Shift Expected Margin

Exploratory Data Analysis and Predictive Models

by: Huang Pan

Notebook Outline

1. Introduction
   a. ETL
   b. Plan of Action
2. Exploratory Data Analysis with Percent to Market Model
3. Predictive Models
   a. Currently Available Research
   b. Feature Exploration
   c. Linear Regression
   d. Gradient Boosted Tree Regression
4. Further Research

Overview

Shift's operations team that is responsible for customer outreach deals with thousands of potential sellers per month. One way our data scientists can make their job easier is by prioritizing which sellers they outreach to first. The goal of this assignment is to derive a score that represents the expected value to Shift of acquiring a specific vehicle. The expected value that we want to derive in this homework assignment is the difference between what Shift can sell the vehicle for at retail and what dealerships buy vehicles for at wholesale.

The Data

The dataset includes a random sample of seller inquiries Shift has received in the last ~5 months for a select set of models. You will find that some sellers have expected value set to $0, and that's because they never ended up selling their vehicle through Shift, so we don't know what we could have sold their car for at retail. The other set of sellers, those that have a non-zero value for expected margin, did sell their vehicle through Shift and the value represents the final selling price less the wholesale value of their vehicle.

Q: What is the wholesale value?
A: The wholesale value is how much this vehicle would have sold for at an auction.

The Assignment

We'd like you to present to our engineering and data science team your approach to the problem, which statistical/machine learning methods you used to come up with the model, the model itself and accuracy of your predictions. A handful of slides will suffice, but in addition to the presentation, we would like to see your code and any model diagnostics that you think will be useful to share.

```
# Python good when using dataframes API, python + jupyter notebooks most popular
# Uses Spark 2.+ cluster
spark # start Spark session

# Already uploaded training_data.csv to DBFS using the Databricks Table UI
# Create a Spark DataFrame from the CSV file by inferring the schema
# training_data_DF =
spark.read.csv('/FileStore/tables/3lrl8qd31476311571313/training_data.csv',
header=True, inferSchema=True)
training_data_DF =
spark.read.csv('/FileStore/tables/035u47tv1477361256947/training_data.csv',
header=True, inferSchema=True)

# quote_id, year, make, model, mileage, location_zip, percent_to_market,
market_avg_days_to_sell, n_similar_vehicles_listed, region_short_code,
exact_list_price, wholesale_price, seller_source, expected_margin
print training_data_DF # inferred schema looks OK
print training_data_DF.count() # 3853 rows of data in original dataset, not including
header
print training_data_DF.rdd.getNumPartitions() # small amount of data, only one
partition used

DataFrame[quote_id: string, year: int, make: string, model: string, mileage: int, loca
tion_zip: int, percent_to_market: double, market_avg_days_to_sell: int, n_similar_vehi
cles_listed: int, region_short_code: string, exact_list_price: int, wholesale_price: i
```

```
nt, seller_source: string, expected_margin: int]
3853
1
```

Data Glossary

Quote ID: Unique Seller Identifier
Location Zip: Postal code of the seller
Percent to Market: Price at which Shift would try to sell the vehicle relative to comparable vehicles currently listed in the market
Market Average Days To Sell: # of days it takes an average dealer to sell a similar vehicle
N Similar Vehicles: Number of similar vehicles for sale nationally
Region Shortcode: Region of the seller
Exact List Price: Price Shift would list the vehicle for (if acquired)
Wholesale Price: What dealers can buy a similar vehicle for at an auction
Seller Source: Whether the seller contacted us through shift.com or through our iOS app
Expected Margin: The value of Shift selling this car for the retail price. The final selling price less the wholesale value of their vehicle. In finance terms, Expected Margin is somewhat like a derivative of used car prices.

```
# NOTE: exact_list_price - wholesale_price doesn't necessarily equal expected_margin; final sales price could be different from exact_list_price
display(training_data_DF.limit(5))

# register the dataframe as a table in order to use SQL commands
training_data_DF.createOrReplaceTempView("training_data")
```

quote_id	year	make	model	mileage	location_zip	percent_to_market	market_avg_da
0061a00000GZsY8AAL	2012	Scion	tC	67850	94022	0.940326152	88
0061a00000F7X4lAAF	2006	Toyota	Tacoma	120400	94107	0.93519453	47
0061a00000FttwmAAB	2014	Dodge	Journey	28600	90049	0.955719563	46
0061a00000F8F2yAAF	2015	Honda	Accord	12000	94518	0.966639796	148
0061a00000F810VAAR	2010	Nissan	Sentra	50000	90402	0.925793869	41

```
%sql
```

```
-- Data Cleaning
DROP TABLE IF EXISTS clean_data1;
DROP TABLE IF EXISTS clean_data2;
DROP TABLE IF EXISTS clean_data3;
DROP TABLE IF EXISTS clean_data4;
DROP TABLE IF EXISTS clean_data5;
DROP TABLE IF EXISTS clean_data6;
DROP TABLE IF EXISTS clean_data7;

-- Remove rows with any missing data (nulls)
CREATE TABLE clean_data1 AS
SELECT * FROM training_data
WHERE year IS NOT NULL AND make IS NOT NULL AND model IS NOT NULL AND mileage IS NOT
NULL AND location_zip IS NOT NULL AND percent_to_market IS NOT NULL AND
market_avg_days_to_sell IS NOT NULL AND n_similar_vehicles_listed IS NOT NULL AND
region_short_code IS NOT NULL AND seller_source IS NOT NULL;

-- Remove rows with null or 0 wholesale price / exact_list_price; expected margin just
don't look right for that those rows
CREATE TABLE clean_data2 AS
SELECT * FROM clean_data1
WHERE wholesale_price IS NOT NULL AND wholesale_price != 0 AND exact_list_price IS NOT
NULL AND exact_list_price != 0;

-- Remove outliers in expected_margin column
CREATE TABLE clean_data3 AS
SELECT * FROM clean_data2
WHERE expected_margin BETWEEN -50000 AND 100000;

-- Remove outliers in percent_to_market column
CREATE TABLE clean_data4 AS
SELECT * FROM clean_data3
WHERE percent_to_market BETWEEN .8 AND 1;

-- Remove outliers in market_avg_days_to_sell column; zeros were screwing up the log
transforms, one outlier >> 500
-- Looking at the data, the zeros are clearly erroneous - similar make/model/year have
different non-zero values
-- In reality, those zeros need to be cleaned up and replaced by something more
realistic
CREATE TABLE clean_data5 AS
SELECT * FROM clean_data4
WHERE market_avg_days_to_sell BETWEEN 1 AND 500;

-- Create Total Margin column
-- exact_list_price - wholesale_price doesn't necessarily equal expected_margin; final
sales price could be different from exact_list_price
-- Let's assume that the exact_list_price - wholesale_price is the total max profit we
```

```
could have made on the deal; let's call this number total_margin
CREATE TABLE clean_data6 AS
SELECT quote_id, year, make, model, mileage, location_zip, percent_to_market,
market_avg_days_to_sell, n_similar_vehicles_listed, region_short_code,
exact_list_price, wholesale_price, seller_source, expected_margin, (exact_list_price -
wholesale_price) AS total_margin FROM clean_data5;

-- Create Calculated Margin column
-- First basic Expected Margin model: use 58% * total margin as an approximate proxy
or target
-- see below for why we use 58%
-- The margin column is an imputation of missing values using calculated_margin when
expected_margin is 0
CREATE TABLE clean_data7 AS
SELECT quote_id, year, make, model, mileage, location_zip, percent_to_market,
market_avg_days_to_sell, n_similar_vehicles_listed, region_short_code,
exact_list_price, wholesale_price, seller_source, expected_margin, total_margin,
INT(.58*total_margin) as calculated_margin, IF(expected_margin != 0, expected_margin,
INT(.58*total_margin)) AS margin FROM clean_data6;
```

OK

Data Issues

- Must have an emphasis on data cleaning: avoid garbage in, garbage out problem
- How accurate is the market price --> percent_to_market? Market price: buy data or scrape websites for used car prices: Kelly Blue Book, Edmunds, NADA, Ebay, Craigslist, etc., compare against similar vehicles (make/model/year/mileage) for sale in area
- Track actual sales price of similar cars? Ebay, dealers, etc.
- Assumption: vehicle sold in area it was listed in? Need extra data point: area sold?
- How about wholesale price? How accurate is the data?
- More systematic/statistical way of removing outliers
- Check for valid Year/Make/Model
- Check for valid range of mileage: >= 0
- Check for valid location_zip
- Check for valid range of n_similar_vehicles_listed
- Check for valid region_short_code
- Check for valid exact_list_price and wholesale_price
- Check for valid seller_source
- Check for valid expected_margin
- Any duplicates in the data set?
- Need time of sales data for time series analysis

- Use caching in Spark as the amount of data gets larger

```
%sql

-- Calculate Profit/Loss statistics
DROP TABLE IF EXISTS temp_pnl;
DROP TABLE IF EXISTS pnl;
DROP TABLE IF EXISTS non_zero_expected_margin;
DROP TABLE IF EXISTS margin_summary;

-- expected_margin: total loss
-- expected_margin: total gain
-- expected_margin: total profit
-- total profit lost by not converting seller to customer: total(calculated_margin) -
total(expected_margin)
-- conversion rate to shift: # of shift sellers / # of shift inquiries
CREATE TABLE temp_pnl AS
SELECT a.total_loss, b.total_gain, c.total_profit, d.total_profit_lost,
e.num_shift_sellers, f.num_shift_inquiries FROM
(SELECT SUM(expected_margin) AS total_loss FROM clean_data7 WHERE expected_margin < 0)
as a,
(SELECT SUM(expected_margin) AS total_gain FROM clean_data7 WHERE expected_margin > 0)
as b,
(SELECT SUM(expected_margin) AS total_profit FROM clean_data7) as c,
(SELECT (SUM(margin) - SUM(expected_margin)) AS total_profit_lost FROM clean_data7) as
d,
(SELECT COUNT(*) AS num_shift_sellers FROM clean_data7 WHERE expected_margin != 0) as
e,
(SELECT COUNT(*) AS num_shift_inquiries FROM clean_data7) as f;

CREATE TABLE pnl AS
SELECT total_loss, total_gain, total_profit, total_profit_lost, num_shift_sellers,
num_shift_inquiries, num_shift_sellers/num_shift_inquiries AS conversion_rate FROM
temp_pnl;

-- non-zero expected_margin vs. total_margin relationship
CREATE TABLE non_zero_expected_margin AS
SELECT * FROM clean_data7
WHERE expected_margin != 0;

-- summarize non-zero expected_margin vs. total_margin relationship
CREATE TABLE margin_summary AS
SELECT INT(avg(expected_margin)) AS average_EM, INT(avg(total_margin)) AS average_TM,
SUM(expected_margin) AS sum_EM, SUM(total_margin) AS sum_TM,
avg(expected_margin)/avg(total_margin) AS EMdivTM FROM non_zero_expected_margin;
```

OK

```
# convert tables back to dataframes
cleaned_data_DF = sqlContext.sql("SELECT * FROM clean_data7")
pnl_DF = sqlContext.sql("SELECT * FROM pnl")
non_zero_EM_DF = sqlContext.sql("SELECT * FROM non_zero_expected_margin")
margin_summary_DF = sqlContext.sql("SELECT * FROM margin_summary")


# Overall Profit & Loss numbers. Conversion rate is only 20% of inquiries, resulting
in $5m total_profit_lost. Need to get that rate up!
display(pnl_DF)
```

total_loss	total_gain	total_profit	total_profit_lost	num_
-175775	1453136	1277361	4946720	731

Plan of Action

NOTE: In this notebook, let's refer to Expected Margin as what we want to calculate. Let's refer to expected_margin as the sales data that has been given to us - the non-zero margin of cars that have been sold by Shift.

In addition, normally we would want to model price. However, in Shift's case, we are modeling Expected Margin = final sales price - the wholesale price. Since the wholesale price is a predetermined, fixed quantity - a constant, we will end up modeling final sales price in the form of Expected Margin.

Keeping the end Business Goals in mind

So what are we exactly trying to do here? The end goal is to increase profits for the company by using all available data.

How do we do that? There are many ways.

For example, from the above, we can use statistics to figure out ways to increase the number of inquiries. We can also use predictive pricing models to figure out ways to best increase the average Expected Margin and conversion rate.

Exploratory Data Analysis

The point of Exploratory Data Analysis (EDA) is to analyze descriptive statistics of the data in order to prioritize which areas of the business to target. We want to maximize usage of available resources.

For example, through EDA we can increase profits by figuring out which locations Shift should focus on. Through outreach campaigns to sellers at these locations, we can increase the number of Shift inquiries. Through marketing campaigns to buyers at these locations, we can increase the conversion rate.

Predictive Models

The point of Predictive Modeling is to generate a pricing model that statistically maximizes profit for the company. This statistical model has 3 uses:

1. Be a better, more consistent predictor of price than arbitrarily picking a percent_to_market price. Through this pricing model, we can identify arbitrage opportunities where cars are underpriced with respect to the market. We can also identify cars that are overpriced, and avoid those cars. By focusing outreach on underpriced cars and avoiding overpriced ones, we can increase the average Expected Margin and total profit for the company.

2. Predict the Expected Margin for cars that we don't have expected_margin data for. Help guide us on how to price those cars in order to maximize profit.

3. From this pricing model, we can also determine if there any underlying factors/features that have a large effect on price. If so, we can use this information to help maximize profits for the company. We can target cars with these features during outreach/marketing campaigns that increase the conversion rate and number of inquiries. This information will help us maximize profits.

With any statistical model, there's a trade off between complexity and effectiveness. Some models may have the best performance, but may be too complex (optimal parameters too hard to find) to be put into production. For our purposes, let's start with the simplest model first, and then try more complex models:

- The simplest model is the Percent to Market Model (PMM). This model is somewhat arbitrary. Let us determine the market rate for the car and multiple it by a percentage < 100% to arrive at a final sales price. This may not be the best way to price cars as: A. It is very labor intensive and time consuming to gather the market rate for each

make/model/year/mileage out there. This process may also be very error prone. B. Results from pricing cars this way may be very random. Still, this is a simple technique and may be worth evaluating.

- Quantitative models may give more consistent results. For our first quantitative model, let us try multivariate linear regression based on car feature set.
- Finally, we can try more sophisticated quantitative models like Gradient Boosted Tree Regression, Support Vector Regression, Neural Networks, etc.

To evaluate and compare each model, a common training loss function that we can use is the Root Mean Square Error (RMSE) of the Expected Margin generated from the model and the expected_margin of the car actually sold. The goal is to use the pricing model with the smallest RMSE. The smaller the RMSE, the stronger the predictive power of the model. By minimizing RMSE, we can minimize any negative Expected Margin.

In general, we should try to use the simplest model possible that gives us the desired results that we want. The simpler the model, the fewer sources of error in our calculations, and the easier it is to understand & debug our machine learning system. And we should upgrade our quantitative model only when it consistently gives us better performance.

Empirical Testing Framework:

Once we have these pricing models, we need a framework to test them in real life.

Given a pricing model, let us look at metrics of how well the pricing model works in the field. The most important metrics are expected_margin, the number of inquiries, and the number of cars sold through Shift. We can use these metrics to calculate total profit, total profit lost, and the conversion rate. Another useful metric would be the number of days it takes to sell a car.

For the price testing framework, we can test different pricing models out by region or zip code. In each zip code we can try out a different pricing model to see which one performs the best empirically according to the metrics above. We can also use Cohort Analysis and break the analyses into further subsets, e.g. by Make/Model/Year/Mileage, etc. E.g. for each zip code, which make/model has the highest Expected Margin? The highest conversion rate?

We can also track the performance over time. If we look at the entire system as closed system with feedback loop, we can adjust model parameters over time to achieve optimal performance/maximum profit. Please note that this production framework must support

quick iteration of models.

