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, Xs = 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

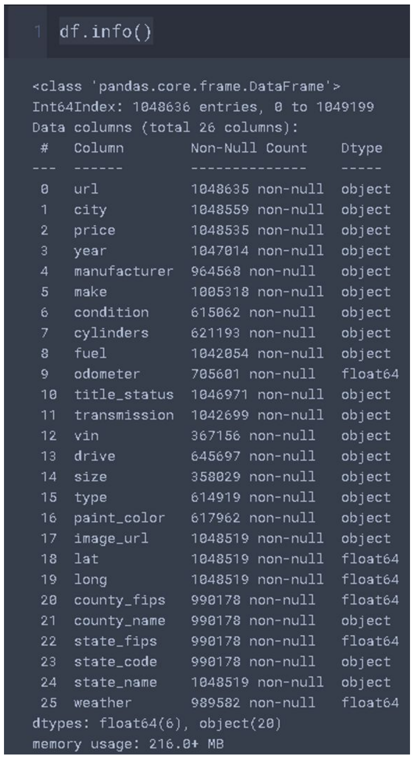
a. Redundancy

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.



```
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

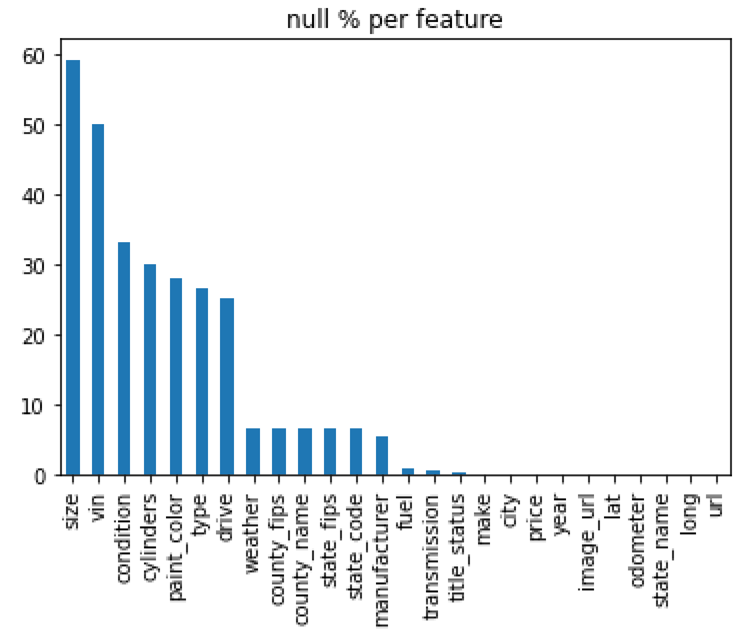
Before dropping null values, it is important to identify which rows should be dropped.

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.

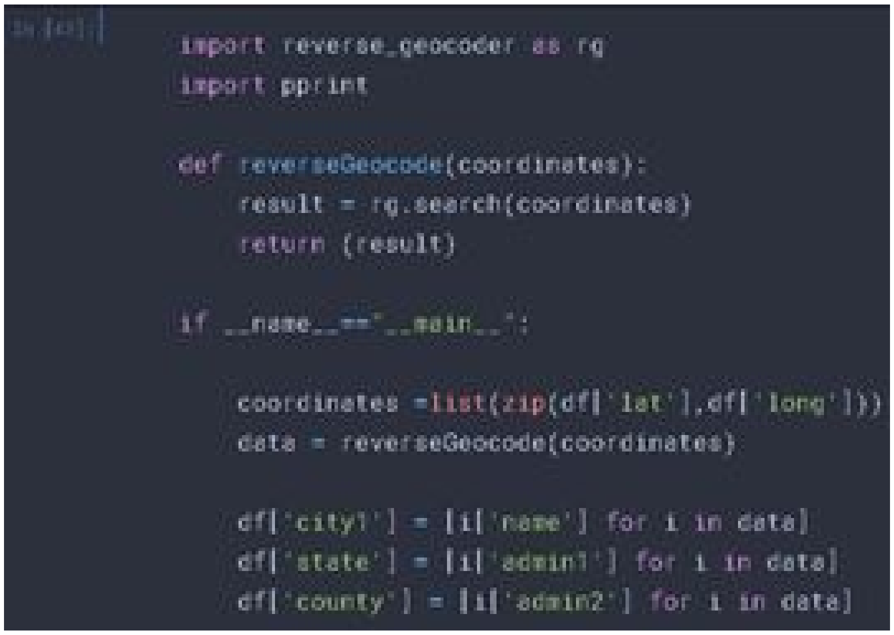
Fourthly, rows with null-valued 'make' and null-valued 'odometer’ were dropped in that null value of both 'make' and ‘odometer’ can not be correlated with 'price'.



