Nubank data challenge - Spend 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/'
```

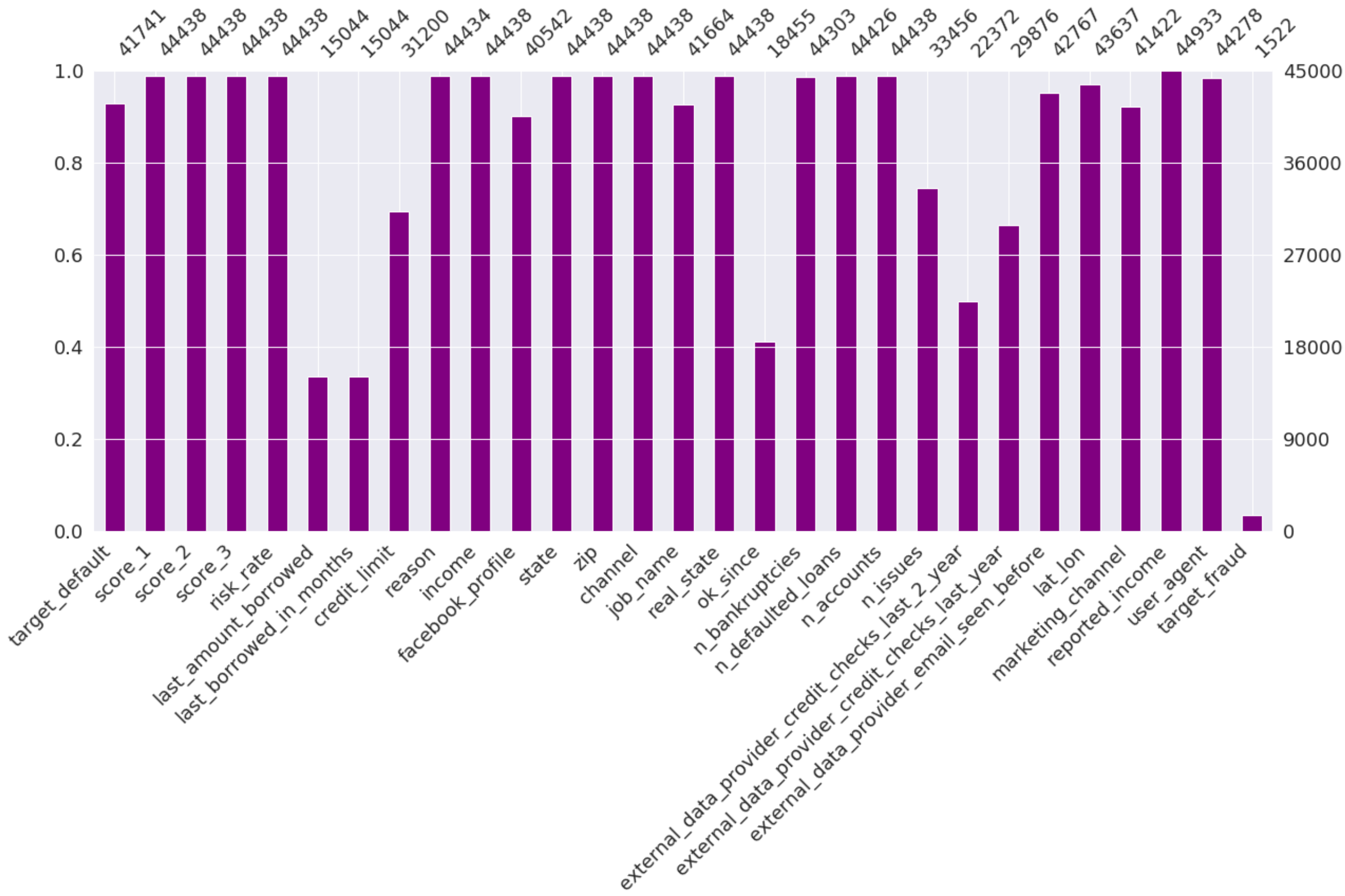
```
df_name = new_data_path + 'acquisition_train.csv'
