Importing

In [3]:
```
# Import libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
import statsmodels.api as sm
from sklearn.ensemble import RandomForestRegressor
from sklearn import decomposition
from sklearn import model_selection
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
```

Engineering features

Read data

In [4]:
```
# Read data
total_units = pd.read_csv('data/BrandTotalUnits.csv')
total_sales = pd.read_csv('data/BrandTotalSales.csv')
details = pd.read_csv('data/BrandDetails.csv')
avg_price = pd.read_csv('data/BrandAverageRetailPrice.csv')
```

Convert months to datatime

In [5]:
```
# Convert months to datatime
total_units['Months'] = pd.to_datetime(total_units['Months'])
total_sales['Months'] = pd.to_datetime(total_sales['Months'])
avg_price['Months'] = pd.to_datetime(avg_price['Months'])
```

Trim 'Total Units'and convert to float

In [6]:
```
# Trim 'Total Units'and convert to float
total_units['Total Units'] = total_units['Total Units'].str[:8]
total_units['Total Units'] = total_units['Total Units'].str.replace(',', '').astype(float)
total_units['Total Units'] = pd.to_numeric(total_units['Total Units'])
```

Trim 'Total Sales'and convert to float

In [7]:
```
# Trim 'Total Sales'and convert to float
total_sales['Total Sales ($)'] = total_sales['Total Sales ($)'].str[:8]
total_sales['Total Sales ($)'] = total_sales['Total Sales ($)'].str.replace(',', '')
total_sales['Total Sales ($)'] = pd.to_numeric(total_sales['Total Sales ($)'])
```

In [8]:
```
total_units.head()
```

Out[8]:

	Brands	Months	Total Units	vs. Prior Period
0	#BlackSeries	2020-08-01	1616.3300	NaN
1	#BlackSeries	2020-09-01	NaN	-1.000000
2	#BlackSeries	2021-01-01	715.5328	NaN
3	#BlackSeries	2021-02-01	766.6691	0.071466
4	#BlackSeries	2021-03-01	NaN	-1.000000

In [9]:
```
total_sales.head()
```

Out[9]:

	Months	Brand	Total Sales ($)
0	2018-09-01	10x Infused	1711.33
1	2018-09-01	1964 Supply Co.	25475.20
2	2018-09-01	3 Bros Grow	120153.00
3	2018-09-01	3 Leaf	6063.52
4	2018-09-01	350 Fire	631510.00

In [10]:
```
details.head()
```

Out[10]:

	State	Channel	Category L1	Category L2	Category L3	Category L4	Category L5	Brand	Product Description	Total Sales ($)	...	Total THC	Tot CE
0	California	Licensed	Inhaleables	Flower	Hybrid	NaN	NaN	#BlackSeries	#BlackSeries - Vanilla Frosting - Flower (Gram)	1,103.964857	...	0	
1	California	Licensed	Inhaleables	Flower	Hybrid	NaN	NaN	#BlackSeries	#BlackSeries - Vanilla Frosting - Flower (Gram)	674.645211	...	0	
2	California	Licensed	Inhaleables	Flower	Sativa Dominant	NaN	NaN	#BlackSeries	#BlackSeries - Blueberry Slushy - Flower (Gram)	2,473.699102	...	0	
3	California	Licensed	Inhaleables	Flower	Sativa Dominant	NaN	NaN	#BlackSeries	#BlackSeries - Blueberry Slushy - Flower (Gram)	14,589.916417	...	0	
4	California	Licensed	Inhaleables	Concentrates	Dabbable Concentrates	Wax	NaN	101 Cannabis Co.	101 Cannabis Co. - Afghan Kush - Wax	145.39627	...	0	

5 rows × 25 columns

In [11]:
```python
avg_price.head()
```

Out[11]:

	Brands	Months	ARP	vs. Prior Period
0	#BlackSeries	2020-08-01	15.684913	NaN
1	#BlackSeries	2020-09-01	NaN	-1.000000
2	#BlackSeries	2021-01-01	13.611428	NaN
3	#BlackSeries	2021-02-01	11.873182	-0.127705
4	#BlackSeries	2021-03-01	NaN	-1.000000

Change column names to merge

In [12]:
```python
# Change column names to merge
total_sales = total_sales.rename(columns={"Brand": "Brands"})
total_units = total_units.rename(columns={"vs. Prior Period": "vs. Prior Period(total_units)"})
avg_price = avg_price.rename(columns={"vs. Prior Period": "vs. Prior Period(avg_price)"})
```