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

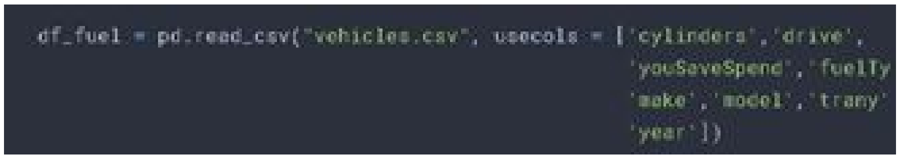
1. **Reverse Geocoding**
   In order to find the exact locations, 'reverse_geocode' was applied according to such information as latitude as 'lat' longitude as 'long'.



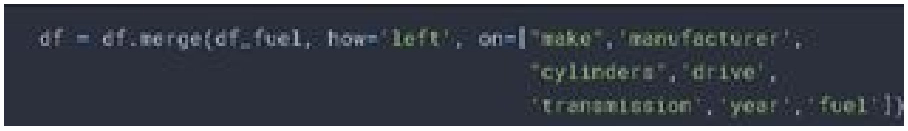
```
In [xx]:
        import reverse_geocoder as rg
        import pprint

        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'.



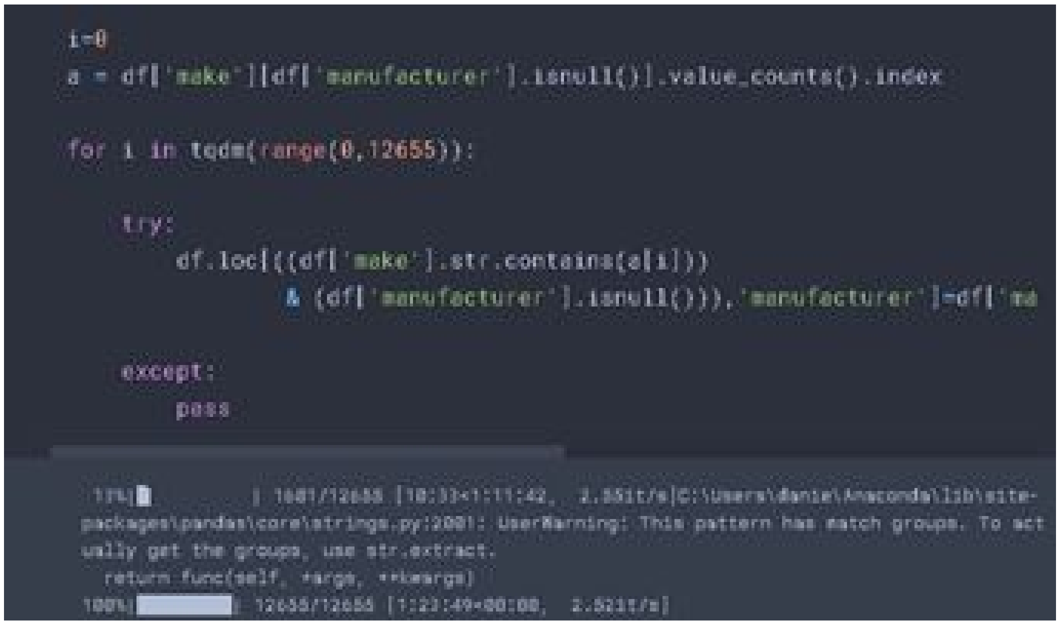
```
df_fuel = pd.read_csv("vehicles.csv", usecols = ['cylinders','drive',
