Problem formulation

Where will the person go this month?

In order to predict trips of a particular person during the year we break countries around the world into groups with respect to their specifics relatively each month of a year.

Data preparation

Firstly, we read data, and created two new columns: month and year of a purchase. This information was saved in a file V04.

In [2]:
```
import sklearn
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import xgboost as xgb
warnings.filterwarnings("ignore")
```

In [ ]:
```
data = pd.read_csv('MSU_DATA_TO_SHARE_V03.tsv', delimiter = '\t')
```

In [ ]:
```
data.drop(['sample_data'], 1, inplace = True)
```

In [ ]:
```
import datetime

temp1 = data['purchase_dt']
temp1 = temp1.apply(lambda s: datetime.datetime.strptime(s, '%Y-%m-

data['purchase_real_data'] = temp1

data['purchase_month'] = temp1.apply(lambda s: s.month)
data['purchase_year'] = temp1.apply(lambda s: s.year)

data.drop(['purchase_dt'], 1, inplace = True)

data.to_csv('MSU_DATA_TO_SHARE_V04.csv', index = False)
```

Then we deleted E-COMM payments and payments which were made in Russian Federation — thus we got information about purchases which were made only abroad and only face-to-face. This information went to file V05.

In [ ]:
```
tempData = data.copy()

tempData = tempData[~tempData['f2f_ecomm_flg'].isin(['E-COMM'])]
tempData = tempData[~tempData['mrch_country_nm'].isin(['RUSSIAN FED

tempData.to_csv('MSU_DATA_TO_SHARE_V05.csv', index = False)
```

Data exploration

Analyze how many people travel to a particular country in a particular month and also their average bill in them.

In [16]:
```
data = pd.read_csv('MSU_DATA_TO_SHARE_V05.csv')
```

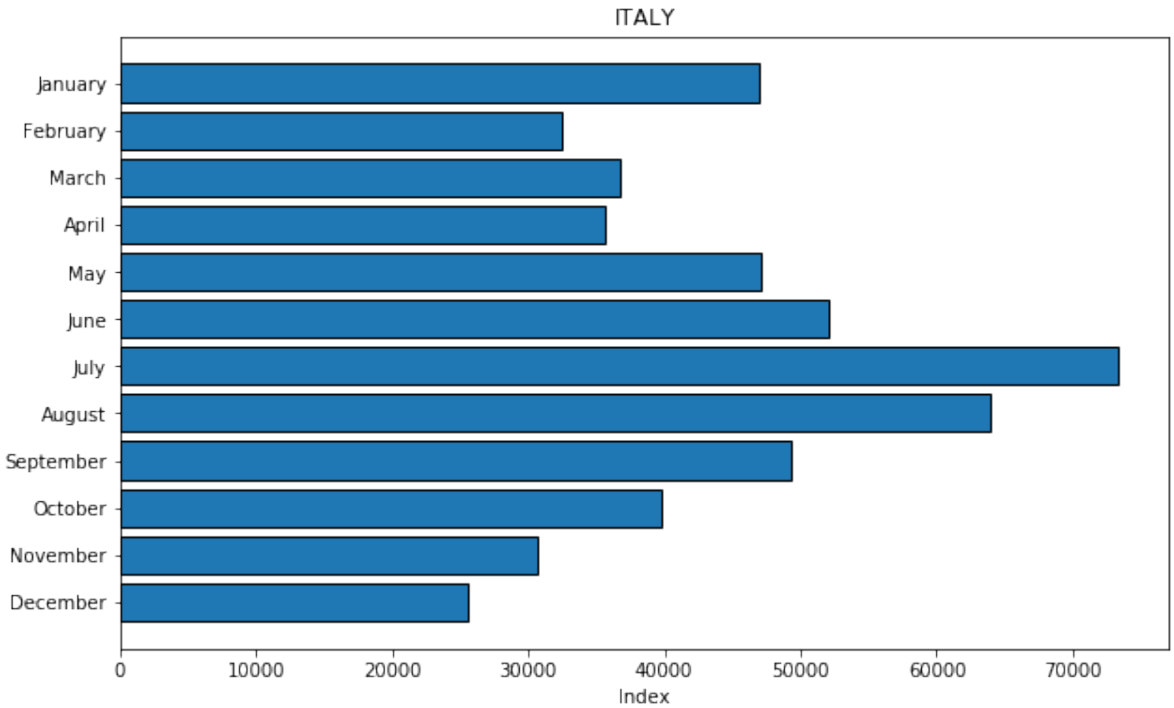
In [17]:
```
months = ['January', 'February',
          'March', 'April', 'May',
          'June', 'July', 'August',
          'September', 'October', 'November',
          'December']
```

Let's look at the most typical examples.

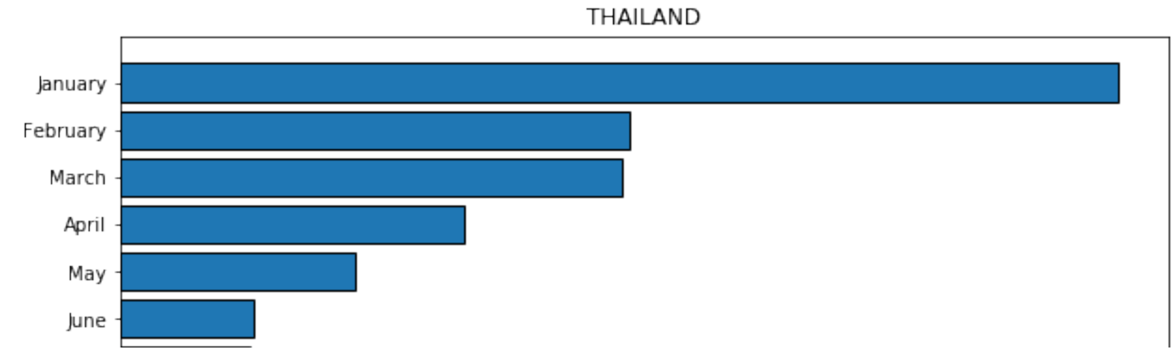
In [18]:
```
exampleCountries = ['ITALY', 'THAILAND', 'UNITED ARAB EMIRATES',
                    'GREECE', 'SPAIN', 'TURKEY', 'ANDORRA',
                    'AUSTRIA', 'GERMANY']

for country in exampleCountries:
    print(country)
    tempCountry = data.loc[(data['mrch_country_nm'] == country)]
    print("Mean check: ", int(tempCountry['rub_amt'].mean()))
    print("Max check: ", int(tempCountry['rub_amt'].max()))
    print("Median check: ", int(tempCountry['rub_amt'].median()))
    print("Sum check: ", int(tempCountry['rub_amt'].sum()))

    array = []

    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
    array.append(data.loc[(data['mrch_country_nm'] == country) & (d
```

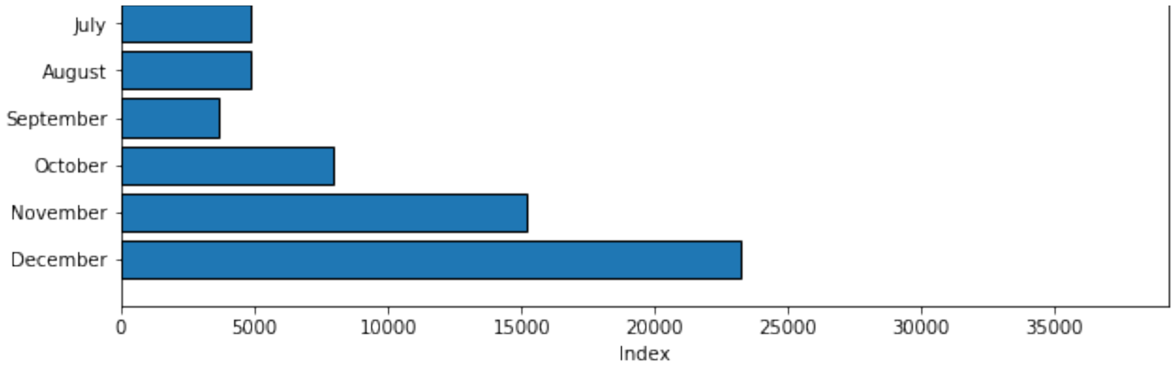
```
    plt.figure(figsize = (10, 6))
    ax = plt.subplot()

    ax.barh(list(reversed(list(months))), list(reversed(list(array)

    ax.set_yticks(list(reversed(list(months))))
    ax.set_yticklabels(list(reversed(list(months))))

    plt.xlabel('Index');
    plt.title(country)
    plt.show()
```

```
ITALY
Mean check:  8462
Max check:  12987080
Median check:  2347
Sum check:  4514720208
```

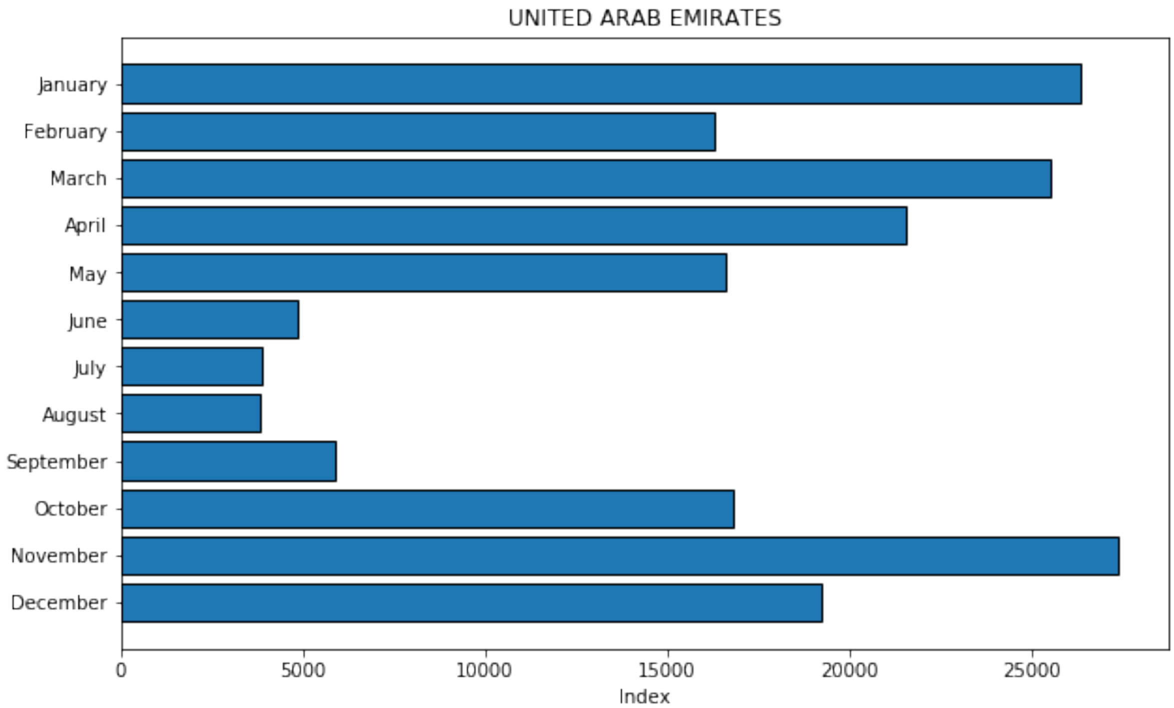



```
THAILAND
Mean check:  6865
Max check:  1563142
Median check:  2103
Sum check:  1110406528
```

