1st Time Payments Analysis

Patrick Tjahjadi

A dataset is given that includes sample data on completed ride-hailing car rides worldwide for users that pay using credit card for the first time. Let the status of a successful credit card payment be represented with is_successful_payment and assume that the table is named *ride*. A simple SQL query allows us to conclude that **79033 out of 225020 rides (35.1%) failed their credit card payments**.

```
SELECT is_successful_payment, COUNT(*) AS num_successful_trx
FROM ride
GROUP BY is_successful_payment;
```

SQL query to retrieve the number of failed and successful credit card payments.

Failed credit card payments are usually a form of fraudulent transaction where such rides were not genuine (i.e., a customer did not use a ride-hailing service to go from origin to destination using a random driver with a valid payment method). We can prevent this by developing fraud rules that prevent failed payments from happening for customers who use credit card for the first time.

Several assumptions must be made upon developing these fraud rules:

1. If price = 0, then obviously is_successful_payment will be 0 because there would not be a credit card transaction. Since there can be no successful credit card transactions when the price is 0, a rule could not be developed for free rides. The solution is not to accept the credit card altogether for all free rides, with or without discounts.
2. The parameter ride_distance covers the distance of the trip done by the driver, as opposed to a pre-calculated GPS route, regardless of it being optimised.
3. The parameter price_review is conducted before initiating the credit card payment, and hence the audit can determine whether the ride price is too high before a failed payment.
4. Unsuccessful credit card transactions are usually caused by illegitimate orders. Hence, rides with unusual characteristics (e.g. very high trip distance or price) most likely cause these unsuccessful transactions.
5. These developments should not automatically reject credit card transactions but rather be reviewed further using human inspection as the rules would still capture some possibly legitimate transactions. Moreover, these developments can be implemented after finishing the trip but before the payment is done. Hence, predictions can be made using all the data including the ride distance with ample time to assess the success of a credit card transaction before the payment.

Development Rule 1: Distance-based ruling

Fraudulent rides usually come in the form of very short or long-distance rides, particularly when the distance between origin and destination is short but the distance covered during the ride is very long. Some data derivation can be done to create a distance-based rule.

Let lat1 and lng1 be the latitude and longitude for the origin, respectively. Let lat2 and lng2 be the latitude and longitude for the destination, respectively. Let r be the radius of the earth. The geodesic distance between the origin and destination (i.e., the literal shortest routes between two points) can be defined using the formula:

$$V = r * \arccos\left(\sin\left(lat1 * \frac{\pi}{180}\right) * \sin\left(lat2 * \frac{\pi}{180}\right) + \cos\left(lat1 * \frac{\pi}{180}\right) * \cos\left(lat2 * \frac{\pi}{180}\right) * \cos\left(lng2 * \frac{\pi}{180} - lng1 * \frac{\pi}{180}\right)\right)$$

Then V is defined as the Vincenty's distance: the geodesic distance between two points for an oblate spheroid. Assume that the radius of the Earth is constant at 6371 kilometres for all points. Hence, the Vincenty distance of all rides can be calculated.

lat	lng	real_destination_lat	real_destination_lng	vincenty_distance
58.37822	26.7104	58.363243	26.737696	2.303609169
59.42413	24.64636	59.397548	24.660957	3.069034267
59.41351	24.74371	59.4485	24.804887	5.206717167
59.41994	24.7448	59.431686	24.720801	1.883658458
59.47133	24.89056	59.427836	24.77446	8.151377162
59.42759	24.77496	59.395243	24.663802	7.24591114
59.42727	24.77446	59.404176	24.636704	8.205852925
59.40886	24.72388	59.417374	24.797121	4.251070868
59.38357	24.84299	59.428953	24.720008	8.597252534
59.42131	24.67007	59.421664	24.670887	0.060534486
59.4268	24.55029	59.415956	24.798762	14.10669893
58.37359	24.50303	58.382956	24.487734	1.370874827
59.42154	24.79188	59.306443	24.832972	13.00866047
59.43328	24.79124	59.401443	24.696787	6.410088958

A sample of Vincenty distance calculated using two points for each ride, in kilometres.

The parameter vincenty_distance can provide insights when compared to ride_distance, such as whether long trips for short distances are suspicious (i.e., short vincenty_distance with long ride_distance). Based on the dataset, long distance trips usually end up with failed credit card transactions. A hypothesised scenario is that drivers can take exorbitantly long routes to reach the intended destination. Such suspicious trips usually end up in failed credit card transactions because genuine trips do not exhibit such behaviour.

