



Predicting Used Car Resale Price

Daniel Kim

Problem Statement

Used car prices generally have been fluctuating up and down with time, with some probability of crashing. With price increase, or decrease, many drivers (either looking to buy or sell) are affected. The goal is to predict current values of the used cars so that buyers and sellers can make informed decisions.

The Data

This data contains most all relevant information that Craigslist provides on car sales including columns like price, condition, manufacturer, latitude/longitude, and 18 other categories.

1. Target, y = 'price'
2. Other variables, X_s = all other features than 'price'
3. Total number of rows = 1.1 million
4. Total number of columns = 26 features

Data Wrangling

1. Cleansing Data

a. Redundancy

The original dataset contains 1.1 million rows and 26 columns. Once the dataset was read into python, cleaning and transformation of the data was required to prepare the dataset for analysis. With 26 columns in the original dataset, we need to cut down the number of features the model will be built upon.

First of all, many columns and rows containing redundant information were dropped, by using 'df.drop_duplicates(inplace=True)'. For instance, if a seller would like to expose his car information more often, the person might post the car's information multiple times onto "www.craigslist.org". These kinds of transactions definitely generate redundancies.

b. Null Values, NaN

By using 'df.info()', there are many null values as seen below.

```
1 df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1048636 entries, 0 to 1049199
Data columns (total 26 columns):
 #   Column          Non-Null Count  Dtype
---  -
0   url              1048635 non-null object
1   city             1048559 non-null object
2   price            1048535 non-null object
3   year             1047014 non-null object
4   manufacturer     964568 non-null object
5   make             1005318 non-null object
6   condition        615062 non-null object
7   cylinders        621193 non-null object
8   fuel             1042054 non-null object
9   odometer         705601 non-null float64
10  title_status     1046971 non-null object
11  transmission     1042699 non-null object
12  vin              367156 non-null object
13  drive            645697 non-null object
14  size             358029 non-null object
15  type             614919 non-null object
16  paint_color      617962 non-null object
17  image_url        1048519 non-null object
18  lat              1048519 non-null float64
19  long             1048519 non-null float64
20  county_fips      990178 non-null float64
21  county_name      990178 non-null object
22  state_fips       990178 non-null float64
23  state_code       990178 non-null object
24  state_name       1048519 non-null object
25  weather          989582 non-null float64
dtypes: float64(6), object(20)
memory usage: 216.0+ MB
```

In consideration with total numbers of rows are 1,049,200, there are many non-null values across all columns; thus, you can identify that many null values exist over all features: null values = all values - non-null values.

i. Drop NaN

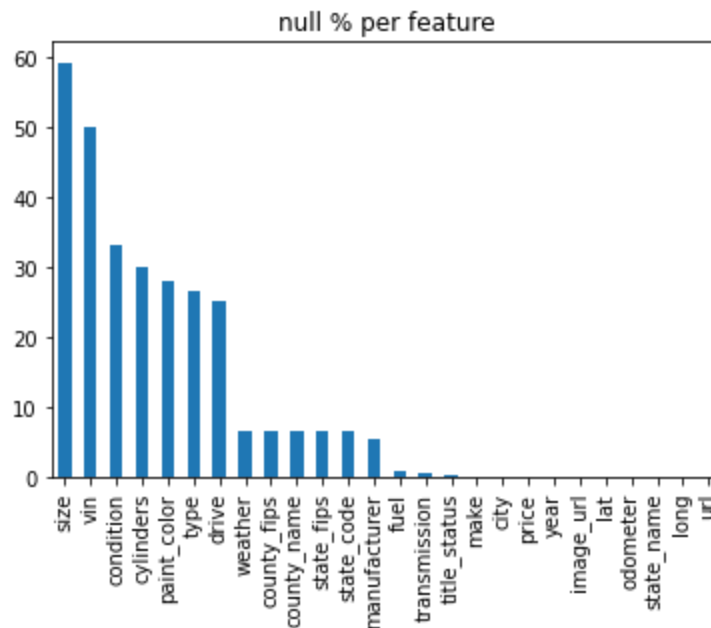
Before dropping null values, it is important to identify which rows should be dropped.

Firstly, considering 'price' as the target, all rows having null valued 'price' were dropped in that it is impossible to assume 'price'.

Secondly, rows with null-valued 'lat' were dropped because the null value of either 'lat' or 'long' cannot give you correlatable geographical information.