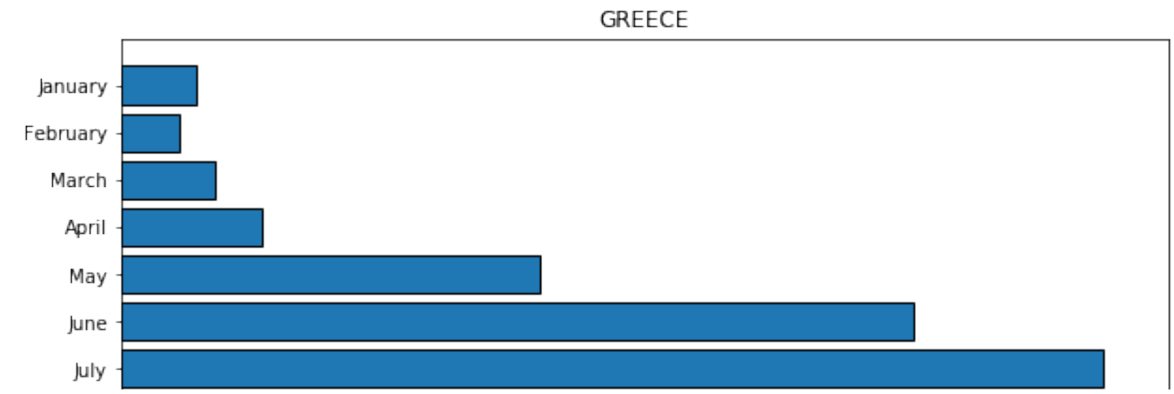


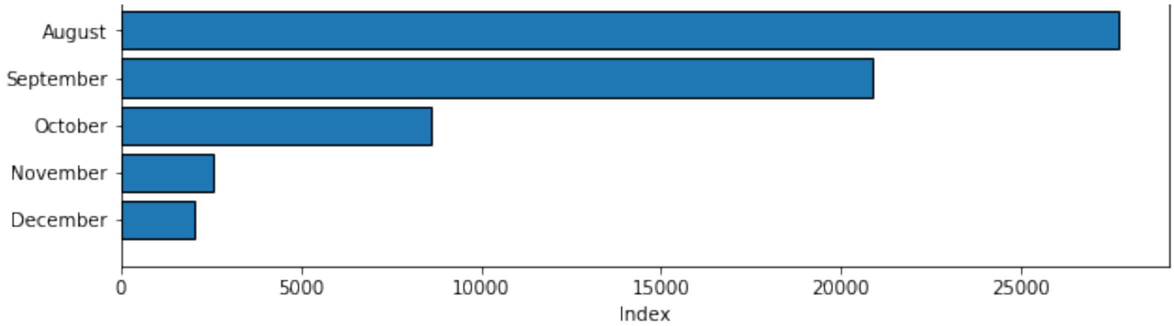



```
UNITED ARAB EMIRATES
Mean check:  10992
Max check:  4148614
Median check:  2102
Sum check:  2067880487
```

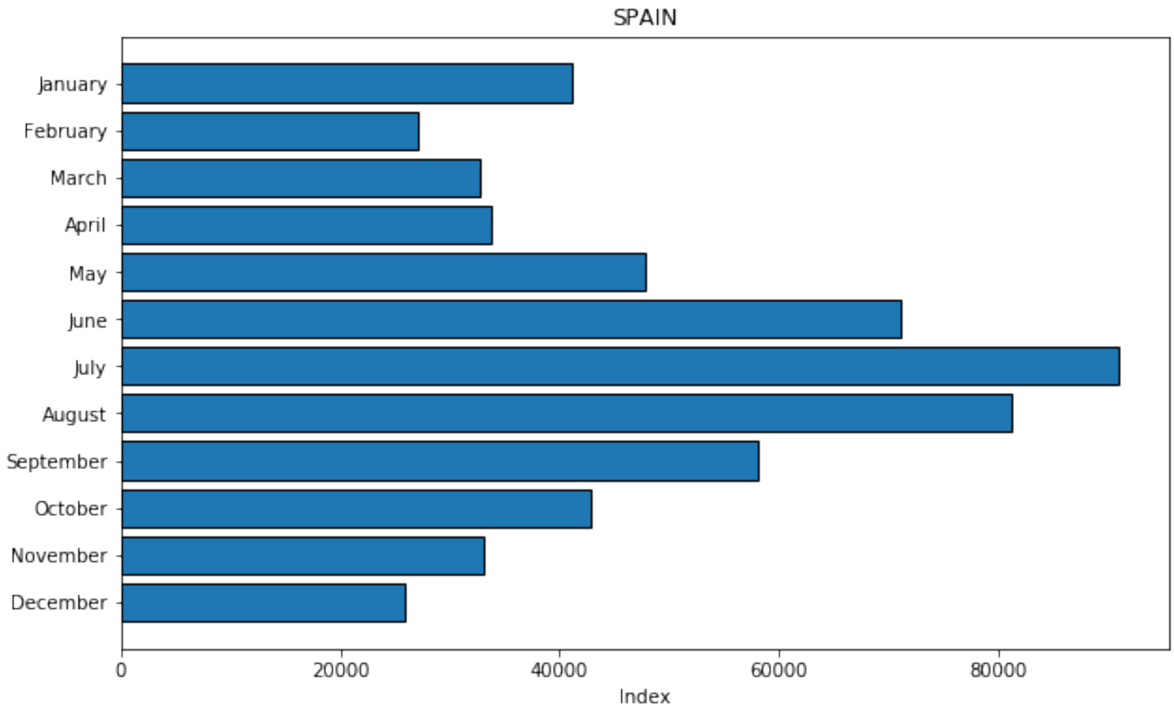



```
GREECE
Mean check:  5774
Max check:  4636074
Median check:  1681
Sum check:  767736002
```

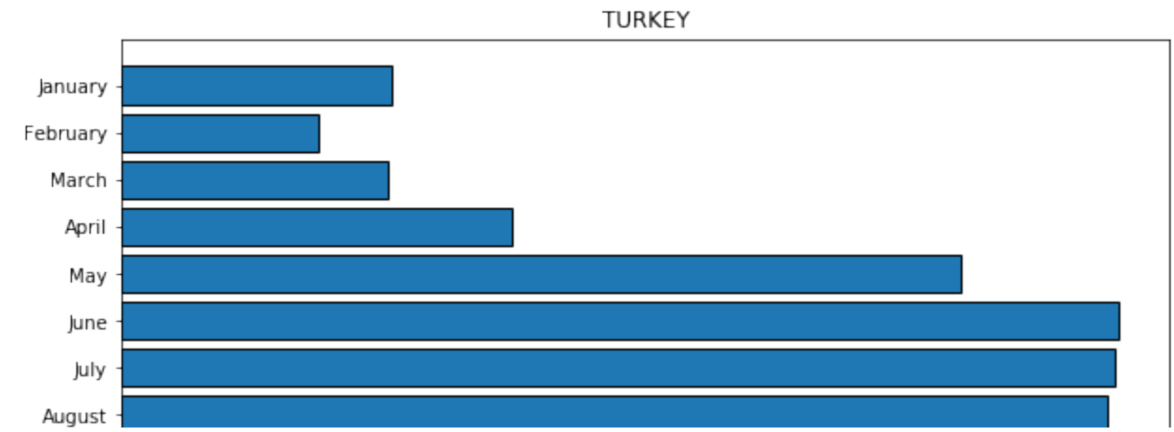


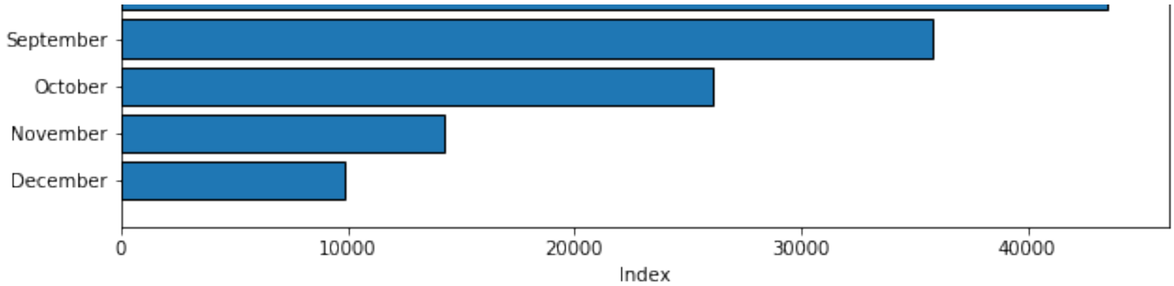



```
SPAIN
Mean check:  5365
Max check:  9060643
Median check:  1634
Sum check:  3142283140
```

