Writer: Zhenjian(Tom) Wang on 08/13/2021

Overview

Many college students, like me, want to buy a car so that we can travel around. However, since we have a limited amount of money, we want to find a non-expensive car that does not depreciate much to save our money. This project analyzes the top ten non-expensive cars that have the lowest depreciation rate.

Data and Model

data citation: we use the used cars dataset that can be downloaded from https://www.kaggle.com/austinreese/craigslist-carstrucks-data. This dataset keeps updating and it shows the most recent used car data from different car dealer websites.

In [1]:
```
import pandas as pd
import numpy as np

dataframe_original = pd.read_csv("vehicles.csv")
dataframe_original.head(5)
dataframe_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426880 entries, 0 to 426879
Data columns (total 26 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            426880 non-null  int64  
 1   url           426880 non-null  object 
 2   region        426880 non-null  object 
 3   region_url    426880 non-null  object 
 4   price         426880 non-null  int64  
 5   year          425675 non-null  float64
 6   manufacturer  409234 non-null  object 
 7   model         421603 non-null  object 
 8   condition     252776 non-null  object 
 9   cylinders     249202 non-null  object 
 10  fuel          423867 non-null  object 
 11  odometer      422480 non-null  float64
 12  title_status  418638 non-null  object 
 13  transmission  424324 non-null  object 
 14  VIN           265838 non-null  object 
 15  drive         296313 non-null  object 
 16  size          120519 non-null  object 
 17  type          334022 non-null  object 
 18  paint_color   296677 non-null  object 
 19  image_url     426812 non-null  object 
 20  description   426810 non-null  object 
 21  county        0 non-null       float64
 22  state         426880 non-null  object 
 23  lat           420331 non-null  float64
 24  long          420331 non-null  float64
 25  posting_date  426812 non-null  object 
dtypes: float64(5), int64(2), object(19)
memory usage: 84.7+ MB
```

The dataframe has 26 columns and we only need the price, year, manufacturer, model of the

cars.

Data Selection and Cleaning

To clean up the dataset, we are going to choose only the columns we need and drop all the na values.

In [2]:
```
df = dataframe_original[['price','year','manufacturer','model','condition']].dropna()
print(df.size)
print(df.head(10))
```

```
1198390
    price    year manufacturer                      model  condition
27  33590  2014.0          gmc     sierra 1500 crew cab slt       good
28  22590  2010.0    chevrolet               silverado 1500       good
29  39590  2020.0    chevrolet          silverado 1500 crew       good
30  30990  2017.0       toyota         tundra double cab sr       good
31  15000  2013.0         ford                    f-150 xlt  excellent
32  27990  2012.0          gmc  sierra 2500 hd extended cab       good
33  34590  2016.0    chevrolet        silverado 1500 double       good
34  35000  2019.0       toyota                      tacoma  excellent
35  29990  2016.0    chevrolet        colorado extended cab       good
36  38590  2011.0    chevrolet         corvette grand sport       good
```

To avoid any duplicate model names that are actually from a different manufacturer, we change the model name to be the combination of manufacturer name and model name.

In [3]:
```
new_model_name = df["manufacturer"] + df["model"]
df["model"] = new_model_name
```

Now we can take a look at our dataset. We have 1198390 rows of data, and we have the price, year, manufacturer, model and condition of the cars for our columns. The next step is to exclude cars that are in bad conditions. As a consumer, we do not want to buy cars in bad condition because that will cost us a lot of extra money to fix it.

In [4]:
```
print(df.groupby('condition').size())
```

```
condition
excellent     96781
fair           5918
good         115347
like new      19970
new            1130
salvage         532
dtype: int64
```

As we can see, there are 532 cars in our dataframe that has the "salvage" title, meaning that it has been damaged before and the is declared a total loss by the insurance company. we definitely want to exclude that from our dataset. And we want to exclude "fair" from our dataset, too, because most sellers have a tendency to prettify their car's condition so that they could sell it in a higher price. A car that has fair condition usually indicates that the condition is bad.

Besides, we are only choosing cars that are newer than 2015 and more than 10000 dollars and less than 50000 because the cars that are too old do not have a normal depreciation rate and non-expensive cars usually cost between 10000 to 50000 dollars.

In [5]:
```
df = df[df.condition != "salvage"]
df = df[df.condition != "fair"]
df = df[df.year >= 2015.0]
df = df[df["price"].between(10000, 50000)]
print(df.head(10))
print(df.size)
```

```
    price    year manufacturer                          model  condition
29  39590  2020.0    chevrolet     chevroletsilverado 1500 crew       good
30  30990  2017.0       toyota       toyotatundra double cab sr       good
33  34590  2016.0    chevrolet   chevroletsilverado 1500 double       good
34  35000  2019.0       toyota                    toyotatacoma  excellent
35  29990  2016.0    chevrolet   chevroletcolorado extended cab       good
38  32990  2017.0         jeep     jeepwrangler unlimited sport       good
39  24590  2017.0    chevrolet  chevroletsilverado 1500 regular       good
40  30990  2016.0    chevrolet   chevroletcolorado crew cab z71       good
42  37990  2016.0    chevrolet      chevroletcamaro ss coupe 2d       good
44  30990  2019.0         ford   fordranger supercrew xl pickup       good
422435
```

