Rent & tenants

Comments

Input

- Have two datasets “Rent **contracts**” and “Payment **transactions**”

See *data/contracts.csv* and *data/transactions.csv*

- **Contracts** contain:
  - Tenant info
  - Duration
  - Account number
- **Transactions** contain:
  - Account number
  - Payment details
  - Amount paid

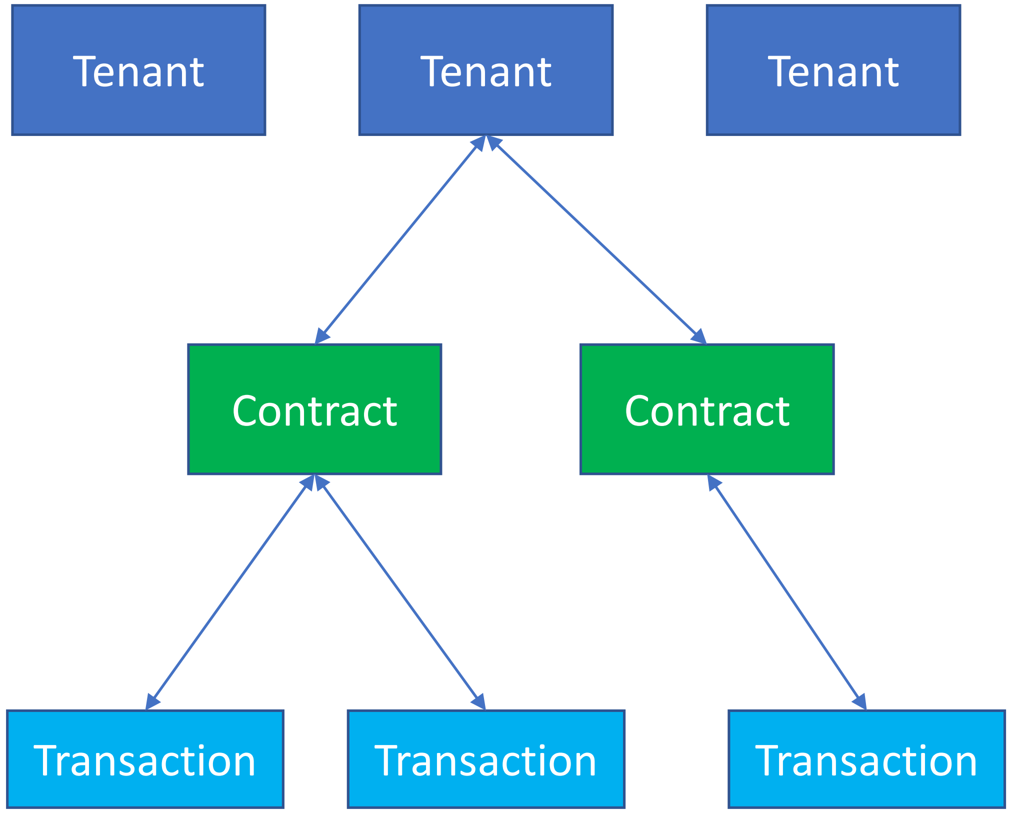
The Problem

1. Process and Cleanse the data
2. Classify tenants depending on reliability
3. **Predict** tenants reliability

Structuring and cleansing

Structure

- Each valid transaction belongs to one contract
- Each valid contract is connected to one Person/Tenant
- Represent the data as a FOREST of trees with:
  - Tenants as roots
  - Contracts on the first level
  - Transactions on the second



Transaction content

- It contain all information relevant to the actual money transfer
- Plus a link to a contract
- Data analysis show that there are 3 payment methods passible (See the code snippet)

```
class Transaction:

    class Type(Enum):
        CASH = 1
        DIRECT_DEBIT = 2
        BANK_TRANSFER = 3

    contract: Contract
    date: datetime
    amount: float
    method: Type
```

Contract content

- Contains basic contract information
- As well as a link to a Tenant
- A list of links to the transactions
- A set of transaction payments in the form:
  - Key = date
  - Value = transaction amount

```
class Contract:

    id: int
    postcode: str
    rent: float
    start: datetime
    end: datetime

    person: Person
    transactions_obj: list
    payments: dict
```

Person content

- Person data is gathered from the contracts file as a unique combination of the
  - Name and
  - Date of birth
- The ZIP code is intentionally omitted for this challenge since it is not a unique person identifier and it is not clear how the ZIP code encodes the real geo-location

```
class Person:

    first_name: str
    last_name: str
    dob: datetime
    contracts: list
```

Reading the data

1. Read Contracts file
2. create unique persons/tenants
3. Create Persons objects
4. Connect each contract to a person
5. Read Transactions files
6. Find existing contracts by id (if any)
7. Connect Transactions objects to the contracts