```

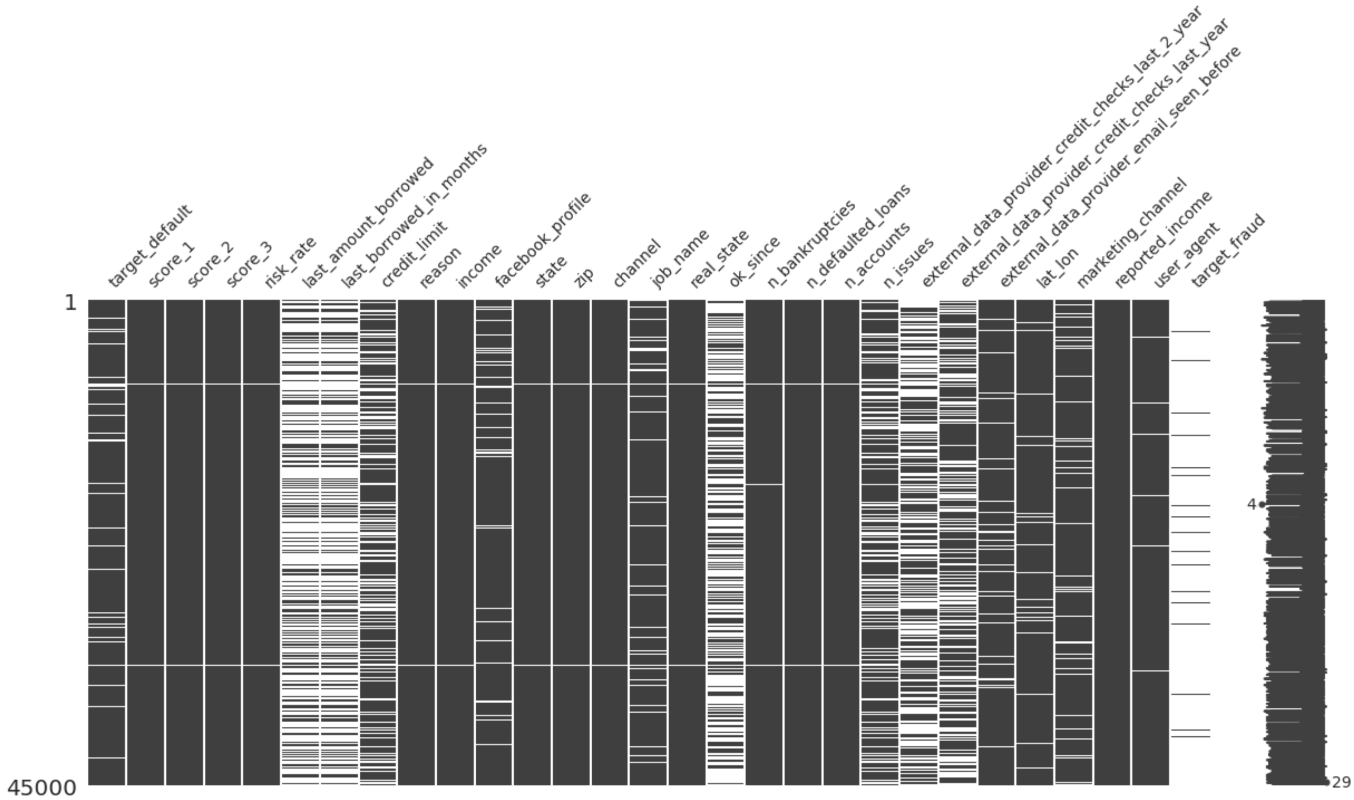
```
df = pd.read_csv(df_name)
df.shape
df.info()
df.describe()
df.head()
```

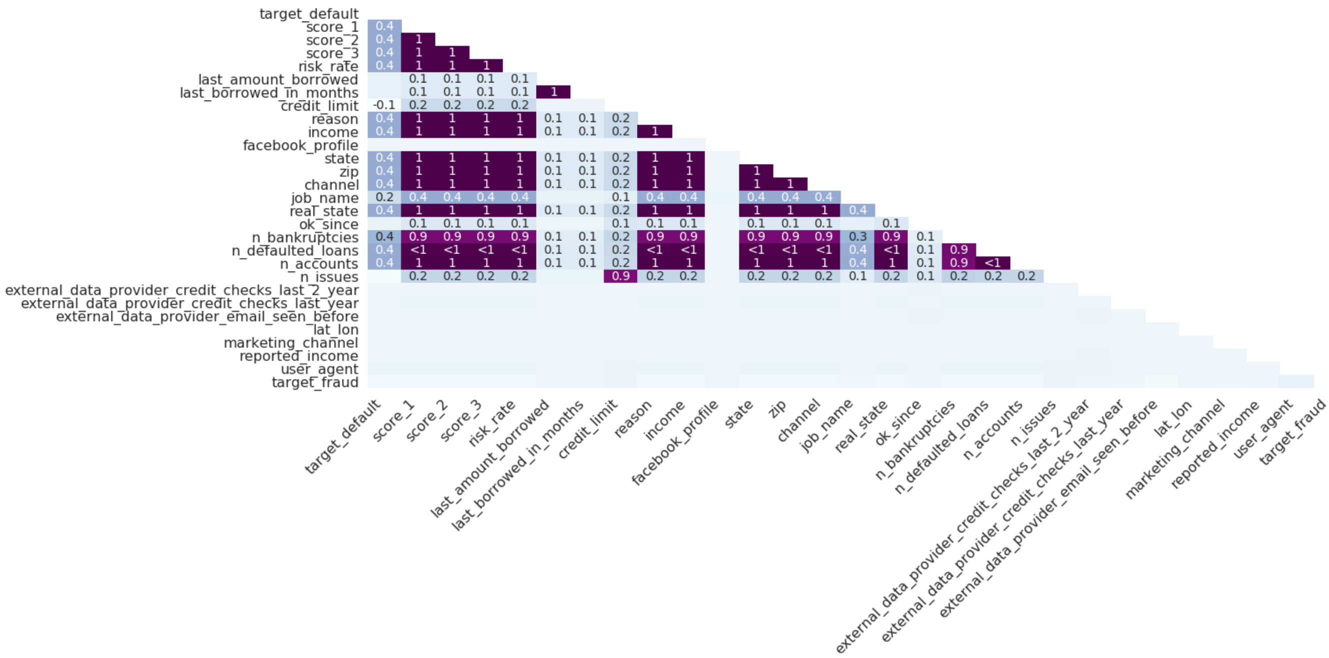
Missing values