The mean of the price of different models:

In [6]:
```
man_model_df = df.groupby(["manufacturer", "model","year"]).mean()
print(man_model_df)
```

```
                                                          price
manufacturer model                          year
acura        acurailx                       2015.0  12190.000000
                                            2016.0  16928.166667
                                            2017.0  17914.666667
                                            2018.0  19762.800000
                                            2019.0  24147.500000
...                                                       ...
volvo        volvoxc90 t6 momentum sport    2018.0  43990.000000
                                            2020.0  45990.000000
             volvoxc90 t8 hybrid inscription 2019.0  34333.333333
             volvoxc90 t8 inscription       2016.0  25000.000000
             volvoxc90t6 awd 24733 miles    2018.0  42864.466667

[10562 rows x 1 columns]
```

The range of years:

In [7]:
```
print(df.sort_values(by = ["year"], ascending=False))
# we could see that the latest year is 2022
```

```
        price    year manufacturer                          model  condition
402550  22998  2022.0   mitsubishi         mitsubishieclipse cross        new
404508  29998  2022.0   mitsubishi         mitsubishieclipse cross        new
404014  27998  2022.0   mitsubishi         mitsubishieclipse cross        new
404131  27998  2022.0   mitsubishi         mitsubishieclipse cross        new
403565  37500  2022.0   mitsubishi             mitsubishioutlander        new
...       ...     ...          ...                          ...        ...
213777  11999  2015.0       nissan                  nissanaltima  excellent
213778  48995  2015.0    chevrolet       chevroletsilverado 3500hd   like new
365670  34990  2015.0    chevrolet     chevroletsuburban ltz sport       good
37638   32990  2015.0    chevrolet  chevroletcamaro ss convertible       good
342597  24495  2015.0       toyota                    toyotatacoma  excellent

[84487 rows x 5 columns]
```

We want to keep the models that have more than 1 data because we cannot calculate the depreciation rate based on single data. We also delete all the rows that have car from a single

year for the same reason.

In [8]:
```
df = df[df.duplicated(subset=["model"], keep=False)]
group_by_data = df.groupby(["model","year"], as_index=False).mean()

# delete all the rows that only has car from a single year.
df_clean = group_by_data[group_by_data.duplicated(subset=["model"], keep=False)]
print(df_clean)
```

```
                              model    year         price
0                          acurailx  2015.0  12190.000000
1                          acurailx  2016.0  16928.166667
2                          acurailx  2017.0  17914.666667
3                          acurailx  2018.0  19762.800000
4                          acurailx  2019.0  24147.500000
...                             ...     ...           ...
8582  volvoxc90 t6 inscription sport  2018.0  38990.000000
8583     volvoxc90 t6 momentum sport  2016.0  31590.000000
8584     volvoxc90 t6 momentum sport  2017.0  36990.000000
8585     volvoxc90 t6 momentum sport  2018.0  43990.000000
8586     volvoxc90 t6 momentum sport  2020.0  45990.000000

[7139 rows x 3 columns]
```

Now we complete all the data cleaning and selection process. We have 7145 data. Since we used the group by statement, the year is already in ascending order.

Model

We use the straight line formula depreciation formula to calculate the depreciation rate.The formula is (latest year price - oldest year price) / oldest year price / year difference. We use a while loop to calculate all depreciation rate based on the groupby dataframe above.

In [9]:
```
df_clean.reset_index()


row_num = df_clean.shape[0]  # number of rows in the clean dataframe
depreciation_ratio_lst = [];
i = 0
model_new_price = 0
newer_year = 0
temp_lst_model_year = []
temp_lst_model_price = []

while i < row_num:
    model_name = df_clean.iat[i, 0]
    try:
        if df_clean.iat[i + 1, 0] == model_name:
            temp_lst_model_year.append(df_clean.iat[i, 1])
            temp_lst_model_price.append(df_clean.iat[i, 2])
            i += 1
            if df_clean.iat[i + 1, 0] != model_name:
                newer_year = df_clean.iat[i, 1]
                model_new_price = df_clean.iat[i, 2]
                depreciation_rate = (model_new_price - temp_lst_model_price[0]) / tem
                                    (newer_year - temp_lst_model_year[0])
                depreciation_ratio_lst.append([model_name, depreciation_rate])
                temp_lst_model_year = []
                temp_lst_model_price = []
                i += 1
                continue
```

```
            else:
                newer_year = df_clean.iat[i, 1]
                model_new_price = df_clean.iat[i, 2]
                i += 1
                continue
        else:
            # we use the straight line formula depreciation formula here to calculate
            newer_year = df_clean.iat[i, 1]
            model_new_price = df_clean.iat[i, 2]
            depreciation_rate = (model_new_price - temp_lst_model_price[0]) / temp_ls
                                (newer_year - temp_lst_model_year[0])
            depreciation_ratio_lst.append([model_name, depreciation_rate])
            temp_lst_model_year = []
            temp_lst_model_price = []
            i += 1
            continue
    # this is used to handle the last element that could be out of index bound because
    except IndexError as error:
        break

depreciation_ratio_lst.sort(key=lambda x: x[1])

top_value_model_name = []
for i in range(50):
    top_value_model_name.append(depreciation_ratio_lst[i])

result = df[np.isin(df, top_value_model_name).any(axis=1)]

group_by_result = result.groupby(["model"]).count()
group_by_result.reset_index()
# we decide to require all car models to have more than 10 sets of data to add accurac
top_ten_group_by = group_by_result[(group_by_result > 10).any(axis=1)]


top_ten_list = list(top_ten_group_by.index)

final_df = df_clean[np.isin(df_clean, top_ten_list).any(axis=1)]
```