Preliminary observations

Persons/tenants: **9951**

Contracts with transactions: **11112**

Contracts with no transactions: **53**

Total transactions with contract: **258904**

Transactions with no contract: **2424**

Average contracts for multi-contract person **2.06**

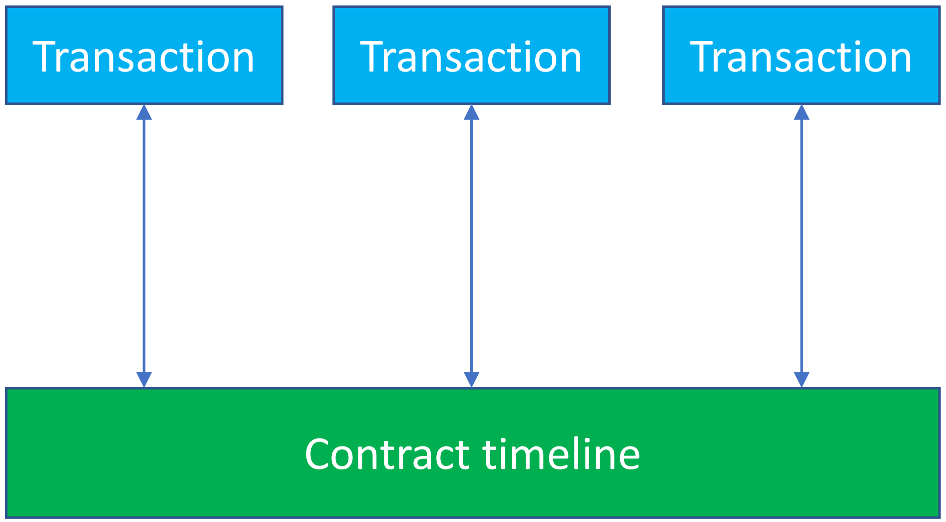
Persons with one contract: **1142**

Persons with one payment method **9607**

Persons with multiple payment methods **344**

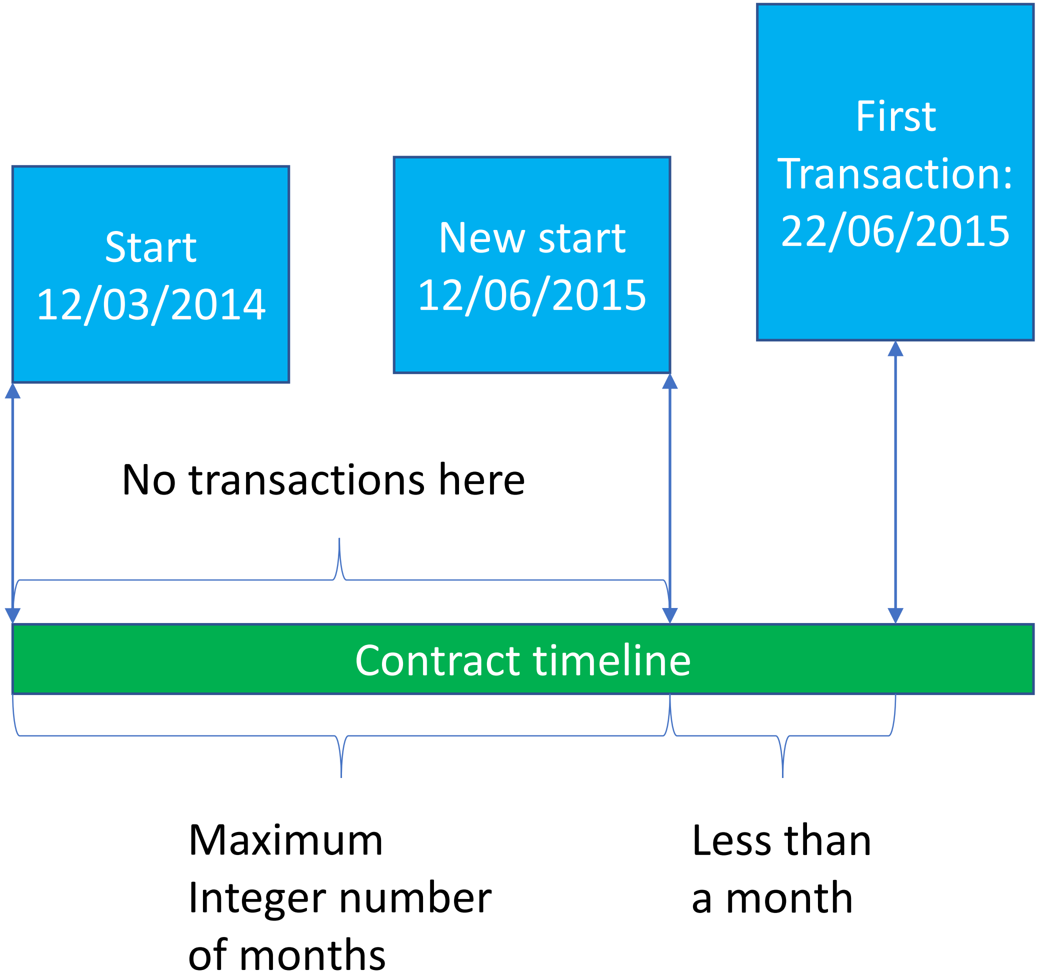
Evidences

- There are contracts without an ending date – treat those as active – the ending date will be determined the last child transaction day
- ALL the transaction from the data has their date WITHIN the parent contract duration range
- All the data in files has valid types (i.e. no NaN or number-string mismatches)