We can develop a percentile table for distances and analyse the bottom and top percentiles for suspicious rides.

Percentile	0.001	0.01	0.1	0.5	0.9	0.99	0.999
Ride_distance (m)	2	676.7	3766	6153	13997.8	28051.3	56114.7
Vincenty_distance (km)	0.007	1.4	2.5	4.63	7.94	12.7	26.8

The top 0.1% of rides has trips of 56114 metres or longer. The bottom 0.1% of rides have trips of 2 metres of shorter. Data filtering suggests that most failed transactions happen in these distances. Using the SQL query below, we can retrieve the proportion of failed credit card transactions for these trips:

```
SELECT is_successful_payment, COUNT(*) AS num_successful_trx
FROM ride
WHERE ride_distance >= 56114 OR ride_distance <= 2
GROUP BY is_successful_payment;
```

SQL query to retrieve the number of failed and successful credit card payments for ride distance less than or equal to 2 metres and ride distance more than or equal to 56114 metres.

This query yields the following table:

Query	
is_successful_payment	**num_successful_trx**
0	1017
1	779

Although the current rule can capture 1017 failed credit card transactions, it would hypothetically prevent some credit card transactions which would otherwise have been a legitimate transaction. In this case, the capture rate is 1017 out of 1796 cases (56.6%).

This can be increased by considering Vincenty's distance. If the Vincenty's distance is small but ride distance is high, it is usually a sign of an illegitimate ride, which results in fraudulent credit card transactions.

We introduce a new parameter: distance_ratio which is the proportion of ride_distance compared to vincenty_distance. A high distance ratio implies the distance covered by the ride is significantly higher than the actual distance between the origin and destination.

$$distance\_ratio = \frac{ride\_distance}{(vincenty\_distance \ * 1000)}$$

Data is pre-processed so that rides with a vincenty_distance of 0 (i.e., same latitude and longitude for origin and destination) are given an arbitrarily small value to prevent a zero division.

There are two conditions to filter regarding distance_ratio:

1. If distance_ratio < 1, it implies that the trip covers less distance than the geodesic distance between the origin and destination. This is illogical and possibly fraudulent. However, to accommodate for noise, we will filter for distance_ ratio < 0.2.
2. If distance_ratio > 5, it implies that the trip covers significantly more distance than what perhaps is the most optimised route. This is possibly fraudulent as the driver may have driven using longer routes to abuse the system.

```
SELECT is_successful_payment, COUNT(*) AS num_successful_trx

FROM ride

WHERE (ride_distance >= 56114 OR ride_distance <= 2) AND ((ride_distance / 1000 /
vincenty_distance) < 0.5 OR (ride_distance / 1000 / vincenty_distance ) > 5)

GROUP BY is_successful_payment;
```

SQL query to retrieve the number of failed and successful credit card payments using two conditions: either ride_distance is more than or equal to 56114 or less than or equal to 2 metres and the distance ratio is more than 5 or less than 0.5.

This query yields the following table:

Query	
is_successful_payment	**num_successful_trx**
0	618
1	312

The updated rule can capture 618 out of 930 cases. This yields an accuracy of 618/930 = 66.5% which is an increase from 56.6% without considering distance ratio, which is a fair prediction for failed credit card transactions.

A more optimised approach may be acquired through simulation, that is, setting different thresholds for ride_distance and distance_ratio resulting in different accuracies. The highest accuracy would be the optimal solution for ride_distance and distance_ratio.

To summarise:

Flag first-time credit card transactions for ride distances more than or equal to 56114 or less than or equal to 2, and with the distance ratio more than 5 or less than 0.5. This rule can be used to capture and reduce many failed transactions.

Development Rule 2: Binomial regression-based ruling

It is helpful to consider all possible variables and the interaction between these variables to predict whether a credit card transaction would be successful. Predictions can be made by considering these variables through training a regression model.

There are several possible dependent variables that can be used to predict is_successful_payment:

Dependent Variable	Description
vincenty_distance	Geodesic distance between origin and destination in kilometres
ride_distance	Distance covered by the ride in metres
distance	Distance between the driver and the customer pickup spot
price	Fee paid by the customer after discounts
ride_price	Base fee of the ride
price_review_reason	Reason of the price review to be requested, either automatically or through audit
failed_attempts	Number of failed order attempts before the ride's order

Other dependent variables, such as user_id, lat, lng, and order_id would make no sense in predicting the success of a credit card transactions.

