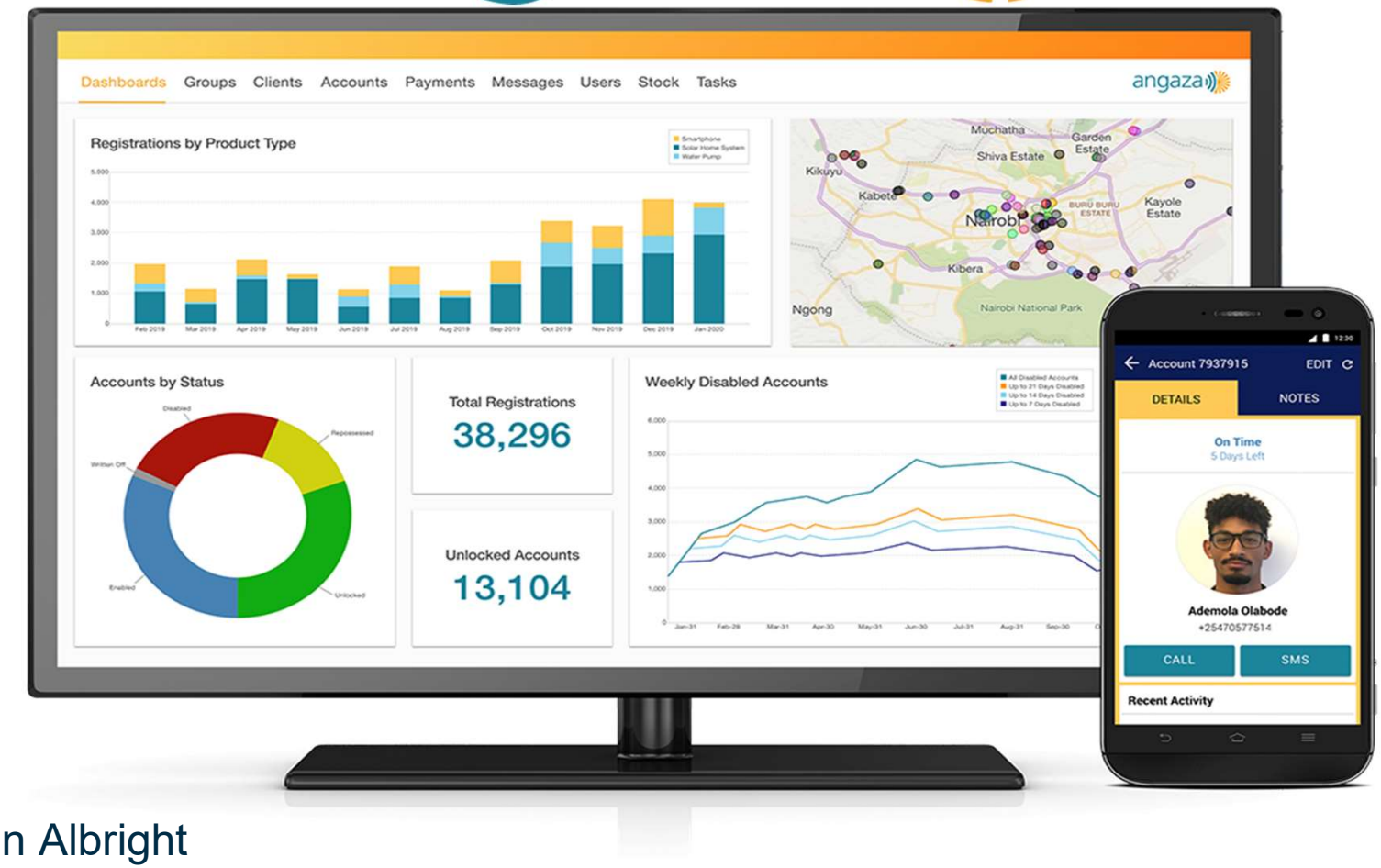
Data Analysis Assignment



By: Justin Albright

Data Analysis Assignment

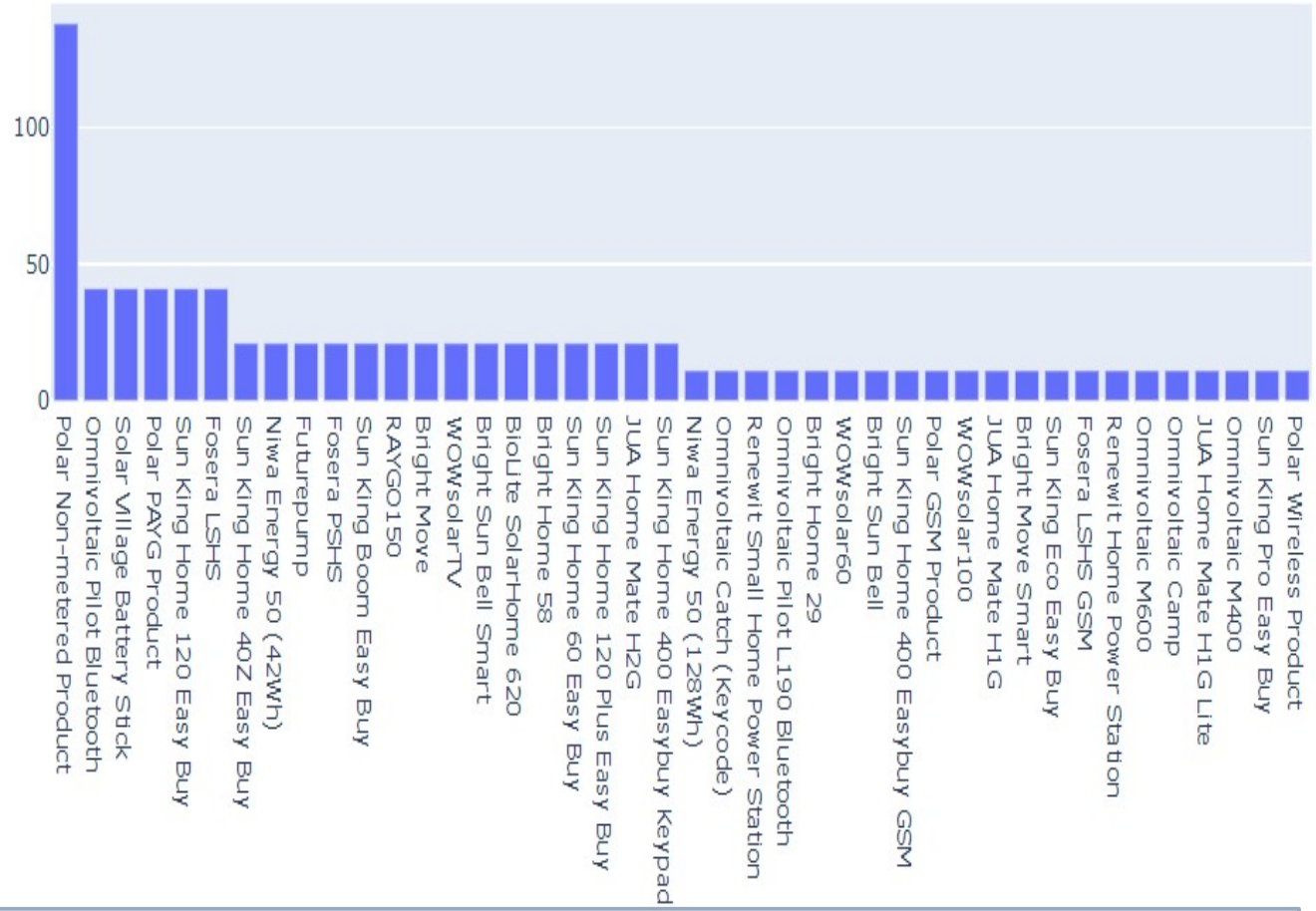
Review of Data Provided	Entity Relationships and Insights
Data Handling Tasks (4)	• Generate a bar chart visualizing the relative frequency of Products registered with Accounts. • Devise a method of generating a table of all Accounts with their State as of any arbitrary timestamp in the past and the total time they had been in that state as of the timestamp. • Devise a method of generating a table of all Accounts with their cumulative payments to date as of any arbitrary timestamp in the past. • Devise a method of generating a table of all Accounts with their "nominal" expected payments as of any arbitrary timestamp in the past. In other words, we're looking for the amount the distributor would have collected as of the test date assuming the client had paid the minimum payment instantaneously upon going to a "disabled" state until reaching the unlock price.
Exploration & Visualization Questions (2)	• How long does it typically take Account holders to repay the unlock price as a % of the nominal loan term? • What % of the total loan value is typically paid at 100% of the nominal term?
Questions for Further Analysis	How, when, and why do accounts miss timely payments and become disabled?