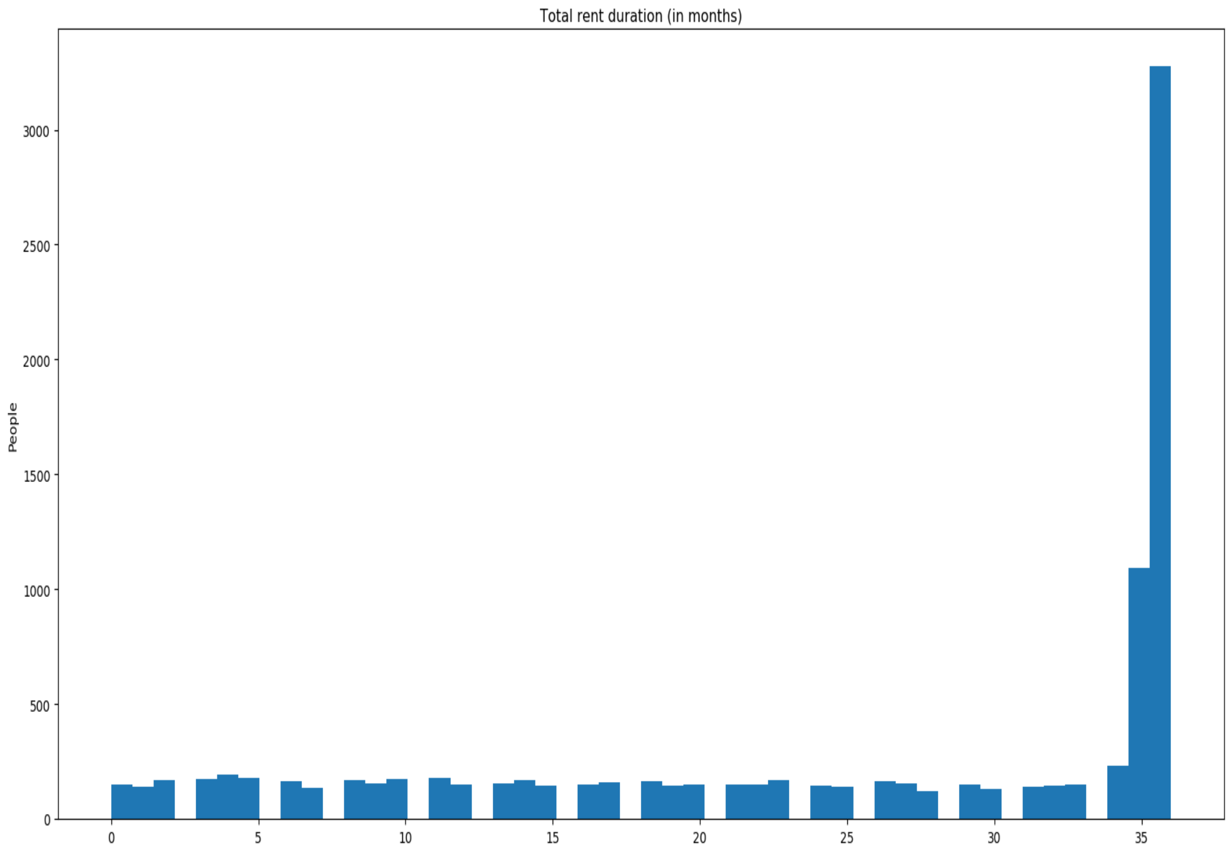


Evidences

- There are contracts that have their first transaction few years after the start – treat those as an incomplete data (and NOT as a non-payment period)
- The start for such contracts will be set-up as most recent months and day that does not exceeds the first transaction



