A. Linear Regression Analysis for Wine Quality

(a). Regression analysis:

variable	estimate	std. error	t-value	p-value
f0	-0.00555	0.02481	-0.22356	0.82318
f1	0.04398	0.02799	1.5709	0.11674
f2	0.31396	0.03477	9.0285	2.43E-18
f3	0.01859	0.04154	0.44744	0.65472
f4	-0.00348	0.0382	-0.091	0.92752
f5	-0.07399	0.0298	-2.48278	0.01331
f6	-0.07105	0.02591	-2.74177	0.0063
f7	0.02352	0.02756	0.8533	0.39384
f8	0.04101	0.0189	2.1705	0.03037
f10	-0.04459	0.02104	-2.11903	0.0345
f11	-0.02915	0.02027	-1.43816	0.15092
f12	-0.00059	0.022	-0.02694	0.97852
f13	0.03363	0.02382	1.41163	0.15859
f14	-0.18322	0.02059	-8.89792	6.90E-18
f15	-0.10609	0.01906	-5.56523	3.97E-08
f16	-0.03577	0.02037	-1.75608	0.07959
f17	0.06331	0.01857	3.40883	0.0007
f18	-0.19038	0.02071	-9.1944	6.36E-19
f19	0.02775	0.02642	1.05071	0.29382
f20	0.01256	0.01952	0.64361	0.52008
f21	-0.03567	0.02825	-1.26293	0.20711
f22	0.07468	0.02114	3.53299	0.00044
f23	-0.00877	0.01984	-0.442	0.65865
f24	0.0193	0.02412	0.8004	0.4238
f25	-0.06792	0.01994	-3.40589	0.0007
f26	-0.03601	0.02215	-1.62523	0.10465
f27	-0.0062	0.01917	-0.32352	0.74642

- R-squared: 0.495
- Adjusted R-squared: 0.472

(b). The fitting of the linear regression is a good idea?

i. No, R-squared 與調整後的 R-squared 皆小於 0.5，代表此模型對於資料的解釋能力不盡理想。

ii. Possible reason of poor fitting: 此資料中有許多變數僅有兩、三種值。導致它雖然是數值資料，但就分佈來看比較像 categorical，

可能不適用於此線性迴歸分析。

Specific value	
只有一種	f9 -> 刪除
只有兩種	f3, f6, f7, f10, f11, f13, f14, f15, f16, f17, f18, f20, f21
三種	f4, f8, f22, f25, f26, f27
三種以上	f0, f1, f2, f5, f19, f22, f23, f24

iii. Based on the results, rank the independent variables by p-values and which one are statistically significant variables with p-values<0.01?

```
pv = result.pvalues
pv_df = pd.DataFrame(list(zip(pv, pv<0.01)), columns=["p_values", "<0.01"], index=[pv.index])
pv_df.sort_values(by = 'p_values')
```

	p_values	<0.01
const	0.000000e+00	True
f18	6.355895e-19	True
f2	2.430368e-18	True
f14	6.896905e-18	True
f15	3.971326e-08	True
f22	4.429125e-04	True
f17	6.967770e-04	True
f25	7.041639e-04	True
f6	6.295708e-03	True

(c). Testify the underlying assumptions of regression (1) Normality, (2) Independence, and (3) Homogeneity of Variance with respect to residual.

```
result.resid  # 從回歸模型找殘差值
```

```
0      0.504562
1      1.160926
2      0.322394
3      0.177112
4      0.194610
         ...
615   -0.586052
616    0.013865
617    0.169389
618   -0.781153
619   -0.138202
Length: 620, dtype: float64
```

(1). 常態性假設

```
# 常態性假設
shapiro_test = stats.shapiro(result.resid)
shapiro_test
```

```
ShapiroResult(statistic=0.9345124363899231, pvalue=7.536043131996446e-16)
```

在常態性假設中，虛無假設 H0 為「殘差服從常態分配」。因 p-value < 0.05 足以拒絕 H0，故殘差不符合常態分配。

(2). 獨立性假設

```
# 獨立性假設
from statsmodels.stats.stattools import durbin_watson
durbin_watson(result.resid)
```

```
2.0040534804722103
```

Durbin Watson 檢定值為 2 代表不相關，0 代表最大正相關值，4 代表最大負相關值。這裡檢定值為 2，故殘差具有獨立性。

(3). 變異數同質性假設

```
# 變異數同質性假設
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
name = ['Lagrange multiplier statistic', 'p-value',
        'f-value', 'f p-value']
test = sms.het_breuschpagan(result.resid, result.model.exog)
lzip(name, test)
```

```
[('Lagrange multiplier statistic', 140.62034965182607),
 ('p-value', 2.5277189408433144e-17),
 ('f-value', 6.431710999631267),
 ('f p-value', 5.311815675353658e-20)]
```

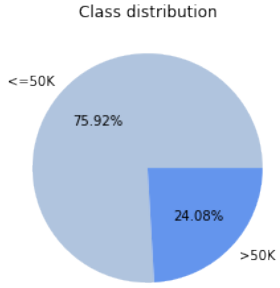
Breusch–Pagan 檢定值中 p value < 0.05，拒絕同質性假設，故殘差不具有同質性。推測我們的模型可能有問題。

