This chapter describes how the input file are converted into the observations files that are used for model development.

Four sets of input files are used.

- Deeds. A deed records the transfer of property or rights on the property from one party to another. The input data contains 8 deeds files `data/raw/deeds-all-columns/CAC06037F?.txt`. These are tab-separated files with a header record preceeding data records. CoreLogic claims that all deeds for Los Angeles County (that's what the 06037 means) for the years 1975 - 2009 are in the files. This project examines only the deeds for single family properties.

- Taxroll. A taxroll record records information used by the tax assessor to generate property tax bills. The input data contains 8 deeds files `data/raw/taxroll-all-column/CAC06037F?.txt`. The taxroll file is for the year 2009. It contains descriptions of the properties called parcels. For some parcels, there are no deeds; for others, many deeds.

- Census. File `data/raw/neighborhood-data/census.csv` contains data on the census tracts. Each parcel belongs to exactly one census tract. Data on the census tract are from the US govertment census carried out every 10 years. Three features are developed from the census data: average commute time, average income, and fraction of residences that are occupied by their owners. The census data augment the features that are in the taxroll files.

- Geocoding. File `data/raw/geocoing.tsv` contains the latitude and longitude of each parcel. These data come from a geomapping service and were purchased for the project. The geocoding data also augment the features that are in the taxroll file.

Two output observations sets are generated by joining the four sets of input files using the parcel identifier field APN (tax accessor's parcel number).

- Observations set 1A. This observation set mimics that one developed by Sumit Chopra in his thesis work. Sumit identified 17 (TODO: check) fields of interest in the taxroll file. Some of these fields occur sparsely in the data. A taxroll record was used in observation set 1A if and only if all the fields were present. So "1A" means observation set "1", which contains "All" taxroll fields of interest

- Observation set 2R. The "R" stands for "Restricted", because the sparsely-occuring fields in the taxroll files are never examined, so that many more taxroll records can be included.

If it were just a matter of joining the files, this chapter would be much shorter. However, several issues arose. The remained of this chapter is organized around the processing and joining of the files.

- Section 1 discusses importing the deeds into a SQLite3 database, the issues encountered, and how the issues were resolved.
- Section 2 is a similar discussion for the taxroll.
- Section 3 discusses importing the census files.
- Section 4 discusses importing the geocoding file.
- Section 5 describes how the four imported files were joined to create the transaction sets.

This document was generated from noweb and hence contains the code used to carry out all the processing steps.

@

1 Importing the Deeds

The first order of business is to create the database. SQLite3 stores all the table in one file on disk. Several iterations were required ending at version 5. The `sqlite3` command creates the data base if it does not exist.

Set sqlite3 command line options to stop processing if an error occurs (`-bail`).

2a ⟨*start-sqlite* 2a⟩≡ (2b 6a 7 15b 21 22b 41 47 49b 50 54a)

```
sqlite3 -bail ../data/v5/outputs/db.sqlite3
```

2b ⟨*db-create-database.sh* 2b⟩≡

⟨*start-sqlite* 2a⟩

The deeds_all table will hold all of the data from the deeds, as this makes loading the table relatively easy. Later, relevant subsets of the deeds are created.

A logical primary key for table deeds_all is the APN, the accessor parcel number, a ten-digit number. It's role is to be a unique identifier for the parcel. Unfortunately, it is not always present and when present, is not always correctly coded.

Fortunately, the APN is present in two fields: apn_unformatted contains the unformated number and apn_formatted inserts hyphens so that the number is in the form "9999-999-999".

After the deeds files are imported into table deeds_all, a program will be run that creates a new apn field apn_recoded that contains the "best" of the two versions of the in-file APN fields. This approach allows many more records to be used than just always using one of the APN fields. Details are given below. The primary key of the deeds table is just an integer generated sequentially by SQLite3.

The SQL command to create the table is below. In case the script is run more than once (typical during development), delete the table and re-create it. This command is found by combining the documentation for CoreLogic (CoreLogic calls the record layout "1080") with the header in the supplied files, because not all fields that are documented are actually present. Fields that are documented as present and are missing are commented out.

3 ⟨*create-table-deeds-all* 3⟩≡ (6b)

```
DROP TABLE IF EXISTS deeds_all;
CREATE TABLE deeds_all (
  /* primary key is rowid, generated by SQLite3 */

  /* key information */
  fips_code                         TEXT,
  fips_sub_code                     TEXT,
  municipality_code                 TEXT,
  apn_unformatted                   INTEGER,
  apn_formatted                     TEXT,
  apn_sequence_number               TEXT,
  --original_apn                    TEXT,
  --account_number                  TEXT,

  /* name and address */
  --owner_corporate_indicator_flag  TEXT,
  --owner_buyer_last_name           TEXT,
  --owner_buyer_first_name          TEXT,
  --owner_etal_indicator_code       TEXT /* code table ETAL */,
  --owner_co_name                   TEXT,
  --owner_ownership_rights_code     TEXT /* code table OWNSH */,
  --owner_relationship_type_coe     TEXT /* code table RELAT */,
```

```
    /* mailing address */
    mail_house_number_prefix          TEXT,
    mail_house_number                 TEXT,
    mail_house_number_suffix          TEXT,
    mail_street_direction             TEXT,
    mail_street_name                  TEXT,
    mail_mode                         TEXT,
    mail_quadrant                     TEXT,
    mail_apartment_unit_number        TEXT,
    mail_city                         TEXT,
    mail_state                        TEXT,
    mail_zip_code                     TEXT,
    mail_carrier_route                TEXT,
    mail_match_code                   TEXT /* code table MATCH */,

    /* property address */
    property_address_indicator_code   TEXT /* code table ADDRIND */,
    property_house_number_prefix      TEXT,
    property_house_number             TEXT,
    property_house_number_suffix      TEXT,
    property_street_name              TEXT,
    property_mode                     TEXT,
    property_direction                TEXT,
    property_quadrant                 TEXT,
    property_apartment_unit_number    TEXT,
    property_city                     TEXT,
    property_state                    TEXT,
    property_zip_code                 TEXT,
    property_carrier_route            TEXT,

    /* sale information */
    batch_id                          TEXT,
    batch_seq                         TEXT,
    document_year                     TEXT,
    --seller_name                     TEXT,
    sale_amount                       INTEGER,
    mortgage_amount                   INTEGER,
    sale_date                         TEXT,
    recording_date                    TEXT,
    document_type_code                TEXT /* code table DEEDC */,
    transaction_type_code             TEXT /* code table TRNTP */,
    --document_number                 TEXT,
    --book_page                       TEXT,
    --lender_last_name                TEXT,
    --lender_first_name               TEXT,
    --lender_address                  TEXT,
```

```
--lender_city                      TEXT,
--lender_state                     TEXT,
--lender_zip_code                  TEXT,
--lender_company_code              TEXT,
sale_code                          TEXT /* code table SCODE */,
--owner_buyer_middle_initial       TEXT,
--filler01                         TEXT,
multi_apn_flag_code                TEXT /* code table SLMLT */,
multi_apn_count                    TEXT,
title_company_code                 TEXT,
residential_model_indicator_flag   TEXT,
mortgage_date                      TEXT,
mortgage_loan_type_code            TEXT /* code table MTGTP */,
mortgage_deed_type_code            TEXT /* code table DOCTY */,
mortgage_term_code                 TEXT /* code table MTGTC */,
mortgage_term                      TEXT,
mortgage_due_date                  TEXT,
mortgage_assumption_amount         INTEGER,
second_mortgate_amount             INTEGER,
second_mortgage_loan_type_code     TEXT /* code table MTGTP */,
second_mortgage_deed_type_code     TEXT /* code table DOCTY */,
--prior_doc_year                   TEXT, /* no prior field is populated */
--prior_doc_number                 TEXT,
--prior_book_page                  TEXT,
--prior_document_type_code         TEXT /* code table DEEDC */,
prior_recording_date               TEXT,
prior_sales_date                   TEXT,
prior_sales_amount                 INTEGER,
prior_sales_code                   TEXT /* code table SCODE */,
prior_sales_transaction_type_code  TEXT /* code table TRNTP */,
prior_multi_apn_flag_code          TEXT /* code table SLMLT */,
prior_multi_apn_count              TEXT,
prior_mortgage_amount              INTEGER,
prior_mortgage_deed_type_code      TEXT /* code table DOCTY */,
absentee_indicator_code            TEXT /* code table ABSIND */,
property_indicator_code            TEXT /* code table PROPN */,
buiding_square_feet                INTEGER,
partial_interest_indicator_flag    TEXT,
ownership_transfer_percentage      TEXT,
universal_land_use_code            TEXT /* code tabel LUSEI */,
pri_cat_code                       TEXT /* code table PRICATCODE */,
mortgage_interest_rate_type_code   TEXT /* code table INTRT */,
--title_company_name               TEXT,
seller_carry_back_flag             TEXT,
private_party_lender_flag          TEXT,
construction_loan_flag             TEXT,
```

```
    resale_new_construction_code      TEXT /* code table RESNEW */,
    inter_family_flag                 TEXT,
    cash_mortgage_purchase_code       TEXT /* code tabel CASHMTGTP */,
    foreclosure_code                  TEXT /* code table FORECLOSURE */,
    refi_flag_code                    TEXT /* code table REFIFLAG */,
    equity_flag_code                  TEXT /* code table EQUITY */,
    census_tract                      TEXT,
    census_block_group                TEXT,
    census_block                      TEXT,
    census_block_suffix               TEXT,
    --latitude                        FLOAT,
    --longitude                       FLOAT,
    record_type_code                  TEXT /* code table RECTYPE */
    --filler02                        TEXT
  );
```