Thirdly, rows with null-valued 'year' were dropped in that it is not possible to find correlation between 'price' and 'unidentified year of use', which is null value in year.

Fourthly, rows with null-valued 'make' and null-valued 'odometer' were dropped in that null value of both 'make' and 'odometer' can not be correlated with 'price'.



Although such features as 'size' and 'vin' have the most null values than other features, they could be filled with the same or similar car's information by referring external datasets.

ii. Referring external data

1. Reverse Geocoding

In order to find the exact locations, 'reverse_geocode' was applied according to such information as latitude as 'lat' longitude as 'long'.

```

25 [x]:
import reverse_geocoder as rg
import pprint

def reverseGeocode(coordinates):
    result = rg.search(coordinates)
    return (result)

if __name__=="__main__":

    coordinates = list(zip(df['lat'],df['long']))
    data = reverseGeocode(coordinates)

    df['city1'] = [i['name'] for i in data]
    df['state'] = [i['admin1'] for i in data]
    df['county'] = [i['admin2'] for i in data]

```

2. Merging with other datasets

'vehicles.csv' was used for replacing 'cylinder', 'drive', 'year', 'transmission', 'manufacturer', and 'make'.

```
df_fuel = pd.read_csv("vehicles.csv", usecols = ['cylinders', 'drive',
                                                'youSaveSpend', 'fuelTy',
                                                'make', 'model', 'trany',
                                                'year'])
```

[illegible]

iii. Fill NaN with Majority Values

Rows with null values were filled by using for-loops on a basis of 'make' as seen below.

```
i=0
a = df['make'][df['manufacturer'].isnull()].value_counts().index

for i in tqdm(range(0,12655)):

    try:
        df.loc[((df['make'].str.contains(a[i]))
                & (df['manufacturer'].isnull())),'manufacturer']=df['make']

    except:
        pass

100%|██████████| 12655/12655 [10:33<1:11:42, 2.85it/s] C:\Users\Daniel\Anaconda\lib\site-packages\pandas\core\strings.py:2001: UserWarning: This pattern has match groups. To actually get the groups, use str.extract.
return func(self, args, *kwargs)
100%|██████████| 12655/12655 [1:23:49<00:00, 2.52it/s]
```

By doing so, such other columns' NaN values were filled by their most values on a basis of 'make': 'manufacturer', 'transmission', 'fuel', 'drive', 'type', 'size', 'youSaveSpend', and 'VClass'.

