Nubank data challenge - Default Analysis

by Adriano Freitas

```
%%capture

""" Useful notebook definitions

Some usefull notebook definitions, like plots color scheme
and cell behavior were extracted to another notebook just
for a cleaner view
"""
%run ./utils.ipynb

default_color = 'purple'
colormap = 'BuPu'
```

Importing data and first look

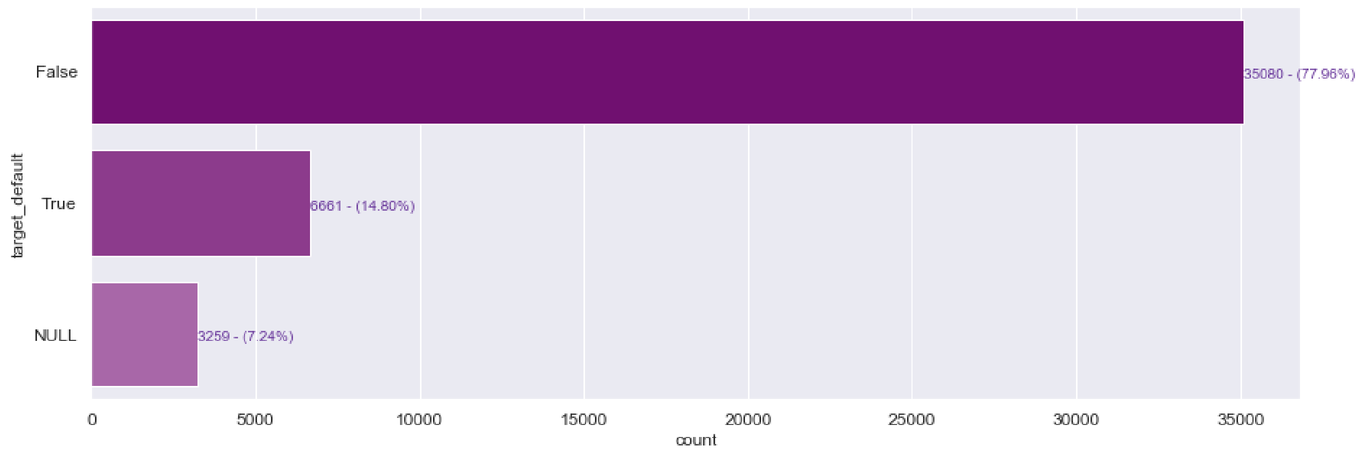
```
new_data_path = '../data/interim/'
```

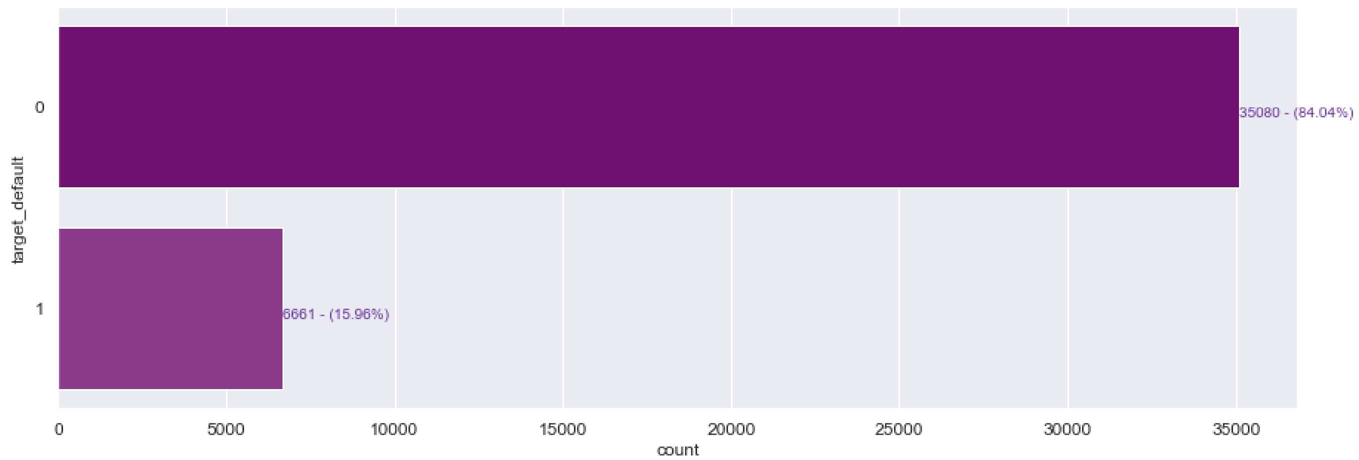
```
df_name = new_data_path + 'acquisition_train.csv'
```

```
df = pd.read_csv(df_name)
df.shape
df.info()
df.describe()
df.head()
```

```
# nulls on target
plot_count(df, 'target_default')
df.dropna(subset=['target_default'], inplace=True)
df['target_default'] = df['target_default'].apply(lambda x: 1 if x else 0)
plot_count(df, 'target_default')
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
AxesSubplot(0.125,0.125;0.775x0.755)
```



