Frequent Itemset Mining

Yutong Wang

Data

Dataset Source:
Kohavi, R. and Becker, B. (1996). UCI Machine Learning Repository, Adult Data Set [http://archive.ics.uci.edu/ml/datasets/Adult]. Irvine, CA: University of California, School of Information and Computer Science.

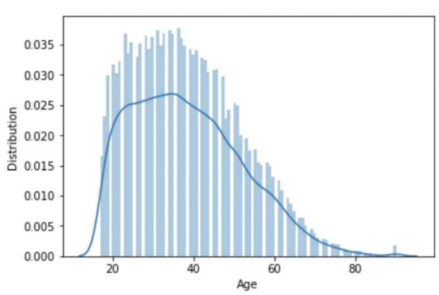
Dataset Info:
Size: (32561, 15)
Attribute(15): object(9) & int(6)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
Age               32561 non-null int64
Workclass         30725 non-null object
Fnlwgt            32561 non-null int64
Education         32561 non-null object
Education_num     32561 non-null int64
Marital_status    32561 non-null object
Occupation        30718 non-null object
Relationship      32561 non-null object
Race              32561 non-null object
Sex               32561 non-null object
Capital_gain      32561 non-null int64
Capital_loss      32561 non-null int64
Hours_per_week    32561 non-null int64
Native_country    31978 non-null object
Income            32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

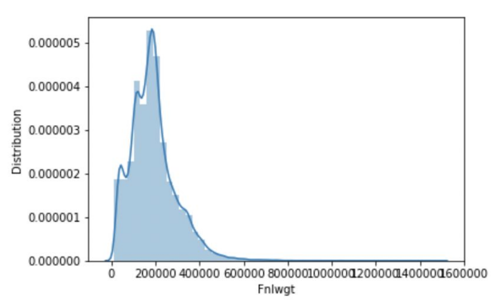
Data Visualization

Numerical attributes' distribution:

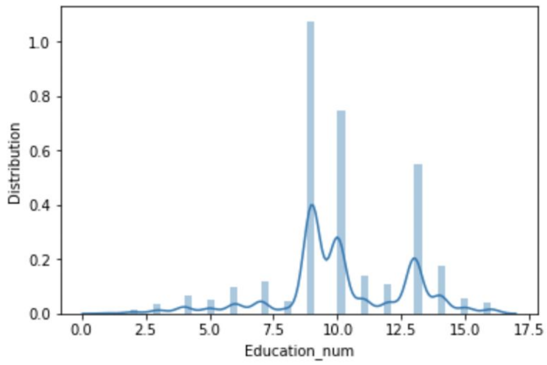
```
The minimum age is 17
The maximum age is 90
```



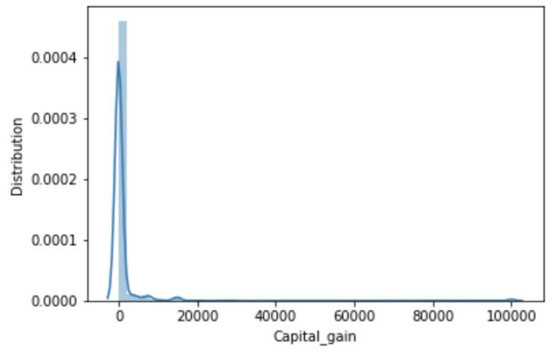

```
The minimum age is 12285
The maximum age is 1484705
```



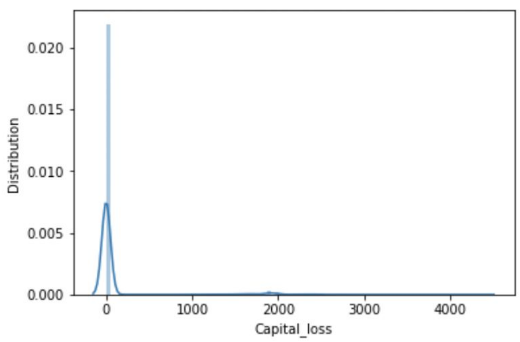
```
The minimum age is 1
The maximum age is 16
```



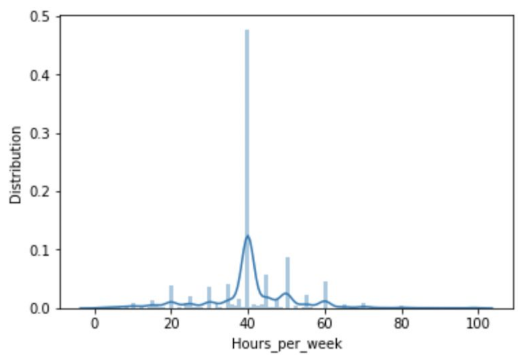

```
The minimum age is 0
The maximum age is 99999
```