                                                 'youSaveSpend','fuelTy
                                                 'make','model','trany'
                                                 'year'])
```



```
df = df.merge(df_fuel, how='left', on=['make','manufacturer',
                                       'cylinders','drive',
                                       'transmission','year','fuel'])
```

iii. Fill NaN with Majority Values

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

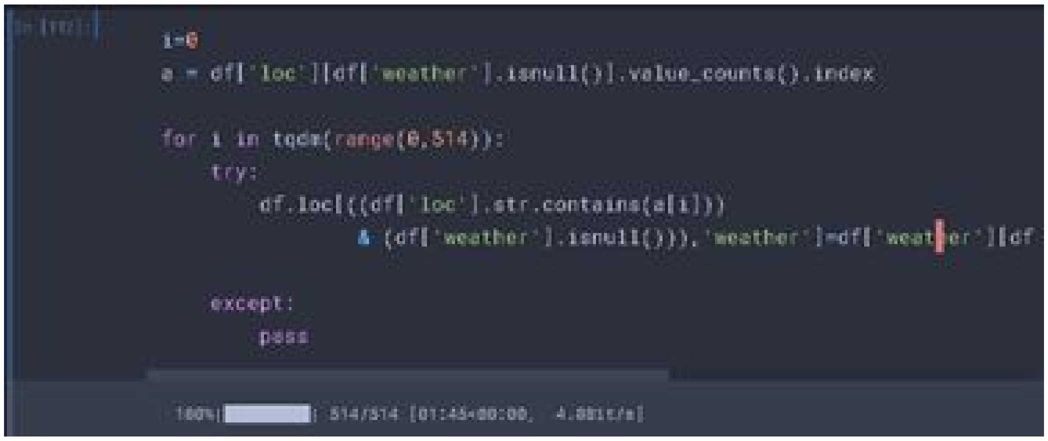


```