To populate `deeds_all`, create the table, then read the 8 deeds files and insert their content into the table.

6a ⟨*db-populate-deeds-all.sh* 6a⟩≡

```
⟨start-sqlite 2a⟩ < db_populate-deeds-all.cmd
```

The commands to sqlite3 needs to be in a file separate from the shell script file.

6b ⟨*db-populate-deeds-all.cmd* 6b⟩≡

```
.echo on
⟨create-table-deeds-all 3⟩
.separator "\t"
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F1.txt deeds_all
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F2.txt deeds_all
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F3.txt deeds_all
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F4.txt deeds_all
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F5.txt deeds_all
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F6.txt deeds_all
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F7.txt deeds_all
.import ../data/raw/from-laufer-2010-05-11/deeds/CAC06037F8.txt deeds_all
.tables
SELECT count(rowid) from deeds_all; -- record count
⟨deeds-all-counts (never defined)⟩
```

Running the script will create the `deeds_all` table containing about 16 million records.

@

2 Creating the Subset of `deeds_all` of interest

The next step is to create table `deeds_relevant`, a subset of the records and columns in `deeds_all`. The subset of records are the ones of interest in these experiments: deeds that reflect sales at arms length (presumedly market prices) for one single family houses transfered in full from one party to another. The subset of columns is simply the feature set of interest in the models. For the deeds, those features are WRITE ME.

Returning to selecting sales transactions for one single family residence sold in full, several columns in `deeds_all` can be combined to identify this subset. These columns are:

`[document_type_code]` the type of the transfer

`[pri_cat_code]` the primary category of the transaction

`[universal_land_use_code]` how the land is used, including the type of improvement (buildings) that are on the land

`[transaction_type_code]` the type of transaction

`[multi_apn_count` and `multi_apn_flag_code]` number of parcels covered by the deed, including the possibility of fractional parcels

How these fields were handled is discussed for each in turn before describing how the subset was created. Some exploratory data analysis was needed to figure out how the fields were populated. That requires a script and set of commands.

Below is the script. The files are compressed and must be de-compressed before using. Because I'm tight on disk space, I delete the de-compressed versions.

7 ⟨*db-count-deeds-all.sh* 7⟩≡

```
pushd ../data/raw/from-laufer-2010-05-11/deeds
tar -zxvf CAC06037F1.txt.gz
tar -zxvf CAC06037F2.txt.gz
tar -zxvf CAC06037F3.txt.gz
tar -zxvf CAC06037F4.txt.gz
tar -zxvf CAC06037F5.txt.gz
tar -zxvf CAC06037F6.txt.gz
tar -zxvf CAC06037F7.txt.gz
tar -zxvf CAC06037F8.txt.gz
popd
⟨start-sqlite 2a⟩ < db-count-deeds_all.cmd
```

```
pushd ../data/raw/from-laufer-2010-05-11/deeds
rm CAC06037F1.txt
rm CAC06037F2.txt
rm CAC06037F3.txt
rm CAC06037F4.txt
rm CAC06037F5.txt
rm CAC06037F6.txt
rm CAC06037F7.txt
rm CAC06037F8.txt
popd
```

The commands are built up in the narrative below.

8 ⟨*db-count-deeds-all.cmd* 8⟩≡ 10a ▷

```
.echo on
.tables
```

2.1 document_type_code

To identify sales transactions, one examines the document type code using code table DEEDC.

It has these values:

C construction loan

CD correction deed

F final judgment, as in a legal proceeding

G grant deed, the sale or transfer of property from one individual to another. Definitely of interest.

L liens, the owner has granted a security interest (an encumberance, possibly a mortgage) on the property to some other party

N notice of default, given by a mortgage servicing company when the mortgage has not been paid for a certain amount of time

Q quit claim, the owner terminates (quits) her interest in the property and transfers it to another person, often used to transfer property among family members or as a result of a divorce or as a result of a public auction designed to collect a tax debt

R release, removing a previous claim

S loan assignment, used by mortgage investors to transfer the loans from one to another

T deed of trust, used in California by mortgage lenders to secure mortgage loans according to http://homeguides.sfgate.com/assignment-deed-trust-definition-6750.html (accessed 2012-11-09).

U foreclosure

Z multi-county deed or open-end mortgage; an open-end mortgage allows the borrower to increase the size of the loan under certain conditions

Z nominal, perhaps a deed in which the transfer price is not the full value of the property; for example, "selling" a property for $10 in a transfer between a parent and child

If all these document type codes, I used only those for grant deeds, which number about 6.8 million) and trust deeds (which number about 5.8 million).

9 ⟨*selected-deeds-all-document-type-code* 9⟩≡ (10 12–15)

```
(document_type_code = 'G' or
 document_type_code = 'T')
```

10a ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁8 10c▷

```
SELECT count(rowid)
FROM   deeds_all
WHERE  document_type_code = 'G';

SELECT count(rowid)
FROM   deeds_all
WHERE  document_type_code = 'T';

SELECT count(rowid)
FROM   deeds_all
WHERE  ⟨selected-deeds-all-document-type-code 9⟩;
```

2.2 pri_cat_code

To identify arms-length sales transactions, I examined the `pri_cat_code` field which has code table `PRICATCODE` containing these values and counts for the deeds of interest:

A arms-length transaction, about 4.1 million

B non-arms-length transaction for a purchase, about 2.2 million

C non-arms-length transaction, not a purchase (example: a foreclosure), about 0.6 million

D non-purchase, about 5.8 million (possibly many of these are mortgage-related)

E timeshare, a few hundred

F notice of default; a mortgage has not been paid for several months, none

G assignment, none

H release, none

Of these, the only type of interest is arms-length transactions.

10b ⟨*selected-deeds-all-pri-cat-code* 10b⟩≡ (15a)

```
(pri_cat_code = 'A')
```

10c ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁10a 12a▷

```
SELECT   pri_cat_code, count(*)
FROM     deeds_all
WHERE    ⟨selected-deeds-all-document-type-code 9⟩
GROUP BY pri_cat_code;
```

2.3 universal_land_use_code

To identify sales of single family residences, the `universal_land_use_code` is used. It has code table `LUSEI` with many values including

100 residential not otherwise classified

102 townhouse or row house

103 apartment or hotel

106 apartment

109 cabin

111 cooperative

112 condominium

...

127 hotel

130 resort hotel

131 multi family 10 units plus

132 multi family 10 unites less

...

148 PUD, planned urban devlopment

...

160 rural homesite

163 SFR, single family residence, about 5.8 million

885 well/water

886 well/gas/oil

899 well/gasl/oil II

999 type unknown

I excluded from the list the huge number of commercial property types. For this work, we want only the single family residences, of which there are about 5.8 million associated with grant and trust deeds. There are about 4.3 million parcels with a empty string as their classification.