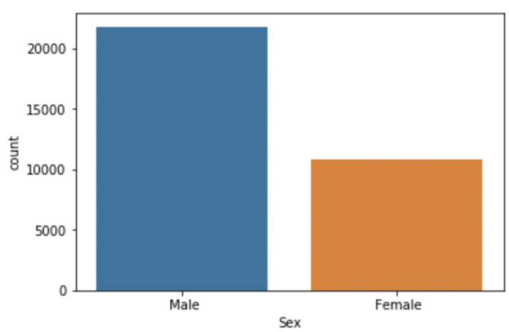

```
The minimum age is 0
The maximum age is 4356
```

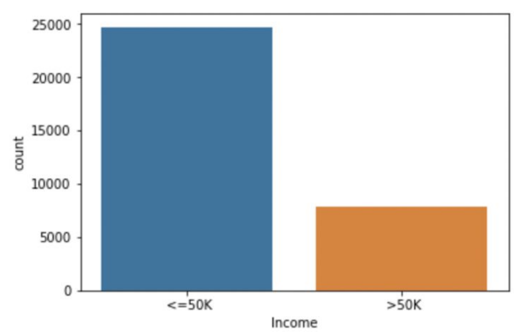


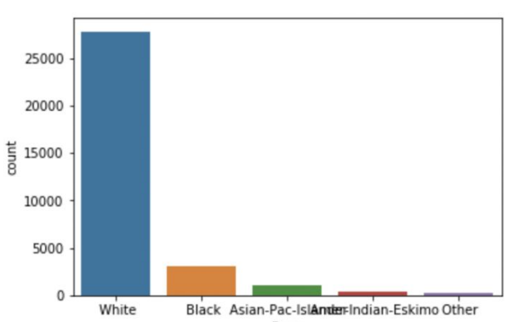

```
The minimum age is 1
The maximum age is 99
```

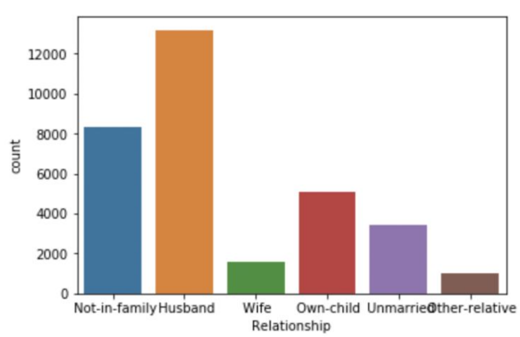


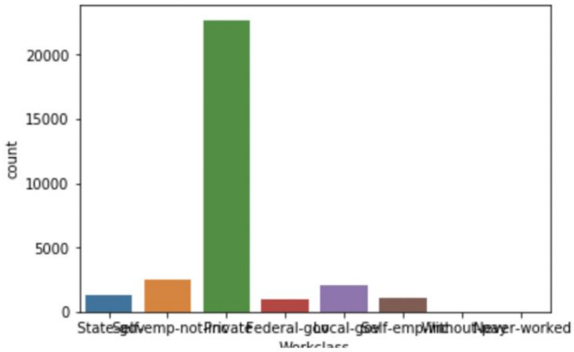
Categorical attributes' distribution:

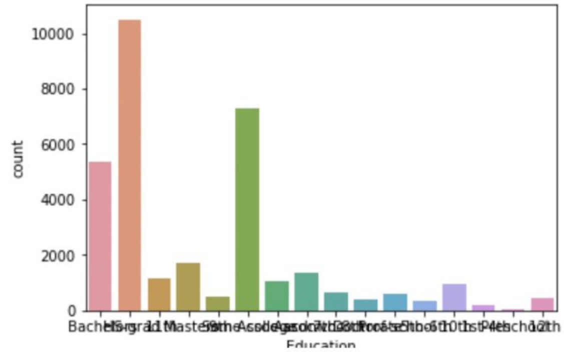


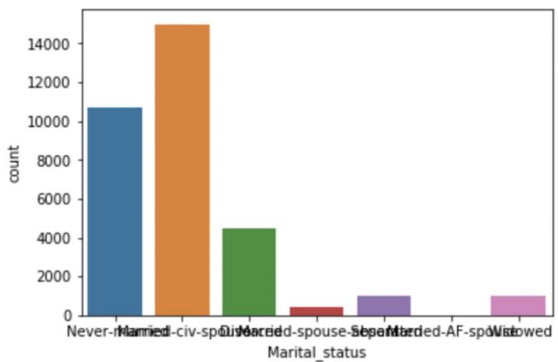


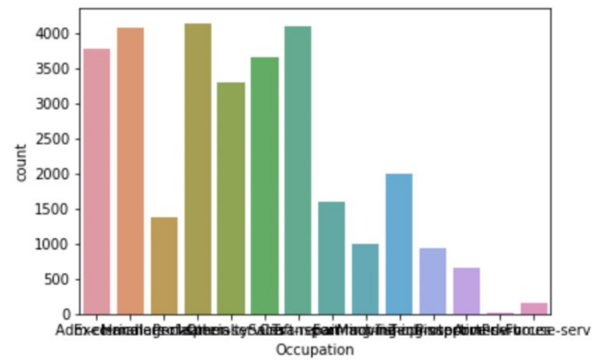


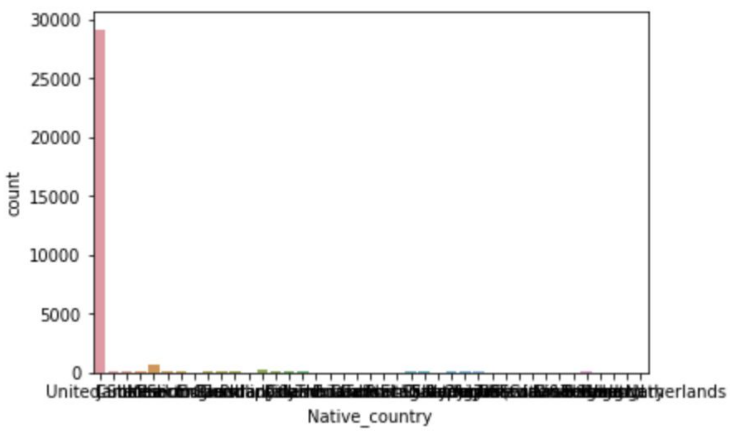












Since we want to build frequent itemset mining models, we only look at the categorical attributes at this time. We can get a general idea of frequent items by looking at the plot of countings of each attribute. For example, under native_country attribute(the plot above), ‘United States’ has a very high count value, and we can hardly see other categories' bars. Therefore, we can expect to see ‘United States’ as a subset in most of the final frequent itemsets.

Apriori Method

Functions:

1. InitialC1L1(df, minsup)
    a. Input: df as the original dataset in DataFrame format, minsup as the minimum support threshold in int format.
    b. Output: C1 list in two columns data frame format with single itemsets as the index, support counts and support as the columns; L1 list in two columns data frame format with single frequent itemsets as the index, support counts and support as the columns.
2. CreateC2L2(df, L, minsup)
    a. Input: df as the original dataset in DataFrame format; L as the prior L1 list in DataFrame format; minsup as the minimum support threshold in int format.
    b. Output: C2 list in two columns data frame format with length 2 itemsets as the index, support counts and support as the columns; L2 list in two columns data frame format with length 2 frequent itemsets as the index, support counts and support as the columns.
3. Pruning(Citem, Lsort)
    a. Input: Citem as the testing itemset in list format, Lsort as the prior sorted Lk list in order to test the itemsets.
    b. Output: True/False
4. CreateCkLk(df, L, k, minsup)
    a. Input: df as the original dataset in DataFrame format; L as the prior Lk list in DataFrame format; k as the index of the output list Ck and Lk; minsup as the minimum support threshold in int format.
    b. Output: Ck list in two columns data frame format with length k itemsets as the index, support counts and support as the columns; Lk list in two columns data frame format with length k frequent itemsets as the index, support counts and support as the columns.