i=0
a = df['make'][df['manufacturer'].isnull()].value_counts().index

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

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

    except:
        pass
```

```
 13%|█          | 1681/12655 [10:33<1:11:42,  2.55it/s]C:\Users\danie\Anaconda\lib\site-
packages\pandas\core\strings.py:2001: UserWarning: This pattern has match groups. To act
ually 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.

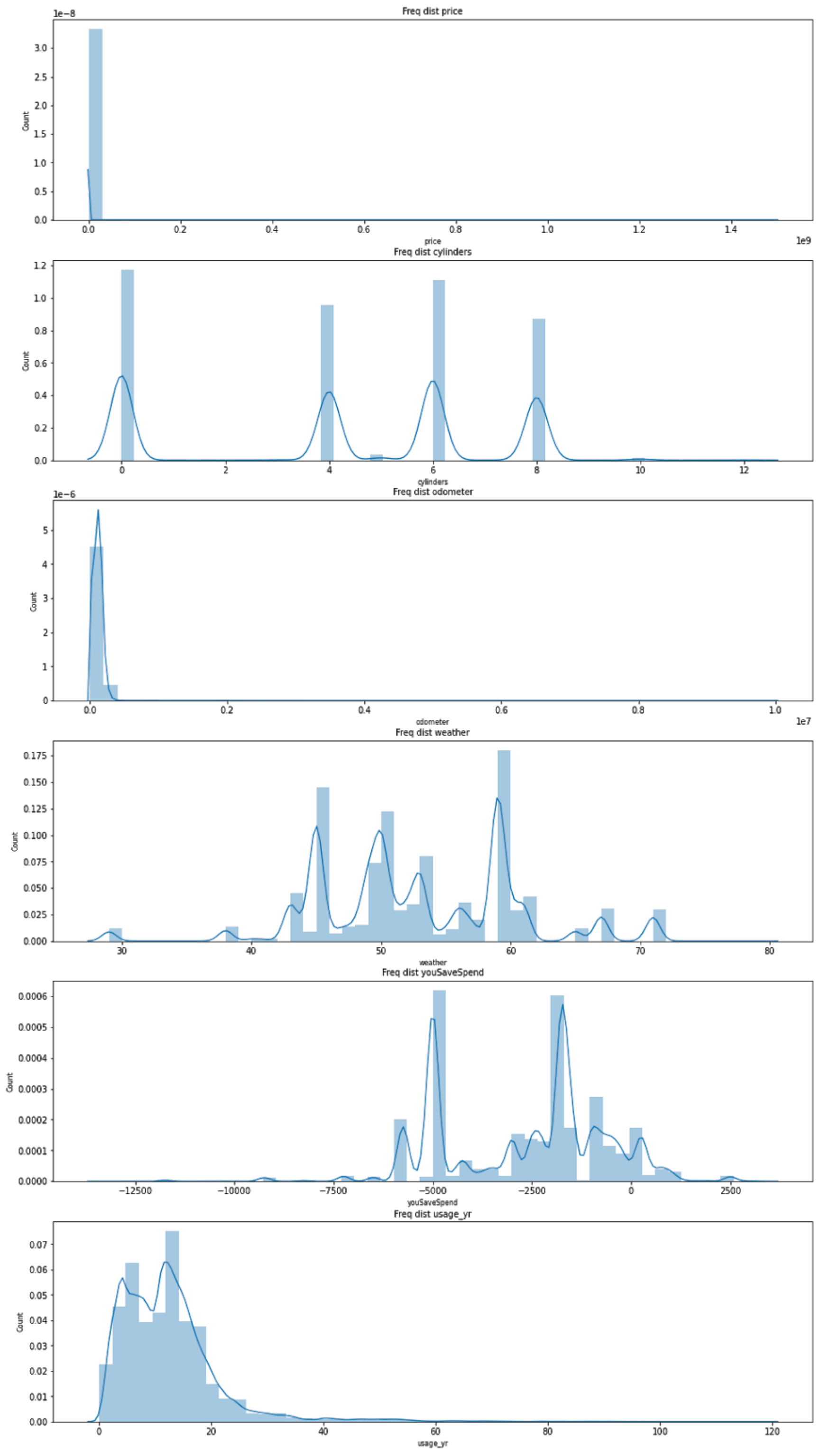


```
In [11]:
i=0
a = df['loc'][df['weather'].isnull()].value_counts().index