```
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 0x7f28079340f0>

<matplotlib.axes._subplots.AxesSubplot at 0x7f2806d73dd8>

<matplotlib.axes._subplots.AxesSubplot at 0x7f2806a29b70>
```







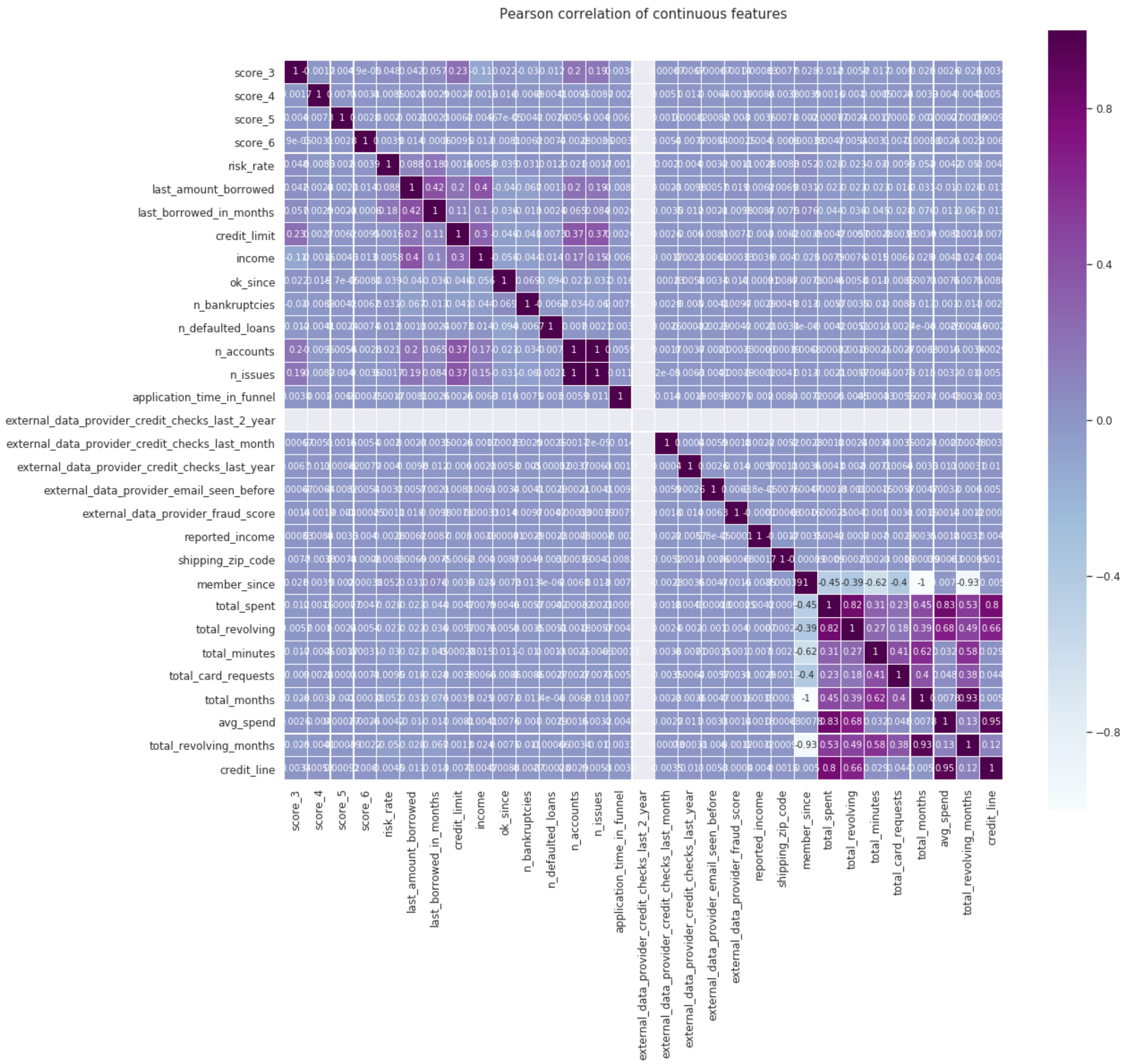
Pearson correlation matrix