Tenants rent duration

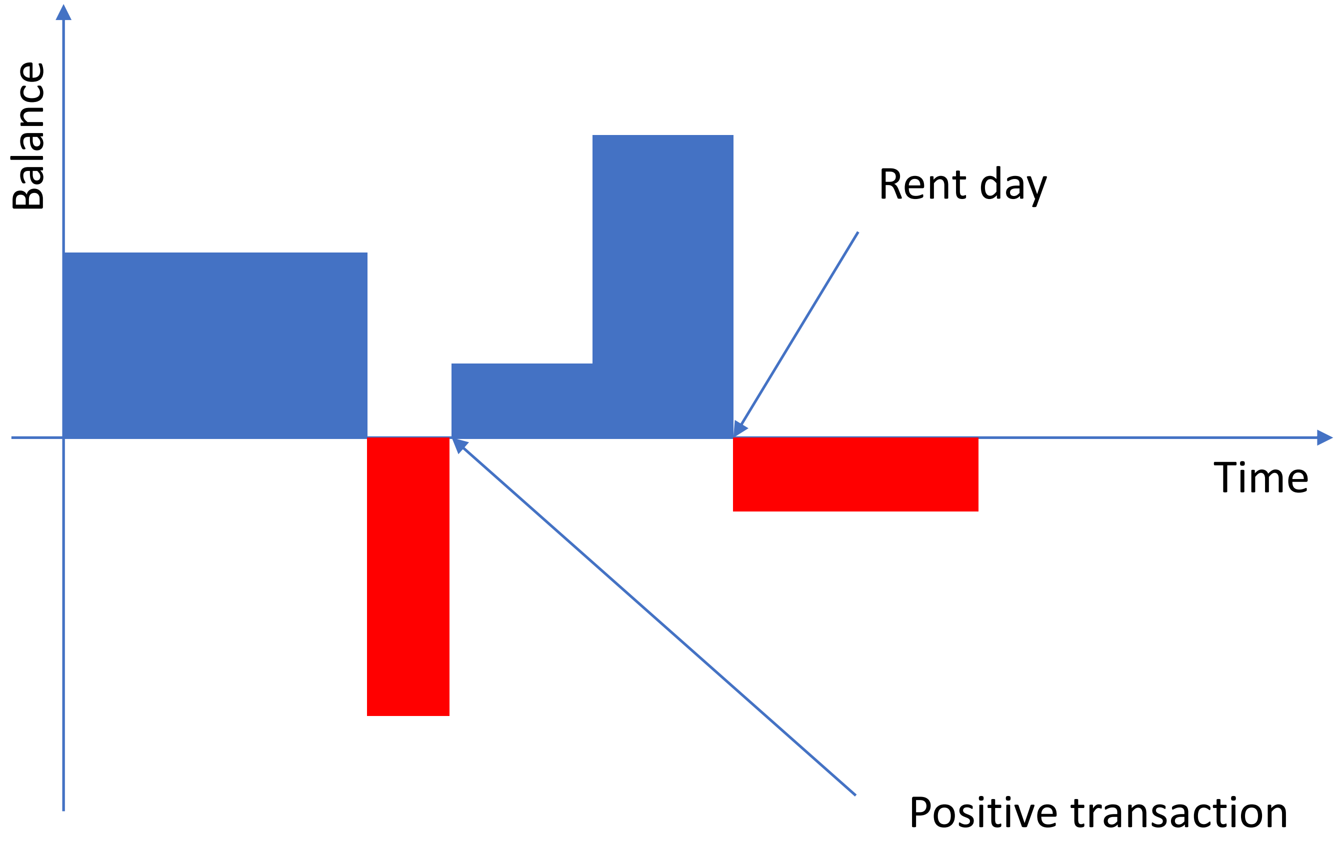


% of people with rent duration not greater than

- 3 months 6.37
- 6 months 11.77
- 12 months 21.43
- 18 months 30.94
- 24 months 40.14

Balance function

- Consider the timeline of payments: both transactions and monthly rent deductions
- The total balance can be presented as const-interval function
- $Balance = \sum transactions - \sum rent$

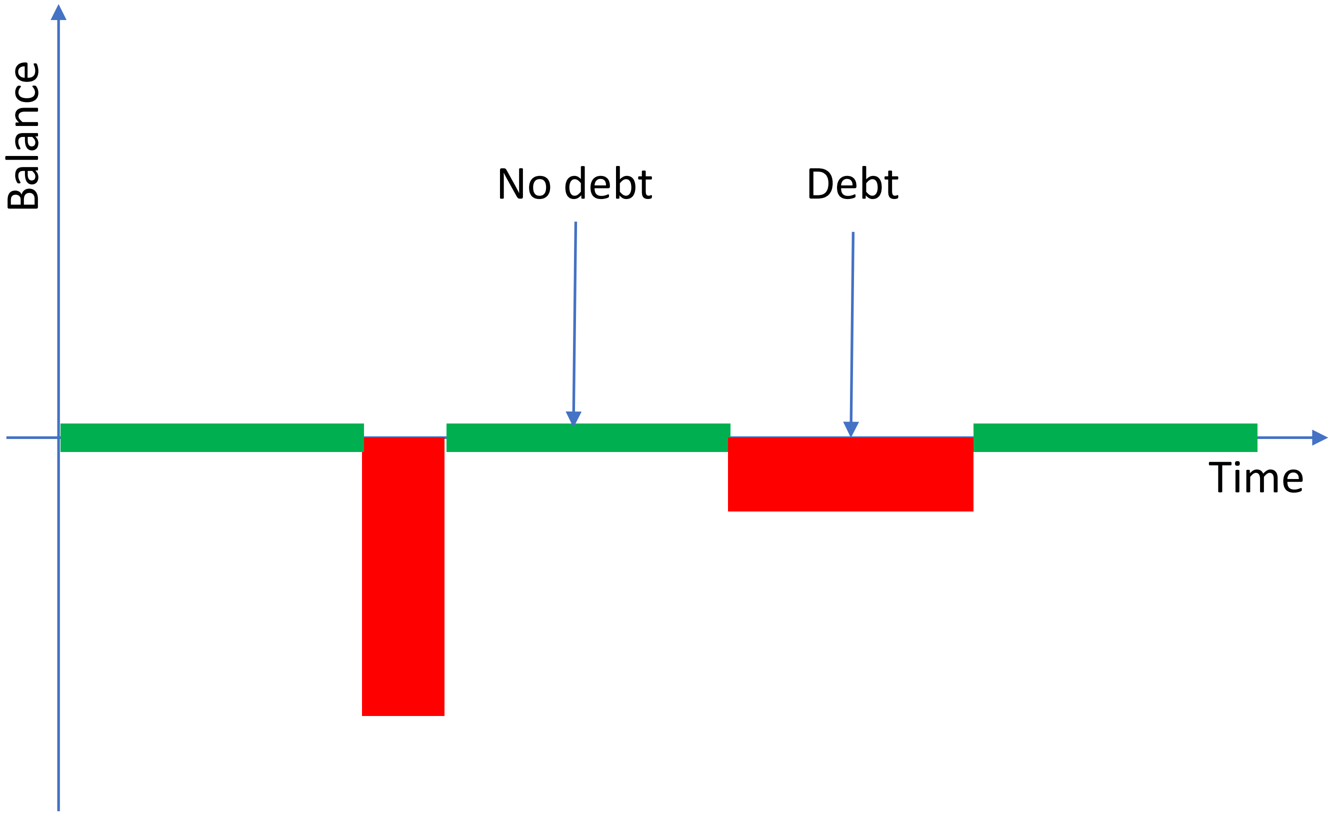


"~~Happy families~~ *Good tenants* are all alike; every ~~unhappy family~~ *bad tenant* is ~~unhappy~~ *bad* in its own way."