B. Data Preprocessing and Generalized Linear Model (GLM)/Logistic Regression

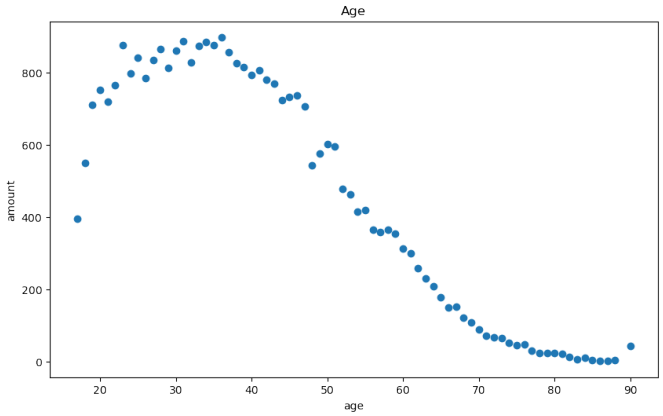
(1). Descriptive statistics

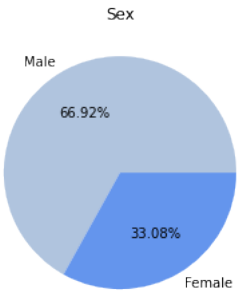
i. Data distribution

(1). 在 class 的分布上，**收入少於五萬**的人佔了 **3/4**

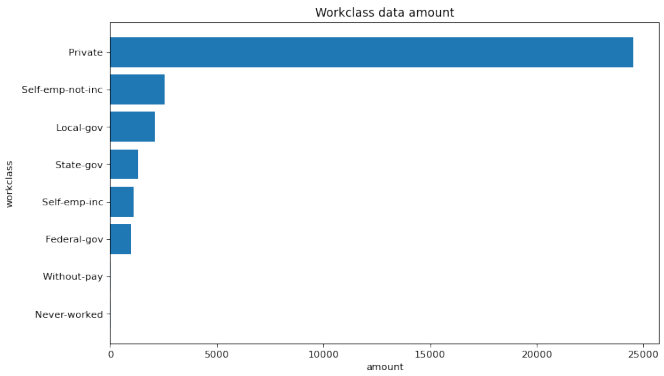


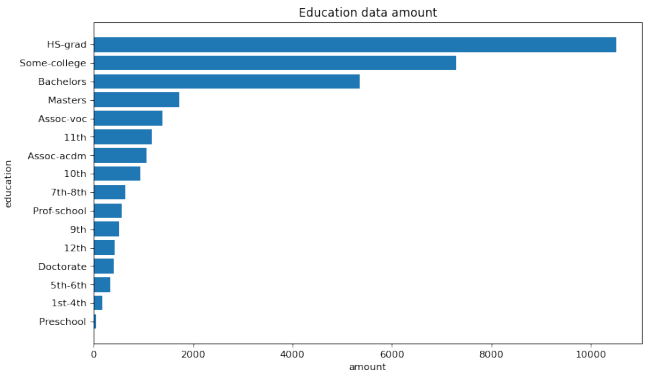
(2). 年齡分佈主要集中在 20-50 歲的青壯年，性別則是男性居多。

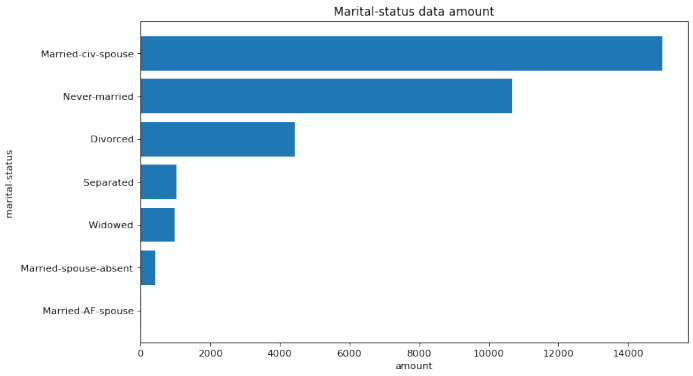


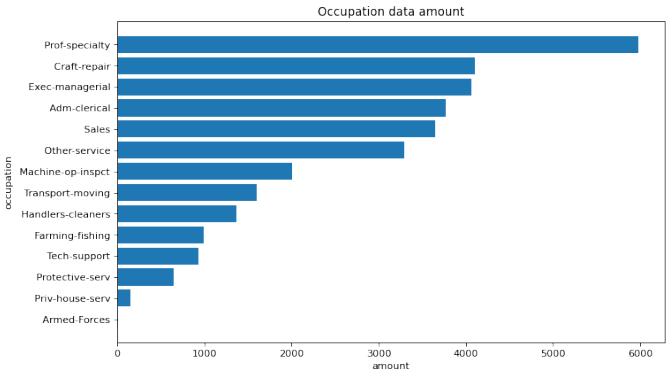


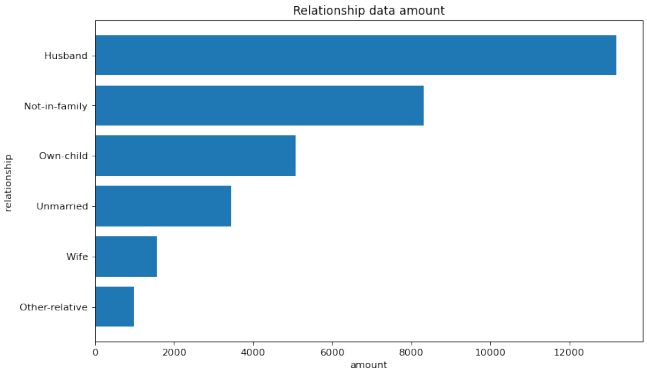
(3). 更多欄位的資料數量分佈：

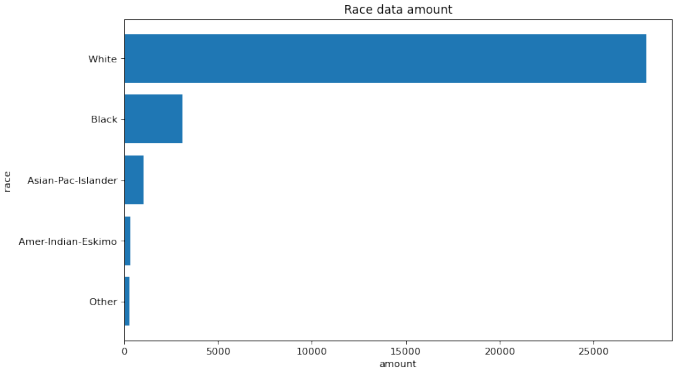


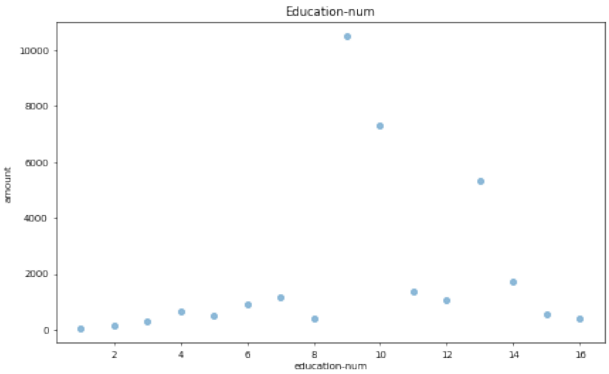


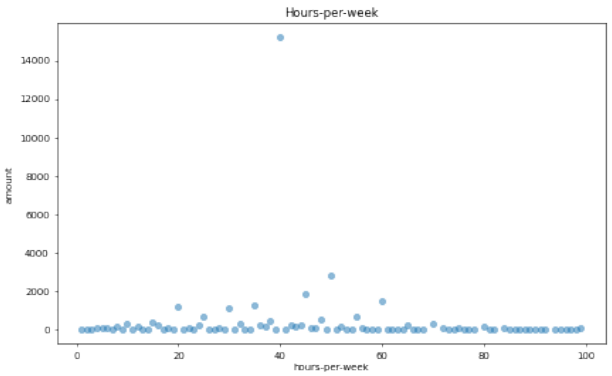

Marital-status data amount
Married-civ-spouse
Never-married
Divorced
Separated
Widowed
Married-spouse-absent
Married-AF-spouse
marital-status
0
2000
4000
6000
8000
10000
12000
14000
amount


Occupation data amount
Prof-specialty
Craft-repair
Exec-managerial
Adm-clerical
Sales
Other-service
Machine-op-inspct
Transport-moving
Handlers-cleaners
Farming-fishing
Tech-support
Protective-serv
Priv-house-serv
Armed-Forces
occupation
0
1000
2000
3000
4000
5000
6000
amount


Relationship data amount
Husband
Not-in-family
Own-child
Unmarried
Wife
Other-relative
relationship
0
2000
4000
6000
8000
10000
12000
amount


Race data amount
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
race
0
5000
10000
15000
20000
25000
amount


Education-num
amount
0
2000
4000
6000
8000
10000
2
4
6
8
10
12
14
16
education-num


Hours-per-week
amount
0
2000
4000
6000
8000
10000
12000
14000
0
20
40
60
80
100
hours-per-week

ii. Mean, standard deviation, variation, min max value, and quartiles:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
Mean	38.581647	189778.4	10.080679	1077.648844	87.303830	40.437456
Standard deviation	13.640433	105550	2.572720	7385.292085	402.960219	12.347429
Variation	186.0614	1.114080e+10	6.618890	54542540	162376.9	152.4590
min	17	12285	1	0	0	1
25% (Q1)	28	117827	9	0	0	40
50% (Q2)	37	178356	10	0	0	40
75% (Q3)	48	237051	12	0	0	45
max	90	1484705	16	99999	4356	99

iii. # of missing value = 2399