In [13]:
```python
brands = total_units["Brands"].unique()
brands
```

Out[13]:
```
array(['#BlackSeries', '101 Cannabis Co.', '10x Infused', ..., 'Zlixir',
       'Zoma', 'Zuma Topicals'], dtype=object)
```

Create Time Series Features and merge data

In [14]:
```
# Create Time Series Features and merge data
merged = pd.DataFrame()
for brand in brands:
    units = total_units[total_units.Brands == brand]
    units.loc[:,'Previous Month Units'] = units.loc[:,'Total Units'].shift(1)
    units.loc[:,'Rolling Average Units'] = (units.loc[:,'Total Units'].shift(1) + units.loc[:,'Total Units'].shi
                                  units.loc[:,'Total Units'].shift(3))/3

    sales = total_sales[total_sales.Brands == brand]
    sales.loc[:,'Previous Month Sales'] = sales.loc[:,'Total Sales ($)'].shift(1)
    sales.loc[:,'Rolling Average Sales'] = (sales.loc[:,'Total Sales ($)'].shift(1) + sales.loc[:,'Total Sales (
                                  sales.loc[:,'Total Sales ($)'].shift(3))/3

    arp = avg_price[avg_price.Brands == brand]
    arp.loc[:,'Previous Month ARP'] = arp.loc[:,'ARP'].shift(1)
    arp.loc[:,'Rolling Average ARP'] = (arp.loc[:,'ARP'].shift(1) + arp.loc[:,'ARP'].shift(2) +
                                  arp.loc[:,'ARP'].shift(3))/3

    branddetails = details[details.Brand == brand]
    productcount = (branddetails.Brand == brand).count()

    units = units.merge(sales, on=['Brands', 'Months'], how='left')
    units = units.merge(arp, on=['Brands', 'Months'], how='left')
    units['ProdCount'] = productcount
    merged = pd.concat([merged, units], ignore_index=True)
```

```
D:\software\Anaconda\lib\site-packages\pandas\core\indexing.py:1667: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  self.obj[key] = value
```

In [15]:
```
merged_copy = merged.copy()
```

In [16]:
```
#merged_copy.head(10)
```

Convert "Months" to numerical values

In [17]:
```
# Convert "Months" to numerical values
merged_copy.insert(2, "Month", merged_copy['Months'].
                   apply(lambda datetime: str(datetime).split('-')[1].replace('0', '')).astype(int))
merged_copy = merged_copy.loc[merged_copy['ProdCount'] != 0]
```

In [18]:
```
#merged_copy
```

Drop rows with nan total sales and rolling average sales. These two are very important features. I think that if these two features are nan, then imputing on that row is highly likely inappropriate.

In [19]:
```
merged_copy = merged_copy.dropna(subset=['Total Sales ($)', 'Rolling Average Sales']).reset_index(drop=True)
```

Impute other features with nan with median of the same brands

In [20]:
```
# Impute with median of the same brands
for i, brand in enumerate(brands):
    for col in merged_copy.columns[merged_copy.isna().any()].tolist():
        median = merged_copy.loc[merged_copy.Brands==brand, col].median()
        merged_copy.loc[merged_copy['Brands'] == brand, col] = median
```

Augment features

In [21]:
```
# Augment features
merged_copy['sales_per_prod'] = merged_copy['Total Sales ($)'] / merged_copy['ProdCount']
final_data = merged_copy.copy()
```

Drop all the rows with nan

In [22]:
```
final_data = final_data.dropna()
```

Extract labels

In [23]:
```
# Extract labels
labels = final_data['Total Sales ($)']
```

Drop columns that are NOT useful. Many of these are highly correlated to the labels. Brands are important, but my thought is that, if the total sales are high because of the brands, then we can simply extract that information from the sales information from the past, so rolling average sale is already a good indication. Moreover, there are too many brands in the dataset, one hot encoding them makes the dataset even more complex.