Overview of Entity Relationships

6 data tables, with at least one common field to join on

- 772 accounts, each belonging to one of 42 unique groups (plans)
- 42 products, each belonging to one of 42 unique groups containing one or more products. Each product is one of 14 types.
- Over 9,000 payments have been logged in the span of a year (April 2019 to March 2020) across all accounts. Over 14,000 account state transitions have occurred in relation to payments, or lack thereof.

Generate a bar chart visualizing the relative frequency of Products registered with Accounts



Explanation and Insights

- Four tables (Accounts, Group, GPA, and Products) are merged to compare three columns: account_id, product_id, and product_name.
- Total number of accounts and products, along with association to each is calculated in Python. Bar chart generated using Plotly.
- "Polar Non-metered Product" is by far the most frequent product, present in 138 of 772 accounts (18%).
- The next 5 most frequent products are each present in 41 accounts (5%), followed by the next 15 products each in 21 accounts (3%), and the last 21 products each in 11 accounts (1%) .

Devise a method of generating a table of all Accounts with their State as of any arbitrary timestamp in the past and the total time they had been in that state as of the timestamp.

account_id	to_state	time_in_state
1	ENABLED	5 days 17:59:17.428608
2	ENABLED	0 days 23:17:54.428608
3	ENABLED	0 days 21:54:10.428608
4	ENABLED	20 days 20:41:06.428608
5	DISABLED	4 days 15:12:16.954924
6	DISABLED	2 days 20:46:02.533871
7	DISABLED	8 days 05:41:42.060187
9	ENABLED	1 days 00:48:45.428608
10	ENABLED	16 days 20:21:28.428608
11	DISABLED	0 days 23:58:49.218082
12	DISABLED	5 days 05:08:54.428608
13	DISABLED	4 days 15:31:25.228608
14	DISABLED	4 days 15:47:34.028608
15	DISABLED	60 days 15:31:02.428608
16	DISABLED	6 days 00:40:53.628608

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 470 entries, 11860 to 11762
Data columns (total 3 columns):
account_id       470 non-null int64
to_state         470 non-null object
time_in_state    470 non-null timedelta64[ns]
dtypes: int64(1), object(1), timedelta64[ns](1)
memory usage: 14.7+ KB
```

Explanation and Insight

- Data from 4 columns of the AST table are utilized to produce this (ast_id, account_id, started_when, and to_state).
- An arbitrary timestamp is defined, and then used to calculate a new “time in state” column (timestamp – started_when).
- The timestamp is then used to subset and filter for the last state transition occurring before the timestamp for each account.
- Example here is timestamped January 1, 2020 at midnight. This value can be changed in the first line of the code to calculate any current state and its total time as of the timestamp.
- Note in this example only 470 accounts are shown. That is because an account is not established/does not exist until the first payment is made and thus registered.

Devise a method of generating a table of all Accounts with their cumulative payments to date as of any arbitrary timestamp in the past.

account_id	amount
1	1900
2	4998
3	4448
4	3501
5	4357
6	5740
7	3124
9	5563
10	3084
11	5826
12	4731
13	3783
14	5285
15	3224
16	5283

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 470 entries, 1 to 772
Data columns (total 1 columns):
amount    470 non-null int64
dtypes: int64(1)
memory usage: 7.3 KB
```

Explanation and Insight

- Data from 3 columns of the Payments table are utilized to produce this (account_id, amount, and effective_when).
- An arbitrary timestamp is defined, similar to the previous table, and then used to subset for the last payment occurring before the timestamp for each account.
- Cumulative amounts per account are calculated using a group by and sum function.
- Example here is timestamped January 1, 2020 at midnight. This value can be changed in the first line of the code to calculate any cumulative payments as of the date.