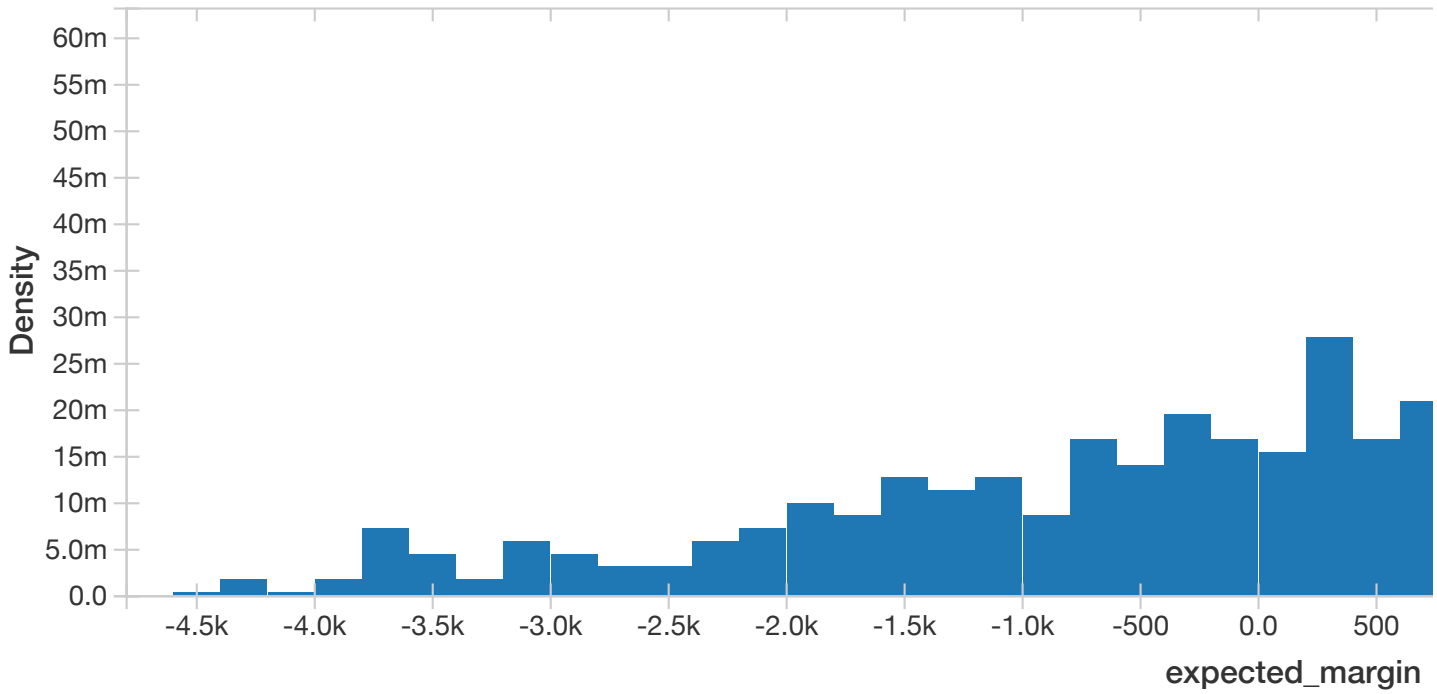
Exploratory Data Analysis

- I looked at the data in Excel first to come up with plan of action for the data cleaning and exploration.
- Data exploration is necessary to get an idea of which models to apply.
  You get a sense for which features are important. (feature engineering)
  It is also catches outliers in the data.
- In the data exploration below, I compare expected_margin with calculated_margin from the Percent to Market Model.

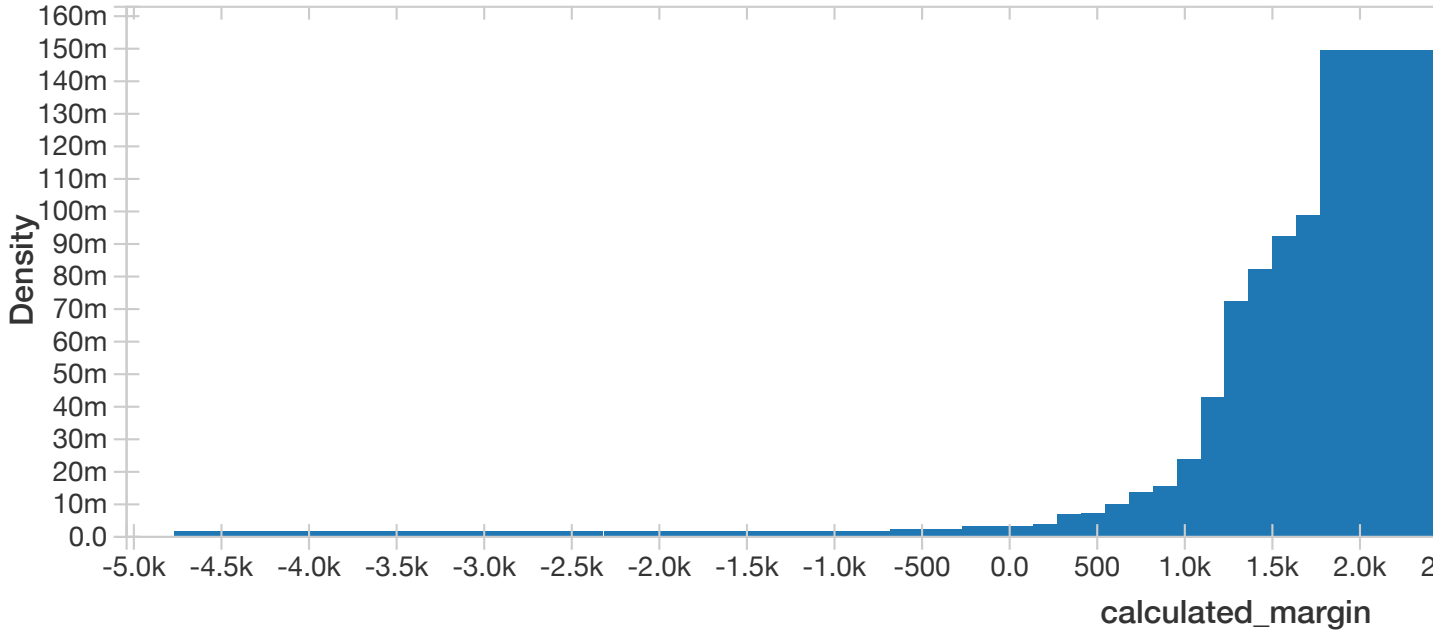
```
# For the most basic model of calculating Expected Margin for the rows where expected_margin is 0, let's compare expected_margin against total_margin for the rows where we do have data for expected_margin. We find that on average, expected_margin is 58% of total_margin.
# We use this percentage (58%) to calculate the calculated_margin column = 58% * total_margin. This is the Percent to Market Model.
display(margin_summary_DF)
```

average_EM	average_TM	sum_EM
1747	3008	1277361

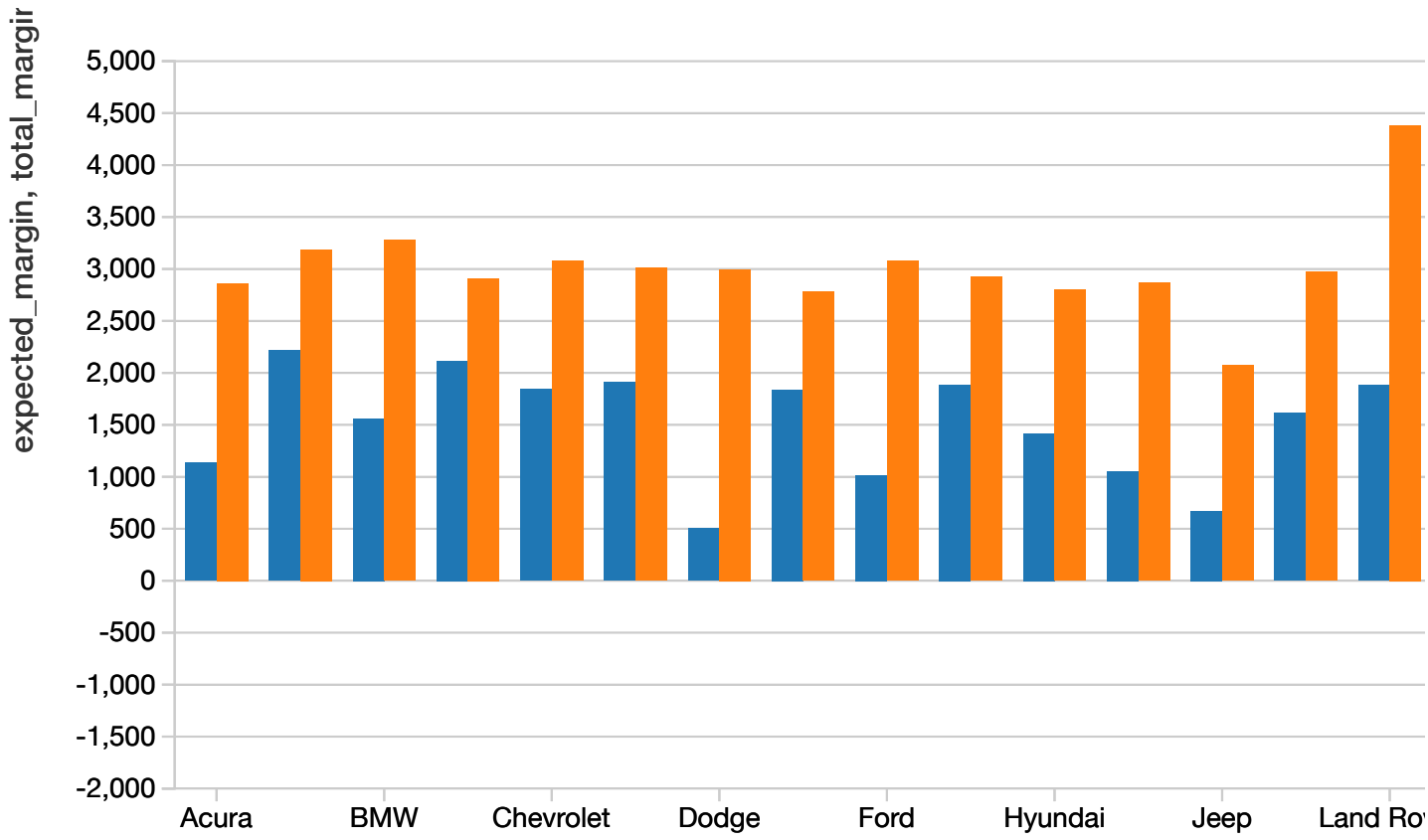
```
# What is the RMSE of the 58% PMM?
# Note: the RMSE for a 68% PMM on Honda Civic is $1639
from sklearn.metrics import mean_squared_error
from math import sqrt

rmse = sqrt(mean_squared_error(non_zero_EM_DF.toPandas()
['expected_margin'].values.tolist(), non_zero_EM_DF.toPandas()
['calculated_margin'].values.tolist()))
print('PMM RMSE: ', rmse)

('PMM RMSE: ', 1737.7977304969595)

# Plot the distribution of expected_margin
# Later on we can subset this by location and make/model/year/mileage, etc. when we have more data
display(non_zero_EM_DF.sort("make"))
```



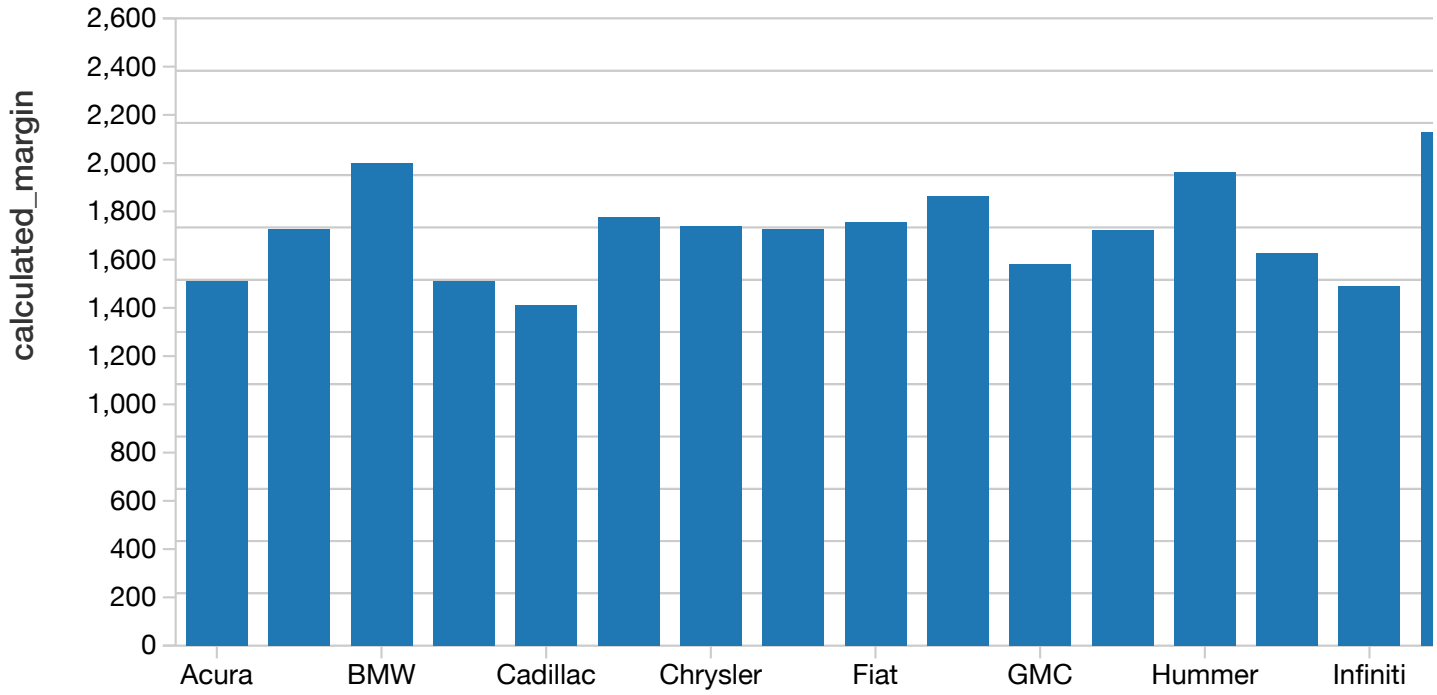

```
# Plot the distribution of calculated_margin
# Spark histogram plot has issues, the distribution of calculated_margin and
expected_margin have somewhat similar characteristics
display(cleaned_data_DF)
```



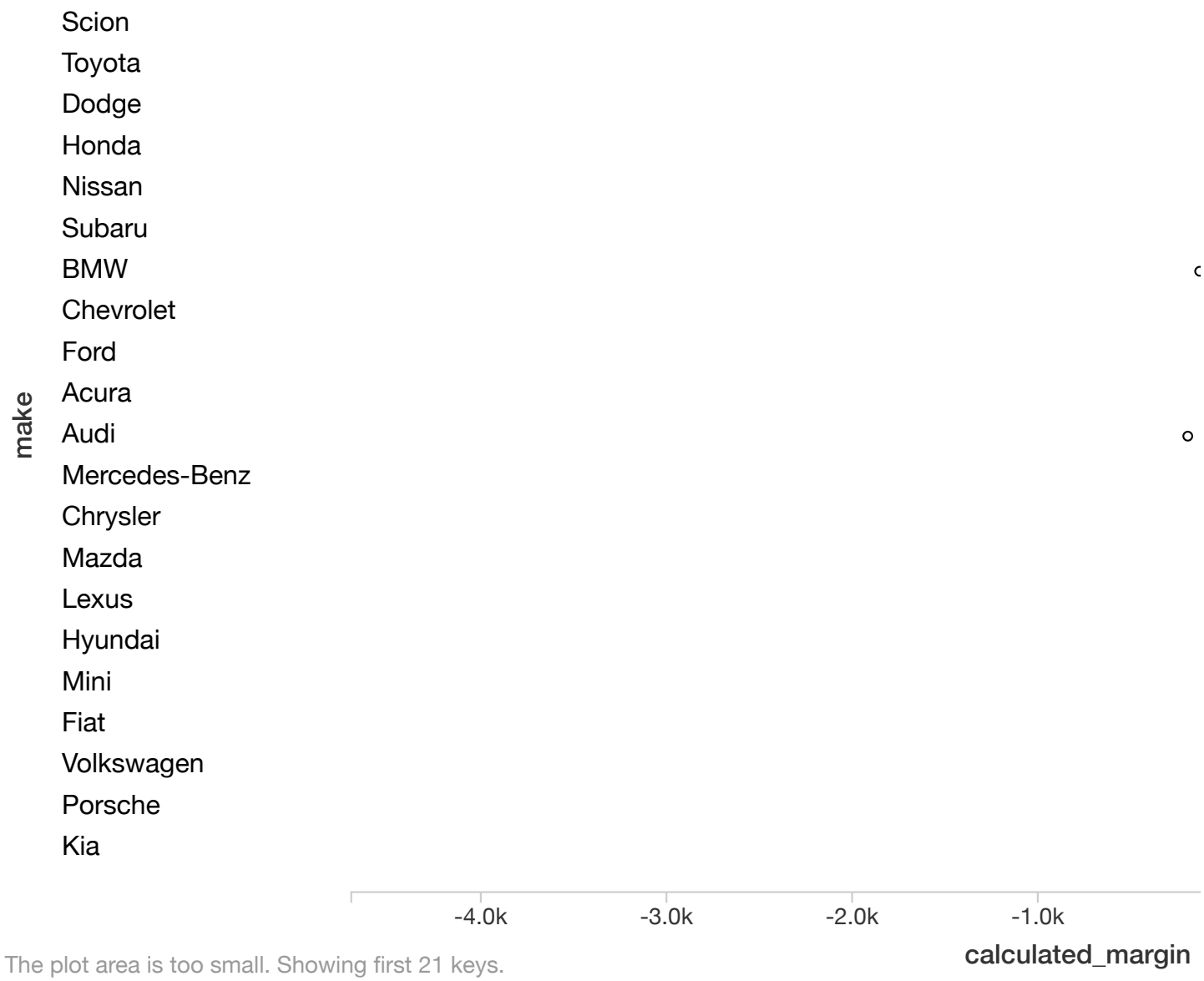
```
# Compare expected_margin against total_margin for all makes where expected_margin is not 0
# On average expected_margin is around 58% of total_margin, based on the data that we have
# For more sophisticated models we can compute this for each make/model/year/mileage, etc. segmented by location
display(non_zero_EM_DF.sort("make"))
```




```
# Looking our simple 58% model, plot average calculated_margin by brand
# Looking at the data, here are the makes that didn't have a successful sale through Shift: Buick, GMC, Hummer, Jaguar, Mitsubishi
display(cleaned_data_DF)
```



