataset across the performance metrics **Accuracy, Precision, Recall, F1_score and AreaUnderROC.***
- *From the test results, we see that RecallByLabel = [0.035069313586685275, 0.9853016642734399] which is excellent for the Positive class but fares poorly for the Negative class classification.*
- *The **Precision-Recall graph** is then plotted to visualize the metrics over the entire probability threshold range. Here we can see that the two metric values converge approximately at probability value = 0.63, which can then be considered as the optimum point for the cutoff probability threshold.*
- *The model is then re-trained with hyper-parameter tuning as [threshold = 0.63] and the performance metrics are re-evaluated to yield the final model values –*

Evaluation metrics for LogisticRegression Model :

1. Accuracy = 0.6351784713859786
2. Precision = 0.7335212223468531
3. Recall = 0.7327511122619524
4. F1_score = 0.73313596506723
5. AreaUnderROC = 0.633965520575469



Decision Trees

- *The transformed dataset from earlier will be retrieved for the DecisionTree model building.*
- *The transformed dataset including the 'features' vector is then split into random Train and Test portions in a 70:30 ratio with seed=42 for consistent splits over multiple iterations.*
- *We will use the **ParamGridBuilder** class with **maxDepth=[10,20,30]** and **maxBins=[5,10,15]** for the hyper-parameter tuning.*
- *We will then use the **BinaryClassificationEvaluator**, **DecisionTreeClassifier** and **CrossValidator** classes to run the cross-validation steps with 3 folds against this grid.*
- *The best model is extracted from the cross-validation results and evaluated against the Test dataset using the **BinaryClassificationEvaluator** and **MulticlassClassificationEvaluator** classes.*
- *The performance metrics **Accuracy, Precision, Recall, F1_score and AreaUnderROC** are calculated to yield the final model values –*

Evaluation metrics for the best DecisionTree Model :

1. Accuracy = 0.8785352544096166
2. Precision = 0.8790522005363621
3. Recall = 0.8785352544096166
4. F1_score = 0.8787936514502295
5. AreaUnderROC = 0.7396132709059026

Random Forest

- *The transformed dataset from earlier will be retrieved for the RandomForest model building.*
- *The transformed dataset including the 'features' vector is then split into random Train and Test portions in a 70:30 ratio with seed=42 for consistent splits over multiple iterations.*

- *We will use the **ParamGridBuilder** class with **impurity=['gini', 'entropy']** for the hyper-parameter tuning.*
- *We will then use the **BinaryClassificationEvaluator**, **RandomForestClassifier** and **CrossValidator** classes to run the cross-validation steps with 3 folds against this grid.*
- *The best model is extracted from the cross-validation results and evaluated against the Test dataset using the **BinaryClassificationEvaluator** and **MulticlassClassificationEvaluator** classes.*
- *The performance metrics **Accuracy, Precision, Recall, F1_score and AreaUnderROC** are calculated to yield the final model values –*

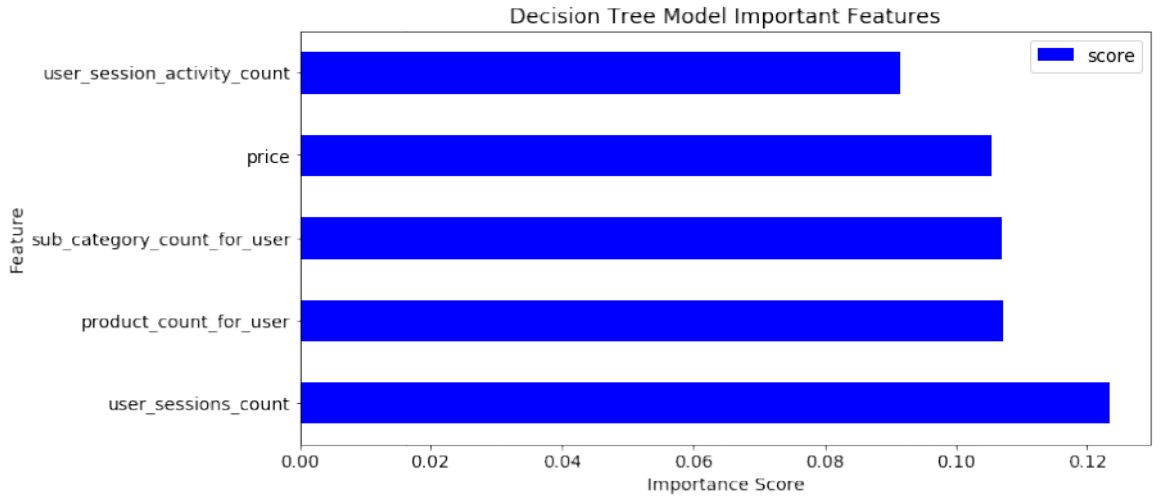
Evaluation metrics for the best RandomForestClassifier Model :

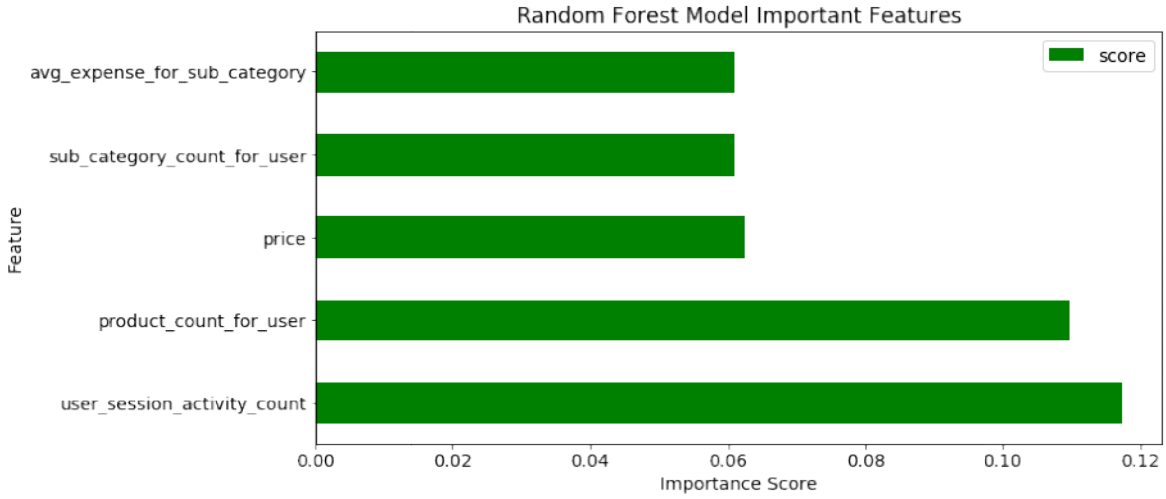
1. Accuracy = 0.7175046171468284
2. Precision = 0.7223446670705886
3. Recall = 0.7175046171468284
4. F1_score = 0.7199165072005896
5. AreaUnderROC = 0.7299482874299179

Task 4 - Model Inference:

Feature Importance and Feature Exploration

- *We will plot the top 5 Feature Importance scores as learnt by the DecisionTree and RandomForest models -*



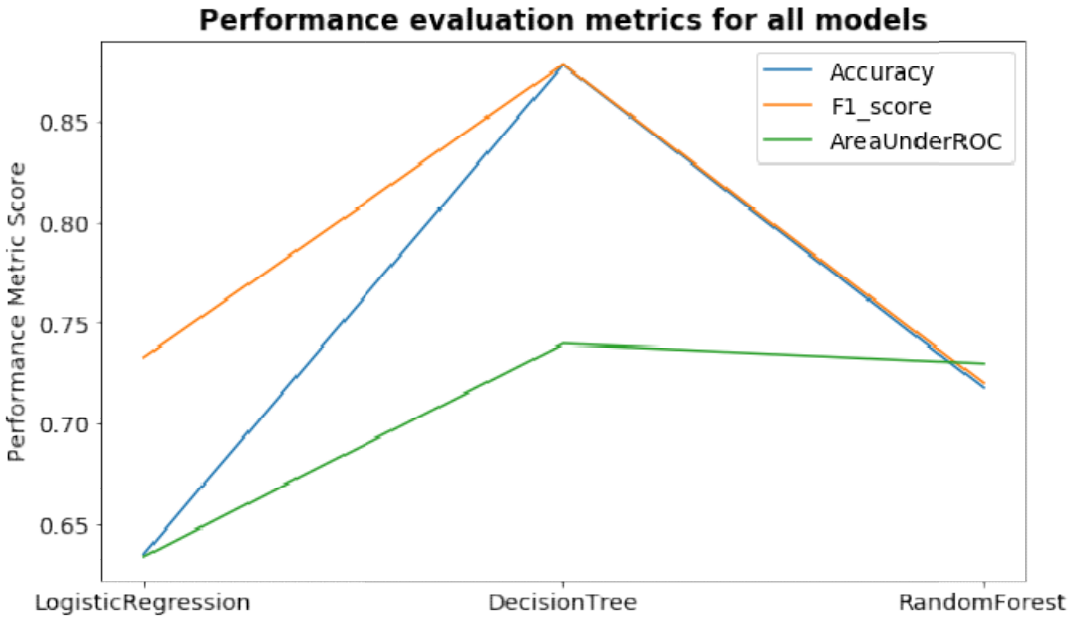


- *Based on the Important Features as learnt by the DecisionTree and RandomForest models, the recommendations for the E-commerce company would be :*

1. Explore marketing campaigns to drive traffic to the platform in order to increase the 'user_sessions_count' and 'user_session_activity_count' derived feature numbers
2. As indicated by the 'sub_category_count_for_user' and 'avg_expense_for_sub_category' derived features, display similar products belonging to the same sub_category and price range based on the user cart activity to increase the probability of purchase
3. Based on the 'product_count_for_user' and 'price' features, consider offering spot discounts for specific products when the view counts within the user session exceed certain configurable thresholds

Model Inference

- *The performance metrics - **Accuracy**, **F1_score** (derived from **Precision & Recall** scores), **AreaUnderROC** - are plotted together for all the models*
- *We can see that DecisionTree scores the highest across all metrics, with RandomForest coming very close on the AreaUnderROC metric.*



Considering the AreaUnderROC metric as the selection basis for this assignment, the recommended model would be RandomForest for its expected performance against unseen data due to a lower risk of overfitting based on its ensemble learning approach.