# 目录

# Pre

Xinzhi Huo:

The first step of the program is to find an appropriate dataset to use. First I check the dataset given by teacher which is either unfamiliar or has few properties. I am also worried that we choose the same dataset as other groups. So I turn to the website given by teacher where has so many datasets so that I can choose one we are familiar aswell as appropriate to do machine learning. It is really a little hard to choose a best one in so many datasets. Finally, I choose the Black Friday dataset, and my classmates choose the TMDB 5000 Movie Dataset. We cannot decide which one to use. From my point of view, this is our first time with machine learning, we should choose a dataset which has numerable things to predict. So Black Friday is better, it has very clear relation between properties. Even if we make mistakes in the prediction we can easily discover it, because the relation between data is so strong that we can find the unusual data by simple decision. However, TMDB 5000 Movie Dataset is more difficult to do prediction. The main goal of this dataset is recommending movie for specific person and predict the box office of a given movie. There are 20 columns in the dataset and almost every column is related to the prediction which makes it harder to train the model and control the error bound. Even if we have known the difficulties, because of the interest of the TMDB 5000 Movie Dataset, we still cannot decide which one to choose. Luckily teacher always concerns our condition, and after knows our difficulty, he helps us to choose.

# I. Initialization step

## 1. Study the problem context by choosing the data you want to mine

Difficulty:(Xinzhi Huo)

Since it is the first time for me to do machine learning, I do not have a total idea of how to do it. And that makes me really worried. Without comprehension of machine learning, it is hard to find the difficulty in this program. So I took few days to learn the related knowledge. After known the whole flow and technology to be used, I listed the difficulties both in function and in technique.

Function:

The standard to classify the customer to low, middle and high.

Technique:

An appropriate method viewing the data generally to find some internal relation.

A method to solve the problem of lacking data, add it or ignore it.

An efficient method for this project to calculate the data.

A combination of classification, clustering and association rule discovery.

Compare algorithms and models to choose the most appropriate one.

Li Ma:

In the first step, we decided to choose the dataset BlackFriday for our project, then elaborated the use-case diagram and detailed description of the most important cases and defined the global architecture of our project. After the initial discussion, we assigned our respective tasks and I was responsible for back-end machine learning (Regression for prediction)

The part of backend is machine learning of prediction and classification., my initial vision for the backend framework was Ngnix + uWGSI + Flask(python) + Cloud server and the process is as follows

Web browser sends an HTTP request to Nginx

Nginx forwarded to uWSGI

uWSGI processes it according to the WSGI specification

uWSGI will send HTTP requests to the Flask application for processing

Flask application handles HTTP

Get data from an HTTP request and set it as input for the prediction model(trained)

Flask generates an HTTP response (the prediction result), which is given to uWSGI according to the WSGI specification.

uWSGI sends HTTP response to Nginx via TCP

Nginx sends the HTTP response to the web browser via TCP

Zhelin Liang:

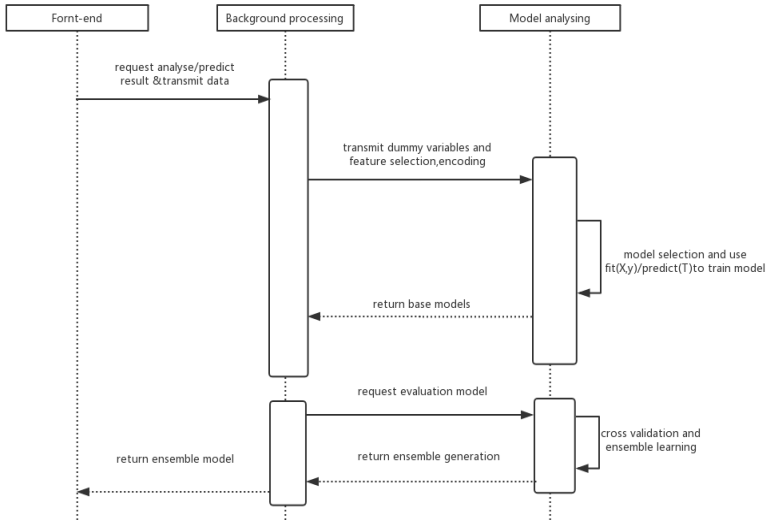
The BlackFriday.csv, includes 13 eigenquantities: User_ID, Product_ID, Gender, Age, Occupation, City, Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2, Product_Category_3 and Purchase. Based on these trait quantities, we can predict the purchase of a specific group of people, and predict a person's gender or marital status based on the goods they buy and other valid information

## 2. Elaborate the Use-case diagram and detailed description of the most important cases

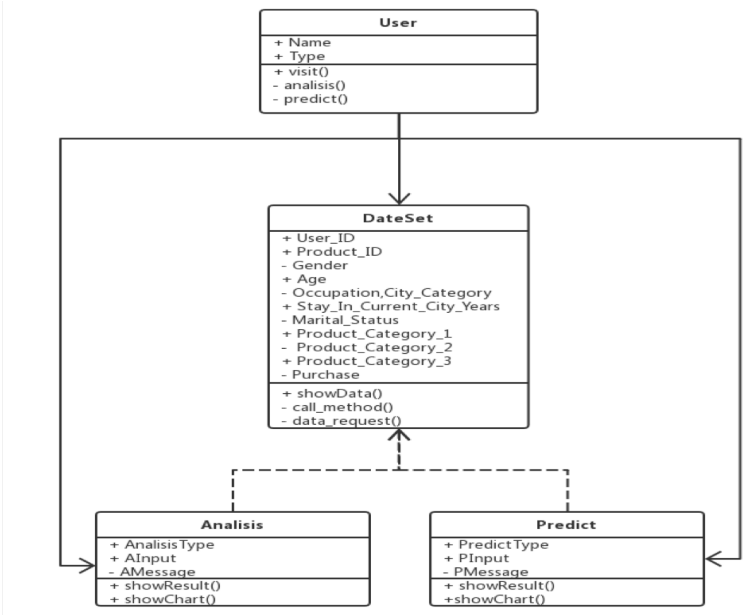
Yuxuan He:

Use case:

Sequence_Diagram : Learning model

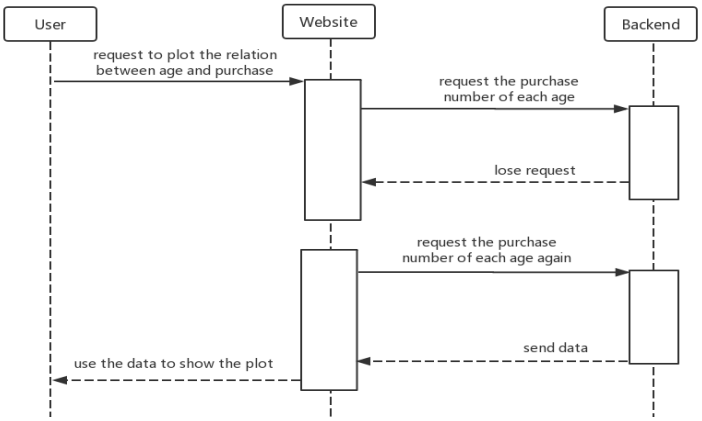



Class_diagram :



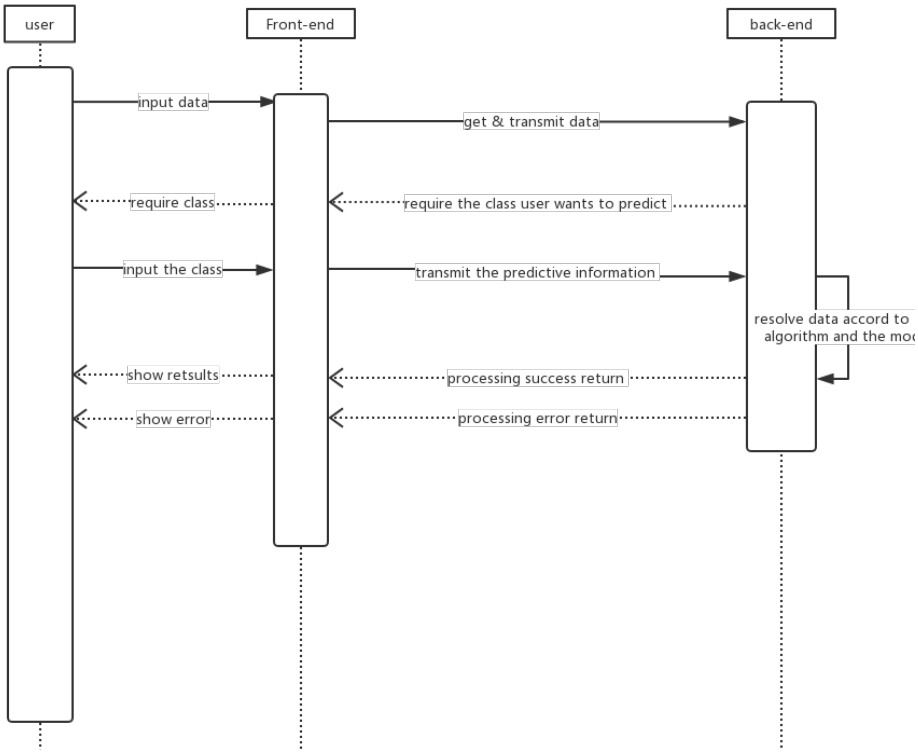

Xinzhi Huo:

Sequence_Diagram : Statistical data visulization



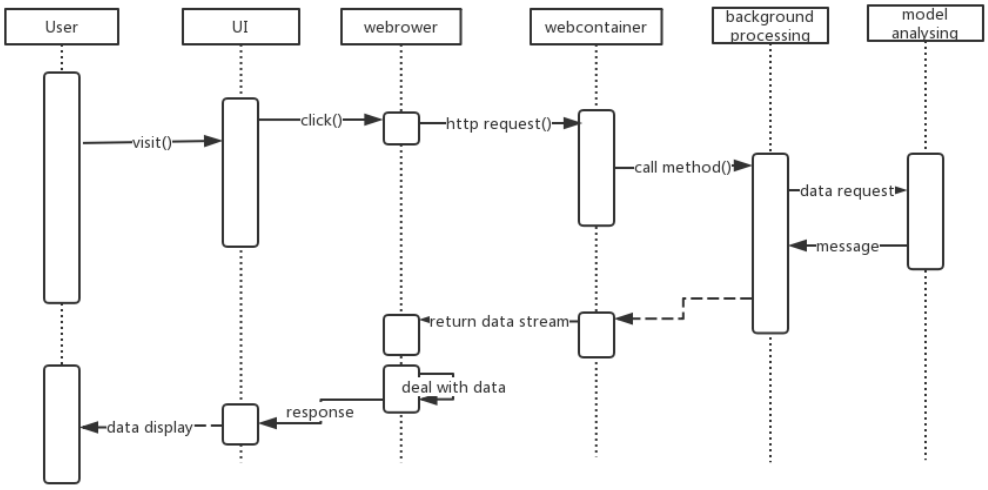

Zhelin Liang :

data_prediction

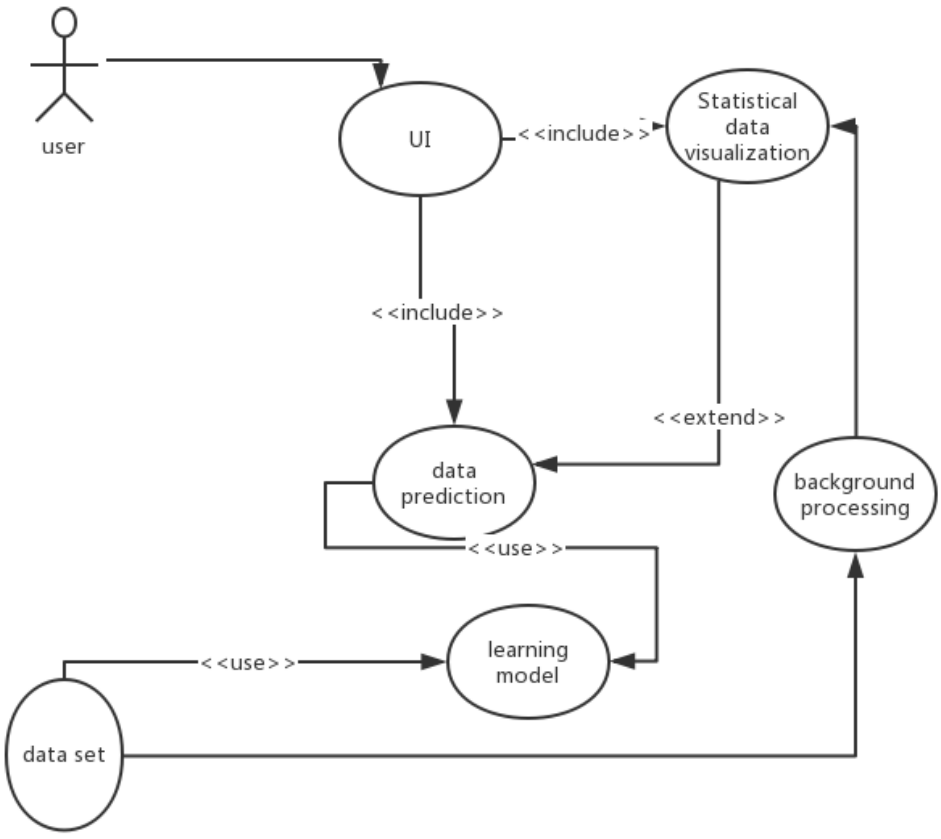



Li Ma:

Sequence_Diagram : data-analysis



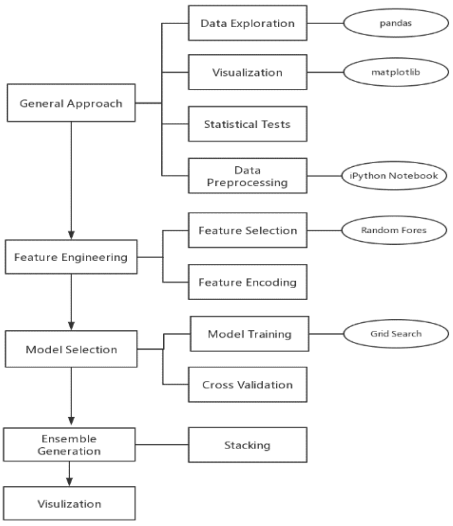

data-use-case :



## 3. Define the global architecture of the Project

Xinzhi Huo:

With the knowledge I learned, I write a global architecture for our program. When I wrote this, I just knew some rough knowledge. But I try to make it reasonable and feasible. And later I found that it is useful for the timeline and separating work. Although I am not sure if the technology I list in the graph is the best, it at least lead to a direction.




## 4. Separate the work and Make timeline

Here is the original timeline of the program. It is hardly changed later. And we Add more details on it.

Timeline

| Task | assignment | Time |
|---|---|---|
| Graphical User Interface | Zhelin Liang | 5.6- 5.19 |
| Data analysis/statistics and visualization. | Xinzhi Huo | 5.6- 5.19 |
| Machine Learning-Data<br>(1)Analyze the correlation of data in each column.<br>(2)Set predictive goals.<br>(2)Select appropriate features and preprocess the data. | Li Ma with Xinzhi Huo | 5.6-6.10 |
| Machine Learning-Model<br>(1)Select several models to calculate and compare their performance.<br>(2)Perhaps use multiple models together. | Yuxuan He with Zhelin Liang | 5.6-6.10 |

# II. Elaboration step

## 1. Detailed architecture of the Project by describing all the functionalities and the employed languages

Zhelin Liang:

We need to do is a web app, reading black Friday from the python client data, first carries on the data cleaning and transformation, and then extract the data in the useful information for us, carries on the analysis and using charts (such as a bar chart or pie chart) on a web page, and then we through machine learning structure data model using regression or classification algorithm, finally, we can on the web page input data to predict and forecast results are presented through the python. In the back-end we use Python and flask, In the front end we use JavaScript, bootstrap, Echarts and GoogleFont.

## 2. Scraping and collect the data

Yuxuan He:

(1)Use pandas' s function —— "read_csv" to read data:

train_df = pd.read_csv('C:/Users/apple/Desktop/dataAnalysis/van_ai_coding_challenge_4-master/van_ai_coding_challenge_4-master/data/train.csv')

(2)Observe the size of the data set：

train_df.shape

Li Ma:

quick overview of the data (data structure):

Data columns (total 12 columns):

| | |
|---|---|
| User_ID | 537577 non-null int64 |
| Product_ID | 537577 non-null object |
| Gender | 537577 non-null object |
| Age | 537577 non-null object |
| Occupation | 537577 non-null int64 |

City_Category 537577 non-null object

Stay_In_Current_City_Years 537577 non-null object

Marital_Status 537577 non-null int64

Product_Category_1 537577 non-null int64

Product_Category_2 370591 non-null float64

Product_Category_3 164278 non-null float64

Purchase 537577 non-null int64

dtypes: float64(2), int64(5), object(5)

User_ID unique element: 5891

Product_ID unique element: 3623

Gender unique element: 2

Age unique element: 7

Occupation unique element: 21

City_Category unique element: 3

Stay_In_Current_City_Years unique element: 5

Marital_Status unique element: 2

Product_Category_1 unique element: 18

Product_Category_2 unique element: 18

Product_Category_3 unique element: 16

Purchase unique element: 17959

This dataset has 537577 entries with 12 columns (potential features ). The unique User_ID and Product_ID are 5891 and 3623, respectively. They are small compared to the total number of entries, therefore, it can be inferred that a great portion of User_ID and Product_ID repeat many times and they may contain crucial information. However, these information are only available for the target variable Purchase. We want to predict Purchase based on the information Gender, Age, etc so that they can reflect those in their marketing strategies that target new customers. And according to the info, there are 2 columns where data is missing, Product_Category2 and Product_Category_3 is missing more, so they are not considered in this analysis.

The dataset explored through eda (exploratory data analysis) to. provide the necessary conclusions for subsequent processing and modeling. And my group partners Xinzhi Huo and Zhelin Liang did the data analysis section and showed the relationship between data.

## 3. Data cleaning and transformation

Yuxuan He:

(1) To understand the specific data form of a dataset, first check the first 10 rows of the dataset：train_df.head(10)

(2) convert the discrete columns into dummy variables for machine learning friendly：
dummy_cols = ['Product_ID', 'Gender', 'Age', 'City_Category', 'Stay_In_Current_City_Years']
train_df_with_dummies = pd.get_dummies(train_df, columns= dummy_cols)
(3)check the train_df with dummies:
train_df_with_dummies.head(10)
By now the dimensions still have a good ratio. I didn’ t meet dimensionality problem here because my observations are greater than 10 times the number of features.But I still have to be careful not to overfit.
(4)To create X and Y arrays for classification in training phase,I should drop NaN value in the dataframe by using the function dropna() in the Pandas:
train_df_with_dummies_noNaN = train_df_with_dummies.dropna()

```
Console 1/A
In [1]:

In [1]: import sys
   ...: sys.version
   ...: sys.version_info
   ...: # important dependencies
   ...: import os
   ...: import numpy as np
   ...: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: import math
   ...: from sklearn.linear_model import ElasticNet, ElasticNetCV, LinearRegression
   ...: from sklearn.ensemble import RandomForestClassifier
   ...: from sklearn.metrics import mean_squared_error
   ...: from sklearn.model_selection import train_test_split
   ...: from scipy import stats

In [2]: train_df = pd.read_csv('C:/Users/apple/Desktop/dataAnalysis/
van_ai_coding_challenge_4-master/van_ai_coding_challenge_4-master/data/train.csv')
   ...: train_df.shape
   ...: # convert the discrete columns into dummy variables
   ...:
   ...: dummy_cols = ['Product_ID', 'Gender', 'Age', 'City_Category',
'Stay_In_Current_City_Years']
   ...: train_df_with_dummies = pd.get_dummies(train_df, columns= dummy_cols)

In [3]: # drop NaN in the dataframe
   ...: train_df_with_dummies_noNaN = train_df_with_dummies.dropna()
   ...:
   ...: train_df_with_dummies_noNaN.head(3)
Out[3]:
    Unnamed: 0  ...  Stay_In_Current_City_Years_4+
0       161273  ...                              0
6       161279  ...                              0
17      161290  ...                              1

[3 rows x 3595 columns]
```

Zhelin Liang:

When we want to fill the null values of Product_Category_2 & Product_Category3, it comes to us that, Product2 may be the subset of Product1 and Product3 may be the subset of Product2. Just like we have three kinds of oranges, big; normal and small, we know that big and small are subsets of orange, but normal orange is the subset of itself, so we do not have to fill Product_Category2 or Product_Category3. Maybe the picture below will give us a clear understanding

| Catrgory1 | Catrgory2 | Catrgory3 |
|---|---|---|
| Orange | | |
| Orange | Big Orange | |
| Orange | Small Orange | |

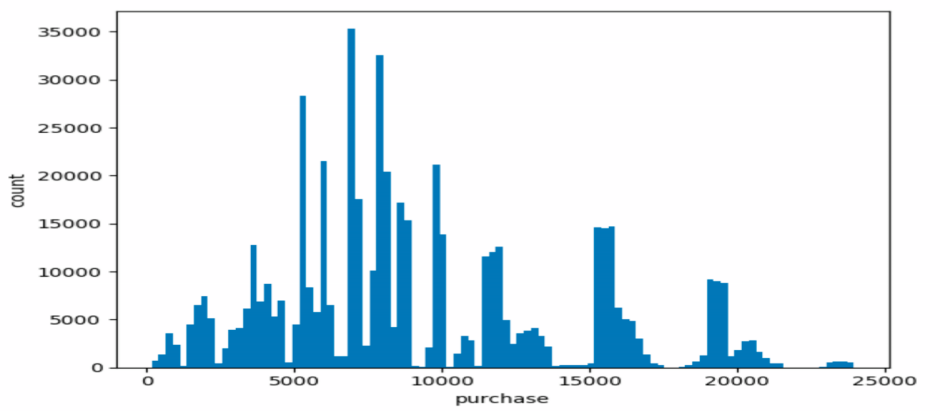
(We should know normal orange is orange)