Results

In [10]:

```
for i in depreciation_ratio_lst:
    for j in top_ten_list:
        if i[0] == j:
            print(i)
```

```
['gmcsavana commercial cutaway', -0.37509377344336087]
['kiaoptima ex sedan 4d', -0.3684728525343102]
['fordsuper duty f-550 drw', -0.2627625678793183]
['gmcacadia slt-2 sport utility', -0.20060316492155467]
['lexusis 250', -0.19486351344341274]
['dodgedurango limited', -0.14757397270897873]
['bmw4 series 428i xdrive', -0.14520955940260896]
['bmw4 series 430i xdrive', -0.13895647472289807]
['dodgedart', -0.12577209237032358]
['chryslertown & country', -0.11216859376563239]
```

These are the top ten non-expensive cars with least depreciation ratio. The data of these models are shown as below:

In [11]:

```
print(final_df)
print(final_df.groupby(["model", "year"]).mean())
```

	model	year	price
555	bmw4 series 428i xdrive	2015.0	29098.750000
556	bmw4 series 428i xdrive	2016.0	24873.333333
562	bmw4 series 430i xdrive	2018.0	36990.000000
563	bmw4 series 430i xdrive	2019.0	31850.000000
2000	chryslertown & country	2015.0	15046.666667
2001	chryslertown & country	2016.0	13358.903226
2145	dodgedart	2015.0	14247.000000
2146	dodgedart	2016.0	12455.125000
2179	dodgedurango limited	2015.0	22271.727273
2180	dodgedurango limited	2016.0	18985.000000
3197	fordsuper duty f-550 drw	2015.0	49758.958333
3198	fordsuper duty f-550 drw	2016.0	36684.166667
3442	gmcacadia slt-2 sport utility	2018.0	32790.000000
3443	gmcacadia slt-2 sport utility	2019.0	26212.222222
3494	gmcsavana commercial cutaway	2016.0	39990.000000
3495	gmcsavana commercial cutaway	2017.0	24990.000000
5104	kiaoptima ex sedan 4d	2015.0	16626.363636
5105	kiaoptima ex sedan 4d	2016.0	10500.000000
5387	lexusis 250	2015.0	21113.190476
5388	lexusis 250	2016.0	16999.000000

		price
model	year	
bmw4 series 428i xdrive	2015.0	29098.750000
	2016.0	24873.333333
bmw4 series 430i xdrive	2018.0	36990.000000
	2019.0	31850.000000
chryslertown & country	2015.0	15046.666667
	2016.0	13358.903226
dodgedart	2015.0	14247.000000
	2016.0	12455.125000
dodgedurango limited	2015.0	22271.727273
	2016.0	18985.000000
fordsuper duty f-550 drw	2015.0	49758.958333
	2016.0	36684.166667
gmcacadia slt-2 sport utility	2018.0	32790.000000
	2019.0	26212.222222
gmcsavana commercial cutaway	2016.0	39990.000000
	2017.0	24990.000000
kiaoptima ex sedan 4d	2015.0	16626.363636
	2016.0	10500.000000
lexusis 250	2015.0	21113.190476
	2016.0	16999.000000

Conclusion

The top 10 non-expensive cars that has the lowest depreciation ratio are:

1. Gmc savana commercial cutaway
2. kiao ptima ex sedan 4d
3. ford super duty f-550 drw
4. gmc acadia slt-2 sport utility
5. lexus is 250
6. dodge durango limited
7. bmw4 series 428i xdrive
8. bmw4 series 430i xdrive
9. dodge dart 10.chrysler town & country

The finding is very interesting because all of the cars above have negative inflation rate. However, we know that the calculation is correct by looking at the final_df above and do some

calculation by ourselves. We also went on cardealer websites such as carmax to check if it is possible for cars to have negative inflation rate. Surprisingly, we do find some cars listed on our top-10 rankings usually have a very low depreciation ratio and sometimes negative ratio on Carmax.

The explaination for the negativce depreciation ratio is the inflation we are experiencing now in the US. Because the microchip industry has been heavily affected by Covid-19 and many factories in worldwide had to shut down, the car industry was also strongly affected because cars need microchip. To avoid waiting a long period time for a car, the consumer demand of cars goes from buying a new car to buying a used car,making the used car market heated in the past year.