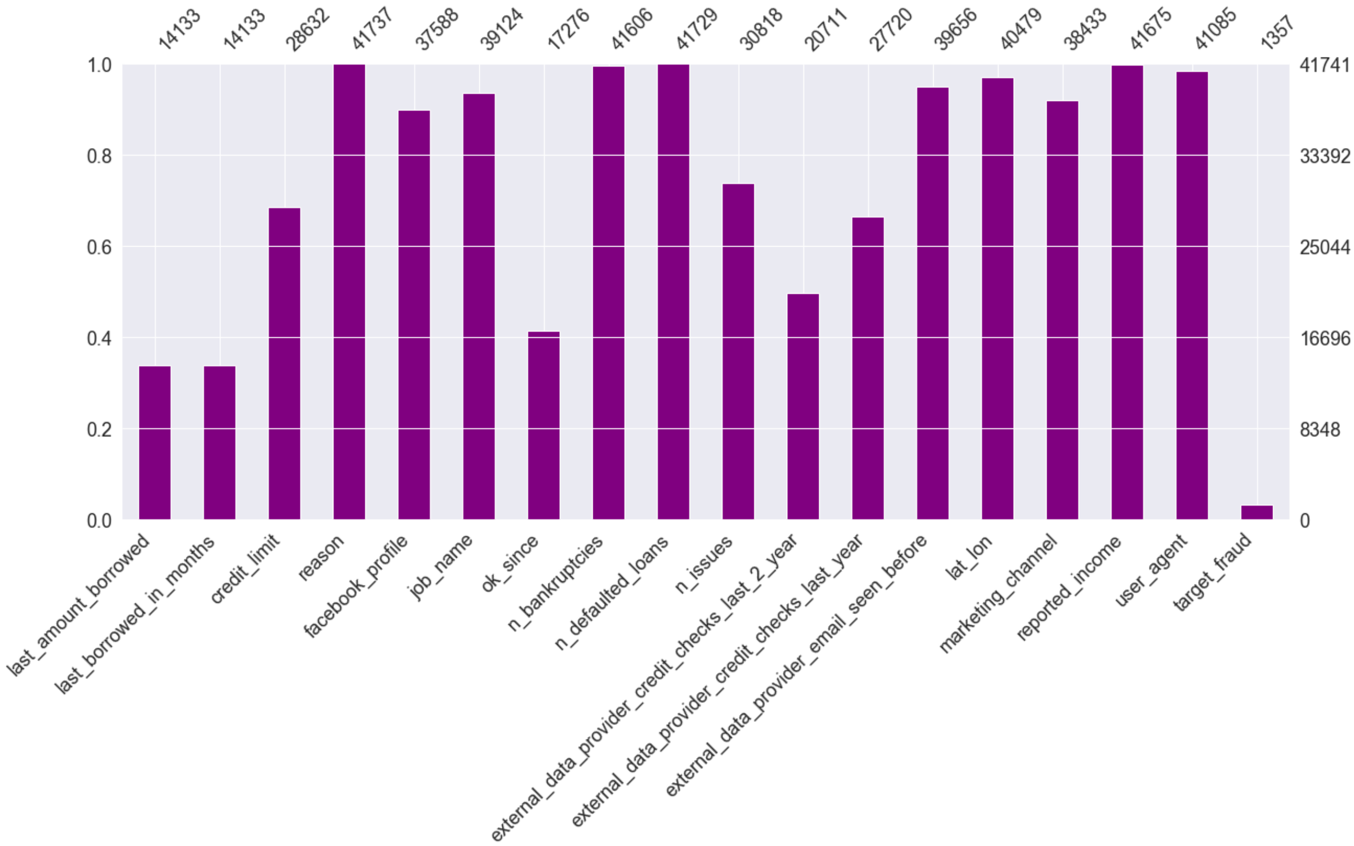


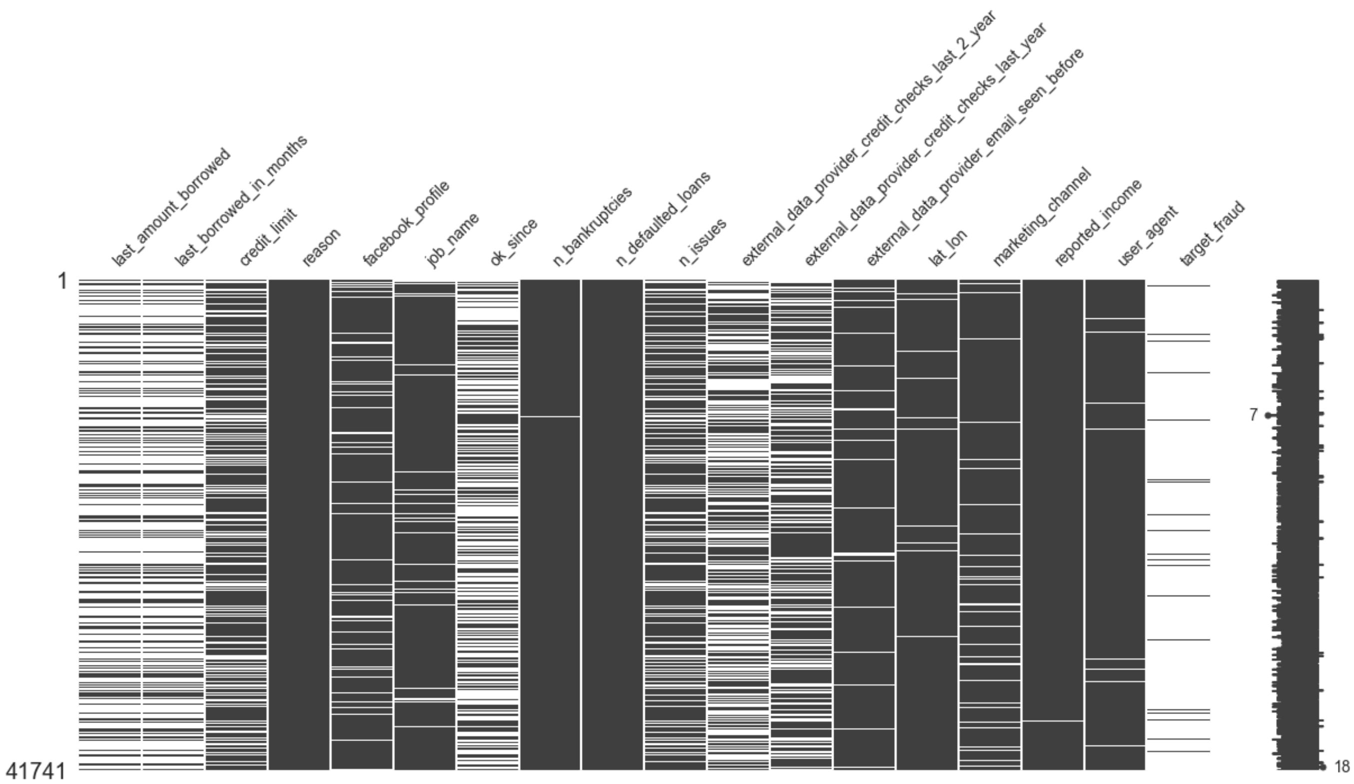
Missing values

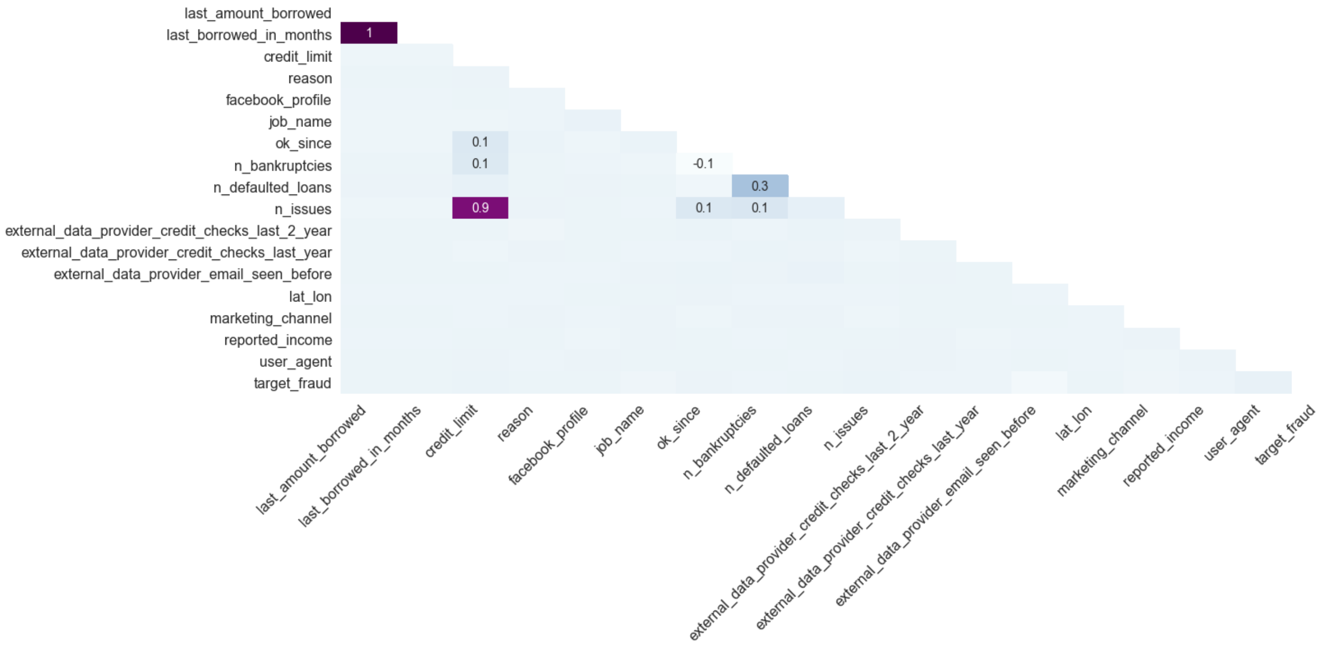
```
missing_value_columns = df.columns[df.isnull().any()].tolist()
df_missing = df[missing_value_columns]

msno.bar(df_missing,figsize=(20,8),color=default_color,fontsize=18,labels=True)
msno.matrix(df_missing,figsize=(20,8),fontsize=14)
msno.heatmap(df_missing,figsize=(20,8),cmap=colormap)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c1d4080f0>
<matplotlib.axes._subplots.AxesSubplot at 0x1c1bdcaa90>
<matplotlib.axes._subplots.AxesSubplot at 0x1c1a773cc0>
```