```
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 0x7f28049f7a58>
```



Drop features

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

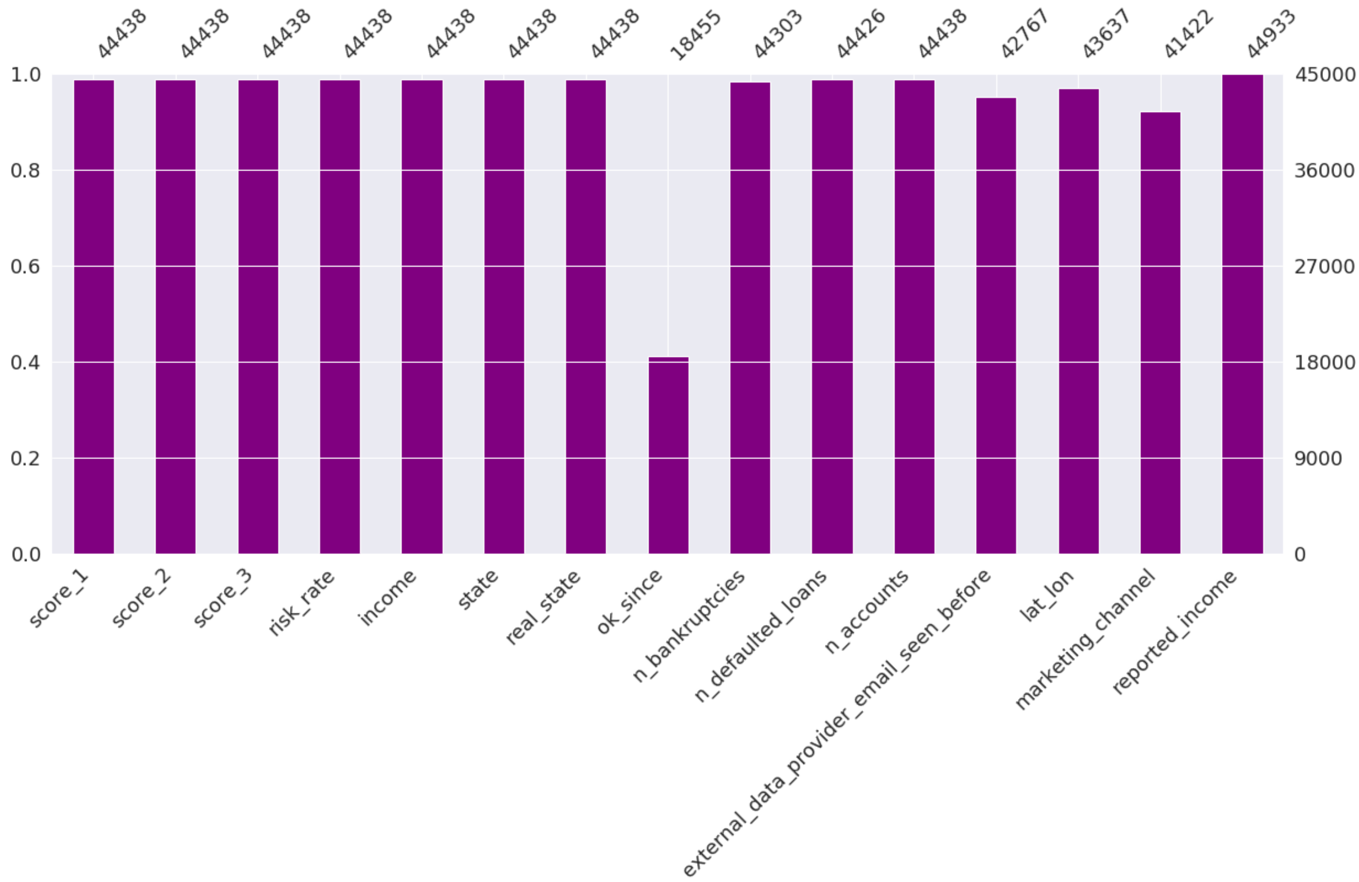
```python
# unecessary columns
drop_cols = [
            'ids', 'credit_limit', 'channel', 'reason', 'job_name', 'reason'
            'external_data_provider_first_name', 'profile_phone_number',