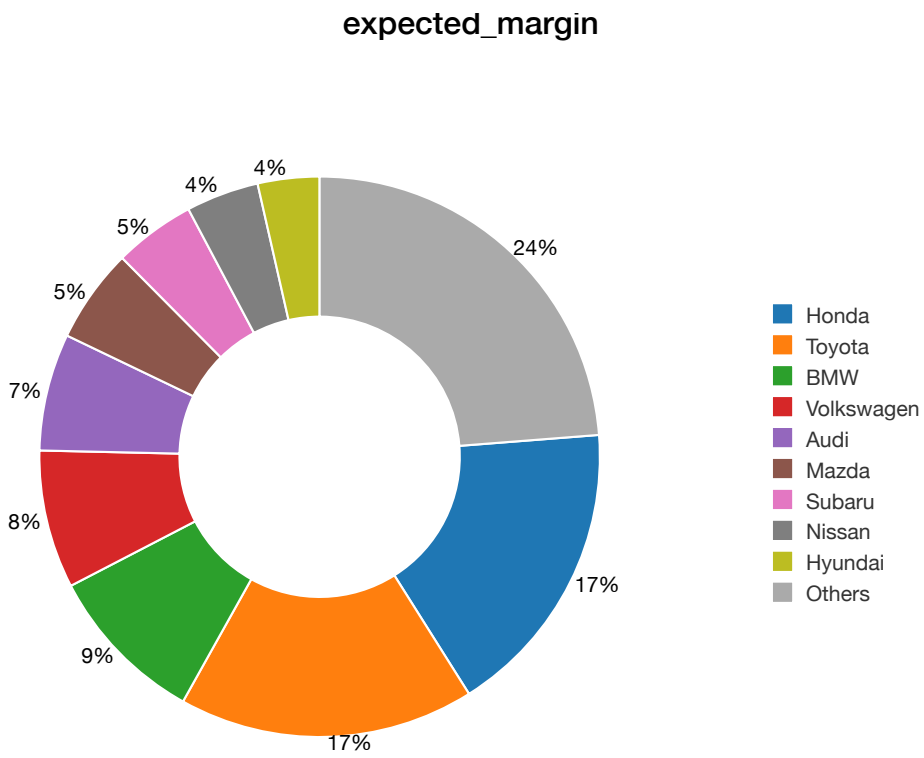

```
# Box plot calculated_margin by brand
# In general, linear regression will work better on makes with less variance in margin
display(cleaned_data_DF)
```



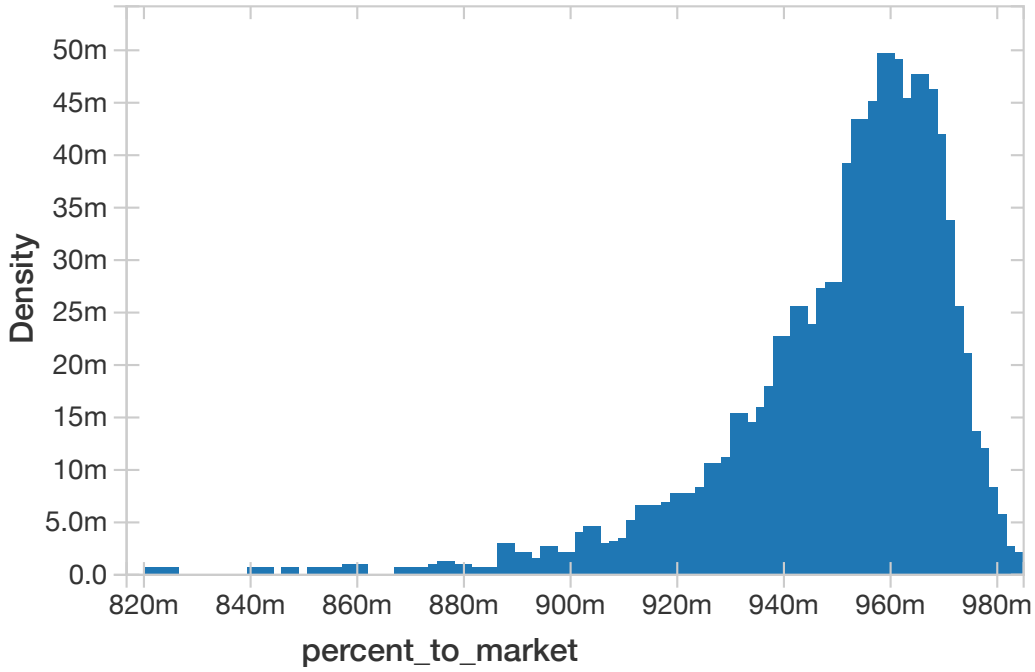
The plot area is too small. Showing first 21 keys.

Showing sample based on the first 1000 rows.

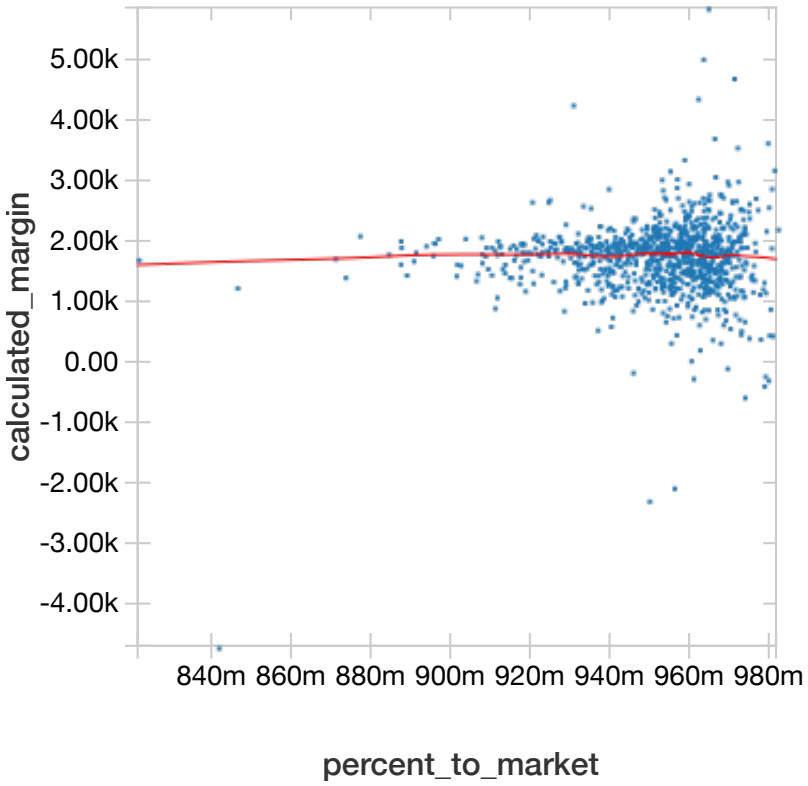
```
# Plot total expected/calculated margin by brand
# Top 3 brands are Honda, Toyota, BMW across all charts
display(cleaned_data_DF)
```




```
# Histogram of percent_to_market
display(cleaned_data_DF)
```

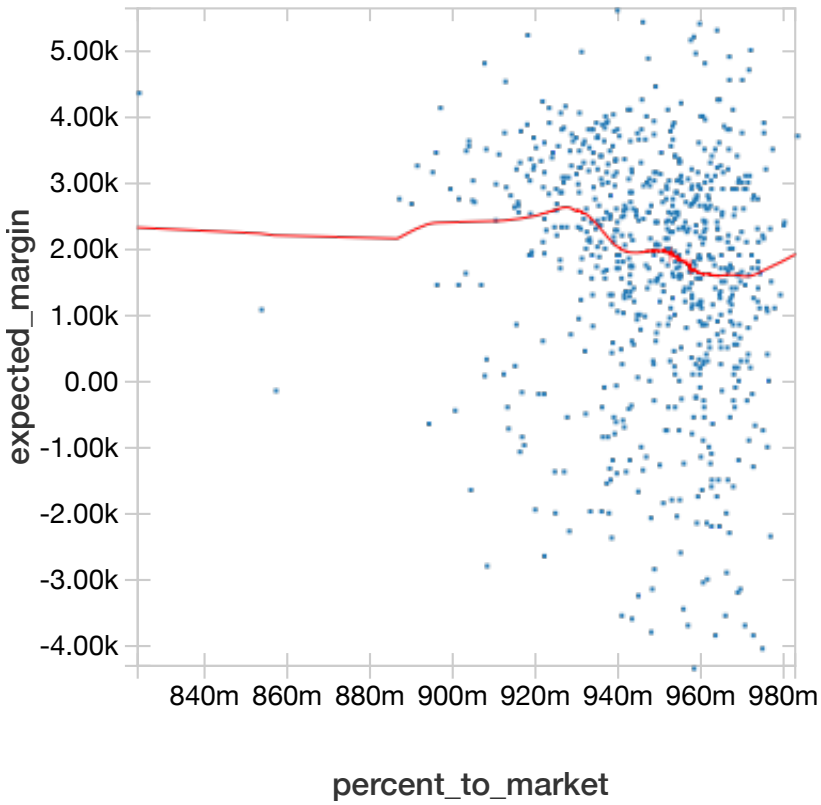


```
# Scatterplot percent_to_market vs calculated_margin
# It seems that the pricing team is trying to target a calculated_margin of around $2k for each car by adjusting the exact_list_price for each car
# The red line uses LOESS: https://en.wikipedia.org/wiki/Local_regression
# We use the default alpha of .3 for LOESS. "Useful values of the alpha smoothing parameter typically lie in the range 0.25 to 0.5 for most LOESS applications."
display(cleaned_data_DF)
```

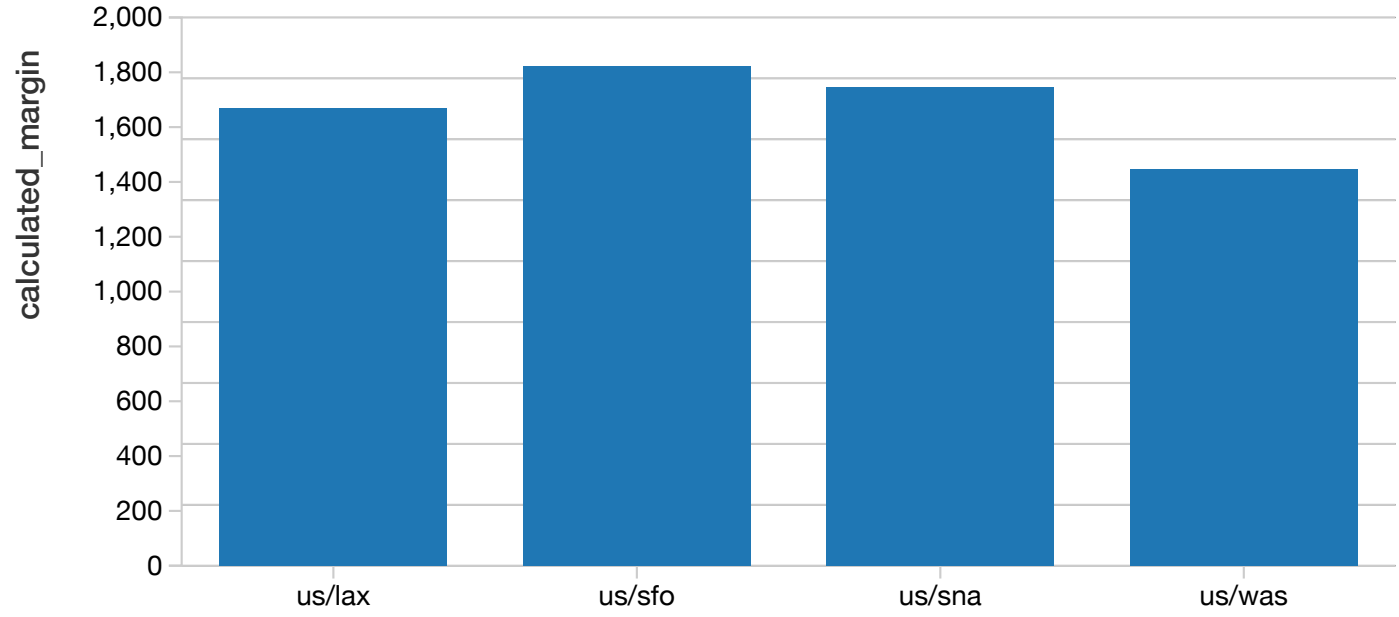


Showing sample based on the first 1000 rows.

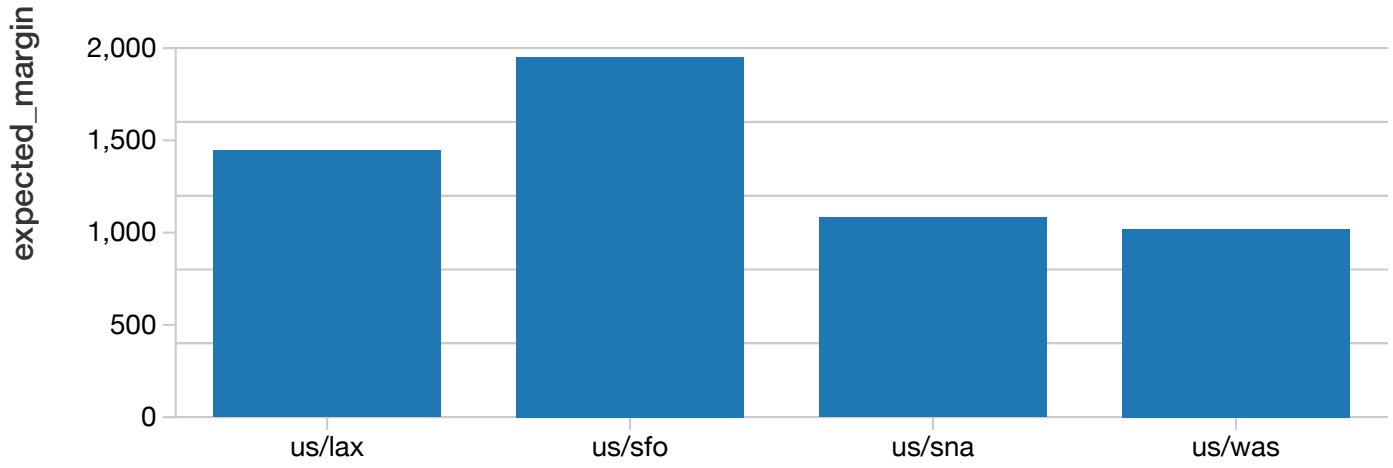
```
# Scatterplot percent_to_market vs expected_margin
# Variance of expected_margin is a lot higher, need to tighten it up through the use of predictive models
display(non_zero_EM_DF)
```



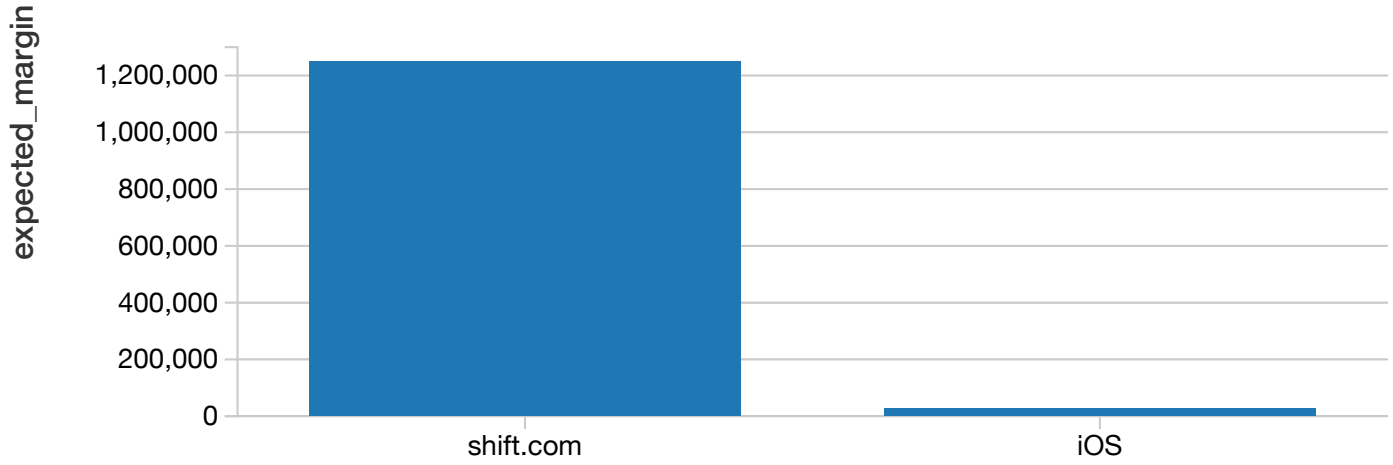

```
# Plot average calculated_margin by region_short_code
display(cleaned_data_DF)
```