Devise a method of generating a table of all Accounts with their "nominal" expected payments as of any arbitrary timestamp in the past. In other words, we're looking for the amount the distributor would have collected as of the test date assuming the client had paid the minimum payment instantaneously upon going to a "disabled" state until reaching the unlock price.

account_id	expected_payments
1	950.0
2	7528.0
3	6384.0
4	2809.0
5	8100.0
6	8100.0
7	6169.5
9	8100.0
10	2237.0
11	8100.0
12	6100.0
13	6100.0
14	6100.0
15	6100.0
16	6100.0

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 470 entries, 8 to 770
Data columns (total 2 columns):
account_id           470 non-null int64
expected_payments    470 non-null float64
dtypes: float64(1), int64(1)
memory usage: 11.0 KB
```

Explanation and Insight

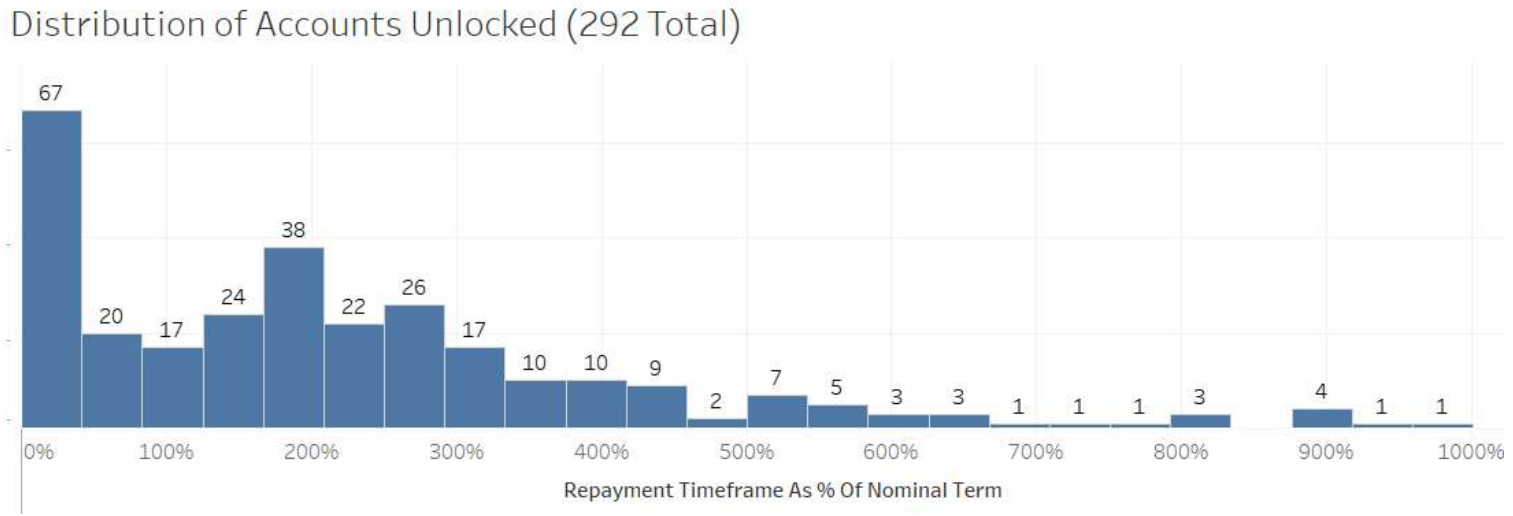
- Two tables, Accounts and Group, are merged. Data from 6 columns are utilized to produce this (account_id, registration_date, price_upfront, price_unlock, price_clock_hour, and minimum_payment).
- An arbitrary timestamp is defined, similarly as the previous tables, and then used to calculate the nominal expected payments.
- Expected payments to left calculated as follows:

1) Find total time (timestamp less registration_date) and convert to hours (total_hours).