for i in tqdm(range(0,514)):
    try:
        df.loc[((df['loc'].str.contains(a[i]))
                & (df['weather'].isnull())),'weather']=df['weather'][df

    except:
        pass
```

```
100%|██████████| 514/514 [01:45<00:00,  4.88it/s]
```

Exploratory Data Analysis

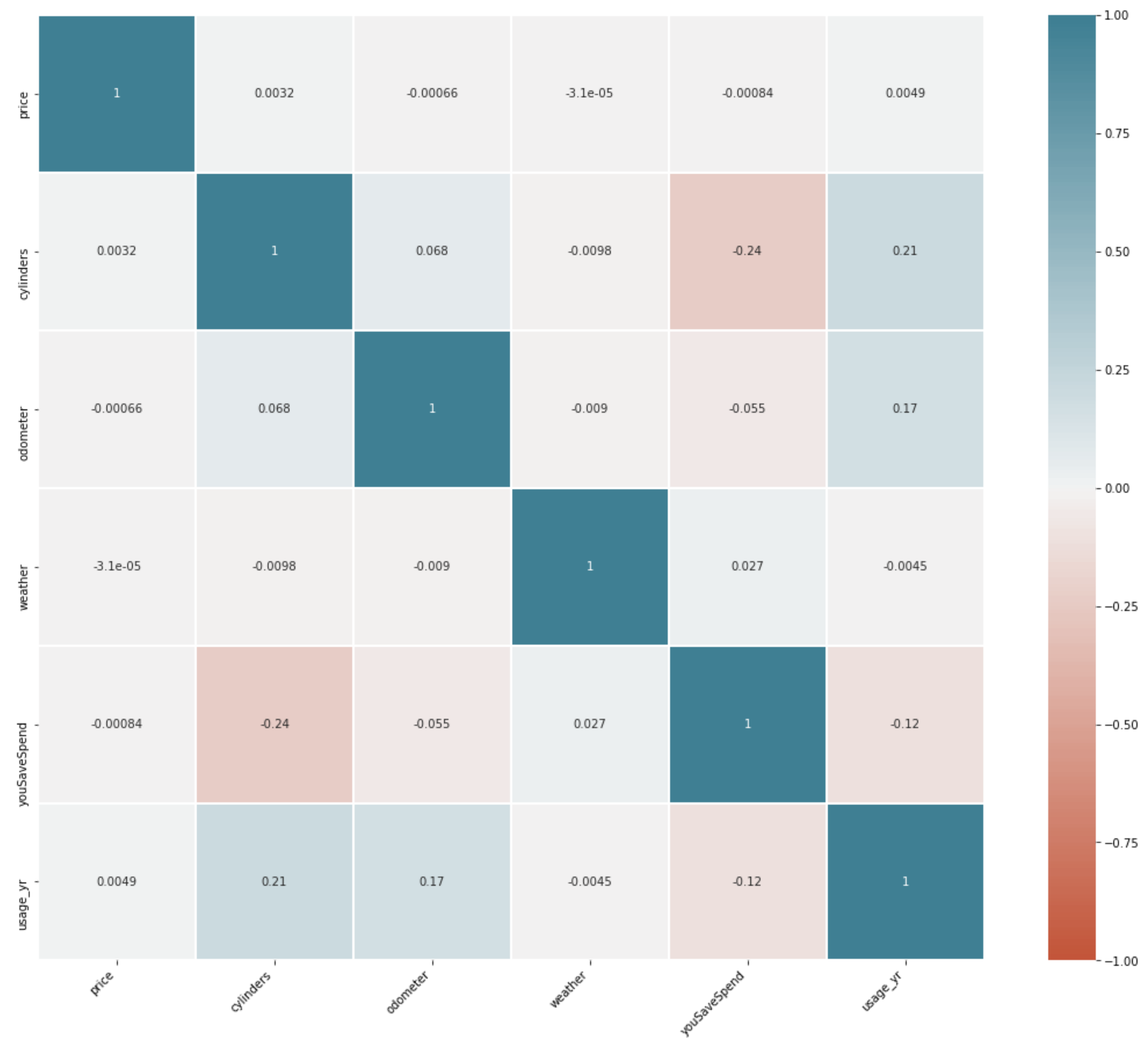
1. Histogram



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

2. Correlation

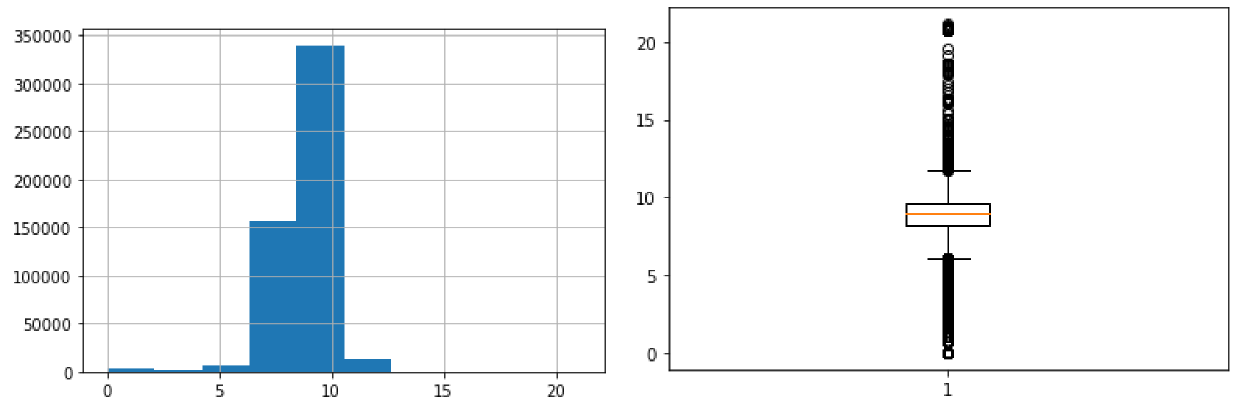
As seen below, numerical features as variables have not distinctive correlation to ‘price’ as target.