Moreover, it is suspected that there is some dependency and interaction between vincenty_distance and ride_distance. There could possibly be an interaction between price and ride_price as well. Other dependent variables should be independent of each other.

```r
# read and analyse csv ------------------------------------------------------
ride = read.csv(file = 'preprocessed.csv')


# remove white spaces from price_review_reason and categorise it
ride$price_review_reason = trimws(ride$price_review_reason)
ride$price_review_reason = factor(ride$price_review_reason)

# remove rows with price = 0
for (i in 1:nrow(ride)){
  if (ride[i, 16] == 0) {
    ride = ride[-c(i),]
  }
}

summary(ride)
```

R code used to read and pre-process the dataset.

The challenge, however, is that price may vary between countries since the currencies differ. Moreover, some pre-processing is required that removes free rides from the dataset since they are irrelevant in predicting the success of credit card transactions. Lastly, we suspect that the distribution for each dependent variable is not normally distributed, and therefore a generalised linear model should be developed. A logistic regression is appropriate since we are interested in predicting either of the two outcomes: successful payment (1) or failed payment (0).

We can separate the dataset into two: a training set that is used to train and build a predictive model and a testing set that is used to test the model prediction against the actual outcome to evaluate its accuracy. 80% of the dataset is used for training while 20% is used for testing.

```
# use 80% of the dataset to train the regression model
train_ride = ride[1:(nrow(ride)*4/5), ]
test_ride = ride[(nrow(ride)*4/5+1):nrow(ride), ]
```

R code used to separate the ride dataset into training and testing.

The logistic regression model can be developed using R, by considering the variables mentioned above.

```
# create a logistic regression model ------------------------------------
model = glm(is_successful_payment ~ vincenty_distance*ride_distance + distance +
            + price * ride_price + price_review_reason + failed_attempts,
            family = "binomial", data = train_ride)

summary(model)
```

R code used to build the logistic regression model.

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.5272   0.3050   0.3334   0.3820   8.4904

Coefficients:
                                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                                  3.201e+00  1.625e-02 196.983  < 2e-16 ***
vincenty_distance                            3.261e-02  2.201e-03  14.817  < 2e-16 ***
ride_distance                               -6.790e-05  1.343e-06 -50.556  < 2e-16 ***
distance                                     6.101e-06  4.400e-06   1.387 0.165560
price                                        6.760e-04  9.052e-05   7.468 8.13e-14 ***
ride_price                                  -6.364e-04  8.912e-05  -7.141 9.26e-13 ***
price_review_reasoncalculation_failed       -1.393e+00  4.093e-01  -3.403 0.000666 ***
price_review_reasonprice_calculation_warnings  8.851e+00  4.080e+01   0.217 0.828247
price_review_reasonprice_too_high            1.775e-01  1.202e-01   1.476 0.139818
failed_attempts                             -2.721e-01  6.307e-03 -43.138  < 2e-16 ***
vincenty_distance:ride_distance              9.806e-08  9.953e-09   9.852  < 2e-16 ***
price:ride_price                             2.756e-10  1.040e-09   0.265 0.791026
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 102796  on 193687  degrees of freedom
Residual deviance:  94709  on 193676  degrees of freedom
AIC: 94733
```

Summary of the logistic regression model.

Several observations can be analysed from this model iteration:

1. The median deviance residual is 0.3334 which is close to 0. In this case, deviance residuals measure how well can is_successful_payment be predicted using current dependent variables. A small deviance residual indicate a good fit.
2. The p-values for distance, price_calculation_warnings, price_too_high and price:ride_price are high, as seen from the "Pr(>|z|)". High p-values signify that these variables are not significant and does little in predicting the variable is_successful_payment. What this means for each dependent variable are:
    a. Distance does not play a role on whether a transaction is successful.
    b. Having price_calculation_warnings or price_too_high does not affect the success of the transaction.
    c. The interaction between price and ride_price does not affect the success of the transaction.
3. The Akaike Information Criterion (AIC) is very high. AIC measures how well the model fits with the data. This can be reduced by only selecting significant variables.

We can, hence, exclude some of these dependent variables through stepwise selection. This will reduce the model's AIC as a result.

```r
# check if removing variables would have a better model fit ---------------
step_model = step(model)

summary(step_model)
```

R code to perform stepwise selection and create a new model based on the selection.

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.5370   0.3048   0.3333   0.3822   8.4904

Coefficients:
                                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                                  3.210e+00  1.484e-02 216.350  < 2e-16 ***
vincenty_distance                            3.287e-02  2.190e-03  15.005  < 2e-16 ***
ride_distance                               -6.795e-05  1.339e-06 -50.735  < 2e-16 ***
price                                        6.792e-04  8.990e-05   7.555 4.18e-14 ***
ride_price                                  -6.372e-04  8.909e-05  -7.152 8.53e-13 ***
price_review_reasoncalculation_failed       -1.395e+00  4.091e-01  -3.409 0.000651 ***
price_review_reasonprice_calculation_warnings  8.856e+00  4.079e+01   0.217 0.828138
price_review_reasonprice_too_high            1.877e-01  1.192e-01   1.575 0.115334
failed_attempts                             -2.719e-01  6.305e-03 -43.134  < 2e-16 ***
vincenty_distance:ride_distance              9.728e-08  9.894e-09   9.833  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 102796  on 193687  degrees of freedom
Residual deviance:  94711  on 193678  degrees of freedom
AIC: 94731
```

Summary of the step model.

The model's AIC has decreased from 94733 to 94711, and the deviance residuals has decreased from 0.3334 to 0.3333. The dependent variable distance and price:ride_price has also been excluded. Price_review_reason is kept since cases with "calculation_failed" can provide information on predicting successful credit card transactions as shown by its low p-value.

Based on this model, an equation can be made by reading the estimate column to measure the odds of a successful credit card transaction. Since this is a logistic regression:

$$p = \frac{1}{1 + e^{-\eta}}$$

Where *p* is the probability of the credit card transaction being successful, *e* is Euler's number and η is the odds of a successful credit card transaction. η can be estimated from this formula:

$$\eta = 3.21 + 3.287 * 10^{-2} * vincenty\_distance - 6.795 * 10^{-5} * ride\_distance + 6.792 * 10^{-4} * price - 6.372 * 10^{-4} * ride\_price - 1.395 * calculation\_failed + 8.856 * price\_calculation\_warnings + 1.877 * 10^{-1} * price\_too\_high - 2.719 * 10^{-1} * failed\_attempts + 9.728 * 10^{-8} * vincenty\_distance * ride\_distance$$

Upon inspection, the equation makes sense. Farther ride distance, higher ride price, more failed attempts and a price calculation warning lowers the probability of the credit card transaction to be successful and are signs of fraud.

Finally, we can use this updated model to predict for failed credit card transactions.

```r
# predict using test data ---------------------------------------------------
testdata = data.frame(vincenty_distance = test_ride[, 11],
                      distance = test_ride[, 15],
                      ride_distance = test_ride[, 16],
                      price = test_ride[, 17],
                      ride_price = test_ride[, 18],
                      price_review_reason = test_ride[, 20],
                      failed_attempts = test_ride[, 24])

results = predict(step_model, testdata, type = "response")

# evaluate accuracy of the model --------------------------------------------
true_positive = 0 # correctly predicted failed payment
false_positive = 0 # incorrectly predicted failed payment
true_negative = 0 # correctly predicted successful payment
false_negative = 0 # incorrectly predicted successful payment

for (i in 1:nrow(testdata)){
  #  Let probabilities of less than 0.5 be categorised as failed transaction
  if (results[i] < 0.5) {
    results[i] = 0
  }
  else {
    results[i] = 1
  }

  # check whether the prediction matches the actual payment outcome
  if (results[i] == test_ride[i, "is_successful_payment"]) {
    # model predicts failed payment correctly
    if (results[i] == 0){
      true_positive = true_positive + 1
    }
    # model predicts successful payment correctly
    else{
      true_negative = true_negative + 1
    }
  }
  else {
    # model predicts failed payment for successful payments
    if (results[i] == 0){
      false_positive = false_positive + 1
    }
    # model predicts successful payment for failed payments
    else {
      false_negative = false_negative + 1
    }
  }
}
```

R code used to test the updated model. For rides with a probability of less than 0.5, we predict that the credit card transaction would fail.

Using this method, we can determine the number of correct and incorrect predictions using a confusion matrix:

		Predicted Label	
		Failed transaction	Successful transactions
Actual Label	Failed transaction	96	2546
	Successful transactions	123	45657

This model has an accuracy of (96 + 45657) / (96 + 123 + 2546 + 45657) = 94.45%. The model is also able to capture 96 failed transactions while incorrectly predicting 123 successful transactions as failed.

Similar to development rule 1, we can optimise the model by modifying the proportion of training and test sets, along with modifying the threshold of predicting a failed transaction besides 0.5.

To summarise:

Develop a regression model with significant dependent variables and flag first-time credit card transactions for rides with a probability of a successful credit card transaction less than 0.5. This model can capture and reduce several failed transactions as a result.