Tesco Data Scientist Test

October 22, 2018

Q1

The mean of the transactions dataset is

```
Out[48]: 286.17605
```

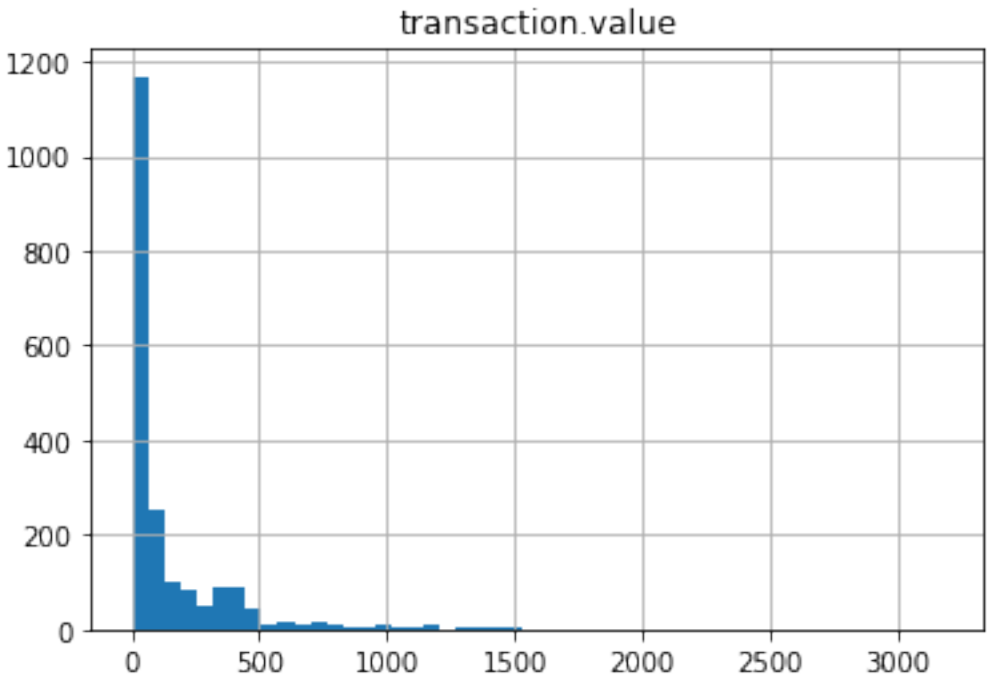
whereas the median is

```
Out[49]: 47.275000000000006
```

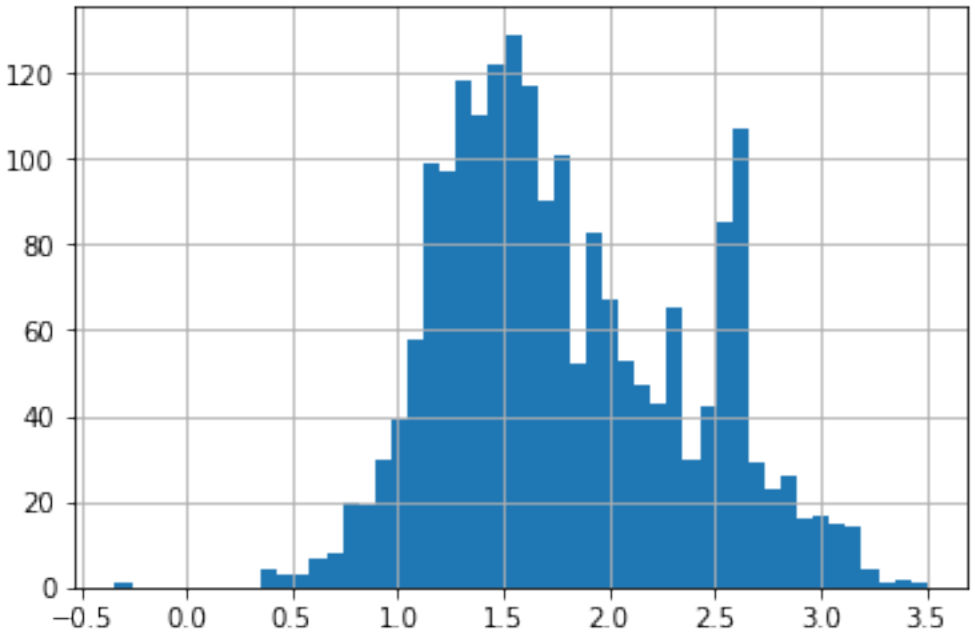
There are 2 reasons for this large difference between the mean and median. First there are a few very unusual large transactions which should be investigated:

```
Out[50]:       transaction.value  gender store.type
         1428          150032.00  Female      Metro
         50            100000.00  Female      Extra
         1382           11840.25  Female      Extra
         1375            3180.76    Male      Extra
         667             2502.39  Female      Extra
         675             2288.23    Male      Extra
         1251            2174.41    Male      Extra
         1284            1832.73  Female      Extra
         481             1701.38  Female      Extra
         1499            1615.51  Female      Extra
```

Removing the 3 largest entries and plotting a histogram we see a typical Gamma-like skewed distribution which is the second reason for the difference in mean and median.



Using a log plot on the x-axis we see that there are actually several peaks in the distribution with the main one being at 1.6 (which equates to £40 under log base 10) and several more peaks for larger values.



Q2

To determine if there is a significant difference in the average spend we will assume that each transaction is an independent random variable drawn from some (unknown) probability distribution. We can then use the central limit theorem that any sum of independent random variables is approximated by a standard normal distribution. Given the sample mean and standard deviation of transaction value for each store type we can use a t-test on pairs of store type means to test for significant difference. Note: the 3 largest values have been removed from consideration as outliers that need to be investigated.

Out[53]:

	transaction.value	
	mean	std
store.type		
Express	86.349468	191.863120
Extra	256.231125	387.696657
Metro	113.176605	218.942121
Superstore	155.166490	215.615642

We can use the scipy t-test `ttest_ind(...)` to perform a 2-sided t-test for each of the pairs of store types. The results below indicate using 5% (and even a 1%) level there are significant differences in average spend across all store types aparat from Express and Metro.

Out[54]:

	Pairs	p-value
0	express-metro	8.742951e-02
1	express-superstore	2.224495e-06
2	express-extra	2.382553e-11
3	metro-superstore	7.213495e-04
4	metro-extra	9.507566e-12
5	superstore-extra	5.935764e-09

Methodological note: comparing several sets of variables like this using a standard t-test can be problematic as the t-values ought to be adjusted e.g. using Bonferroni method. However given the magnitudes of the the p-values we wouldn't get a different outcome.

Q3

The key issue with this type of problem is the highly imbalanced dataset. Very few people click on the content, in the case of content_1 ~1%

To approach the problem we will use a Random Forest classifier, split the data randomly 80:20, train on the 80% and valiate the approach on 20%. The resulting model is then used to predict on the test data. There are a number of different metrics that can be used to measure performance on models for an unbalanced dataset, a commonly used one is Receiver Operating Characterstic (ROC). The code to train and validate the model is in the accompanying python notebook. The ROC curve is shown below.

The test data has been passed through the model and the predicted outputs included in the accompanying csv file.

Q4

A system to determine what content to show a user could use a real-time predictive model using, for example, Random Forests.

Training

Assumption: Labelled historical production data is available to an Apache Spark system

Using Apache Spark MLLib train a number of Random Forest (RF) models on the historical data to predict each of the different content interaction labels. Export the models as PMML (Predictive Model Markup Language).

Real-time model scoring

Model scoring is run as a REST-based service e.g. openscoring. The RF models are uploaded to the open scoring service using provided APIs. At runtime scoring is performed by a REST call to the openscoring service. Performance of the model is monitored via the openscoring metrics API.

Multipe openscoring instances fronted by a load-balancer can be used to provide resilience and scaling.

Model updating

Blue-green updating of the model can be supported by versioning the model using openscoring APIs and building support for calling different model versions into the REST calling code. When a new version of the model is being trialled both the current and new version can be evaluated and the results compared off-line by data scientists. When the performance of the new model is judged to be satisfactory the old model can be retired.