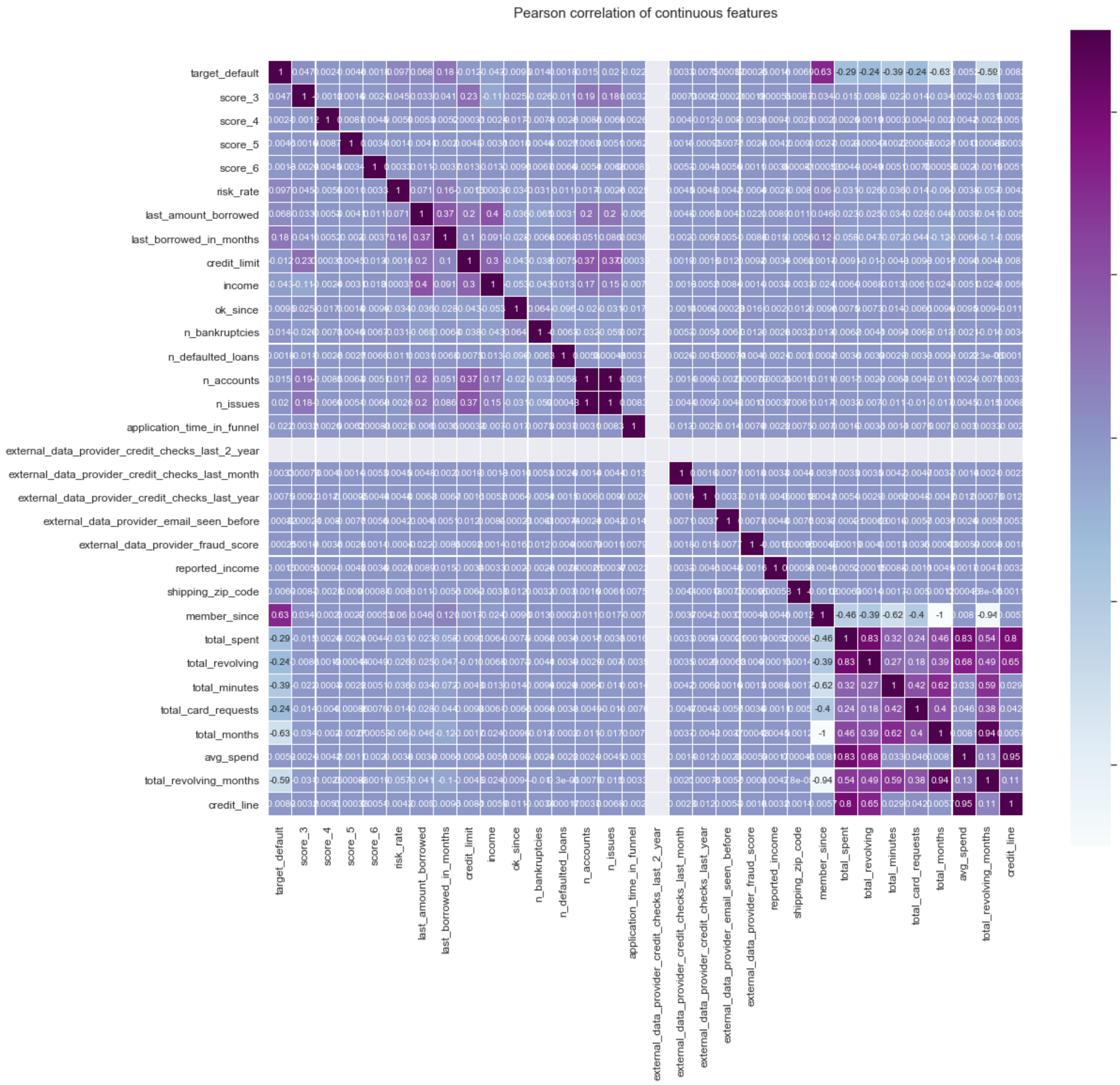




Pearson correlation matrix

```
plt.figure(figsize=(18,16))
plt.title('Pearson correlation of continuous features', y=1.05, size=15)
sns.heatmap(df.corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linec
olor='white', annot=True)
```

```
<Figure size 1296x1152 with 0 Axes>
Text(0.5,1.05,'Pearson correlation of continuous features')
<matplotlib.axes._subplots.AxesSubplot at 0x1c1d221b00>
```


Pearson correlation of continuous features
target_default
score_3
score_4
score_5
score_6
risk_rate
last_amount_borrowed
last_borrowed_in_months
credit_limit
income
ok_since
n_bankruptcies
n_defaulted_loans
n_accounts
n_issues
application_time_in_funnel
external_data_provider_credit_checks_last_2_year
external_data_provider_credit_checks_last_month
external_data_provider_credit_checks_last_year
external_data_provider_email_seen_before
external_data_provider_fraud_score
reported_income
shipping_zip_code
member_since
total_spent
total_revolving
total_minutes
total_card_requests
total_months
avg_spend
total_revolving_months
credit_line
0.8
0.4
0.0
−0.4
−0.8

Drop features

This features do not contribute too much to this model, let's drop them

```
# unecessary columns
drop_cols = [
            'ids', 'credit_limit', 'channel', 'reason', 'job_name', 'reason'
            'external_data_provider_first_name', 'profile_phone_number',
            'target_fraud', 'avg_spend', 'facebook_profile', 'profile_tags',
            'last_amount_borrowed', 'last_borrowed_in_months',
            'zip', 'email', 'user_agent', 'n_issues',
            'application_time_applied', 'application_time_in_funnel',
            'external_data_provider_credit_checks_last_2_year',
            'external_data_provider_credit_checks_last_month',
            'external_data_provider_credit_checks_last_year',
            'external_data_provider_first_name',
            'class', 'member_since', 'credit_line',
            'total_spent', 'total_revolving', 'total_minutes',
            'total_card_requests', 'total_months', 'total_revolving_months']

for col in drop_cols:
    if col in df.columns:
        df.drop(col, axis=1, inplace=True)
```