iv. # of outlier = 2733

(2). How to identify the outlier? How to impute the missing value?

i. Using **standard scaler**, the data without 3 standard deviations could be seen as outliers.

```
# find outlier index
from sklearn.preprocessing import StandardScaler
X = df.drop(["workclass","education","marital-status","occupation","relationship","race","sex","native-country","class"

ss = StandardScaler()
scaled_df = ss.fit_transform(X)
scaled_df = pd.DataFrame(scaled_df, columns = X.columns)
abs_scaled_df = (abs(scaled_df)>3)
all_abs_scaled = abs_scaled_df.any(axis=1)
outlier_index = all_abs_scaled[all_abs_scaled==True]
outlier_index.index
# abs_scaled_df.iloc[1,range(0,6,1)]
```

```
Int64Index([   10,    23,    28,    32,    37,    40,    52,    77,    93,
               96,
            ...
            32445, 32458, 32459, 32469, 32476, 32494, 32511, 32518, 32525,
            32531],
           dtype='int64', length=2733)
```

```
# drop outlier
df_no_outlier = df.drop(outlier_index.index)
```

```
df_no_outlier.shape
```

```
(29828, 15)
```

```
scaled_df.shape
```

```
(32561, 6)
```

After dropping outliers, the amount of data is 29,828.

ii. Missing data in this dataset are all categorical variables, I replaced column **workclass**, **occupation** and **native-country** with their most frequent value, which were “Private”, “Prof-specialty”, and “United-States”.

```
# impute missing value_workclass
count = df["workclass"].value_counts(sort=True)
print(count)
df['workclass'] = df['workclass'].replace(np.nan, " Private")
```

```
# impute missing value_occupation
count = df["occupation"].value_counts(sort=True)
print(count)
df['occupation'] = df['occupation'].replace(np.nan, " Prof-specialty")
```

```
# impute missing value_native-country
count = df["native-country"].value_counts(sort=True)
print(count)
df['native-country'] = df['native-country'].replace(np.nan, " United-States")
```

註：此處有嘗試直接刪除 missing values，結果預測能力表現只略差一些些，詳見最後的 Model evaluation。

(3). How to transform the categorical variable to dummy variable?