            'target_fraud', 'target_default', 'facebook_profile', 'profile_tags'
,
            'last_amount_borrowed', '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)
```

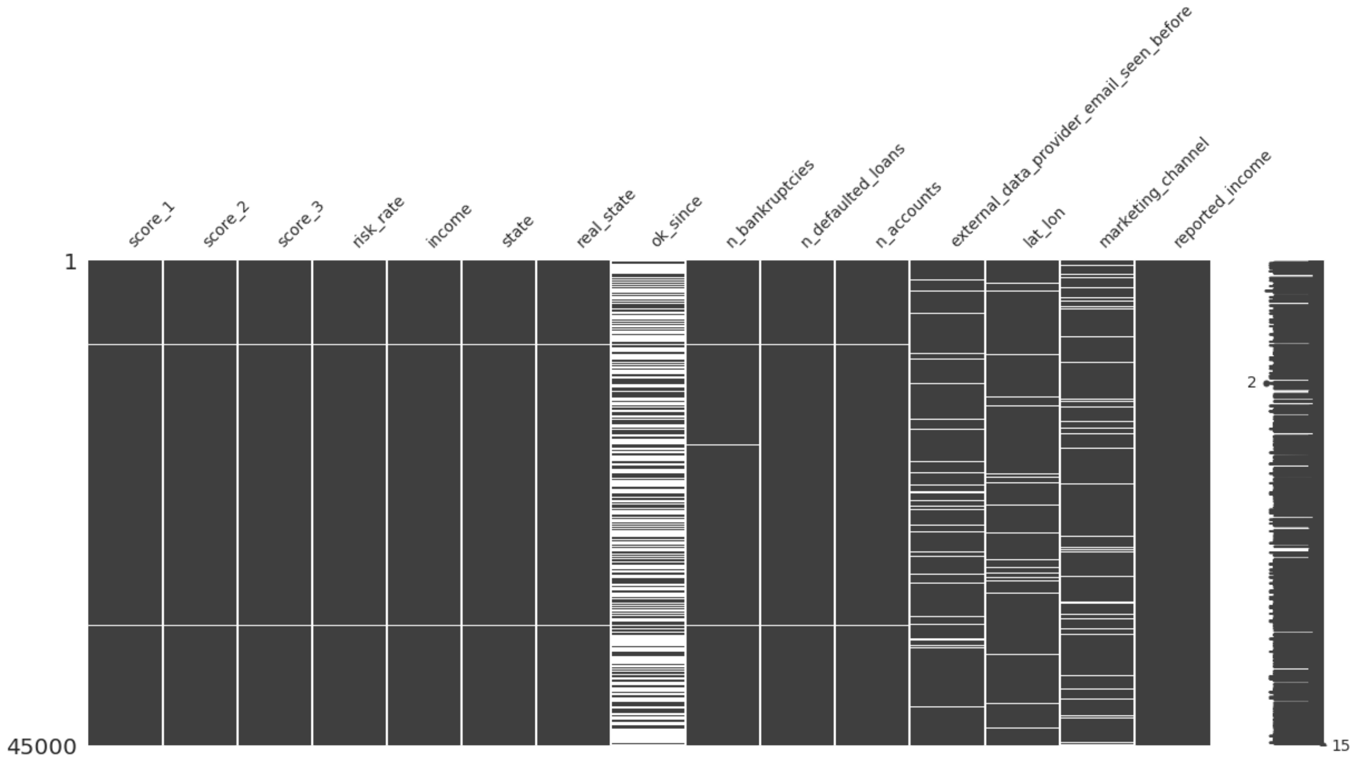
Dealing with Missing values

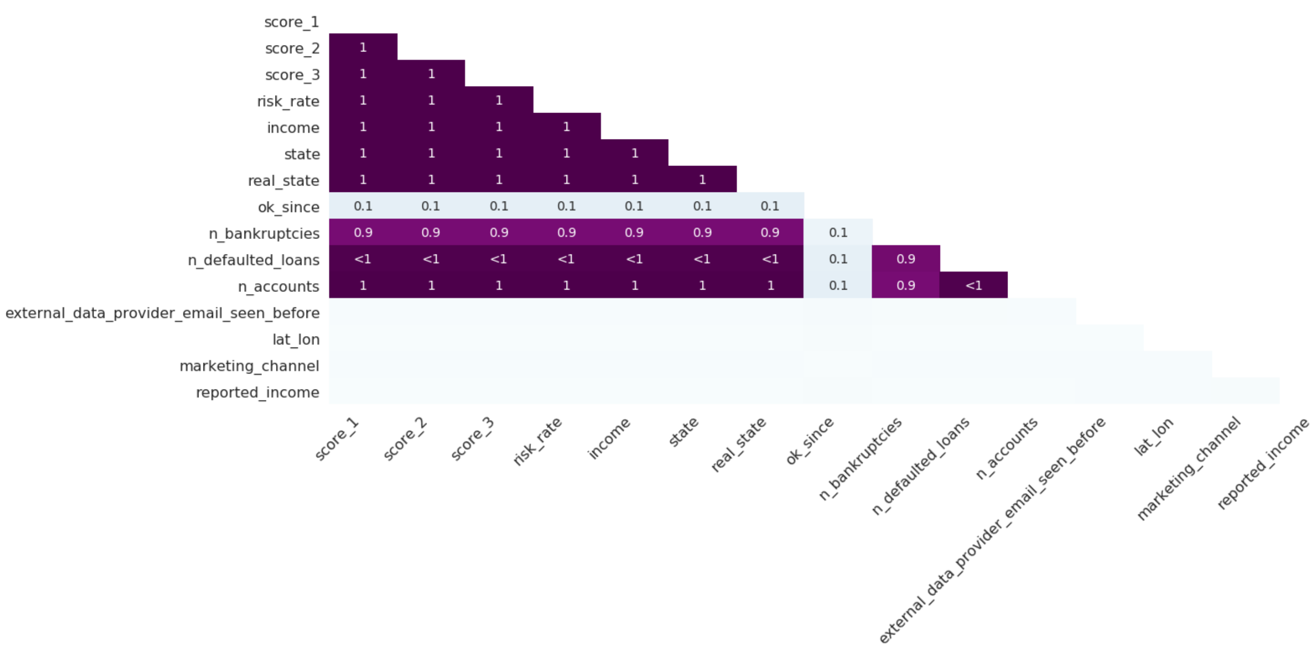
First let's take a look into missing values. Them let's treat each one in the best way possible.