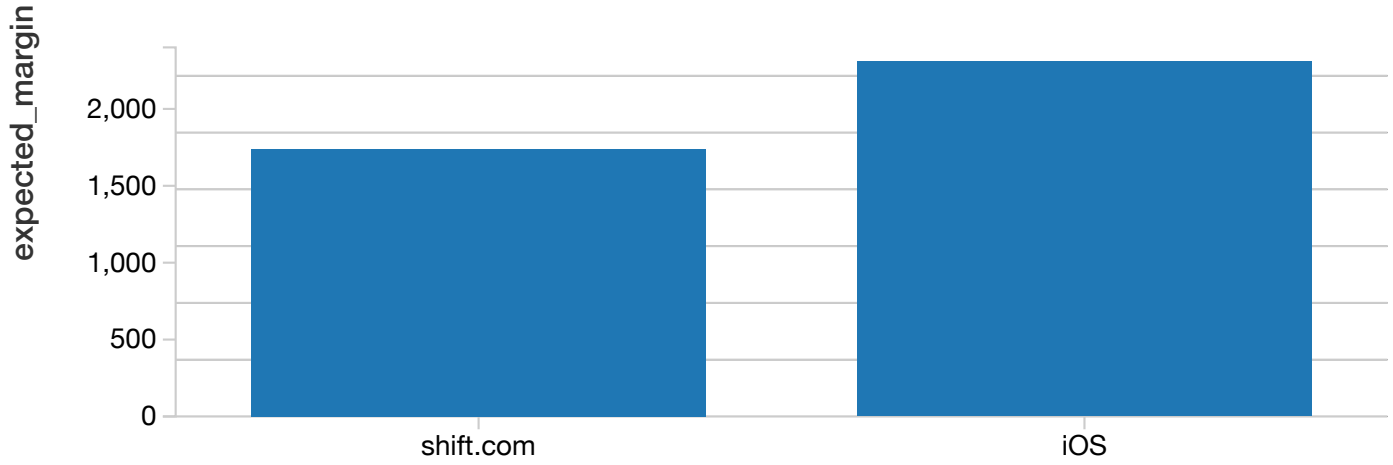

```
# Plot average expected_margin by region_short_code
# Need to increase margin in non-SF areas
display(non_zero_EM_DF.sort("region_short_code"))
```



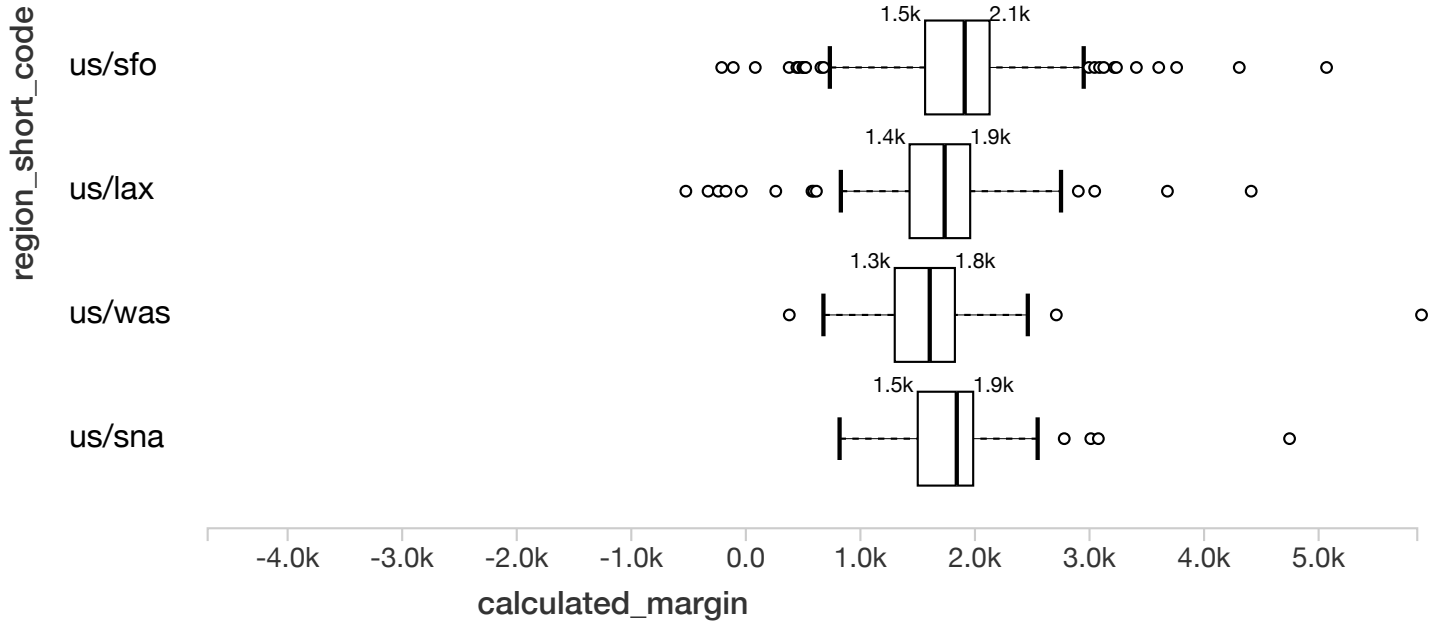

```
# Plot total expected_margin by seller_source
display(non_zero_EM_DF)
```




```
# Plot average expected_margin by seller_source
# Market Shift more heavily to smart phone users? Higher average expected_margin
display(non_zero_EM_DF)
```

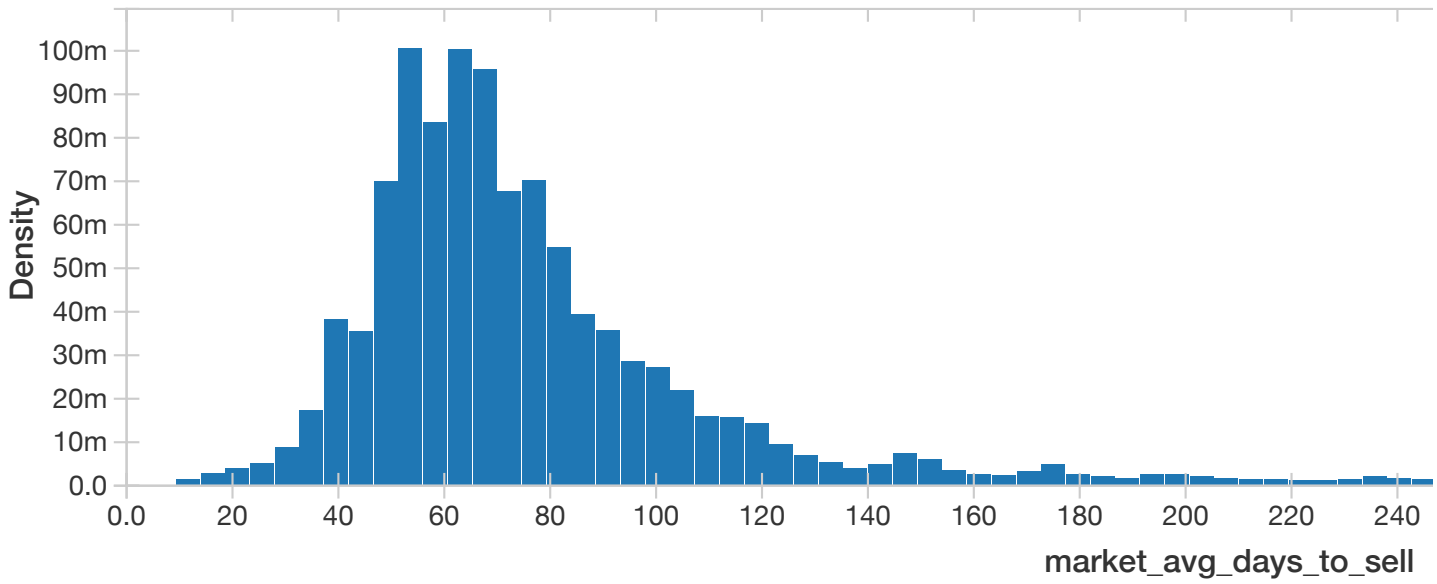



```
# Box plot region_short_code vs calculated_margin
# Higher variance in SFO/LAX, probably due to more cars being sold there
display(cleaned_data_DF)
```

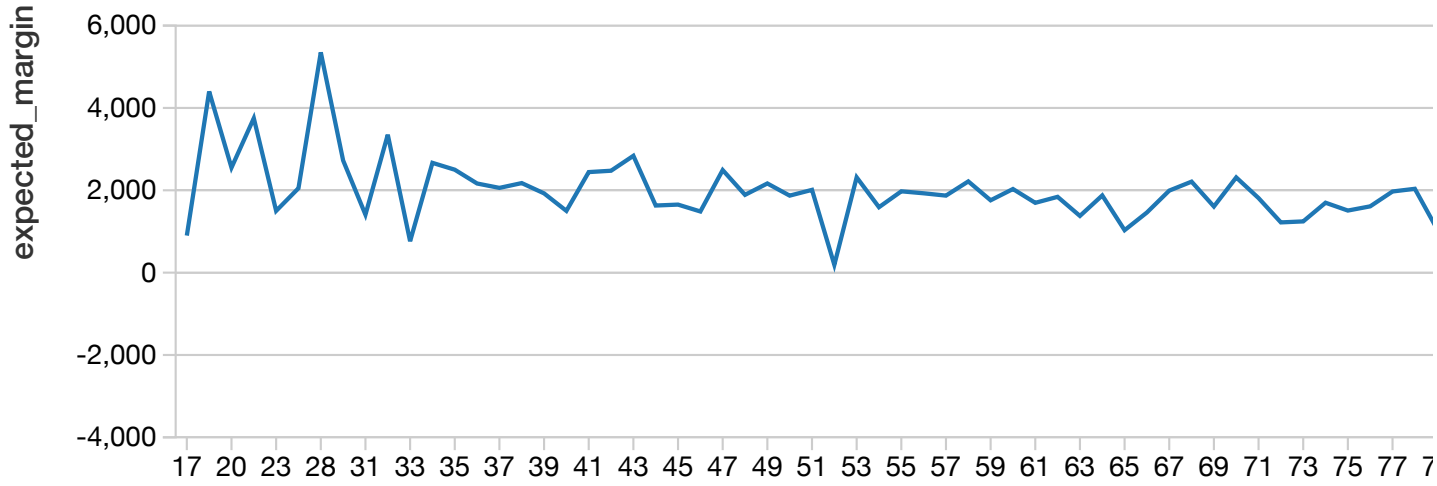


Showing sample based on the first 1000 rows.

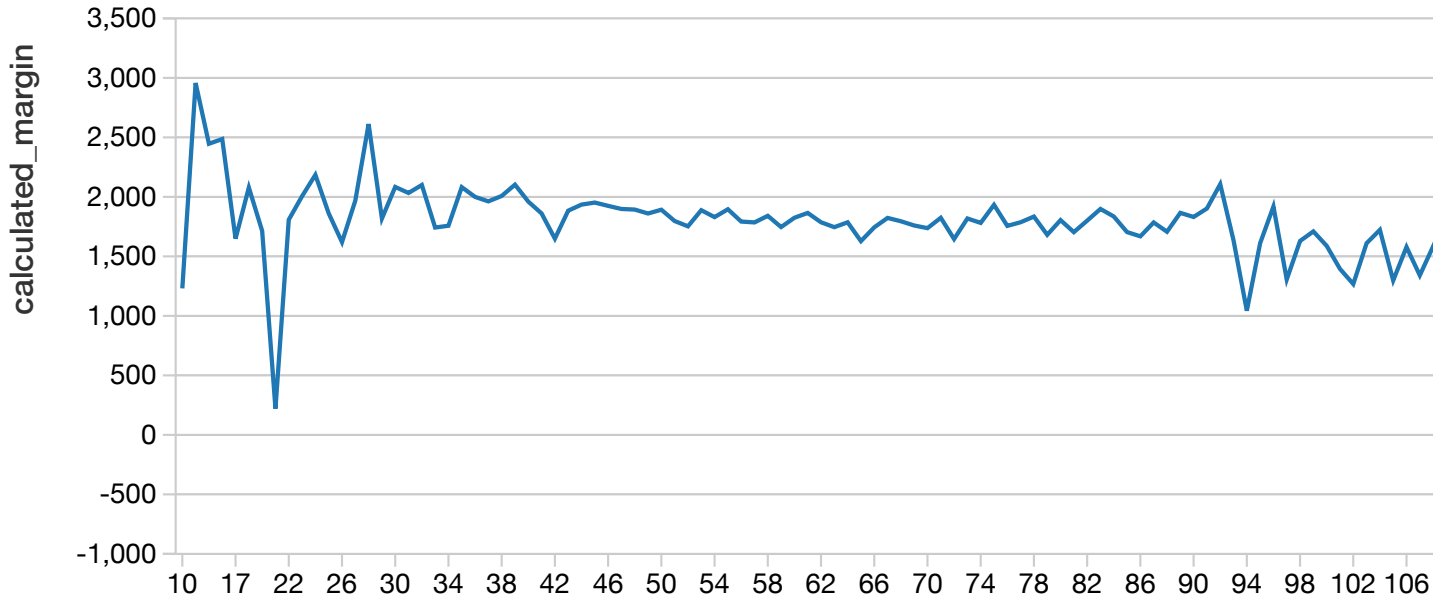
```
# Histogram market_avg_days_to_sell
# Potential outliers: Not sure that 0 and 1469 are correct?
display(cleaned_data_DF)
```



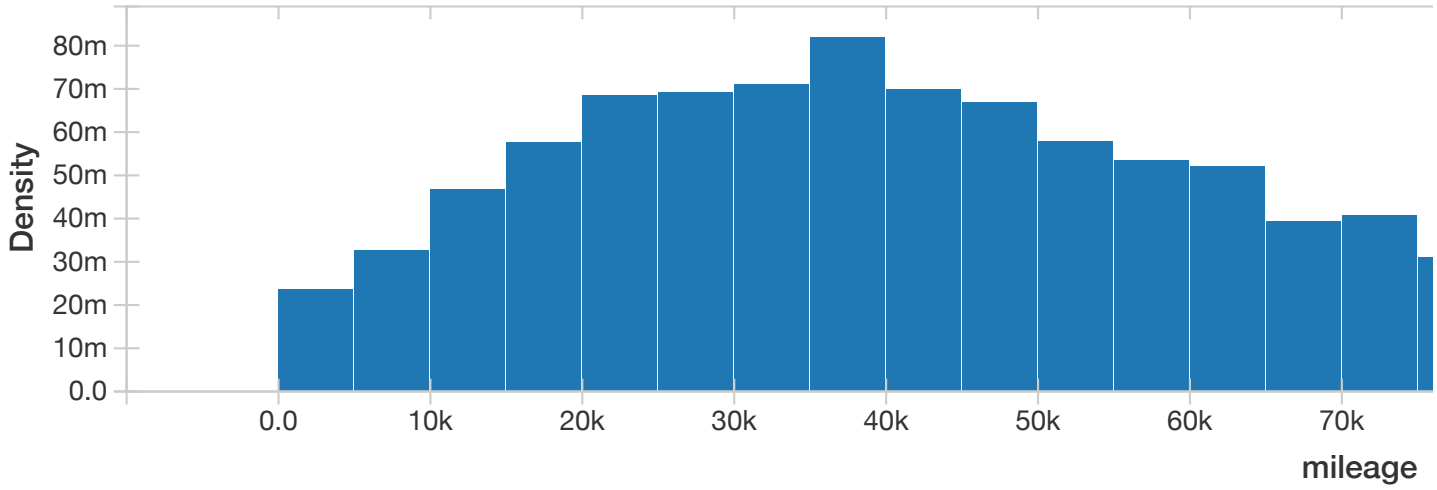

```
# Line chart market_avg_days_to_sell vs. average expected_margin
# More variance in expected_margin as market_avg_days_to_sell goes up
# average expected_margin also goes down as market_avg_days_to_sell goes up
display(non_zero_EM_DF.sort("market_avg_days_to_sell"))
```



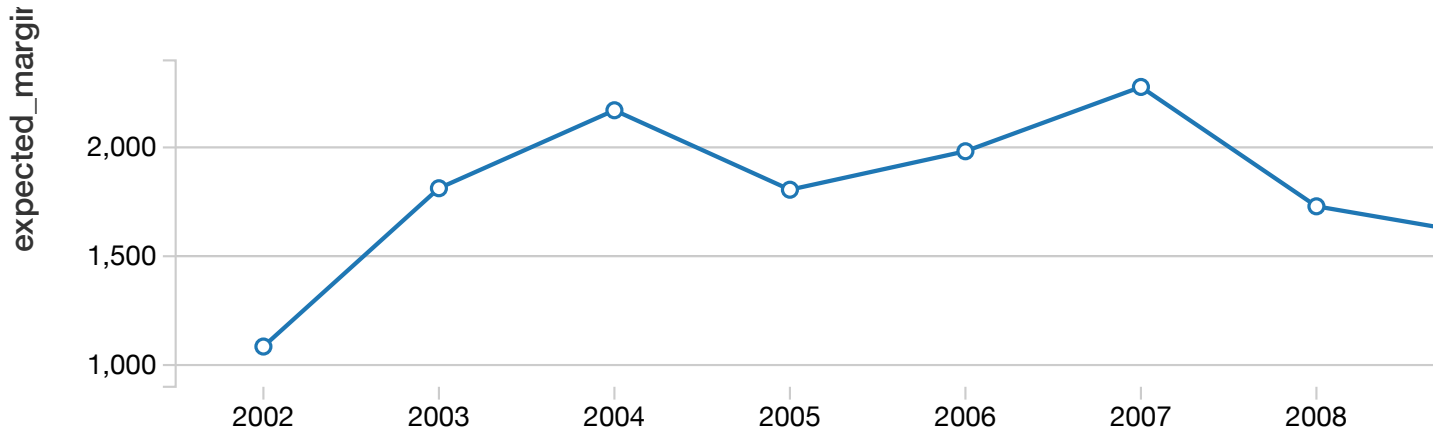

```
# Line chart market_avg_days_to_sell vs average calculated_margin
display(cleaned_data_DF)
```