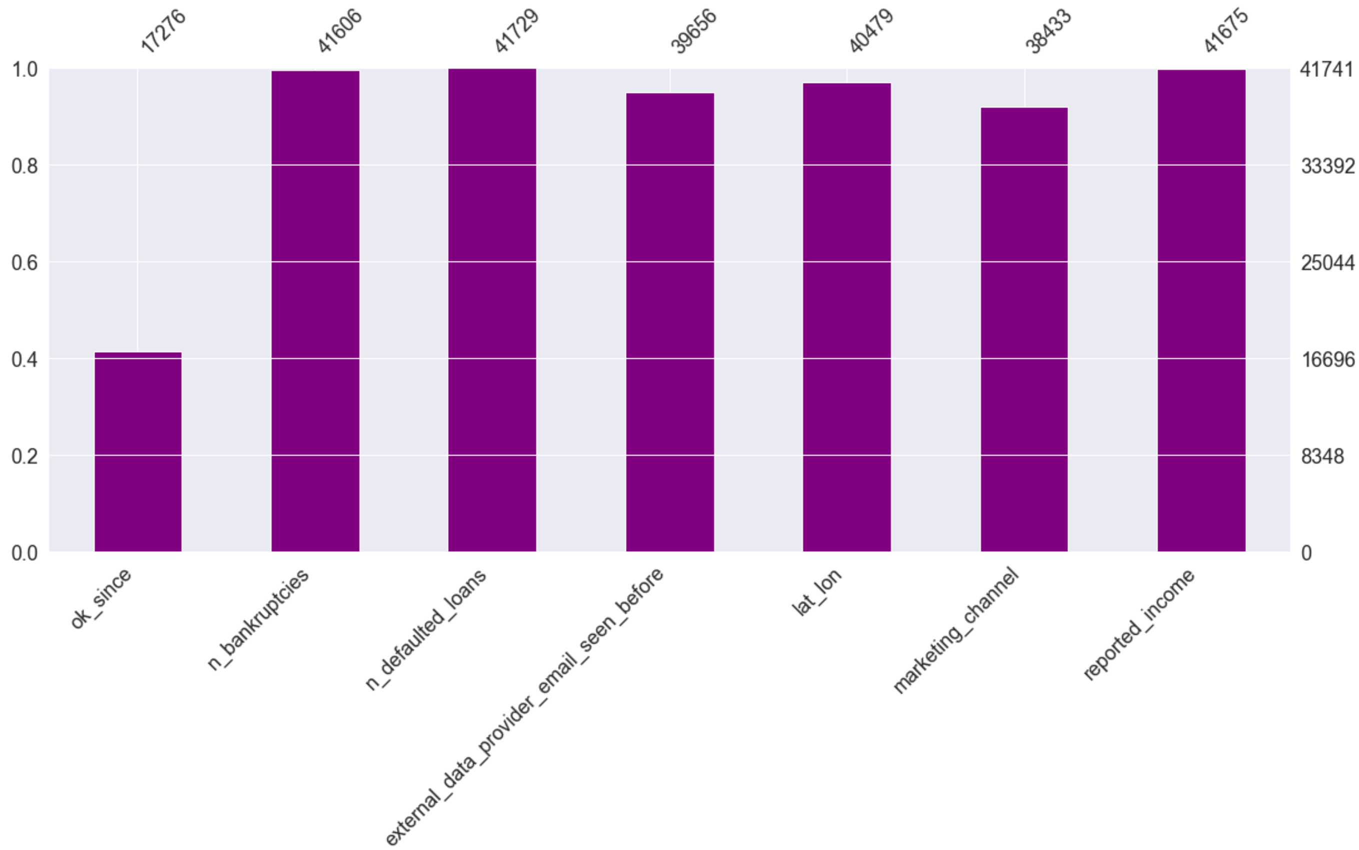
Dealing with Missing values

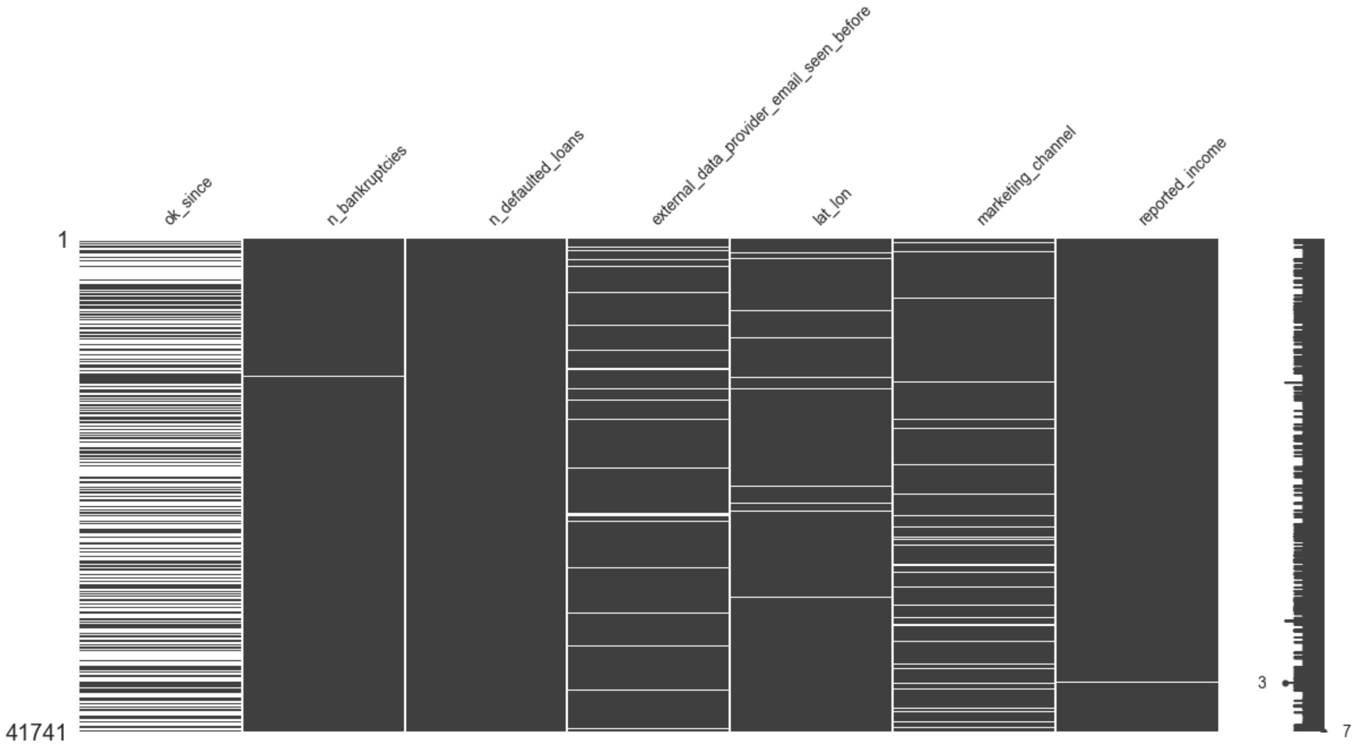
First let's take a look into missing values. Them let's treat each one in the best way possible.