Leo Tolstoy

Negative balance

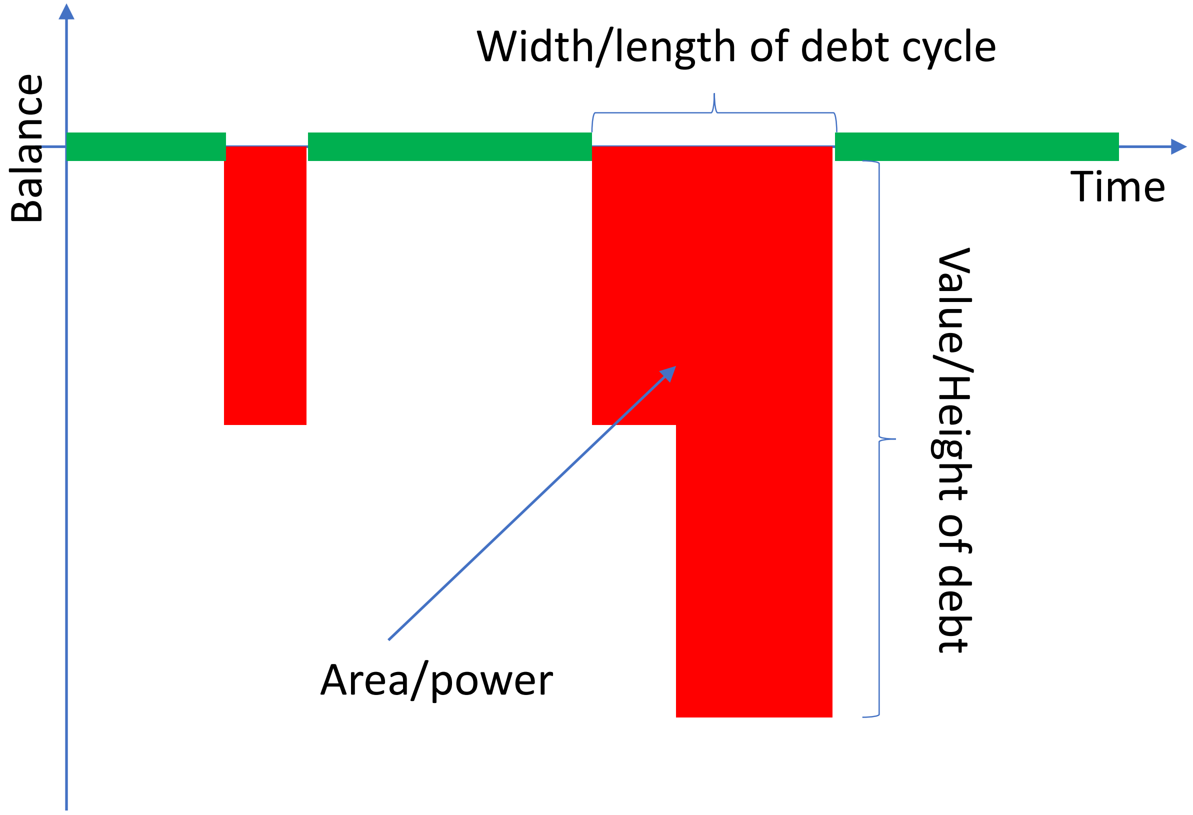
- In order to analyze how reliable is a tenant we consider the balance graph for each contract
- **only the negative part**
- Balance is calculated relative to the rent amount
- -1 = a debt in one rent amount



debt features

Features:

- Max value
- Max width
- Max area
- Number of cycles
- Average width
- Average value
- Average area
- Green/red ratio
- Etc.