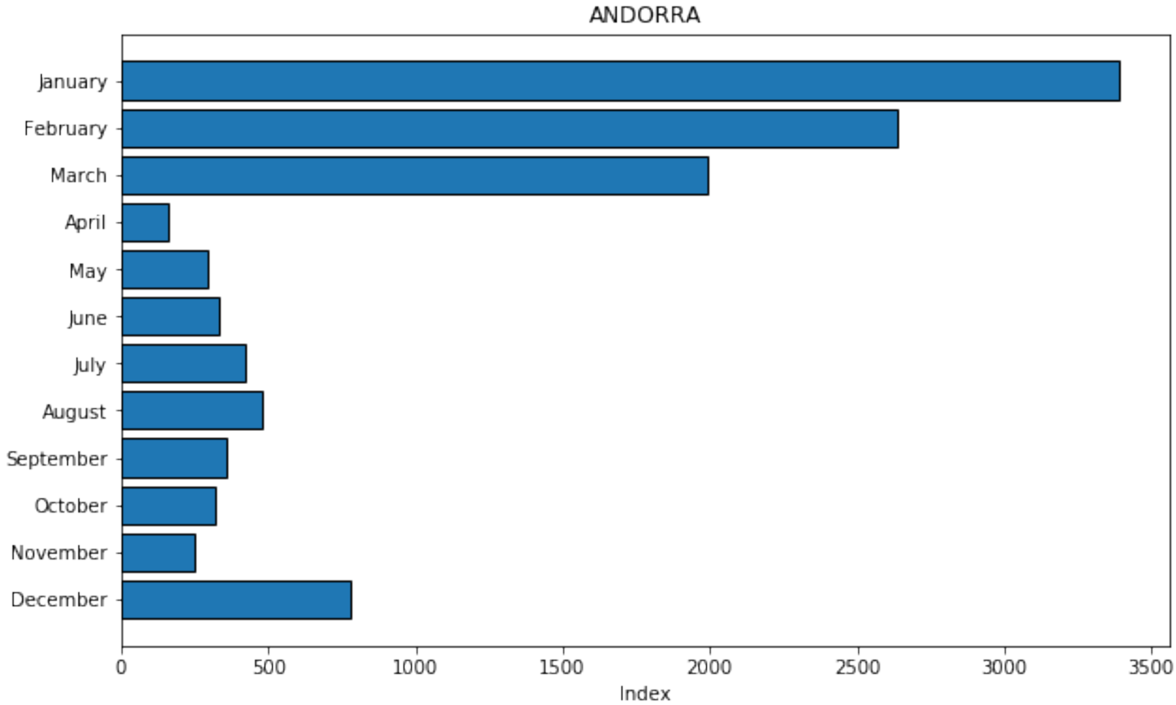



```
TURKEY
Mean check:  4264
Max check:  3377308
Median check:  1023
Sum check:  1296859439
```

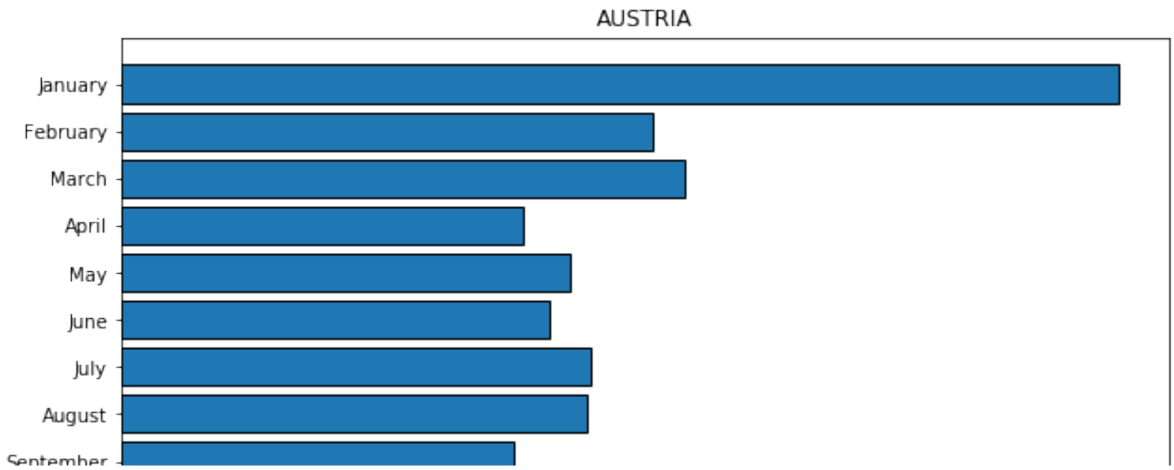


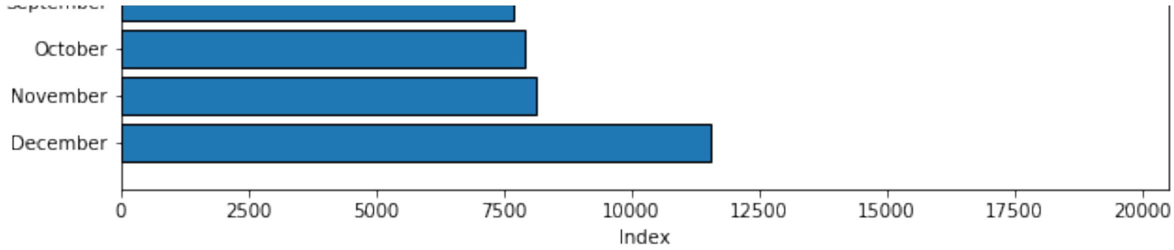



```
ANDORRA
Mean check:  5688
Max check:  545193
Median check:  2321
Sum check:  64895507
```

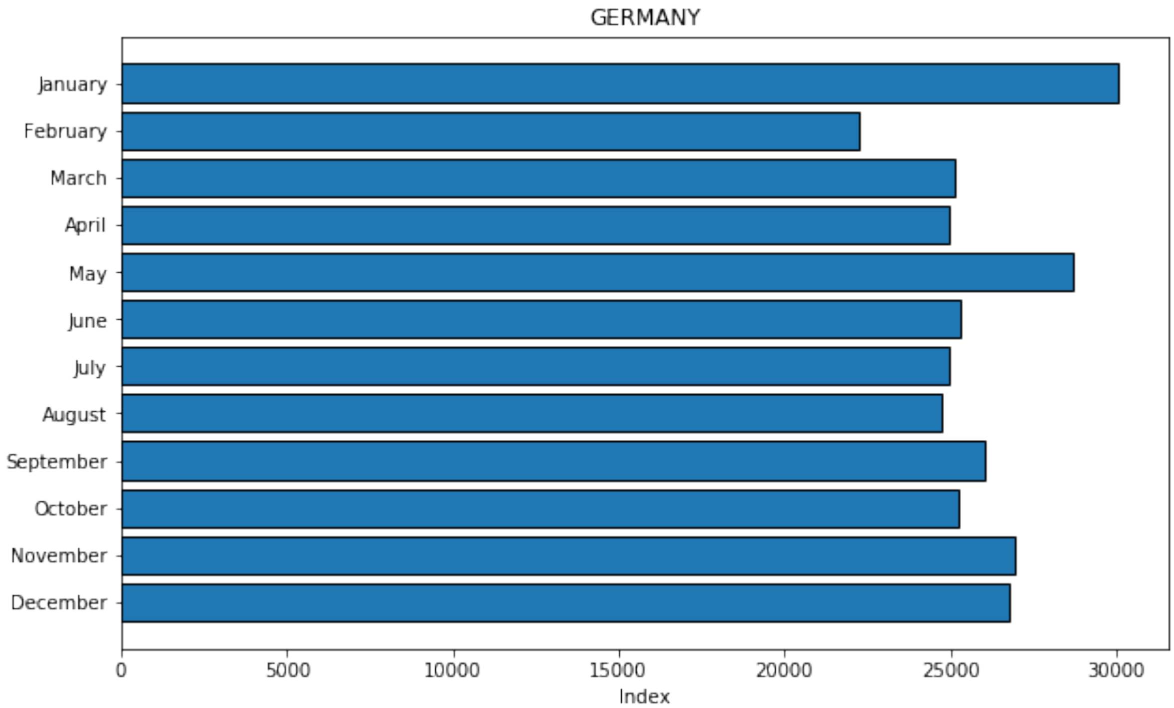



```
AUSTRIA
Mean check:  7748
Max check:  2898934
Median check:  2122
Sum check:  926852385
```



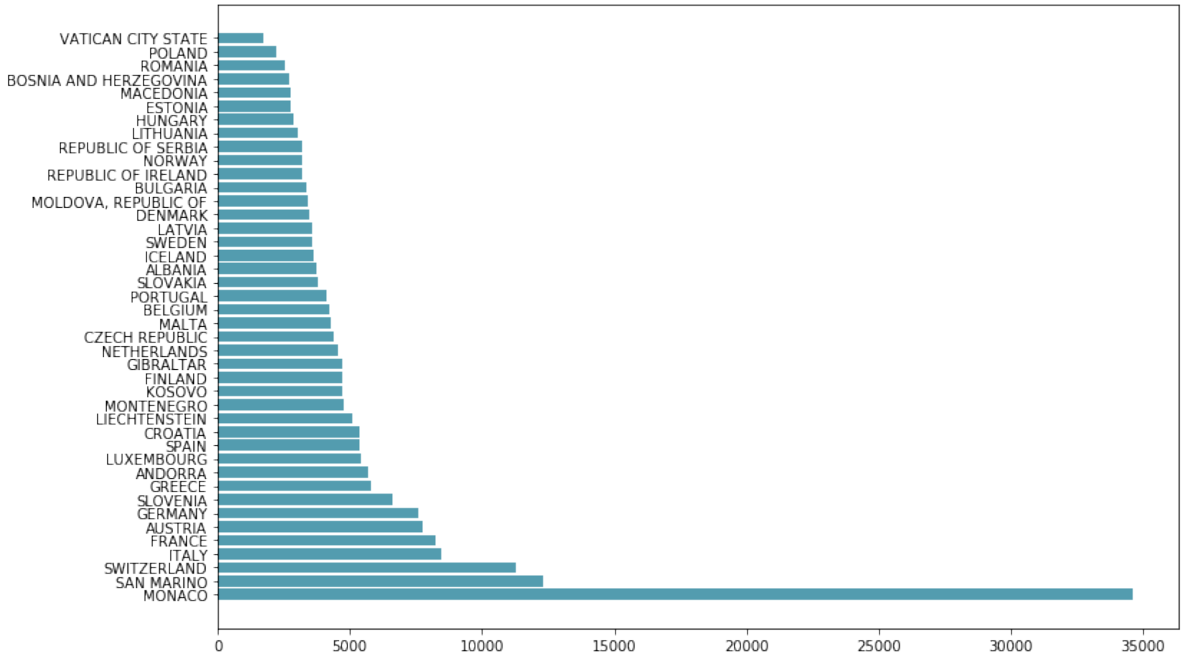



```
GERMANY
Mean check:  7610
Max check:  2931647
Median check:  2005
Sum check:  2368173203
```



However, there are several countries which are difficult to characterize with parameters below. That is why we group them according to their geographical position, costs and categories of purchases made in these countries.