In order to fill in the weather's NaN, let's make another column 'loc' which combines 'city', 'county', and 'state'. And then the similar for-loop was applied as below.

```
In [101]: i=0
a = df['loc'][df['weather'].isnull()].value_counts().index

for i in tqdm(range(0,514)):

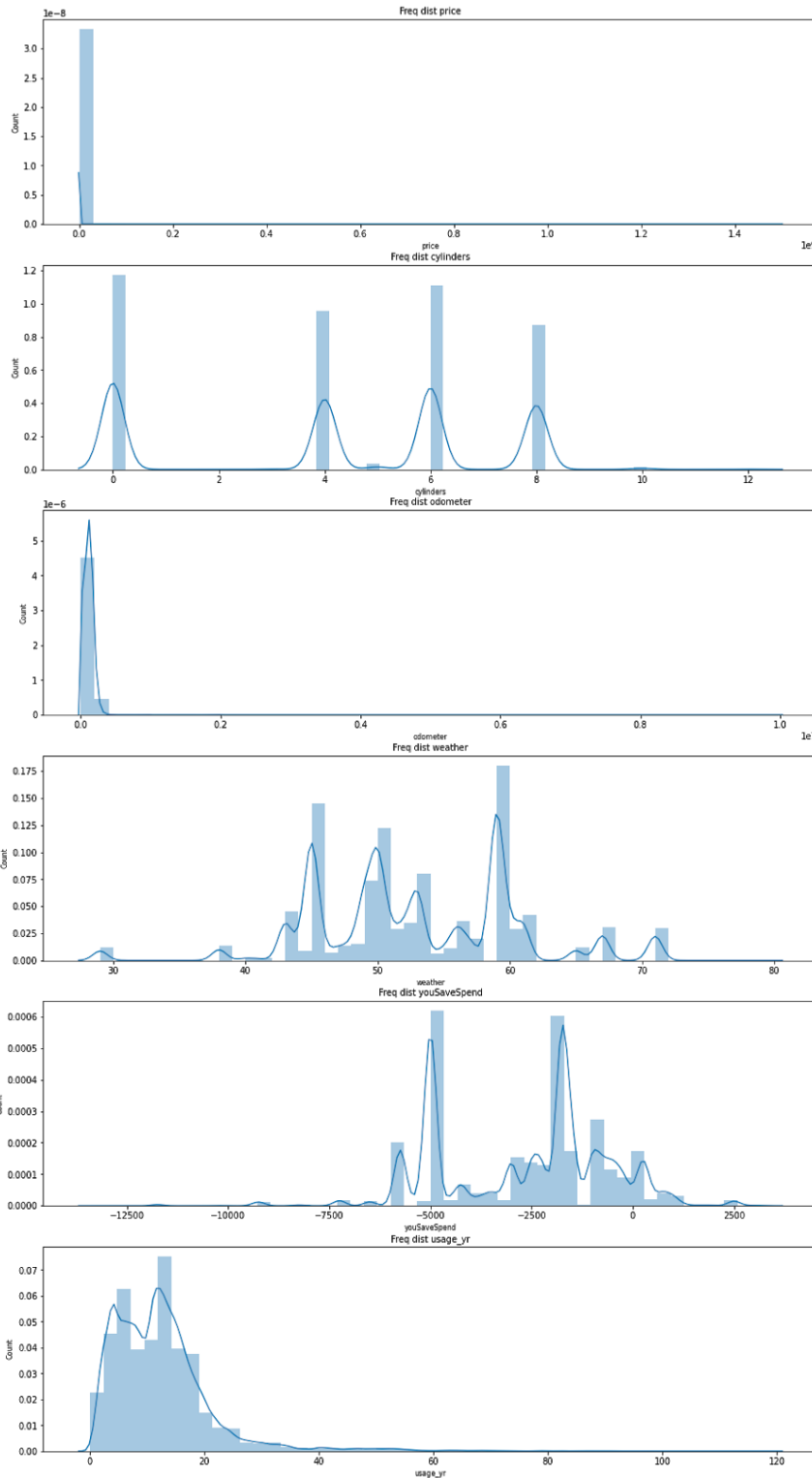
    try:
        df.loc[((df['loc'].str.contains(a[i]))
                & (df['weather'].isnull())),'weather']=df['weather']

    except:
        pass

100%|██████████| 514/514 [01:45<00:00, 4.92it/s]
```