Using **get_dummies** and set parameter **drop_first=True** to get n-1 dummy variables.

```
du = df
du = pd.get_dummies(du, columns=["workclass",
                                 "education",
                                 "marital-status",
                                 "occupation",
                                 "relationship",
                                 "race",
                                 "sex",
                                 "native-country"], drop_first=True)
du.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 98 columns):
```

(4). How to "randomly" split the dataset into training dataset and testing dataset (eg. 80% vs. 20%)?

Using **train_test_split**, we can split the dataset into training and testing dataset:

```
# split dataset
from sklearn.model_selection import train_test_split

x = du
y = du[['class_ >50K']]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(23862, 95) (5966, 95) (23862, 1) (5966, 1)
```

(5). Please use the Generalized Linear Model (GLM) (OR Logistic Regression) to predict the "Class" in the testing dataset.

```
from sklearn.linear_model import LogisticRegression

# create classifier
clf = LogisticRegression(penalty='l2').fit(x_train_std, y_train.values.ravel())

# predict test data
predict = clf.predict(x_test_std)
clf.predict_proba(x_test_std)[:10]
```

```
array([[4.52785501e-01, 5.47214499e-01],
       [7.18345312e-01, 2.81654688e-01],
       [6.53207132e-01, 3.46792868e-01],
       [9.90476253e-01, 9.52374743e-03],
       [7.79017171e-01, 2.20982829e-01],
       [4.70698635e-01, 5.29301365e-01],
       [9.99874969e-01, 1.25031057e-04],
       [7.46565588e-01, 2.53434412e-01],
       [9.75455300e-01, 2.45447001e-02],
       [9.89696811e-01, 1.03031892e-02]])
```

The last 10 predict results are [1, 0, 0, 0, 0, 1, 0, 0, 0, 0].

(1: class > 50K, 0: class <= 50K)

Discussion

A. Model coefficient

```
# coefficient of prediction model
coef = clf.coef_[0]
for i in range(len(du.columns[:-1])):
    print(du.columns[i], ":", coef[i])
```

```
age : 0.3579073713111122
fnlwgt : 0.04923542315876943
education-num : 0.6045567038938487
capital-gain : 0.7934957677449259
capital-loss : -0.09710853030512402
hours-per-week : 0.4126305272838166
workclass_ Local-gov : -0.16852716392050196
workclass_ Never-worked : -0.09992473496509088
workclass_ Private : -0.2641779513931872
workclass_ Self-emp-inc : -0.06522540433360347
workclass_ Self-emp-not-inc : -0.29858425225484625
workclass_ State-gov : -0.16927345502717706
workclass_ Without-pay : -0.1489954371906052
education_ 11th : -0.01400567499366151
```

從係數來看，對於 class 有較大影響力的因子有：

- education-num(0.6046): 受教育的時間較長，可能獲得較高收入。
- capital-gain(0.7935): 資本獲利越多，可能獲得較高收入。
- Married-civ-spouse(1.1032): 已婚者且配偶為平民，收入可能較高。(有另一選項為已婚且配偶為軍人，係數相對較小)

B. Model evaluation

- 本次預測模型準確率為 85.67%

```
# model accuracy
from sklearn.metrics import confusion_matrix,classification_report
print("Accuracy score:", clf.score(x_test_std, y_test))
```

```
Accuracy score: 0.856687898089172
```

- 以下是更詳細的評估如 precision, recall, F1-score, ROC 曲線

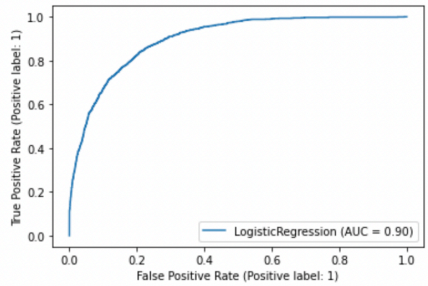
```
# model evaluation_report
print("report:\n",classification_report(y_test,predict,labels=[1,0],target_names=[">50K","<=50K"]))
```

```
report:
               precision    recall  f1-score   support

        >50K       0.71      0.58      0.64      1294
       <=50K       0.89      0.93      0.91      4672

    accuracy                           0.86      5966
   macro avg       0.80      0.76      0.77      5966
weighted avg       0.85      0.86      0.85      5966
```

此模型 precision 大於 recall，代表其預測結果為 >50K 的人，有七成確實收入大於五萬；而實際上收入大於五萬的人，卻不是都能找出來，僅能「召回」約六成。

另外，能從 F1-score=0.64 與下圖的 ROC 曲線 (0.5<AUC<1.0) 看出，此模型雖不到完美，但仍有預測價值。

```
from sklearn.metrics import roc_curve, auc, plot_roc_curve
plot_roc_curve(clf, x_test_std, y_test)
plt.show()
```



而當初如果直接 drop missing values 的表現如下：

```
# drop missing value

df_drop = df.dropna()
df_drop.head(28)
print(df_drop.shape)
print(df.shape)
```

```
report:
               precision    recall  f1-score   support

        >50K       0.70      0.57      0.63      1294
       <=50K       0.89      0.93      0.91      4672

    accuracy                           0.85      5966
   macro avg       0.80      0.75      0.77      5966
weighted avg       0.85      0.85      0.85      5966
```

C. Association Rule- Market Basket Analysis

(1). How to handle the raw dataset via data preprocessing?

i. 將原始轉為交易清單並去除 NaN

```
# create transaction list
trans = []
for i in range(0, 9835):
    trans.append([str(df.values[i,j]) for j in range(0, 32)])

# remove NA values
for idx, item in enumerate(trans):
    while 'nan' in trans[idx]:
        trans[idx].remove("nan")
```

ii. 透過 transaction encoder 將清單轉為以商品名稱為欄位的形式

```
# transaction encoder
te = TransactionEncoder()
trans = te.fit_transform(trans)
trans_df = pd.DataFrame(trans, columns = te.columns_)
trans_df.shape
```

```
(9835, 169)
```

```
trans_df
```

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	bags	baking powder	bathroom cleaner	beef	...	turkey	vinegar	waffles	whipped/sour cream	whisky	white bread	white wine
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

iii. 找出僅賣出一次的商品，為 ”baby food” 與 ”sound storage medium”

```
# 找出總銷售量低者
df.stack().value_counts().tail()
```

```
bags                        4
kitchen utensil             4
preservation products       2
baby food                   1
sound storage medium        1
dtype: int64
```

刪除欄位 ”baby food” 與 ”sound storage medium”

```
# 去除僅賣出一次的商品
trans_df.drop(["baby food","sound storage medium"], axis=1, inplace=True)
trans_df.shape
```

```
(9835, 167)
```

iv. 進行關聯分析 (Apriori Algorithm)

```
# Apply apriori analysis
frequent_itemsets = apriori(trans_df, min_support=0.001, use_colnames=True)
frequent_itemsets
```

	support	itemsets
0	0.008033	(Instant food products)
1	0.033452	(UHT-milk)
2	0.003559	(abrasive cleaner)
3	0.003254	(artif. sweetener)
4	0.017692	(baking powder)
...	...	...
13487	0.001017	(root vegetables, yogurt, citrus fruit, other ...
13488	0.001017	(root vegetables, yogurt, other vegetables, tr...
13489	0.001322	(root vegetables, yogurt, other vegetables, tr...
13490	0.001322	(root vegetables, rolls/buns, yogurt, other ve...
13491	0.001118	(root vegetables, yogurt, other vegetables, tr...

13492 rows × 2 columns

可以看到多達 13492 條關聯規則，此時用 min_support 與 min_confidence 過濾出的前五筆資料如下：(min_support=0.001, min_confidence=0.015)

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(Instant food products)	(butter)	0.008033	0.055414	0.001220	0.151899	2.741145
1	(Instant food products)	(curd)	0.008033	0.053279	0.001322	0.164557	3.088583
2	(Instant food products)	(hamburger meat)	0.008033	0.033249	0.003050	0.379747	11.421438
3	(Instant food products)	(other vegetables)	0.008033	0.193493	0.002745	0.341772	1.766332
4	(Instant food products)	(pastry)	0.008033	0.088968	0.001423	0.177215	1.991899

(2). What's the top 5 association rules? Show the support, confidence, and lift to each specific rule, respectively?

續上，試著以較高的 confidence 與 lift 門檻找出欲關注的商品組合，使用的 threshold 來自 quantile=0.95 時，confidence = 0.67, lift = 7.73，篩選出前五名：

```
ar.quantile(0.95)[["support","confidence","lift"]]
```

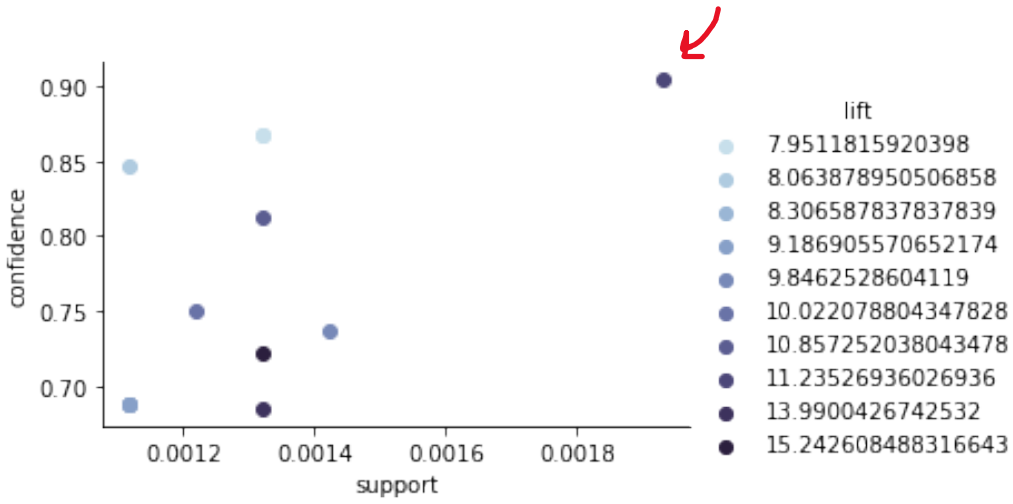
```
support       0.004270
confidence    0.666667
lift          7.727058
Name: 0.95, dtype: float64
```

```
# Filtering association rules
top_rules = ar[(ar["support"]>0.001118)&
               (ar["confidence"]>=0.67)&
               (ar["lift"]>=7.73)].sort_values(by=["lift"],ascending=False)
top_rules.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
31746	(rice, yogurt, whole milk)	(other vegetables, root vegetables)	0.001830	0.047382	0.001322	0.722222	15.242608
31747	(other vegetables, rice, yogurt)	(root vegetables, whole milk)	0.001932	0.048907	0.001322	0.684211	13.990043
2428	(red/blush wine, liquor)	(bottled beer)	0.002135	0.080529	0.001932	0.904762	11.235269
31745	(rice, root vegetables, yogurt)	(other vegetables, whole milk)	0.001627	0.074835	0.001322	0.812500	10.857252
28851	(brown bread, root vegetables, pip fruit)	(other vegetables, whole milk)	0.001627	0.074835	0.001220	0.750000	10.022079

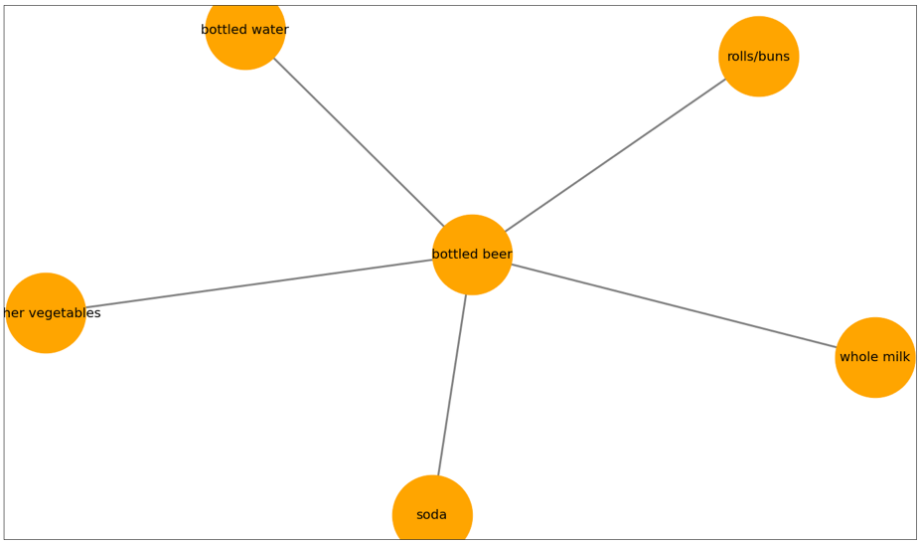
(3). Please provide/guess the "story" to interpret **one of top-5 rules** you are interested in.

觀察 top rules(共 13 筆資料)的參數分佈後，決定探討右上角的這筆資料。



(red/blush wine, liquor) → (bottled beer): 顯示此資料集中，有九成的顧客在購買紅酒或其他酒類後，會購買罐裝啤酒。而購買紅酒或其他酒類的顧客，比起其他顧客，購買啤酒的機率大約是 11 倍。

A. 藉由網路圖觀察 (1) red/blush wine, (2) liquor, (3) bottled beer 在原始資料中，各自關聯性較強的商品：



其中有滿多重複的商品如 whole milk, rolls/buns, other vegetables…，但這些商品好像和酒類沒什麼關聯，看起來像幫家裡添購牛奶、麵包、蔬菜的家庭主婦順便幫老公買酒，但她們只會買紅酒和啤酒。

B. 觀察商品中其他酒類，除了紅酒與啤酒外，還出現白蘭地、開胃酒、prosecco 氣泡酒、威士忌、白酒...。其中啤酒銷售量為 792，紅酒銷售量為 189，其他酒類就更低了。因此或許能解讀啤酒是必備品，買其他酒類的顧客有很大機率會買啤酒，相反地買啤酒的顧客不一定會買其他酒類。

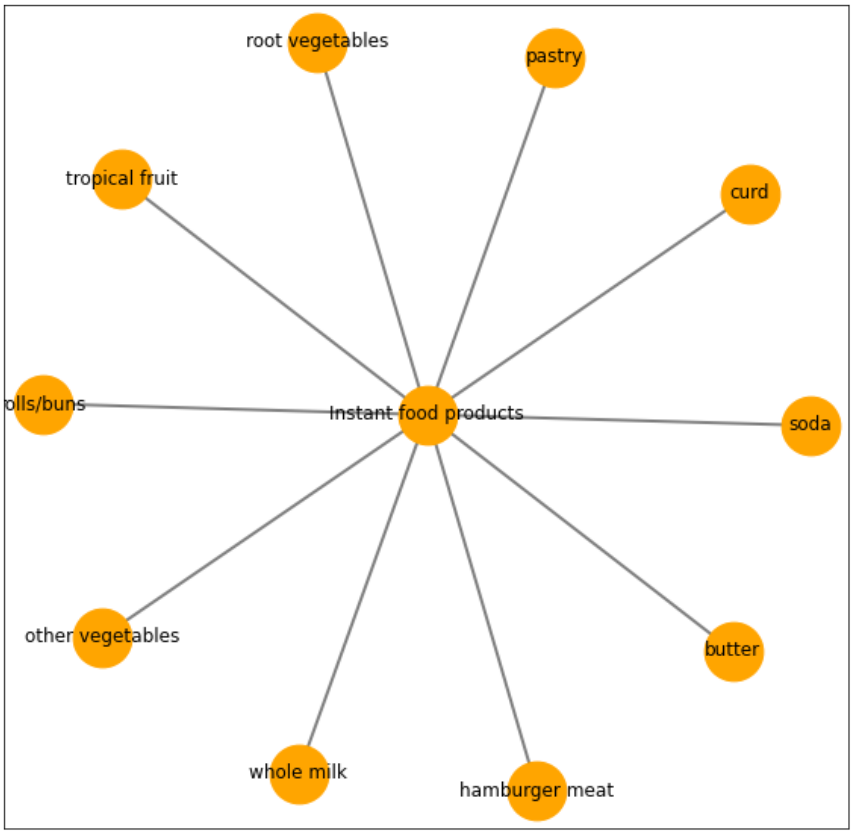
(4). Give a visualization graph of your association rules.

i. Network of initial association rules
(min_support=0.001, min_confidence=0.15)

由題目準則篩選出來的關聯規則中，主要以 Instant food products 為主：