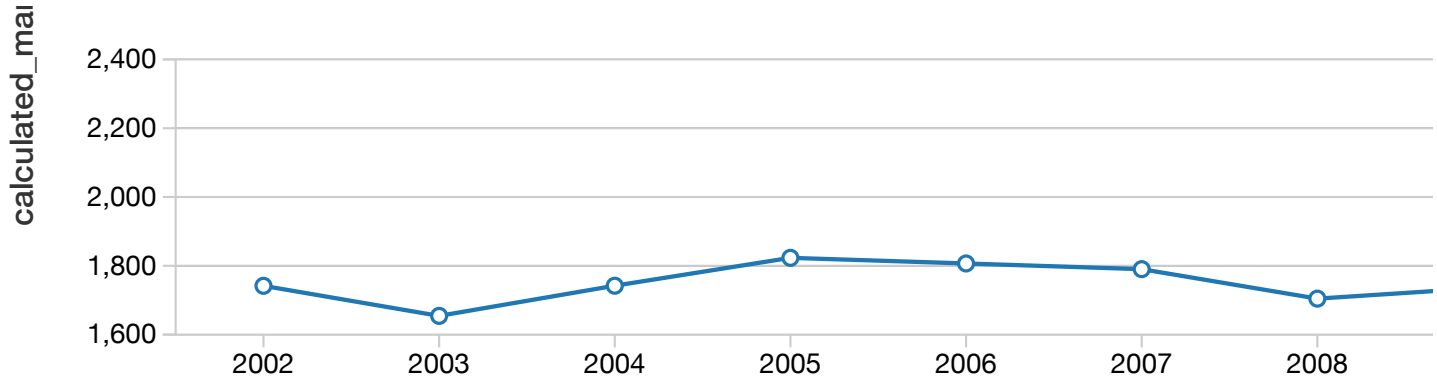

```
# Histogram of mileage
display(cleaned_data_DF)
```



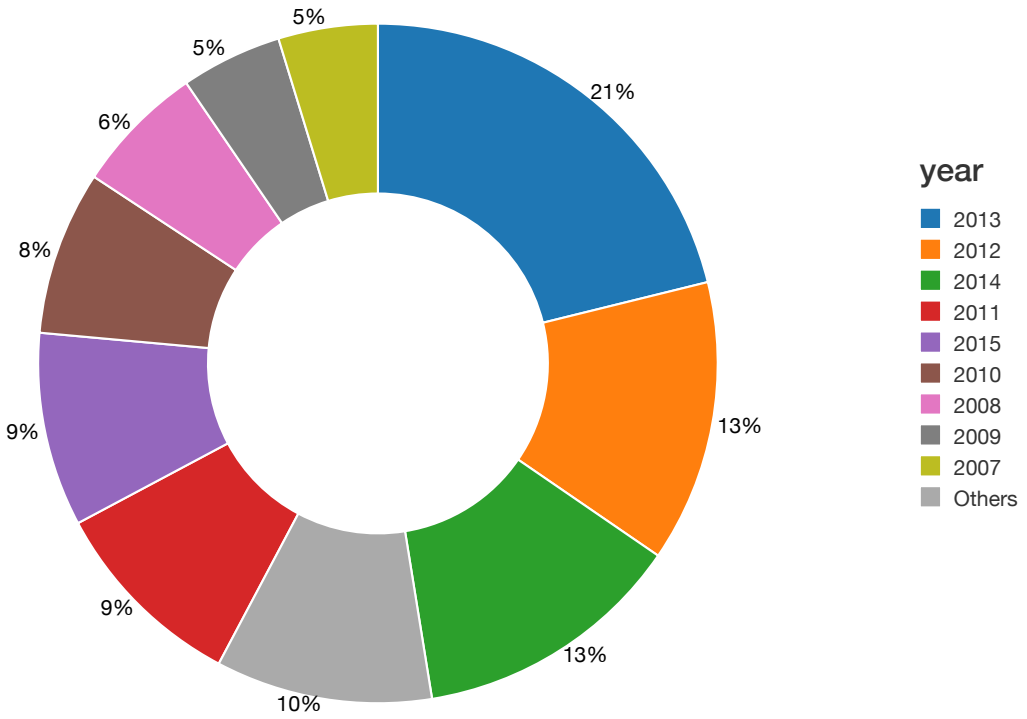

```
# Line chart average expected_margin by year
display(non_zero_EM_DF.sort("year"))
```



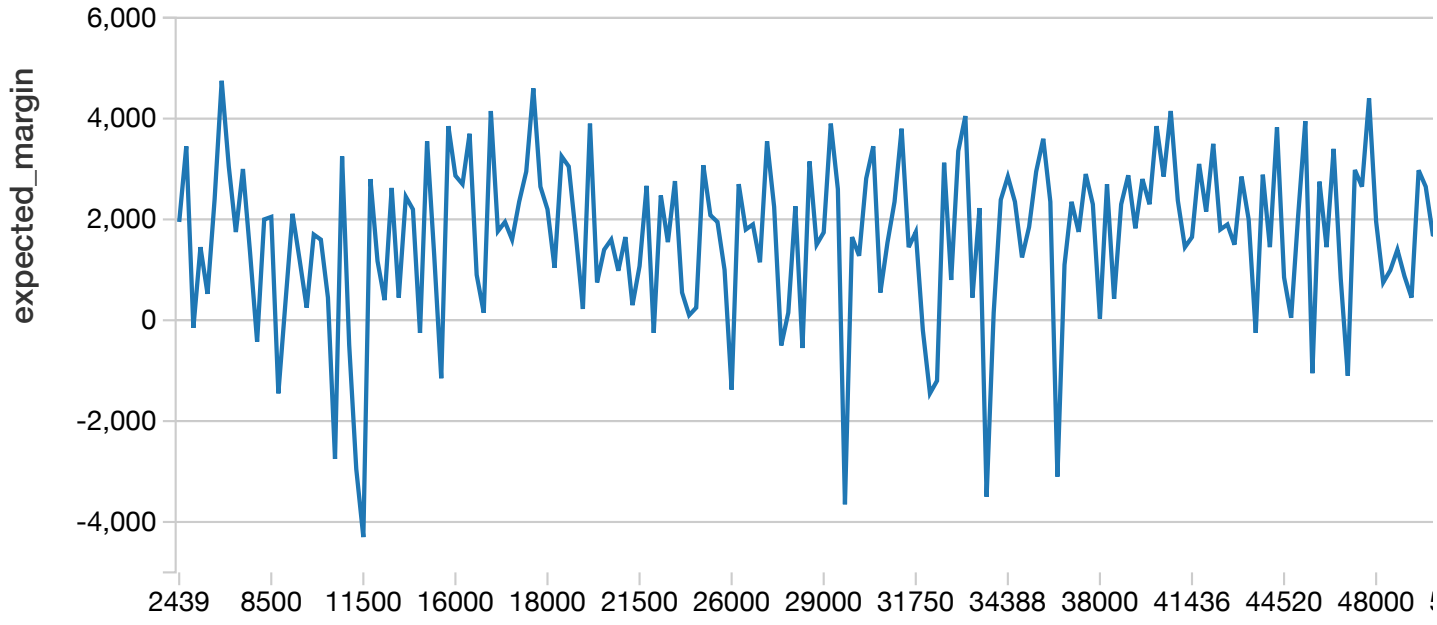

```
# Line chart average calculated_margin by year
# expected_margin should be higher for really new cars, but it's not
display(cleaned_data_DF)
```



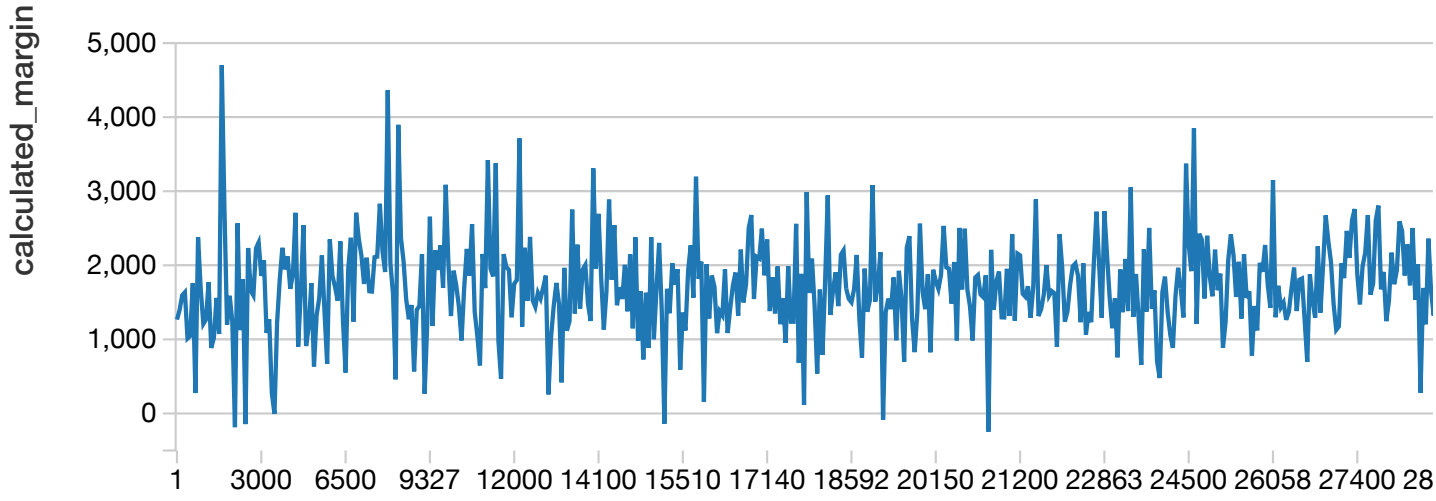

```
# Pie chart of year, breakdown of top years for sale on Shift
display(cleaned_data_DF)
```




```
# Line chart mileage vs average expected_margin
# No discernable effect on expected_margin by mileage
display(non_zero_EM_DF.sort("mileage"))
```

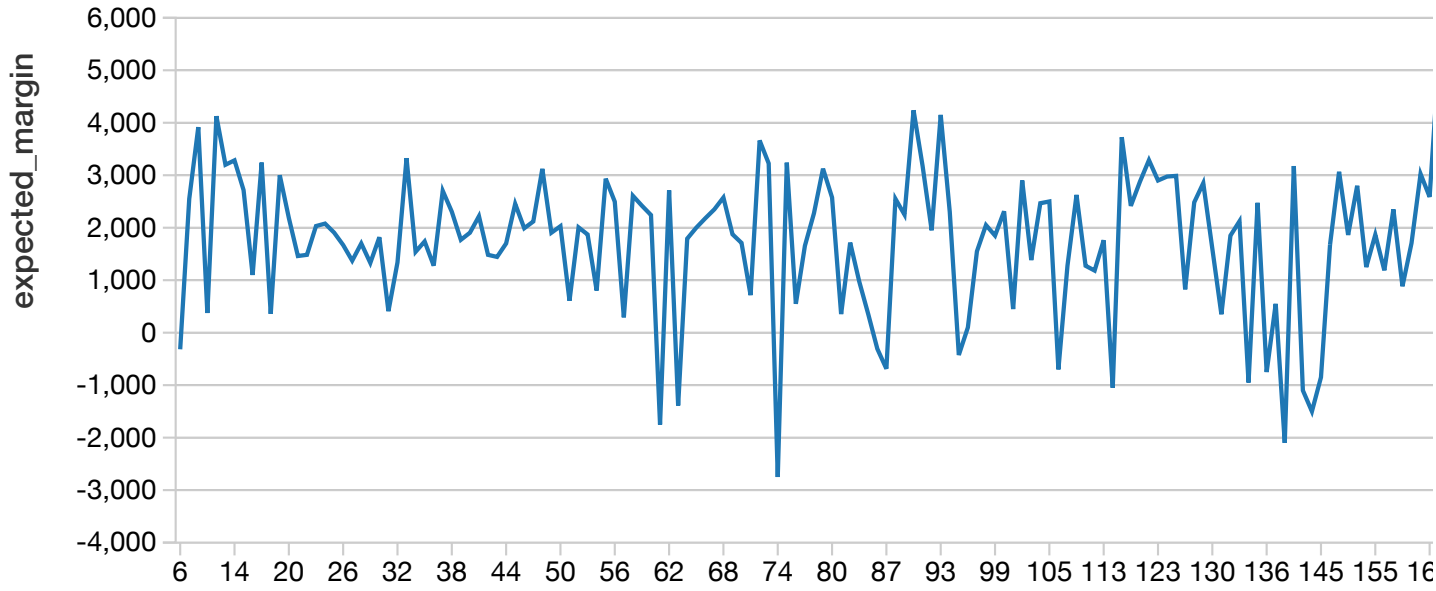



```
# Line chart mileage vs average calculated_margin
display(cleaned_data_DF)
```

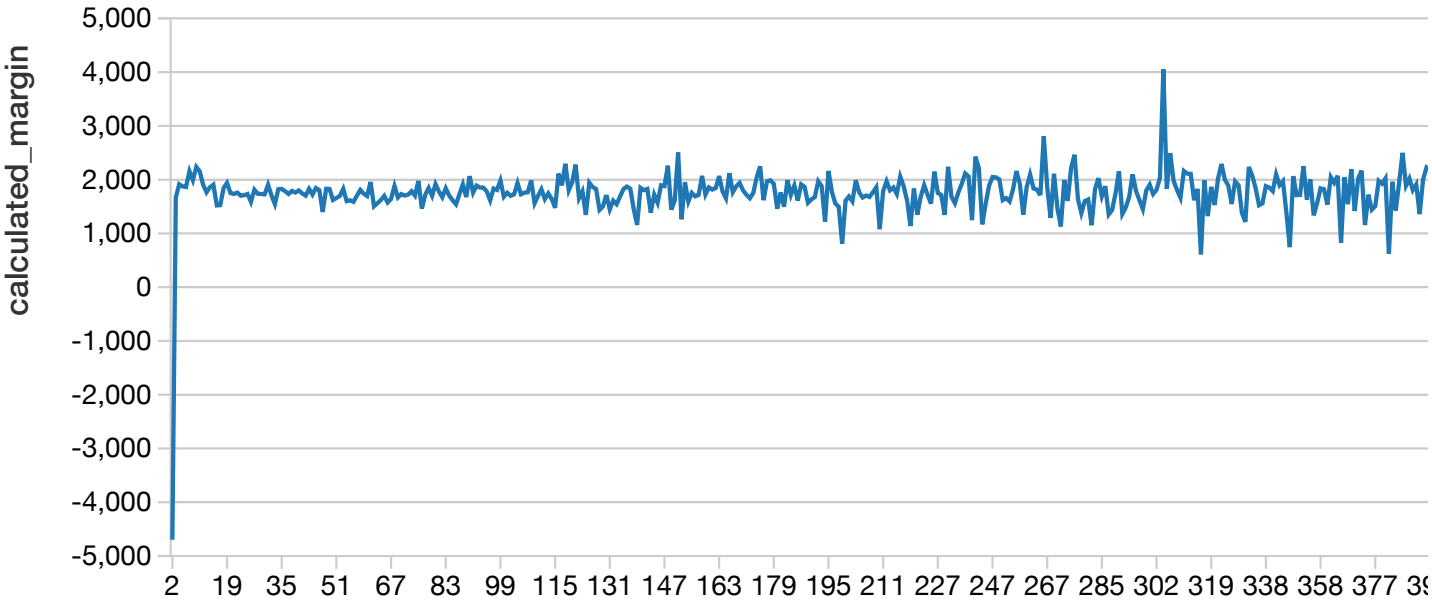


Showing the first 1000 rows.

```
# Line chart n_similar_vehicles_listed vs average expected_margin
# No discernable effect on expected_margin by n_similar_vehicles_listed
display(non_zero_EM_DF.sort("n_similar_vehicles_listed"))
```




```
# Line chart n_similar_vehicles_listed vs average calculated_margin
display(cleaned_data_DF)
```



Further Data Exploration

- For all charts above compare: expected_margin vs more advanced predictive model for Expected Margin
- Geographic analysis: plot statistics on maps (average expected_margin, most popular cars, etc.); Spark map plot broken - can't plot by zip code
- Time analysis: Look at supply and demand over time, see how buyers' tastes change over time depending on what's actually sold
- In general use more sophisticated plots: R (ggplot, googleVis, etc.) or python (matplotlib, seaborn); you can use different languages in the same Spark notebook
- Create Data Refinery: dashboard that creates customizable reports, enables cohort analysis, used by Data Analysts

Currently Available Research (CAR)

- **Predicting the Price of Used Cars using Machine Learning Techniques** by Pudaruth
- **Predicting the Price of Second-hand Cars using Artificial Neural Networks** by Peerun, Chummun, and Pudaruth
- **Support Vector Regression Analysis for Price Prediction in a Car Leasing Application** by Listiani
- http://www.datasciencecentral.com/profiles/blogs/predicting-car-prices-part-1-linear-

regression (http://www.datasciencecentral.com/profiles/blogs/predicting-car-prices-part-1-linear-regression)
- https://www.kaggle.com/juliencs/house-prices-advanced-regression-techniques/a-study-on-regression-applied-to-the-ames-dataset (https://www.kaggle.com/juliencs/house-prices-advanced-regression-techniques/a-study-on-regression-applied-to-the-ames-dataset)
- https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models (https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models)
- https://www.kaggle.com/c/house-prices-advanced-regression-techniques/kernels (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/kernels)

CAR Models

Comparable Performance

- (Multivariate) Linear Regression Analysis
- Decision Trees / Random Forest / Gradient Boosted Tree
- K-nearest neighbors
- Artificial Neural Networks / Multi-layer perceptron: used before to predict commodity prices
- Naive Bayes: categorical range output
- Neuro-fuzzy knowledge based system
- Optimal Distribution of Auction Vehicles (ODAV)

Best Performance

- Support Vector Machine Regression (with genetic algorithm to find optimal SVM parameters)
  - Some predicted values far away from actual prices for higher priced cars
  - This method is also a lot more complex than simple linear regression: requires a hyperparameter grid search/genetic algo/training time to find the optimal SVM parameters.
  - It may be too complex to put into production? Try Spark ParamGrid function

CAR Factors

Primary

- Age/Year
- Make
- Model
- Mileage
- Location
- Country of Manufacture
- Horsepower
- Fuel Economy

Secondary

- Physical state
- Type of fuel
- Interior style
- Braking System
- Acceleration
- Volume of cylinders
- Safety Index
- Size / weight of car
- Number of doors
- Paint color
- Consumer reviews
- Awards
- Cruise control
- Automatic/manual transmission
- Previous owner corporation or individual
- Previous accidents
- Other: A/C, sound, power steering, wheels, GPS, etc.