```
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 0x7f27fe8e1320>
<matplotlib.axes._subplots.AxesSubplot at 0x7f27fe776898>
<matplotlib.axes._subplots.AxesSubplot at 0x7f27fe4e1c18>
```




score_1
score_2
score_3
risk_rate
income
state
real_state
ok_since
n_bankruptcies
n_defaulted_loans
n_accounts
external_data_provider_email_seen_before
lat_lon
marketing_channel
reported_income
1
45000
2
15


score_1
score_2
score_3
risk_rate
income
state
real_state
ok_since
n_bankruptcies
n_defaulted_loans
n_accounts
external_data_provider_email_seen_before
lat_lon
marketing_channel
reported_income
1
0.1
0.9
<1

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

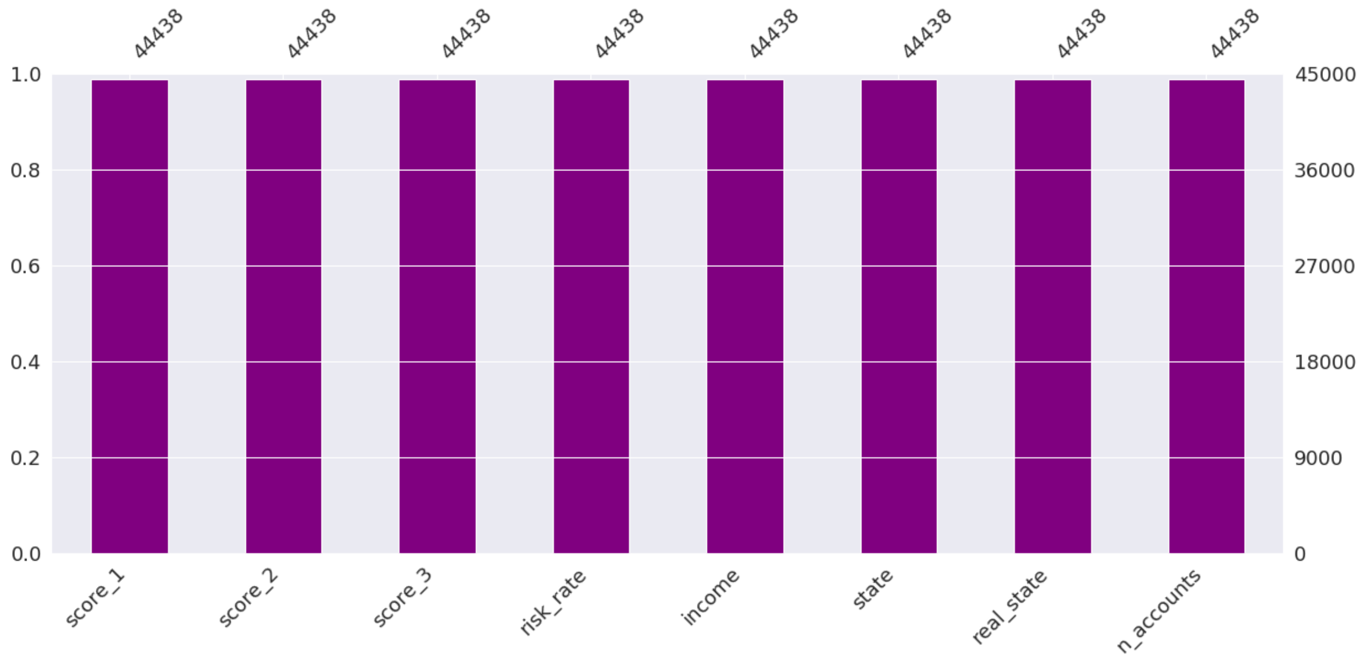
```
# lat lon
df['lat'] = df['lat_lon'].apply(lambda x: ast.literal_eval(x)[0])
df['lon'] = df['lat_lon'].apply(lambda x: ast.literal_eval(x)[1])
df.drop('lat_lon', axis=1, inplace=True)
```

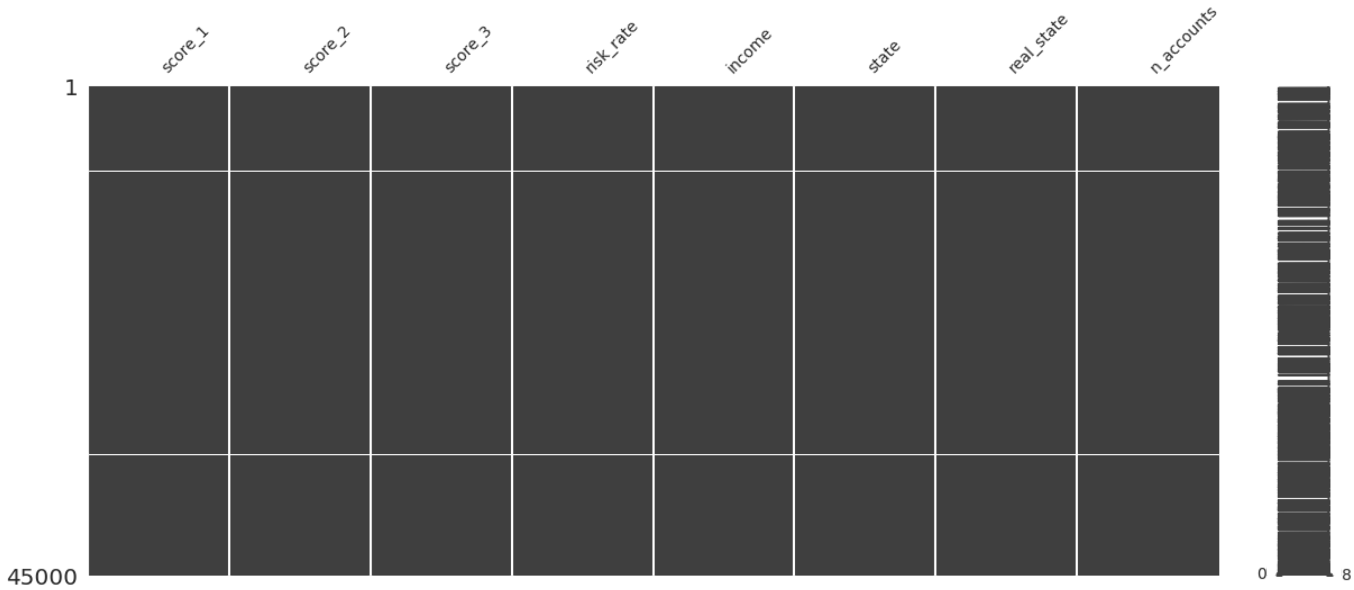
Drop the rest of missing values

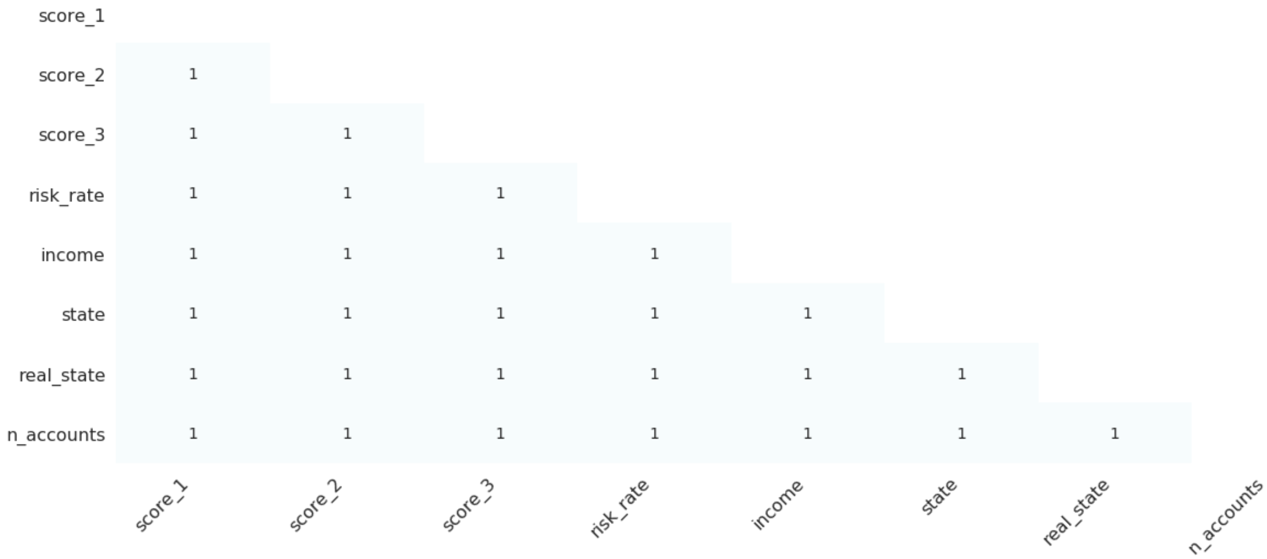
First let's take a look into missing values. Them let's treat each one in the best way possible.