Clean the original dataset with only categorical attributes, and run through the functions until Lk is empty. Then combined all L lists: L1, … Lk into one frequent itemsets list. Lk list outputs:

```
L1
```

	supcount	support
HS-grad	10501	0.322502
Never-married	10683	0.328092
Female	10771	0.330795
Husband	13193	0.405178
Married-civ-spouse	14976	0.459937
Male	21790	0.669205
Private	22696	0.697030
<=50K	24720	0.759190
White	27816	0.854274
United-States	29170	0.895857

```
L2
```

	supcount	support
(Never-married, <=50K)	10192	0.313012
(Husband, United-States)	11861	0.364270
(Husband, White)	11940	0.366696
(Husband, Married-civ-spouse)	13184	0.404902
(Husband, Male)	13192	0.405147
(Married-civ-spouse, Male)	13319	0.409048
(Married-civ-spouse, United-States)	13368	0.410553
(Married-civ-spouse, White)	13410	0.411842
(Male, Private)	14944	0.458954
(Male, <=50K)	15128	0.464605
(Private, <=50K)	17733	0.544609
(Male, White)	19174	0.588864
(Private, White)	19404	0.595928
(Male, United-States)	19488	0.598507
(Private, United-States)	20135	0.618378
(<=50K, White)	20699	0.635699
(<=50K, United-States)	21999	0.675624
(White, United-States)	25621	0.786862

```
L3
```

	supcount	support
(<=50K, Private, Male)	10707	0.328829
(White, United-States, Husband)	11053	0.339455
(Married-civ-spouse, United-States, Husband)	11852	0.363994
(United-States, Male, Husband)	11860	0.364239
(Married-civ-spouse, White, Husband)	11931	0.366420
(White, Male, Husband)	11939	0.366666
(Married-civ-spouse, United-States, Male)	11947	0.366911
(Married-civ-spouse, White, Male)	12036	0.369645
(Married-civ-spouse, White, United-States)	12369	0.379872
(<=50K, White, Male)	13085	0.401861
(Private, White, Male)	13123	0.403028
(Married-civ-spouse, Male, Husband)	13183	0.404871
(Private, United-States, Male)	13209	0.405669
(<=50K, United-States, Male)	13389	0.411197
(<=50K, White, Private)	14872	0.456743
(<=50K, United-States, Private)	15594	0.478916
(United-States, White, Male)	17653	0.542152
(Private, United-States, White)	17728	0.544455
(<=50K, United-States, White)	18917	0.580971

```
L4
```

	supcount	support
(Married-civ-spouse, United-States, Husband, White)	11044	0.339179
(United-States, Husband, White, Male)	11052	0.339424
(Married-civ-spouse, United-States, White, Male)	11125	0.341666
(Married-civ-spouse, United-States, Husband, Male)	11851	0.363963
(United-States, <=50K, White, Male)	11913	0.365867
(Married-civ-spouse, Husband, White, Male)	11930	0.366389
(United-States, Private, White, Male)	11956	0.367188
(Private, United-States, <=50K, White)	13452	0.413132

```
L5
```

	supcount	support
(Married-civ-spouse, Husband, White, Male, United-States)	11043	0.339148

L1~L5, L6 is empty.

Combined all Lk lists:

- [[' HS-grad', ' Never-married', ' Female', ' Husband', ' Married-civ-spouse', ' Male', ' Private', ' <=50K', ' White', ' United-States'],
- [(' Never-married', ' <=50K'), (' Husband', ' United-States'), (' Husband', ' White'), (' Husband', ' Married-civ-spouse'), (' Husband', ' Male'), (' Married-civ-spouse', ' Male'), (' Married-civ-spouse', ' United-States'), (' Married-civ-spouse', ' White'), (' Male', ' Private'), (' Male', ' <=50K'), (' Private', ' <=50K'), (' Male', ' White'), (' Private', ' White'), (' Male', ' United-States'), (' Private', ' United-States'), (' <=50K', ' White'), (' <=50K', ' United-States'), (' White', ' United-States')],
- [(' <=50K', ' Private', ' Male'), (' White', ' United-States', ' Husband'), (' Married-civ-spouse', ' United-States', ' Husband'), (' United-States', ' Male', ' Husband'), (' Married-civ-spouse', ' White', ' Husband'), (' White', ' Male', ' Husband'), (' Married-civ-spouse', ' United-States', ' Male'), (' Married-civ-spouse', ' White', ' Male'), (' Married-civ-spouse', ' White', ' United-States'), (' <=50K', ' White', ' Male'), (' Private', ' White', ' Male'), (' Married-civ-spouse', ' Male', ' Husband'), (' Private', ' United-States', ' Male'), (' <=50K', ' United-States', ' Male'), (' <=50K', ' White', ' Private'), (' <=50K', ' United-States', ' Private'), (' United-States', ' White', ' Male'), (' Private', ' United-States', ' White'), (' <=50K', ' United-States', ' White')],
- [(' Married-civ-spouse', ' United-States', ' Husband', ' White'), (' United-States', ' Husband', ' White', ' Male'), (' Married-civ-spouse', ' United-States', ' White', ' Male'), (' Married-civ-spouse', ' United-States', ' Husband', ' Male'), (' United-States', ' <=50K', ' White', ' Male'), (' Married-civ-spouse', ' Husband', ' White', ' Male'), (' United-States', ' Private', ' White', ' Male'), (' Private', ' United-States', ' <=50K', ' White')],
- [(' Married-civ-spouse', ' Husband', ' White', ' Male', ' United-States')]]