In [5]:
```
EUROPE_ALL = ['AUSTRIA', 'BELGIUM', 'GERMANY', 'REPUBLIC OF IRELAND
              'MONACO', 'NETHERLANDS', 'FRANCE', 'SWITZERLAND', 'BU
              'MOLDOVA, REPUBLIC OF', 'POLAND', 'ROMANIA', 'SLOVAKI
              'DENMARK', 'ICELAND', 'NORWAY', 'LATVIA', 'LITHUANIA'
              'ESTONIA', 'ALBANIA', 'ANDORRA', 'BOSNIA AND HERZEGOV
              'GREECE', 'SPAIN', 'ITALY', 'MALTA', 'PORTUGAL', 'SAN
              'REPUBLIC OF SERBIA', 'KOSOVO', 'SLOVENIA', 'CROATIA'

data_eur = data[data['mrch_country_nm'].isin(EUROPE_ALL)]
data_eur = data_eur[['mrch_country_nm', 'rub_amt']]

temp_mean_eur = data_eur.groupby(['mrch_country_nm'], as_index=Fals
temp_mean_eur.rename(columns={'rub_amt': 'rub_amt_mean'}, inplace=T

temp_max_eur = data_eur.groupby(['mrch_country_nm'], as_index=False
temp_max_eur.rename(columns={'rub_amt': 'rub_amt_max'}, inplace=Tru
temp_eur = pd.merge(temp_mean_eur, temp_max_eur, how = 'left', on='

temp_median_eur = data_eur.groupby(['mrch_country_nm'], as_index=Fa
temp_median_eur.rename(columns={'rub_amt': 'rub_amt_median'}, inpla
temp_eur = pd.merge(temp_eur, temp_median_eur, how = 'left', on='mr

temp_sum_eur = data_eur.groupby(['mrch_country_nm'], as_index=False
temp_sum_eur.rename(columns={'rub_amt': 'rub_amt_sum'}, inplace=Tru
temp_eur = pd.merge(temp_eur, temp_sum_eur, how = 'left', on='mrch_

temp_eur = temp_eur.sort_values(by=['rub_amt_mean'], ascending=Fals

plt.figure(figsize = (12, 8))
ax = plt.subplot()
ax.barh(temp_eur['mrch_country_nm'].values, temp_eur['rub_amt_mean'
plt.show()
```



For example, this is distribution into groups in January. We use data about only the first year for creation of predictors.

In [6]:
```
dataFirst = data.loc[(
    ((data['purchase_year'] == 2016) & (data['purchase_month'] == 1
    ((data['purchase_year'] == 2016) & (data['purchase_month'] == 1
    ((data['purchase_year'] == 2016) & (data['purchase_month'] == 1
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 1
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 2
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 3
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 4
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 5
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 6
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 7
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 8
    ((data['purchase_year'] == 2017) & (data['purchase_month'] == 9
)]

dataFirst.head(5)
```

Out[6]:

	acct_num_share	issr_code_numb	product_nm	funding_source	mrch
12	fcaae931422688b8a0134e51a7a2fb12	9	VISA PLATINUM	C	NE
13	fcaae931422688b8a0134e51a7a2fb12	9	VISA PLATINUM	C	NE
14	fcaae931422688b8a0134e51a7a2fb12	9	VISA PLATINUM	C	NE
15	fcaae931422688b8a0134e51a7a2fb12	9	VISA PLATINUM	C	UN
16	fcaae931422688b8a0134e51a7a2fb12	9	VISA PLATINUM	C	UN

Now we will replace each country by the group to which it belongs. For example, Africa can be divided due to the list of "least developed countries" according to the United Nations.

In [ ]:
```
EUROPE_HIGH_MEAN_CHECK = ['MONACO', 'SAN MARINO', 'SWITZERLAND', 'I
                         'SLOVENIA', 'GREECE', 'ANDORRA', 'LUXEMBOU
                         'LIECHTENSTEIN', 'FINLAND', 'DENMARK', 'GI
                          'BELGIUM', 'REPUBLIC OF IRELAND']

EUROPE_LAW_MEAN_CHEK = ['BULGARIA', 'HUNGARY', 'MOLDOVA, REPUBLIC O
                        'SLOVAKIA', 'CZECH REPUBLIC',
                       'ICELAND', 'NORWAY', 'LATVIA', 'LITHUANIA',
                        'ESTONIA', 'ALBANIA', 'BOSNIA AND HERZEGOVI
                        'VATICAN CITY STATE', 'UKRAINE', 'MALTA', '
                        'REPUBLIC OF SERBIA', 'KOSOVO', 'MONTENEGRO

AFRICA_DEVELOPED = ['EGYPT', 'ALGERIA', 'SOUTH AFRICA', 'MOROCCO',
```

```
AFRICA_DEVELOPED = ['EGYPT', 'NIGERIA', 'SOUTH AFRICA', 'MOROCCO',
                    'ZIMBABWE', 'SWAZILAND', 'MAURITANIA', 'EQUATOR
                    'GHANA', 'CONGO', 'FRENCH GUIANA', 'NAMIBIA', '

AFRICA_UNDERDEVELOPED = ['ANGOLA', 'BENIN', 'BURKINA FASO', 'BURUND
                    'DEMOCRATIC REPUBLIC CONGO', 'LESOTHO', 'MADAGA
                    'RWANDA', 'SENEGAL', 'SOMALIA', 'SIERRA LEONE',
                    'UGANDA', 'CHAD', 'ETHIOPIA', 'SOUTH SUDAN']

MIDEAST = ['AZERBAIJAN', 'ARMENIA', 'BAHRAIN', 'GEORGIA', 'ISRAEL',
           'LEBANON', 'OCCUPIED PALESTINIAN TERR', 'IRAQ', 'KUWAIT'
           'UNITED ARAB EMIRATES', 'QATAR']

AUSTRALIA_INDOCHINE = ['AUSTRALIA', 'NEW ZEALAND', 'INDONESIA',
             'SINGAPORE', 'PHILIPPINES', 'PALAU', 'BRUNEI DARUSSALA
             'TIMOR-LESTE', 'MYANMAR', 'THAILAND', 'LAO PEOPLE\'S D
             'BANGLADESH', 'INDIA', 'SRI LANKA']

SOUTH_AMERICA = ['PERU', 'BRAZIL', 'CURACAO', 'MAURITIUS', 'ARGENTI
                 'CHILE', 'COLOMBIA', 'URUGUAY', 'ECUADOR', 'SURINA
                 'EL SALVADOR', 'GUYANA', 'VENEZUELA']

CENTRAL_AMERICA = ['REPUBLICA DOMINICANA', 'CUBA', 'PANAMA', 'GUATE
                   'COSTA RICA', 'GRENADA', 'GUAM', 'TRINIDAD AND T
                   'HONDURAS', 'NICARAGUA', 'PUERTO RICO', 'DOMINIC

ASIA_MIDASIA = ['SOUTH KOREA', 'CHINA', 'JAPAN', 'HONG KONG, CHINA'
        'MACAU, CHINA', 'BHUTAN', 'NEPAL', 'KAZAKHSTAN', 'UZBEKISTA
        'TAJIKISTAN', 'TURKMENISTAN','PAKISTAN', 'AFGHANISTAN']

ISLANDS = ['MALDIVES', 'SEYCHELLES', 'BES ISLANDS', 'TURKS & CAICOS
           'GUADELOUPE', 'FIJI', 'NORTHERN MARIANA ISLANDS', 'U.S.
           'AMERICAN SAMOA', 'CAYMAN ISLANDS', 'BARBADOS', 'MARTINI
            'ARUBA', 'VANUATU', 'BRITISH VIRGIN ISLANDS', 'ST. LUCI
            'ST. VINCENT & GRENADINES', 'ANGUILLA', 'REUNION', 'COO
            'FAEROE ISLANDS', 'MICRONESIA', 'SAMOA', 'BERMUDA', 'ST
            'CAPE VERDE', 'FRENCH POLYNESIA']

mapJan = {}

mapJan['UNITED STATES OF AMERICA'] = 'USA_CANADA'
mapJan['BELARUS'] = 'BELARUS'
mapJan['UNITED KINGDOM'] = 'UNITED KINGDOM'
mapJan['CANADA'] = 'USA_CANADA'

for country in EUROPE_HIGH_MEAN_CHECK:
    mapJan[country] = 'EUROPE'

for country in EUROPE_LAW_MEAN_CHEK:
    mapJan[country] = 'EUROPE'

for country in AFRICA_DEVELOPED:
    mapJan[country] = 'OTHERS'

for country in AFRICA_UNDERDEVELOPED:
```

```
for country in AFRICA_UNDERDEVELOPED:
    mapJan[country] = 'OTHERS'

for country in AUSTRALIA_INDOCHINE:
    mapJan[country] = 'OTHERS'

for country in ASIA_MIDASIA:
    mapJan[country] = 'ASIA_MIDASIA'

for country in ISLANDS:
    mapJan[country] = 'OTHERS'

for country in CENTRAL_AMERICA:
    mapJan[country] = 'OTHERS'

for country in MIDEAST:
    mapJan[country] = 'OTHERS'

for country in SOUTH_AMERICA:
    mapJan[country] = 'OTHERS'


SEA = ['AUSTRALIA', 'CUBA', 'INDIA', 'FIJI', 'SRI LANKA', 'EGYPT',
       'UNITED ARAB EMIRATES', 'QATAR', 'SEYCHELLES',
       'MALTA', 'MALDIVES', 'VIETNAM', 'NEW ZEALAND', 'THAILAND', '
       'REPUBLICA DOMINICANA', 'BRAZIL', 'CURACAO', 'PANAMA', 'GUAT
       'COSTA RICA', 'GRENADA', 'GUAM', 'MADAGASCAR', 'TRINIDAD AND
       'HONDURAS', 'NICARAGUA', 'PUERTO RICO', 'DOMINICA', 'ARGENTI

for country in SEA:
    mapJan[country] = 'SEA'

SKI = ['ITALY', 'FRANCE', 'AUSTRIA', 'ANDORRA', 'SWITZERLAND', 'NOR

for country in SKI:
    mapJan[country] = 'SKI'

dataFirst['mrch_country_nm_jan'] = dataFirst['mrch_country_nm'].map
dataFirst.head(5)
```

Formation of the table where each card (each client) is aligned to a vector of 0 and 1. In it i-th coordinate equals 1 if this particular person used to be in i-th group of countries.