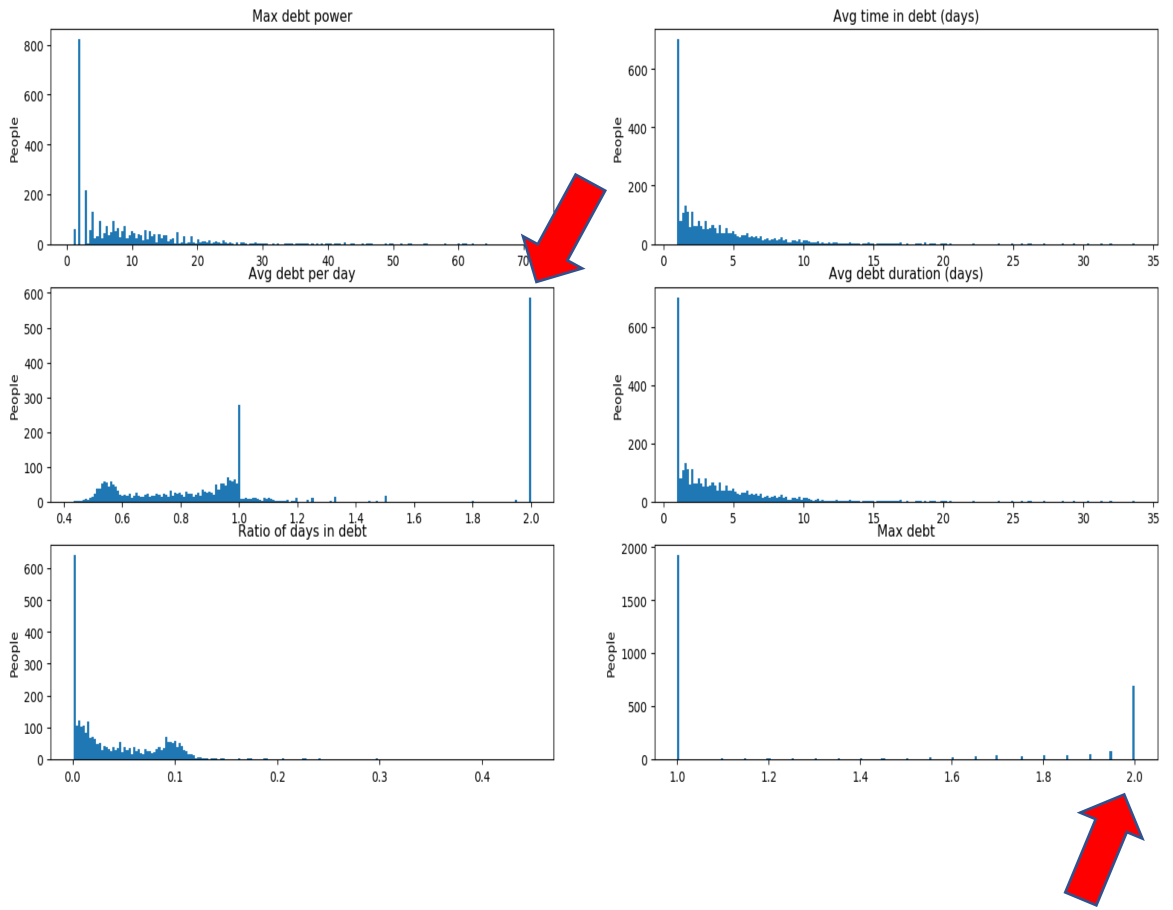


Area/power = Debt-day (similar to **kilowatt hour**)

Plot some histograms

- For every person calculate the debt history features
- Plot the histograms for non-perfect (with at least some debt in some time) tenants
- Notice the peaks (see red arrows)
- **The Deposit/Rent ratio is 3**
- The peaks are deposit returns:

$$final\ balance = payment\ - 3rent$$
$$= rent\ - 3rent = \ -2\, rent$$

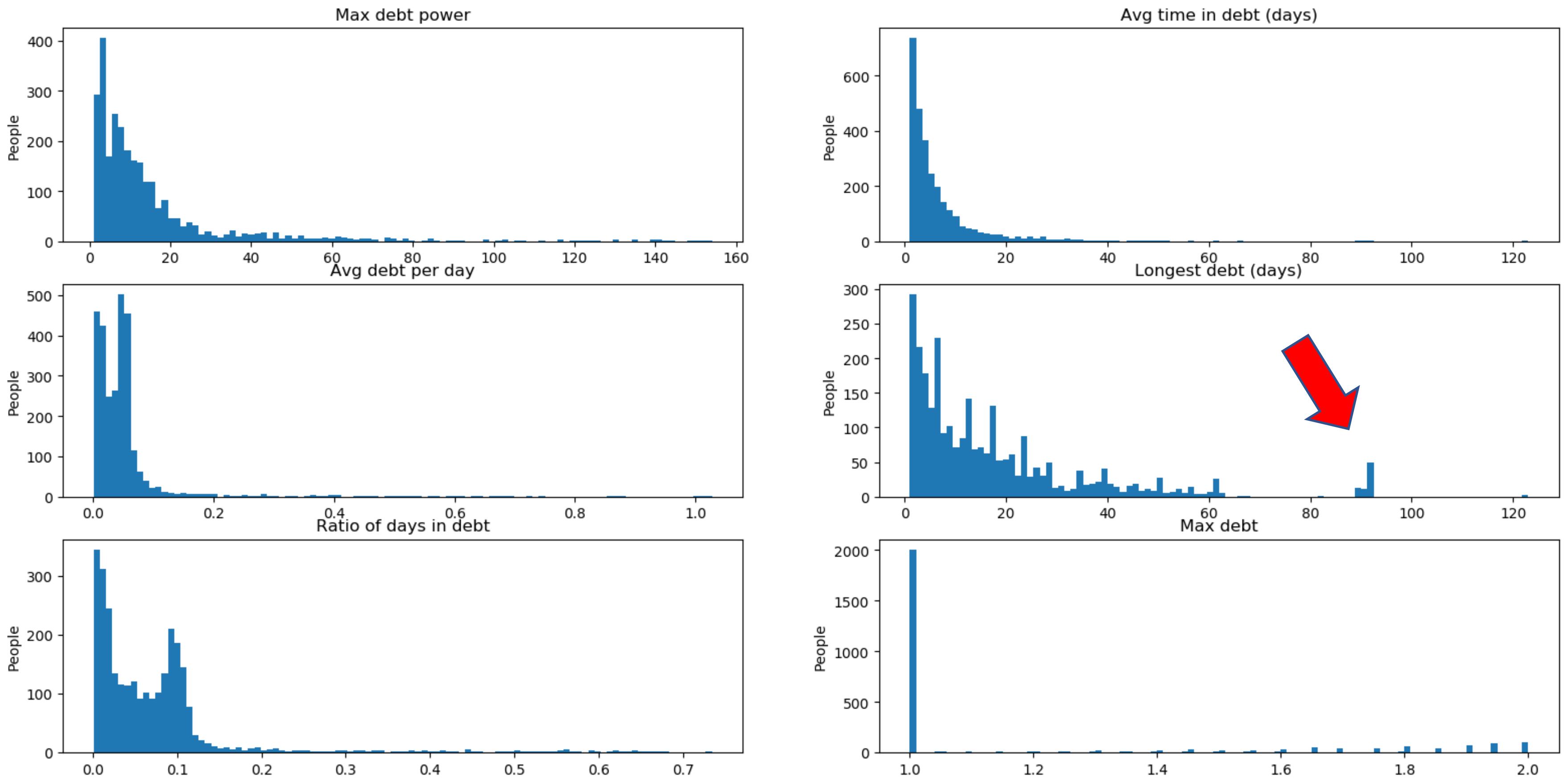


Rent return adjustment:

- If the balance becomes -2 after the last transaction on the last day of the contract – we consider is as a deposit return and do not add this data to the debt
- Contracts with deposit returns **665**

Plot some histograms (again)

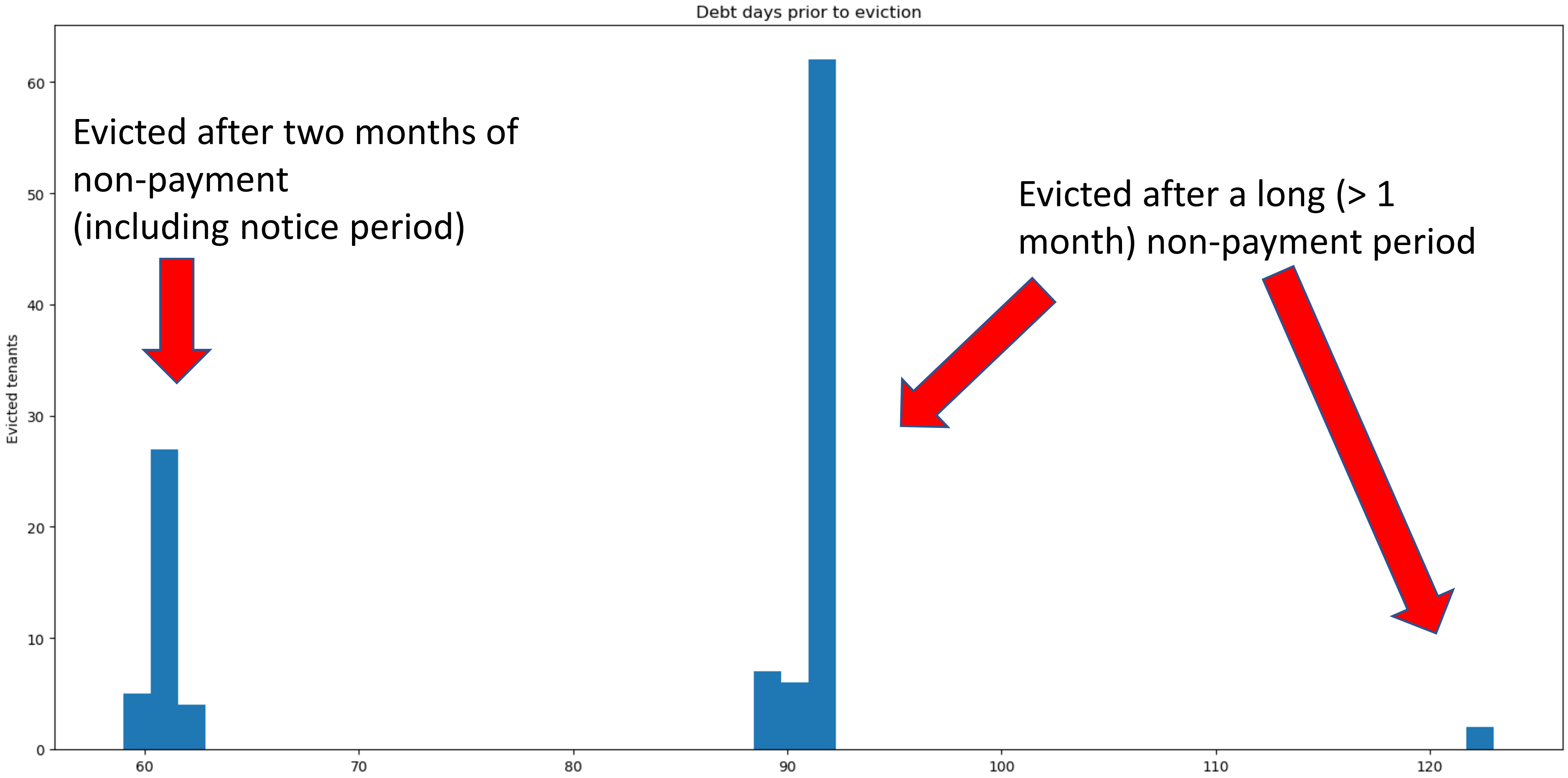
- Those are definitely evictions



Evictions:

- Assume deposit returns are already considered and removed from the data
- If the balance is negative after the last transaction on the last day of the contract – we consider is as an **eviction** indicator
- Tenants with an eviction: **113**
- Most of them were evicted after **2, 3 or 4** months of non-payment: see the histogram on the next slide