11 ⟨*selected-deeds-all-universal-land-use-code* 11⟩≡ (15a)

```
(universal_land_use_code = '163')
```

12a ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁10c 12c▷

```
SELECT   universal_land_use_code, count(*)
FROM     deeds_all
WHERE    ⟨selected-deeds-all-document-type-code 9⟩
GROUP BY universal_land_use_code;
```

2.4 transaction_type_code

Some of the deeds are not for sales and the type of sale is recorded in the `transaction_type_code` field which has code table `TRNTP` containing these values and counts for deeds of interest:

001 resale of existing property, about 4.2 million

002 refinance of the property, about 5.8 million

003 subdivision or new construction; the parcel was sold with a house on it for the first time, about 0.1 million

004 timeshare, a few hundred

006 construction loan, about 0.06 million

007 seller carryback, in which the seller provides the mortgage, about 0.1 million

009 nominal, presumedly the price does not reflect the value, about 2.4 million, a surprisingly large number to me

NULL and other values not supposed to occur, but there are a few hundred

For this modeling exercise, I am interested only in resales of existing property (001) and first-time sales (003). One slight complication is that the values in the field are supposed to have three characters, but always are either NULL or of length 1. In addition, there are a few non-numeric codes.

12b ⟨*selected-deeds-all-transaction-type-code* 12b⟩≡ (15a)

```
(transaction_type_code = '1' or
 transaction_type_code = '3')
```

12c ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁12a 13a▷

```
SELECT   transaction_type_code, count(*)
FROM     deeds_all
WHERE    ⟨selected-deeds-all-document-type-code 9⟩
GROUP BY transaction_type_code;
```

2.5 multi_apn_count and multi_apn_flag_code

To identify deeds involving a single parcel, one could examine some combination of these columns:

- multi_apn_count, defined to be the number of parcels associated with the sale
- multi_apn_flag_code, defined to indicate whether more than one parcel was associated with the sale
- ownership_transfer_percentage, defined to be the percent of ownership transfered

and/or .

To explore multi_apn_count, I ran the SQL command below.

13a ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁12c 13b▷

```
SELECT   multi_apn_count, count(*)
FROM     deeds_all
WHERE    ⟨selected-deeds-all-document-type-code 9⟩
GROUP BY multi_apn_count;
```

I found that multi_apn_count content does not follow its definition, as most often (about 12.7 million times) it is zero for grant and trust deeds. It has value one exactly 1 time. Less than 100 transactions are coded as being for multiple APNs.

To explore multi_apn_flag_code, I ran the SQL command below.

13b ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁13a 14a▷

```
SELECT   multi_apn_flag_code, count(*)
FROM     deeds_all
WHERE    ⟨selected-deeds-all-document-type-code 9⟩
GROUP BY multi_apn_flag_code;
```

multi_apn_flag_code follows the SLMLT code table. It has these values and the indicated record counts for the grant and trust deeds:

D multi or detail parcel sale, 0.15 million

M multiple parcel sale, 0.40 million

S split parcel sale, 0.02 million

X multi county or split parcel sale, ¡ 0.01 million

empty string presumedly a single parcel sale, 12.09 million

TO explore ownership_transfer_percentage, I ran the SQL command below.

14a ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁13b 14c▷

```
SELECT  ownership_transfer_percentage, count(*)
FROM    deeds_all
WHERE   ⟨selected-deeds-all-document-type-code 9⟩
GROUP BY ownership_transfer_percentage;
```

I found that ownership_transfer_percentage is non-zero only 1 time for the grant and trust deeds. VERIFY
I decided to classify as the sale of a single parcel in its entirety as those deeds with a NULL multi_apn_flag_code and no more than 1 multi_apn_count. There is a detail: the multi_apn_count fields is a TEXT field, not an INTEGER field, so one needs to test using strings (so that multi_apn_count <= 1, which as accepted by SQlite3 without an error, will fail).

14b ⟨*selected-deeds-all-single-apn* 14b⟩≡ (15a)

```
(multi_apn_flag_code = '' and
 (multi_apn_count = '0000' or multi_apn_count = '0001'))
```

14c ⟨*db-count-deeds-all.cmd* 8⟩+≡ ◁14a

```
SELECT count(*)
FROM   deeds_all
WHERE  ⟨selected-deeds-all-document-type-code 9⟩
AND    ⟨selected-deeds-all-multi-apn-count (never defined)⟩;
```

There are about 12.1 million such deeds.

2.6 Creating deeds_relevant

The goal of this section is to create table deeds_relevant to contain only deeds for arms length sales of one single family house transfered in full from one party to another can be found by *and*-ing all the conditions and only columns that are potentially relevant in subsequent work.

The potentially relevant columns are those that identify the deed, record the price of the transaction, contain features that may be useful in estimating prices, and fields that could be used to join deeds with the other input files.

To create table deeds_relevant, I ran:

15a ⟨*db-populate-deeds-relevant.cmd* 15a⟩≡

```
.echo on
DROP TABLE IF EXISTS deeds_relevant;
CREATE TABLE deeds_relevant
AS
SELECT
  /* key information */
  apn_unformatted,
  apn_formatted,
  apn_sequence_number,

  /* prices and features of the sale transaction*/
  sale_amount,
  mortgage_amount,
  sale_date,
  recording_date,
  document_type_code,
  transaction_type_code

 FROM deeds_all
 WHERE ⟨selected-deeds-all-document-type-code 9⟩
 AND   ⟨selected-deeds-all-pri-cat-code 10b⟩
 AND   ⟨selected-deeds-all-universal-land-use-code 11⟩
 AND   ⟨selected-deeds-all-transaction-type-code 12b⟩
 AND   ⟨selected-deeds-all-single-apn 14b⟩
;
SELECT COUNT(*) FROM deeds_relevant;
```

The script to run the command is

15b ⟨*db-populate-deeds-relevant.sh* 15b⟩≡

```
⟨start-sqlite 2a⟩ < db-populate-deeds_relevant.cmd
```

Running the command creates about 1.1 million deeds in table `deeds_relevant`.

3 Importing the Taxroll

The `taxroll_all` table will hold all the data from the taxroll files. Table `taxroll_relevant` will hold the taxroll records relevant to these experiments.

The taxroll file contains both an unformatted APNs and a formatted APN, either of which could serve as the primary key for the table, if it were always present. But neither is, so in implicit primary key is generated by SQLite3. A common key field is needed to join the deeds and taxroll data. How this is done is described as part of the joining process.

The SQL command to create the table is just below. Column definitions commented our are defined in the CoreLogic documentation (where the table is called "2580") and not present in the files provided.

16 ⟨*create-table-taxroll-all* 16⟩≡ (22a)

```
DROP TABLE IF EXISTS taxroll_all;
CREATE TABLE taxroll_all (
  /* primary key is rowid, generated by SQLite3 */

  /* key information */

  fips_code             TEXT,
  fips_sub_code         TEXT,
  apn_unformatted       INTEGER,  -- was TEXT
  apn_sequence_number   TEXT,

  /* parcel identification information */
  apn_formatted         TEXT,
  --original_apn        TEXT,
  --account_number      TEXT,

  /* parcel information */

  map_reference_1               TEXT,
  map_reference_2               TEXT,
  census_tract                  INTEGER,
  census_block_group            TEXT,
  census_block                  TEXT,
  census_block_suffix           TEXT,
  zoning                        TEXT,
  block_number                  TEXT,
  lot_number                    TEXT,
  range                         TEXT,
  township                      TEXT,
  section                       TEXT,
```

```
quarter_section                TEXT,
thomas_bros_map_number         TEXT,
flood_zone_community_panel_id  TEXT,
--latitude                     FLOAT,
--longitude                    FLOAT,
centroid_code                  TEXT,
homestead_exempt               TEXT,
absentee_indicator_code        TEXT /* code table ABSIND */,
tax_code_area                  TEXT,
universal_land_use_code        TEXT /* code table LUSEI */,
county_land_use_1              TEXT,
county_land_use_2              TEXT,
property_indicator_code        TEXT /* code table PROPIN */,
municipality_name              TEXT,
view                           TEXT /* code table VIEW */,
location_influence_code        TEXT /* code table LOCIN */,
number_of_buildings            TEXT,

/* subdivision information */

subdivision_tract_number       TEXT,
subdivision_plat_book          TEXT,
subdivision_plat_page          TEXT,
subdivision_name               TEXT,

/* propperty address information */

property_address_indicator_code   TEXT /* code table ADDRIND */,
property_house_number_prefix      TEXT,
property_house_number             TEXT,
property_house_number_suffix      TEXT,
property_direction                TEXT,
property_street_name              TEXT,
property_mode                     TEXT,
property_quadrant                 TEXT,
property_apartment_unit_number    TEXT,
property_city                     TEXT,
property_state                    TEXT,
property_zip_code                 TEXT,
property_carrier_route            TEXT,
property_match_code               TEXT /* code table MATCH */,

/* owner information */

owner_corporate_indicator_flag    TEXT,
--owner_name                      TEXT,
```

```
    --owner_name_2                    TEXT,
    --owner_name_paren_1              TEXT,
    --owner_name_parent_2             TEXT,
    --owner_phone                     TEXT,
    --owner_phone_opt_out_flag        TEXT,
    --owner_etal_indicator_code       TEXT /* code table ETAL */,
    --owner_ownership_rights_code     TEXT /* code table OWNSH */,
    --owner_relationship_type_coe     TEXT /* code table RELAT */,

    /* owner mail address information */

    mail_house_number_prefix        TEXT,
    mail_house_number               TEXT,
    mail_house_number_suffix        TEXT,
    mail_direction                  TEXT,
    mail_street_name                TEXT,
    mail_mode                       TEXT,
    mail_quadrant                   TEXT,
    mail_apartment_unit_number      TEXT,
    mail_city                       TEXT,
    mail_state                      TEXT,
    mail_zip_code                   TEXT,
    mail_carrier_route              TEXT,
    mail_match_code                 TEXT /* code table MATCH */,
    mail_opt_out_flag               TEXT,

    /* values information */

    total_value_calculated                      INTEGER /* land + improvement */,
    land_value_calculated                       INTEGER,
    improvement_value_calculated                INTEGER,
    total_value_calculated_indicator_flag       TEXT /* code table VALTY */,
    land_value_calculated_indicator_flag        TEXT /* code table VALTY */,
    improvement_value_calculated_indicator_flag TEXT /* code table VALTY */,
    assd_total_value                            INTEGER,
    assd_land_value                             INTEGER,
    assd_improvement_value                      INTEGER,
    mkt_total_value                             INTEGER,
    mkt_land_value                              INTEGER,
    mkt_improvement_value                       INTEGER,
    appr_total_value                            INTEGER,
    appr_land_value                             INTEGER,
    appr_improvement_value                      INTEGER,
    tax_amount                                  INTEGER,
    tax_year                                    TEXT,
```

```
    /* current sale information */

    batch_id                          TEXT,
    batch_seq                         TEXT,
    document_year                     TEXT,
    --document_number                 TEXT,
    --book_page                       TEXT,
    sales_document_type_code          TEXT /* code table DEEDC */,
    recording_date                    TEXT,
    sale_date                         TEXT,
    sale_amount                       INTEGER,
    sale_code                         TEXT /* code table SCODE */,
    --seller_name                     TEXT,
    sales_transaction_type_code       TEXT /* code table TRNTP */,
    multi_apn_flag_code               TEXT /* code table SLMLT */,
    multi_apn_code                    TEXT,
    --title_company_code              TEXT,
    --title_company_name              TEXT,
    residential_model_indicator_flag  TEXT,

    /* current trust deed information */

    first_mortgage_amount                INTEGER,
    first_mortgage_date                  TEXT,
    first_mortgage_loan_type_code        TEXT /* code table MTGTP */,
    first_mortgage_deed_type_code        TEXT /* code table DOCTY */,
    first_mortgage_term_code             TEXT /* code table MTGTC */,
    first_mortgage_term                  TEXT,
    first_mortgage_due_date              TEXT,
    first_mortgage_assumption_amount     INTEGER,
    --first_mortgage_lender_company_code TEXT,
    --first_mortgage_lender_name         TEXT,
    second_mortgage_amount               INTEGER,
    second_mortgage_loan_type_code       TEXT /* code table MTGTP */,
    second_mortgage_deed_type_code       TEXT /* code table DOCTY */,

    /* prior sale information */

    prior_sale_transaction_id          TEXT,
    prior_sale_document_year           TEXT,
    prior_sale_document_number         TEXT,
    prior_sale_book_page               TEXT,
    prior_sale_document_type_code      TEXT /* code table DEEDC */,
    prior_sale_recording_date          TEXT,
    prior_sale_date                    TEXT,
    prior_sale_amount                  INTEGER,
```

```
    prior_sale_code                     TEXT /* code table SLMLT */,
    prior_sale_transaction_type_code    TEXT /* code table TRNTP */,
    prior_sale_multi_apn_flag_code      TEXT /* code table SLMLT */,
    prior_sale_multi_apn_count          TEXT,
    prior_sale_mortgage_amount          INTEGER,
    prior_sale_deed_type_code           TEXT /* code table DOCTY */,

    /* lot/land information */

    front_footage                       TEXT,
    depth_footage                       TEXT,
    acres                               INTEGER /* coded 9999(.)9999 */,
    land_square_footage                 INTEGER,
    lot_area                            TEXT  /* textual description */,

    /* square footage information */

    universal_building_square_feet                  INTEGER,
    universal_building_square_feet_indicator_code   TEXT /* code table BLDSF */,
    building_square_feet                            INTEGER,
    living_square_feet                              INTEGER,
    ground_floor_square_feet                        INTEGER,
    gross_square_feet                               INTEGER,
    adjusted_gross_square_feet                      INTEGER,
    basement_square_feet                            INTEGER,
    garage_parking_square_feet                      INTEGER,

    /* building information */

    year_built                  TEXT,
    effective_year_built        TEXT,
    bedrooms                    TEXT,
    total_rooms                 TEXT,
    total_baths_calculated      TEXT /* encoded 999(.)99 */,
    total_baths                 TEXT /* encoded 999(.)99 */,
    full_baths                  TEXT,
    half_baths                  TEXT,
    one_quarter_baths           TEXT,
    three_quarter_baths         TEXT,
    bath_fixtures               TEXT,
    air_conditioning_code       TEXT /* code table AC */,
    basement_finish_code        TEXT /* code table BSMTF */,
    bldg_code                   TEXT /* code table BLDG */,
    bldg_improvement_code       TEXT /* code table IMPRV */,
    condition_code              TEXT /* code table COND */,
    construction_type_code      TEXT /* code table CNSTR */,
```

```
  exterior_walls_code         TEXT /* code table EXTNW */,
  fireplace_indicator_flag    TEXT /* "Y" if fireplace in building */,
  fireplace_number            TEXT,
  fireplace_type_code         TEXT /* code table FIREP */,
  foundation_code             TEXT /* code table FOUND */,
  floor_code                  TEXT /* code table FLTYP */,
  frame_code                  TEXT /* code table RFFRM */,
  garage_code                 TEXT /* code table GRGCD */,
  heating_code                TEXT /* code table HEAT */,
  mobile_home_indicator_flag  TEXT /* "Y" if a mobile home */,
  parking_spaces              TEXT,
  parking_type_code           TEXT /* code table PARKG */,
  pool_flag                   TEXT /* "Y if pool is present */,
  pool_code                   TEXT /* code table POOL */,
  quality_code                TEXT /* code table QUAL */,
  roof_cover_code             TEXT /* code table RFCOV */,
  roof_type_code              TEXT /* code table RFSHP */,
  stories_code                TEXT /* code table STORY */,
  stories_number              TEXT /* encoded 9(.)99 */,
  style_code                  TEXT /* code table STYLE */,
  units_number                TEXT /* number of apartments */,
  electric_energy_code        TEXT /* code table ELEC */,
  fuel_code                   TEXT /* code table FUEL */,
  sewer_code                  TEXT /* code table SEWER */,
  water_code                  TEXT /* code table WATER */,

  /* legal description */

  legal_1                     TEXT,
  legal_2                     TEXT,
  legal_3                     TEXT
);
```

To populate `taxroll_all`, create the table and then read the 8 taxroll files, inserting their content into the table.

21 ⟨*db-populate-taxroll-all.sh* 21⟩≡

```
⟨start-sqlite 2a⟩ < db-populate-taxroll_all.cmd
```

The command are

22a ⟨*db-populate-taxroll-all.cmd* 22a⟩≡

```
.echo on
⟨create-table-taxroll-all 16⟩
.separator "\t"
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F1.txt taxroll_all
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F2.txt taxroll_all
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F3.txt taxroll_all
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F4.txt taxroll_all
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F5.txt taxroll_all
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F6.txt taxroll_all
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F7.txt taxroll_all
.import ../data/raw/from-laufer-2010-05-11/tax/CAC06037F8.txt taxroll_all
.tables
SELECT count(*) FROM taxroll_all; -- record count
```

Running the script creates about 2.4 million taxroll records.

3.1 Exploring the Taxroll Data

The 2.4 million taxroll records include all types of parcels. We want to focus only on single family residences, which can be identified by the same field used for that purpose in the deeds, the `univeral_land_use_code` field.

22b ⟨*db-explore-taxroll-all.sh* 22b⟩≡

```
⟨start-sqlite 2a⟩ < db-explore-taxroll_all.cmd
```

22c ⟨*db-explore-taxroll-all.cmd* 22c⟩≡ 22e▷

```
.echo on
--SELECT   universal_land_use_code, count(*)
--FROM     taxroll_all
--GROUP BY universal_land_use_code;
```

I found that there are about 1.4 million parcels identified as single family residences.

22d ⟨*taxroll-all-single-family-residences* 22d⟩≡ (22–26 28 31 42)

```
universal_land_use_code = '163'
```

The tax roll is updated periodically to include an estimate of the market value of the parcels. Market values are converted using a county-specific formula to assess values. Asses values are used to generate property tax bills. What year do we have:

22e ⟨*db-explore-taxroll-all.cmd* 22c⟩+≡ ◁22c 23b▷

```
--SELECT   tax_year, count(*)
--FROM     taxroll_all
--WHERE    ⟨taxroll-all-single-family-residences 22d⟩
--GROUP BY tax_year;
```

I found that several thousand records had tax year 0000 and that all the others have year 2008.

23a ⟨*taxroll-all-proper-year* 23a⟩≡
```
tax_year = '2008'
```

There is a large number of potential features one could use. These are grouped into various categories by the CoreLogic documentation. I now explore each category.

3.1.1 Exploring Values Information

The project estimates prices of houses, but we already have an estimate of the values of the houses in 2008. These estimates are used by the tax assess as an input in determining assessed values.

23b ⟨*db-explore-taxroll-all.cmd* 22c⟩+≡ ◁22e 24▷
```
--SELECT min(mkt_total_value),
--       avg(mkt_total_value),
--       max(mkt_total_value)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

--SELECT min(mkt_land_value), avg(mkt_land_value), max(mkt_land_value)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

--SELECT min(mkt_improvement_value),
--       avg(mkt_improvement_value),
--       max(mkt_improvement_value)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;
```

I found these values were all ways zero. So I tested the similar calculated values, which are either market values or assessed values.

24 ⟨*db-explore-taxroll-all.cmd* 22c⟩+≡ ◁23b 25▷

```
--SELECT min(total_value_calculated),
--       avg(total_value_calculated),
--       max(total_value_calculated)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;
--
--SELECT min(land_value_calculated),
--      avg(land_value_calculated),
--       max(land_value_calculated)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

--SELECT min(improvement_value_calculated),
--       avg(improvement_value_calculated),
--       max(improvement_value_calculated)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;
```

I found that these values were populated:

- Total value ranges from $0 to $97 million, with an average of $337,000.
- Land value ranges from $0 to $54 million, with an average of $203,000.
- Improvement value ranges from $0 to $96 million, with an average of $134,000.

3.1.2 Exploring Current Sale Information and the Like

I ignored all the current and prior sale information, as that information is in the deeds file for all dates, not just for the most recent transaction. I ignored the mortgage information because this project is focused on the values, not the financing.

3.1.3 Exploring Lot/Land Information

The size of the lot would seem to be important. It appears twice: once as measured by `acres`, once measured by `land_square_footage`. It's pretty stable and hard to change for most properties.

25 ⟨*db-explore-taxroll-all.cmd* 22c⟩+≡ ◁24 26▷

```
--SELECT min(acres),
--       avg(acres),
--       max(acres)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

--SELECT min(land_square_footage),
--       avg(land_square_footage),
--       max(land_square_footage)
--FROM   taxroll_all
--WHERE  ⟨taxroll-all-single-family-residences 22d⟩;
```

I found that the values were populated:

- Acres ranges from 0 to 120,000 (not a typo!), with an average value of 0.6265.
- Land square footage ranges from 0 to 436,000,000 with an average value of 15,138.

An acre has 43,560 square feet. With perfect data, one could translate from acres to square footage and reach agreement, but that doesn't work here. I decided to use both values.

3.1.4 Exploring Square Footage Information

Square footage information measure the square feet inside buildings. Most of the terms are self-explanatory, but a few clarifications may help: "building square feet" means the footage in the interior of the buiding; "living square feet" is for the part of the building used for living and typically excludes garages and porches; the "universal" prefix designates the measurement used for value comparable properties for tax assessment purposes.

26 ⟨*db-explore-taxroll-all.cmd* 22c⟩+≡ ◁25 28▷

```
SELECT min(universal_building_square_feet),
       avg(universal_building_square_feet),
       max(universal_building_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(building_square_feet),
       avg(building_square_feet),
       max(building_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(living_square_feet),
       avg(living_square_feet),
       max(living_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(ground_floor_square_feet),
       avg(ground_floor_square_feet),
       max(ground_floor_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(gross_square_feet),
       avg(gross_square_feet),
```

```
       max(gross_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(adjusted_gross_square_feet),
       avg(adjusted_gross_square_feet),
       max(adjusted_gross_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(basement_square_feet),
       avg(basement_square_feet),
       max(basement_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(garage_parking_square_feet),
       avg(garage_parking_square_feet),
       max(garage_parking_square_feet)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;
```

I found these values

```
COLUMN                          MIN   AVG           MAX
universal_building_square_feet    0 1,738       354,707
building_square_feet              0 1,130     1,320,623
living_square_feet                0 1,735        57,660
ground_floor_square_feet          0     0             0
gross_square_feet                 0     1.7      25,443
adjusted_gross_square_feet        0     0             0
basement_square_feet              0     7.1   1,110,101
garage_parking_square_feet        0     0.02      4,368
```

I decided that the last five of these fields would not be used as features because their averages are so low that most of the values must be 0. I decided to use the first 3 fields as features.

3.1.5 Exploring Building Information

We come to a long list of features called "building information" in the file. The are in two main groups:

- One group is comprised of numerical information fields, including `effective_year_built`, `bedrooms`, `bathrooms`, `fireplace_number` and similar.
- The other group consists of codes including `air_conditioning_code` (the type of air conditioning) and `water_code` (the type of water service: public, well, cistern).

To explore the numerical building information fields, I determined min, average, and max values.

28 ⟨*db-explore-taxroll-all.cmd* 22c⟩+≡ ◁26 31▷

```
SELECT min(year_built),
       avg(year_built),
       max(year_built)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(effective_year_built),
       avg(effective_year_built),
       max(effective_year_built)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(bedrooms),
       avg(bedrooms),
       max(bedrooms)
FROM   taxroll_all
```

```
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(total_rooms),
       avg(total_rooms),
       max(total_rooms)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(total_baths_calculated),
       avg(total_baths_calculated),
       max(total_baths_calculated)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(total_baths),
       avg(total_baths),
       max(total_baths)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(full_baths),
       avg(full_baths),
       max(full_baths)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(half_baths),
       avg(half_baths),
       max(half_baths)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(one_quarter_baths),
       avg(one_quarter_baths),
       max(one_quarter_baths)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(three_quarter_baths),
       avg(three_quarter_baths),
       max(three_quarter_baths)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(bath_fixtures),
       avg(bath_fixtures),
```

```
       max(bath_fixtures)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(fireplace_number),
       avg(fireplace_number),
       max(fireplace_number)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(parking_spaces),
       avg(parking_spaces),
       max(parking_spaces)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(stories_number),
       avg(stories_number),
       max(stories_number)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;

SELECT min(units_number),
       avg(units_number),
       max(units_number)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩;
```

I found these values

```
COLUMN                  MIN    AVG   MAX
year_build                0 1948.3  2009
effective_year_built      0 1951.5  2009
bedrooms                  0    3.2    99
total_rooms               0    4.8    84
total_baths_calculated    0    2.09   99
total_baths               0    2.09   99
full_baths                0    2.09   99
half_baths                0    0       0
one_quarter_baths         0    0       0
three_quarter_baths       0    0       0
bath_fixtures             0    0       0
fireplace_number          0    0.59    9
parking_spaces            0    1.56  800
stories_number            0    0.96    6
units_number              0    0.997 843
```

Some observations:

- Three fields appear to have been set identically: `totao_baths_calculated` .. `full_baths`.
- Several fields appear to not be populated: `half_baths` ... `bath_fixtures`.
- Several fields have unusually high maximum values: `bedrooms`, `total_rooms`, number of bath rooms fields, `parking_spaces`, `units_number`. These taxroll records are probably not for single family houses.

YANN: Why not simply keep all the buiding information features, dropping only those fields that seem to be redundant with each other (such as the number of bathroom fields above)? Picking and choosing seems to reflect my biases about what is relevant and the quality of the input file. In particular, who not keep the all-zero fields?

To explore the coded building information fields, I determined how frequently each code was used.

31 ⟨*db-explore-taxroll-all.cmd* 22c⟩+≡ ◁28

```
SELECT air_conditioning_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY air_conditioning_code;

SELECT basement_finish_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY basement_finish_code;
```

```
SELECT bldg_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY bldg_code;

SELECT bldg_improvement_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY bldg_improvement_code;

SELECT condition_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY condition_code;

SELECT construction_type_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY construction_type_code;

SELECT exterior_walls_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY exterior_walls_code;

SELECT fireplace_type_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY fireplace_type_code;

SELECT foundation_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY foundation_code;

SELECT floor_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY floor_code;

SELECT frame_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY frame_code;
```

```
SELECT garage_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY garage_code;

SELECT heating_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY heating_code;

SELECT parking_type_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY parking_type_code;

SELECT pool_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY pool_code;

SELECT quality_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY quality_code;

SELECT roof_cover_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY roof_cover_code;

SELECT roof_type_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY roof_type_code;

SELECT stories_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY stories_code;

SELECT style_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY style_code;

SELECT electric_energy_code, count(*)
```

```
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY electric_energy_code;

SELECT fuel_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY fuel_code;

SELECT water_code, count(*)
FROM   taxroll_all
WHERE  ⟨taxroll-all-single-family-residences 22d⟩
GROUP BY water_code;
```

Here's the stdout

```
SELECT air_conditioning_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY air_conditioning_code;
|1044959
001|6
ACE|261527
ACH|1
AEV|63197
AHT|20
AWA|40387
AWI|13519

SELECT basement_finish_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY basement_finish_code;
|1423616

SELECT bldg_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY bldg_code;
|1423616

SELECT bldg_improvement_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY bldg_improvement_code;
|1423616

SELECT condition_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY condition_code;
|949863
AVE|150284
EXC|33311
FAI|14158
GOO|273704
POO|2296

SELECT construction_type_code, count(*)
FROM   taxroll_all
```

```
WHERE  universal_land_use_code = '163'
GROUP BY construction_type_code;
|1178213
ADB|4
BRK|170
CCB|216
CRE|4323
CUS|118
DOM|1
FRM|239357
LOG|1
MAN|190
MAS|981
MET|9
SRO|32
STE|1

SELECT exterior_walls_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY exterior_walls_code;
|268790
ASB|628
BLO|2547
BRO|710
BRV|47
CNB|7
COM|89
GLA|2
LOG|15
MET|244
ROC|1
SGS|123464
SHI|756
SNW|36
STU|1024213
VIN|11
WOO|2056

SELECT fireplace_type_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY fireplace_type_code;
|645412
001|778204
```

```
SELECT foundation_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY foundation_code;
|275252
001|986
CRE|180
MSN|4
PIR|29258
RAS|753485
SLB|353673
UCR|10778

SELECT floor_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY floor_code;
|1094702
A00|328914

SELECT frame_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY frame_code;
|1423610
00S|2
00T|4

SELECT garage_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY garage_code;
|297939
001|2922
110|195
120|12793
450|232412
910|8
920|90935
A00|786412

SELECT heating_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY heating_code;
|184428
```

```
001|398065
00S|1938
BBE|517
CL0|357072
FA0|257391
FF0|85732
GR0|9681
HP0|178
HW0|74
RD0|1269
SP0|175
ST0|44
SV0|30
WF0|127022

SELECT parking_type_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY parking_type_code;
|273674
001|2
110|57
120|10907
140|71470
450|205516
910|2
920|72902
A00|786412
ASP|3
CVP|1
OSP|280
PAP|58
SBP|1
UU0|5
Z00|2326

SELECT pool_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY pool_code;
|1173334
001|233552
002|1
00H|556
00O|18
0V0|5
```

```
300|6689
E00|19
I00|10
M00|14
S00|9418

SELECT quality_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY quality_code;
|968533
QAV|312326
QEX|2220
QFA|17347
QGO|120487
QLU|2212
QPO|491

SELECT roof_cover_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY roof_cover_code;
|264680
003|723
010|129
014|70543
015|504031
024|117
025|533
027|62598
029|174812
030|174773
151|170677

SELECT roof_type_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY roof_type_code;
|307455
F00|68606
G00|693572
I00|353983

SELECT stories_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
```

```
GROUP BY stories_code;
|1423616

SELECT style_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY style_code;
|265482
001|14719
AFR|4
BUN|7053
CAP|136
CNT|55814
COL|5236
CON|988183
FRE|1976
GRG|16
MED|1718
MOD|10919
RAN|5354
RRA|69
SPA|57055
TUD|9744
VIC|138

SELECT electric_energy_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY electric_energy_code;
|1423616

SELECT fuel_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY fuel_code;
|1423616

SELECT water_code, count(*)
FROM   taxroll_all
WHERE  universal_land_use_code = '163'
GROUP BY water_code;
|1062282
WCI|14
WCO|1124
WPU|360196
```

Yann: Why not take them all?

3.2 Creating `taxroll_relevant`

This section creates table `taxroll_relevant` by select the single-family residence records in table `taxroll_all` and columns from it that are potentially relevant as features or in joining other datasets.

41 ⟨*db-populate-taxroll-relevant.sh* 41⟩≡
⟨*start-sqlite* 2a⟩ `< db-populate-taxroll-relevant.cmd`

42 ⟨*db-populate-taxroll-relevant.cmd* 42⟩≡

```
.echo on
DROP TABLE IF EXISTS taxroll_relevant;
CREATE TABLE taxroll_relevant
AS
SELECT

  /* key and parcel identification information */
  /* and information to join with deeds and geocoding data */

  fips_code,
  fips_sub_code,
  apn_unformatted,
  apn_sequence_number,
  apn_formatted,

  /* potential features: parcel information */
  /* and information to join with census data */

  map_reference_1,
  map_reference_2,
  census_tract,
  census_block_group,
  census_block,
  census_block_suffix,
  zoning,
  block_number,
  lot_number,
  range,
  township,
  section,
  quarter_section,
  thomas_bros_map_number,
  flood_zone_community_panel_id,
  centroid_code,
  homestead_exempt,
  absentee_indicator_code        /* code table ABSIND */,
  tax_code_area,
  universal_land_use_code        /* code table LUSEI */,
  county_land_use_1,
  county_land_use_2,
  property_indicator_code        /* code table PROPIN */,
  municipality_name,
  view                           /* code table VIEW */,
  location_influence_code        /* code table LOCIN */,
  number_of_buildings,
```

```
/* potential features: propperty address information */

property_address_indicator_code   /* code table ADDRIND */,
property_house_number_prefix,
property_house_number,
property_house_number_suffix,
property_direction,
property_street_name,
property_mode,
property_quadrant,
property_apartment_unit_number,
property_city,
property_state,
property_zip_code,
property_carrier_route,
property_match_code                /* code table MATCH */,

/* potential features: owner information */
owner_corporate_indicator_flag,

/* potential feature: values information */
total_value_calculated                        /* land + improvement */,
land_value_calculated,
improvement_value_calculated,
tax_amount,
tax_year,

/* potential featurs: subdivision information */

subdivision_tract_number,
subdivision_plat_book,
subdivision_plat_page,
subdivision_name,

/* potential features: lot/land information */

front_footage,
depth_footage,
acres                                 /* coded 9999(.)9999 */,
land_square_footage,
lot_area                              /* textual description */,

/* potential features: square footage information */

universal_building_square_feet,
```

```
universal_building_square_feet_indicator_code   /* code table BLDSF */,
building_square_feet,
living_square_feet,
ground_floor_square_feet,
gross_square_feet,
adjusted_gross_square_feet,
basement_square_feet,
garage_parking_square_feet,

/* potential features: building information */

year_built,
effective_year_built,
bedrooms,
total_rooms,
total_baths_calculated       /* encoded 999(.)99 */,
total_baths                  /* encoded 999(.)99 */,
full_baths,
half_baths,
one_quarter_baths,
three_quarter_baths,
bath_fixtures,
air_conditioning_code        /* code table AC */,
basement_finish_code         /* code table BSMTF */,
bldg_code                    /* code table BLDG */,
bldg_improvement_code        /* code table IMPRV */,
condition_code               /* code table COND */,
construction_type_code       /* code table CNSTR */,
exterior_walls_code          /* code table EXTNW */,
fireplace_indicator_flag     /* "Y" if fireplace in building */,
fireplace_number,
fireplace_type_code          /* code table FIREP */,
foundation_code              /* code table FOUND */,
floor_code                   /* code table FLTYP */,
frame_code                   /* code table RFFRM */,
garage_code                  /* code table GRGCD */,
heating_code                 /* code table HEAT */,
mobile_home_indicator_flag   /* "Y" if a mobile home */,
parking_spaces,
parking_type_code            /* code table PARKG */,
pool_flag                    /* "Y if pool is present */,
pool_code                    /* code table POOL */,
quality_code                 /* code table QUAL */,
roof_cover_code              /* code table RFCOV */,
roof_type_code               /* code table RFSHP */,
stories_code                 /* code table STORY */,
```

```
  stories_number              /* encoded 9(.)99 */,
  style_code                  /* code table STYLE */,
  units_number                /* number of apartments */,
  electric_energy_code        /* code table ELEC */,
  fuel_code                   /* code table FUEL */,
  sewer_code                  /* code table SEWER */,
  water_code                  /* code table WATER */

FROM taxroll_all
WHERE ⟨taxroll-all-single-family-residences 22d⟩
;
SELECT COUNT(*) FROM taxroll_relevant;
```

Table taxroll_relevant has about 1.4 million records.

4 Importing the Census Data

The census table will contain all the data from the census file for year 1999. The primary key for the record is the column geo_id2 which contains the FIPS code (county number) concatenated with the census tract number. Later, this field must be split so that the census tract number can be used to join these records with the taxroll records.

46 ⟨*create-table-census* 46⟩≡ (47b)

```
DROP TABLE IF EXISTS census;
CREATE TABLE census (
  /* identification */

  geo_id        TEXT,
  geo_id2       INTEGER PRIMARY KEY,     -- fips code + census tract number
  geo_sumlevel  INTEGER,
  geo_name      TEXT,

  /* data on workers 16 years of age and over */

  workers_total             INTEGER,
  workers_not_work_at_home  INTEGER,

  /* travel times for workers not working at home in minutes */

  travel_less_5      INTEGER,
  travel_5_to_9      INTEGER,
  travel_10_to_14    INTEGER,
  travel_15_to_19    INTEGER,
  travel_20_to_24    INTEGER,
  travel_25_to_29    INTEGER,
  travel_30_to_34    INTEGER,
  travel_35_to_39    INTEGER,
  travel_40_to_44    INTEGER,
  travel_45_to_59    INTEGER,
  travel_60_to_89    INTEGER,
  travel_90_or_more  INTEGER,

  workers_work_at_home  INTEGER,

  /* statistics on households and families in 1999 */
  /* a household may or may not be a family */

  household_median_income            INTEGER,
```

```
  family_household_median_income     INTEGER,
  nonfamily_household_median_income  INTEGER,
  nonfamily_median_income            INTEGER,

  /* housing units and occupancy */

  owner_occupied       INTEGER,
  renter_occupied      INTEGER,
  median_number_rooms  INTEGER
);
```

To populate table `census`, I ran the script below. Because some columns have INTEGER types and the SQLite import statement reads all the input lines, it's necessary to drop the first two lines of the data file, as these each contain a header line.

47a ⟨*db-populate-census.sh* 47a⟩≡

```
tail -n +3 ../data/raw/neighborhood-data/census.csv > /tmp/census.txt
⟨start-sqlite 2a⟩ < db-populate-census.cmd
```

47b ⟨*db-populate-census.cmd* 47b⟩≡

```
.echo on
⟨create-table-census 46⟩
.separator "\t"
.import /tmp/census.txt census
.tables
SELECT count(*) FROM census;
```

The script loads 2054 data records into the `census` table.

I want to use average commuting time as a feature. It can be derived from the existing columns in `census`.

TODO: add fraction owner-occupied as well

47c ⟨*db-alter-census.sh* 47c⟩≡

```
⟨start-sqlite 2a⟩ < db-alter-census.cmd
```

48a ⟨*db-alter-census.cmd* 48a⟩≡ 48b▷

```
ALTER TABLE census ADD COLUMN average_commute FLOAT DEFAULT 0;
UPDATE census
SET average_commute = ((travel_less_5     * ((0 + 5) / 2)) +
                       (travel_5_to_9     * ((5 + 10) / 2)) +
                       (travel_10_to_14   * ((10 + 15) / 2)) +
                       (travel_15_to_19   * ((15 + 20) / 2)) +
                       (travel_20_to_24   * ((20 + 25) / 2)) +
                       (travel_25_to_29   * ((25 + 30) / 2)) +
                       (travel_30_to_34   * ((30 + 25) / 2)) +
                       (travel_35_to_39   * ((35 + 39) / 2)) +
                       (travel_40_to_44   * ((40 + 45) / 2)) +
                       (travel_45_to_59   * ((45 + 60) / 2)) +
                       (travel_60_to_89   * ((60 + 90) / 2)) +
                       (travel_90_or_more * 120)) /
                      workers_not_work_at_home
WHERE workers_not_work_at_home > 0;
```

In the above, the WHERE clause is needed because some census tracts have no workers.

Another desireable features is the fraction of houses that are occupied by their owners.

48b ⟨*db-alter-census.cmd* 48a⟩+≡ ◁48a 48c▷

```
ALTER TABLE census ADD COLUMN fraction_owner_occupied FLOAT DEFAULT 0;
UPDATE census
SET fraction_owner_occupied =
  owner_occupied / (1.0 * owner_occupied + renter_occupied)
WHERE (owner_occupied + renter_occupied) > 0;
```

In the above, the where clause is needed because some census tracts have no one occupying housing.

Finally, to join the `census` table with the taxroll table, I will need the census tract as a separate column.

48c ⟨*db-alter-census.cmd* 48a⟩+≡ ◁48b

```
ALTER TABLE census ADD COLUMN census_tract INTEGER;
UPDATE census
SET census_tract = geo_id2 % 10000;
```

5 Importing the Geocoding File

Although the deeds and taxroll file definitions from CoreLogic contain latitude and longitude fields, those fields are not populated. Someone working on the project long ago purchased a "geocoding" file that associates APNs with latitude and longitude.

The SQL statements below will create table `geocoding`.

49a ⟨*create-geocoding* 49a⟩≡ (49c)

```
DROP TABLE IF EXISTS geocoding;
CREATE TABLE geocoding (
  apn_unformatted        INTEGER PRIMARY KEY,
  latitude               FLOAT,
  longitude              FLOAT
);
```

The commands are run with the script below. A complication was that the tab-separated geocoding file has a 2 headers that should not be imported; it is discarded with the tail command. Another complication is that the tab-separated geocoding file has a space after the tabs, and that confuses SQLite3's import routine. The solution is to use `tr` to convert the tabs to commas.

49b ⟨*db-populate-geocoding.sh* 49b⟩≡

```
tail -n +2 ../data/raw/geocoding.tsv | sed 's/\t /,/g' > /tmp/geocoding.txt
#sed 's/\t /,/g' /tmp/geocoding1.txt > /tmp/geocoding2.txt
#head /tmp/geocoding2.txt > /tmp/geocoding.txt
⟨start-sqlite 2a⟩ < db-populate-geocoding.cmd
```

49c ⟨*db-populate-geocoding.cmd* 49c⟩≡

```
.echo on
⟨create-geocoding 49a⟩
.separator ','
.import /tmp/geocoding.txt geocoding
SELECT count(*) FROM geocoding;
```

Running this script creates about 2.4 million geocoding records.

6 The Big Join

We have populated the SQLite data base with about

- 1.1 million relevant deeds (for entire arms-length transactions on single-family houses), containing an APN field
- 1.4 million relevant taxroll records (for single family houses), containing an APN field stored and a census tract field
- 2,000 census tract records, containing a census tract field stored as an INTEGER
- 2.4 million geocoding records, containing an APN field

All that needs to be done is to join the four tables to form table `observations`. Here is the script:

50 ⟨*db-join.sh* 50⟩≡
⟨*start-sqlite* 2a⟩ `< db-join.cmd`

Here are the SQL commands. The vacuum command physically deletes any logically deleted space in the files. I run it because my development system is disk-space challenged. Originally, there was one big join.

51 ⟨*db-join.cmd* 51⟩≡

```
.echo on
VACCUM;

DROP TABLE IF EXISTS observations;
CREATE TABLE observations
AS
SELECT
taxroll_relevant.apn_unformatted,  -- key field 1
deeds_relevant.sale_date,          -- key field 2

/* deeds features */

deeds_relevant.sale_amount,
deeds_relevant.mortgage_amount,
deeds_relevant.sale_date,
deeds_relevant.recording_date,
deeds_relevant.document_type_code,
deeds_relevant.transaction_type_code,

/* taxroll features : parcel information */

taxroll_relevant.census_tract,
taxroll_relevant.zoning,
taxroll_relevant.township,
taxroll_relevant.thomas_bros_map_number,
taxroll_relevant.flood_zone_community_panel_id,
taxroll_relevant.centroid_code,
taxroll_relevant.homestead_exempt,
taxroll_relevant.absentee_indicator_code,
taxroll_relevant.tax_code_area,
taxroll_relevant.universal_land_use_code,
taxroll_relevant.county_land_use_1,
taxroll_relevant.county_land_use_2,
taxroll_relevant.property_indicator_code,
taxroll_relevant.municipality_name,
taxroll_relevant.view,
taxroll_relevant.location_influence_code,
taxroll_relevant.number_of_buildings,

/* taxroll features: propperty address information */
```

```
taxroll_relevant.property_city,
taxroll_relevant.property_state,
taxroll_relevant.property_zip_code,
taxroll_relevant.property_carrier_route,
taxroll_relevant.property_match_code,

/* potential features: owner information */

taxroll_relevant.owner_corporate_indicator_flag,

/* potential feature: values information */

taxroll_relevant.total_value_calculated,
taxroll_relevant.land_value_calculated,
taxroll_relevant.improvement_value_calculated,
taxroll_relevant.tax_amount,
taxroll_relevant.tax_year,

/* potential featurs: subdivision information */

taxroll_relevant.subdivision_tract_number,
taxroll_relevant.subdivision_plat_book,
taxroll_relevant.subdivision_plat_page,
taxroll_relevant.subdivision_name,

/* potential features: lot/land information */

taxroll_relevant.front_footage,
taxroll_relevant.depth_footage,
taxroll_relevant.acres                    /* coded 9999(.)9999 */,
taxroll_relevant.land_square_footage,
taxroll_relevant.lot_area,                /* textual description */

/* potential features: square footage information */

taxroll_relevant.universal_building_square_feet,
taxroll_relevant.universal_building_square_feet_indicator_code,
taxroll_relevant.building_square_feet,
taxroll_relevant.living_square_feet,
taxroll_relevant.ground_floor_square_feet,
taxroll_relevant.gross_square_feet,
taxroll_relevant.adjusted_gross_square_feet,
taxroll_relevant.basement_square_feet,
taxroll_relevant.garage_parking_square_feet,

/* potential features: building information */
```

```
taxroll_relevant.year_built,
taxroll_relevant.effective_year_built,
taxroll_relevant.bedrooms,
taxroll_relevant.total_rooms,
taxroll_relevant.total_baths_calculated       /* encoded 999(.)99 */,
taxroll_relevant.total_baths                  /* encoded 999(.)99 */,
taxroll_relevant.full_baths,
taxroll_relevant.half_baths,
taxroll_relevant.one_quarter_baths,
taxroll_relevant.three_quarter_baths,
taxroll_relevant.bath_fixtures,
taxroll_relevant.air_conditioning_code,
taxroll_relevant.basement_finish_code,
taxroll_relevant.bldg_code,
taxroll_relevant.bldg_improvement_code,
taxroll_relevant.condition_code,
taxroll_relevant.construction_type_code,
taxroll_relevant.exterior_walls_code,
taxroll_relevant.fireplace_indicator_flag    /* "Y" if fireplace in building */,
taxroll_relevant.fireplace_number,
taxroll_relevant.fireplace_type_code,
taxroll_relevant.foundation_code,
taxroll_relevant.floor_code,
taxroll_relevant.frame_code,
taxroll_relevant.garage_code,
taxroll_relevant.heating_code,
taxroll_relevant.mobile_home_indicator_flag   /* "Y" if a mobile home */,
taxroll_relevant.parking_spaces,
taxroll_relevant.parking_type_code,
taxroll_relevant.pool_flag                    /* "Y if pool is present */,
taxroll_relevant.pool_code,
taxroll_relevant.quality_code,
taxroll_relevant.roof_cover_code,
taxroll_relevant.roof_type_code,
taxroll_relevant.stories_code,
taxroll_relevant.stories_number               /* encoded 9(.)99 */,
taxroll_relevant.style_code,
taxroll_relevant.units_number                 /* number of apartments */,
taxroll_relevant.electric_energy_code,
taxroll_relevant.fuel_code,
taxroll_relevant.sewer_code,
taxroll_relevant.water_code,

/* geocoding features */
geocoding.latitude,
```

```
  geocoding.longitude,

  /* census features */
  census.average_commute,
  census.fraction_owner_occupied,
  census.household_median_income

  FROM taxroll_relevant
  NATURAL JOIN deeds_relevant, geocoding, census
  WHERE sale_date != ''
  AND   sale_amount != 0

  ;
  SELECT COUNT(*) FROM observations;
```

Running the join with just deeds_relevant generates about 0.9 million records, with just deeds_relevant and geocoding about

Running the command genearates about 0.9 million records in `observations` without joining geocoding.
The final step is to free the all the tables except for the `observations` table.

54a ⟨*db-free.sh* 54a⟩≡

```
⟨start-sqlite 2a⟩ < db-free.cmd
```

The VACUUM commands defragments space in the file system and then returns it to the file system.

54b ⟨*db-free.cmd* 54b⟩≡

```
DROP TABLE census;
DROP TABLE deeds_all;
DROP TABLE deeds_relevant;
DROP TABLE taxroll_all;
DROP TABLE taxroll_relevant;

VACUUM;
```

Final thoughts go here.