3. Outlier

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

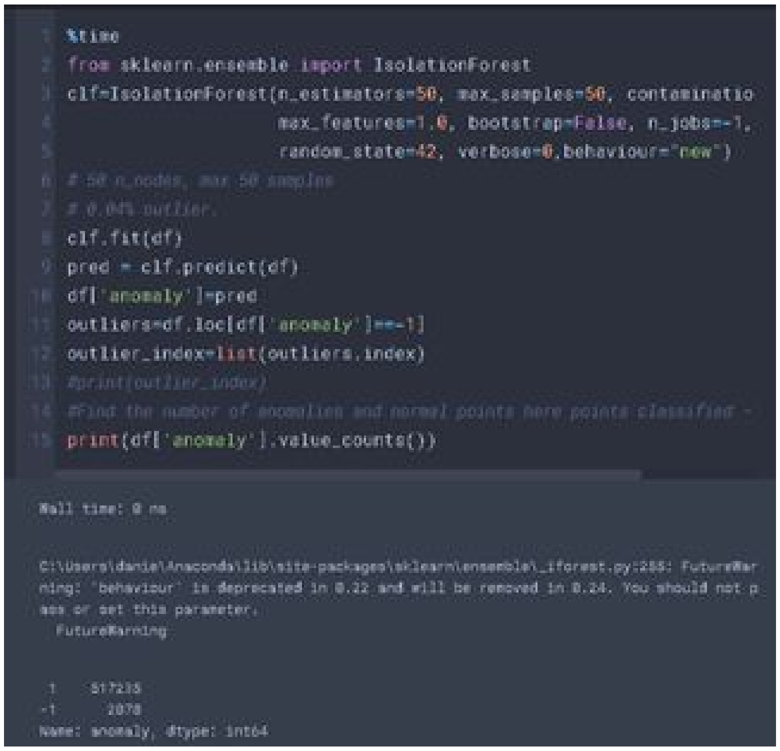
Let's make another feature 'price_log'.



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’.

Removing Outlier by Machine Learning Algorithm : Isolation Forest

It is necessary to remove outliers effectively and efficiently in order to predict on the basis of machine learning algorithms. Judging outliers by limited numbers of features can lead to a big bias which may result in wrong prediction. So in order to run Isolation Forest, the dataset was processed with One Hot Encoding, and then it was processed with the Isolation Forest algorithm as seen below.