2) "Hours post upfront period" is derived, since the down payment (price_upfront) pays for some initial hours: Hours_post_upfront_period = total_hours – (price_upfront / price_clock_hour)
3) An "hours interval" is calculated to determine how often the minimum payment would need to be made to avoid an account being disabled (minimum_payment / price_clock_hour).
4) The number of minimum payments expected is calculated as: interval_payments = hours_post_upfront_period / hours_interval
5) Finally, expected payments are quantified by: expected_payments = price_upfront + (interval_payments x minimum_payment)

- Example to the left is timestamped January 1, 2020 at midnight. This value can be changed in its line of the code to calculate any expected payments as of the date.

How long does it typically take Account holders to repay the unlock price as a % of the nominal loan term?

- On average it takes an account more than twice its nominal term, **223%**, to unlock and repay its loan.
- 292 accounts were unlocked during this year. Only 87 (30%) paid by the end of nominal term (first 2 bins).
- Over 50 accounts took 4 times the term (400%) or longer to repay.

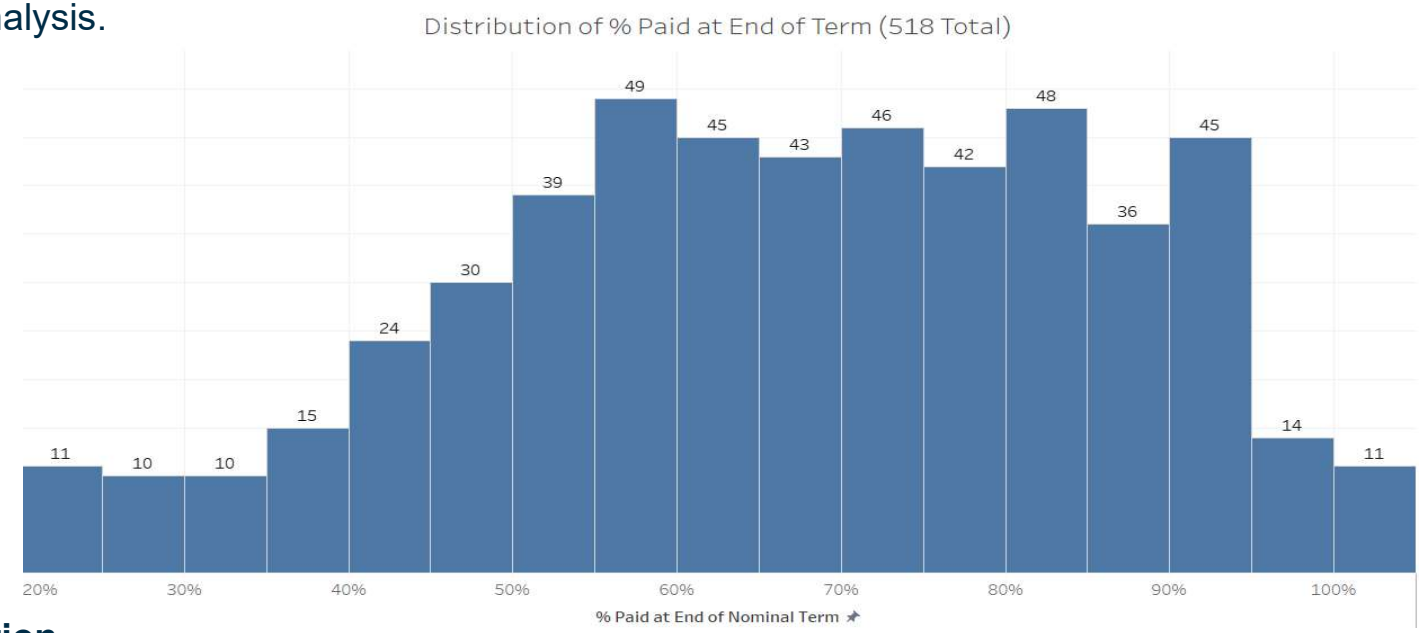


Explanation

- Three tables, Accounts, Group, and AST, are merged. Data from six columns are utilized to produce this (account_id, group_id, registration_date, to_state, price_unlock, and price_clock_hour).