```
missing_value_columns = df.columns[df.isnull().any()].tolist()
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=True)
    msno.matrix(df_missing,figsize=(20,8),fontsize=14)
    msno.heatmap(df_missing,figsize=(20,8),cmap=colormap)
else:
    print('No Missing values')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f27fe6dd438>

<matplotlib.axes._subplots.AxesSubplot at 0x7f27fe56d940>

<matplotlib.axes._subplots.AxesSubplot at 0x7f27fc247198>
```








```
df.dropna(inplace=True)
df.shape
```

```
(44438, 23)
```

Encoding categorical columns

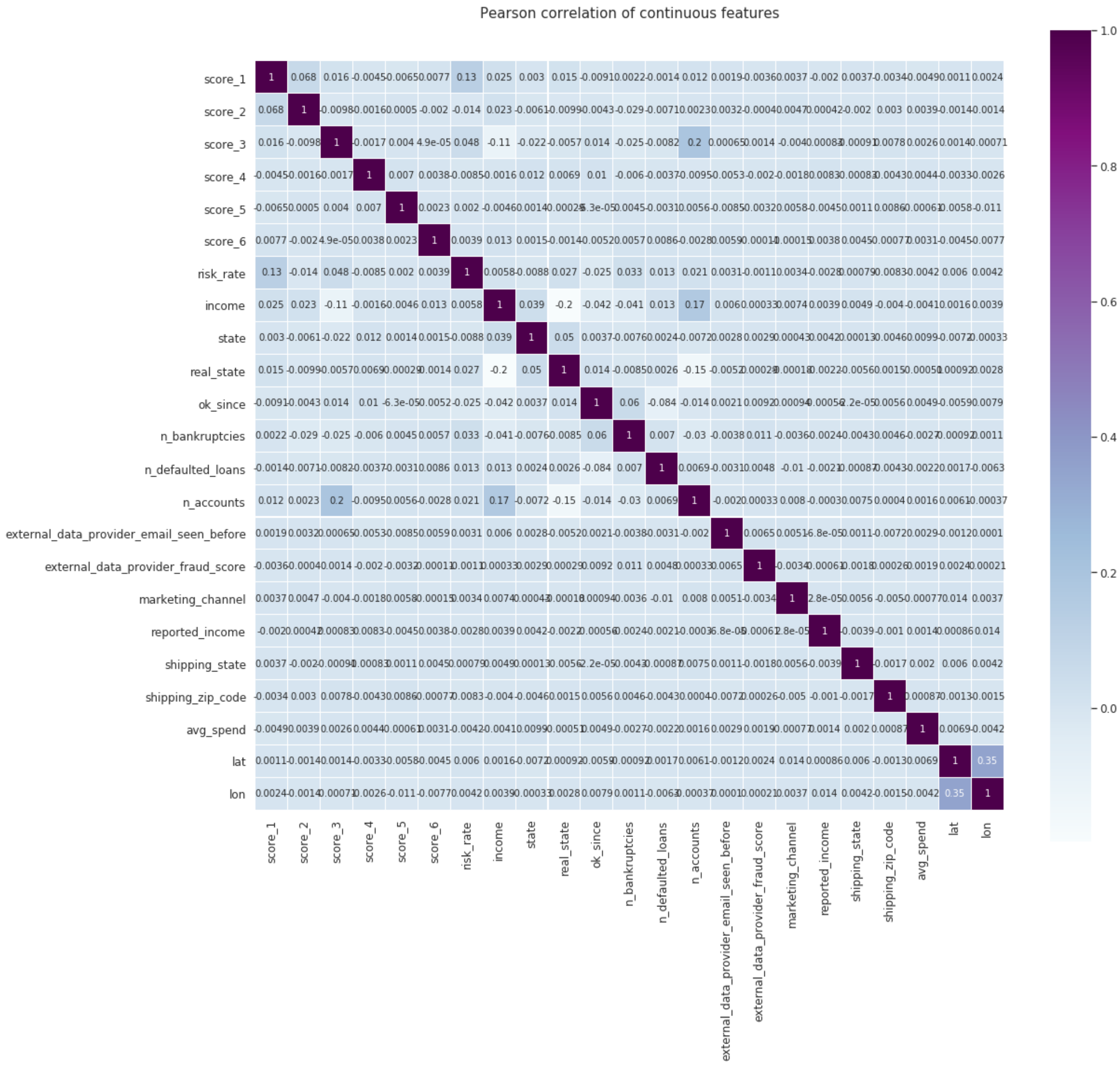
```
encode_columns = [
    'score_1', 'score_2', 'reason', 'state', 'job_name',
    'real_state', 'marketing_channel', 'shipping_state',
    'shipping_zip_code'
]
l_e = LabelEncoder()
for col in encode_columns:
    if col in df.columns:
        df[col] = l_e.fit_transform(df[col])
```

```
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 0x7f27fe5ebba8>
```




```
one_hot = {c: list(df[c].unique()) for c in df.columns if c not in ['target_defa
ult']}
df = OHE_by_unique(df, one_hot, 7)
```

Creating X and y and trainning models

```
X = df.drop('avg_spend',axis=1)
y = df['avg_spend']
```

Linear Regression

```
linear_params = {}
linear_params['normalize'] = True
linear_params['n_jobs'] = -1

reg = LinearRegression(**linear_params)
X_train, y_train, X_val, y_val = cross_val_model(X, y, reg, scoring='neg_mean_sq
uared_error', model_type='reg')
```

```
Fit LinearRegression fold 1
    cross_score: -19689012117.50000
Fit LinearRegression fold 2
    cross_score: -11367853165384.00000
Fit LinearRegression fold 3
    cross_score: -62276190203.33334
```

```
min(y_val)
max(y_val)
y_pred = reg.predict(X_val)
min(y_pred)
max(y_pred)
y_val[:10]
y_pred[:10]
```

```
38.836666

15705.142

2000.0

421815600.0

array([ 634.2636 ,  835.9883 , 4228.071  ,  905.56274, 4068.5205 ,
       4486.4966 , 1797.4261 , 2745.9666 , 1910.112  , 2683.6367 ],
      dtype=float32)

array([2952., 2920., 2840., 2824., 2888., 2936., 2856., 3008., 2944.,
       2840.], dtype=float32)
```

Decision tree regression

This was chosen for having a better performance

```
dtr = DecisionTreeRegressor(max_features='auto', random_state=7)
X_train, y_train, X_val, y_val = cross_val_model(X, y, dtr, scoring='neg_mean_sq
uared_error', model_type='reg')
```

```
Fit DecisionTreeRegressor fold 1
    cross_score: -9760418.93444
Fit DecisionTreeRegressor fold 2
    cross_score: -9734089.53173
Fit DecisionTreeRegressor fold 3
    cross_score: -9548743.89521
```

```
min(y_val)
max(y_val)
y_pred = dtr.predict(X_val)
min(y_pred)
max(y_pred)
y_val[:10]
y_pred[:10]
```

```
38.836666

15705.142

90.5199966430664

14847.857421875

array([ 634.2636 ,  835.9883 , 4228.071  ,  905.56274, 4068.5205 ,
       4486.4966 , 1797.4261 , 2745.9666 , 1910.112  , 2683.6367 ],
      dtype=float32)

array([1596.96313477,  720.61883545, 3346.49633789, 7510.30126953,
       6492.15185547, 5612.31591797, 3740.4753418 ,  436.84268188,
       2522.01220703, 3019.62548828])
```