So we decide to delete these two columns for the reason that they are of little use.(Or fill with 0)

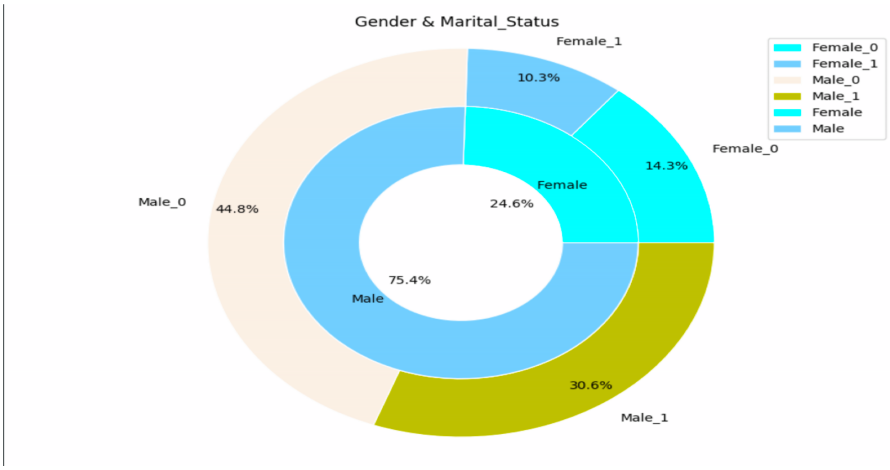
## 4. Analysis of the dataset

Zhelin Liang:

After data processing, we found that the user’ s consumption is generally between 0-15000

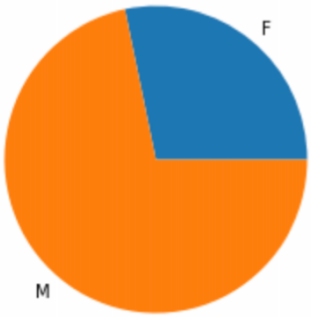


And we want to know gender and marriage impact on purchases:

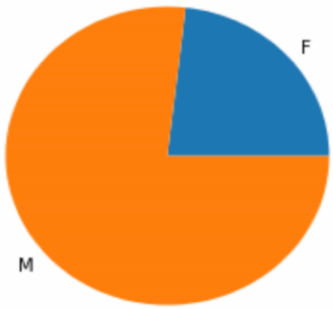



We know that either Male or Female, unmarried people buy more than married people, but we don't know whether men consume more than women. So we conducted a comparison of the number and consumption of men and women. Results are as follows

The ratio of female to male is 1:2.5



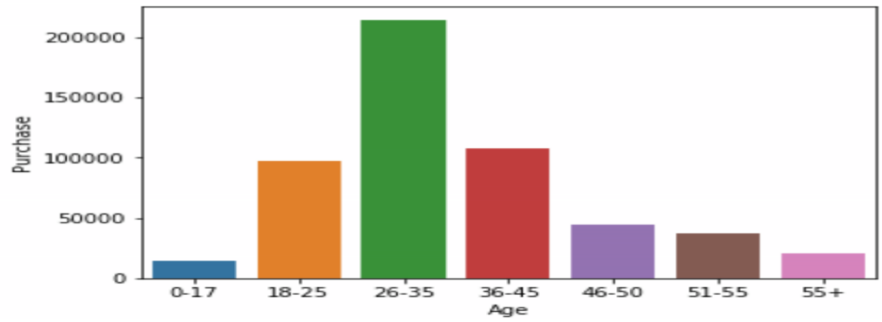

And the ratio of consumption is 1:3.3



So we can conclude that Men’ s purchasing power is stronger than women’ s

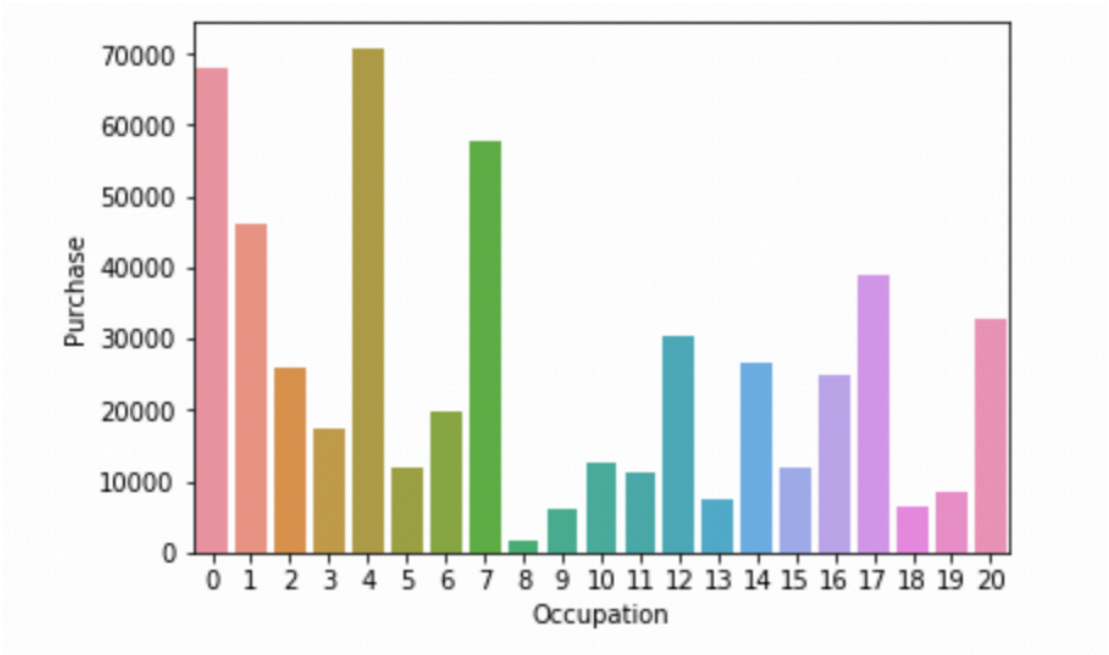
Then the Age:

When we compare the purchase of different ages, we find that the main group purchased is concentrated in the middle-aged group of 18-45 years old, showing a similar normal distribution pattern, among which the youth groups of 26-35 years old contribute the most.



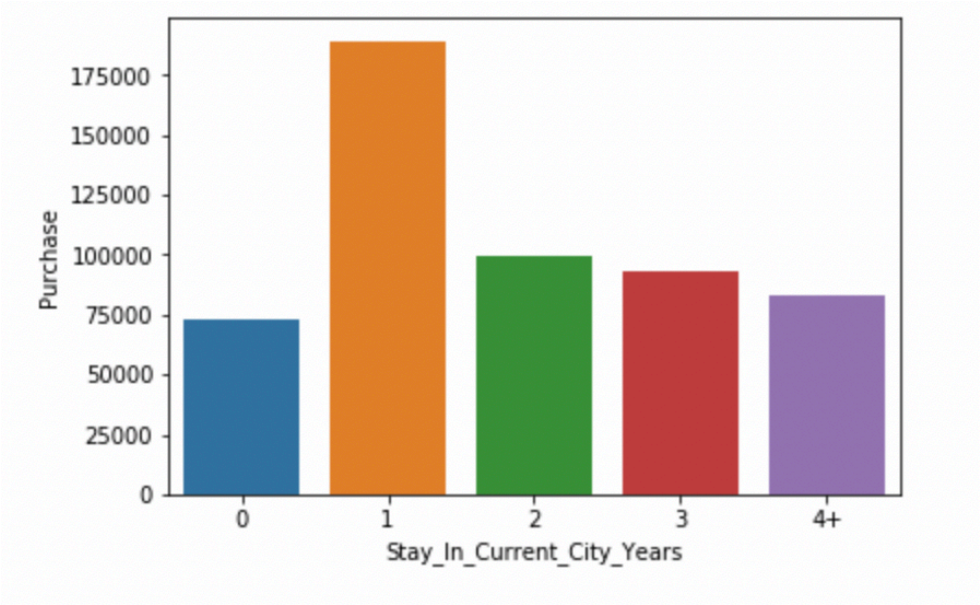

Occupation:

When we compare the purchase of different occupations, we find that the group with occupation code 0, 4, 7 has more purchase, and 8, 13 18 are the least



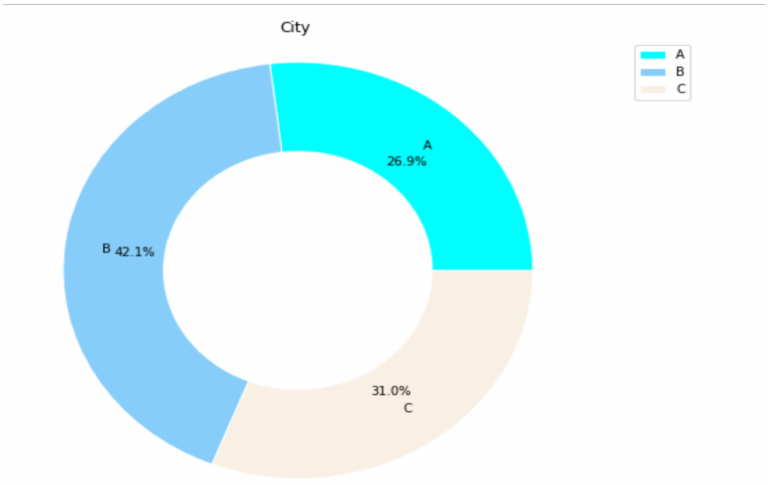

Stay_Years:

When we compare the purchases of different residence times, we find that the group that lives for one year buys more, and the rest of the purchases are closer.