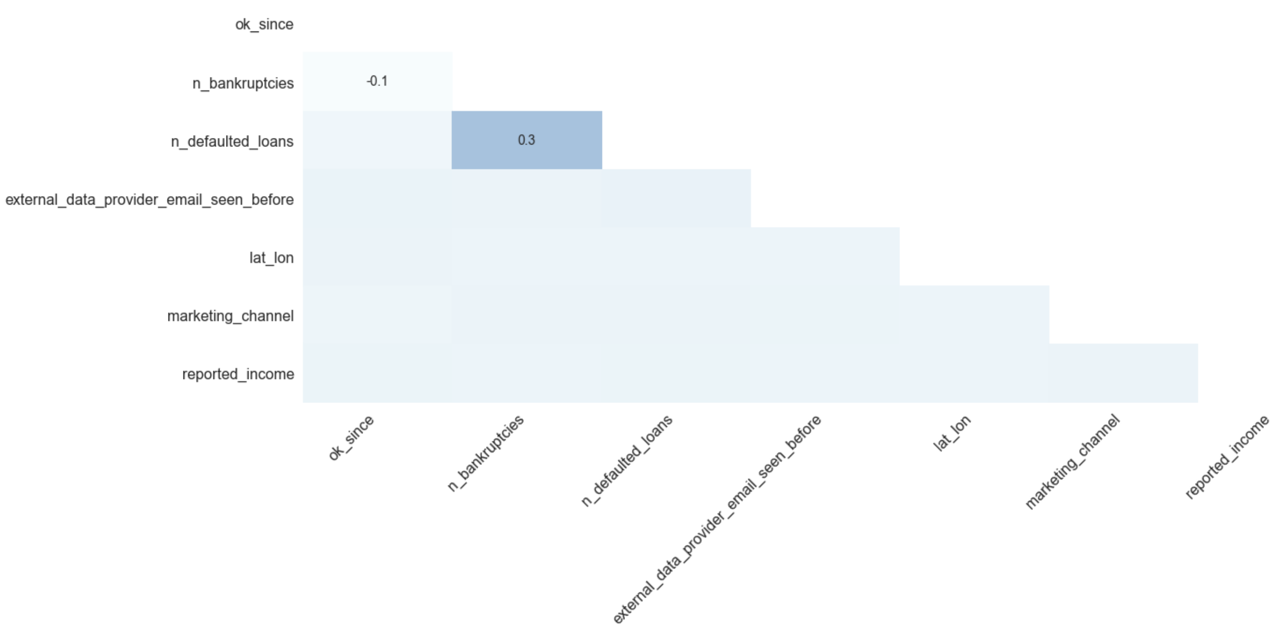
```
missing_value_columns = df.columns[df.isnull().any()].tolist()
df_missing = df[missing_value_columns]

msno.bar(df_missing,figsize=(20,8),color=default_color,fontsize=18,labels=True)
msno.matrix(df_missing,figsize=(20,8),fontsize=14)
msno.heatmap(df_missing,figsize=(20,8),cmap=colormap)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10ba9ca90>
<matplotlib.axes._subplots.AxesSubplot at 0x1c1d6c9710>
<matplotlib.axes._subplots.AxesSubplot at 0x1c1aef1e80>
```




ok_since
n_bankruptcies
n_defaulted_loans
external_data_provider_email_seen_before
lat_lon
marketing_channel
reported_income
1
41741
3
7


ok_since
n_bankruptcies
n_defaulted_loans
external_data_provider_email_seen_before
lat_lon
marketing_channel
reported_income
-0.1
0.3
ok_since
n_bankruptcies
n_defaulted_loans
external_data_provider_email_seen_before
lat_lon
marketing_channel
reported_income

Fill nulls

```
# fill nulls
df['ok_since'].fillna((df['ok_since'].mean()), inplace=True)
df['n_bankruptcies'].fillna(-1, inplace=True)
df['n_defaulted_loans'].fillna(-1, inplace=True)
df['external_data_provider_email_seen_before'].fillna((df['external_data_provide
r_email_seen_before'].mean()), inplace=True)
df['reported_income'].fillna((df['reported_income'].mean()), inplace=True)
df['marketing_channel'].fillna('NA', inplace=True)
df['lat_lon'].fillna('(0,0)', inplace=True)
```

Lat Lon

Let's transform lat_lon into two separate columns

```
# lat lon
df['lat'] = df['lat_lon'].apply(lambda x: ast.literal_eval(x)[0])
df['lon'] = df['lat_lon'].apply(lambda x: ast.literal_eval(x)[1])
df.drop('lat_lon', axis=1, inplace=True)
```

```
missing_value_columns = df.columns[df.isnull().any()].tolist()
if len(missing_value_columns) > 0:
    df_missing = df[missing_value_columns]

    msno.bar(df_missing,figsize=(20,8),color=default_color,fontsize=18,labels=Tr
ue)
    msno.matrix(df_missing,figsize=(20,8),fontsize=14)
    msno.heatmap(df_missing,figsize=(20,8),cmap=colormap)
else:
    print('No Missing values')
```

```
No Missing values
```