In [9]:
```
dataJan = dataFirst.loc[(
    ((dataFirst['purchase_month'] == 1))
)]
clients = data['acct_num_share'].drop_duplicates().values
countries = dataFirst['mrch_country_nm_jan'].drop_duplicates().valu

stringsJan = []
columnsJan = []

columnsJan.append('CLIENTS')

for country in countries:
    columnsJan.append(country+'_Jan')

for client in clients:
    string = []
    string.append(client)
    dataClientJan = dataJan.loc[dataJan['acct_num_share'] == client
    for country in countries:
        size = dataClientJan.loc[dataClientJan['mrch_country_nm_jan
        if (size == 0):
            string.append(0)
        else:
            string.append(1)
    stringsJan.append(string)

finalJan = pd.DataFrame(np.array(stringsJan), columns = columnsJan)
finalJan.head(10)
```

Out[9]:

	CLIENTS	EUROPE_WEST_Jan	UNITED STATES OF AMERICA_Jan	EUROPE_NORT
0	a35eecb35b444d4abee26ea2a0916aff	0	0	
1	fcaae931422688b8a0134e51a7a2fb12	1	0	
2	47ce49be6ae7b918bc8744c9260a6d79	0	0	
3	1ccfe2c0b2410701a7ea1f4f3d3e046c	0	0	
4	61091a69688cefe04370072bdc1c92ea	0	0	
5	d7da56a6f56cca9702e84a2bc9907ce0	0	0	
6	401535e16e38d28c27a82357df0322f2	1	0	
7	8d870db2894ccea8297c998249fe8b91	0	0	
8	715735a90f4b1c43c73170b1343a65d2	0	0	
9	665bcbafed3945a02e84d34e0f616022	0	0	

10 rows × 23 columns

In [ ]:
```
finalJan.to_csv('FinalJan.csv', index = False)
```

Category features engineering

In [ ]:
```
cat1_data = dataFirst.groupby(['acct_num_share','mrch_category_01']
cat2_data = dataFirst.groupby(['acct_num_share','mrch_category_02']
cat3_data = dataFirst.groupby(['acct_num_share','mrch_category_03']
cat4_data = dataFirst.groupby(['acct_num_share','mrch_category_04']
```

In [ ]:
```
cat1_data_uns = cat1_data.unstack(level=-1)
cat2_data_uns = cat2_data.unstack(level=-1)
cat3_data_uns = cat3_data.unstack(level=-1)
cat4_data_uns = cat4_data.unstack(level=-1)
```

In [ ]:
```
cat1_data_uns.columns = cat1_data_uns.columns.droplevel()
cat1_data_uns.columns = ['_'.join(reversed(x)) + '_01' for x in cat

cat2_data_uns.columns = cat2_data_uns.columns.droplevel()
cat2_data_uns.columns = ['_'.join(reversed(x)) + '_02' for x in cat

cat3_data_uns.columns = cat3_data_uns.columns.droplevel()
cat3_data_uns.columns = ['_'.join(reversed(x)) + '_03' for x in cat

cat4_data_uns.columns = cat4_data_uns.columns.droplevel()
cat4_data_uns.columns = ['_'.join(reversed(x)) + '_04' for x in cat

cat1_data_uns.head(10)
```

In [ ]:
```
concated = pd.concat([cat1_data_uns, cat2_data_uns, cat3_data_uns,
```

In [ ]:
```
concated.to_csv('result.csv')
```

Final data preparation

In [ ]:
```
FinalJan = pd.read_csv('FinalJan.csv')
FinalJan.rename(columns = {'CLIENTS': 'acct_num_share'}, inplace =

ChecksAbroad = pd.read_csv('ChecksAbroad.csv')
ChecksAbroad.rename(columns = {'CLIENTS': 'acct_num_share'}, inplac

data = pd.read_csv('MSU_DATA_TO_SHARE_V04.csv')

tempData = data.copy()
tempData.drop(['mrch_country_nm',
               'f2f_ecomm_flg',
               'pos_cash_flg',
               'rub_amt',
               'mrch_category_01',
               'mrch_category_02',
              'mrch_category_03',
               'mrch_category_04',
               'purchase_real_data',
               'purchase_month',
               'purchase_year'],
1,inplace = True)
```

Decoding names of cards according to their kind. After that we create a table in which every card occur only once in contrast with the previous one, where all payments from all the cards were placed. Later we collate each card to its kind, type (debit or credit) and bank-issuer.

In [ ]:
```
CardsMap = {}

CardsMap['VISA INFINITE'] = 7
CardsMap['VISA SIGNATURE'] = 6
CardsMap['VISA PLATINUM'] = 5
CardsMap['VISA REWARDS'] = 4
CardsMap['VISA GOLD'] = 3
CardsMap['VISA CLASSIC'] = 2
CardsMap['VISA ELECTRON'] = 1

tempData['product_nm'] = tempData['product_nm'].map(CardsMap)
```

In [ ]:
```
funding = {}

funding['C'] = 0
funding['D'] = 1

tempData['funding_source'] = tempData['funding_source'].map(funding
```

In [ ]:
```
temp = tempData.groupby(['acct_num_share'], as_index=False).median(
```

In [ ]:
```
finalData = pd.merge(temp, FinalJan, how = 'left', on = 'acct_num_s
finalData = finalData.fillna(value=0.0)
```

Then we unite these tables with the initial one with the information about cards which was mentioned above. Then we delete columns where the major part of data is missed and save it into a final file.

In [ ]:
```
finalData = pd.merge(finalData, ChecksAbroad, how = 'left', on = 'a
```

In [ ]:
```
clmns = finalData.columns[finalData.isnull().any()]
missed = pd.DataFrame(finalData[clmns].isnull().sum().sort_values(a
```

In [ ]:
```
finalData.drop(missed[missed['% NULL'] > 0.9].index, 1, inplace=Tru
finalData = finalData.fillna(value=0.0)
finalData.to_csv('DataForLearning.csv', index = False)
```

Target generation

Finally with information about the second year we make a target which predicts in which country a particular person will go. From the mentioned above it is obvious that such can be not the only one. This way a target will consist of a country groups' code to which most probably a particular client go. Consequently it will be one-dimensional.

In [ ]:
```
data = pd.read_csv('DataForLearning.csv')
```

In [ ]:
```
tempTarget = pd.read_csv('MSU_DATA_TO_SHARE_V05.csv')

tempTarget = tempTarget.loc[(
    ((tempTarget['purchase_year'] == 2017) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2017) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2017) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
    ((tempTarget['purchase_year'] == 2018) & (tempTarget['purchase_
)]
```

In [3]:
```
EUROPE_HIGH_MEAN_CHECK = ['MONACO', 'SAN MARINO', 'SWITZERLAND', 'I
                         'SLOVENIA', 'GREECE', 'ANDORRA', 'LUXEMBOU
                         'LIECHTENSTEIN', 'FINLAND', 'DENMARK', 'GI

EUROPE_LAW_MEAN_CHEK = ['BULGARIA', 'HUNGARY', 'MOLDOVA, REPUBLIC O
```

```
                        'ICELAND', 'NORWAY', 'LATVIA', 'LITHUANIA',
                         'VATICAN CITY STATE', 'MALTA', 'PORTUGAL',

AFRICA_DEVELOPED = ['EGYPT', 'ALGERIA', 'SOUTH AFRICA', 'MOROCCO',
                    'ZIMBABWE', 'SWAZILAND', 'MAURITANIA', 'EQUATOR
                    'GHANA', 'CONGO', 'FRENCH GUIANA', 'NAMIBIA', '

AFRICA_UNDERDEVELOPED = ['ANGOLA', 'BENIN', 'BURKINA FASO', 'BURUND
                    'DEMOCRATIC REPUBLIC CONGO', 'LESOTHO', 'MADAGA
                    'RWANDA', 'SENEGAL', 'SOMALIA', 'SIERRA LEONE',
                    'UGANDA', 'CHAD', 'ETHIOPIA', 'SOUTH SUDAN']

MIDEAST = ['AZERBAIJAN', 'ARMENIA', 'BAHRAIN', 'GEORGIA', 'ISRAEL',
           'LEBANON', 'OCCUPIED PALESTINIAN TERR', 'IRAQ', 'KUWAIT'
           'UNITED ARAB EMIRATES', 'QATAR']

AUSTRALIA_INDOCHINE = ['AUSTRALIA', 'NEW ZEALAND', 'INDONESIA',
             'SINGAPORE', 'PHILIPPINES', 'PALAU', 'BRUNEI DARUSSALA
             'TIMOR-LESTE', 'MYANMAR', 'THAILAND', 'LAO PEOPLE\'S D
             'BANGLADESH', 'INDIA', 'SRI LANKA']

SOUTH_AMERICA = ['PERU', 'BRAZIL', 'CURACAO', 'MAURITIUS', 'ARGENTI
                 'CHILE', 'COLOMBIA', 'URUGUAY', 'ECUADOR', 'SURINA
                 'EL SALVADOR', 'GUYANA', 'VENEZUELA']

CENTRAL_AMERICA = ['REPUBLICA DOMINICANA', 'CUBA', 'PANAMA', 'GUATE
                   'COSTA RICA', 'GRENADA', 'GUAM', 'TRINIDAD AND T
                   'HONDURAS', 'NICARAGUA', 'PUERTO RICO', 'DOMINIC

ASIA_MIDASIA = ['SOUTH KOREA', 'CHINA', 'JAPAN', 'HONG KONG, CHINA'
        'MACAU, CHINA', 'BHUTAN', 'NEPAL', 'KAZAKHSTAN', 'UZBEKISTA
        'TAJIKISTAN', 'TURKMENISTAN','PAKISTAN', 'AFGHANISTAN']

ISLANDS = ['MALDIVES', 'SEYCHELLES', 'BES ISLANDS', 'TURKS & CAICOS
           'GUADELOUPE', 'FIJI', 'NORTHERN MARIANA ISLANDS', 'U.S.
           'AMERICAN SAMOA', 'CAYMAN ISLANDS', 'BARBADOS', 'MARTINI
            'ARUBA', 'VANUATU', 'BRITISH VIRGIN ISLANDS', 'ST. LUCI
            'ST. VINCENT & GRENADINES', 'ANGUILLA', 'REUNION', 'COO
            'FAEROE ISLANDS', 'MICRONESIA', 'SAMOA', 'BERMUDA', 'ST
            'CAPE VERDE', 'FRENCH POLYNESIA']

mapJan = {}

mapJan['UNITED STATES OF AMERICA'] = 'USA_CANADA'
mapJan['BELARUS'] = 'BELARUS'
mapJan['UKRAINE'] = 'UKRAINE'
mapJan['UNITED KINGDOM'] = 'UNITED KINGDOM'
mapJan['CANADA'] = 'USA_CANADA'

for country in EUROPE_HIGH_MEAN_CHECK:
    mapJan[country] = 'EUROPE_HIGH_MEAN_CHECK'

for country in EUROPE_LAW_MEAN_CHEK:
    mapJan[country] = 'EUROPE_LAW_MEAN_CHEK'
```

```
for country in AFRICA_DEVELOPED:
    mapJan[country] = 'OTHERS'

for country in AFRICA_UNDERDEVELOPED:
    mapJan[country] = 'OTHERS'

for country in AUSTRALIA_INDOCHINE:
    mapJan[country] = 'OTHERS'

for country in ASIA_MIDASIA:
    mapJan[country] = 'ASIA_MIDASIA'

for country in ISLANDS:
    mapJan[country] = 'OTHERS'

for country in CENTRAL_AMERICA:
    mapJan[country] = 'OTHERS'

for country in MIDEAST:
    mapJan[country] = 'OTHERS'

for country in SOUTH_AMERICA:
    mapJan[country] = 'OTHERS'


SEA = ['CUBA', 'INDIA', 'FIJI', 'SRI LANKA', 'EGYPT', 'TURKEY', 'UN
       'MALTA', 'MALDIVES', 'VIETNAM', 'THAILAND', 'PHILIPPINES', '

for country in SEA:
    mapJan[country] = 'SEA'

SKI = ['ITALY', 'FRANCE', 'AUSTRIA', 'ANDORRA', 'SWITZERLAND', 'NOR

for country in SKI:
    mapJan[country] = 'SKI'
```