City:

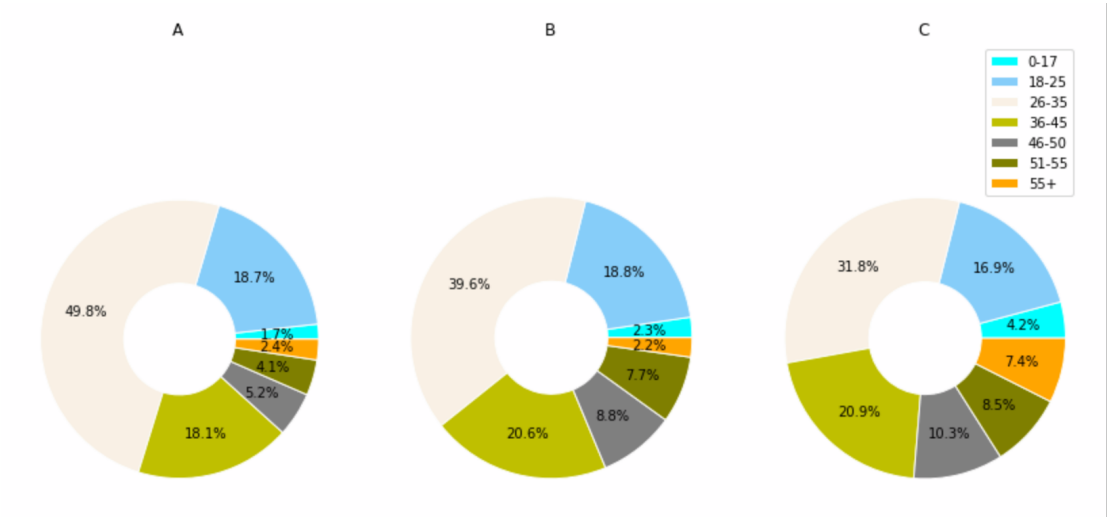
When we compare the purchase of different cities, we find that the number of groups purchased in B city is most, C City is the second, and A city is the least.



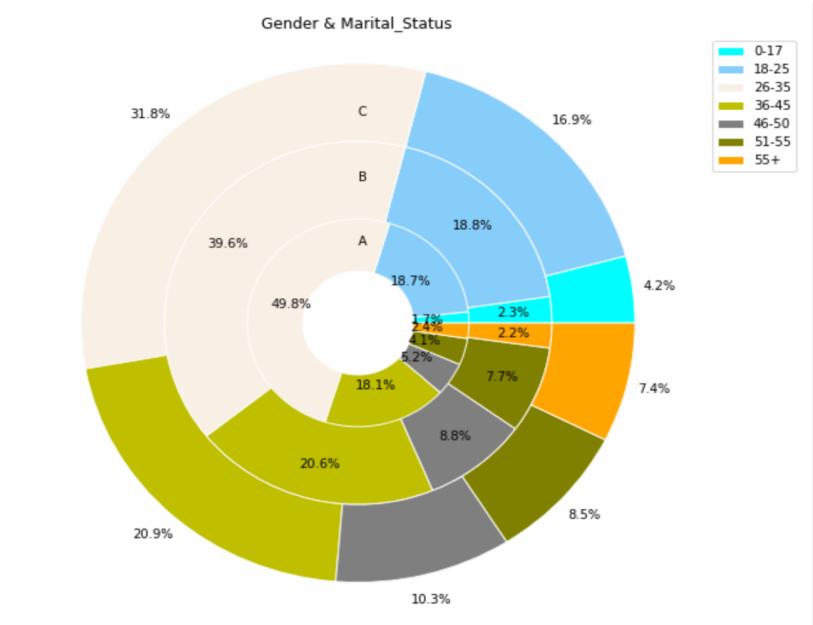

Population composition of each city:

When we compare the age groups of different cities, we find that Fifty percent of A's consumption is between 26 and 35 percent, 18-45 year old group is 86.6%, but A city' s purchasing power is the lowest, indicating that A city is a very young city, the economic strength is not very strong, but it has great development and consumption potential; B city is a relatively robust city with strong purchasing power and young people occupying the

majority of the population; In the C city, the population over 46 years old accounts for 26.2 Considering that its 0-17 age group accounts for 4.2% the highest among the three, the demand for baby products and juvenile books id higher, so the purchasing power ranks first




We can also show it by Ring diagram




Xinzhi Huo:

Analyze the dataset

I thorough learned our dataset and list some descriptive information which

can be obtained from the dataset.

We can find out:

TOP 10 items of sales.

Consumption of different age, occupation, gender, city, marital status, years of residence.

Analyze the data

The first step of analyzation is to calculate some descriptive information from the dataset. We set some goals to calculate.

Top 10 sales.

Top 10 product category

Top 10 buyers.

Top 10 product that man like most.

Top 10 product that woman like most.

Top 10 product that people of every age division like most.

Top 10 product that people of every different occupation like most.

Top 10 product that people in each different city like most.

Top 10 product that people in different living time like most.

Top 10 product that people who has been married like most.

Top 10 product that people who has not been married like most.

Top 10 product that costumer who are age of 26-35 like most.

I calculate these data and give them to the front end.

Find the correlation of the products

This is the most difficult part in my work. We want to find two products which are always bought together. To begin with I plan to write it on my own without any algorithms. I calculate the purchase of each product and the purchase of each pair of products. But later I find that it is difficult to decide the standard of the best partner. Because if one pair is decided to always be bought together, it should satisfy the condition that the two products are always bought and at the same time they are always bought together. It is hard to decide the standard of 'always'. Then I turn to python's function, but it is not suitable for our data. Finally, I find the Apriori algorithm. Apriori[1] is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The

frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

The pseudo code for the algorithm is given below for a transaction database T, and a support threshold of ∈. Usual set theoretic notation is employed, though note that T is a multiset. $C_k$ is the candidate set for level k. At each step, the algorithm is assumed to generate the candidate sets from the large item sets of the preceding level, heeding the downward closure lemma. count[c] accesses a field of the data structure that represents candidate set c, which is initially assumed to be zero. Many details are omitted below, usually the most important part of the implementation is the data structure used for storing the candidate sets, and counting their frequencies.

$$
\begin{aligned}
&\text{Apriori}(T, \epsilon) \\
&\quad L_1 \leftarrow \{\text{large } 1-\text{itemsets}\} \\
&\quad k \leftarrow 2 \\
&\quad \textbf{while } L_{k-1} \neq \emptyset \\
&\qquad C_k \leftarrow \{c = a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a, \{s \subseteq c \mid |s| = k-1\} \subseteq L_{k-1}\} \\
&\qquad \textbf{for } \text{transactions } t \in T \\
&\qquad\quad D_t \leftarrow \{c \in C_k \mid c \subseteq t\} \\
&\qquad\quad \textbf{for } \text{candidates } c \in D_t \\
&\qquad\qquad count[c] \leftarrow count[c] + 1 \\
&\qquad L_k \leftarrow \{c \in C_k \mid count[c] \geq \epsilon\} \\
&\qquad k \leftarrow k + 1 \\
&\quad \textbf{return } \bigcup_k L_k
\end{aligned}
$$

Here is the result of our dataset.

# III. Construction step

## 1. Machine Learning

Yuxuan He:

After data cleaning and chansfer, I would like to use Randon Forest Classifier to classify the customer’ s marital status from the black data set.The features is all the other columns including “Purchase”, and the label is “Martial Staus”.

(1) First to check the processed data set by listing 3 rows.

train_df_with_dummies_noNaN.head(3)

(2) Create X and y array for classification, it’ s the training phase, create Marital_Status label:

y_classf = train_df_with_dummies_noNaN['Marital_Status'].values

y_classf.shape

(3) Create features:

X_classf = train_df_with_dummies_noNaN.drop(['Marital_Status'], axis=1).values

X_classf.shape

(4) Split test set and training set for classification:

X_train_classf, X_test_classf, y_train_classf, y_test_classf = train_test_split(X_classf, y_classf, test_size=0.2, random_state=1)

(5) Define the random forest classification model:

RF_classifier = RandomForestClassifier(n_estimators=100, max_depth= 300, random_state=0)

(6) Use functions to train the model:

RF_classifier.fit(X_train_classf, y_train_classf)

(7) examine this classifier on the validation set:

RF_classifier.score(X= X_test_classf, y= y_test_classf)

```
    ....
    ...: train_df_with_dummies_noNaN.head(3)
Out[3]:
   Unnamed: 0  ...  Stay_In_Current_City_Years_4+
0      161273  ...                              0
6      161279  ...                              0
17     161290  ...                              1

[3 rows x 3595 columns]

In [4]: # create X and y array for classification (training phase)
    ...:
    ...: # create label
    ...:
    ...: y_classf = train_df_with_dummies_noNaN['Marital_Status'].values
    ...: y_classf.shape
Out[4]: (114928,)
```

```
In [5]: # create features
   ...:
   ...: X_classf = train_df_with_dummies_noNaN.drop(['Marital_Status'], axis=1).values
   ...: X_classf.shape
Out[5]: (114928, 3594)

In [6]: # create train-validation split for classification
   ...:
   ...: X_train_classf, X_test_classf, y_train_classf, y_test_classf =
train_test_split(X_classf, y_classf, test_size=0.2, random_state=1)

In [7]: RF_classifier = RandomForestClassifier(n_estimators=100, max_depth= 300,
random_state=0)

In [8]: RF_classifier.fit(X_train_classf, y_train_classf)
Out[8]:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=300, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)

In [9]: RF_classifier.score(X= X_test_classf, y= y_test_classf)
Out[9]: 0.812451057165231

In [10]:
```

All of these work is to predict the "Marital_Status" label, like the same way, I can use a similar method to predict the" Gender" label, just need to conver the Gender label' s values from string "F" /" M" to int 0/1.

```
Console 5/A
   ...: from flask import Flask

In [2]: train_df = pd.read_csv('C:/Users/apple/Desktop/dataAnalysis/
van_ai_coding_challenge_4-master/van_ai_coding_challenge_4-master/data/
train.csv')
   ...: train_df.shape
   ...: train_df.head(10)
   ...: # convert the discrete columns into dummy variables
   ...:
   ...: dummy_cols = ['Product_ID', 'Marital_Status', 'Age', 'City_Category',
'Stay_In_Current_City_Years']
   ...: train_df_with_dummies = pd.get_dummies(train_df, columns= dummy_cols)

In [3]: train_df_with_dummies.head(10)
Out[3]:
   Unnamed: 0  ...  Stay_In_Current_City_Years_4+
0      161273  ...                              0
1      161274  ...                              1
2      161275  ...                              1
3      161276  ...                              1
4      161277  ...                              1
5      161278  ...                              1
6      161279  ...                              0
7      161280  ...                              0
8      161281  ...                              0
9      161282  ...                              0

[10 rows x 3595 columns]

In [4]: train_df_with_dummies_noNaN = train_df_with_dummies.dropna()

In [5]: train_df_with_dummies_noNaN['Gender'] =
train_df_with_dummies_noNaN['Gender'].apply({'M':1, 'F':0}.get)
   ...: y_classf = train_df_with_dummies_noNaN['Gender'].values
__main__:1: 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: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy

In [6]: y_classf.shape
```

From the values of y_classf, we can see that the value is already converd.

```
In [9]: y_classf
Out[9]: array([1, 0, 1, ..., 1, 1, 1], dtype=int64)
```

And then use the same way to train the model, and test the model, we can get the score of the model is 0.81540937.

```
In [13]: X_train_classf, X_test_classf, y_train_classf, y_test_classf =
train_test_split(X_classf, y_classf, test_size=0.2, random_state=1)
    ...:
    ...: RF_classifier = RandomForestClassifier(n_estimators=100, max_depth= 300,
random_state=0)
    ...:
    ...: RF_classifier.fit(X_train_classf, y_train_classf)
Out[13]:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=300, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)

In [14]: RF_classifier.score(X= X_test_classf, y= y_test_classf)
Out[14]: 0.8154093796223788
```

What' s more, I would like to use DecisionTree classifier to predict Product Type from Product_category_1 of a customer based on the rest of available data like 'Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status'.

