31/01/2021

Όνομα Φοιτητή: Παντελεήμων Μαθιουδάκης

Μητρώο Φοιτητή: ΜΕΣ20022

Τίτλος Εργασίας: Στατιστικές Μέθοδοι Εξόρυξης Δεδομένων (Εργασία Εξαμήνου)

Καθηγητές: Ν. Πελέκης | Π. Ταμπάκης

ΠΜΣ Εφαρμοσμένης Στατιστικής

Πανεπιστήμιο Πειραιά

31 Ιανουαρίου 2021

Περιεχόμενα

1 Εργασία 1 (Preprocessing) 3

2 Εργασία 3 Μέρος Α (Outlier Detection) 7

3 Εργασία 2 (Feature Selection) 12

4 Εργασία 5 (Clustering) 12

5 Εργασίας 3 Μέρος Β (Outlier Detection) 16

6 Εργασία 4 (Classification) 20

7 Εργασία 6 (Association Rules) 23

Κατάλογος Σχημάτων

1 Διάγραμμα ανα δύο συσχετίσεων των ποσοτικών μεταβλητών . . . 4
2 Ιστογράμματα ποσοτικών μεταβλητών 6
3 Διάγραμμα ποσοτικών μεταβλητών που δύναται να κατηγοριοποιηθούν 8
4 Διάγραμμα pair-plot DBscan 14
5 Διάγραμμα pair-plot KMeans 16
6 Διάγραμμα ανα δύο μεταβλητών με KMeans υπο την επίδραση θορύβου 19

1 Εργασία 1 (Preprocessing)

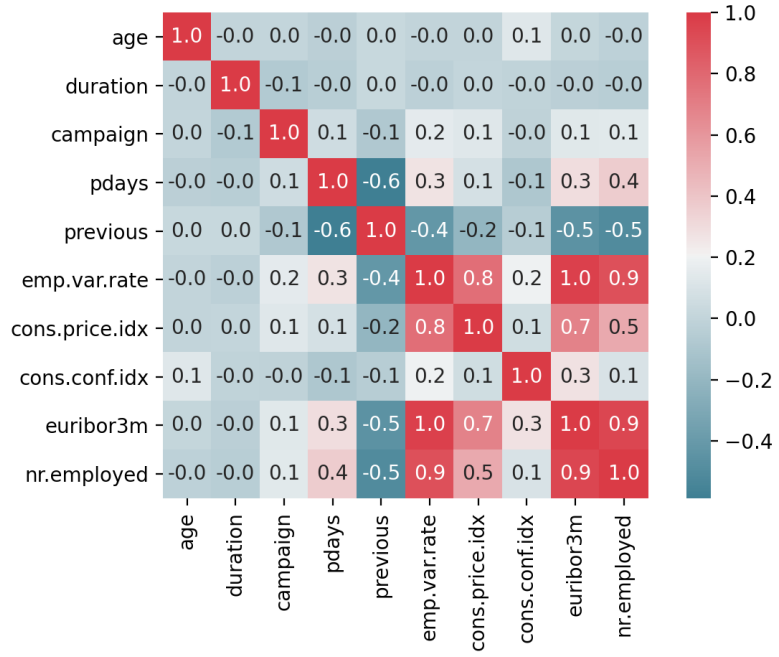
Αρχικά, εισάγουμε τα δεδομένα απο το αρχείο bank-additional-full.csv

```
>>> # PREPROCESSING
>>> import time as time
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> import numpy as np
>>> from scipy.stats import norm
>>> from sklearn.preprocessing import StandardScaler
>>> from scipy import stats
>>> import warnings

>>> df  = pd.read_csv('/home/user/Downloads/ergasia_dm/bank-a
↪  dditional/bank-additional-full.csv',sep=";")
```

Παρακάτω, κάποια βασικά στοιχεία των δεδομένων

```
>>> df.head(10)
   age          job  marital  ... euribor3m nr.employed   y
0   56    housemaid  married  ...     4.857      5191.0  no
1   57     services  married  ...     4.857      5191.0  no
2   37     services  married  ...     4.857      5191.0  no
3   40       admin.  married  ...     4.857      5191.0  no
4   56     services  married  ...     4.857      5191.0  no
5   45     services  married  ...     4.857      5191.0  no
6   59       admin.  married  ...     4.857      5191.0  no
7   41  blue-collar  married  ...     4.857      5191.0  no
8   24   technician   single  ...     4.857      5191.0  no
9   25     services   single  ...     4.857      5191.0  no

[10 rows x 21 columns]
>>> df.shape
(41188, 21)
>>> type(df)
<class 'pandas.core.frame.DataFrame'>
>>> df.isnull().sum().sum()
0

>>> nums = df.select_dtypes(include=[np.number])
>>> nums_new = nums.copy()
>>> for column in nums_new.columns :
...   nums_new.loc[:,column] = sorted(nums_new.loc[:,column])
```

```
...
>>> # CORR-PLOT
>>> f, ax = plt.subplots()
>>> corr = df.corr()
>>> sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
↪  cmap=sns.diverging_palette(220, 10, as_cmap=True),annot=True,
...            fmt='.1f',  square=True, ax=ax)
<AxesSubplot:>
>>> plt.tight_layout()
>>> f.savefig('corplot',dpi=200)
```

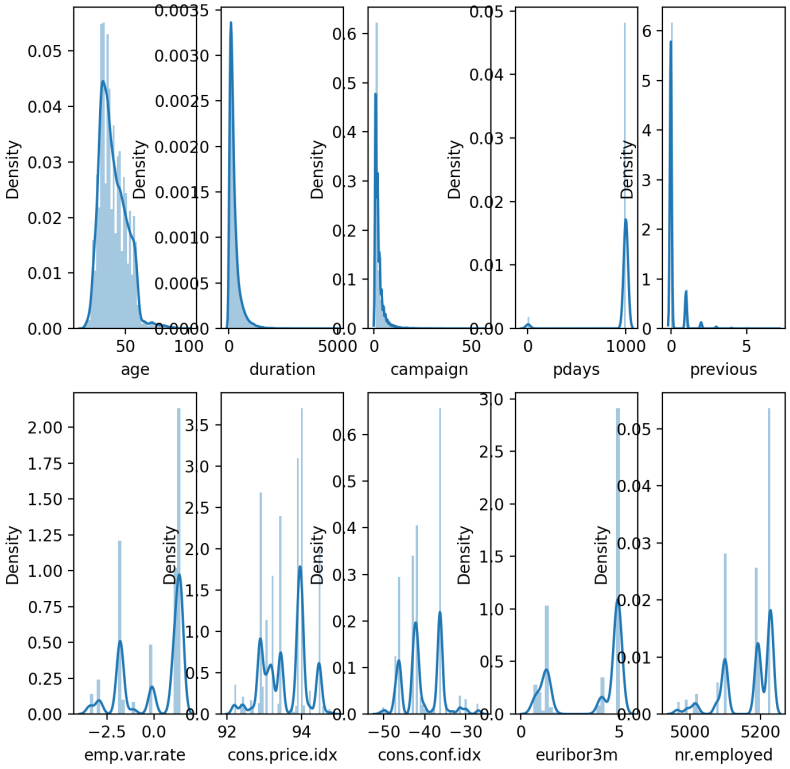


Σχήμα 1: Διάγραμμα ανα δύο συσχετίσεων των ποσοτικών μεταβλητών

Όπως φαίνεται απο το corr-plot, με κόκκινο χρώμα διακρίνονται οι υψηλά γραμμικώς συσχετισμένες μεταβλητές (euribor3m - emp.var.rate , nr.employed - emp.var.rate , nr.employed - euribor3m)
Στη συνέχεια, παρουσιάζονται Ιστογράμματα μόνο των ποσοτικών μεταβλητών των δεδομένων

```
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> import numpy as np
```

```
>>> from scipy.stats import norm
>>> from sklearn.preprocessing import StandardScaler
>>> from scipy import stats

>>> nums = df.select_dtypes(include=[np.number])
>>> tetm = [0,1,2,3,4]*2
>>> teta  = np.repeat(np.arange(0,2),5)

>>> # DENSITY-PLOT
>>> fig, axs = plt.subplots(ncols=5,nrows=2,figsize=(8,8))
>>> for i in [0,1,2,3,4,5,6,7,8,9]:
...   sns.distplot(nums.iloc[:,i],ax=axs[teta[i],tetm[i]])
...
<AxesSubplot:xlabel='age', ylabel='Density'>
<AxesSubplot:xlabel='duration', ylabel='Density'>
<AxesSubplot:xlabel='campaign', ylabel='Density'>
<AxesSubplot:xlabel='pdays', ylabel='Density'>
<AxesSubplot:xlabel='previous', ylabel='Density'>
<AxesSubplot:xlabel='emp.var.rate', ylabel='Density'>
<AxesSubplot:xlabel='cons.price.idx', ylabel='Density'>
<AxesSubplot:xlabel='cons.conf.idx', ylabel='Density'>
<AxesSubplot:xlabel='euribor3m', ylabel='Density'>
<AxesSubplot:xlabel='nr.employed', ylabel='Density'>
>>> fig.savefig('plot_nums.png',dpi=200)
```



Σχήμα 2: Ιστογράμματα ποσοτικών μεταβλητών

Διακρίνεται πως αρκετές απο τις ποσοτικές μεταβλητές παρουσιάζουν κάποιες ασυμμετρίες, όπως οι Age, Previous, Pdays, Duration(στην οποία τιμές $= 0$ δεν έχουν σημασία),emp.var.rate κτλ.Μία εύλογη αντιμετώπιση του θέματος θα ήταν να δημιουργήσουμε ένα καινούριο σετ ποσοτικών δεδομένων το οποίο θα αποτελείται απο περικομμένες μεταβλητές. Δηλαδή απο κάθε μεταβλητή θα κόψουμε ένα ποσοστό 4% του συνόλου απο την κάτω και άνω ουρά κάθε κατανομής.
Συνεπώς, αν και βρίσκεται η διαδικασία στο στάδιο preprocessing, με αυτο λύνεται και το θέμα των outliers

Στη συνέχεια, για την καλύτερη μελέτη των χαρακτηριστικών των δεδομένων, διαχωρίζονται οι μεταβλητές σε ποσοτικές και κατηγορικές

```
>>> # DIAXWRISMOS NUM KAI CATEGORICAL DATA
>>> df_num =df.select_dtypes(include=['number','float64'])
>>> df_cat = df.select_dtypes(include=[object])

>>> dfc = df.copy()
>>> dfc_num =dfc.select_dtypes(include=['number','float64'])
>>> dfc_cat = dfc.select_dtypes(include=[object])
```

Όπως προαναφέρθηκε, μόνο για τα συνεχή δεδομένα, αφαιρείται ένα μικρό ποσοστό παρατηρήσεων απο την κάτω και άνω ουρά της κατανομής της κάθε μεταβλητής, δίνοντας έτσι την δυνατότητα οι παρακάτω μέθοδοι να μην επηρεαστούν απο τυχόν ακραίες τιμές.

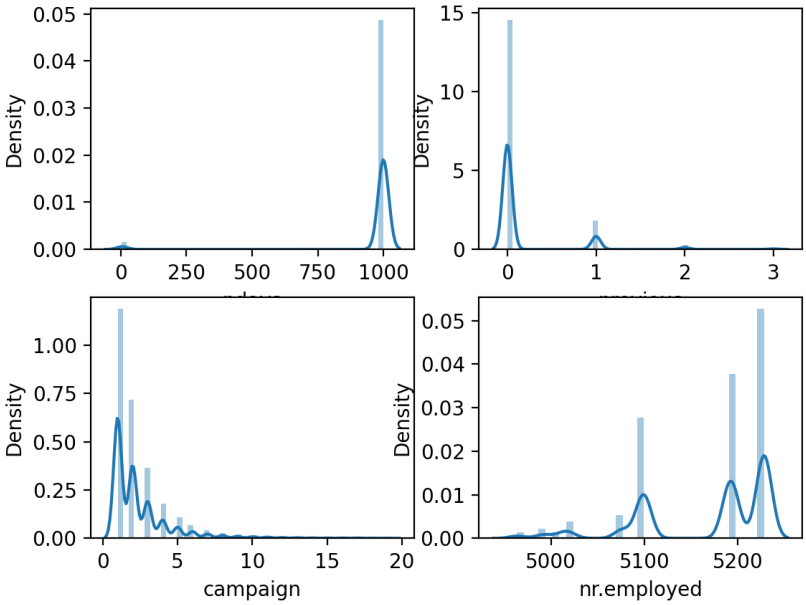
2 Εργασία 3 Μέρος Α (Outlier Detection)

```
>>> # OUTLIERS basei krithriou : <200 kai >40988 apo #diatetagm. times
>>> dfc_num.shape[0]*1/100
411.88
>>> sorted(dfc_num['age'])[1030]
24

>>> place = []
>>> for col in dfc_num.columns :
...       k = sorted(dfc_num[col])[200]
...       a = sorted(dfc_num[col])[40988]
...       matrix = list(np.where((dfc_num[col]<k)|(dfc_num[col]>a))[0])
...       place.extend(matrix)
...
>>> print(place[0:9],len(place)/dfc_num.shape[0])
[1353, 1552, 1641, 6575, 10939, 12911, 13407, 15512, 15634] 0.04129843643779742

>>> print('Percentage of Values reduced = ',len(place)/dfc_num.shape[0])
Percentage of Values reduced =  0.04129843643779742

>>> dfc_cut = df_num.drop(labels=list(set(place)),axis=0)
>>> dfc_cat = df_cat.drop(labels=list(set(place)),axis=0)
>>> data_prim = dfc.drop(labels=list(set(place)),axis=0)
>>> y = dfc_cat['y']
>>> y.reset_index(drop=True,inplace=True)
>>> dfc_cat = dfc_cat.drop(columns='y')
>>> dfc_cat_tr = dfc_cat.copy()

>>> data_num = dfc_cut.copy()
>>> data_cat = dfc_cat.copy()
>>> # Problematic Values
```

```
>>> np.sum(data_num.duration==0)
0
```

Ερευνώντας τα δεδομένα, μπορεί να ανιχνευθεί πως κάποιες απο τις ποσοτικές μεταβλητές δύναται να τροποποιηθούν σε κατηγορικές είτε λόγω κάποιων ευδιάκριτων ασυμμετριών ή επειδή ορίζονται σε διακριτή κλίμακα.
Παρακάτω παρουσιάζεται γράφημα των χαρακτηριστικών με τέτοιες ιδιοτροπίες

```
>>> fig, axs = plt.subplots(ncols=2,nrows=2)
>>> sns.distplot(data_num.pdays,ax=axs[0,0])
<AxesSubplot:xlabel='pdays', ylabel='Density'>
>>> sns.distplot(data_num.previous,ax=axs[0,1])
<AxesSubplot:xlabel='previous', ylabel='Density'>
>>> sns.distplot(data_num.campaign,ax=axs[1,0])
<AxesSubplot:xlabel='campaign', ylabel='Density'>
>>> sns.distplot(data_num['nr.employed'],ax=axs[1,1])
<AxesSubplot:xlabel='nr.employed', ylabel='Density'>
>>> fig.savefig('num_to_cat_plot.png',dpi=200)
```



Σχήμα 3: Διάγραμμα ποσοτικών μεταβλητών που δύναται να κατηγοριοποιηθούν

```
>>> # KWDIKOPOIHSH PDAYS
>>> pdays_coded =
↪   pd.DataFrame(np.where(data_num['pdays']<100,0,1))
```

```
>>> data_cat.reset_index(drop=True,inplace=True)
>>> data_num.reset_index(drop=True,inplace=True)
>>> data_cat = pd.concat([data_cat,pdays_coded],axis=1)
>>> data_cat.rename(columns={0:'pdays_coded'},inplace=True)
>>> data_num = data_num.drop(columns='pdays',axis=1)

>>> # KWDIKOPOIHSH PREVIOUS
>>> dfc_cut['previous'].astype('category')
0        0
1        0
2        0
3        0
4        0
        ..
41055    1
41056    1
41057    1
41058    3
41059    0
Name: previous, Length: 39687, dtype: category
Categories (4, int64): [0, 1, 2, 3]
>>> type(dfc_cut['previous'])
<class 'pandas.core.series.Series'>
>>> previous = dfc_cut['previous']
>>> previous.reset_index(drop=True,inplace=True)
>>> data_cat = pd.concat([data_cat,previous],axis=1)
>>> data_cat.rename(columns={0:'previous_coded'},inplace=True)
>>> data_num = data_num.drop(columns='previous',axis=1)
>>> data_num.columns
Index(['age', 'duration', 'campaign', 'emp.var.rate',
↪  'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed'],
      dtype='object')

>>>   # KWDIKOPOIHSH CAMPAIGN
>>> set(dfc_cut['campaign'].values)
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
↪  19}
>>> dfc_cut['campaign'].describe()
count    39687.000000
mean         2.475420
std          2.269954
min          1.000000
25%          1.000000
50%          2.000000
75%          3.000000
```

```
max         19.000000
Name: campaign, dtype: float64
>>> campaign_coded = np.where(dfc_cut['campaign']<=3,0,1)
>>> campaign_coded = pd.DataFrame(campaign_coded)
>>> data_cat = pd.concat([data_cat,campaign_coded],axis=1)
>>> data_cat.rename(columns={0:'campaign_coded'},inplace=True)

>>> data_num = data_num.drop(columns='campaign',axis=1)
>>> data_num.columns
Index(['age', 'duration', 'emp.var.rate', 'cons.price.idx',
↪  'cons.conf.idx',
       'euribor3m', 'nr.employed'],
      dtype='object')

>>> # KWDIKOPOIHSH NR.EMPLOYED
>>> nrempl_coded =  pd.cut(data_num['nr.employed'],bins=[
↪  0,5050,5150,6000],labels=[0,1,2])
>>> data_cat = pd.concat([data_cat,nrempl_coded],axis=1)
>>> data_cat.rename(columns={0:'nrempl_coded'},inplace=True)
>>> data_num = data_num.drop(columns='nr.employed',axis=1)
```

Το αποτέλεσμα, μια καινούρια μορφή πινάκων με χαρακτηριστικά :

Πίνακες	Κατηγορικές	Ποσοτικές
Πρίν	(41188, 11)	(41188, 10)
Μετά	(39687, 14)	(39687, 6)

Ακολουθεί τυποποίηση των ποσοτικών μεταβλητών

```
>>> # KANONIKOPOIHSH
>>> from sklearn.preprocessing import StandardScaler
>>> std = StandardScaler()
>>> cl = data_num.columns
>>> data_num = std.fit_transform(data_num)
>>> data_num = pd.DataFrame(data_num)
>>> data_num.columns = cl
>>> data_num.head()
        age  duration  emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m
0  1.630291  0.045419      0.633013        0.731246       0.903817   0.696434
1  1.731194 -0.446433      0.633013        0.731246       0.903817   0.696434
2 -0.286870 -0.108285      0.633013        0.731246       0.903817   0.696434
3  0.015840 -0.437650      0.633013        0.731246       0.903817   0.696434
4  1.630291  0.247430      0.633013        0.731246       0.903817   0.696434
```

Για τις κατηγορικές μεταβλητές, αυτές κωδικοποιούνται ως προς ακέραιους αριθμούς για να γίνει δυνατή η επεξεργασία τους με μεθόδους επιλογής χαρακτηριστικών.

```
>>> # LABEL_ENCODER
>>> from sklearn.preprocessing import LabelEncoder
>>> l_e = LabelEncoder()
>>> data_cat1 = data_cat.copy()

>>> for col in data_cat1.columns :
...     data_cat1[col] = l_e.fit_transform(data_cat1[col])
...
>>> data_cat1.reset_index(drop=True,inplace=True)
```

Εν συνεχεία, εκτελείται αλγόριθμος μείωσης διαστάσεων ποσοτικών δεδομένων (PCA)

```
>>> # PRINCIPAL COMP ANALYSIS
>>> from mpl_toolkits.mplot3d import Axes3D
>>> from sklearn import decomposition
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.impute import SimpleImputer

>>> X = pd.DataFrame(data_num).to_numpy()

>>> pca = decomposition.PCA(n_components='mle',svd_solver='full')
>>> pca.fit(X)
PCA(n_components='mle', svd_solver='full')
>>> X = pca.transform(X)

>>> data_pca = X.copy()
```

Ενα ακόμη χαρακτηριστικό της προπαρασκευής δεδομένων μπορεί να θεωρηθεί και το feature selection βάσει του οποίου επιλέγονται μόνο οι πιο επεξηγηματικές κατηγορικές μεταβλητές για την μεταβλητή στόχο y. Συνήθως χρησιμοποιείται ο μετρικός Gini ως μέτρο εντροπίας. Γενικά επιθυμούμε οι κατηγορίες της μεταβλητής να οδηγούν σε έναν καλό διαχωρισμό, δηλαδή σε κάθε κατηγορία να είμαστε σχεδόν σίγουροι αν η y είναι yes ή no.

3 Εργασία 2 (Feature Selection)

```
>>> # FEATURE SELECTION TREES
>>> from mpl_toolkits.mplot3d import Axes3D
>>> from sklearn.ensemble import ExtraTreesClassifier
>>> from sklearn.feature_selection import SelectFromModel
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.impute import SimpleImputer

>>> y = y.str.get_dummies().iloc[:,1]
>>> y = np.array(y)
>>> X = np.array(data_cat1)
>>> classf = ExtraTreesClassifier()
>>> classf = classf.fit(X,y)
>>> importances = classf.feature_importances_
>>> fs_modeltr = SelectFromModel(classf, prefit=True).transform(X)

>>> print(fs_modeltr[0])
[3 0 6 1 2]
>>> print(X[0])
[3 1 0 0 0 0 1 6 1 1 1 0 0 2]
>>> print(df.head(1))
   age        job  marital education  ... cons.conf.idx euribor3m nr.employed   y
0   56  housemaid  married  basic.4y  ...         -36.4     4.857      5191.0  no

[1 rows x 21 columns]
>>> print(fs_modeltr.shape)
(39687, 5)
```

Βέλτιστες κατηγορικές μεταβλητές που επέλεξε η διαδικασία Feature Selection Trees : job, education, month, day_of_week, nr.employed

4 Εργασία 5 (Clustering)

Παρακάτω, επιλέγονται τα ποσοτικά δεδομένα και δοκιμάζονται διάφοροι αλγόριθμοι συσταδοποίησης για την ανίχνευση τυχόν ομάδων των δεδομένων. Συγκεκριμένα οι διεργασίες θα εκτελεστούν στο υποσύνολο των δεδομένων μόνο των ποσοτικών μεταβλητών, που περιέχονται στον πίνακα data_pca

```
>>> # DBSCAN

>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
```

```
>>> from sklearn.cluster import DBSCAN
>>> from sklearn import metrics
>>> from sklearn.datasets import make_blobs
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.impute import SimpleImputer

>>> xclust = data_pca

>>> tdb1 = time.time()
>>> db = DBSCAN(eps=0.97, min_samples=5, algorithm='kd_tree',
↪  n_jobs=8).fit(xclust)
>>> core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
>>> tdb2 = time.time()

>>> core_samples_mask[db.core_sample_indices_] = True
>>> labels = db.labels_

>>> n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
>>> n_noise_ = list(labels).count(-1)

>>> print('Time Cost of DBscan = ',tdb2-tdb1)
Time Cost of DBscan =  11.288012742996216
>>> print('Estimated number of clusters: %d' % n_clusters_)
Estimated number of clusters: 9
>>> print('Estimated number of noise points: %d' % n_noise_)
Estimated number of noise points: 30
>>> print("Silhouette Coefficient: %0.3f"
...       % metrics.silhouette_score(xclust, labels))
Silhouette Coefficient: 0.241
```

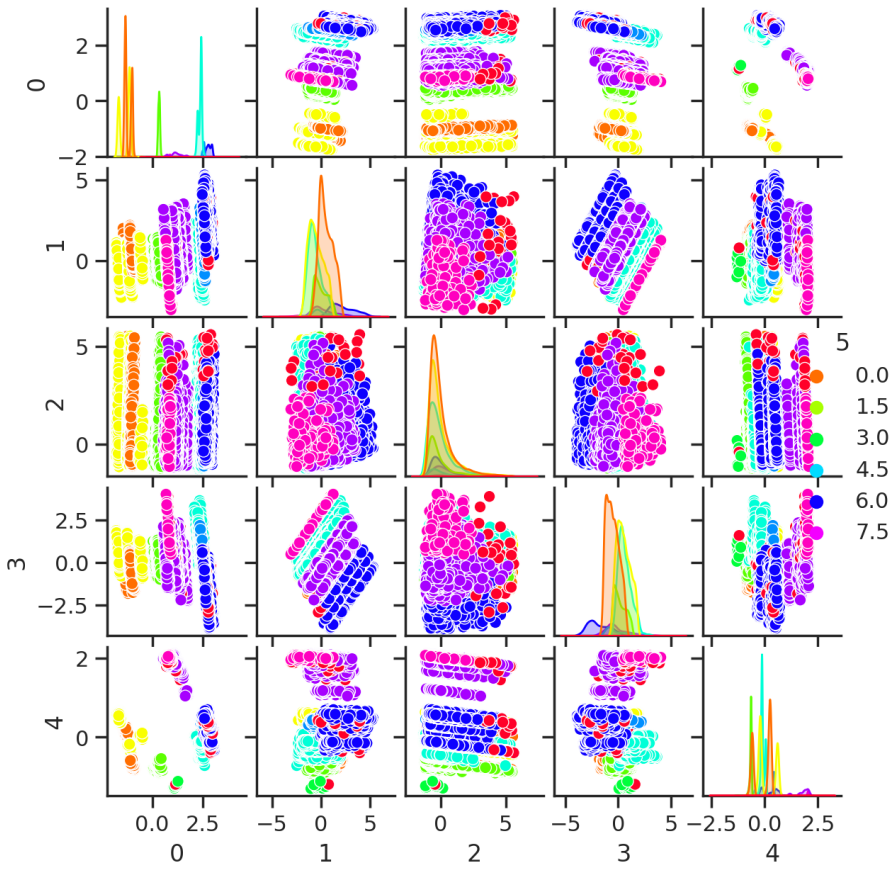
Ο πρώτος αλγόριθμος που εκτελέστηκε ήταν ο DBscan.Παρακάτω κάποια αριθμητικά αποτελέσματα για να παρουσιαστεί η αποτελεσματικότητά του.

Χρονικό Κόστος	11.288012742996216
Εκτιμώμενος Αριθμός Συστάδων	9
Εκτιμώμενος Αριθμός Θορύβου	30
Συντελεστής Προσαρμογής	0.24107647674435023

Εν συνεχεία, για την καλύτερη κατανόηση του αποτελέσματος του DBscan,ένα διάγραμμα pair-plot θα μπορούσε να βοηθήσει στην γεωμετρική εποπτεία της διαδικασίας

```
>>> lab = np.reshape(db.labels_,[39687,1])
>>> dat =  np.concatenate([xclust,lab],axis=1)
```

```
>>> dat = pd.DataFrame(dat)
>>> dat.columns = ['0','1','2','3','4','5']

>>> ####### DIAGRAMMATA ANA 2 ME LABELS TWN CLUSTERS ######
>>> sns.set(style='ticks',color_codes=True)
>>> pair = sns.pairplot(dat,vars=['0','1','2','3','4'],hue='5'
... ,palette='gist_rainbow')
>>> pair.fig.set_size_inches(6,6)
>>> pair.savefig('DBscan_plot.png',dpi=200)
```



Σχήμα 4: Διάγραμμα pair-plot DBscan

Διακρίνεται σχετικά καλή συσταδοποίηση κυρίως στις διαστάσεις $[x, y] = [1, 3], [2, 3]$

Συνεχίζωντας με διαδικασία KMeans, παρατίθονται τα αντίστοιχα αποτελέσματα της διαδικασίας

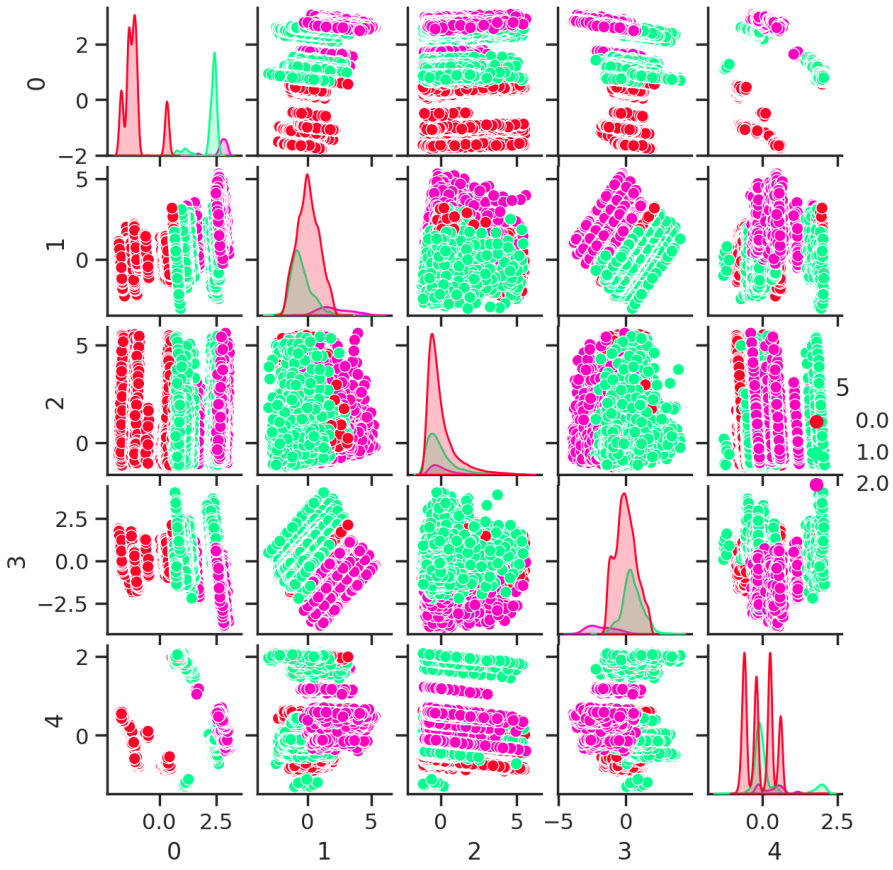
```
>>> # KMEANS
>>> from sklearn.cluster import KMeans
>>> from sklearn.cluster import MiniBatchKMeans

>>> from sklearn import metrics
>>> from sklearn.datasets import make_blobs
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.impute import SimpleImputer

>>> xclust = data_pca

>>> tkm1 = time.time()
>>> db = KMeans(n_clusters=3, max_iter=1000).fit(xclust)
>>> tkm2 = time.time()

>>> centroids = db.cluster_centers_
>>> labels = db.labels_

>>> n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
>>> n_noise_ = list(labels).count(-1)

>>> print('Time Cost of Kmeans = ',tkm2-tkm1)
Time Cost of Kmeans =  0.5592219829559326
>>> print('Number of iterations: %d' % db.n_iter_)
Number of iterations: 5
>>> print('Estimated number of clusters: %d' % n_clusters_)
Estimated number of clusters: 3
>>> print('Estimated number of noise points: %d' % n_noise_)
Estimated number of noise points: 0
>>> print("Silhouette Coefficient: %0.3f" %
↪  metrics.silhouette_score(xclust, labels))
Silhouette Coefficient: 0.454
>>> print("Mean Squared Error: %0.3f" % db.inertia_)
Mean Squared Error: 117183.814
```

Τα αποτελέσματα του αλγορίθμου Συσταδοποίησης Kmeans :

Χρονικό Κόστος	0.5592219829559326
Εκτιμώμενος Αριθμός Συστάδων	3
Εκτιμώμενος Αριθμός Θορύβου	0
Συντελεστής Προσαρμογής	0.4538666089852375

```
>>> lab = np.reshape(db.labels_,[39687,1])
>>> dat =  np.concatenate([xclust,lab],axis=1)
>>> dat = pd.DataFrame(dat)
```

```
>>> dat.columns = ['0','1','2','3','4','5']

>>> ####### DIAGRAMMATA ANA 2 ME LABELS TWN CLUSTERS ######
>>> sns.set(style='ticks',color_codes=True)
>>> pair = sns.pairplot(dat,vars=['0','1','2','3','4'],hue='5'
...  ,palette='gist_rainbow')
>>> pair.fig.set_size_inches(6,6)
>>> pair.savefig('KMeans_plot.png',dpi=200)
```



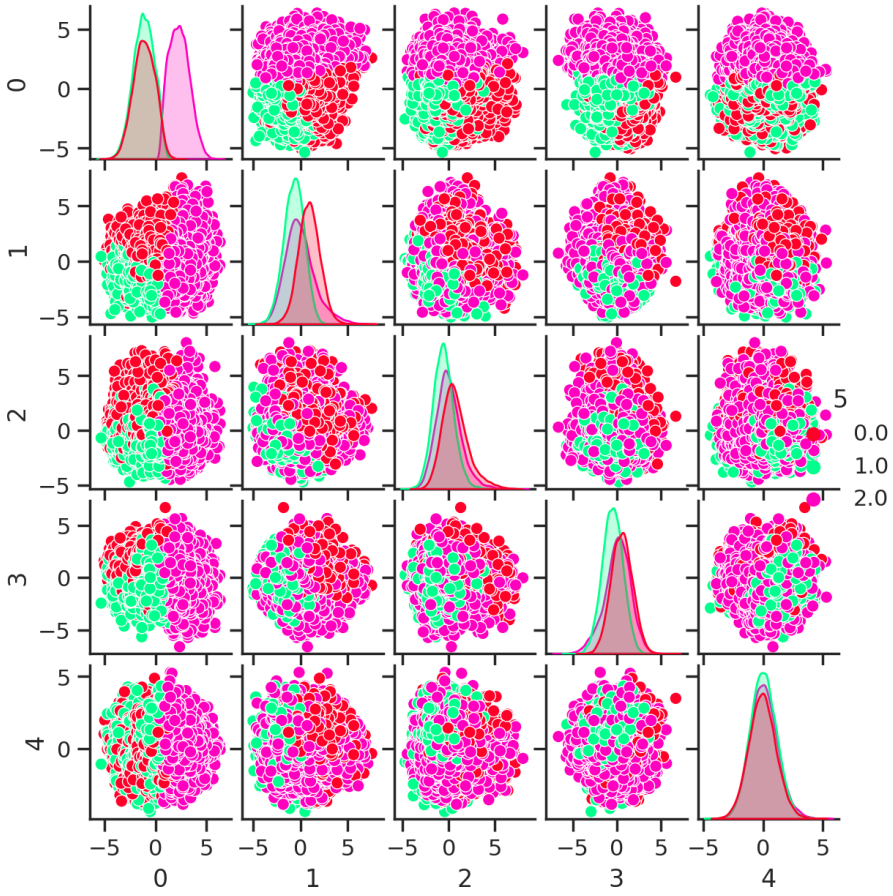
Σχήμα 5: Διάγραμμα pair-plot KMeans

5 Εργασίας 3 Μέρος Β (Outlier Detection)

Σε αυτό το σημείο θα μποτρούσε να εξεταστεί η ικανότητα των μεθόδων στην συσταδοποίηση δεδομένων με ενσωματωμένο θόρυβο.
Γι αυτο το λόγο, στο σύνολο δεδομένων θα προστεθεί θόρυβος προερχόμενος απο την Normal(0,1) Κατανομή με σκοπό την διερεύνηση της αντοχής (συγκεκριμένα του KMeans) των κανόνων Συσταδοποίησης.

```
>>> ##################### noise ######################
>>> import random
>>> noise = np.reshape(np.random.normal(0,1,xclust.shape[0]*xclu
↪   st.shape[1]),[xclust.shape[0],xclust.shape[1]])

>>> tkm_noise1 = time.time()
>>> db = KMeans(n_clusters=3, max_iter=1000).fit(noise+xclust)
>>> tkm_noise2 = time.time()

>>> centroids = db.cluster_centers_
>>> labels = db.labels_

>>> n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
>>> n_noise_ = list(labels).count(-1)

>>> print('Time Cost of Kmeans = ',tkm_noise2-tkm_noise1)
Time Cost of Kmeans =  0.9884028434753418
>>> print('Number of iterations: %d' % db.n_iter_)
Number of iterations: 14
>>> print('Estimated number of clusters: %d' % n_clusters_)
Estimated number of clusters: 3
>>> print('Estimated number of noise points: %d' % n_noise_)
Estimated number of noise points: 0
>>> print("Silhouette Coefficient: %0.3f" %
↪   metrics.silhouette_score(noise+xclust, labels))
Silhouette Coefficient: 0.165

>>> ###
>>> #noisedf = pd.DataFrame(noise+xclust)
>>> #pair_noise = sns.pairplot(noisedf,vars=[0,1,2,3,4])
>>> #pair_noise.fig.set_size_inches(5,5)
>>> #plt.show()
>>> ###
>>> lab = np.reshape(db.labels_,[39687,1])
>>> np.concatenate([xclust+noise,lab],axis=1)
array([[-1.64265014e+00,  1.08118927e+00, -1.93416870e+00,
         3.07654848e-01, -1.36980970e-01,  1.00000000e+00],
       [-4.45903646e-01,  1.12941104e+00, -4.21801328e-01,
         1.79420313e+00, -8.75279875e-01,  0.00000000e+00],
```

```
       [-1.82087080e+00, -2.66432892e-01, -1.92598738e+00,
        -1.35909861e-01, -3.60519356e-01,  1.00000000e+00],
       ...,
       [-5.20652864e-01, -2.65694734e+00, -1.36395895e+00,
        -3.28999062e-03,  1.29815856e+00,  1.00000000e+00],
       [ 2.51566366e-01, -1.26713332e+00,  2.27694071e+00,
         3.56286006e+00,  1.11626027e+00,  0.00000000e+00],
       [-3.92531632e-01, -3.07513138e+00, -2.12860982e+00,
         3.08995505e+00,  2.80076294e+00,  1.00000000e+00]])
>>> dat =  np.concatenate([xclust+noise,lab],axis=1)
>>> dat = pd.DataFrame(dat)
>>> dat.columns = ['0','1','2','3','4','5']
>>> sns.set(style='ticks',color_codes=True)

>>> pair_n = sns.pairplot(dat,vars=['0','1','2','3','4'],hue='5'
↪  ,palette='gist_rainbow')
>>> pair_n.fig.set_size_inches(6,6)
>>> pair_n.savefig('noise_plot.png',dpi=200)
```



Σχήμα 6: Διάγραμμα ανα δύο μεταβλητών με KMeans υπο την επίδραση θορύβου

Όπως φαίνεται κι απο το διάγραμμα, η προσθήκη θορύβου διατάραξε την αρχική καλή εικόνα της συσταδοποίησης, δημιουργώντας περίεργα και αλληλοκαλυπτόμενα clusters.
Επιπρόσθετα, μειώθηκε και το silouhette coef.

Στον επόμενο τομέα ανάλυσης, διερευνώνται μέθοδοι κατηγοριοποήσης classification ως κομμάτι πλέον της εποπτευόμενης μάθησης
Θα χρησιμοποιηθεί το σύνολο δεδομένων των κατηγορικών μεταβλητών

6 Εργασία 4 (Classification)

```
>>>
↪  ##################################################################
>>>                         # CLASSIFICATION #
>>>
↪  ##################################################################

>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.impute import SimpleImputer
>>> from sklearn.naive_bayes import GaussianNB
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.svm import LinearSVC
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> from sklearn.metrics import confusion_matrix
>>> from sklearn.metrics import precision_recall_fscore_support
>>> from numpy import asarray

>>> X = fs_modeltr.copy()
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
↪  test_size=0.4)

>>> clf = GaussianNB()
>>> #clf = LogisticRegression()
>>> #clf = LinearSVC(C=1.0, max_iter=4000)
>>> #clf = RandomForestClassifier()
>>> #clf = DecisionTreeClassifier(random_state=1234)

>>> clf.fit(X_train, y_train)
GaussianNB()
>>> y_pred = clf.predict(X_test)

>>> cm = confusion_matrix(y_test, y_pred)
>>> print(cm)
[[13740   506]
 [ 1149   480]]

>>> print('Accuracy = ' + str(clf.score(X_test,y_test)))
Accuracy = 0.895748031496063
```

Ακρίβεια	0.895748031496063

Πίνακας Σύγχυσης :

```
array([[13740,   506],
       [ 1149,   480]])
```

Διακρίνεται πολύ καλή ακρίβεια του αλγορίθμου (περίπου 0.9), καθως και απο τον πίνακα σύγχυσης φαίνεται αρκετά επιτυχής κατηγοριοποίηση

Για την Λογιστική Παλινδρόμηση

```
>>> X = pd.concat([data_num,data_cat1],axis=1)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
↪  test_size=0.4)

>>> #clf = GaussianNB()
>>> clf = LogisticRegression(max_iter=4000)
>>> #clf = LinearSVC(C=1.0, max_iter=4000)
>>> #clf = RandomForestClassifier()
>>> #clf = DecisionTreeClassifier(random_state=1234)

>>> clf.fit(X_train, y_train)
LogisticRegression(max_iter=4000)
>>> y_pred = clf.predict(X_test)

>>> cm = confusion_matrix(y_test, y_pred)
>>> print(cm)
[[13861   353]
 [  998   663]]

>>> print('Accuracy = ' + str(clf.score(X_test,y_test)))
Accuracy = 0.9148976377952756

>>> ### STATISTIKA LOGIT ###
>>> from sklearn import datasets, linear_model
>>> from sklearn.linear_model import LinearRegression
>>> import statsmodels.api as sm
>>> from scipy import stats

>>> X2 = sm.add_constant(X)
>>> est = sm.Logit(y, X2)
>>> est2 = est.fit()
Optimization terminated successfully.
         Current function value: 0.197326
         Iterations 8
```

est2.summary())

Logit Regression Results

Dep. Variable:	y	No. Observations:	39687
Model:	Logit	Df Residuals:	39666
Method:	MLE	Df Model:	20
Date:	Sun, 31 Jan 2021	Pseudo R-squ.:	0.4073
Time:	00:40:40	Log-Likelihood:	-7831.3
converged:	True	LL-Null:	-13214.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.1242	0.374	-3.008	0.003	-1.857	-0.392
age	0.0378	0.021	1.776	0.076	-0.004	0.080
duration	1.1945	0.019	64.531	0.000	1.158	1.231
emp.var.rate	-1.6578	0.114	-14.555	0.000	-1.881	-1.435
cons.price.idx	0.6304	0.055	11.515	0.000	0.523	0.738
cons.conf.idx	0.1597	0.031	5.203	0.000	0.100	0.220
euribor3m	0.7059	0.148	4.768	0.000	0.416	0.996
job	0.0072	0.006	1.222	0.222	-0.004	0.019
marital	0.1150	0.038	3.006	0.003	0.040	0.190
education	0.0634	0.011	5.983	0.000	0.043	0.084
default	-0.4308	0.069	-6.243	0.000	-0.566	-0.296
housing	-0.0068	0.022	-0.315	0.753	-0.049	0.036
loan	-0.0401	0.030	-1.346	0.178	-0.099	0.018
contact	-0.7173	0.071	-10.083	0.000	-0.857	-0.578
month	-0.0950	0.009	-10.559	0.000	-0.113	-0.077
day_of_week	0.0484	0.015	3.150	0.002	0.018	0.079
poutcome	0.4735	0.098	4.855	0.000	0.282	0.665
pdays_coded	-1.0300	0.202	-5.101	0.000	-1.426	-0.634
previous	-0.0145	0.081	-0.179	0.858	-0.173	0.144
campaign_coded	-0.0810	0.064	-1.274	0.202	-0.206	0.044
nr.employed	-0.7930	0.134	-5.914	0.000	-1.056	-0.530

Το Λογιστικό μοντέλο :

$$logit(\pi_i) = X \cdot B + \epsilon \ , \quad logit(\pi_i) = log(\frac{\pi_i}{1-\pi_i})$$

όπου $\pi_i = P(y_i = 1)$ πίνακας X = τα δεδομένα, B = πίνακας συντελεστών (βλέπε coef), ϵ = τα κατάλοιπα
φαίνεται να έχει αρκετά καλή προσαρμογή με ακρίβεια $\simeq 0.91$, ενώ απο τον στατιστικό πίνακα, σχεδόν όλες οι μεταβλητές είναι στατιστικά σημαντικές ($(P \geq |t|) \leq 0.05$)

7 Εργασία 6 (Association Rules)

```
>>> ############################################################
>>>                     #ASSOCIATION_RULES#
>>> ############################################################

>>> from mlxtend.preprocessing import TransactionEncoder
>>> from mlxtend.frequent_patterns import apriori
>>> from mlxtend.frequent_patterns import association_rules
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.impute import SimpleImputer
>>> from sklearn.preprocessing import LabelEncoder

>>> ydf = pd.DataFrame(y)
>>> repl = {0:'Not_Open_Account',1:'Open_Account'}
>>> ydf = ydf.replace(repl)

>>> dfc_cat_tr.reset_index(drop=True,inplace=True)
>>> X = pd.concat([dfc_cat_tr,ydf],axis=1)
>>> X.reset_index(drop=True,inplace=True)
>>> X.rename(columns={0:'y'},inplace=True)

>>> X = X.values.tolist()
>>> te = TransactionEncoder()
>>> te_ary = te.fit(X).transform(X)

>>> dfar = pd.DataFrame(te_ary, columns=te.columns_)
>>> frequent_itemsets = apriori(dfar, min_support=0.5,
... use_colnames=True)
>>> print(frequent_itemsets)
     support                              itemsets
0   0.896389                   (Not_Open_Account)
1   0.630383                           (cellular)
2   0.608083                            (married)
3   0.976491                                (no)
4   0.871242                        (nonexistent)
5   0.584322                                (yes)
6   0.544158         (Not_Open_Account, cellular)
7   0.550810          (Not_Open_Account, married)
8   0.873913               (no, Not_Open_Account)
9   0.798901      (nonexistent, Not_Open_Account)
10  0.522665              (yes, Not_Open_Account)
```

```
11  0.618338                        (no, cellular)
12  0.510394                (nonexistent, cellular)
13  0.591327                          (no, married)
14  0.536019                 (nonexistent, married)
15  0.849145                      (nonexistent, no)
16  0.566458                               (no, yes)
17  0.505304                     (nonexistent, yes)
18  0.532769      (no, Not_Open_Account, cellular)
19  0.534759       (no, Not_Open_Account, married)
20  0.777736  (nonexistent, no, Not_Open_Account)
21  0.505581           (yes, no, Not_Open_Account)
22  0.520170            (nonexistent, no, married)

>>> rules = association_rules(frequent_itemsets,
↪   metric="confidence", min_threshold=0.7)

>>> print(rules)
                        antecedents  ... conviction
0                          (cellular)  ...   0.757491
1                           (married)  ...   1.100061
2                                (no)  ...   0.986326
3                 (Not_Open_Account)  ...   0.937591
4                      (nonexistent)  ...   1.247840
5                 (Not_Open_Account)  ...   1.183910
6                               (yes)  ...   0.981910
7                          (cellular)  ...   1.230433
8                          (cellular)  ...   0.676450
9                           (married)  ...   0.853145
10                          (married)  ...   1.086470
11                     (nonexistent)  ...   0.926875
12                               (no)  ...   0.987311
13                              (yes)  ...   0.768932
14                              (yes)  ...   0.952132
15                    (no, cellular)  ...   0.748707
16     (Not_Open_Account, cellular)  ...   1.123229
17                         (cellular)  ...   0.814258
18                    (no, married)  ...   1.083090
19      (Not_Open_Account, married)  ...   0.806760
20                          (married)  ...   1.045652
21                 (nonexistent, no)  ...   1.232069
22  (nonexistent, Not_Open_Account)  ...   0.887351
23           (no, Not_Open_Account)  ...   1.169950
24                     (nonexistent)  ...   1.174804
25                               (no)  ...   0.988004
26                (Not_Open_Account)  ...   1.139665
27                          (no, yes)  ...   0.964103
```

```
28          (Not_Open_Account, yes)  ...   0.719242
29                            (yes)  ...   0.935664
30           (nonexistent, married)  ...   0.795081
31                    (no, married)  ...   1.070001
32                        (married)  ...   1.043449

[33 rows x 9 columns]
```

Εν κατακλείδι, ο τελευταίος πίνακας φανερώνει τους κανόνες συσχέτισης στα στοιχεία με ελάχιστη πιθανότητα εμφάνισης min_support = 0.5 κατώφλι min_threshold = 0.7

Antecedents(X)	Consequents(Y)	Confidence $P(Y\|X)$
cellular	Not_Open_Account	0.863
married	Not_Open_Account	0.906
Not_Open_Account	no_default	0.775
no_default	Not_Open_Account	0.881
Not_Open_Account	no_loan	0.823
no_loan	Not_Open_Account	0.895
Not_Open_Account	nonexistent	0.891
nonexistent	Not_Open_Account	0.917
cellular	no_default	0.831
cellular	no_loan	0.823
cellular	nonexistent	0.810
married	no_loan	0.824
married	nonexistent	0.881
no_loan	no_default	0.788
no_default	no_loan	0.824
nonexistent	no_default	0.772
no_default	nonexistent	0.854
no_loan	nonexistent	0.871
nonexistent	no_loan	0.824
Not_Open_Account	no_default	0.774
Not_Open_Account	no_loan	0.823
no_loan	Not_Open_Account	0.880
no_default	Not_Open_Account	0.725
Not_Open_Account	no_default	0.763
Not_Open_Account	nonexistent	0.878
nonexistent	Not_Open_Account	0.906
no_default	Not_Open_Account	0.774
Not_Open_Account	nonexistent	0.892
Not_Open_Account	no_loan	0.824
no_loan	Not_Open_Account	0.916
Not_Open_Account	no_loan	0.734
no_loan	Not_Open_Account	0.798
nonexistent	Not_Open_Account	0.755