Predictive Modeling Outline

- Practically speaking, a Data Scientist's job is: Data Gathering / Ingestion --> ETL --> Exploratory Data Analysis --> Model Selection --> Feature Extraction (A/B Testing) --> Parameter Tuning --> Model Scoring --> Model Update over time based on feedback
  - *Ultimately you need to automate this process*
  - datarobot.com, skytree.net, h2o.ai?

- Predictive Modeling attempted with 1. scikit-learn then 2. Spark ML: pyspark.ml chosen, it runs on dataframes on Spark 2.0
- Multivariate Linear Regression tried first. Assumes normal distribution of variables. Linear Regression also assumes that the relationship between the dependent variable and binary independent variables are homoscedastic. This means that the residuals are approximately equal for all predicted dependent variable scores.
- For Linear Regression tried 1. Standard 2. Ridge 3. Lasso 4. ElasticNet
- Then tried Gradient Boosting Tree Regression.
- In terms of feature selection, tried: 1. region, make, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed then 2. region, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed for Honda Civic (most popular Make & Model)

Feature Selection

Categorical (Nominal) Features

- region_short_code: string
- location_zip: int, remove, not enough consistent information, not good predictor variable, do Cohort Analysis by zip later
- seller_source: string, remove, already explored through EDA
- make: string
- model: string, not enough data for each model, do Cohort Analysis later

Numerical Features

- year: int
- mileage: int
- percent_to_market: double, let's not use this as a feature because this is a not a property of the car itself, it's an arbitrary value created by Shift
- market_avg_days_to_sell: int
- n_similar_vehicles_listed: int

Dependent Variable

- expected_margin: int

```
%sql
SELECT make, count(*) as NUM FROM (
  SELECT make FROM non_zero_expected_margin
) GROUP BY make ORDER BY NUM DESC
```

make
Honda
Toyota
BMW
Volkswagen

```
# Create a dataframe for linear regression
# Log transform of the skewed numerical features to lessen impact of outliers
# As a general rule of thumb, a skewness with an absolute value > 0.5 is considered at least moderately skewed
# Apply log transforms to variables with skew > .5
# Don't apply to expected_margin as it makes the skew worse; also expected_margin is already somewhat a derivative of price
#
http://oak.ucc.nau.edu/rh232/courses/eps625/handouts/data%20transformation%20handout.pdf
#LR_DF = sqlContext.sql("SELECT region_short_code, location_zip, seller_source, make, CONCAT(make, '_', model) AS model, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM non_zero_expected_margin")

#LR_DF = sqlContext.sql("SELECT region_short_code, seller_source, make, CONCAT(make, '_', model) AS model, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM non_zero_expected_margin")

#LR_DF = sqlContext.sql("SELECT region_short_code, location_zip, seller_source, make, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM non_zero_expected_margin")

#LR_DF = sqlContext.sql("SELECT region_short_code, make, CONCAT(make, '_', model) AS model, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM non_zero_expected_margin")

#LR_DF = sqlContext.sql("SELECT region_short_code, seller_source, make, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM non_zero_expected_margin")

#LR_DF = sqlContext.sql("SELECT region_short_code, make, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM non_zero_expected_margin")
```

```
# Feature Set 1
##LR_DF = sqlContext.sql("SELECT region_short_code, make, year, mileage,
LOG10(market_avg_days_to_sell) AS market_avg_days_to_sell,
LOG10(n_similar_vehicles_listed) AS n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin")

#LR_DF = sqlContext.sql("SELECT region_short_code, make, year, mileage,
LOG10(market_avg_days_to_sell) AS market_avg_days_to_sell,
LOG10(n_similar_vehicles_listed) AS n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin")

#LR_DF = sqlContext.sql("SELECT region_short_code, make, year, mileage,
market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin WHERE make='Honda' or make='Toyota' or make='BMW' or
make='Volkswagen' or make='Audi' or make='Nissan' or make='Lexus' or make='Hyundia' or
make='Ford' or make='Subaru' or make='Mazda'")

#LR_DF = sqlContext.sql("SELECT region_short_code, make, year, mileage,
LOG10(market_avg_days_to_sell) AS market_avg_days_to_sell,
LOG10(n_similar_vehicles_listed) AS n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin WHERE make='Honda' or make='Toyota' or make='BMW' or
make='Volkswagen' or make='Audi' or make='Nissan' or make='Lexus' or make='Hyundia' or
make='Ford' or make='Subaru' or make='Mazda'")

#LR_DF = sqlContext.sql("SELECT region_short_code, year, mileage,
market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin WHERE make='Honda'")

#LR_DF = sqlContext.sql("SELECT region_short_code, year, mileage,
LOG10(market_avg_days_to_sell) AS market_avg_days_to_sell,
LOG10(n_similar_vehicles_listed) AS n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin WHERE make='Honda'")

# Feature Set 2
LR_DF = sqlContext.sql("SELECT region_short_code, year, mileage,
LOG10(market_avg_days_to_sell) AS market_avg_days_to_sell,
LOG10(n_similar_vehicles_listed) AS n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin WHERE make='Honda' AND model='Civic'")

# even worse results than Honda Civic; need more data
#LR_DF = sqlContext.sql("SELECT region_short_code, year, mileage,
LOG10(market_avg_days_to_sell) AS market_avg_days_to_sell,
LOG10(n_similar_vehicles_listed) AS n_similar_vehicles_listed, expected_margin FROM
non_zero_expected_margin WHERE make='BMW' AND model='3 Series'")

LR_DF.createOrReplaceTempView("LR_DF")
display(LR_DF)
```

region_short_code	year	mileage	market_avg_days_to_sell
us/sfo	2008	80000	1.6334684555795864
us/sfo	2006	123000	1.6434526764861874
us/sfo	2007	65000	1.662757831681574
us/sfo	2006	99500	1.7853298350107671
us/sfo	2012	43000	1.7853298350107671
us/sfo	2006	94482	1.7781512503836436

```
# Import packages
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV, ElasticNetCV
from sklearn.metrics import mean_squared_error, make_scorer
from scipy.stats import skew
from ggplot import *
import seaborn as sns

train = LR_DF.toPandas() # Spark to pandas dataframe
#train.shape[0]
train.head()

Out[33]:
  region_short_code  year  mileage  market_avg_days_to_sell  \
0            us/sfo  2008    80000                 1.633468
1            us/sfo  2006   123000                 1.643453
2            us/sfo  2007    65000                 1.662758
3            us/sfo  2006    99500                 1.785330
4            us/sfo  2012    43000                 1.785330

   n_similar_vehicles_listed  expected_margin
0                   2.414973             3025
1                   1.949390             3725
2                   2.303196             2975
3                   2.240549             2900
4                   2.793790             1950
```

```
# Calculate skewness
print('year:', skew(train['year']))
print('mileage:', skew(train['mileage']))
print('market_avg_days_to_sell:', skew(train['market_avg_days_to_sell'])) # log
transform
print('n_similar_vehicles_listed:', skew(train['n_similar_vehicles_listed'])) # log
transform
print('expected_margin:', skew(train['expected_margin'])) # log transform
```

```
('year:', -0.012401218344636013)
('mileage:', 0.24885296384543934)
('market_avg_days_to_sell:', 1.6005361491194643)
('n_similar_vehicles_listed:', 0.24250349982025943)
('expected_margin:', -0.7723347844745558)
```

```
# One Hot Encoding of Categorical features
# http://stackoverflow.com/questions/11587782/creating-dummy-variables-in-pandas-for-
python
# Must re-add in dropped dummy variable later, or drop non-useful dummy variable

dummies = pd.get_dummies(train['region_short_code']).rename(columns=lambda x:
'region_' + str(x))
train1 = pd.concat([train, dummies], axis=1)
train1 = train1.drop(['region_short_code'], axis=1)
train1 = train1.drop(['region_us/sna'], axis=1)
# dummy variable trap, mutually exclusive and exhaustive, drop one dummy variable

#dummies = pd.get_dummies(train1['location_zip']).rename(columns=lambda x: 'zip_' +
str(x))
#train1 = pd.concat([train1, dummies], axis=1)
#train1 = train1.drop(['location_zip'], axis=1)
#train1 = train1.drop(['zip_20001'], axis=1)
# dummy variable trap, mutually exclusive and exhaustive, drop one dummy variable

#dummies = pd.get_dummies(train1['seller_source']).rename(columns=lambda x: 'seller_'
+ str(x))
#train1 = pd.concat([train1, dummies], axis=1)
#train1 = train1.drop(['seller_source'], axis=1)
#train1 = train1.drop(['seller_iOS'], axis=1)
# dummy variable trap, mutually exclusive and exhaustive, drop one dummy variable

#dummies = pd.get_dummies(train1['model']).rename(columns=lambda x: 'model_' + str(x))
#train1 = pd.concat([train1, dummies], axis=1)
#train1 = train1.drop(['model'], axis=1)
#train1 = train1.drop(['model_Subaru_BRZ'], axis=1)
# dummy variable trap, mutually exclusive and exhaustive, drop one dummy variable

#dummies = pd.get_dummies(train1['make']).rename(columns=lambda x: 'make_' + str(x))
#trainOHE = pd.concat([train1, dummies], axis=1)
#trainOHE = trainOHE.drop(['make'], axis=1)
#trainOHE = trainOHE.drop(['make_Lincoln'], axis=1)
# dummy variable trap, mutually exclusive and exhaustive, drop one dummy variable

trainOHE=train1
list(trainOHE.columns.values)
#trainOHE.head()

Out[35]:
['year',
 'mileage',
 'market_avg_days_to_sell',
 'n_similar_vehicles_listed',
```

```
 'expected_margin',
 'region_us/lax',
 'region_us/sfo',
 'region_us/was']

# Find most important features relative to target
# We can also create Spark LOESS scatterplots to dive into the correlation between
each feature and the expected_margin
print("Find most important features relative to target")
corr = trainOHE.corr()
corr.sort(["expected_margin"], ascending = False, inplace = True)
print(corr.expected_margin)

Find most important features relative to target
expected_margin              1.000000
region_us/was                0.124794
market_avg_days_to_sell      0.117641
region_us/sfo                0.044792
n_similar_vehicles_listed   -0.025313
mileage                     -0.055816
region_us/lax               -0.110959
year                        -0.123218
Name: expected_margin, dtype: float64

# Partition the dataset in train + validation sets
y = trainOHE.expected_margin
train = trainOHE.drop(['expected_margin'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(train, y, test_size = 0.3,
random_state = 0)
print("X_train : " + str(X_train.shape))
print("X_test : " + str(X_test.shape))
print("y_train : " + str(y_train.shape))
print("y_test : " + str(y_test.shape))
X_train.head()

X_train : (32, 7)
X_test : (15, 7)
y_train : (32,)
y_test : (15,)
Out[37]:
    year  mileage  market_avg_days_to_sell  n_similar_vehicles_listed  \
45  2009    46510                 1.698970                   1.633468
26  2013    42000                 1.698970                   3.396548
15  2013    27400                 1.707570                   3.431846
25  2013    13000                 1.845098                   2.442480
16  2014     9175                 1.785330                   2.447158

    region_us/lax  region_us/sfo  region_us/was
```

```
45              0              1              0
26              0              0              1
15              1              0              0
25              0              1              0
16              1              0              0
```

```
# Standardize numerical features
num_feat = trainOHE[['year', 'mileage', 'market_avg_days_to_sell',
'n_similar_vehicles_listed']]
numerical_features = num_feat.select_dtypes(exclude = ["object"]).columns
stdSc = StandardScaler()
X_train.loc[:, numerical_features] = stdSc.fit_transform(X_train.loc[:,
numerical_features])
X_test.loc[:, numerical_features] = stdSc.transform(X_test.loc[:, numerical_features])
```

```
X_train.head()
```

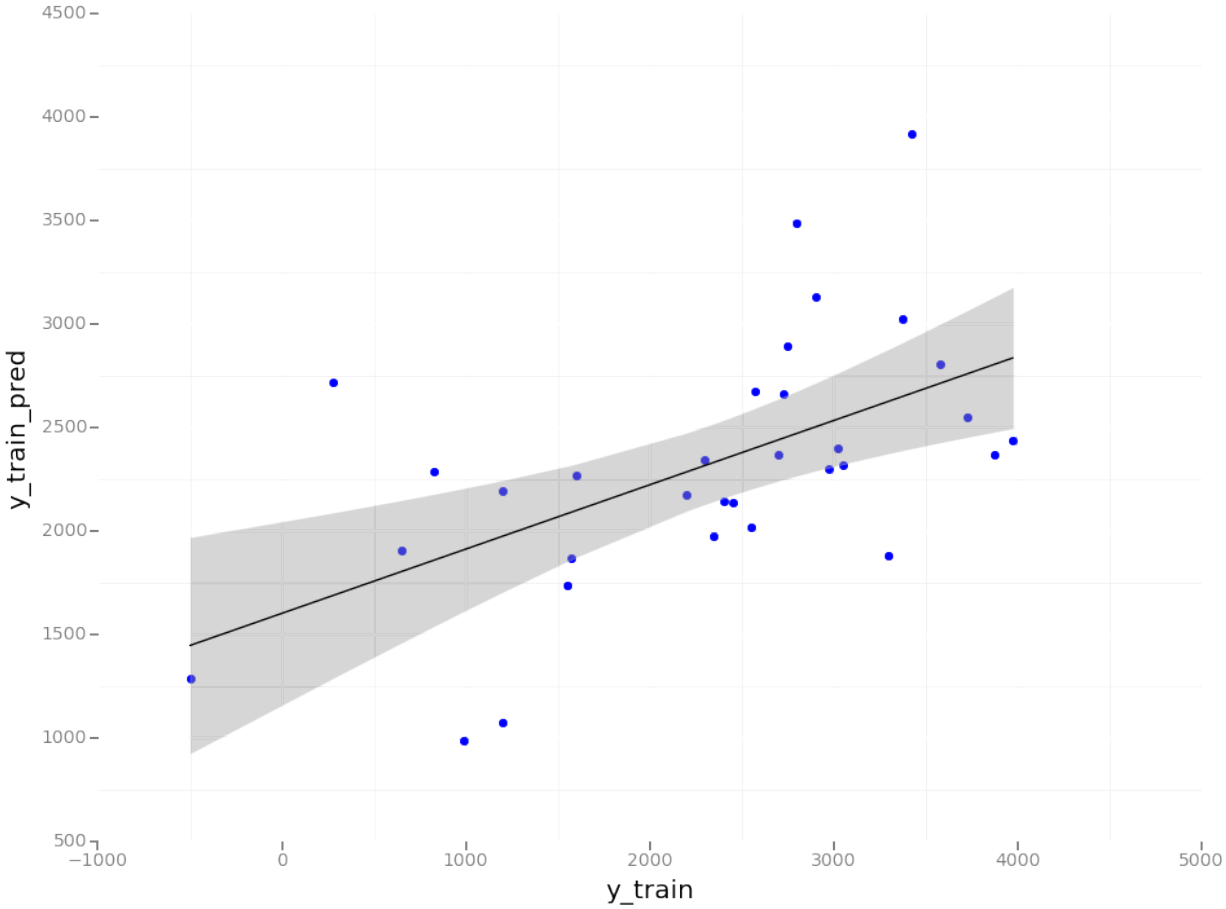
```
/databricks/python/local/lib/python2.7/site-packages/pandas/core/indexing.py:389: Sett
ingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_index,col_indexer] = value instead
  self.obj[item] = s
Out[38]:
        year   mileage  market_avg_days_to_sell  n_similar_vehicles_listed  \
45 -0.010328 -0.505863                -0.350907                  -1.237434
26  1.311650 -0.635785                -0.350907                   2.176524
15  1.311650 -1.056374                -0.283068                   2.244873
25  1.311650 -1.471201                 0.801776                   0.329104
16  1.642145 -1.581389                 0.330314                   0.338162

    region_us/lax  region_us/sfo  region_us/was
45              0              1              0
26              0              0              1
15              1              0              0
25              0              1              0
16              1              0              0
```

```
# Define error measure for official scoring : RMSE
# 3 fold cross-validation
scorer = make_scorer(mean_squared_error, greater_is_better = False)

def rmse_cv_train(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y_train, scoring = scorer, cv = 3))
    return(rmse)

def rmse_cv_test(model):
    rmse= np.sqrt(-cross_val_score(model, X_test, y_test, scoring = scorer, cv = 3))
    return(rmse)
```

```
# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)

# Look at predictions on training and validation set
print("RMSE on Training set :", rmse_cv_train(lr).mean())
print("RMSE on Test set :", rmse_cv_test(lr).mean())

# Look at fit, R^2 between 0 and 1, 1 best fit
print("R^2 of training set:", lr.score(X_train, y_train))

# Look at coefficients
coefs = pd.Series(lr.coef_, index = X_train.columns)
coefs.sort()
print(coefs)

# Apply model
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
```

```
('RMSE on Training set :', 1415.6720911589109)
('RMSE on Test set :', 1256.0856240679768)
('R^2 of training set:', 0.31050798446675387)
year                         -681.715807
mileage                      -159.543005
n_similar_vehicles_listed     148.315335
market_avg_days_to_sell       349.580243
region_us/lax                 918.878170
region_us/was                1467.806124
region_us/sfo                1645.064227
dtype: float64
```

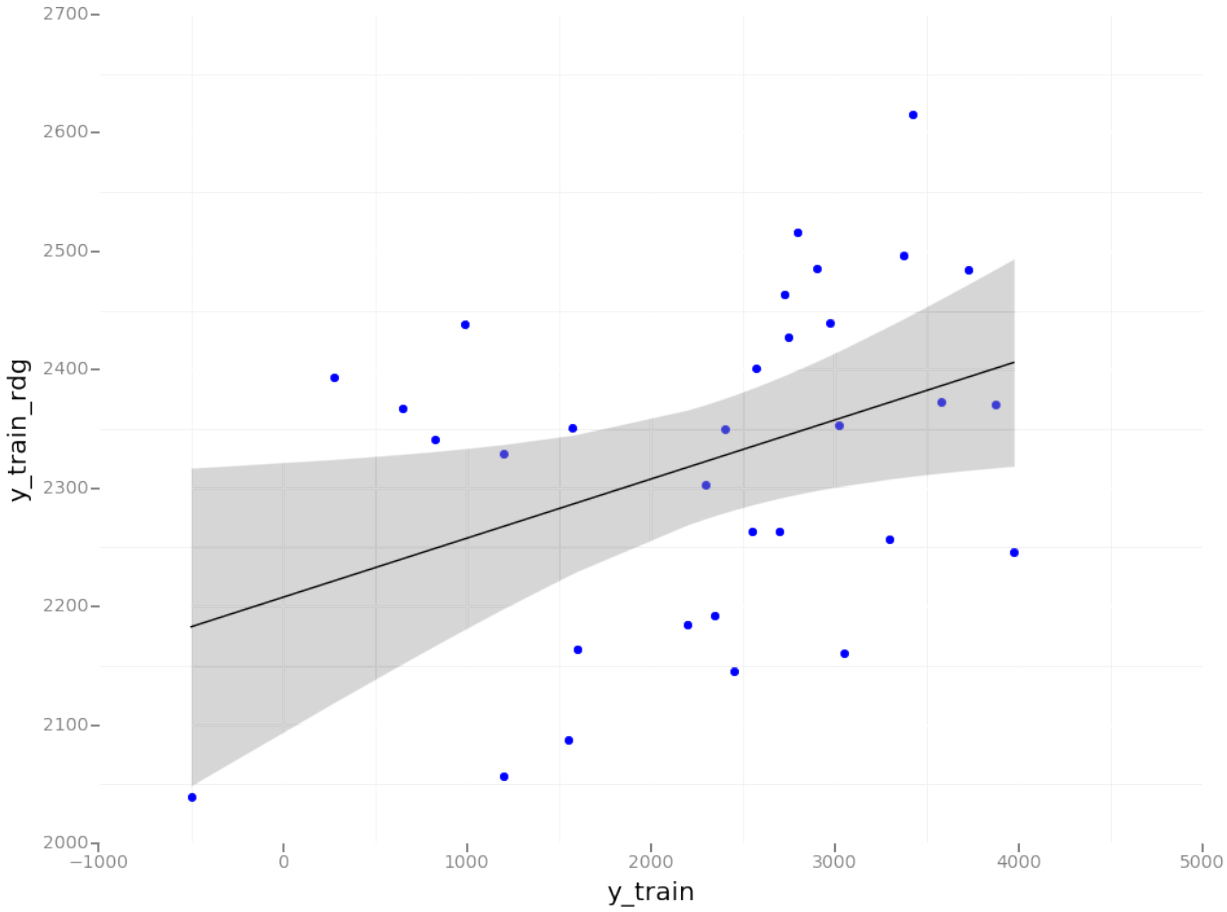
```
pydf = pd.DataFrame({'y_train':y_train,'y_train_pred':y_train_pred})
p = ggplot(pydf, aes('y_train','y_train_pred')) + geom_point(color='blue') +
geom_smooth(method='lm')
display(p)
```


4500
4000
3500
3000
2500
2000
1500
1000
500
y_train_pred
−1000
0
1000
2000
3000
4000
5000
y_train

```
# Ridge Regression
ridge = RidgeCV(alphas = [0.01, 0.03, 0.06, 0.1, 0.3, 0.6, 1, 3, 6, 10, 30, 60])
ridge.fit(X_train, y_train)
alpha = ridge.alpha_
print("Best alpha :", alpha)

print("Try again for more precision with alphas centered around " + str(alpha))
ridge = RidgeCV(alphas = [alpha * .6, alpha * .65, alpha * .7, alpha * .75, alpha *
.8, alpha * .85,
                          alpha * .9, alpha * .95, alpha, alpha * 1.05, alpha * 1.1,
alpha * 1.15,
                          alpha * 1.25, alpha * 1.3, alpha * 1.35, alpha * 1.4],
                cv = 3)
ridge.fit(X_train, y_train)
alpha = ridge.alpha_
print("Best alpha :", alpha)

# Look at predictions on training and validation set
print("Ridge RMSE on Training set :", rmse_cv_train(ridge).mean())
print("Ridge RMSE on Test set :", rmse_cv_test(ridge).mean())

# Look at fit, R^2 between 0 and 1, 1 best fit
print("R^2 of training set:", ridge.score(X_train, y_train))

# Look at coefficients
coefs = pd.Series(ridge.coef_, index = X_train.columns)
coefs.sort()
print(coefs)

# Apply model
y_train_rdg = ridge.predict(X_train)
y_test_rdg = ridge.predict(X_test)
```

```
('Best alpha :', 60.0)
Try again for more precision with alphas centered around 60.0
('Best alpha :', 84.0)
('Ridge RMSE on Training set :', 1312.1914084963262)
('Ridge RMSE on Test set :', 1422.2452231805146)
('R^2 of training set:', 0.083210743864887027)
year                         -94.242621
region_us/lax                -30.559438
n_similar_vehicles_listed    -24.184032
region_us/was                  2.637134
market_avg_days_to_sell       34.921919
mileage                       36.152237
region_us/sfo                 45.175729
dtype: float64
```

```
pydf = pd.DataFrame({'y_train':y_train,'y_train_rdg':y_train_rdg})
p = ggplot(pydf, aes('y_train','y_train_rdg')) + geom_point(color='blue') +
geom_smooth(method='lm')
display(p)
```

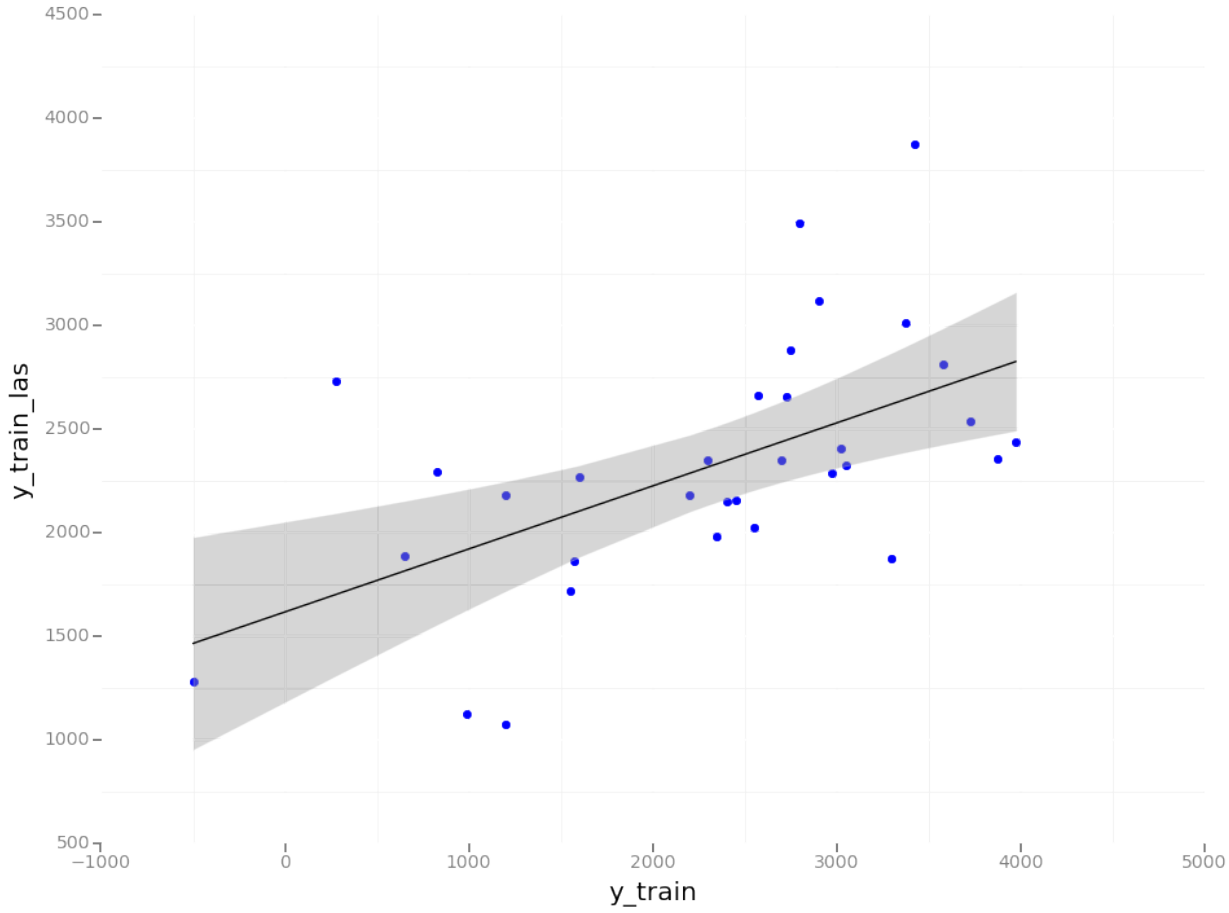


```
# Lasso Regression
lasso = LassoCV(alphas = [0.0001, 0.0003, 0.0006, 0.001, 0.003, 0.006, 0.01, 0.03,
0.06, 0.1,
                          0.3, 0.6, 1],
                max_iter = 50000, cv = 3)
lasso.fit(X_train, y_train)
alpha = lasso.alpha_
print("Best alpha :", alpha)

print("Try again for more precision with alphas centered around " + str(alpha))
lasso = LassoCV(alphas = [alpha * .6, alpha * .65, alpha * .7, alpha * .75, alpha *
.8,
                          alpha * .85, alpha * .9, alpha * .95, alpha, alpha * 1.05,
                          alpha * 1.1, alpha * 1.15, alpha * 1.25, alpha * 1.3, alpha
* 1.35,
                          alpha * 1.4],
                max_iter = 50000, cv = 3)
lasso.fit(X_train, y_train)
alpha = lasso.alpha_
print("Best alpha :", alpha)

# Look at predictions on training and validation set
print("Lasso RMSE on Training set :", rmse_cv_train(lasso).mean())
print("Lasso RMSE on Test set :", rmse_cv_test(lasso).mean())

# Look at fit, R^2 between 0 and 1, 1 best fit
print("R^2 of training set:", lasso.score(X_train, y_train))

# Look at coefficients
coefs = pd.Series(lasso.coef_, index = X_train.columns)
coefs.sort()
print(coefs)

# Apply model
y_train_las = lasso.predict(X_train)
y_test_las = lasso.predict(X_test)
```

```
('Best alpha :', 1.0)
Try again for more precision with alphas centered around 1.0
('Best alpha :', 1.3999999999999999)
('Lasso RMSE on Training set :', 1418.5142137077255)
('Lasso RMSE on Test set :', 1257.3934657652351)
('R^2 of training set:', 0.30990365914242768)
year                         -683.396028
mileage                      -169.290906
n_similar_vehicles_listed     148.737038
market_avg_days_to_sell       340.835950
```

```
region_us/lax                  746.239903
region_us/was                 1288.184995
region_us/sfo                 1487.492539
dtype: float64
```

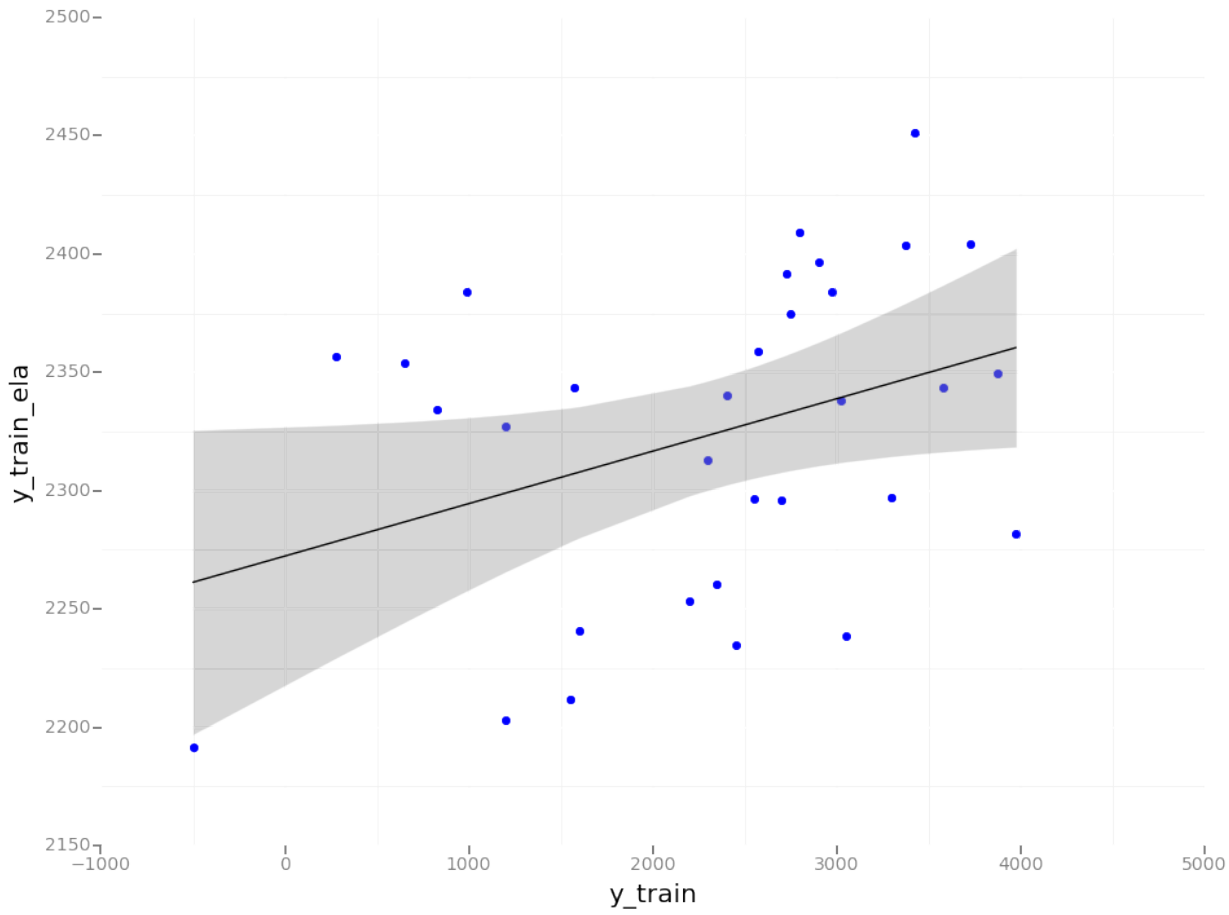
```
pydf = pd.DataFrame({'y_train':y_train,'y_train_las':y_train_las})
p = ggplot(pydf, aes('y_train','y_train_las')) + geom_point(color='blue') +
geom_smooth(method='lm')
display(p)
```




```
# ElasticNet Regression
elasticNet = ElasticNetCV(l1_ratio = [0.1, 0.3, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.95,
1],
                          alphas = [0.0001, 0.0003, 0.0006, 0.001, 0.003, 0.006,
                                    0.01, 0.03, 0.06, 0.1, 0.3, 0.6, 1, 3, 6],
                          max_iter = 10000, cv = 3)
elasticNet.fit(X_train, y_train)
alpha = elasticNet.alpha_
ratio = elasticNet.l1_ratio_
print("Best l1_ratio :", ratio)
print("Best alpha :", alpha )
```

```
print("Try again for more precision with l1_ratio centered around " + str(ratio))
elasticNet = ElasticNetCV(l1_ratio = [ratio * .85, ratio * .9, ratio * .95, ratio,
ratio * 1.05, ratio * 1.1, ratio * 1.15],
                          alphas = [0.0001, 0.0003, 0.0006, 0.001, 0.003, 0.006, 0.01,
0.03, 0.06, 0.1, 0.3, 0.6, 1, 3, 6],
                          max_iter = 10000, cv = 3)
elasticNet.fit(X_train, y_train)
if (elasticNet.l1_ratio_ > 1):
    elasticNet.l1_ratio_ = 1
alpha = elasticNet.alpha_
ratio = elasticNet.l1_ratio_
print("Best l1_ratio :", ratio)
print("Best alpha :", alpha )

print("Now try again for more precision on alpha, with l1_ratio fixed at " +
str(ratio) +
      " and alpha centered around " + str(alpha))
elasticNet = ElasticNetCV(l1_ratio = ratio,
                          alphas = [alpha * .6, alpha * .65, alpha * .7, alpha * .75,
alpha * .8, alpha * .85, alpha * .9,
                                    alpha * .95, alpha, alpha * 1.05, alpha * 1.1,
alpha * 1.15, alpha * 1.25, alpha * 1.3,
                                    alpha * 1.35, alpha * 1.4],
                          max_iter = 10000, cv = 3)
elasticNet.fit(X_train, y_train)
if (elasticNet.l1_ratio_ > 1):
    elasticNet.l1_ratio_ = 1
alpha = elasticNet.alpha_
ratio = elasticNet.l1_ratio_
print("Best l1_ratio :", ratio)
print("Best alpha :", alpha )

# Look at predictions on training and validation set
print("ElasticNet RMSE on Training set :", rmse_cv_train(elasticNet).mean())
print("ElasticNet RMSE on Test set :", rmse_cv_test(elasticNet).mean())

# Look at fit, R^2 between 0 and 1, 1 best fit
print("R^2 of training set:", elasticNet.score(X_train, y_train))

# Look at coefficients
coefs = pd.Series(elasticNet.coef_, index = X_train.columns)
coefs.sort()
print(coefs)

# Apply model
y_train_ela = elasticNet.predict(X_train)
y_test_ela = elasticNet.predict(X_test)
```

```
('Best l1_ratio :', 0.10000000000000001)
('Best alpha :', 6.0)
Try again for more precision with l1_ratio centered around 0.1
('Best l1_ratio :', 0.085000000000000006)
('Best alpha :', 6.0)
Now try again for more precision on alpha, with l1_ratio fixed at 0.085 and alpha cent
ered around 6.0
('Best l1_ratio :', 0.085000000000000006)
('Best alpha :', 8.3999999999999986)
('ElasticNet RMSE on Training set :', 1303.7360232350966)
('ElasticNet RMSE on Test set :', 1410.3514147136891)
('R^2 of training set:', 0.040601857844922207)
year                         -40.276436
n_similar_vehicles_listed    -14.654228
region_us/lax                -11.576781
region_us/was                  0.384869
market_avg_days_to_sell       12.695639
region_us/sfo                 16.772200
mileage                       20.177190
dtype: float64

pydf = pd.DataFrame({'y_train':y_train,'y_train_ela':y_train_ela})
p = ggplot(pydf, aes('y_train','y_train_ela')) + geom_point(color='blue') +
geom_smooth(method='lm')
display(p)
```




```
# Now do the same thing with pyspark and Spark ML - create a ML pipeline to handle
larger data sets
# Import packages/modules:
http://spark.apache.org/docs/latest/api/python/pyspark.ml.html
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder,
StandardScaler
from pyspark.ml.regression import LinearRegression, GBTRegressor
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator, TrainValidationSplit
from pyspark.ml.evaluation import RegressionEvaluator
```

```
# Pre-processing
#LR_DF = sqlContext.sql("SELECT region_short_code, make, year, mileage,
market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin AS label FROM
non_zero_expected_margin") # No log transforms of data since we use pyspark
StandardScaler, need expected_margin to be renamed label for pyspark
#categorical_features = ['region_short_code', 'make']
#numerical_features = ['year', 'mileage', 'market_avg_days_to_sell',
'n_similar_vehicles_listed']

LR_DF = sqlContext.sql("SELECT region_short_code, year, mileage,
market_avg_days_to_sell, n_similar_vehicles_listed, expected_margin AS label FROM
non_zero_expected_margin WHERE make='Honda' AND model='Civic'") # No log transforms of
data since we use pyspark StandardScaler, need expected_margin to be renamed label for
pyspark
categorical_features = ['region_short_code']
numerical_features = ['year', 'mileage', 'market_avg_days_to_sell',
'n_similar_vehicles_listed']

features = categorical_features + numerical_features
train_DF, test_DF = LR_DF.randomSplit(weights=[0.7, 0.3], seed=1) # use same seed each
time to get same results
display(LR_DF)
#display(train_DF)
#display(test_DF)
```

region_short_code	year	mileage	market_avg_days_to_se
us/sfo	2008	80000	43
us/sfo	2006	123000	44
us/sfo	2007	65000	46
us/sfo	2006	99500	61
us/sfo	2012	43000	61
us/sfo	2006	94482	60
us/sfo	2012	29000	66
us/lax	2008	56000	46
us/lax	2006	106000	41

```
# Feature Transformation pipeline
region_index = StringIndexer(inputCol='region_short_code', outputCol='region_index')
region_OHE = OneHotEncoder(inputCol='region_index', outputCol='region_OHE') #
automatically drops the last dummy variable
#make_index = StringIndexer(inputCol='make', outputCol='make_index')
#make_OHE = OneHotEncoder(inputCol='make_index', outputCol='make_OHE') # automatically
drops the last dummy variable
#cat_assembler = VectorAssembler(inputCols=['region_OHE', 'make_OHE'],
outputCol='cat_assembler')
cat_assembler = VectorAssembler(inputCols=['region_OHE'], outputCol='cat_assembler')
#cat_pipeline = Pipeline(stages=[region_index, region_OHE, make_index, make_OHE,
cat_assembler])
#model = cat_pipeline.fit(train_DF)
#feat_DF = model.transform(train_DF)
#display(feat_DF)

num_assembler = VectorAssembler(inputCols=numerical_features,
outputCol='num_unscaled')
num_scaled = StandardScaler(inputCol='num_unscaled', outputCol='num_scaled')
#num_pipeline = Pipeline(stages=[num_assembler, num_scaled])
#model = num_pipeline.fit(train_DF)
#feat_DF = model.transform(train_DF)
#display(feat_DF)

feat_assembler = VectorAssembler(inputCols=['cat_assembler', 'num_scaled'],
outputCol='features')
#pipeline = Pipeline(stages=[region_index, region_OHE, make_index, make_OHE,
cat_assembler, num_assembler, num_scaled, feat_assembler])
pipeline = Pipeline(stages=[region_index, region_OHE, cat_assembler, num_assembler,
num_scaled, feat_assembler])
ftModel = pipeline.fit(train_DF)
trainFeat_DF = ftModel.transform(train_DF)
#display(trainFeat_DF)
```

```
# Linear Regression fitting
#
https://github.com/apache/spark/blob/master/examples/src/main/python/ml/linear_regress
ion_with_elastic_net.py
lr = LinearRegression()
lrModel = lr.fit(trainFeat_DF)

# Linear Regression Summary
trainingSummary = lrModel.summary
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show()

# Print the coefficients and intercept for linear regression
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))

RMSE: 1044.382703
r2: 0.240332
numIterations: 1
objectiveHistory: [0.0]
+-------------------+
|          residuals|
+-------------------+
| 1105.3900823031436|
|-1293.3848816786194|
|  1072.419851440296|
|  -884.425052064471|
|-11.627712187415455|
| -2270.104075665935|
|  925.6855512580369|
| -2297.356207257719|
|  564.8665081544896|
| 1804.5101742455736|
|-190.77999094233382|
|  534.6793992117164|
| 1324.7266693483107|
| -7.735943142382894|
| 1001.2696864805184|
|  567.0305064866552|
|-1657.2341249743477|
| -2138.409296764061|
| 184.50841596367536|
| 1266.4319794333423|
+-------------------+
```

```
only showing top 20 rows

Coefficients: [-676.50195776,-781.757023732,-452.694667765,-370.259798779,245.766558677,-372.095130671]
Intercept: 329958.551609

# Fit the pipeline to training set
lrPipeline = Pipeline(stages=[pipeline, lr])
lrpModel = lrPipeline.fit(train_DF)

# Linear Regression prediction on training set
prediction = lrpModel.transform(train_DF)
#display(prediction)
evaluator = RegressionEvaluator(metricName='rmse')
print('RMSE: ', evaluator.evaluate(prediction))
print('R^2: ', evaluator.evaluate(prediction, {evaluator.metricName: "r2"}))

('RMSE: ', 1044.382702650478)
('R^2: ', 0.24033233401316667)

# Linear Regression prediction on test set
# Probably failing because the test set is too small with Honda Civic
#prediction1 = lrpModel.transform(test_DF)
#display(prediction1)
#print('RMSE: ', evaluator.evaluate(prediction1))
#print('R^2: ', evaluator.evaluate(prediction1, {evaluator.metricName: "r2"}))


# Linear Regression Cross Validation
# Hmm, couldn't get it to work, need to debug more
#lrPipe = Pipeline(stages=[lr])

#paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0, .5, 1]).addGrid(lr.elasticNetParam, [0, .5, 1]).build()
#paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0, .5, 1]).build()
paramGrid = ParamGridBuilder().build()

crossval = CrossValidator(
#    estimator=lrPipe,
    estimator=lrPipeline,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator,
    numFolds=3)

#cvModel = crossval.fit(trainFeat_DF)

# uncomment below to try out
#cvModel = crossval.fit(train_DF)
```

```
# Linear Regression CV prediction on training set
#predictionA = cvModel.transform(train_DF)
#print('RMSE: ', evaluator.evaluate(predictionA))
#print('R^2: ', evaluator.evaluate(predictionA, {evaluator.metricName: "r2"}))


# Linear Regression CV prediction on test set
# Probably failing because the test set is too small with Honda Civic
#predictionB = cvModel.transform(test_DF)
#print('RMSE: ', evaluator.evaluate(predictionB))
#print('R^2: ', evaluator.evaluate(predictionB, {evaluator.metricName: "r2"}))


# Try TrainValidationSplit instead; it's basically Cross Validation with numFolds=1
# Failing with Honda Civic
#paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0, .5,
1]).addGrid(lr.elasticNetParam, [0, .5, 1]).build()
#paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0, .5, 1]).build()
#paramGrid = ParamGridBuilder().build()

#tv = TrainValidationSplit(
#    estimator=lrPipeline,
#    estimatorParamMaps=paramGrid,
#    evaluator=evaluator,
#    trainRatio=.66)

#tvModel = tv.fit(train_DF)


# Linear Regression TrainValidationSplit training set prediction: not much difference
#prediction2 = tvModel.transform(train_DF)
#evaluator = RegressionEvaluator(metricName='rmse')
#print('RMSE: ', evaluator.evaluate(prediction2))
#print('R^2: ', evaluator.evaluate(prediction2, {evaluator.metricName: "r2"}))


# Linear Regression TrainValidationSplit test set prediction: not much difference
#prediction3 = tvModel.transform(test_DF)
#print('RMSE: ', evaluator.evaluate(prediction3))
#print('R^2: ', evaluator.evaluate(prediction3, {evaluator.metricName: "r2"}))
```

```
# Just for fun, let's try the Gradient Boosting Tree Regressor
# Training set: Much better results than linear regression!
# Unrealistic results for Honda Civic, sample size too small
gbt = GBTRegressor()
#gbtModel = gbt.fit(trainFeat_DF)

# Fit the pipeline to training set
gbtPipeline = Pipeline(stages=[pipeline, gbt])
gbtModel = gbtPipeline.fit(train_DF)

# Linear Regression prediction on training set
prediction4 = gbtModel.transform(train_DF)
#display(prediction4)
print('RMSE: ', evaluator.evaluate(prediction4))
print('R^2: ', evaluator.evaluate(prediction4, {evaluator.metricName: "r2"}))

('RMSE: ', 19.823127100816905)
('R^2: ', 0.999726316489231)

# Gradient Boosting Tree Regressor prediction on test set
# Unfortunately performance is around that of Linear Regression on the test set =(
# Doesn't work for Honda Civic, sample size too small?
#prediction5 = gbtModel.transform(test_DF)
#display(prediction5)
#print('RMSE: ', evaluator.evaluate(prediction5))
#print('R^2: ', evaluator.evaluate(prediction5, {evaluator.metricName: "r2"}))
```

Summary

- Linear Regression and Gradient Boosted Tree Regression doesn't give better results than the Percent to Market Model for the feature selection of: region, make, year, mileage, market_avg_days_to_sell, n_similar_vehicles_listed
- Linear Regression does give better results than the Percent to Market Model for the Honda Civic. But we need more data to make sure.

Further Research

- Implement Feature Factory: automated feature selection and A/B scoring; use Spark, genetic algo for parameter optimization
- Try other models above, including *Support Vector Regression with Genetic Parameter Optimization*, Artificial Neural Networks (see papers)
- Pricing model adapts to market conditions: adjust price according to supply/demand and

price trajectory, price decays over time to agreed upon minimum

- Cohort Analysis: segregate by region/zip, make/model, track cohorts over time, look for repeat behavior: http://www.gregreda.com/2015/08/23/cohort-analysis-with-python/ (http://www.gregreda.com/2015/08/23/cohort-analysis-with-python/)
- Time Series Analysis: need time of sale data for seasonal cycles, predictive trendlines, regimes, repeat activity over time: seasonal discounts? use Digital Signal Processing
- We can try to apply the models above to the cars where expected_margin = 0 to see what we get
- Boruta feature importance: https://www.kaggle.com/jimthompson/house-prices-advanced-regression-techniques/boruta-feature-importance-analysis (https://www.kaggle.com/jimthompson/house-prices-advanced-regression-techniques/boruta-feature-importance-analysis)
- Try xgboost regression, couldn't get xgboost to work on Databricks Spark: https://xgboost.readthedocs.io/en/latest/ (https://xgboost.readthedocs.io/en/latest/) https://github.com/dmlc/xgboost/tree/master/demo/regression (https://github.com/dmlc/xgboost/tree/master/demo/regression) https://www.kaggle.com/zoupet/house-prices-advanced-regression-techniques/xgboost-ridge-lasso (https://www.kaggle.com/zoupet/house-prices-advanced-regression-techniques/xgboost-ridge-lasso)
- Gather information about sellers & buyers and market to sellers & buyers with similar characteristics to improve number of inquiries and conversion rate? Track their behavior over time, get repeat business
- Graph analysis: relationships, map out buyers and seller relationships, target sellers with multiple cars, find repeat buyers, also useful for fraud detection
- Simulate Empirical Testing Framework: Model seller/buyer behavior: example - model probability of seller withdrawal: poisson process (used in credit defaults)
- Find missing Expected Margin data using Predictive Imputation / MICE (some sort of interpolation between existing points based on distribution of existing data): http://statisticalhorizons.com/predictive-mean-matching (http://statisticalhorizons.com/predictive-mean-matching), https://www.kaggle.com/mrisdal/titanic/exploring-survival-on-the-titanic/notebook (https://www.kaggle.com/mrisdal/titanic/exploring-survival-on-the-titanic/notebook)
- Digital Signal Processing: up sampling interpolation
- More sophisticated models: ensemble methods of different models: Spark MLLIB, https://www.kaggle.com/jimthompson/house-prices-advanced-regression-techniques/ensemble-model-stacked-model-example

(https://www.kaggle.com/jimthompson/house-prices-advanced-regression-techniques/ensemble-model-stacked-model-example)
- Principle Component Analysis on factors?
- Cluster analysis? kMeans/Gaussian Mixture Model
- Reinforcement learning? Electrical Engineering Control Systems
- More data is usually better than algorithm selection, this is why Spark was chosen for this assignment: https://www.quora.com/In-machine-learning-is-more-data-always-better-than-better-algorithms (https://www.quora.com/In-machine-learning-is-more-data-always-better-than-better-algorithms)
- How to put models into production and iterate quickly on them? E.g. Mleap, PMML, prediction.io, pipeline.io, Cloudera, etc.

```
%sql
-- Cohort Analysis
-- Count the number of points of data we have for expected_margin for the most numerous Make-Models
-- As you can see, the sample size is too small to create reasonable predictive models based on a Make-Model breakdown
-- The sample size is too small to have both a training and test set for each Make-Model
SELECT make_model, count(*) as NUM FROM (
  SELECT CONCAT(make, '_', model) AS make_model FROM non_zero_expected_margin
) GROUP BY make_model ORDER BY NUM DESC
```

make_model
Honda_Civic
BMW_3 Series
Toyota_Corolla
Toyota_Prius
Honda_Accord

```
%sql
-- Cohort Analysis
-- So for this notebook, let's do our analysis on Makes only; we have enough samples for cross-validation
SELECT make, count(*) as NUM FROM (
  SELECT make FROM non_zero_expected_margin
) GROUP BY make ORDER BY NUM DESC
```

make
Honda
Toyota
BMW
Volkswagen
Audi
Nissan
Lexus
Hyundai
Ford
Mazda
Subaru
Mercedes-Benz