Convert raw data to decision friendly

Use the function apply(self, X[, check_input]): Returns the index of the leaf that each sample is predicted as.There are some tags need to be converted:Gender', 'Age', 'City_Category','Stay_In_Current_City_Years',

```
70
71
72 #predict
73 bf_df.head()
74
75 bf_df.columns
76
77 train_raw = bf_df[['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status']]
78 test = bf_df['Product_Category_1']
79
80 train_raw['Stay_In_Current_City_Years'] = train_raw['Stay_In_Current_City_Years'].apply(lambda x: '4' if x == '4+' else
81 train_raw['Stay_In_Current_City_Years'] = train_raw['Stay_In_Current_City_Years'].apply(lambda x: int(x))
82
83 train_raw['City_Category'] = train_raw['City_Category'].apply(lambda x: 0 if x == 'A' else (1 if x == 'B' else 2))
84
85 train_raw['Age'] = train_raw['Age'].apply(lambda x: '00-17' if x == '0-17' else x)
86 train_raw['Age'] = train_raw['Age'].apply(lambda x: '56-60' if x == '55+' else x)
87 train_raw['Age'] = train_raw['Age'].apply(lambda x: 0.5 * (int(x[0])*10 + int(x[1]) + int(x[-2])*10 + int(x[-1])))
88
89 train_raw['Gender'] = train_raw['Gender'].apply({'M':1, 'F':0}.get)
90
```

Split validation set:x_train, y_train, x_test, y_test = train_test_split(train, test, test_size=0.1, random_state=42)

Training model:

tree = DecisionTreeClassifier()

tree.fit(x_train, x_test)

y_pred = tree.predict(y_train)

Next let' s use socre(,) to test the model, we can see that the decision tree model get a low score of 0.3362, which means it doesn' t work well in this problem.

```
In [25]: train = train_raw.copy()

In [26]: x_train, y_train, x_test, y_test = train_test_split(train, test,
test_size=0.1, random_state=42)

In [27]: tree = DecisionTreeClassifier()

In [28]: tree.fit(x_train, x_test)
Out[28]:
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

In [29]: y_pred = tree.predict(y_train)

In [30]: accuracy_score(y_test, y_pred)
    ...:
Out[30]: 0.3362121655018469

In [31]:
```

To learn more from the black Friday data set, we need to visualization of the data set.

So I put forward two question:

(1)Who is more likely to spend more in a black Friday sale?

Men or Women.

Married or Un Married

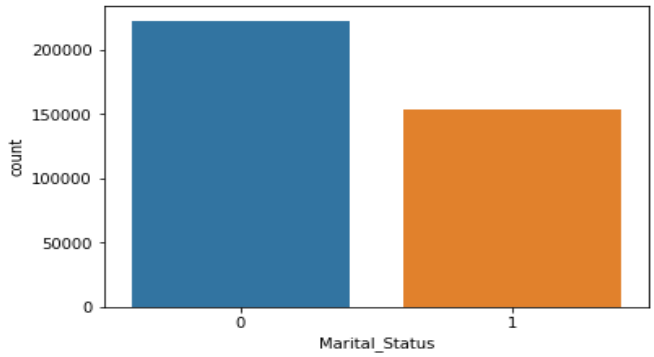
Old Residents or new residents

Which type of products are more likely to be sold in a black Friday sale?

In the first question, we want to know who is more likely to spend more.

sns.countplot(bf_df['Marital_Status'])

When people get married, their spending power declined in black Friday.



plt.subplot(1,2,1)

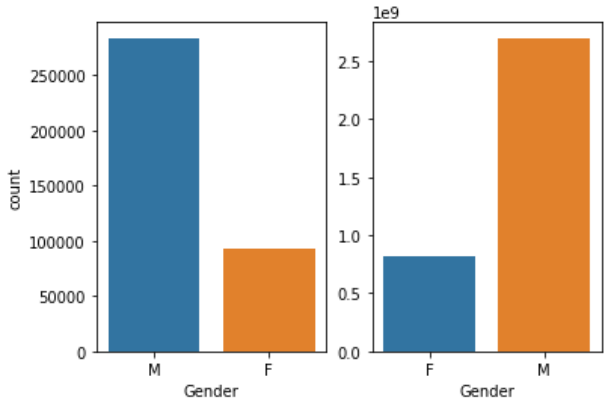
sns.countplot(bf_df['Gender']) #attendance

m_purchase = bf_df.groupby(['Gender'])['Purchase'].sum()

plt.subplot(1,2,2)

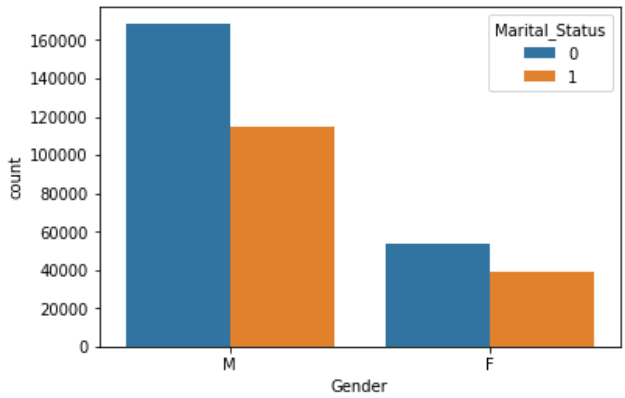
sns.barplot(m_purchase.index, m_purchase.values) #dollar value

Show the proportion of purchase amount, Men spend 2.5 times as much as women




sns.countplot(bf_df['Gender'], hue = bf_df['Marital_Status'])

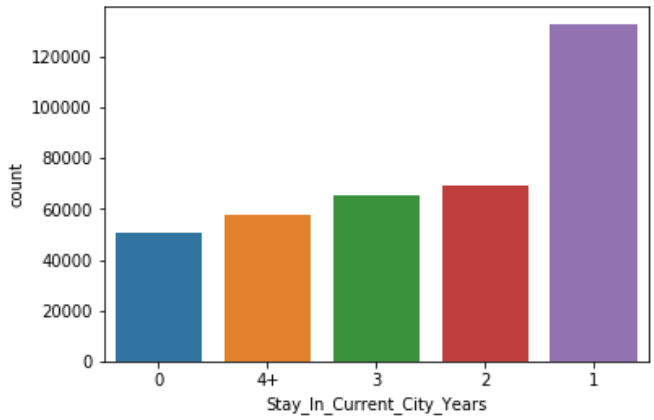
We could find that people tent to pay less money in black firday when they get married, and when people get married, man' s purchasing power are getting more and more similar with woman' s.




sns.countplot(bf_df['Stay_In_Current_City_Years'])

We can see that when people first come to a city, they won' t spend too much, but if they live in that city 1~2 years, they spend most, with the day pass by, they will spend less and less. I guess that in 0~1year, people need to dradually integrated into the city, so may be they will just look on the black city sales, or may be they won' t live long in the city, so they don' t need to by much. When people live in the city in 1~2 years, people need like to

improve life quality, so they tent eo spend more. As the time pass by,  People have almost everything they need to buy, so they would like to buy less.




Which type of products are more likely to be sold in a black Friday sale?

pie = bf_df.groupby(['Product_Category_1']).count()['User_ID']

pie_sorted = pie.sort_values(ascending = False)

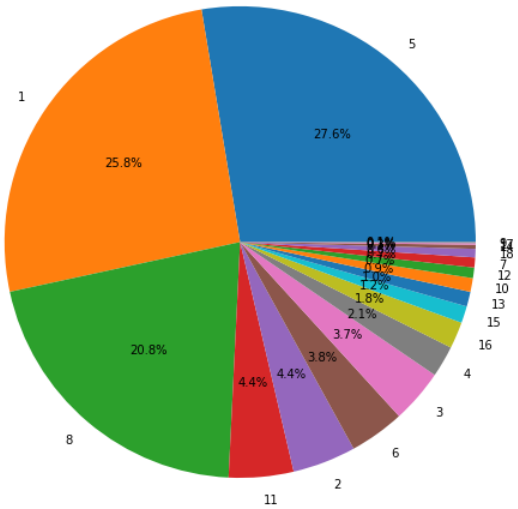
plt.figure(figsize = (10,8))

plt.pie(x = pie_sorted.values, labels = pie_sorted.index, autopct='%1.1f%%')

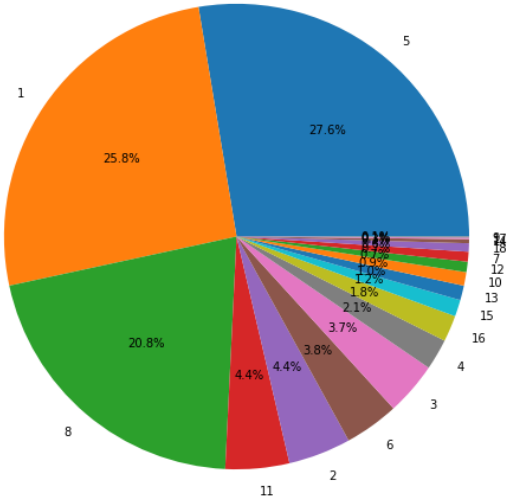
plt.axis('equal')

plt.show()

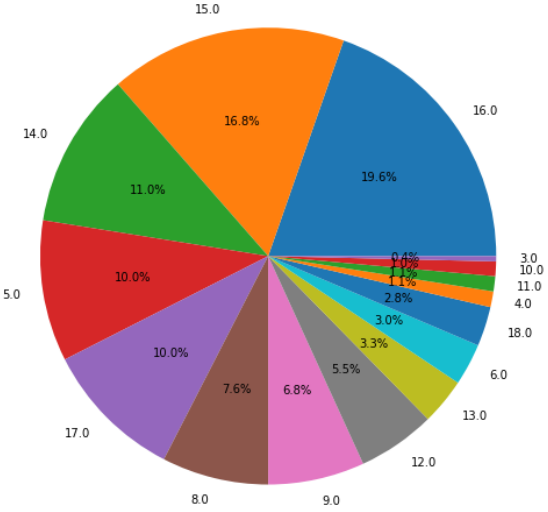
type 1、5、8  are the most higest sales in category1.




type 8、14、2  are the most higest sales in category2.