Exploratory Data Analysis

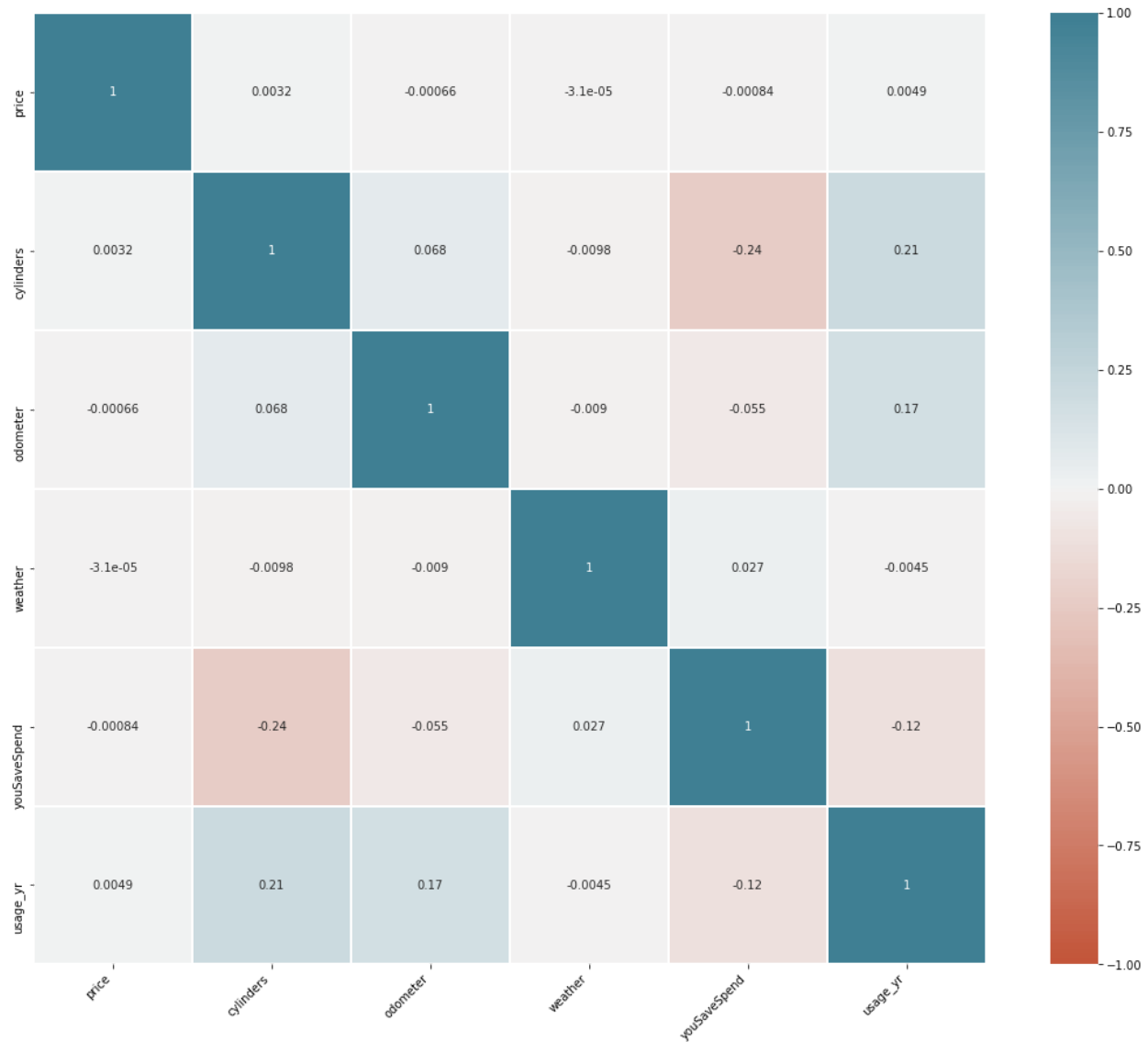
1. Histogram



Among numerical valued features, only 'usage_yr' shows closest normal distribution with right-skewness.

2. Correlation

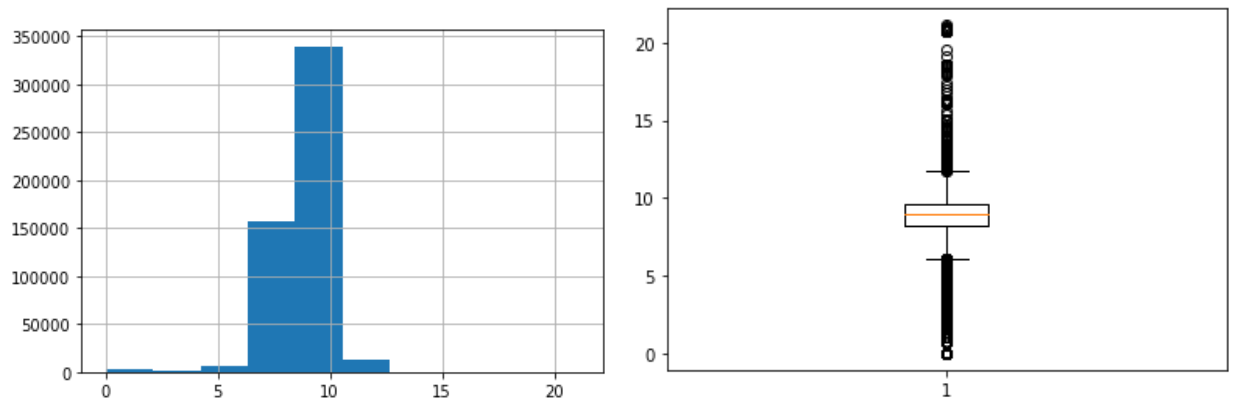
As seen below, numerical features as variables have not distinctive correlation to 'price' as target.



3. Outlier

Target, 'price' seems to have a timid correlation with other features. So let's remove outliers of 'price'.

Let's make another feature 'price_log'.



The above 'price_log' seems to have too many outliers, so it is better to remove outliers by other methods, not by considering 'price_log' only'.

Conclusion

- In order to remove outliers, an algorithm is needed to remove anomalies, which consider all round aspects of all features.
- By using different machine learning algorithm, compare important features, prediction accuracies, and other insights for better prediction.