In [24]:
```
# Drop columns that are NOT useful
final_data = final_data.drop(columns=['Brands', 'Months', 'vs. Prior Period(total_units)', 'Total Units', 'Total
                              'ARP', 'vs. Prior Period(avg_price)'])
```

In [25]:
```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18127 entries, 0 to 18238
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Month                   18127 non-null  int32  
 1   Previous Month Units    18127 non-null  float64
 2   Rolling Average Units   18127 non-null  float64
 3   Previous Month Sales    18127 non-null  float64
 4   Rolling Average Sales   18127 non-null  float64
 5   Previous Month ARP      18127 non-null  float64
 6   Rolling Average ARP     18127 non-null  float64
 7   ProdCount               18127 non-null  int64  
 8   sales_per_prod          18127 non-null  float64
dtypes: float64(7), int32(1), int64(1)
memory usage: 1.3 MB
```

In [26]:
```
final_data[final_data.isna().any(axis=1)].head(10)
```

Out[26]:

Month	Previous Month Units	Rolling Average Units	Previous Month Sales	Rolling Average Sales	Previous Month ARP	Rolling Average ARP	ProdCount	sales_per_prod

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(final_data, labels, test_size=0.2, random_state=42)
```

Linear Regression

In [26]:
```
linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_prediciton_train = linreg.predict(X_train)
y_prediciton_test = linreg.predict(X_test)

mse = metrics.mean_squared_error(y_train, y_prediciton_train)
r2 = metrics.r2_score(y_train, y_prediciton_train)
print("Training Root Mean Squared Error:", mse)
print("Test r2:", r2)

mse = metrics.mean_squared_error(y_test, y_prediciton_test)
r2 = metrics.r2_score(y_test, y_prediciton_test)
print("Training Root Mean Squared Error:", mse)
print("Test r2:", r2)
```

```
Training Root Mean Squared Error: 11973830199.457659
Test r2: 0.7111748690489963
Training Root Mean Squared Error: 12616248996.257673
Test r2: 0.7127367646798193
```

OLS

In [29]:
```
stats = sm.OLS(labels, final_data)
result = stats.fit()
print(result.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:        Total Sales ($)   R-squared (uncentered):                   0.829
Model:                            OLS   Adj. R-squared (uncentered):              0.829
Method:                 Least Squares   F-statistic:                              9755.
Date:                Wed, 16 Mar 2022   Prob (F-statistic):                        0.00
Time:                        22:53:25   Log-Likelihood:                     -2.3617e+05
No. Observations:               18127   AIC:                                  4.724e+05
Df Residuals:                   18118   BIC:                                  4.724e+05
Df Model:                           9
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
Month                   657.5951    187.355      3.510      0.000     290.362    1024.828
Previous Month Units     -0.0486      0.167     -0.291      0.771      -0.375       0.278
Rolling Average Units     0.2096      0.171      1.227      0.220      -0.125       0.545
Previous Month Sales      0.4639      0.010     44.469      0.000       0.443       0.484
Rolling Average Sales     0.3960      0.011     35.034      0.000       0.374       0.418
Previous Month ARP      185.1808    621.462      0.298      0.766   -1032.944    1403.305
Rolling Average ARP      78.6431    617.290      0.127      0.899   -1131.303    1288.590
ProdCount                 8.2403      1.784      4.618      0.000       4.743      11.738
sales_per_prod            2.0428      0.107     19.013      0.000       1.832       2.253
==============================================================================
Omnibus:                     6655.482   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           295762.533
Skew:                           1.038   Prob(JB):                         0.00
Kurtosis:                      22.679   Cond. No.                     3.95e+05
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 3.95e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

We can see that the features about units and arp have high p-value, so they are insignificant.