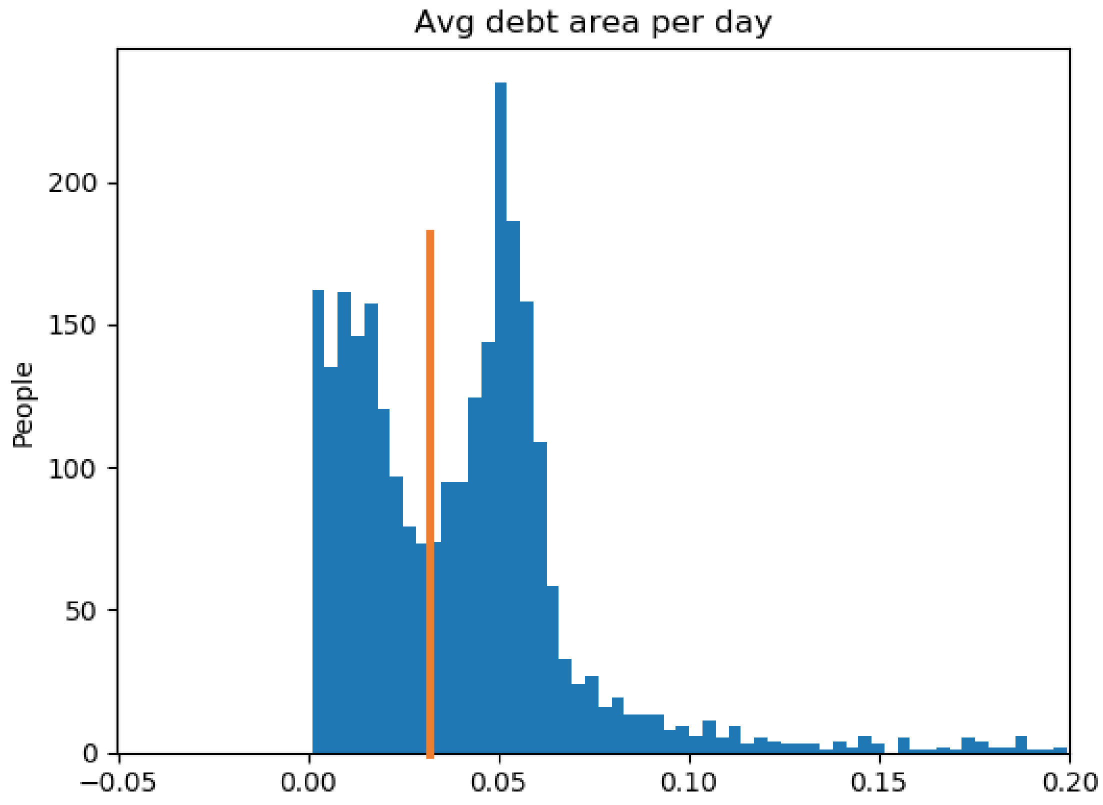
Payment delay histogram for evicted



Labels and Features

Labeling of tenants

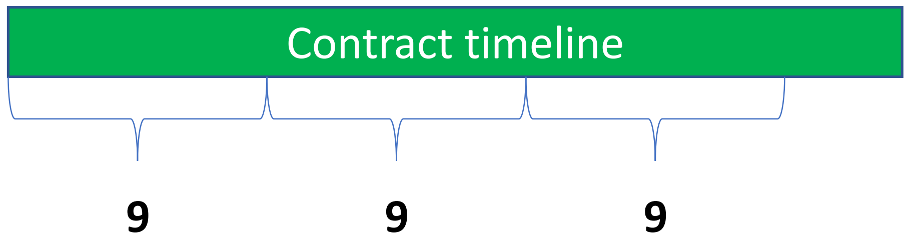
- Tenants with no debt history : GOOD
- Tenants with evictions: BAD
- Notice a distinctive split for the following histogram:
- Other tenants are either AVERAGE or BAD depending on the following split: Average debt area per day = 0.031
- Labeling window: **ALL AVAILABE TIMELINE FOR THE TENANT**



Features for regression input: **totally 10**

- ~85% of tenants have rent duration > **9 month**
- For each contract split each contract duration into integer number of **9 months** periods
- Evaluate debt features for each period
- Label with the true label for the corresponding Tenant

- Add other features:
  - Age
  - Most frequent payment method
  - Number of contracts



Regression

Input data

- Sample row of the input data (header on top):

Contracts	payment_ method	avg_area _act	avg_area _all	avg_area_ cyc	avg_act _length	ratio_active	max_active	max_area	age	label
1	Type.DIREC T_DEBIT	0.983333	0.021612	1.966667	2	0.021978	1	3	29	Label.AVERAGE

- Columns: ['contracts', 'payment_method', 'avg_area_act', 'avg_area_all', 'avg_area_cyc', 'avg_act_length', 'ratio_active', 'max_active', 'max_area', 'age', 'label’]
- GOOD, AVERAGE, BAD sizes: 14610, 4055 and 5435

Task

Create and train the classifier in to predict the LABEL based on all other features

Split data and equalizing the classes

- Train data size: 16870 (70%)
- Test data size: 7230 (30%)

- GOOD, AVERAGE, BAD ratios in the train data: 0.60 : 0.17 : 0.23
- Need to oversample to get the equal number in each of 3 classes
- Use SMOTE algorithm for the oversampling
- After SMOTE each class contains 10198 10198 10198 rows

Logistic regression (training)

- Feed the logistic regression with the data and get the fit:

1. Feature ranking output – all of the selected features are significant
2. Fit the model
3. Hope for convergence

```
model = LogisticRegression(solver='lbfgs', multi_class='multinomial', max_iter=2000)
result = model.fit(X_train, y_train)
```

Validation

Success score on the TEST data (see the split): 82.4%

Validation

Confusion matrix for the TEST (see the split) input:

	GOOD(true)	AVERAGE(true)	BAD(true)
GOOD(predicted)	4379	33	0
AVERAGE(predicted)	551	584	51
BAD(predicted)	404	232	996

Observation: the trained classifier prefers to lower the goodness of the tenant

Code:

- The code is located in the root of the provided directory and in the src folder
- Two main python files:
  - prepare_data.py – reads the data, analyzes it and creates the final input for the regression model in /data/labels_features.csv
  - regression.py – trains and validates the model based on the provided data
- Run them in the order