In [ ]:
```
tempTarget['mrch_country_nm_jan'] = tempTarget['mrch_country_nm'].m
```

In [ ]:
```
clients = data['acct_num_share'].drop_duplicates().values
```

In [ ]:
```
dataJan = tempTarget.loc[(
    ((tempTarget['purchase_month'] == 1))
)]

countries = tempTarget['mrch_country_nm_jan'].drop_duplicates().val

stringsJan = []
columnsJan = []

columnsJan.append('acct_num_share')

for country in countries:
    columnsJan.append(country+'_Jan')

for client in clients:

    string = []

    string.append(client)

    dataClientJan = dataJan.loc[dataJan['acct_num_share'] == client

    for country in countries:
        size = dataClientJan.loc[dataClientJan['mrch_country_nm_jan

        if (size == 0):
            string.append(0)
        else:
            string.append(1)

    stringsJan.append(string)
```

In [ ]:
```
target = pd.DataFrame(np.array(stringsJan), columns = columnsJan)
target.to_csv('target.csv', index = False)
```

Delete from target those people who didn’t travel anywhere during January of the second year.

In [ ]:
```
target = pd.read_csv('target.csv')
```

In [ ]:
```
for country in columnsJan:
    if country == 'acct_num_share':
        continue
    target[country] = pd.to_numeric(target[country])

target['SUM'] = 0

for country in columnsJan:
    if country == 'acct_num_share':
        continue
    target['SUM'] = target['SUM'] + target[country]

target['Final'] = 0
target['index'] = target['acct_num_share']
target = target.set_index('index')
```

In [ ]:
```
for client in clients:
    for i in range (0, countries.size):
        if target.loc[client, countries[i]+'_Jan'] == 1:
            target.loc[client, 'Final'] = i+1
            break
```

In [ ]:
```
temporTar = target[['acct_num_share', 'Final']]
```

Then we add target to the main table and now we are able to start learning.

In [ ]:
```
finData = pd.merge(temporTar, data, how = 'left', on = 'acct_num_sh
finData.rename(columns = {'Final': 'target'}, inplace = True)
finData.drop(['target'], 1, inplace = True)
finData.to_csv('dataWithTarget.csv', index = False)
```

Learning

In [3]:
```
data = pd.read_csv('dataWithTarget.csv')
```

In [4]:
```
data.drop('acct_num_share', 1, inplace = True)
```

In [5]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

X = data.loc[:, data.columns != 'target']
y = data.loc[:, data.columns == 'target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

forest = RandomForestClassifier(n_estimators = 400, max_depth = 6)
forest.fit(X_train, y_train.values.ravel())
y_pred = forest.predict(X_test)
y_pred_train = forest.predict(X_train)
y_pred_proba = forest.predict_proba(X_test)
y_pred_proba_train = forest.predict_proba(X_train)
```

In [15]:
```
metrics.precision_score(y_test, y_pred, average='micro')
```

Out[15]: 0.39585848268177176

In [16]:
```
metrics.precision_score(y_train, y_pred_train, average='micro')
```

Out[16]: 0.41562524874631857

In [6]:
```
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimato
indices = np.argsort(importances)[::-1]

print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. %s (%f)" % (f + 1, X.columns[indices[f]], importance

plt.figure(figsize=(10,10))
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices], color="r", align="

plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. SKI_Jan (0.053785)
2. BELARUS_Jan (0.050771)
3. EUROPE_Jan (0.036764)
4. SEA_Jan (0.030419)
5. RESTAURANT & QSR_max_03 (0.013655)
6. SERVICE STATIONS_max_01_x (0.012190)
7. TOLLS AND BRIDGE FEES_sum_01_x (0.010461)
8. SERVICE STATIONS_min_01_x (0.009777)
9. MISC FOOD STORES - DEFAULT_sum_01_x (0.009774)
10. SERVICE STATIONS_median_01_x (0.009709)
11. RESTAURANTS_max_02 (0.009455)
12. SERVICE STATIONS_mean_01_x (0.009280)
13. Eating places and restaurants_mean_01_x (0.008931)
14. Eating places and restaurants_median_01_x (0.008909)
15. GROCERY STORES/SUPERMARKETS_max_01_x (0.008211)
16. MISC FOOD STORES - DEFAULT_min_01_x (0.008048)
17. SERVICE STATIONS_sum_01_x (0.007998)
18. MISC FOOD STORES - DEFAULT_max_01_x (0.007851)
```

In [7]:
```
for f in range(X.shape[1]):
    if importances[indices[f]] < 0.001:
        data.drop([X.columns[indices[f]]], 1, inplace=True)
```

In [11]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

X = data.loc[:, data.columns != 'target']
y = data.loc[:, data.columns == 'target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

forest = RandomForestClassifier(n_estimators = 400, max_depth = 6)
forest.fit(X_train, y_train.values.ravel())
y_pred = forest.predict(X_test)
y_pred_train = forest.predict(X_train)
y_pred_proba = forest.predict_proba(X_test)
y_pred_proba_train = forest.predict_proba(X_train)
```

In [12]:
```
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimato
indices = np.argsort(importances)[::-1]

print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. %s (%f)" % (f + 1, X.columns[indices[f]], importance

plt.figure(figsize=(10,10))
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices], color="r", align="

plt.xlim([-1, X.shape[1]])
plt.show()
```