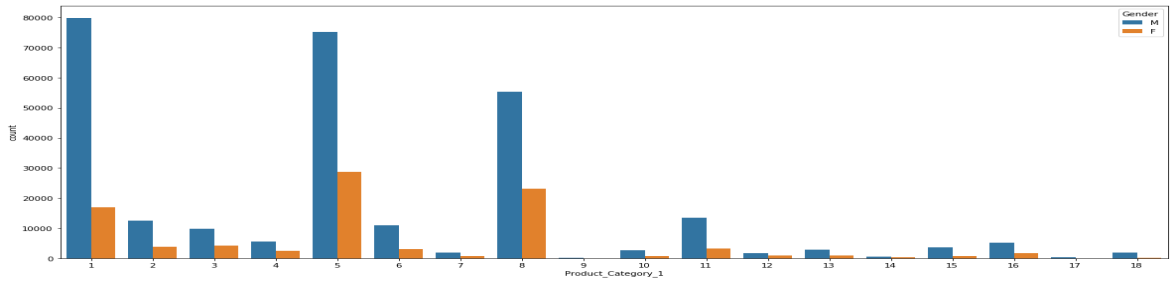

type 16、15、14 are the most higest sales in category3.




(2)Which type of products in category1 are common among men and which among women?

```
In [15]: plt.figure(figsize = (20,8))
    ...: sns.countplot(bf_df['Product_Category_1'], hue = bf_df['Gender'])
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1e9a22c5550>
```

in man, products 1、5、6 are the most popular. And in women, products 5、8、1 are the most common.



Li Ma:

The main goal of the project is to train a model that can predict purchase according to the customers' information. it is going be a straightforward regression problem where the model performance is evaluated by root mean squared error (RMSE)

Simple Features selection:

According to the step II, I found that User_ID and Product_ID are the most relevant to purchases and other attributes are not so highly. So I considered training two models:

Case1: Train with all features except purchases (In this way, we can predict the purchase amount of old customers in the store based on the established information.)

Case2: Train with all features except purchases,User_ID and Product_ID (This way we can predict the purchases of new customers in the store based on given information.)

Simple Features encodeing:

Convert gender strings to binary encoding so that It is better for numerical calculations

Case1: while the ordinal categorical features can be simply encoded with integers, the nominal categorical features need to be one-hot-encoded. In this case, however, the number of unique entires for User_ID and Product_ID is too big and one-hot-encoding these features will unnecessarily increase the data dimension and therefore cardinality. In real case, User_ID and Product_ID tended to be label-encoded. And other features tended to be one-hot-encoded

Case2: all of the features tended to be one-hot-encoded

then split our train and test data and will standardize the data using StandardScaler

Model selection:

Once the Feature was ready, I started training with some common models. The most commonly used models on Kaggle are basically tree-based models:

Gradient Boosting

Random Forest

Extra Randomized Trees

These models can basically be used by sklearn.

Excellent performance of Gradient Boosting plus Xgboost efficient implementation also make it a good choice for regression.

Through simple learning, considering performance and time, I decided to use two types of models to train: RandomForestRegressor and XgboostRegressor for Regression Prediction.

Before model tuning, I determined the best set of parameters through a process called Grid Search. In fact, this process is arguing that it is a violent search for all combinations based on a given parameter candidate.

```
param_grid = {'n_estimators': [10, 30, 100, 150, 300], 'max_features': [1,3,5,7,9,11]}
model = grid_search.GridSearchCV(estimator=rfr, param_grid=param_grid, n_jobs=1, cv=10, verbose=20, scoring=RMSE)
model.fit(X_train, y_train)
```

Model paremeters

According to the official document:

The parameters that RF needs to adjust include two parts. The first part is the parameters of the Bagging framework, and the second part is the parameters of the CART decision tree.

RF Bagging framework parameters:

1) n_estimators: This is the maximum number of iterations of the weak learner, or the maximum number of weak learners. In general, n_estimators are too small, easy to fit, n_estimators is too large, the amount of calculation will be too large, and after n_estimators reaches a certain number, the model increase obtained by increasing n_estimators will be small, so generally choose a moderate value. The default is 100.

2) oob_score: Whether to use the sample outside the bag to evaluate the quality of the model. The default is False. I tended to set to True because the out-of-bag score reflects the generalization ability of a model after fitting.

3) criterion: The evaluation criterion of the feature when the CART tree is divided. The CART regression tree corresponding to the regression RF defaults to the mean square error MSE, and another criterion that can be selected is the absolute value difference MAE In

general, choosing the default standard is already very good. And the document said that RMSE could be used too.

RF decision tree parameters:

1) The maximum number of features considered in RF partitioning max_features: There are many types of values that can be used. The default is "auto", which means that sqrtN features are considered in the division; if "log2" means division Consider up to log2N features; if it is "sqrt" or "auto" means that considering up to N features. If it is an integer, it represents the absolute number of features considered. If it is a floating point number, it means taking into account the feature percentage, that is, taking into account (percent xN) the number of features after rounding. Where N is the total number of features of the sample. Generally I used the default "auto". If the number of features is very large, we can flexibly use the other values just described to control the maximum number of features considered in the partitioning to control the generation time of the decision tree. (By the way, Random Forest generally gets the best results near the square root of max_features set to the number of features)

2) The maximum depth of the decision tree max_depth: The default is not to enter. If not entered, the decision tree does not limit the depth of the subtree when creating the subtree. In general, this value can be ignored when there is little data or features. If the model sample size is large and there are many features, it is recommended to limit this maximum depth. The specific value depends on the distribution of the data. Commonly used values can be between 10-100

3) The minimum number of samples required for internal node subdivision min_samples_split: This value limits the conditions under which the subtree continues to be partitioned. If the number of samples of a node is less than min_samples_split, it will not continue to try to select the optimal feature for partitioning. The default is 2. If the sample size is not large, I do not need to control this value. If the sample size is very large, it is recommended to increase this value.

4) The minimum number of samples of the leaf node min_samples_leaf: This value limits the minimum number of samples of the leaf node. If the number of leaf nodes is smaller than the number of samples, it will be pruned together with the sibling node. The default is 1, enter the minimum number of samples, or the minimum number of samples as a percentage of the total number of samples. If the sample size is not large, do not need to control this value. If the sample size is very large, it is recommended to increase this value.

5) The smallest sample weight of the leaf node and min_weight_fraction_leaf: This value limits the minimum value of the sum of all sample weights of the leaf nodes. If it is less than this value, it will be pruned together with the sibling node. The default is 0, which means that the weight problem is not considered. In general, if I have more samples with missing values, or if the classification of the classification tree sample is very different, the sample weight will be introduced. At this time, must pay attention to this value.

6) Maximum number of leaf nodes max_leaf_nodes: By limiting the maximum number of leaf nodes, over-fitting can be prevented. The default is "None", that is, the maximum number of leaf nodes is not limited. If a limit is imposed, the algorithm will establish an optimal decision tree within the maximum number of leaf nodes. If there are not many features, ignore this value, but if the feature is divided into many, limit it. The specific value can be obtained by cross-validation.

The most important of the above decision tree parameters include the maximum number of features max_features, the maximum depth max_depth, the minimum number of samples required for internal node subdivision min_samples_split and the minimum number of samples of leaf nodes min_samples_leaf

All the XGBoost parameters divided into three categories:

General parameters: macro function control.

Booster parameter: Controls the booster (tree/regression) for each step.

Learning target parameters: Control the performance of training objectives.

General parameters

These parameters are used to control the macro functions of XGBoost:

1) booster [default gbtree] There are two options for choosing a model for each iteration:

Gbtree: tree-based model

Gbliner: linear model

2) Silent[default 0] When this parameter value is 1, the silent mode is on and no information is output.Normally this parameter will remain at the default of 0, because this will help us better understand the model.

3) Nthread [default is the maximum number of possible threads] This parameter is used for multi-thread control and should be entered into the system's core count.

Booster parameter

From xgboost-unity, the bst: prefix is no longer needed for booster parameters. Parameter with or without bst: prefix will be equivalent(both bst:eta and eta will be valid parameter setting)

1) eta [default 0.3] By reducing the weight of each step, the robustness of the model can be improved.Typical values are 0.01-0.2.

2) min_child_weight [default 1] Determine the minimum leaf node sample weight sum.This parameter of XGBoost is the sum of the minimum sample weights.This parameter is used to avoid overfitting. When its value is large, the model can be avoided to learn local special samples.But if this value is too high, it will lead to under-fitting. This parameter needs to be adjusted using CV.

3) max_depth [default 6] this value is the maximum depth of the tree.This value is also used to avoid overfitting. The larger the max_depth, the more specific and more local samples will be learned.Typical value: 3-10

4) max_leaf_nodes.The maximum number of nodes or leaves on the tree.Can replace the role of max_depth. Because if a binary tree is generated, a tree with a depth of n generates n^2 at most Leaves.

5)gamma [default 0] When the node is split, the node will be split only if the value of the post-split loss function drops. Gamma specifies the minimum loss function drop value required for node splitting.The larger the value of this parameter, the more conservative the algorithm. The value of this parameter is closely related to the loss function, so it needs to be adjusted.

6)max_delta_step [default 0] This parameter limits the maximum step size for each tree weight change. If the value of this parameter is 0, it means there is no constraint. If it is given a positive value, it will make the algorithm more conservative.Usually, this parameter does not need to be set. But when the samples in each category are very unbalanced, it is very helpful for logistic regression.

7)subsample [default 1] It is exactly the same as the subsample parameter in GBM. This parameter controls the proportion of random samples for each tree.By reducing the value of this parameter, the algorithm is more conservative and avoids overfitting. However, if this value is set too small, it may cause an under-fitting.Typical value: 0.5-1.

8)colsample_bytree [default 1] Used to control the proportion of the number of columns per random sample (each column is a feature).Typical value: 0.5-1.

9)colsample_bylevel [default 1] Used to control each split of each level of the tree, the proportion of samples of the number of columns. I personally don't use this parameter very much, because the subsample parameter and the colsample_bytree parameter can play the same role. But if you are interested, you can dig more useful for this parameter.

10)lambda [default 1] The L2 regularization term of the weight. (similar to Ridge regression).This parameter is used to control the regularization part of XGBoost. Although most data scientists rarely use this parameter, this parameter can still be used for more purposes in reducing overfitting.

11) alpha [default 1] The L1 regularization term of the weight. (similar to Lasso regression).Can be applied in the case of very high dimensions, making the algorithm faster.

12) scale_pos_weight [default 1] When the samples in each category are very unbalanced, setting this parameter to a positive value will make the algorithm converge faster.

Learning target parameter

These parameter are used to control the ideal optimization goal and the measurement method of each step result.

1) Objective [default=reg:linear] Define learning tasks and corresponding learning objectives. The optional objective functions for our regression prediction is :"reg:linear" – linear regression.

2) Eval_metric [default according to objective] Rmse for regression .

3) Seed [default 0] Seed of random number.Set it to reproduce the results of random data, and can also be used to adjust parameters.

The tuning of Xgboost. The parameters that are generally considered to have a large impact on its performance are:

Eta: The step size when the weight is updated after each iteration. The smaller the training, the slower the training.

Num_round: The number of iterations in total.

Subsample: The proportion of data used to train each tree when training. Used to prevent Overfitting.

Colsample_bytree: The proportion of features used to train each tree.

Max_depth: The maximum depth limit per tree. Unlike Random Forest, Gradient Boosting will eventually overfit without limiting the depth.