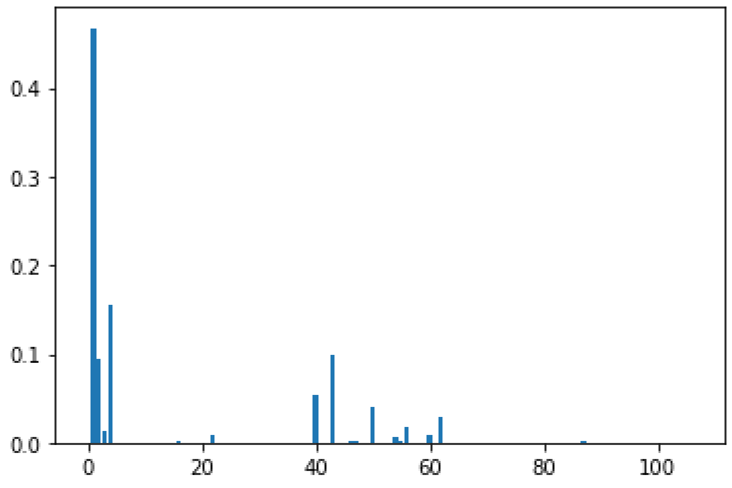
```
%time
from sklearn.ensemble import IsolationForest
clf=IsolationForest(n_estimators=50, max_samples=50, contaminatio
                    max_features=1.0, bootstrap=False, n_jobs=-1,
                    random_state=42, verbose=0,behaviour="new")
# 50 n_nodes, max 50 samples
# 0.04% outlier.
clf.fit(df)
pred = clf.predict(df)
df['anomaly']=pred
outliers=df.loc[df['anomaly']==-1]
outlier_index=list(outliers.index)
#print(outlier_index)
#Find the number of anomalies and normal points here points classified -
print(df['anomaly'].value_counts())
```

```
Wall time: 0 ns


C:\Users\danie\Anaconda\lib\site-packages\sklearn\ensemble\_iforest.py:255: FutureWar
ning: `behaviour` is deprecated in 0.22 and will be removed in 0.24. You should not p
ass or set this parameter.
  FutureWarning


 1    517235
-1      2078
Name: anomaly, dtype: int64
```

Comparison of Feature Importance:

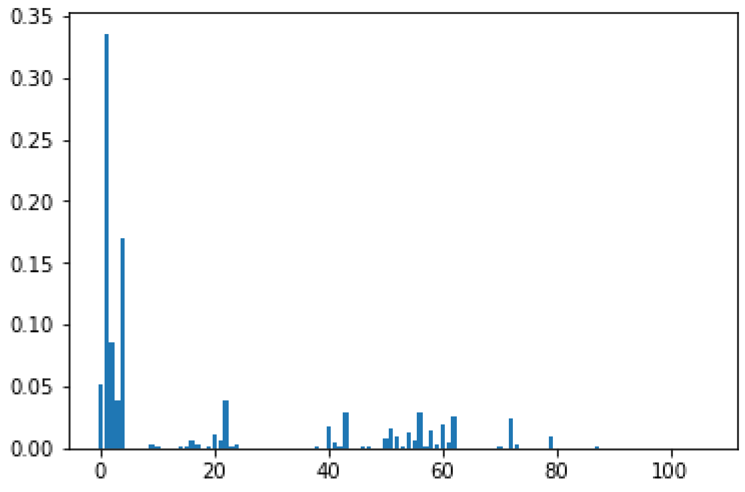
Decision Tree Regressor vs. Random Forest Regressor vs. Gradient Boost Regressor

The below compares each algorithms' feature importances whose importance has more than 5% only.