Encoding categorical columns

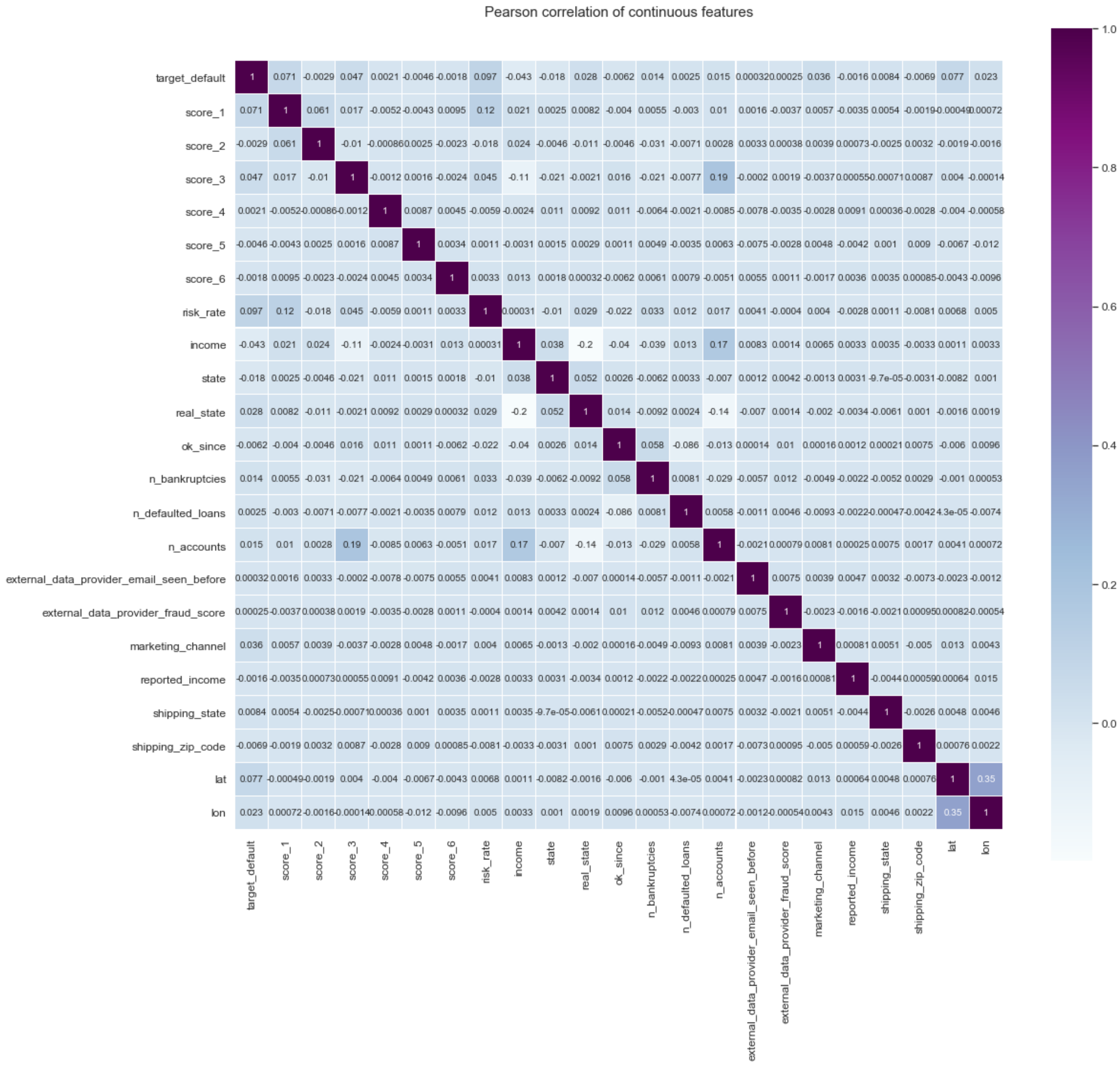
```
encode_columns = [
    'score_1', 'score_2', 'reason', 'state', 'job_name',
    'real_state', 'marketing_channel', 'shipping_state',
    'shipping_zip_code'
]
l_e = LabelEncoder()
for col in encode_columns:
    if col in df.columns:
        df[col] = l_e.fit_transform(df[col])
```

```
plt.figure(figsize=(18,16))
plt.title('Pearson correlation of continuous features', y=1.05, size=15)
sns.heatmap(df.corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linec
olor='white', annot=True)
```

```
<Figure size 1296x1152 with 0 Axes>
```

```
Text(0.5,1.05,'Pearson correlation of continuous features')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c1a884908>
```



Creating X and y and trainning models

```
X = df.drop('target_default',axis=1)
y = df['target_default']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state=42)
```

XGBoost

```
xgb_params = {}

xgb_params['learning_rate'] = 0.01
xgb_params['n_estimators'] = 750
xgb_params['max_depth'] = 6
xgb_params['colsample_bytree'] = 0.6
xgb_params['min_child_weight'] = 0.6
```

```
xgb_model = XGBClassifier(**xgb_params)
```

```
cross_val_model(X_train, y_train, xgb_model)
```

```
Fit XGBClassifier fold 1
    y train:  Counter({0.0: 18663, 1.0: 3598})
    y test:   Counter({0.0: 9332, 1.0: 1799})
    cross_score: 0.71882
[[9189  143]
 [1574  225]]
Fit XGBClassifier fold 2
    y train:  Counter({0.0: 18663, 1.0: 3598})
    y test:   Counter({0.0: 9332, 1.0: 1799})
    cross_score: 0.70471
[[9230  102]
 [1644  155]]
Fit XGBClassifier fold 3
    y train:  Counter({0.0: 18664, 1.0: 3598})
    y test:   Counter({0.0: 9331, 1.0: 1799})
    cross_score: 0.71338
[[9206  125]
 [1595  204]]
```

```
(array([[ 2.0000000e+00,  2.5000000e+01,  3.3000000e+02, ...,
          2.1624000e+04, -1.6772751e+01, -4.3868328e+01],
        [ 3.0000000e+00,  1.5000000e+01,  3.7000000e+02, ...,
          9.1950000e+03, -1.5877224e+01, -5.4987442e+01],
        [ 1.0000000e+00,  3.0000000e+01,  1.6000000e+02, ...,
          2.5688000e+04, -1.5928484e+01, -4.8053417e+01],
        ...,
        [ 2.0000000e+00,  1.4000000e+01,  2.1000000e+02, ...,
          7.8870000e+03, -2.3705956e+01, -5.1252125e+01],
        [ 3.0000000e+00,  9.0000000e+00,  4.4000000e+02, ...,
          1.2501000e+04, -3.1411483e+00, -5.2236736e+01],
        [ 3.0000000e+00,  9.0000000e+00,  2.8000000e+02, ...,
          6.4000000e+03, -3.1265303e+01, -5.4086788e+01]], dtype=float32),
 array([0., 0., 0., ..., 0., 1., 1.], dtype=float32),
 array([[ 3.00000000e+00,  1.60000000e+01,  3.80000000e+02, ...,
          1.33880000e+04, -1.77752533e+01, -5.08819084e+01],
        [ 3.00000000e+00,  1.50000000e+01,  3.80000000e+02, ...,
          9.63900000e+03, -1.41168842e+01, -5.68423309e+01],
        [ 1.00000000e+00,  3.00000000e+01,  2.90000000e+02, ...,
          2.68940000e+04, -2.01292725e+01, -4.45868225e+01],
        ...,
        [ 1.00000000e+00,  3.00000000e+01,  2.70000000e+02, ...,
          1.66660000e+04, -1.48868685e+01, -4.95326424e+01],
        [ 3.00000000e+00,  1.60000000e+01,  1.80000000e+02, ...,
          1.92700000e+03, -3.31989288e+01, -5.30380859e+01],
        [ 0.00000000e+00,  2.80000000e+01,  3.00000000e+02, ...,
          1.30100000e+04, -2.29120903e+01, -4.61198349e+01]], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 1.], dtype=float32))
```