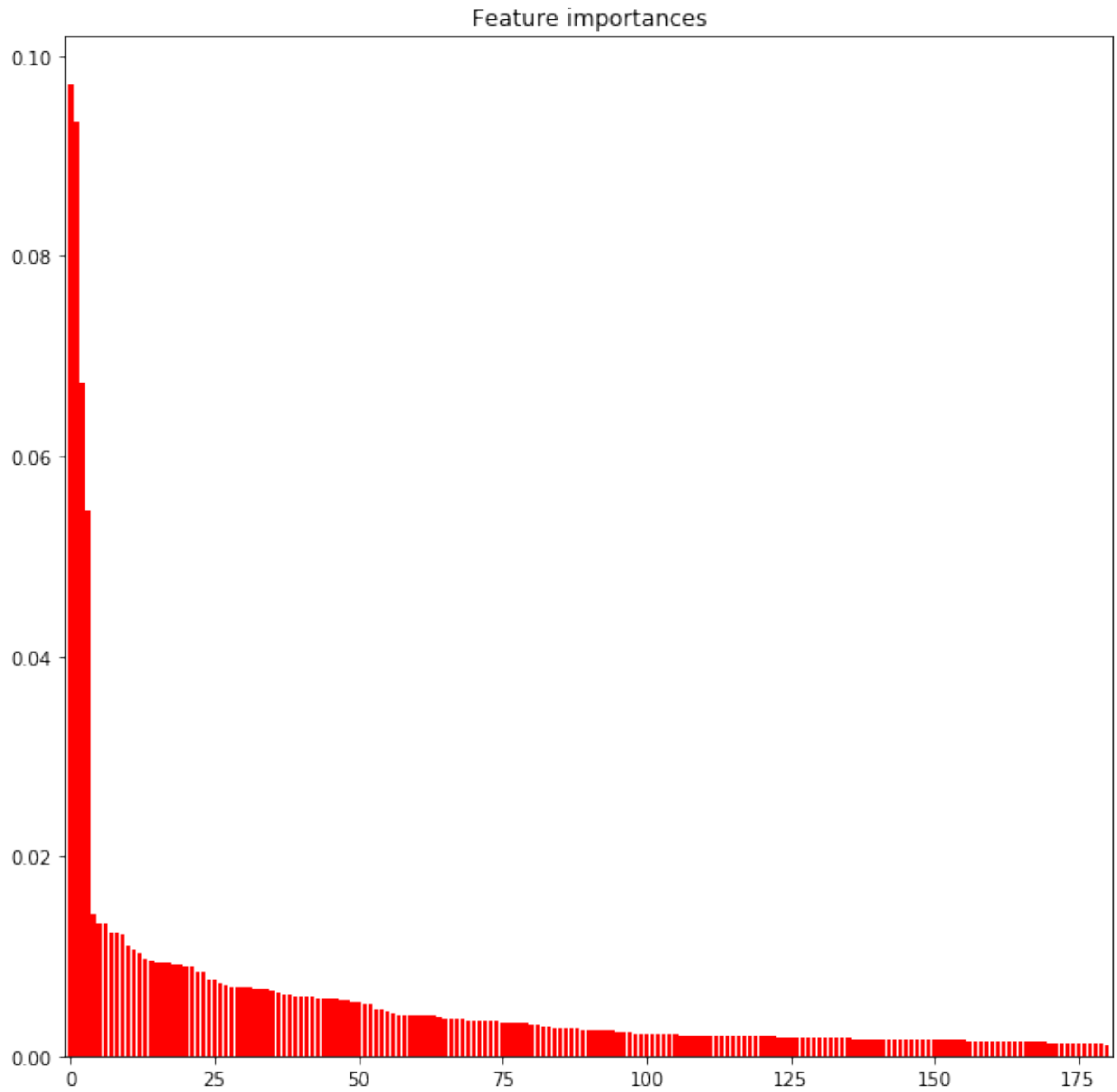
```
Feature ranking:
1. BELARUS_Jan (0.097135)
2. SKI_Jan (0.093472)
3. EUROPE_Jan (0.067237)
4. SEA_Jan (0.054616)
5. RESTAURANTS_max_02 (0.014297)
6. RESTAURANTS_max_04 (0.013284)
7. SERVICE STATIONS_max_01_x (0.013263)
8. TOLLS AND BRIDGE FEES_sum_01_x (0.012303)
9. TRAVEL_max_03 (0.012288)
10. SERVICE STATIONS_sum_01_x (0.012119)
11. SERVICE STATIONS_mean_01_x (0.011024)
12. Eating places and restaurants_mean_01_x (0.010640)
13. SERVICE STATIONS_min_01_x (0.010252)
14. Eating places and restaurants_median_01_x (0.009705)
15. SERVICE STATIONS_median_01_x (0.009516)
16. MISC FOOD STORES - DEFAULT_sum_01_x (0.009340)
17. GROCERY STORES/SUPERMARKETS_max_01_x (0.009272)
18. RESTAURANT & QSR_max_03 (0.009257)
```

```
19. MISC FOOD STORES - DEFAULT_max_01_x (0.009169)
20. GROCERY STORES/SUPERMARKETS_sum_01_x (0.009145)
21. MISC FOOD STORES - DEFAULT_median_01_x (0.008962)
22. RESTAURANTS_sum_02 (0.008936)
23. USA_CANADA_Jan (0.008468)
24. MISC FOOD STORES - DEFAULT_mean_01_x (0.008369)
25. MENS/WOMENS CLOTHING STORES_sum_01_x (0.007742)
26. FAST FOOD RESTAURANTS_mean_01_x (0.007589)
27. VARIETY STORES_max_01_x (0.007287)
28. Eating places and restaurants_max_01_y (0.007120)
29. MISC GENERAL MERCHANDISE_sum_01_x (0.006975)
30. MISC FOOD STORES - DEFAULT_min_01_x (0.006965)
31. GROCERY STORES/SUPERMARKETS_mean_01_x (0.006886)
32. RESTAURANTS_sum_04 (0.006877)
33. RESTAURANT & QSR_sum_03 (0.006751)
34. VARIETY STORES_median_01_x (0.006735)
35. Eating places and restaurants_sum_01_y (0.006726)
36. VARIETY STORES_min_01_x (0.006455)
37. RESTAURANT & QSR_mean_03 (0.006374)
38. Eating places and restaurants_mean_01_y (0.006136)
39. RESTAURANTS_mean_02 (0.006073)
40. LODGING_median_01_x (0.006059)
41. FAST FOOD RESTAURANTS_max_01_x (0.005954)
42. TRAVEL_sum_03 (0.005914)
43. FAST FOOD RESTAURANTS_sum_01_x (0.005913)
44. FAST FOOD RESTAURANTS_median_01_x (0.005854)
45. VARIETY STORES_sum_01_x (0.005794)
46. OTHERS_Jan (0.005771)
47. LODGING_sum_01_x (0.005706)
48. Eating places and restaurants_max_01_x (0.005553)
49. RESTAURANTS_mean_04 (0.005514)
50. FAST FOOD RESTAURANTS_min_01_x (0.005489)
51. VARIETY STORES_mean_01_x (0.005443)
52. GROCERY STORES/SUPERMARKETS_median_01_x (0.005171)
53. LODGING_mean_01_x (0.005156)
54. LODGING_max_01_x (0.004693)
55. DEPARTMENT & APPAREL_max_03 (0.004580)
56. Eating places and restaurants_sum_01_x (0.004443)
57. MENS/WOMENS CLOTHING STORES_mean_01_x (0.004335)
58. FAMILY CLOTHING STORES_mean_01_x (0.004106)
59. ALL AIRLINES_max_01_y (0.004099)
60. DEPARTMENT & APPAREL_mean_03 (0.004088)
61. LODGING_min_01_x (0.004077)
62. ASIA_MIDASIA_Jan (0.004047)
63. product_nm (0.004029)
64. TRAVEL_mean_03 (0.004023)
65. MENS/WOMENS CLOTHING STORES_max_01_x (0.003994)
66. DUTY FREE STORES_max_01_x (0.003748)
67. FAMILY CLOTHING STORES_max_01_x (0.003667)
68. FASHION RETAIL_mean_02 (0.003657)
69. GROCERY STORES/SUPERMARKETS_min_01_x (0.003646)
70. Eating places and restaurants_min_01_x (0.003611)
71. AIRLINES_max_02 (0.003591)
```

```
72. AIRLINES_max_04 (0.003576)
73. LUMBER/BUILD. SUPPLY STORES_sum_01_x (0.003551)
74. DUTY FREE STORES_min_01_x (0.003529)
75. DUTY FREE STORES_median_01_x (0.003464)
76. FOOD & DRUG_mean_03 (0.003401)
77. ALL AIRLINES_sum_01_y (0.003356)
78. FAMILY CLOTHING STORES_sum_01_x (0.003330)
79. POSTAGE STAMPS_sum_01_x (0.003291)
80. DUTY FREE STORES_mean_01_x (0.003260)
81. DUTY FREE STORES_sum_01_x (0.003149)
82. FAMILY CLOTHING STORES_median_01_x (0.003094)
83. UNITED KINGDOM_Jan (0.003002)
84. FASHION RETAIL_sum_02 (0.002989)
85. TRANSPORTATION_sum_04 (0.002859)
86. MENS/WOMENS CLOTHING STORES_median_01_x (0.002856)
87. APPAREL & ACCESSORIES_mean_04 (0.002811)
88. AIRLINES_mean_04 (0.002738)
89. RESTAURANTS_median_04 (0.002706)
90. DEPARTMENT & APPAREL_sum_03 (0.002614)
91. AIRLINES_sum_02 (0.002607)
92. FASHION RETAIL_max_02 (0.002602)
93. AIRLINES_mean_02 (0.002506)
94. FOOD & GROCERY_max_04 (0.002503)
95. AIRLINES_median_04 (0.002496)
96. AIRLINES_sum_04 (0.002479)
97. APPAREL & ACCESSORIES_max_04 (0.002433)
98. FUEL/SERVICE STATION_mean_02 (0.002324)
99. TRANSPORTATION_median_04 (0.002287)
100. LOCAL COMMUTER TRANSPORT_sum_01_y (0.002278)
101. FOOD & DRUG_max_03 (0.002278)
102. RETAIL GOODS_max_04 (0.002277)
103. LOCAL COMMUTER TRANSPORT_max_01_y (0.002271)
104. Eating places and restaurants_median_01_y (0.002227)
105. QSR_max_04 (0.002205)
106. RETAIL GOODS_sum_04 (0.002176)
107. FAMILY CLOTHING STORES_min_01_x (0.002117)
108. TRANSPORTATION_median_02 (0.002108)
109. TOLLS/FEES_max_02 (0.002085)
110. TAX PAYMENTS_sum_01_x (0.002075)
111. TRANSPORTATION_mean_04 (0.002071)
112. FUEL_mean_03 (0.002066)
113. APPAREL & ACCESSORIES_sum_04 (0.002050)
114. GROCERY STORES/SUPERMARKETS_mean_01_y (0.002011)
115. QSR_sum_04 (0.002002)
116. ALL AIRLINES_min_01_y (0.001999)
117. RETAIL SERVICES_max_03 (0.001991)
118. ALL AIRLINES_mean_01_y (0.001983)
119. APPAREL & ACCESSORIES_median_04 (0.001976)
120. RETAIL SERVICES_sum_03 (0.001972)
121. SERVICE STATIONS_mean_01_y (0.001965)
122. DEPARTMENT STORES_median_01_x (0.001960)
123. AIRLINES_min_04 (0.001955)
124. PASSENGER RAILWAYS_sum_01_x (0.001931)
```

```
125. INSURANCE SALES/UNDERWRITE_sum_01_x (0.001900)
126. RESTAURANT & QSR_median_03 (0.001899)
127. RETAIL GOODS_mean_04 (0.001897)
128. RETAIL SERVICES_median_03 (0.001870)
129. LODGING_max_04 (0.001870)
130. FUEL_mean_04 (0.001860)
131. MENS/WOMENS CLOTHING STORES_min_01_x (0.001856)
132. FAST FOOD RESTAURANTS_max_01_y (0.001816)
133. LOCAL COMMUTER TRANSPORT_median_01_y (0.001802)
134. BARS/TAVERNS/LOUNGES/DISCOS_median_01_x (0.001782)
135. LODGING_sum_02 (0.001764)
136. DEPARTMENT STORES_min_01_x (0.001755)
137. FOOD & GROCERY_mean_04 (0.001728)
138. DEPARTMENT STORES_mean_01_x (0.001702)
139. TRANSPORTATION_min_04 (0.001700)
140. FOOD & DRUG_median_03 (0.001689)
141. DRUG STORES & PHARMACIES_mean_01_x (0.001672)
142. LOCAL COMMUTER TRANSPORT_min_01_y (0.001670)
143. TOLLS/FEES_sum_02 (0.001668)
144. SUPERMARKETS_mean_02 (0.001665)
145. RESTAURANTS_median_02 (0.001660)
146. TOLLS AND BRIDGE FEES_max_01_y (0.001658)
147. FOOD & GROCERY_median_04 (0.001641)
148. LODGING_max_02 (0.001632)
149. FASHION RETAIL_median_02 (0.001629)
150. RETAIL GOODS_median_04 (0.001624)
151. RETAIL GOODS_sum_03 (0.001617)
152. DEPARTMENT STORES_max_01_x (0.001610)
153. LODGING_max_01_y (0.001604)
154. TOLLS AND BRIDGE FEES_sum_01_y (0.001601)
155. AIRLINES_min_02 (0.001594)
156. BARS/TAVERNS/LOUNGES/DISCOS_sum_01_x (0.001573)
157. ALL AIRLINES_median_01_y (0.001557)
158. DRUG STORES & PHARMACIES_median_01_x (0.001556)
159. FAST FOOD RESTAURANTS_mean_01_y (0.001540)
160. DRUG STORES & PHARMACIES_min_01_x (0.001536)
161. AIRLINES_median_02 (0.001534)
162. HEALTH CARE_max_03 (0.001531)
163. FUEL_median_03 (0.001528)
164. LOCAL COMMUTER TRANSPORT_mean_01_y (0.001512)
165. SPORTING GOODS STORES_sum_01_x (0.001490)
166. HEALTH CARE_max_04 (0.001466)
167. RETAIL SERVICES_mean_03 (0.001449)
168. SUPERMARKETS_median_02 (0.001439)
169. RETAIL SERVICES_min_03 (0.001415)
170. FAST FOOD RESTAURANTS_sum_01_y (0.001374)
171. MEDICAL/HEALTH SERVICES_median_02 (0.001348)
172. GROCERY STORES/SUPERMARKETS_median_01_y (0.001347)
173. AUTOMOTIVE PARTS STORES_sum_01_x (0.001343)
174. DRUG STORES & PHARMACIES_max_01_x (0.001294)
175. Eating places and restaurants_min_01_y (0.001286)
176. HEALTH CARE_sum_04 (0.001254)
177. DRUG STORES & PHARMACIES_sum_01_x (0.001249)
```

```
178. UTILITIES/ELEC/GAS/H2O/SANI_sum_01_x (0.001222)
179. LODGING_mean_04 (0.001214)
180. HEALTH CARE_median_03 (0.001186)
181. SPA / BEAUTY SERVICES_max_02 (0.000995)
```