The running time and usage is: 10.0 secs 189.6 MByte

Improvement of the Apriori Method

In the traditional Apriori method, we need to check all possible subsets of a itemsets in Ck+1 in prior Lk to make sure it can be frequent. For example, for a length-3 protentical frequent itemset: (A, B, C). We checked (A, B), (B, C) and (A, C) in L2 to make sure all three of the subsets are frequent. In order to improve this step, we know that we get all the possible itemsets in Ck+1 from the combined two of the itemsets in Lk, so the two itemsets we used do not need to be checked. For instance, we see (A, B) and (B, C) in L2, so we combine the two as (A, B, C), and we only want to check if (A, C) is in L2.

Updated functions:

1. ImprovedPruning(Citem, Lsort, I1, I2)
    a. Input: Citem as the testing itemset in list format; Lsort as the prior sorted Lk list in order to test the itemsets; I1 and I2 as the subsets that do not need to check as list form.
    b. Output: True/False
2. ImprovedCreateCkLk(df, L, k, minsup)

   a. Input: df as the original dataset in DataFrame format; L as the prior Lk list in DataFrame format; k as the index of the output list Ck and Lk; minsup as the minimum support threshold in int format.
   b. Output: Ck list in two columns data frame format with length k itemsets as the index, support counts and support as the columns; Lk list in two columns data frame format with length k frequent itemsets as the index, support counts and support as the columns.

The running time and usage is: 10.0 secs 189.6 MByte

Some potential reasons that the time and usage are not significantly shorter with the improved algorithm are: 1. the dataset is not large which it only takes 10 seconds to run the algorithm, so some minor improvement might not appear in the time. 2. The improvement is very minor, it reduces the checking process from n*k to n*(k-2) for which n in this case is very small(<100), so it can't make significant differences.

FP-Growth Method

Functions:

1. FList(df, minsup)
   a. Input: df as the original dataset in DataFrame format; minsup as the minimum support threshold in int format.
   b. Output: Flist in two columns data frame format with single frequent itemsets as the index; support counts and support as the columns.
2. Cleandf(df, FL)
   a. Input : df as the original dataset in DataFrame format; FL as the Flist in DataFrame format.
   b. Output: df_list as a list format which each nested list with only itemset in the Flist and the order of Flist
3. FPTREE(df_list, FL)
   a. Input: df_list as a list format; FL as the Flist in DataFrame format.
   b. Output: FPtree in nested dictionary format with node as the key and the prior transactions as value and the support of each transaction as nested value.
4. ConditionalFPtrees(df, FL, FPtree, minsup)
   a. Input: df_list as a list format; FL as the Flist in DataFrame format; FPtree in nested dictionary format, minsup as the minimum support threshold in int format.
   b. Output: frequentitemsets as parts of the frequent itemsets in dictionary format with keys as the frequent itemsets and values as support; itemsets as a list of frequent itemsets that are checked.
5. lenkfrequentitemsets(itemsets, lenk, FL, FPtree, minsup)
   a. Itemsets as a list of frequent itemsets that are checked; lenk as the number of times checking the tree in int format; FL as the Flist in DataFrame format; FPtree

in nested dictionary format, minsup as the minimum support threshold in int format.

b. Output: frequentitemsets as parts of the frequent itemsets in dictionary format with keys as the frequent itemsets and values as support; itemsets as a list of frequent itemsets that are checked.

The running time and usage is: 6.9 secs 189.6 MByte

Conclusion

The result set of frequent itemsets is shown in the pictures on page 5. All three methods: Apriori Method, Improved Apriori Method and FP-Growth generate the same result. The fastest method is FP-Growth method which used 6.9 seconds in contrast with the other two methods used 10 seconds. The reason is that even though the FP-Growth algorithm seems more complex, but it only scans the original dataset twice. Whereas the other two methods scan the original dataset for every step. The time difference might not be obvious in smaller datasets, but for larger or big data, Apriori Method is way too time consuming.