Random Forest

```python
# RandomForest params
rf_params = {}
rf_params['n_estimators'] = 200
rf_params['max_depth'] = 6
rf_params['min_samples_split'] = 70
rf_params['min_samples_leaf'] = 30
```

```python
rf_model = RandomForestClassifier(**rf_params)
```

```python
cross_val_model(X_train, y_train, rf_model)
```

```
Fit RandomForestClassifier fold 1
    y train:  Counter({0.0: 18663, 1.0: 3598})
    y test:   Counter({0.0: 9332, 1.0: 1799})
    cross_score: 0.70805
[[9332    0]
 [1799    0]]
Fit RandomForestClassifier fold 2
    y train:  Counter({0.0: 18663, 1.0: 3598})
    y test:   Counter({0.0: 9332, 1.0: 1799})
    cross_score: 0.67991
[[9332    0]
 [1799    0]]
Fit RandomForestClassifier fold 3
    y train:  Counter({0.0: 18664, 1.0: 3598})
    y test:   Counter({0.0: 9331, 1.0: 1799})
    cross_score: 0.69544
[[9329    2]
 [1796    3]]
```

```
(array([[ 2.0000000e+00,  2.5000000e+01,  3.3000000e+02, ...,
          2.1624000e+04, -1.6772751e+01, -4.3868328e+01],
        [ 3.0000000e+00,  1.5000000e+01,  3.7000000e+02, ...,
          9.1950000e+03, -1.5877224e+01, -5.4987442e+01],
        [ 1.0000000e+00,  3.0000000e+01,  1.6000000e+02, ...,
          2.5688000e+04, -1.5928484e+01, -4.8053417e+01],
        ...,
        [ 2.0000000e+00,  1.4000000e+01,  2.1000000e+02, ...,
          7.8870000e+03, -2.3705956e+01, -5.1252125e+01],
        [ 3.0000000e+00,  9.0000000e+00,  4.4000000e+02, ...,
          1.2501000e+04, -3.1411483e+00, -5.2236736e+01],
        [ 3.0000000e+00,  9.0000000e+00,  2.8000000e+02, ...,
          6.4000000e+03, -3.1265303e+01, -5.4086788e+01]], dtype=float32),
 array([0., 0., 0., ..., 0., 1., 1.], dtype=float32),
 array([[ 3.00000000e+00,  1.60000000e+01,  3.80000000e+02, ...,
          1.33880000e+04, -1.77752533e+01, -5.08819084e+01],
        [ 3.00000000e+00,  1.50000000e+01,  3.80000000e+02, ...,
          9.63900000e+03, -1.41168842e+01, -5.68423309e+01],
        [ 1.00000000e+00,  3.00000000e+01,  2.90000000e+02, ...,
          2.68940000e+04, -2.01292725e+01, -4.45868225e+01],
        ...,
        [ 1.00000000e+00,  3.00000000e+01,  2.70000000e+02, ...,
          1.66660000e+04, -1.48868685e+01, -4.95326424e+01],
        [ 3.00000000e+00,  1.60000000e+01,  1.80000000e+02, ...,
          1.92700000e+03, -3.31989288e+01, -5.30380859e+01],
        [ 0.00000000e+00,  2.80000000e+01,  3.00000000e+02, ...,
          1.30100000e+04, -2.29120903e+01, -4.61198349e+01]], dtype=float32),
 array([0., 0., 0., ..., 0., 0., 1.], dtype=float32))
```

Stacked models

Why not use both together?

```
log_model = LogisticRegression()
```

```
stack = Ensemble(n_splits=3,
        stacker = log_model,
        base_models = (rf_model, xgb_model))
```

```
y_pred = stack.fit_predict(X_train, y_train, X)
```

```
Fit RandomForestClassifier fold 1
Fit RandomForestClassifier fold 2
Fit RandomForestClassifier fold 3
Fit XGBClassifier fold 1
Fit XGBClassifier fold 2
Fit XGBClassifier fold 3
Stacker score: 0.72794
[[27459   536]
 [ 4541   856]]
```

```
y_pred = stack.predict(X_test)
confusion_matrix(y_test, np.rint(y_pred))
```

```
array([[6957,  128],
       [1055,  209]])
```

Imbalanced learning

Let's balance this dataset using the technique called SMOTE technique (https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis#SMOTE (https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis#SMOTE))

```
X = df.drop('target_default',axis=1)
y = df['target_default']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state=42)
```

```
smt = SMOTE(random_state=42, k_neighbors=1)
X_SMOTE, y_SMOTE = smt.fit_sample(X_train, y_train)
```

```python
import pickle
stack.fit(X_SMOTE, y_SMOTE)
model_name = 'default_ensemble.pkl'
with open(model_name, 'wb') as model_file:
    pickle.dump(stack, model_file)
```

```
Fit RandomForestClassifier fold 1
Fit RandomForestClassifier fold 2
Fit RandomForestClassifier fold 3
Fit XGBClassifier fold 1
Fit XGBClassifier fold 2
Fit XGBClassifier fold 3
Stacker score: 0.94480
[[26881  1114]
 [ 4985 23010]]
```

```python
with open(model_name, 'rb') as model_file:
    stack2 = pickle.load(model_file)
y_pred = stack2.predict(X_test)
confusion_matrix(y_test, np.rint(y_pred))
```

```
array([[6803,  282],
       [1019,  245]])
```