- Data is sliced to return rows only with Unlocked as the state, since this occurs only when loan is paid in full.
- "Actual time to repay" is then calculated: (started_when – registration_date).
- "Nominal loan term" is calculated: (price_unlock / price_clock_hour).
- "Time to repay as % of term" for each unlocked account is then: (actual_time_to_repay / nominal_loan_term).

What % of the total loan value is typically paid at 100% of the nominal term?

- On average an account paid **67.1%** of its loan value by the end of its nominal term.
- 518 accounts reached the end of their nominal term during this year. Newer accounts are excluded from this analysis.



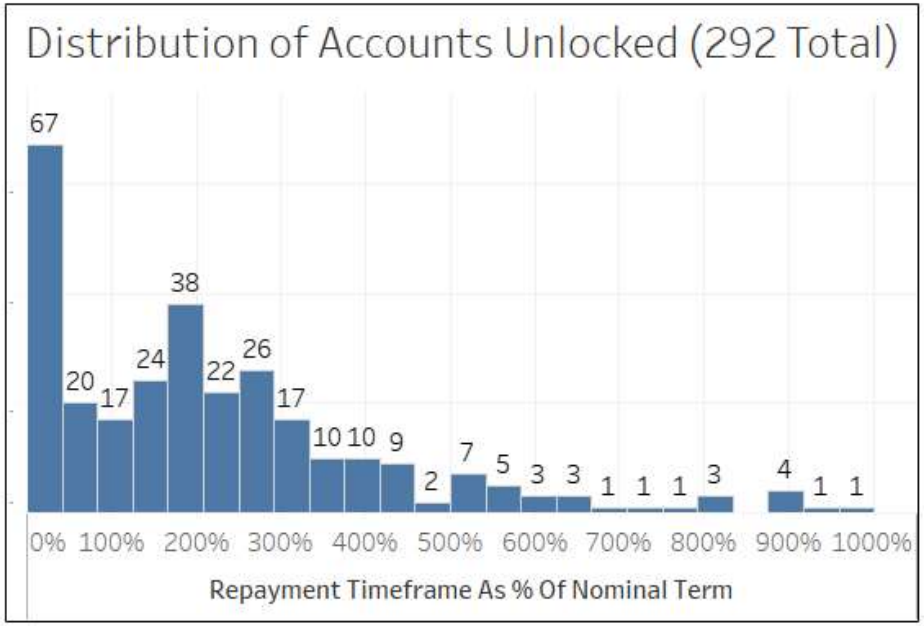
Explanation

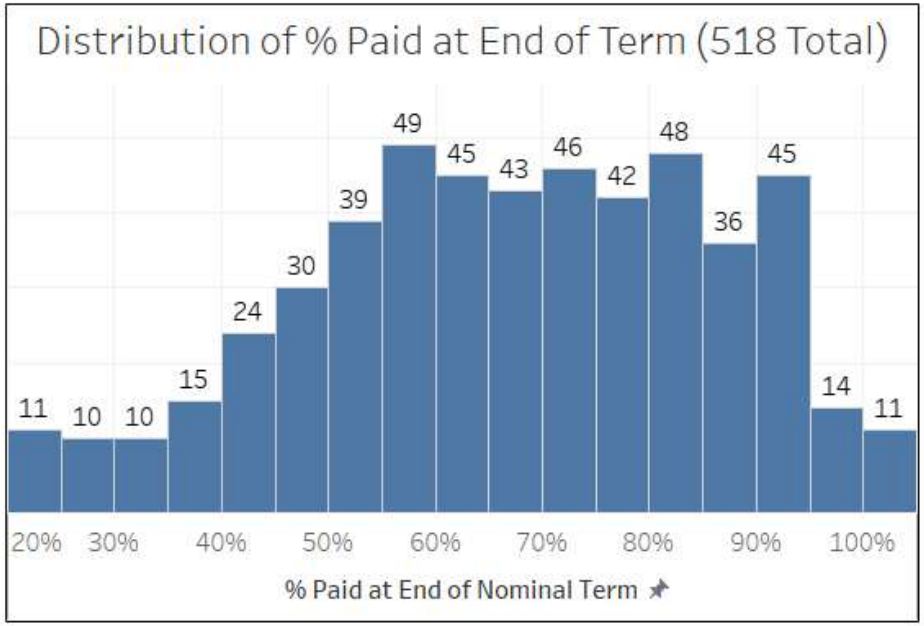
- Three tables, Accounts, Group, and Payment, are merged. Data from nine columns are utilized to produce this (account_id, group_id, registration_date, effective_date, to_state, price_unlock, price_clock_hour, price_unlock,). The created_when column from the AST table is also concatenated.

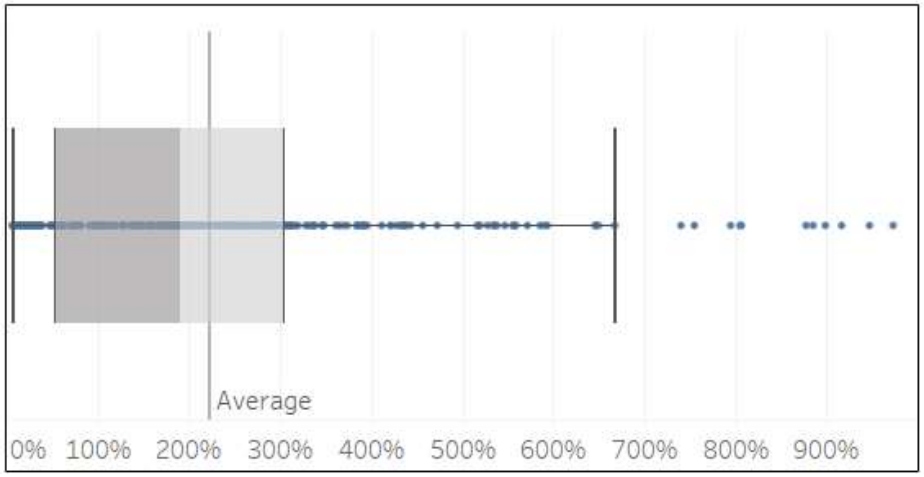
- "Nominal loan term maturity" is given as: (nominal_loan_term + registration_date)
- "Cumulative amount" is calculated using a group by and cumsum function
- "% paid" for each unlocked account is then: (cum_amount/ price_unlock).

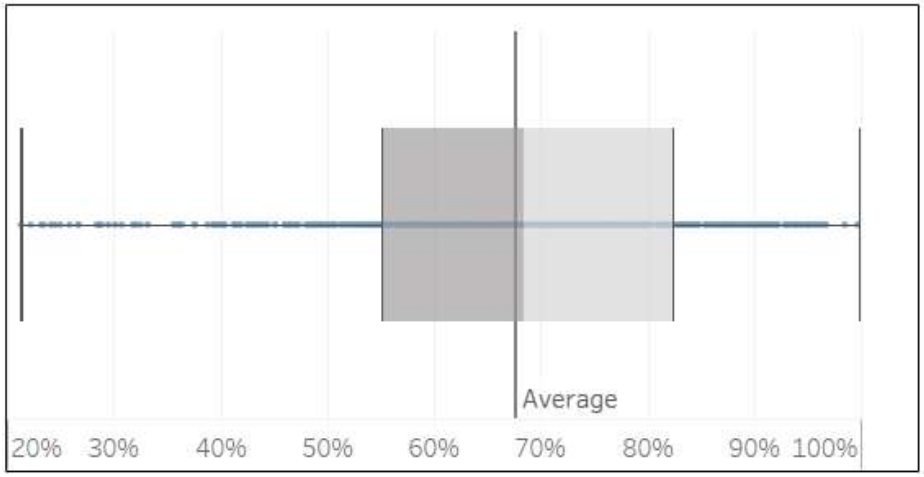
- Data is sliced to return rows where the end of the data set timeframe (created_when) occurs after the nominal loan term maturity date, and then sliced again to return only those where the effective date (last payment made) comes before the nominal loan term maturity date.

Customer Success Dashboard









Questions for Further Analysis

An assumption of the business' objective is to encourage account holders to make timely payments and pay their loans in full as originally scheduled. Ideally, every account would stay enabled, and every account would be unlocked at the end of its nominal loan term. So further analysis would be directed toward understanding how, when, and why accounts do not pay on time and become disabled.

- **Which products and groups have the highest occurrence of disable events? Which take longest to unlock their accounts?**
- **Is there any correlation between nominal loan terms and the frequency of disable events? Correlation between total (unlock) price and disable events?**
- **Is there any seasonality to payments or transitions? (i.e. If account holders live in agrarian communities, their income be tied to harvests and could have the most cash available then. So payment plans tailored to those cycles may be more practical)**

Thank You!