XGBoost

In [8]:
```
X = data.loc[:, data.columns != 'target']
y = data.loc[:, data.columns == 'target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

model = xgb.XGBClassifier(max_depth = 3,
                          min_child_weight = 1,
                          learning_rate = 0.1,
                          n_estimators = 100,
                          subsample = 0.8,
                          max_features = 50)

model.fit(X_train, y_train.values.ravel())

y_pred = model.predict(X_test)
y_pred_train = model.predict(X_train)
y_pred_proba = model.predict_proba(X_test)
y_pred_proba_train = model.predict_proba(X_train)
```

In [23]:
```
metrics.precision_score(y_test.values.ravel(), y_pred, average='mic
```

Out[23]: 0.4305877983099638

In [24]:
```
metrics.precision_score(y_train.values.ravel(), y_pred_train, avera
```

Out[24]: 0.4688768606224628

In [31]:
```
y_pred_proba_train[0]
```

Out[31]:
```
array([0.19759469, 0.19347873, 0.430089  , 0.01397855, 0.00955001,
       0.0925578 , 0.03963443, 0.02311681], dtype=float32)
```

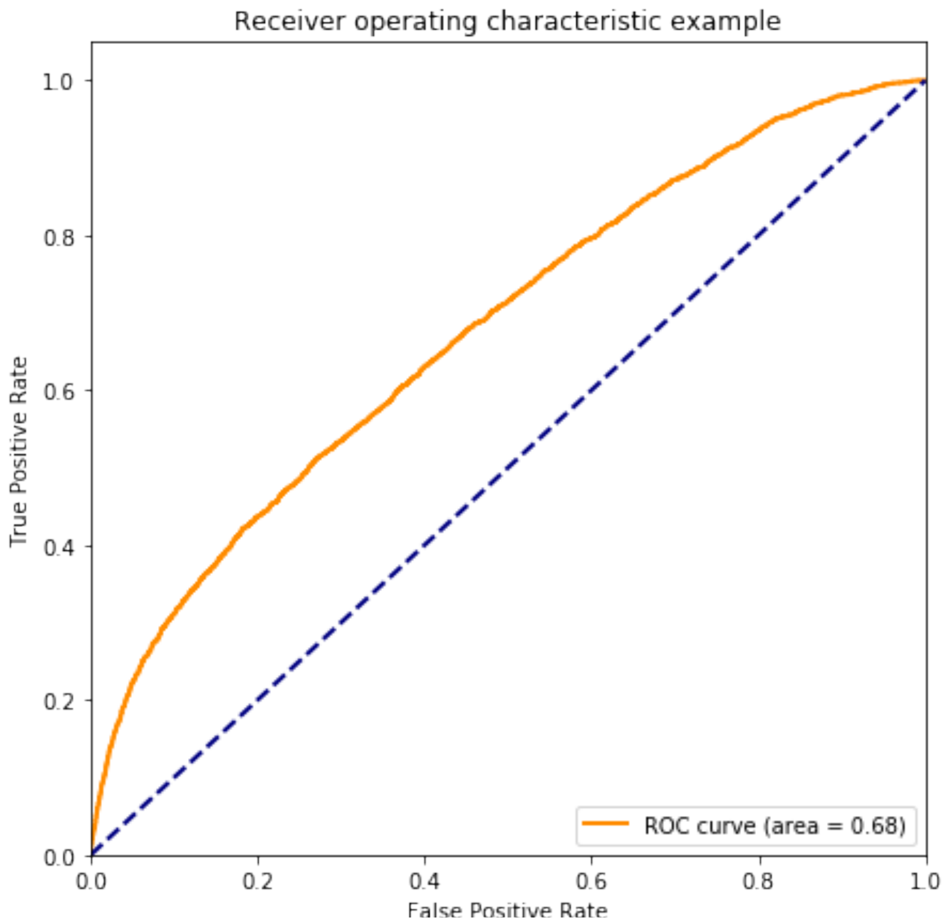
In [29]:
```
y_proba_test = pd.DataFrame(np.array(y_pred_proba))
```

In [32]:
```
y_proba_train = pd.DataFrame(np.array(y_pred_proba_train))
```

In [52]:
```
mapTar = {}
mapTar[1] = 1
mapTar[2] = 0
mapTar[3] = 0
mapTar[4] = 0
mapTar[5] = 0
mapTar[6] = 0
mapTar[7] = 0
mapTar[8] = 0

y_test_1 = y_test.copy()
y_test_1['target'] = y_test_1['target'].map(mapTar)
y_train_1 = y_train.copy()
y_train_1['target'] = y_train_1['target'].map(mapTar)
```

```
from sklearn.metrics import roc_curve, auc

score = metrics.roc_auc_score(y_test_1['target'].values, y_proba_te
print("Test: ", score)
score = metrics.roc_auc_score(y_train_1['target'].values, y_proba_t
print("Train: ", score)

fpr = dict()
tpr = dict()
roc_auc = dict()

fpr, tpr, thresholds = metrics.roc_curve(y_test_1, y_proba_test[0])
roc_auc = auc(fpr, tpr)

plt.figure(figsize = (7,7))
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (are
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

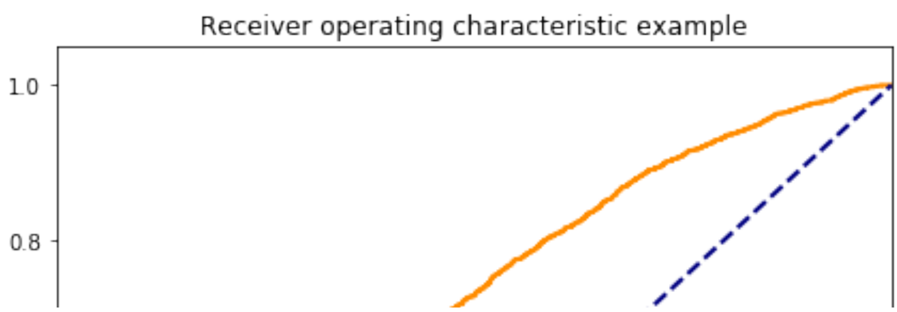
```
('Test: ', 0.6765982533617685)
('Train: ', 0.7177618141666475)
```

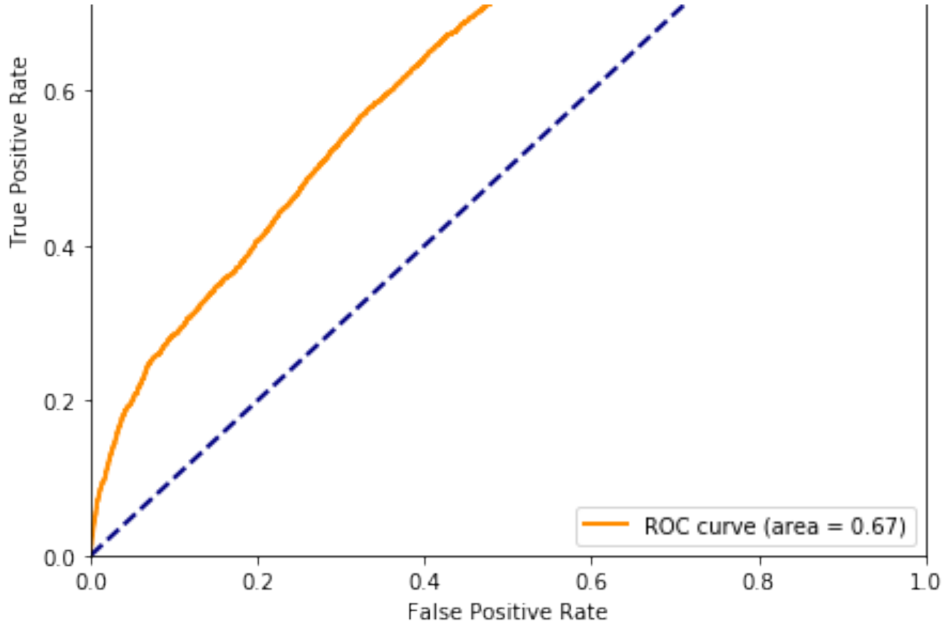


In [53]:

```
mapTar = {}
mapTar[1] = 0
mapTar[2] = 1
mapTar[3] = 0
mapTar[4] = 0
mapTar[5] = 0
mapTar[6] = 0
mapTar[7] = 0
mapTar[8] = 0

y_test_2 = y_test.copy()
y_test_2['target'] = y_test_2['target'].map(mapTar)
y_train_2 = y_train.copy()
y_train_2['target'] = y_train_2['target'].map(mapTar)

from sklearn.metrics import roc_curve, auc

score = metrics.roc_auc_score(y_test_2['target'].values, y_proba_te
print("Test: ", score)
score = metrics.roc_auc_score(y_train_2['target'].values, y_proba_t
print("Train: ", score)

fpr = dict()
tpr = dict()
roc_auc = dict()

fpr, tpr, thresholds = metrics.roc_curve(y_test_2, y_proba_test[1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize = (7,7))
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (are
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
('Test: ', 0.6749429317726717)
('Train: ', 0.7256396405729324)
```




True Positive Rate
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.67)

In [ ]: