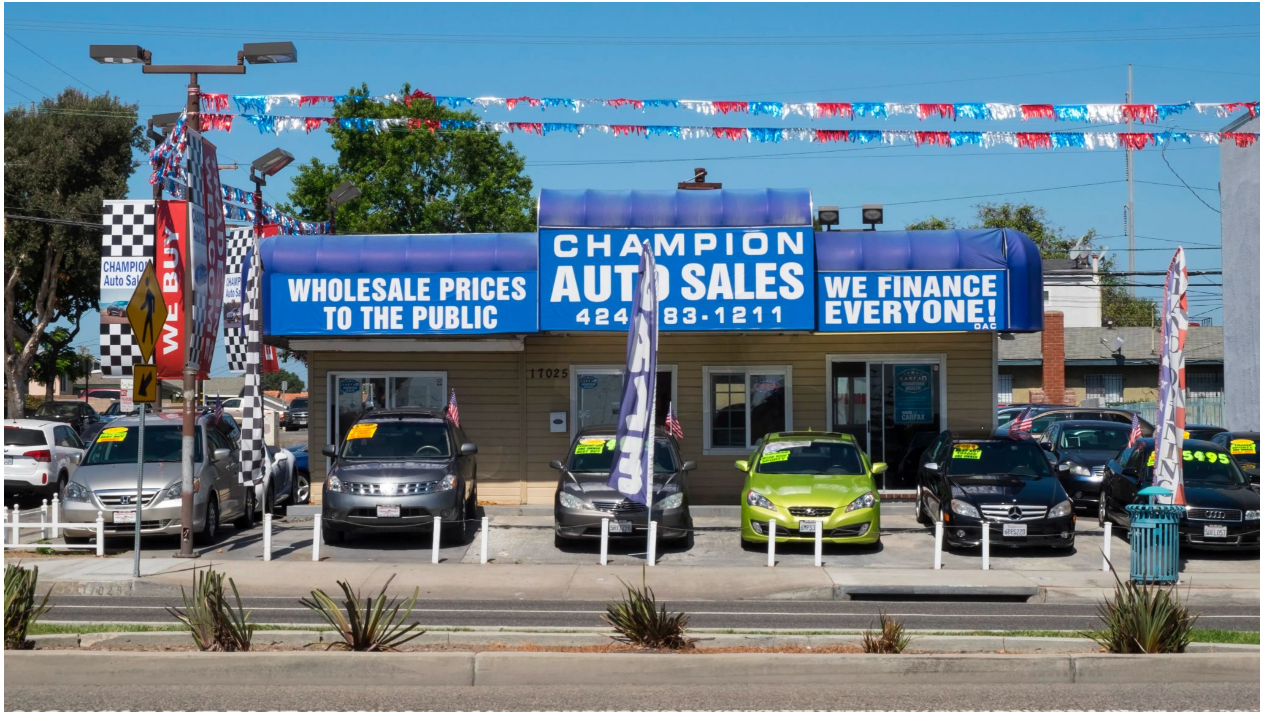
CPSC 437/537 Project: CarMin

Felix Zou and Emily Goldfarb

Used cars!


WHOLESALE PRICES
TO THE PUBLIC
CHAMPION
WE FINANCE
EVERYONE!
WE BUY
17025

Used cars?

- Buying a used car is a notoriously stressful process.
- Unclear prices: Plenty of bad deals disguised as good deals.
- Plus: Dealers have lots of techniques to cheat you out of more money than the car is worth.
  - And still have you leave with a smile on your face…

Dataset 1 and 2

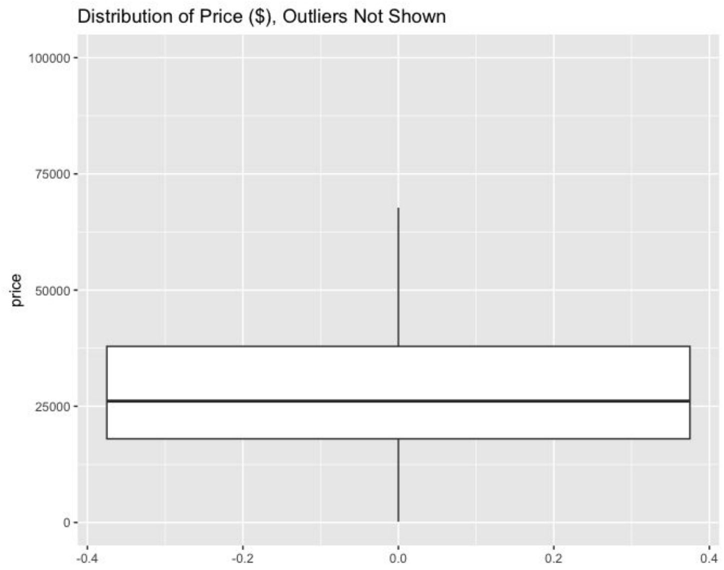
US Used Cars Dataset (Kaggle):

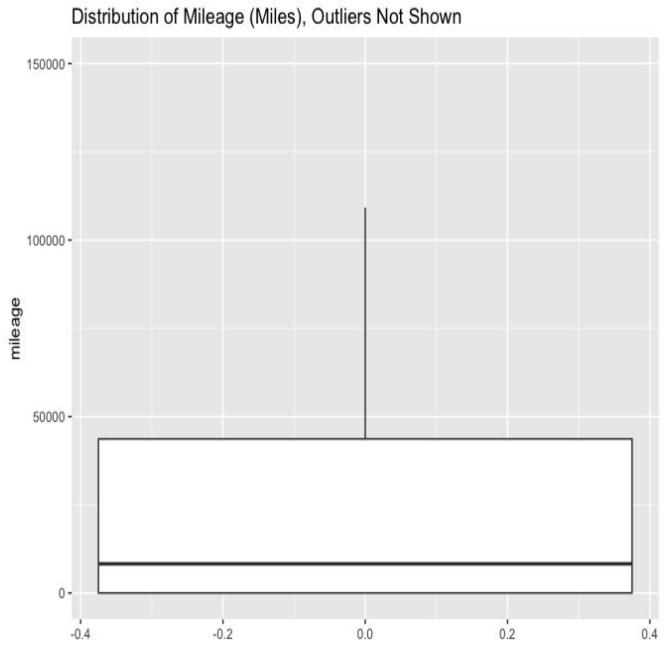
- Approximately 3 million entries
- Data was collected from Car Gurus website inventory
- Example columns: price, mileage, engine type, etc

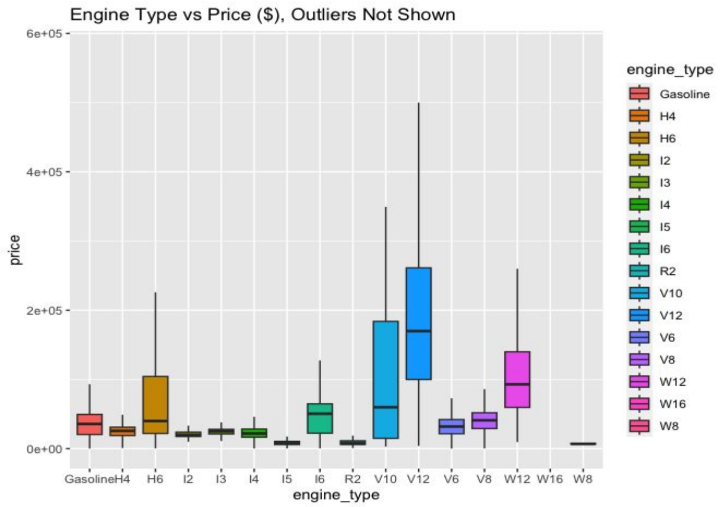
Used Cars Dataset (Kaggle):

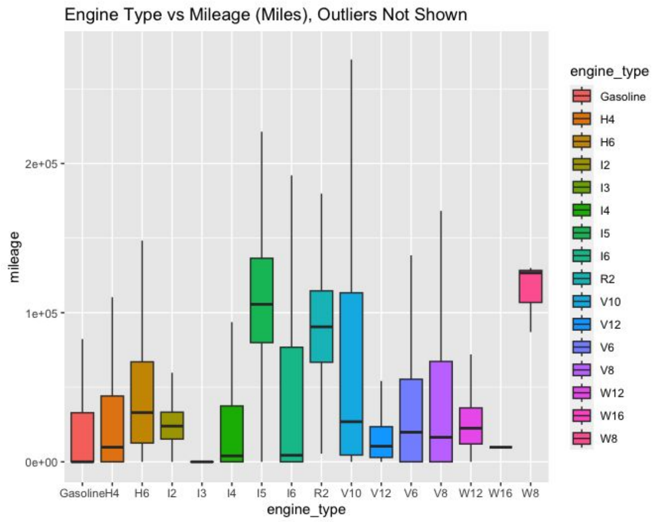
- Approximately 460,000 entries
- Data was collected from Craigslist
- Example columns: price, odometer, transmission type, etc

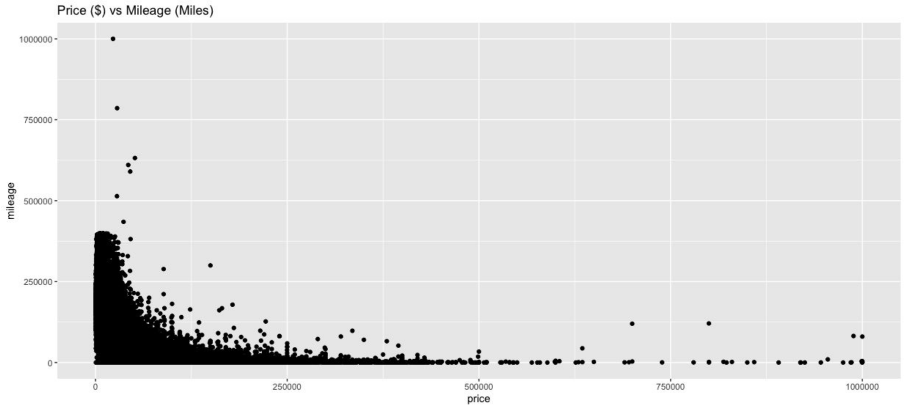
EDA Dataset 1



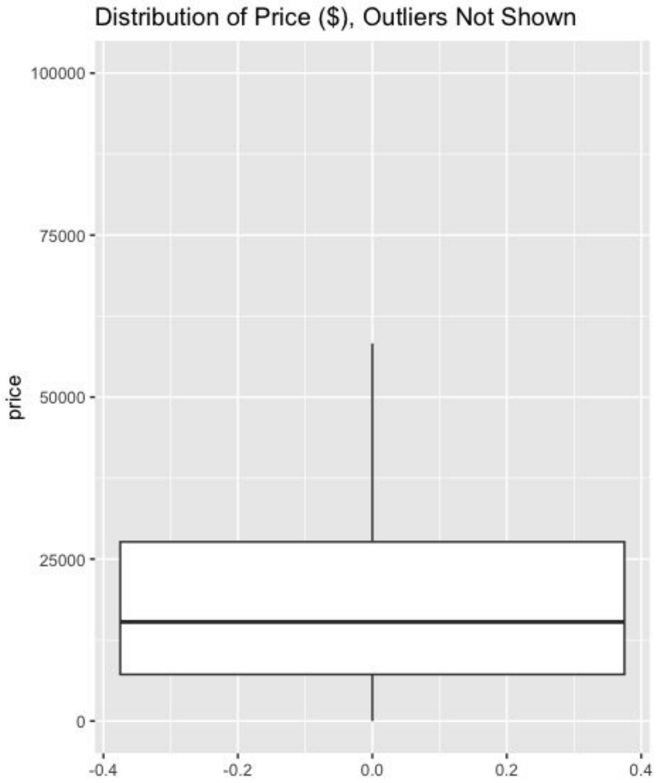


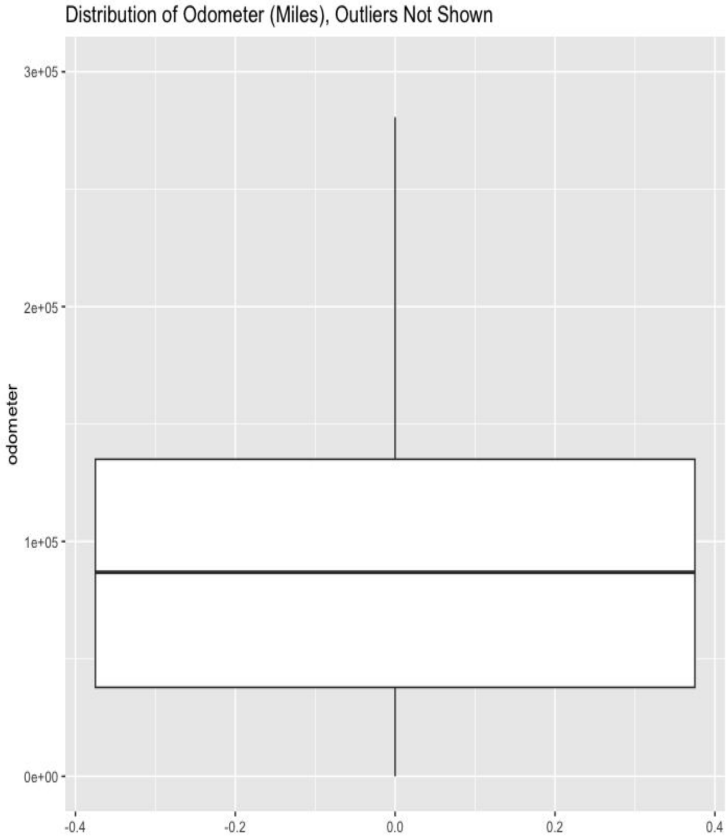

Engine Type vs Price ($), Outliers Not Shown
6e+05
4e+05
2e+05
0e+00
price
GasolineH4 H6 I2 I3 I4 I5 I6 R2 V10 V12 V6 V8 W12 W16 W8
engine_type
engine_type
Gasoline
H4
H6
I2
I3
I4
I5
I6
R2
V10
V12
V6
V8
W12
W16
W8


Engine Type vs Mileage (Miles), Outliers Not Shown
2e+05
1e+05
0e+00
mileage
GasolineH4 H6 I2 I3 I4 I5 I6 R2 V10 V12 V6 V8 W12 W16 W8
engine_type
engine_type
Gasoline
H4
H6
I2
I3
I4
I5
I6
R2
V10
V12
V6
V8
W12
W16
W8

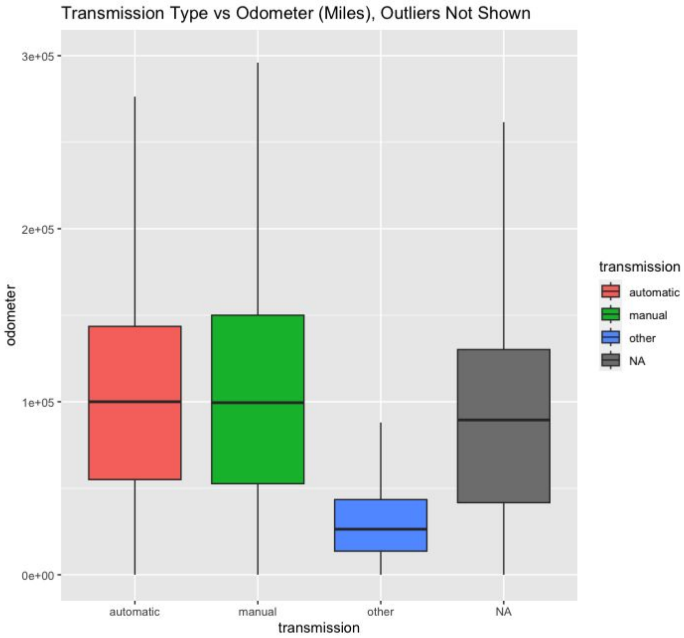

Price ($) vs Mileage (Miles)
1000000
750000
500000
250000
0
mileage
0
250000
500000
750000
1000000
price

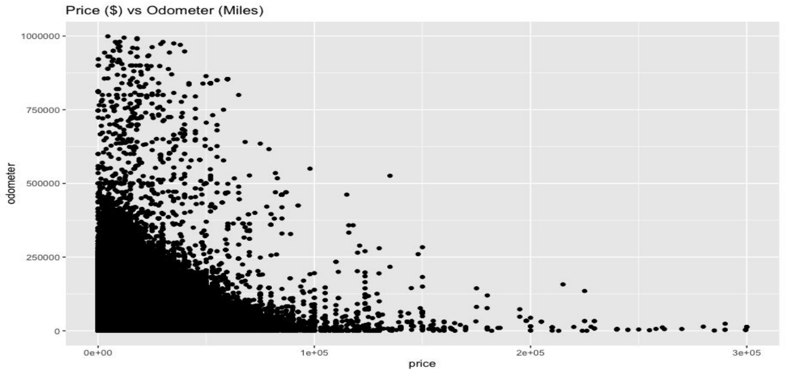
EDA Dataset 2






Transmission Type vs Price ($), Outliers Not Shown
60000
40000
20000
0
price
automatic
manual
other
NA
transmission
transmission
automatic
manual
other
NA


Transmission Type vs Odometer (Miles), Outliers Not Shown
3e+05
2e+05
1e+05
0e+00
odometer
automatic
manual
other
NA
transmission
transmission
automatic
manual
other
NA


Price ($) vs Odometer (Miles)
1000000
750000
500000
250000
0
odometer
0e+00
1e+05
2e+05
3e+05
price

Major Challenges:

- Non-trivial data volume:
  - More than 10 GB of data between the two data sources with lots of redundancy
  - Solution: Normalization and efficient database schema design (final database is less than 1 GB)
- (Very) messy raw data:
  - Solution: Extensive domain-specific data cleaning

Major Challenges (continued):

Very simple data messiness:

Example 1:

`engine_cylinders` has the exact same entries as `engine_type`:

```
(Data1_raw.engine_cylinders.dropna() == Data1_raw.engine_type.dropna()).all()
```

```
True
```

Example 2:

`county` has only null values:

```
Data2_raw.county.isnull().all()
```

```
True
```

Major Challenges (continued):

More elusive data messiness:

As stated before, `VIN` is highly important and should always be 17 digits, is that the case in this dataset?

```
Data2_raw[Data2_raw.VIN.str.len() != 17].VIN
```

```
338       11402312009097
1016          0906419252
1684       116356W169501
1698          7275856000
2524          B1DL120963
                 ...
425059     31847S147227
425300     1Z37L6S418690
426339       CR315045444
426373     CCL449J162701
426376     CCL449F505274
Name: VIN, Length: 1318, dtype: object
```

Nope... don't know what's going on with these VIN's that don't any sense. There are even a couple `VIN` entries with only 1 digit:

```
Data2_raw[Data2_raw.VIN.str.len() == 1].VIN
```

```
25010    C
27561    C
Name: VIN, dtype: object
```

Major Challenges (continued):

More elusive data messiness (example 2):

```
Data2_raw.nlargest(3, "price")
```

	id	region	price	year	manufacturer	model	condition	cylinders	fuel	odometer	...	transmission	VIN	drive	
193736	7315524207	ann arbor	123456789	2015.0	chevrolet	cruze	like new	NaN	gas	64181.0	...	automatic	1G1PC5SB0F7246637	fwd	com
286323	7316737760	akron / canton	12345678	2019.0	chevrolet	NaN	good	8 cylinders	gas	100000.0	...	automatic	00000000000000000	4wd	full
286324	7316737396	akron / canton	12345678	2019.0	chevrolet	NaN	good	8 cylinders	gas	100000.0	...	automatic	00000000000000000	4wd	full

3 rows × 21 columns

There is no way a Chevy Cruze costs 100 million... there has to be some error. Indeed, in these cases, the price is just 1 sequentially entered until 9... Let's

Major Challenges (continued):

Very elusive data messiness:

```
Data1_raw.loc[Data1_raw.trimId == "MMYT047624", ["trimId", "make_name", "model_name", "trim_name", "year", "body_typ
```

	trimId	make_name	model_name	trim_name	year	body_type
129215	MMYT047624	Toyota	Prius Plug-In	Base	2012	Sedan
149803	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
172121	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
196247	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
239168	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
1434532	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
1694471	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
2256008	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
2423967	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback
2529964	MMYT047624	Toyota	Prius Plug-In	Base	2012	Hatchback

Solutions to Major Challenges:

- Messy data:
  - Clean, clean, clean!
  - This part is very long: See Section1_DatabaseCreation.ipynb
- Large data volume:
  - Efficient database schema design (next slide)

Database Schema:

Our database schema is:

- Car(**vin**, *mmyt_id*, odometer, is_certified_preowned, has_accidents, transmission_type, exterior_color, horsepower, max_horsepower_at_rpm, max_torque_at_rpm, engine_type, engine_displacement, fuel_type, city_mpg, highway_mpg)
  - This relation represents a used car instance.
- MMYT(**mmyt_id**, make_name, model_name, production_year, trim_name, body_type, max_seats, fuel_tank_gallons, drivetrain, vehicle_length, vehicle_width, vehicle_height, wheelbase, mmyt_description)
  - This relation represents the make, model, year, and trim of a category of cars.
- Dealer(**dealer_id**, dealer_name, total_listings, avg_rating, location, zipcode, longitude, latitude, is_franchise_dealer)
  - This relationship represents a dealer selling used cars.
- Listing(**listing_id**, *vin*, *dealer_id*, price, listing_year, listing_month, listing_day, days_on_market)
  - This relation represents a sales listing that a dealer has posted for a specific used car.

(Primary keys are **bolded**, foreign keys are *italicized* and have the same name as the primary keys of the relations to which they refer.)

Entity Resolution:

Example: Matching values:

```
Data2_raw["type"] = Data2_raw.type.replace({"sedan":"Sedan",
                                            "SUV":"SUV / Crossover",
                                            "pickup":"Pickup Truck",
                                            "truck":"Pickup Truck",
                                            "other":"Other",
                                            "coupe":"Coupe",
                                            "hatchback":"Hatchback",
                                            "wagon":"Wagon",
                                            "van":"Van",
                                            "convertible":"Convertible",
                                            "mini-van":"Minivan",
                                            "offroad":"SUV / Crossover",
                                            "bus":"Pickup Truck"})
Data2_raw["type"] = Data2_raw.type.fillna("Unknown")
```

Example: Generating ID's:

```
Data1_mmyts.mmyt_id.str[4:].astype(int).describe()
```

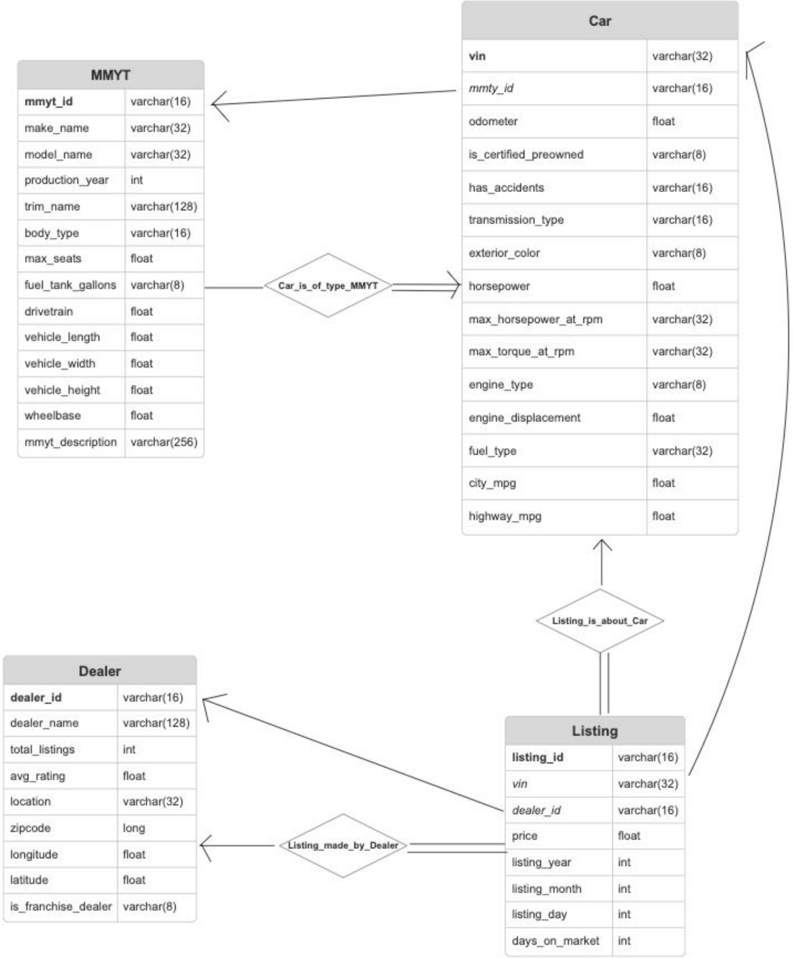
```
count    36964.000000
mean     49635.126231
std      25138.045590
min          3.000000
25%      32179.750000
50%      48896.000000
75%      72480.250000
max      94257.000000
Name: mmyt_id, dtype: float64
```

Looks like we are safe to use anything beyond `"MMYT094257"`! Let's start from `"MMYT095000"` to have some separation in the ID space and go sequentially from there:

```
Data2_mmyts.reset_index(drop=True, inplace=True)
Data2_mmyts["mmyt_id"] = "MMYT" + pd.Series(range(95000, 95000 + Data2_mmyts.shape[0])).astype(str).str.zfill(6)
```

And much more (see notebook)…

Diagram of Schema:



Decomposition is lossless:

This decomposition should be lossless, is it? As a final check, let's make extra sure that, if we merge the splitted relations, we get the original:

```
merged = listings.merge(
            cars.merge(mmyts, on="trimId"),on="vin").merge(
                dealers, on="sp_id")[Data1_raw.columns].sort_values("vin").reset_index(drop=True)
orig_sorted = Data_raw.drop_duplicates().sort_values("vin").reset_index(drop=True)
(merged == orig_sorted).all().all()
```

```
True
```

Yep, exactly the same!

Primary Key Constraints Met:

```
uniquely_determines(["vin"], list(cars.columns), cars)
```

```
Uniques on A: 2666268, uniques on A union B: 2666268
True
```

```
uniquely_determines(["mmyt_id"], list(mmyts.columns), mmyts)
```

```
Uniques on A: 90169, uniques on A union B: 90169
True
```

```
uniquely_determines(["dealer_id"], list(dealers.columns), dealers)
```

```
Uniques on A: 42440, uniques on A union B: 42440
True
```

```
uniquely_determines(["listing_id"], list(listings.columns), listings)
```

```
Uniques on A: 2803235, uniques on A union B: 2803235
True
```

All set! Load into MySQL…

Loading into MySQL

```sql
-- some settings
SET GLOBAL local_infile=1;

-- refresh database
DROP DATABASE IF EXISTS CarMin;
CREATE DATABASE CarMin;
USE CarMin;

-- DATA 1

-- create table for Car and load
DROP TABLE IF EXISTS Car;
CREATE TABLE Car (
    vin VARCHAR(32),
    mmty_id VARCHAR(16),
    odometer FLOAT,
    is_certified_preowned VARCHAR(8),
    has_accidents VARCHAR(16),
    transmission_type VARCHAR(16),
    exterior_color VARCHAR(8),
    horsepower FLOAT,
    max_horsepower_at_rpm VARCHAR(32),
    max_torque_at_rpm VARCHAR(32),
    engine_type VARCHAR(4),
    engine_displacement FLOAT,
    fuel_type VARCHAR(32),
    city_mpg FLOAT,
    highway_mpg FLOAT
);
LOAD DATA LOCAL INFILE "./Data1/Car.csv"
INTO TABLE Car
COLUMNS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

```sql
-- create table for MMYT and load
DROP TABLE IF EXISTS MMYT;
CREATE TABLE MMYT (
    mmyt_id VARCHAR(16),
    make_name VARCHAR(32),
    model_name VARCHAR(32),
    production_year INT,
    trim_name VARCHAR(128),
    body_type VARCHAR(16),
    max_seats FLOAT,
    fuel_tank_gallons FLOAT,
    drivetrain VARCHAR(8),
    vehicle_length FLOAT,
    vehicle_width FLOAT,
    vehicle_height FLOAT,
    wheelbase FLOAT,
    mmyt_description VARCHAR(256),
    PRIMARY KEY(mmyt_id)
);
LOAD DATA LOCAL INFILE "./Data/MMYT.csv"
INTO TABLE MMYT
COLUMNS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

```sql
-- create table for Dealer and load
DROP TABLE IF EXISTS Dealer;
CREATE TABLE Dealer (
    dealer_id VARCHAR(16),
    dealer_name VARCHAR(128),
    total_listings INT,
    avg_rating FLOAT,
    location VARCHAR(32),
    zipcode LONG,
    longitude FLOAT,
    latitude FLOAT,
    is_franchise_dealer VARCHAR(8),
    PRIMARY KEY(dealer_id)
);
LOAD DATA LOCAL INFILE "./Data/Dealer.csv"
INTO TABLE Dealer
COLUMNS TERMINATED BY ';'
LINES TERMINATED BY '\n';

-- create table for Listing and load
DROP TABLE IF EXISTS Listing;
CREATE TABLE Listing (
    listing_id VARCHAR(16),
    vin VARCHAR(32),
    dealer_id VARCHAR(16),
    price FLOAT,
    listing_year INT,
    listing_month INT,
    listing_day INT,
    days_on_market INT,
    PRIMARY KEY(listing_id)
);
LOAD DATA LOCAL INFILE "./Data/Listing.csv"
INTO TABLE Listing
COLUMNS TERMINATED BY ';'
LINES TERMINATED BY '\n';
```

SQL tables

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| CarMin             |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.15 sec)

mysql> USE CarMin
Reading table informatior
You can turn off this fea

Database changed
mysql> SHOW TABLES;
+------------------+
| Tables_in_carmin |
+------------------+
| Car              |
| Dealer           |
| Listing          |
| MMYT             |
+------------------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM MMYT WHERE make_name = "Jeep" AND model_name = "Liberty" LIMIT 5;
+------------+-----------+------------+-----------------+--------------+------------------+-----------+-------------------+------------+-
-------------------------+
| mmyt_id    | make_name | model_name | production_year | trim_name    | body_type        | max_seats | fuel_tank_gallons | drivetrain |
escription               |
+------------+-----------+------------+-----------------+--------------+------------------+-----------+-------------------+------------+-
-------------------------+
| MMYT007003 | Jeep      | Liberty    |            2006 | Limited      | SUV / Crossover  |         5 |                20 | RWD        |
eep Liberty Limited      |
| MMYT007004 | Jeep      | Liberty    |            2006 | Limited 4WD  | SUV / Crossover  |         5 |                20 | 4WD        |
eep Liberty Limited 4WD  |
| MMYT007005 | Jeep      | Liberty    |            2006 | Renegade     | SUV / Crossover  |         5 |                20 | RWD        |
eep Liberty Renegade     |
| MMYT007006 | Jeep      | Liberty    |            2006 | Renegade 4WD | SUV / Crossover  |         5 |                20 | 4WD        |
eep Liberty Renegade 4WD |
| MMYT007007 | Jeep      | Liberty    |            2006 | Sport        | SUV / Crossover  |         5 |                20 | RWD        |
eep Liberty Sport        |
+------------+-----------+------------+-----------------+--------------+------------------+-----------+-------------------+------------+-
-------------------------+
```

Accessing MySQL from Python

Example with a complex query to join all tables together for modelling:

```
# connect to MySQL Server -> assumes Section2_LoadIntoMySQL.sql has already been run
# NOTE: For demo purposes, at present, the password for MySQL server on my local machine
#       is temporarily set to "insecure_password"
mysql_connection = pymysql.connect(host="localhost", user="root", password = "insecure_password", db="CarMin")
cursor = mysql_connection.cursor()

# complex query to decompress compressed data
sql_get_full_table = """
                     SELECT *
                     FROM Listing
                         LEFT OUTER JOIN (Car LEFT OUTER JOIN MMYT USING (mmyt_id)) USING (vin)
                         LEFT OUTER JOIN Dealer USING (dealer_id);
                     """

# run query through MySQL and store result in pandas dataframe
data_all = pd.read_sql_query(sql_get_full_table, mysql_connection)

# finally, close connection
mysql_connection.close()
```

Accessing MySQL from Python

Our code also supports executing arbitrary SQL queries without going through Pandas:

```
init()

# example to joing make/model information with specific car instances
execute_sql_query("""

SELECT make_name, model_name, production_year, odometer, exterior_color
FROM Car LEFT OUTER JOIN MMYT USING (mmyt_id)
WHERE make_name = "Ford" AND model_name = "Focus"
LIMIT 5;

""")

end()
```

```
Ford Focus 2013 100639.0 UNKNOWN
Ford Focus 2013 102699.0 WHITE
Ford Focus 2013 108966.0 BLACK
Ford Focus 2013 150394.0 BLACK
Ford Focus 2013 79369.0 GRAY
```

Additional Feature: Predicting Price

- User enters some information of vehicle (does not have to be complete)
- Our model uses the database to predict its price:

```
init()

print_predicted_price(make_name="Chevrolet", model_name="Camaro", production_year=2018, odometer=40000)
print_predicted_price(make_name="Dodge", model_name="Challenger", production_year=2020, odometer=35000)
print_predicted_price(make_name="Volkswagen", model_name="Golf", production_year=2023, odometer=25000)

end()
```

```
The predicted price of the input vehicle is: 29768.50316614245 dollars.
The predicted price of the input vehicle is: 34032.006530875995 dollars.
The predicted price of the input vehicle is: 32662.41339728704 dollars.
```

Additional Feature: Predicting Price

For reference, our prediction was done using LGBM:

```
lgb_params = {
    "objective":"regression",
    "boosting":"gbdt",
    "learning_rate":0.1,
    "num_leaves":1024, # there are a lot of data and problem is regression
    "max_depth":-1, # lgb grows trees sideways
    "min_data_in_leaf":80,
    "metric":"l1",
    "seed":2830,
    "verbose": -1
}
num_iterations = 1000
lgb_model = lgb.train(lgb_params, dataset_train, num_iterations,
                      valid_sets=[dataset_validate], verbose_eval=False)
lgb_model.save_model("./Models/MODEL_LGBM.txt")
```

```
<lightgbm.basic.Booster at 0x7fe1cf231db0>
```

```
test_predictions = lgb_model.predict(dataset_test.data)
mean_absolute_error(dataset_test.label, test_predictions)
```

```
2204.9372912887557
```

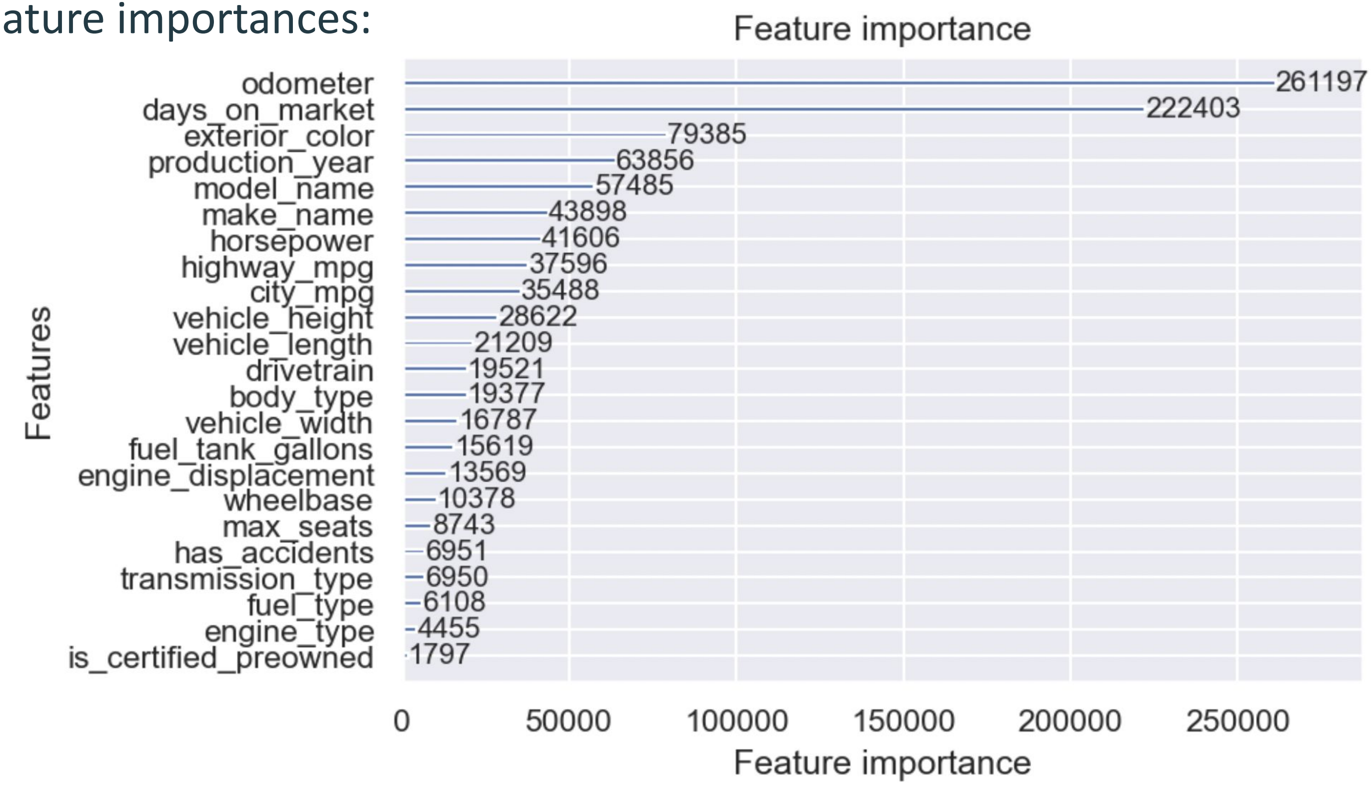
Not bad! Dealers frequency mark cars up/down completely arbitrarily by more than 2k!

NOTE: Stemming from domain knowledge, there is reason to believe that this is about as good as it's going to get, since a variance of 2K in **listing price** is completely normal, even for the exact same car. Some dealers will issue a "dealer discount" of 1-3K (sometimes even more) on the listing to attract potential buyers and then add those 1-3K back as "extra equiptment", "dealer fees", "transportation fees", etc., etc., etc., when you start discussing payment with them (though you can haggle some of this away). Other dealers -- usually the bigger ones -- are more honest and will not do this. I have had a scammy dealer try to pull a 3K (?!) transportation fee on me for a car that got shipped from New York to New Jersey (where they were located)!

I also tried adding the column `dealer_name` into the inputs, and the testing mean absolute error immediately went down to ~$1800... However, from the perspective of someone trying to get a car, they will not know what dealer it will be from and so the column `dealer_name` should logically not be a predictor.

Additional Feature: Predicting Price

Feature importances:



Reflection: So what was learned?

- Used prices vary most by odometer/days_on_market but also significantly by trim:
    - Discrete prediction methods (trees etc.) perform better
- Prices have very high variance even within a specific model!
- Dealers vary wildly from one another in their business practices.
- The whole industry is filled with traps for unknowing buyers:
    - Our work can help!

Reflection: Correct decisions made

- Thorough cleaning of data and careful decomposition.
  - The entire database fits in one git repository without overflowing GitHub’s storage limits!
- Prediction-specific processing, imputation, and binning.
- Using tree models for prediction given discrete nature of the underlying relationship.

Reflection: Things we would do differently

- Do entity resolution as the first step:
  - Doing it towards the end of database creation created a lot of extra headache.
- Looked for a third dataset.
- Experimented with ensembling different models together.

Thank You!