PCA

In [52]:
```
n_components = range(1, 10)
```

In [53]:
```
for n in n_components:
    pca = decomposition.PCA(n_components=n)
    X_pca_train = pca.fit_transform(X_train)

    linreg = LinearRegression()
    linreg.fit(X_pca_train, y_train)

    X_pca_test = pca.fit_transform(X_test)
    y_prediciton_test = linreg.predict(X_pca_test)

    mse = metrics.mean_squared_error(y_test, y_prediciton_test)
    r2 = metrics.r2_score(y_test, y_prediciton_test)
    print("n_components: ", n)
    print("  Training Root Mean Squared Error:", mse)
    print("  Test r2:", r2)
```

```
n_components:  1
  Training Root Mean Squared Error: 12832165794.832943
  Test r2: 0.7078204889993769
n_components:  2
  Training Root Mean Squared Error: 12823655039.202374
  Test r2: 0.7080142729995327
n_components:  3
  Training Root Mean Squared Error: 12828277859.866404
  Test r2: 0.7079090145807547
n_components:  4
  Training Root Mean Squared Error: 12657154625.522444
  Test r2: 0.7118053718855829
n_components:  5
  Training Root Mean Squared Error: 12676691279.377382
  Test r2: 0.7113605358336498
n_components:  6
  Training Root Mean Squared Error: 12653359253.796436
  Test r2: 0.7118917898662029
n_components:  7
  Training Root Mean Squared Error: 12653864465.307362
  Test r2: 0.7118802865506617
n_components:  8
  Training Root Mean Squared Error: 12647882607.913212
  Test r2: 0.7120164892927587
n_components:  9
  Training Root Mean Squared Error: 12647771752.145277
  Test r2: 0.7120190134016725
```

The results indicates that using PCA doesn't improve the model performance.

Ensemble

In [204]:
```
n_estimators = [100, 200, 500, 1000]
max_features = [2, 4, 6, 8]
max_depth = [5, 10, 20, 30]
```

In [205]:
```
best_score = 0
best_model = ()
for n in n_estimators:
    for f in max_features:
        for d in max_depth:
            regr = RandomForestRegressor(n_estimators=n, max_features=f,
                                         max_depth=d, random_state=10)
            regr.fit(X_train, y_train)

            score = regr.score(X_test, y_test, sample_weight=None)
            if (score > best_score):
                best_score = score
                best_model = (n, f, d)

print(n, f, d, ": ", score)
```

```
1000 8 30 :  0.9584578634369649
```

Random forest regressors are used for the ensemble method. I also use a for loop to try to tune the hyperparameter. The score is actually very high.

Cross Validation

In [58]:
```
kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle=True)
```

Cross-validate on linear regression

In [62]:
```
linreg_model_kfold = LinearRegression()
linreg_results_kfold = model_selection.cross_val_score(linreg_model_kfold, X_train, y_train, cv=kfold)

print("Linear Regression Score: %.2f%%" % (linreg_results_kfold.mean()*100.0))
```

```
Linear Regression Score: 71.08%
```

Cross-validate on ensemble

In [61]:
```
rfregr_model_kfold = RandomForestRegressor(n_estimators=1000, max_features=8, max_depth=30, random_state=10)
rfregr_result_kfold = model_selection.cross_val_score(rfregr_model_kfold, X_train, y_train, cv=kfold)

print("Random Forest Regression Score: %.2f%%" % (rfregr_result_kfold.mean()*100.0))
```

```
Random Forest Regression Score: 95.38%
```

The results of cross validations on both models indicates that the performnace is the average performance.

Grid Search

In [70]:
```
params = {
    'n_estimators': [100, 200, 500, 1000],
    'max_features': [2, 4, 6, 8],
    'max_depth': [5, 10, 20, 30]
}

rfregr = RandomForestRegressor()
gridSearchCV = model_selection.GridSearchCV(
    estimator=rfregr,
    param_grid=params,
    cv=10,
    n_jobs=-1
)

grid_result = gridSearchCV.fit(final_data, labels)
```

In [71]:
```
print('best score: ', grid_result.best_score_)
print('best params: ', grid_result.best_params_)
```

```
best score:  0.9523926234788206
best params:  {'max_depth': 30, 'max_features': 8, 'n_estimators': 1000}
```

The results of Grid Search also indicates that the hyperparameters are optimal.

My own model - Neural Netowrk

Grid Search and Cross-validation

In [75]:

```
params = {
    'max_iter': [5, 10, 20, 30, 50],
    'hidden_layer_sizes': [10, 20, 50, 100, 200]
}

nn = MLPRegressor(early_stopping=True,
                    solver='sgd',
                    batch_size=100,
                    learning_rate='adaptive'
)

gridSearchCV_nn = model_selection.GridSearchCV(
    estimator=nn,
    param_grid=params,
    cv=5
)

grid_result_nn = gridSearchCV_nn.fit(final_data, labels)
```

```
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "D:\software\Anaconda\lib\site-packages\sklearn\metrics\_regression.py", line 90, in _check_reg_targe
ts
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "D:\software\Anaconda\lib\site-packages\sklearn\utils\validation.py", line 63, in inner_f
    return f(*args, **kwargs)
  File "D:\software\Anaconda\lib\site-packages\sklearn\utils\validation.py", line 720, in check_array
    _assert_all_finite(array,
  File "D:\software\Anaconda\lib\site-packages\sklearn\utils\validation.py", line 103, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn("Estimator fit failed. The score on this train-test"
D:\software\Anaconda\lib\site-packages\sklearn\model_selection\_search.py:922: UserWarning: One or more of t
he test scores are non-finite: [-1.09535924e+058 -6.92159371e+087 -1.94468664e+102 -2.56207278e+179
 -5.10506079e+056 -1.89661875e+143 -8.35742593e+215 -1.66086048e+256
 -1.10558560e+118 -1.21957864e-001              nan              nan
              nan              nan              nan              nan
              nan              nan              nan              nan
              nan              nan              nan              nan
```

In [76]:
```
print('best score: ', grid_result_nn.best_score_)
print('best params: ', grid_result_nn.best_params_)
```

```
best score:  -0.12195786372558581
best params:  {'hidden_layer_sizes': 20, 'max_iter': 50}
```

My Custom Model - KNN

In [30]:
```
params = {
    'n_neighbors': [5, 10, 20, 50, 100, 200, 500],
    'weights': ['uniform', 'distance'],
    'n_jobs': [-1]
}

knn = KNeighborsRegressor()
gridSearchCV_knn = model_selection.GridSearchCV(
    estimator=knn,
    param_grid=params,
    cv=5
)

grid_result_knn = gridSearchCV_knn.fit(final_data, labels)
```

In [32]:
```
print('best score: ', grid_result_knn.best_score_)
print('best params: ', grid_result_knn.best_params_)
```

```
best score:  0.7344368153851452
best params:  {'n_jobs': -1, 'n_neighbors': 100, 'weights': 'distance'}
```

Comparing Models

In this project, I trained 3 models: linear regression model, random forest regressor, multilayer neural network regressor, and knn regressor. Apparently, the random forest regressor has the best performance. However, the downside is quite obvious too, which is that the amount of time it takes to train such a model is very long.

In [ ]:

Project 3 Report
Lingtao Chen
305547772
03/17/2022

Background / Introduction

The cannabis market is one of the fastest growing markets in the world. There are many opportunities in the market, so there will be more and more companies trying to take a bite. With the power to predict sales every month, companies will have more advantages to grow.

I am trying to train a model based on the provided data to predict the total sale. I would engineer the features to train models better. My models of choice are linear regression, random forest regression, neural network, and k-nearest neighbor.

Methodology

We are provided with 4 important csv files. Each contains important information about units, average product price(ARP), sales, and details about the companies. First of all, we have to merge these data together so that we can access and engineer the features easier.

Time series is an important idea, so we can exploit the data in the past to predict what it will be like in the future. For the same brands, each row will have a few additional time series features, which are previous-month total units, previous-month ARP, and previous-month total sales. These features can be simply extracted from the data in the previous month. Moreover, rolling average total units, rolling average ARP, and rolling average total sales are created for each row by computing the average based on the past three months. I combine the time series features engineering and data merging in a for loop. For each brand, I engineer the time series features in different dataset and merge them together based on brand and month. Finally, I append these dataset into a final dataset.

The next step is to deal with the nan values in the dataset. My first strategy is to impute the nan values with the median within each brand. In this dataset, there are so many brands that don't have sufficient data, so I just drop these rows, even, the entire brands.

Some additional features are also needed to boost my dataset, so I add two additional features: the number of products for each brand and sales per product in the brand level. I think these two are important, because they give us a good insight about the size of the brands.

I later decided not to use the brands, because to some extent, the units and sales can explain well how the brands are doing. As long as we have the rolling average data, we can determine if the sizes of the brands are large or not. Moreover, I think using brands in the models is not a wise choice. One hot encoding the brands will increase the dimensionality so much that models are not simple, and the amount of time it takes to train models will be increased too. Besides the brands, I also drop the 'vs. Prior Period(total_units)', 'Total Units', 'Total Sales ($), 'ARP', 'vs. Prior Period(avg_price)', because these are somehow like total sales, the future data we want to predict.

When training some specific models, we have to tune the hyperparameters to get the "optimal" models, such as the number of nearest neighbors for k-nearest neighbors, the max depth for decision trees, and etc… We have to plug in different combinations of hyperparameters to find the best one. We usually split a dataset into the training part and test part. However, it's also usual that the splitting is just accidentally good and that the model performs very well. Therefore, we need to use cross-validation to make sure that the models are average good. GridSearchCV is used to not only find the best parameters, but also the parameters are average good.

Results

```
Training Root Mean Squared Error: 11973830199.457659
Test r2: 0.7111748690489963
Training Root Mean Squared Error: 12616248996.257673
Test r2: 0.7127367646798193
```

The r-squares for the linear regression model is 0.713, meaning the model doesn't do a very good job on predicting the total sales.

```
                            OLS Regression Results
=======================================================================================
Dep. Variable:        Total Sales ($)   R-squared (uncentered):                   0.829
Model:                            OLS   Adj. R-squared (uncentered):              0.829
Method:                 Least Squares   F-statistic:                              9755.
Date:                Wed, 16 Mar 2022   Prob (F-statistic):                        0.00
Time:                        22:53:25   Log-Likelihood:                     -2.3617e+05
No. Observations:               18127   AIC:                                  4.724e+05
Df Residuals:                   18118   BIC:                                  4.724e+05
Df Model:                           9
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
Month                   657.5951    187.355      3.510      0.000     290.362    1024.828
Previous Month Units     -0.0486      0.167     -0.291      0.771      -0.375       0.278
Rolling Average Units     0.2096      0.171      1.227      0.220      -0.125       0.545
Previous Month Sales      0.4639      0.010     44.469      0.000       0.443       0.484
Rolling Average Sales     0.3960      0.011     35.034      0.000       0.374       0.418
Previous Month ARP      185.1808    621.462      0.298      0.766   -1032.944    1403.305
Rolling Average ARP      78.6431    617.290      0.127      0.899   -1131.303    1288.590
ProdCount                 8.2403      1.784      4.618      0.000       4.743      11.738
sales_per_prod            2.0428      0.107     19.013      0.000       1.832       2.253
==============================================================================
Omnibus:                     6655.482   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           295762.533
Skew:                           1.038   Prob(JB):                         0.00
Kurtosis:                      22.679   Cond. No.                     3.95e+05
==============================================================================
```

We can see that the p-values of the units-related and ARP-related features are greater than 0.05, so they are insignificant.

```
n_components:  1
  Training Root Mean Squared Error: 12832165794.832943
  Test r2: 0.7078204889993769
n_components:  2
  Training Root Mean Squared Error: 12823655039.202374
  Test r2: 0.7080142729995327
n_components:  3
  Training Root Mean Squared Error: 12828277859.866404
  Test r2: 0.7079090145807547
n_components:  4
  Training Root Mean Squared Error: 12657154625.522444
  Test r2: 0.7118053718855829
n_components:  5
  Training Root Mean Squared Error: 12676691279.377382
  Test r2: 0.7113605358336498
n_components:  6
  Training Root Mean Squared Error: 12653359253.796436
  Test r2: 0.7118917898662029
n_components:  7
  Training Root Mean Squared Error: 12653864465.307362
  Test r2: 0.7118802865506617
n_components:  8
  Training Root Mean Squared Error: 12647882607.913212
  Test r2: 0.7120164892927587
n_components:  9
  Training Root Mean Squared Error: 12647771752.145277
  Test r2: 0.7120190134016725
```

The r-squared values for different n-components are round 0.71, so it indicates that the dimensionality is not complex.

For the ensemble method, I use a random forest regressor. After tuning the parameters and cross-validation, this model is doing a great job. It has a score of 0.9538. After employing a GridSearchCV on it, it also has a score of 0.9524.

I also train a multilayer neural network and k-nearest neighbors. After employing GridSearchCV on both models, the multilayer neural network has a score of -0.122 and the k-nearest neighbor has a score of 0.7344.

Discussion

After comparing the models, we can see that the random forest regression model has the best performance. But the downside is quite obvious, it takes a long time to train such a model. If we also consider the time taken, linear regression and k-nearest neighbors may also be good models.

A good model cannot be trained based on nothing. Tremendous data is needed to train a good model. There is a lot of missing data, and many brands only have a few months of data. There is no way to predict correctly just based on that few amounts of data. This is also the reason I don’t train models to predict at the brand level. The past data is a very good indicator of how the brands are doing and therefore a good indicator of the size of the brands.