```
# Filtering association rules
ar = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.15)
ar.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Instant food products)	(butter)	0.008033	0.055414	0.001220	0.151899	2.741145	0.000775	1.113765
1	(Instant food products)	(curd)	0.008033	0.053279	0.001322	0.164557	3.088583	0.000894	1.133196
2	(Instant food products)	(hamburger meat)	0.008033	0.033249	0.003050	0.379747	11.421438	0.002783	1.558640
3	(Instant food products)	(other vegetables)	0.008033	0.193493	0.002745	0.341772	1.766332	0.001191	1.225271
4	(Instant food products)	(pastry)	0.008033	0.088968	0.001423	0.177215	1.991899	0.000709	1.107254

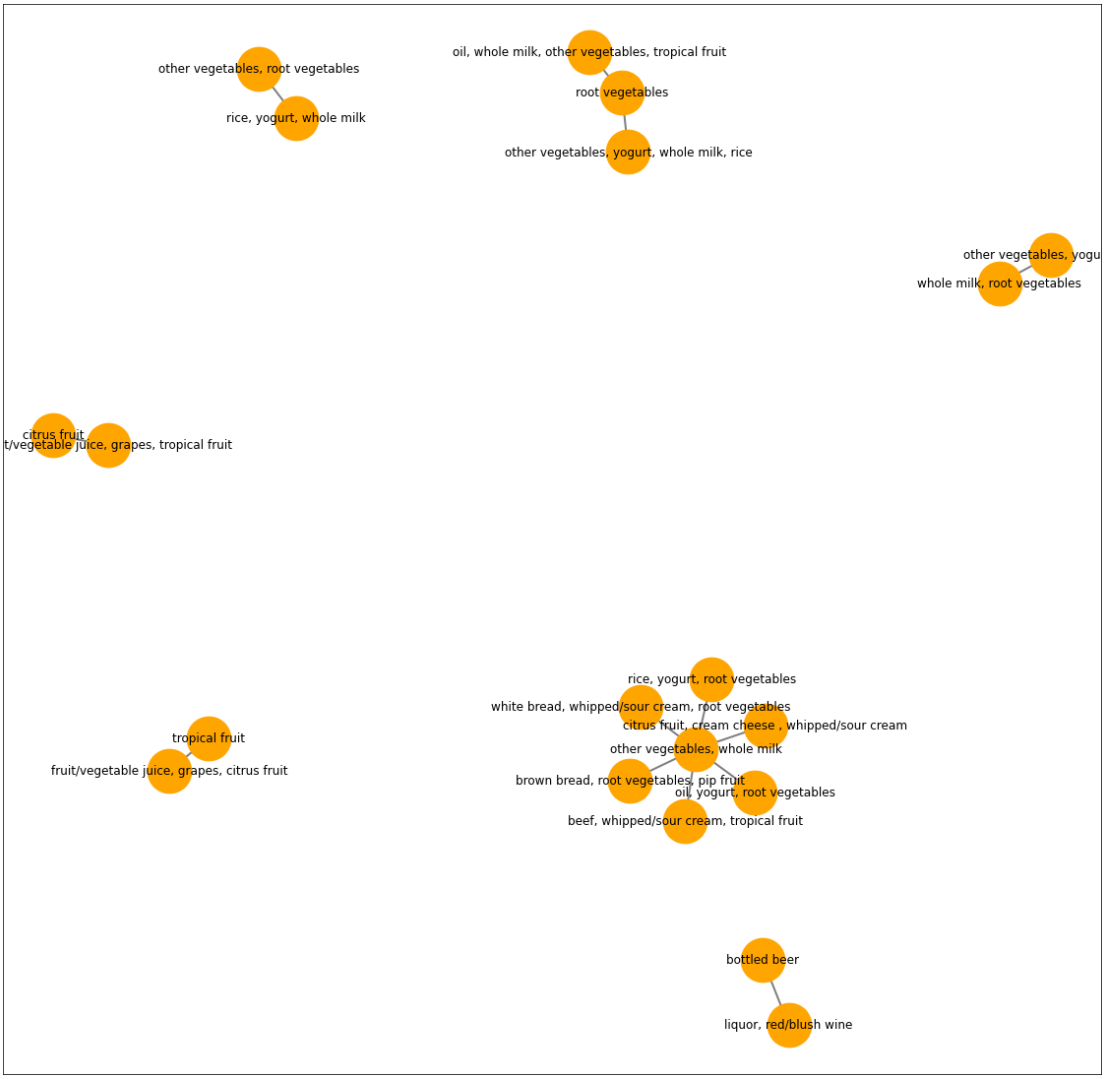


Code:

```
# Network of initial association rules
fig, ax = plt.subplots(figsize=(10,10))
GA = nx.from_pandas_edgelist(ar_list.head(10), source='antecedents', target='consequents')
pos = nx.spring_layout(GA)
nx.draw_networkx_nodes(GA, pos, node_size = 1500, node_color = "orange")
nx.draw_networkx_edges(GA, pos, width = 2, edge_color = 'grey')
nx.draw_networkx_labels(GA, pos, font_size = 12)
plt.show()
```

ii. Network of top_rules:

更進一步使用 confidence 與 lift 篩選出的 top rules 中，可以看出商品間彼此的關聯性，以及分群可能代表的不同客群。

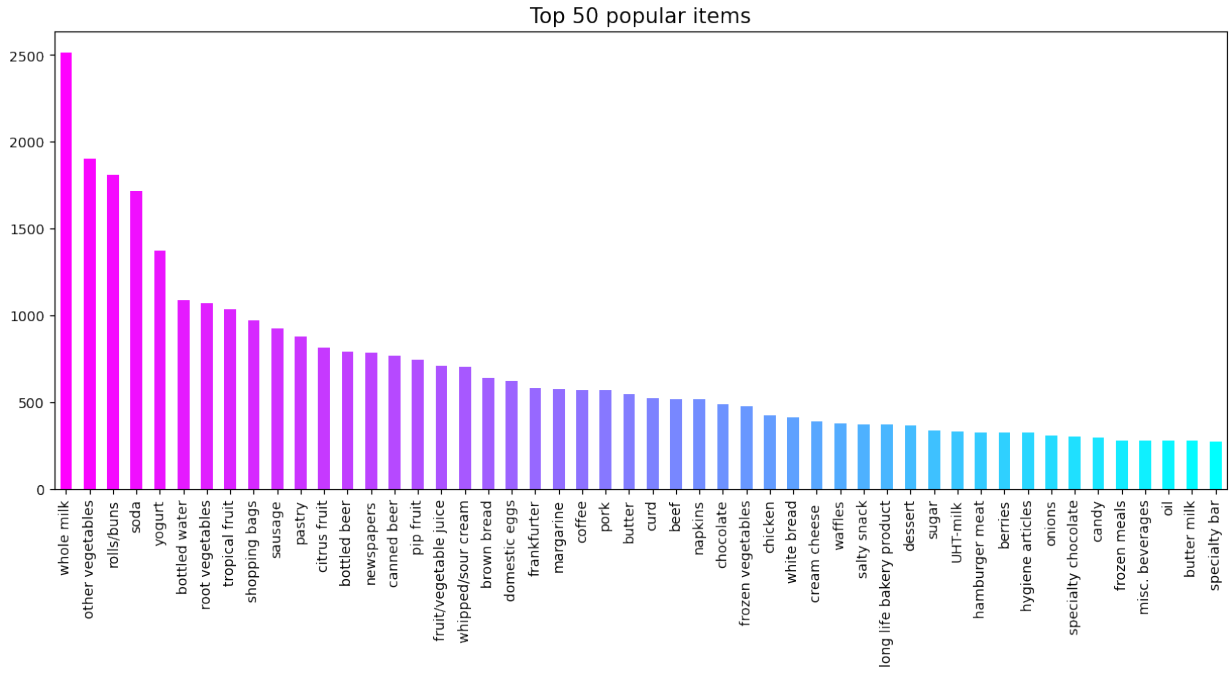


Code:

```
# Visualization
new_rules = top_rules.copy()
new_rules["antecedents"] = top_rules["antecedents"].apply(lambda x: ', '.join(list(x))).astype("unicode")
new_rules["consequents"] = top_rules["consequents"].apply(lambda x: ', '.join(list(x))).astype("unicode")
```

```
# Network of top_rules
fig, ax = plt.subplots(figsize=(20,20))
GA = nx.from_pandas_edgelist(new_rules, source='antecedents', target='consequents')
pos = nx.spring_layout(GA)
nx.draw_networkx_nodes(GA, pos, node_size = 2000, node_color = "orange")
nx.draw_networkx_edges(GA, pos, width = 2, edge_color = 'grey')
nx.draw_networkx_labels(GA, pos, font_size = 12)
plt.show()
```

iii. Top 50 popular items in original dataset



whole milk	2513
other vegetables	1903
rolls/buns	1809
soda	1715
yogurt	1372
bottled water	1087
root vegetables	1072
tropical fruit	1032
shopping bags	969
sausage	924
pastry	875
citrus fruit	814
bottled beer	792
newspapers	785
canned beer	764
pip fruit	744
fruit/vegetable juice	711
whipped/sour cream	705
brown bread	638
domestic eggs	624
frankfurter	580
margarine	576
coffee	571
pork	567
butter	545
curd	524
beef	516
napkins	515
chocolate	488
frozen vegetables	473
chicken	422
white bread	414
cream cheese	390
waffles	378
salty snack	372
long life bakery product	368
dessert	365
sugar	333
UHT-milk	329
berries	327
hamburger meat	327
hygiene articles	324
onions	305
specialty chocolate	299
candy	294
frozen meals	279
misc. beverages	279
oil	276
butter milk	275
specialty bar	269

Reference

[1] Market Basket Analysis. Kaggle.

https://www.kaggle.com/roshansharma/market-basket-analysis

[2] Manufacturing-Data-Science. Github.

https://github.com/PO-LAB/Manufacturing-Data-Science/tree/master/MDS

[3] Introduction to Market Basket Analysis in Python. Practical Business Python.

https://pbpython.com/market-basket-analysis.html

[4] Python 實戰篇：Apriori Algorithm (Mlxtend library)。

https://artsdatascience.wordpress.com/2019/12/10/python-%E5%AF%A6%E6%88%B0%E7%AF%87%EF%BC%9Aapriori-algorithm/