Early_stopping_rounds: Used to control how many iterations of the Out Of Sample's validation set have not been improved and the training is terminated early. Used to prevent Overfitting.

The general tuning steps are:

1.A portion of the training data is drawn out as a validation set.

2.First set eta higher (such as 0.1) and num_round to 300 ~ 500.

3.Search for other parameters with Grid Search

4.Gradually reduce eta and find the best value.

5.With the validation set as watchlist, retrain on the training set with the best combination of parameters found. Observe the output of the algorithm and see how the scores on the validation set change after each iteration to get the best early_stopping_rounds.

Training process:

RandomForestRegressor:

```
params = {
    'n_estimators': [10, 30, 100, 150, 300],
    'max_depth': [3, 5, 7, 9 ]
}
gridsearch=GridSearchCV(RandomForestRegressor(),param_grid=params, cv=3,scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)
preds = grid_search.predict(X_test)
print("最佳参数: {}".format(grid_search.best_params_))
print("RMSE 分数: {}".format(mean_squared_error(y_test, preds) ** 0.5))
```

XGBRegressor:

```
cv_params = {'n_estimators': [400, 500, 600, 700, 800]}
```

```
        other_params = {'learning_rate': 0.1, 'n_estimators': 500, 'max_depth': 5,
'min_child_weight': 1, 'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0,
'reg_alpha': 0, 'reg_lambda': 1}
        cv_params = {'n_estimators': [550, 575, 600, 650, 675]}
        other_params = {'learning_rate': 0.1, 'n_estimators': 600, 'max_depth': 5,
'min_child_weight': 1, 'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0,
'reg_alpha': 0, 'reg_lambda': 1}
        cv_params = {'max_depth': [3, 4, 5, 6, 7, 8, 9, 10], 'min_child_weight': [1, 2, 3, 4, 5, 6]}
        other_params = {'learning_rate': 0.1, 'n_estimators': 550, 'max_depth': 5,
'min_child_weight': 1, 'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0,
'reg_alpha': 0, 'reg_lambda': 1}
        cv_params = {'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]}
        other_params = {'learning_rate': 0.1, 'n_estimators': 550, 'max_depth': 4,
'min_child_weight': 5, 'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0,
'reg_alpha': 0, 'reg_lambda': 1}
        cv_params = {'subsample': [0.6, 0.7, 0.8, 0.9], 'colsample_bytree': [0.6, 0.7, 0.8, 0.9]}
        other_params = {'learning_rate': 0.1, 'n_estimators': 550, 'max_depth': 4,
'min_child_weight': 5, 'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0.1,
'reg_alpha': 0, 'reg_lambda': 1}
        cv_params = {'reg_alpha': [0.05, 0.1, 1, 2, 3], 'reg_lambda': [0.05, 0.1, 1, 2, 3]}
        other_params = {'learning_rate': 0.1, 'n_estimators': 550, 'max_depth': 4,
'min_child_weight': 5, 'seed': 0,'subsample': 0.7, 'colsample_bytree': 0.7, 'gamma': 0.1,
'reg_alpha': 0, 'reg_lambda': 1}
        cv_params = {'learning_rate': [0.01, 0.05, 0.07, 0.1, 0.2]}
        other_params = {'learning_rate': 0.1, 'n_estimators': 550, 'max_depth': 4,
'min_child_weight': 5, 'seed': 0,'subsample': 0.7, 'colsample_bytree': 0.7, 'gamma': 0.1,
'reg_alpha': 1, 'reg_lambda': 1}
        model = xgb.XGBRegressor(**other_params)
        optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='
neg_mean_squared_error ', cv=5, verbose=1, n_jobs=4)
        optimized_GBM.fit(X_train, y_train)
        evalute_result = optimized_GBM.grid_scores_
        optimized_GBM.fit(X_train, y_train)
```

```
preds = optimized_GBM.predict(X_test)
print("RMSE  分数: {}".format(mean_squared_error(y_test, preds) ** 0.5))
print('每轮迭代运行结果:{0}'.format(evalute_result))
print('最佳参数：{0}'.format(optimized_GBM.best_params_))
print('最佳模型得分:{0}'.format(optimized_GBM.best_score_))
```

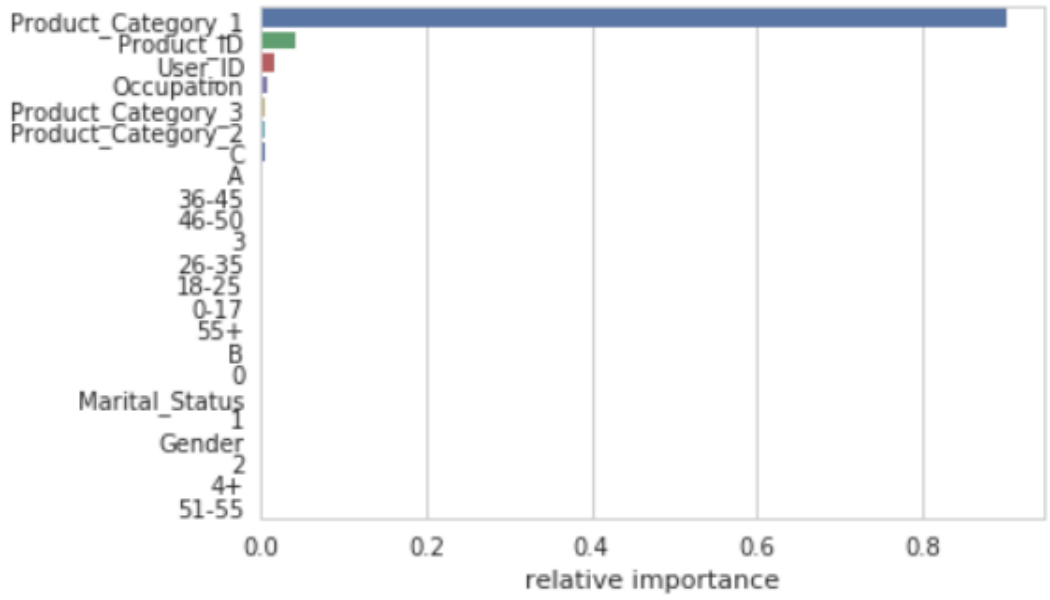
Training result:

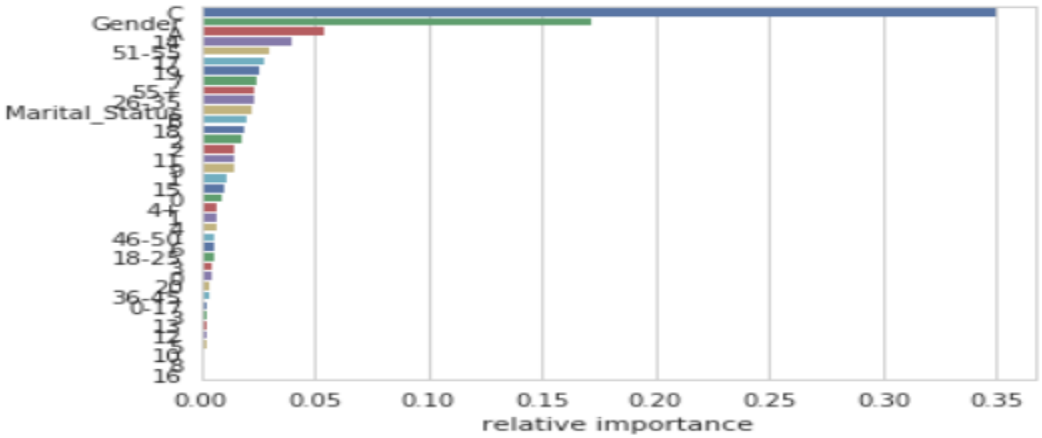
RFregressor:  RMSE for case1 : 2930;  for case2 : 4422

Xgbregressor:  RMSE for case1: 3622;  for case2: 6917

From the results of the assessment: Although xgb has better speed performance, it does not perform satisfactorily on this data set. So I choose the RF model

and the feature importance for case 1 and case 2 is



This told that the current model heavily relies on Product_Category_1 (followed by Product_ID and User_ID) to make prediction on Purchase. This was somewhat expected from EDA as they are directly related to the amount of Purchase that the customers make.and considering only the low-relevant features, now it seemed like the model suffers from high bias issue. Knowing that this dataset is artificial, it could possibly be that the given low-level features do not simply have enough predictive power for Purchase.

As a conclusion, for the purpose of predicting customer's Purchase, the top three features useful included Product_Category_1, Product_ID and User_ID. However, if the goal is to predict Purchase for the new customer who have never been in the store, those customer-specific and product-specific information cannot be utilized. In this case, only the low-relevant features can be utilized to train the model.

Model improvement

different encoding: one-hot encode only for age, city ,and citystay

Delete feature quantities with low correlation like marital_status, gender

The above steps in order to reduce the number of features

Change parameters referenced network tutorial and official document

However, these modifications did not improve performance. Rmse have not broken 2930

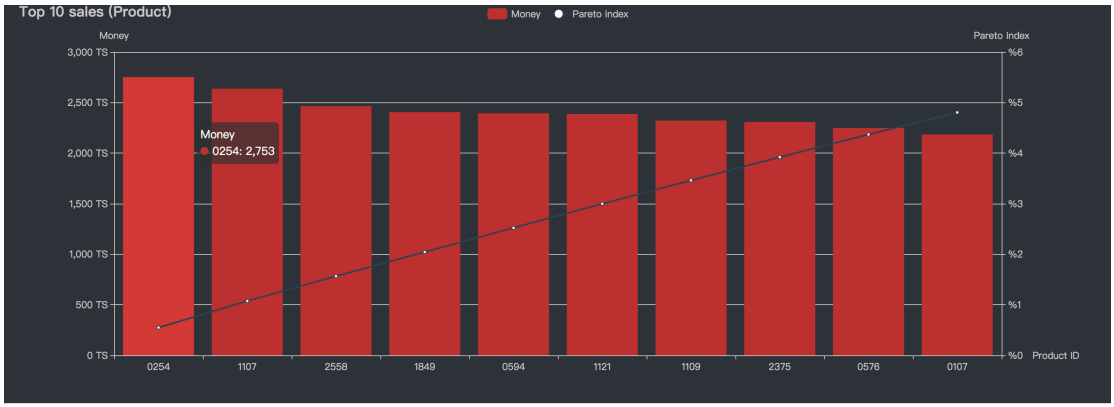
## 2. Visualization of the dataset

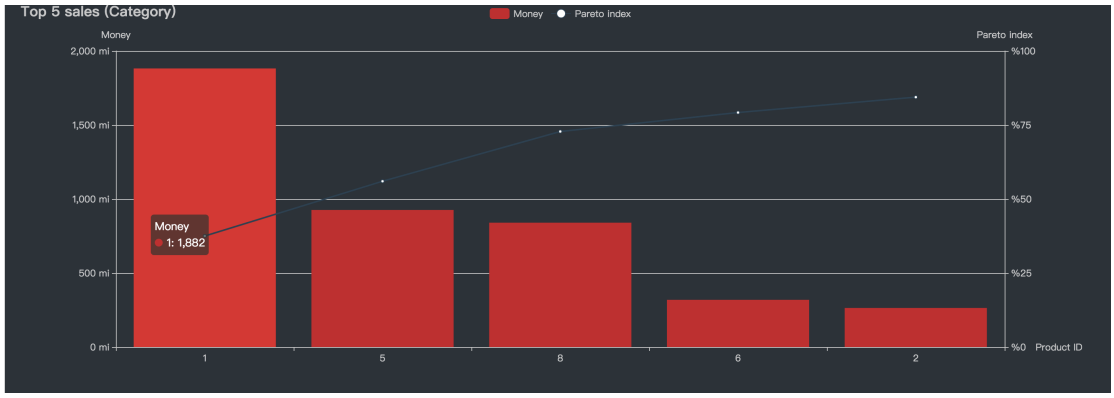
### Zhelin Liang:

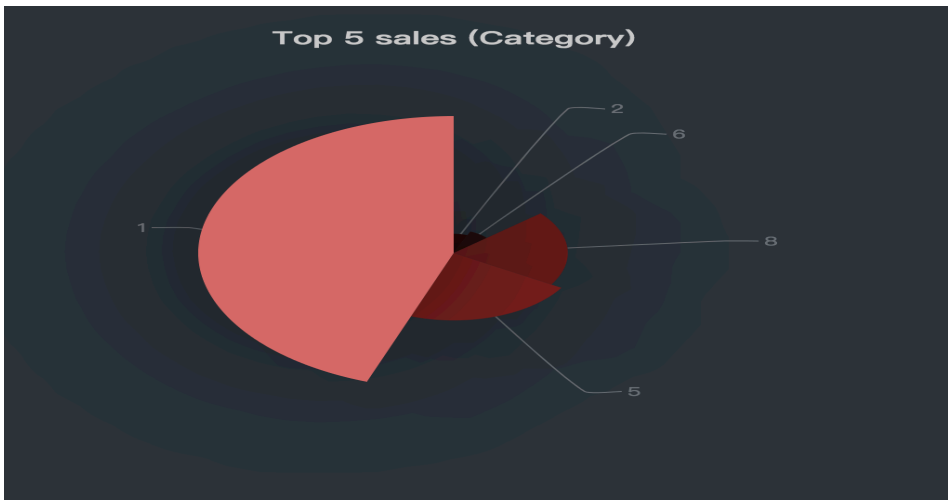
We show the sales of products and the purchase comparison and preference of different ages, genders, cities, residence time, occupation and marital status respectively. At the same time, we also show some combined data. The details are as follows.

Product part:

We show top 10 sales(Product), top 5 sales(Category) and the pareto index of each other.



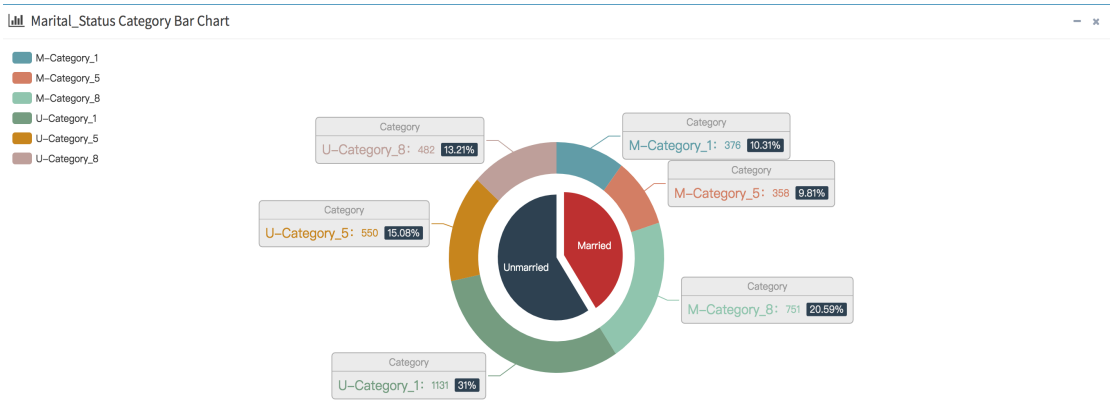




Gender, Age, City, Stay, Occupation and Marital part:

In these part, We showed how much they bought and what their preferences were。

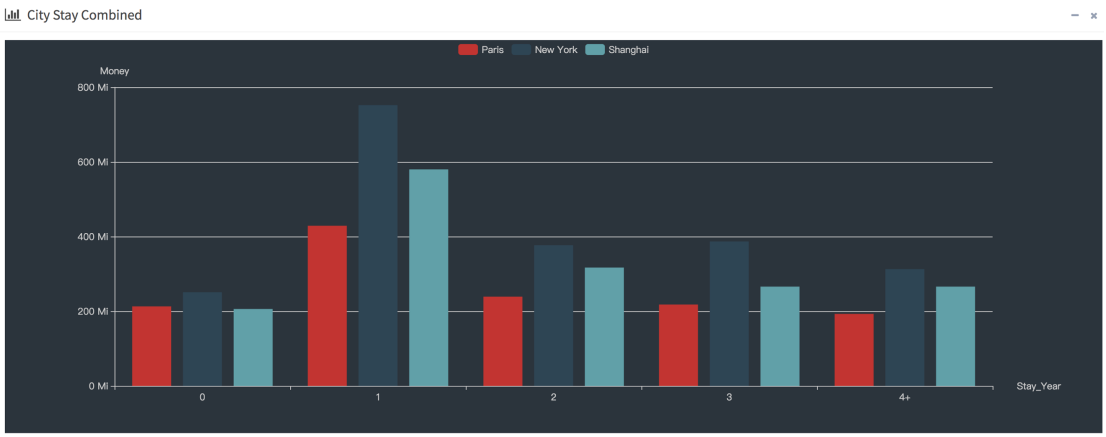
(There are so many pictures that we'll just show a few examples)





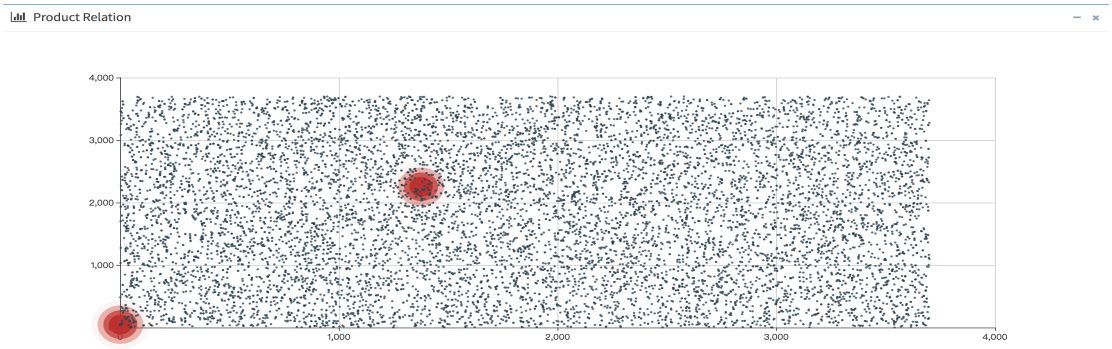

Combined Data Part:

In these part, we show different combinations of gender and marital status and different combinations of cities and residence periods



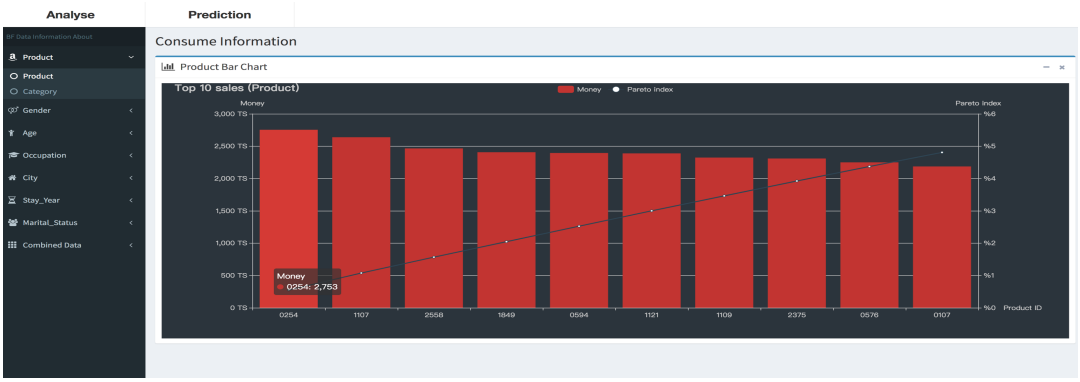

And we looked at all 3,700 items and stored their purchase records to see which two were purchased the most.

We use scatter plots to construct all purchase records, and finally found that goods 1 and 48, 1376 and 2254 times of goods were the most purchased.

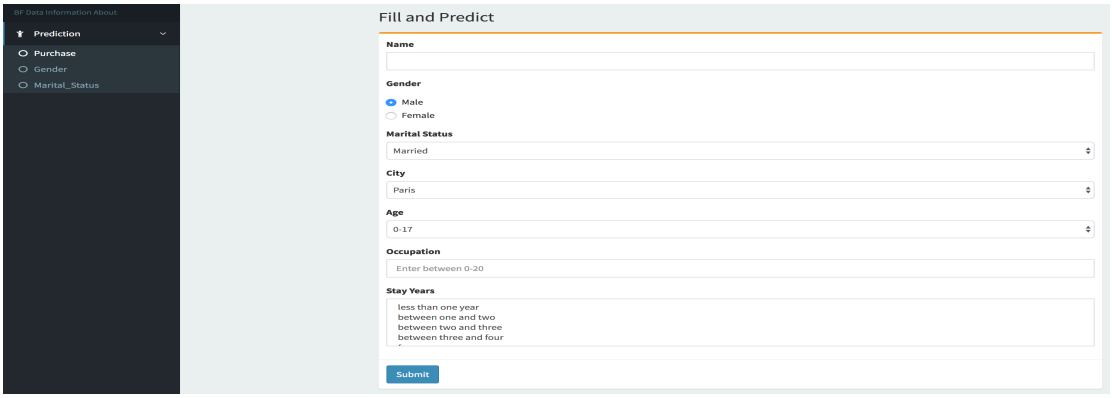



## 3. Program the application and make the main tests

Zhelin Liang:



Here is our application, it has Analyze part and Prediction Part, the Analyze Part(just as I showed a moment ago) Analyze， combine and present the data. The prediction part predict outcomes based on given data, such as purchasing power, gender, marital status, etc.




# IV. Deployment and Reporting

## 1. Deploy the project if possible in the defined environment

We deploy our project locally, the server is the local server. In the data analysis part, we filter out the json data through python and display it statically. In the data prediction part, we input the data we want to predict and send the json data to the python backend. Python uses the model training to return the result to the json data. Front end and show it out.