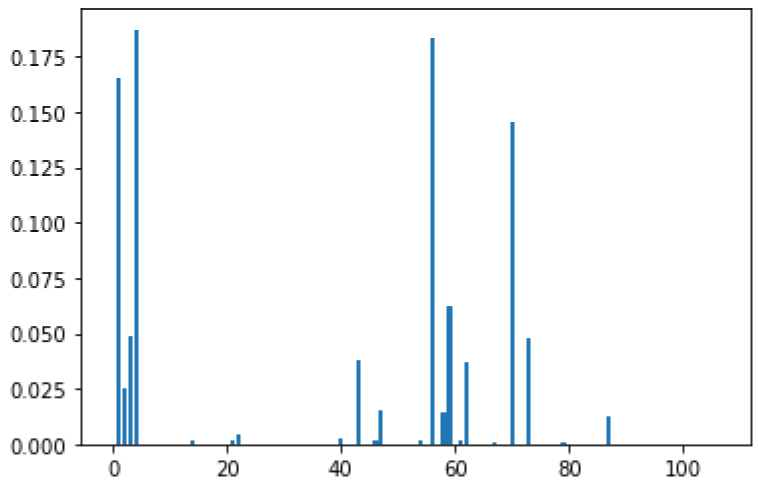
a. Decision Tree Regressor (x>=5% only)

<u>Feature: 'odometer', Score: 0.46751</u>
Feature: 'weather', Score: 0.09520
<u>Feature: 'usage_yr', Score: 0.15504</u>
Feature: 'Vclass_sport utility vehicle - 4wd', Score: 0.05271
Feature: 'VClass_standard pickup trucks 4wd', Score: 0.09778



b. Random Forest Regressor (x>=5% only)

Feature: 'cylinders', Score: 0.05090
<u>Feature: 'odometer', Score: 0.33581</u>
Feature: 'weather', Score: 0.08572
<u>Feature: 'usage_yr', Score: 0.16935</u>



c. Gradient Boost Regressor (x>=5% only)

<u>Feature: 'odometer', Score: 0.16573</u>
<u>Feature: 'usage_yr', Score: 0.18730</u>
<u>Feature: 'division_Middle Atlantic', Score: 0.18348</u>
Feature: 'manufacturer_cadillac', Score: 0.06249

Feature: ‘manufacturer_chevrolet’, Score: 0.14577

By comparing each feature importance from three different machine learning algorithms, ‘odometer’ and ‘ usage_yr’ showed steady strong features when it comes to predicting ‘price’. So we can conclude that ‘odometer’ and ‘usage_yr’ play key roles in predicting ‘price’.

Interestingly, Gradient Boost Regressor showed such other strong features with more than 10% importance rates as 'division_Middle Atlantic' and ‘manufacturer_chevrolet’, which are One Hot Encoded for machine learning processes. This can be goot bridge to analyse further to identify how the target, ‘price’ and features are correlated to each other on such following conditions exclusively:: (1)‘division_Middle Atlantic' only, or (2) ‘manufacturer_chevrolet’, or both (1) and (2) altogether.

Comparison of Accuracy

Algorithms	Decision Tree Regressor	**Random Forest Regressor**	Gradient Boost Regressor
Accuracy (R-Squared)	-0.00018250127492003276	**0.005147258331855031**	-0.30837667790585555
Randomized Search CV Hyper Parameter Run Time	2.5 seconds	**853.5 minutes**	112.4 minutes
Best Parameters by Randomized Search CV	'splitter': 'random', 'min_samples_split': 15, 'min_samples_leaf': 10, 'max_features': 'log2', 'max_depth': 5, 'criterion': 'friedman_mse'	**'n_estimators': 120, 'min_samples_split': 15, 'min_samples_leaf': 5, 'max_features': 'sqrt', 'max_depth': 15, 'criterion': 'mse'**	'n_estimators': 300, 'min_samples_split': 100, 'min_samples_leaf': 5, 'max_features': 'sqrt', 'max_depth': 25, 'loss': 'ls', 'criterion': 'friedman_mse'

Given the above test results, Random Forest Regressor showed the best score (only positive as 0.005) in spite of an enormous amount of Run Time (853.5 minutes). Gradient Boost Regressor showed disappointing results: the lowest score (-0.31) in consideration with a lot of Run Time taken.

Recommendation & Solution

In spite of its low accuracy and long run time, Random Forest Regressor is the most recommendable machine learning algorithm to predict an used car’s ‘price’ from Craigslist. In order to raise accuracy, as well as, to reduce machine learning run time, run the prediction of Random Forest Regressor on the bases of the above “Best Parameters by Randomized Search

CV” condition of Random Forest Regressor as independent variables, while tuning such other features as Geolocation, Make, Model, and etc. as dependant variables.