Data Pre-processing

In the query we are selecting a total of 17 columns. We are grouping everything by add_to_cart, hour, minute, product_name, product_category, and product_price. The SELECT DISTINCT ensures that we do not have any duplicate rows. The query yields a total of 1,236,880 rows.

With the print(hits_df.isnull().sum()), we can see the total of missing values for each column. The bounces column has a total of 1,236,880 which is the maximum row. It makes sense because we wanted to pull only sessions that are not bounced, which is indicated by the null value. We can go ahead and drop this column and the session_id from the data.

To handle the rest of the columns that have null values which are page_views, time_on_site, product_category, product_name, product_price, we need to fill it with the mean or most common value. The following lines will find the average or most common values of each column.

```
avg_page_views = round(cleaned_hits_df['page_views'].mean())

avg_time_on_site = round(cleaned_hits_df['time_on_site'].mean())

common_product_category = cleaned_hits_df['product_category'].value_counts().index[0]

common_product_name = cleaned_hits_df['product_name'].value_counts().index[0]

avg_product_price = round(cleaned_hits_df['product_price'].mean())
```

Then these lines will replace the nulls with new values we found above.

```
cleaned_hits_df['page_views'] = cleaned_hits_df['page_views'].fillna(avg_page_views)

cleaned_hits_df['time_on_site'] = cleaned_hits_df['time_on_site'].fillna(avg_time_on_site)

cleaned_hits_df['product_category'] = cleaned_hits_df['product_category'].fillna(common_product_category)
```

```
cleaned_hits_df['product_name'] = cleaned_hits_df['product_name'].fillna(common_product_name)
```

```
cleaned_hits_df['product_price'] = cleaned_hits_df['product_price'].fillna(avg_product_price)
```

If we check for the nulls again every column will have a total of zero.

The day, month, and year columns are formatted as object/string data type but they should be in int64. To convert these columns to integers we can run the following lines.

```
cleaned_hits_df['day'] = pd.to_numeric(cleaned_hits_df['day'])
cleaned_hits_df['month'] = pd.to_numeric(cleaned_hits_df['month'])
cleaned_hits_df['year'] = pd.to_numeric(cleaned_hits_df['year'])
```

If you haven’t noticed, the product prices are all very big numbers, and the reason for that is because it was timed by 10 to power of 6. To redo this math we can run the following line.

```
cleaned_hits_df['product_price'] = round(cleaned_hits_df['product_price'] / (10 ** 6), 2) # revert the 10^6
```

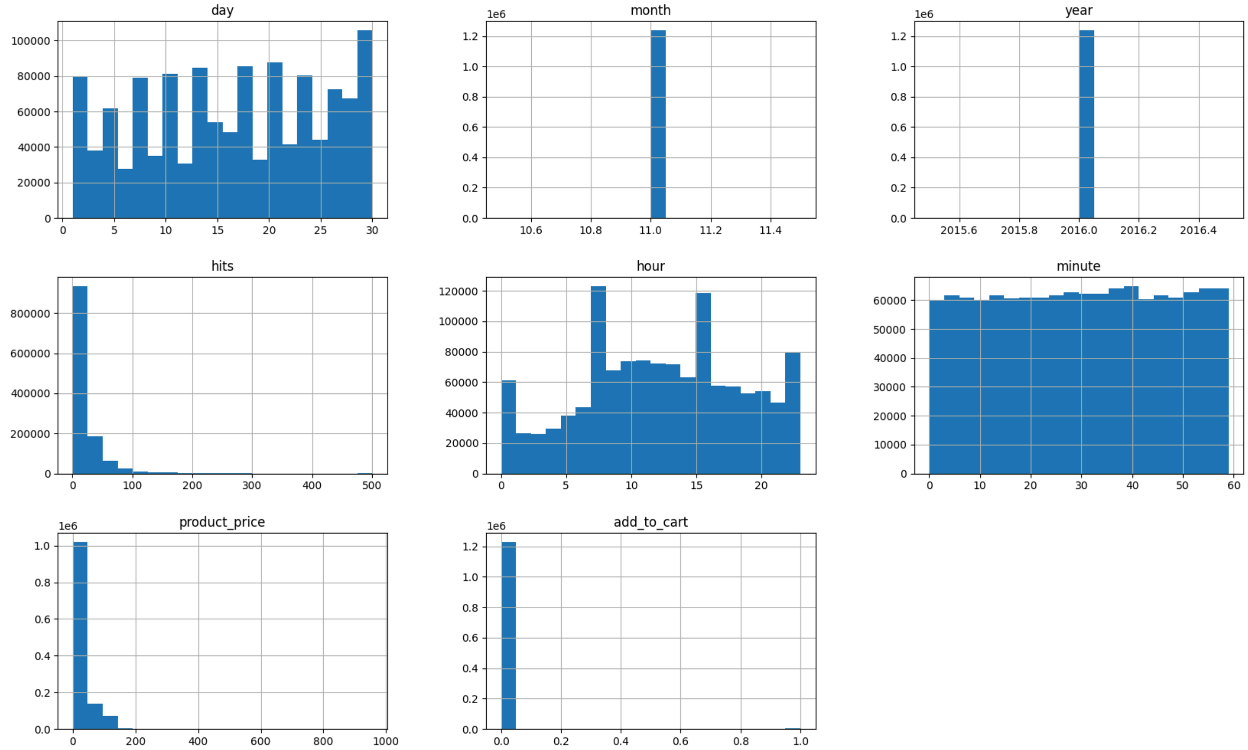
The data should be now reduced to 15 columns and has now null values. We can proceed to EDA.

Exploratory Data Analysis

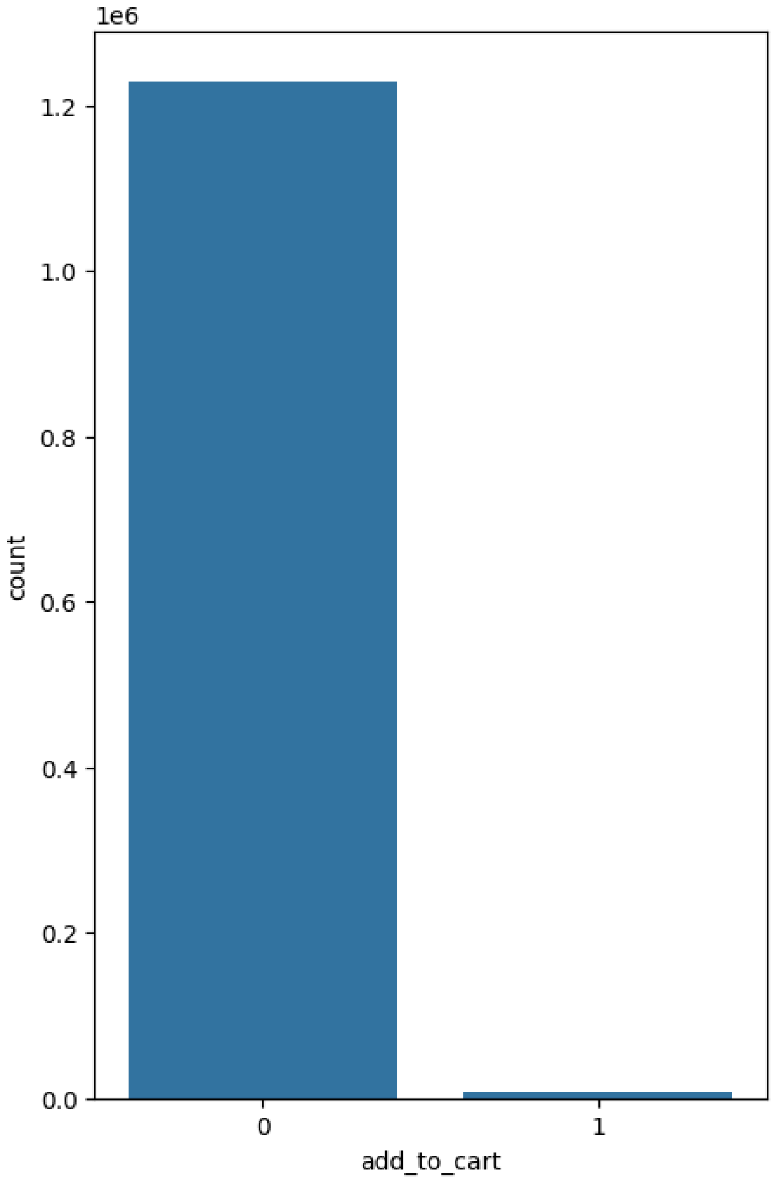
Before we perform any EDA we need to identify which columns are categorical and which ones are numeric. The reason to do this is because some graphs and charts can only read numbers.

categorical_columns = ['device', 'sub_continent', 'country', 'product_category', 'product_name']

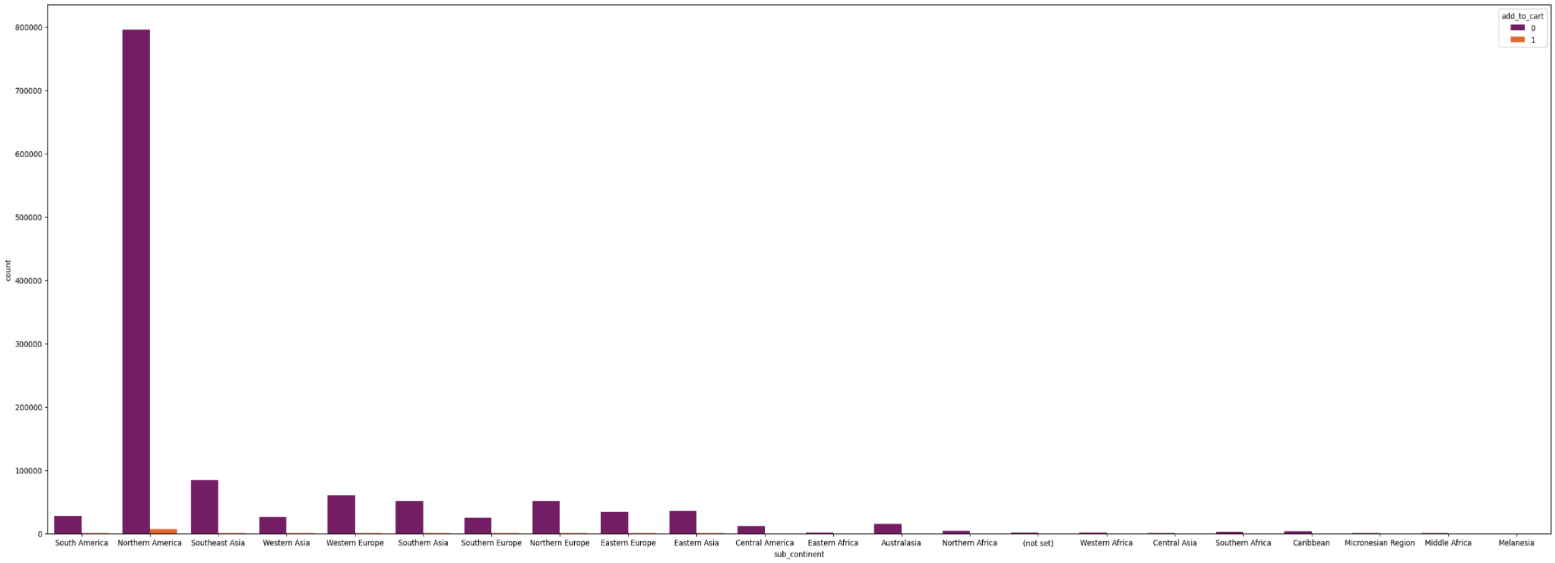
numeric_columns = ['day', 'month', 'year', 'hits', 'page_views', 'time_on_site', 'hour', 'minute', 'product_price', 'add_to_cart']



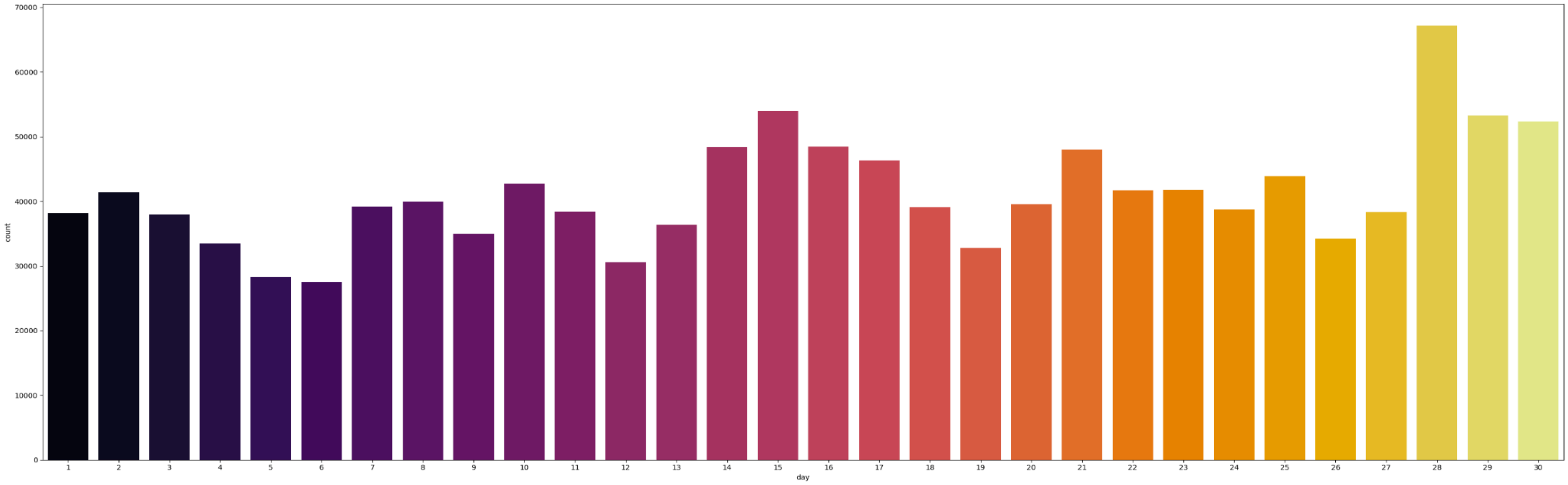
These are histogram charts showing the distribution of numerical columns. This chart is good for identifying outliers. As we can see there are some groups of “hits” that are far from the rest of the groups, it may be a good idea to standardize these numbers so they can fall within a range.



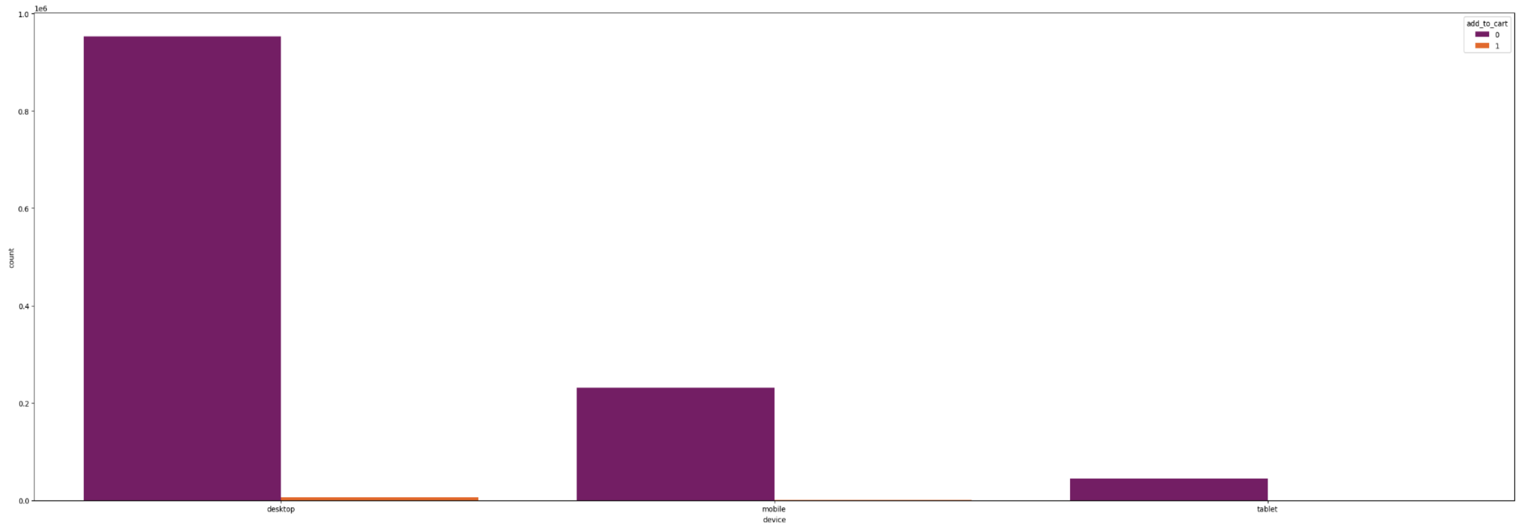
This graph shows the total number of our target, add_to_cart, which are separated into two groups, 1s and 0s. 1 means the add_to_cart action was performed and 0 means it was not. It’s obvious that there are significantly more 0s than 1s.



This graph shows the distribution of add_to_cart actions across different sub continents. At a quick glance we can tell most hits originated from Northern America. This indicates that there are more chances that an add to cart action will occur within Northern America.



This graph shows the distribution of hit sessions across the whole month of November, 2016. There seems to be more hits toward the end of the month, which is the week of Thanksgiving and cyber Monday sales.



This graph shows the total types of devices the hits came from. Most of the hits originated from Desktop.

Model Choice

Three models were run as tests, Support Vectors Classifier, KNeighbors Classifer, and DecisionTree Classifier. All three models yield a similar score of .993 accuracy score, meaning that models predicted about 99% of the labels corrected.

For the final model, the dataset was uploaded to Vertex AI and trained on AutoML.

Definitions

Precision: the ratio between true positive and all positive predictions made by the model, a number closer to 1.0 is better.

True Positives+False Positives
True Positives

Recall: proportion of true positive prediction over actual positive labels, a number closer to 1.0 is better.

True Positives+False Negatives
True Positives

F1: combination of recall and precision, a number closer to 1.0 means better.

Micro-average F1: the average F1 score of each class, give equal weights to each sample. Good for imbalanced classes. A number closer to 1.0 is better.

Macro-average F1: the average of F1 score without considering imbalance classes. Good for balanced classes. A number closer to 1.0 is better.

Micro-average precision: True positives of all classes divided by the sum of true positives and false positives. A number closer to 1.0 is better.

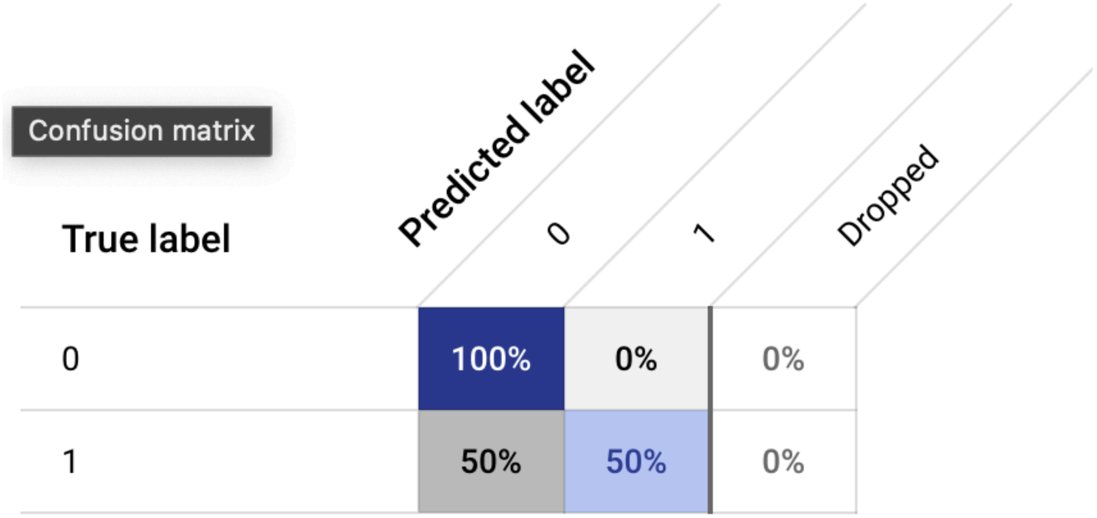
Micro-average recall: True positives of all classes divided by the sum of true positives and false negatives. A number closer to 1.0 is better.

Model’s Metrics

After training the model, the resulting metrics are as below.

Micro-average F1: 0.9962491
Macro-average F1: 0.81699014
Micro-average precision: 99.6%
Micro-average recall: 99.6%

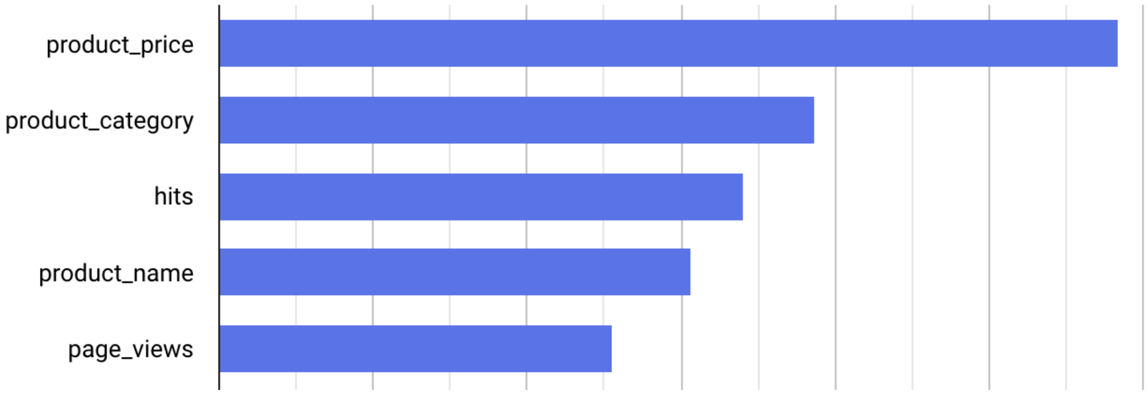
Confusing metrics



Confusion matrix

Predicted label

True label	0	1	Dropped
0	100%	0%	0%
1	50%	50%	0%



This chart shows the importance of each feature. As shown, the feature that contributed the most to predicting the target is product_price.