資管碩一 R10725026 黃奕滔

1.

(a).

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.0339	0.018	110.528	0.000	1.998	2.070
f0	-0.0055	0.025	-0.224	0.823	-0.054	0.043
f1	0.0440	0.028	1.571	0.117	-0.011	0.099
f2	0.3140	0.035	9.028	0.000	0.246	0.382
f3	0.0186	0.042	0.447	0.655	-0.063	0.100
f4	-0.0035	0.038	-0.091	0.928	-0.078	0.072
f5	-0.0740	0.030	-2.483	0.013	-0.133	-0.015
f6	-0.0710	0.026	-2.742	0.006	-0.122	-0.020
f7	0.0235	0.028	0.853	0.394	-0.031	0.078
f8	0.0410	0.019	2.170	0.030	0.004	0.078
f9	1.953e-17	1.48e-17	1.323	0.186	-9.46e-18	4.85e-17
f10	-0.0446	0.021	-2.119	0.035	-0.086	-0.003
f11	-0.0292	0.020	-1.438	0.151	-0.069	0.011
f12	-0.0006	0.022	-0.027	0.979	-0.044	0.043
f13	0.0336	0.024	1.412	0.159	-0.013	0.080
f14	-0.1832	0.021	-8.898	0.000	-0.224	-0.143
f15	-0.1061	0.019	-5.565	0.000	-0.144	-0.069
f16	-0.0358	0.020	-1.756	0.080	-0.076	0.004
f17	0.0633	0.019	3.409	0.001	0.027	0.100
f18	-0.1904	0.021	-9.194	0.000	-0.231	-0.150
f19	0.0278	0.026	1.051	0.294	-0.024	0.080
f20	0.0126	0.020	0.644	0.520	-0.026	0.051
f21	-0.0357	0.028	-1.263	0.207	-0.091	0.020
f22	0.0747	0.021	3.533	0.000	0.033	0.116
f23	-0.0088	0.020	-0.442	0.659	-0.048	0.030
f24	0.0193	0.024	0.800	0.424	-0.028	0.067
f25	-0.0679	0.020	-3.406	0.001	-0.107	-0.029
f26	-0.0360	0.022	-1.625	0.105	-0.080	0.008
f27	-0.0062	0.019	-0.324	0.746	-0.044	0.031

Omnibus:	39.669	Durbin-Watson:	2.004
Prob(Omnibus):	0.000	Jarque-Bera (JB):	147.525
Skew:	0.090	Prob(JB):	9.23e-33
Kurtosis:	5.383	Cond. No.	1.39e+16

R-squared:	0.495
Adj. R-squared:	0.472
F-statistic:	21.52
Prob (F-statistic):	1.16e-70

(b). Before adapting linear regression, we should check whether there truly exist linear relationships between features and labels. Maybe there are some interactions among the features or higher degree polynomial terms.

(c). Sorted p-values and significant features

```
furnace_pvalues = furnace_result.pvalues
print(furnace_pvalues.sort_values())
```

✓ 0.4s

```
Intercept    0.000000e+00
f18          6.355895e-19
f2           2.430368e-18
f14          6.896905e-18
f15          3.971326e-08
f22          4.429125e-04
f17          6.967770e-04
f25          7.041639e-04
f6           6.295708e-03
f5           1.331207e-02
f8           3.036614e-02
f10          3.450386e-02
f16          7.959189e-02
f26          1.046465e-01
f1           1.167404e-01
f11          1.509174e-01
f13          1.585856e-01
f9           1.863555e-01
f21          2.071131e-01
```

```python
for i in range(len(furnace_pvalues.sort_values())):
    if furnace_pvalues.sort_values()[i] < 0.01:
        print(furnace_pvalues.sort_values().index[i])
```

✓ 0.6s

```
Intercept
f18
f2
f14
f15
f22
f17
f25
f6
```

(d).

(1). Normality test

Normality check

H_0 : the residual is normal
H_1 : the residual is not normal

```python
check_norm(furnace_result.resid_pearson)
```

✓ 1.1s

```
Shapiro: statistics=0.935, p=0.000
```

P-value of the Shapiro Normality test < 0.05, we have 95% confident to reject null hypothesis. Thus, residual distribution isn't normal.

(2). Independence (aka check multicollinearity)

Independent assumption

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = [variance_inflation_factor(furnace_X_const.values, i) for i in range(furnace_X_const.shape[1])]
pd.DataFrame({'vif': vif[1:]}, index=furnace_X.columns).T
```

✓ 0.3s Python

	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	...	f18	f19	f20	f21	f22	f23	f24	f25	f26	
vif	1.81784	2.314364	3.571219	5.095433	4.30931	2.623037	1.983188	2.24362	1.054478	NaN	...	1.266249	2.060656	1.12521	2.356358	1.319547	1.162346	1.7176	1.17451	1.449481	1.085

Through checking VIF (Variance Inflation Factor), there’s no strong multicollinearity features (VIF > 10) that must be removed.

(3). Homogeneity of Variance (aka Homoscedasticity)

H_0 : Homoscedasticity
H_1 : Hetroscedasticity

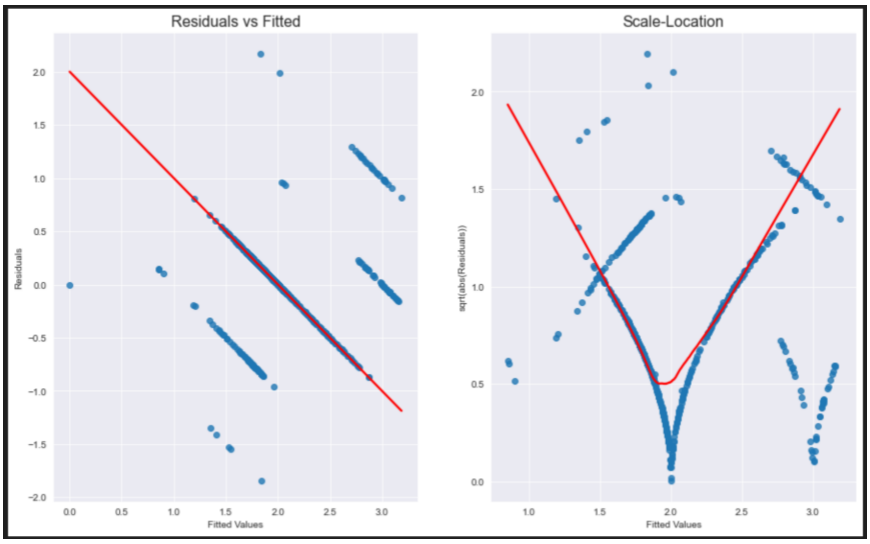
Breusch-Pagan test ----

	value
Lagrange multiplier statistic	1.406203e+02
p-value	5.871843e-17
f-value	6.431711e+00
f p-value	5.311816e-20

Goldfeld-Quandt test ----

	value
F statistic	1.024212
p-value	0.420429

B-P test reject null hypothesis, while G-Q test doesn’t. We can’t surely infer whether Homogeneity of Variance exist or not.



2.

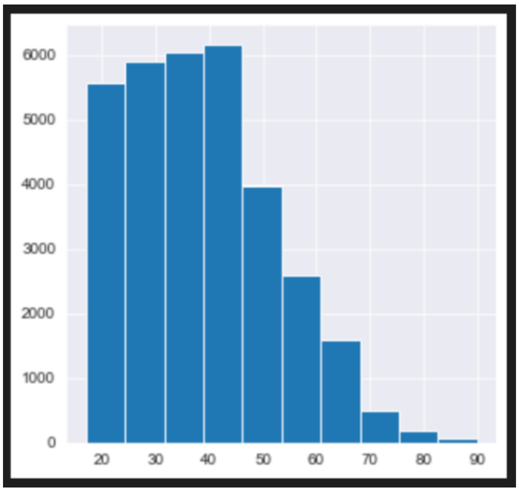
(1). Distribution plot is posed by column number order.

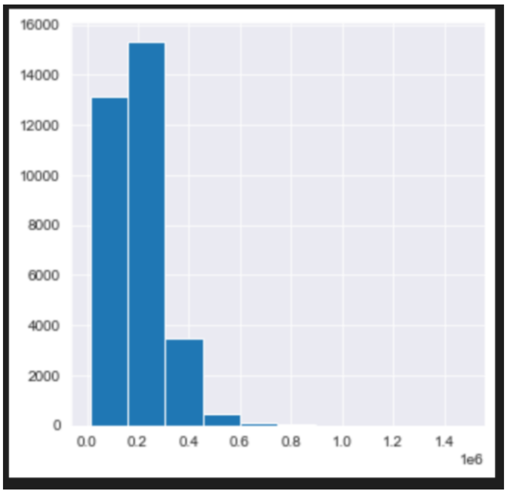
census_des

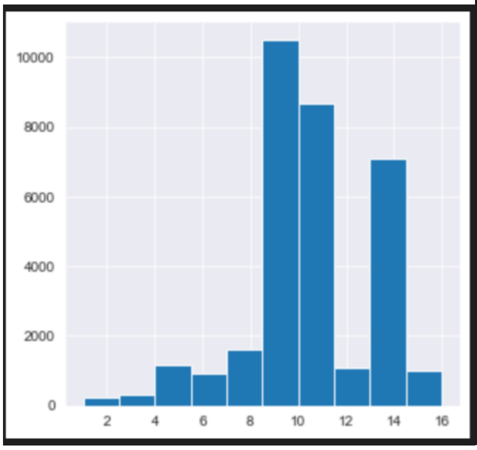
✓ 0.2s

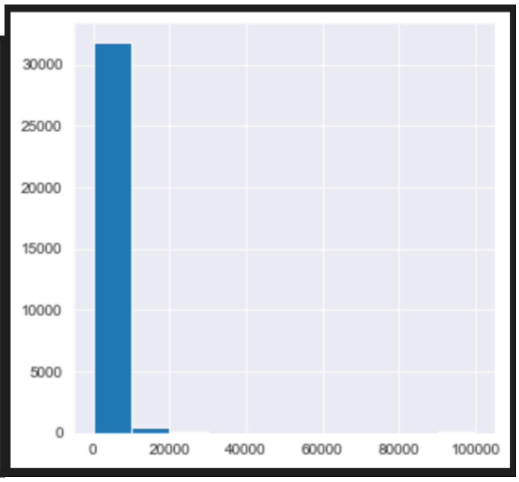
Python

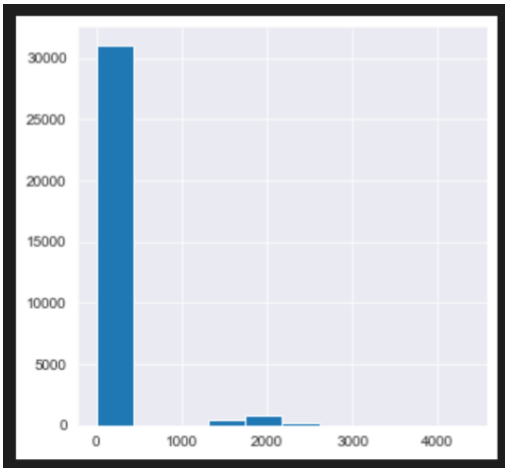
	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	class	education	marital-status	native-country	occupation	race	relationship	sex	workclass
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
na_cnt	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	583.0	1843.0	0.0	0.0	0.0	1836.0
outlier_cnt	121.000000	NaN	347.000000	NaN	219.000000	NaN	NaN	NaN	NaN	NaN	215.0	1470.0	440.0	NaN	NaN

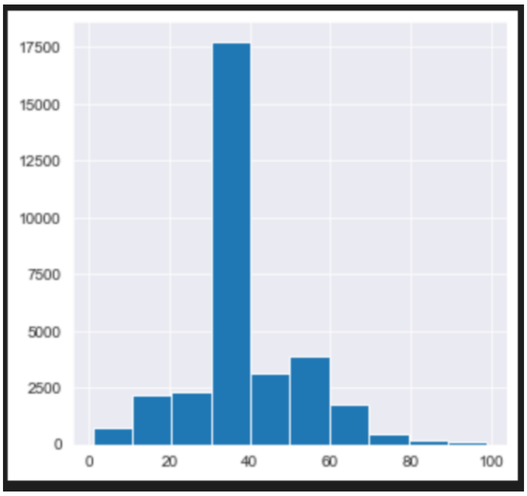












(2). Using z-score : If z-score > threshold, the value is outlier. (thres. is set to 3)

```
def detect_outlier(data_1):

    threshold=3
    mean_1 = np.mean(data_1)
    std_1 =np.std(data_1)
    outliers = []

    for y in data_1:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

census_des

✓ 0.2s

Python

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	class	education	marital-status	native-country	occupation	race	relationship	sex	workclass
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
na_cnt	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	583.0	1843.0	0.0	0.0	0.0	1836.0
outlier_cnt	121.000000	NaN	347.000000	NaN	219.000000	NaN	NaN	NaN	NaN	NaN	215.0	1470.0	440.0	NaN	NaN

Take most-frequent value to replace nan (impute the missing values)

```
imputer = SimpleImputer(strategy='most_frequent')
# imputer = KNNImputer()
imputer.fit(census_data)
imputed_census = imputer.transform(census_data)
```

(3). We use the pandas.get_dummies() method to transform the categorical columns into dummy one. Because the category columns in the dataset is mostly binary, we can use drop_first=True option to make appropriate dummies.

Ex.

```python
census_dummies = pd.get_dummies(data=imputed_census,
columns=["workclass", "education", "marital-status", "occupation",
"relationship", "race", "sex", "native-country", "class"], drop_first=True)
census_dummies
```

✓ 0.3s Python

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	workclass_ Local-gov	workclass_ Never-worked	workclass_ Private	workclass_ Self-emp-inc	workclass_ Self-emp-not-inc	workclass_ State-gov	workclass_ Without-pay	education_ 11th	education_ 12th	education_ 1st-4th	educatic 5th-(
0	39.0	77516.0	13.0	2174.0	0.0	40.0	0	0	0	0	0	1	0	0	0	0	
1	50.0	83311.0	13.0	0.0	0.0	13.0	0	0	0	0	1	0	0	0	0	0	
2	38.0	215646.0	9.0	0.0	0.0	40.0	0	0	1	0	0	0	0	0	0	0	
3	53.0	234721.0	7.0	0.0	0.0	40.0	0	0	1	0	0	0	0	1	0	0	
4	28.0	338409.0	13.0	0.0	0.0	40.0	0	0	1	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27.0	257302.0	12.0	0.0	0.0	38.0	0	0	1	0	0	0	0	0	0	0	
32557	40.0	154374.0	9.0	0.0	0.0	40.0	0	0	1	0	0	0	0	0	0	0	
32558	58.0	151910.0	9.0	0.0	0.0	40.0	0	0	1	0	0	0	0	0	0	0	
32559	22.0	201490.0	9.0	0.0	0.0	20.0	0	0	1	0	0	0	0	0	0	0	
32560	52.0	287927.0	9.0	15024.0	0.0	40.0	0	0	0	1	0	0	0	0	0	0	

32561 rows × 98 columns

(4). We can use the sklean.model_selection.train_test_split() method. With chosen random_state option, we can perform randomly split. Meanwhile, set the proportion of train/test dataset by test_size option.

```python
census_X = census_dummies.drop(columns=["class_ >50K"])
census_y = census_dummies["class_ >50K"]

census_X_train, census_X_test, census_y_train, census_y_test = train_test_split(census_X, census_y, test_size=0.20, random_state=42)
```

✓ 0.1s

(5). Fit the logistic regression model with Xy_train and evaluate model accuracy with score method using Xy_test.

```python
from sklearn.linear_model import LogisticRegression
census_lr = LogisticRegression().fit(census_X_train, census_y_train)
```

✓ 0.6s

```
/Users/cupid/Desktop/projects/MDS_hw/env/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```python
print("Mean accuracy of self.predict(X) wrt. y:", census_lr.score(census_X_test, census_y_test))
```

✓ 0.1s

```
Mean accuracy of self.predict(X) wrt. y: 0.7821280515891295
```

3.

(1). There’s a package named mlxtend with a method called TransactionEncoder(), which can help us transform the transaction data into Boolean table form.

```python
groceries_data = pd.read_csv("MDS_Assignment1_groceries.csv", header=None, sep='\n')
groceries_data = groceries_data.values.tolist()
trans = []
for ele in groceries_data:
    trans.append(ele[0].split(","))
```

✓ 0.7s

```python
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(trans).transform(trans)
trans_bool = pd.DataFrame(te_ary, columns=te.columns_)
trans_bool
```

✓ 0.2s

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	bags	baking powder	bathroom cleaner	beef	berries	beverages	bottled beer	bottled water	brandy	brown bread	butter	butter milk	cake bar	candles	candy
0	False	False	False	False	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	False	False	False	False
2	False	False	False	False	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	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9830	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	True	False	False	False	False
9831	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
9832	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False
9833	False	False	False	False	False	False	False	False	False	False	False	False	True	True	False	False	False	False	False	False	False
9834	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

(2).

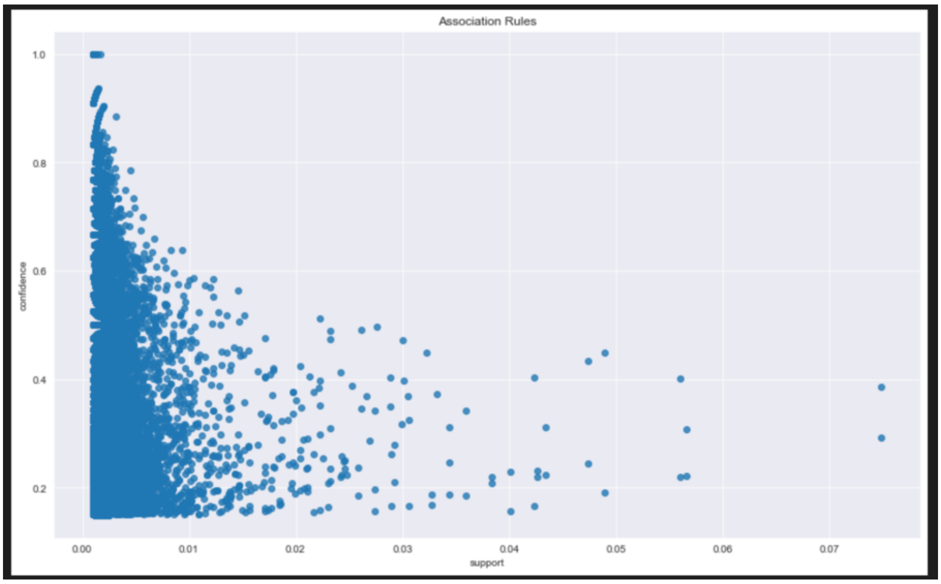
```python
rules = rules[rules['confidence'] >= 0.15].sort_values(by='confidence', ascending=False)
rules[:5]
```

✓ 0.6s

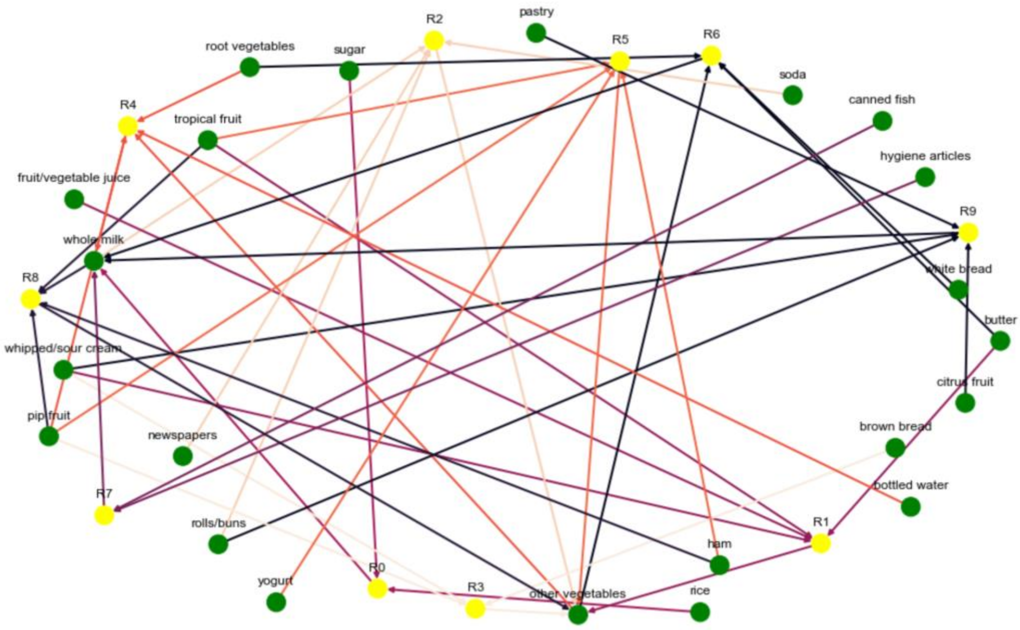
	antecedents	consequents	antecedent support	consequent suppo	support	confidence	lift	leverage	conviction
44790	(rice, sugar)	(whole milk)	0.001220	0.25551	0.001220	1.0	3.913649	0.000908	inf
93000	(tropical fruit, fruit/vegetable juice, whippe...	(other vegetables)	0.001017	0.19349	0.001017	1.0	5.168156	0.000820	inf
98790	(rolls/buns, soda, newspapers, whole milk)	(other vegetables)	0.001017	0.19349	0.001017	1.0	5.168156	0.000820	inf
55141	(pip fruit, brown bread, whipped/sour cream)	(other vegetables)	0.001118	0.19349	0.001118	1.0	5.168156	0.000902	inf
91768	(other vegetables, pip fruit, bottled water, r...	(whole milk)	0.001118	0.25551	0.001118	1.0	3.913649	0.000833	inf

(3). As I see it, row 93000 is the most interpretable row in the first five rules. It's reasonable that people awarded of the important of health to consume balanced amount of vegetables, fruits, protein and fat. Due to the aforementioned reason, when someone buys fruit, juice, hopefully will also buy vegetables to fulfill daily demand of nutrients.

(4). Relationship between confidence and support:



10 of the rules connections:



Ref.

linear regression 回歸模型 and 檢測

https://towardsdatascience.com/verifying-the-assumptions-of-linear-regression-in-python-and-r-f4cd2907d4c0

outlier detection

https://medium.com/datadriveninvestor/finding-outliers-in-dataset-using-python-efc3fce6ce32

impute data (Compensate missing value)

https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779

association rules

http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/
https://artsdatascience.wordpress.com/2019/12/10/python-%E5%AF%A6%E6%88%B0%E7%AF%87%EF%BC%9Aapriori-algorithm/
https://pbpython.com/market-basket-analysis.html
(association rules visualization)
https://intelligentonlinetools.com/blog/2018/02/10/how-to-create-data-visualization-for-association-rules-in-data-mining/