DATA OVERVIEW

rank: The ranking of the billionaire in terms of wealth.
finalWorth: The final net worth of the billionaire in U.S. dollars.
category: The category or industry in which the billionaire's business operates.
personName: The full name of the billionaire.
age: The age of the billionaire.
country: The country in which the billionaire resides.
city: The city in which the billionaire resides.
source: The source of the billionaire's wealth.
industries: The industries associated with the billionaire's business interests.
countryOfCitizenship: The country of citizenship of the billionaire.
organization: The name of the organization or company associated with the billionaire.
selfMade: Indicates whether the billionaire is self-made (True/False).
status: "D" represents self-made billionaires (Founders/Entrepreneurs) and "U" indicates inherited or unearned wealth.
gender: The gender of the billionaire.
birthDate: The birthdate of the billionaire.
lastName: The last name of the billionaire.
firstName: The first name of the billionaire.
title: The title or honorific of the billionaire.
date: The date of data collection.
state: The state in which the billionaire resides.
residenceStateRegion: The region or state of residence of the billionaire.
birthYear: The birth year of the billionaire.
birthMonth: The birth month of the billionaire.
birthDay: The birth day of the billionaire.
cpi_country: Consumer Price Index (CPI) for the billionaire's country.
cpi_change_country: CPI change for the billionaire's country.
gdp_country: Gross Domestic Product (GDP) for the billionaire's country.
gross_tertiary_education_enrollment: Enrollment in tertiary education in the billionaire's country.
gross_primary_education_enrollment_country: Enrollment in primary education in the billionaire's country.
life_expectancy_country: Life expectancy in the billionaire's country.
tax_revenue_country_country: Tax revenue in the billionaire's country.
total_tax_rate_country: Total tax rate in the billionaire's country.

population_country: Population of the billionaire's country.
latitude_country: Latitude coordinate of the billionaire's country.
longitude_country: Longitude coordinate of the billionaire's country.

In [5]:
```
#Load the required modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import matplotlib
from plotly.subplots import make_subplots
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import VarianceThreshold

%matplotlib inline
```

In [12]:

```
#Data Loading

df_full = pd.read_csv("Billionaires Statistics Dataset.csv")
rows=df_full.shape[0]
cols=df_full.shape[1]
print("The Billionaires Statistics Dataset stats:\n ")
print(f"The number of rows are :{rows} \nThe number of columns are : {cols}")
#Tranpose the head
df_full.head().T
```

```
The Billionaires Statistics Dataset stats:

The number of rows are :2640
The number of columns are : 35
```

Out[12]:

	0	1	2	3	4
rank	1	2	3	4	5
finalWorth	211000	180000	114000	107000	106000
category	Fashion & Retail	Automotive	Technology	Technology	Finance & Investments
personName	Bernard Arnault & family	Elon Musk	Jeff Bezos	Larry Ellison	Warren Buffett
age	74.00	51.00	59.00	78.00	92.00
country	France	United States	United States	United States	United States
city	Paris	Austin	Medina	Lanai	Omaha
source	LVMH	Tesla, SpaceX	Amazon	Oracle	Berkshire Hathaway
industries	Fashion & Retail	Automotive	Technology	Technology	Finance & Investments
countryOfCitizenship	France	United States	United States	United States	United States
organization	LVMH Moët Hennessy Louis Vuitton	Tesla	Amazon	Oracle	Berkshire Hathaway Inc. (Cl A)
selfMade	False	True	True	True	True
status	U	D	D	U	D
gender	M	M	M	M	M
birthDate	3/5/1949 0:00	6/28/1971 0:00	1/12/1964 0:00	8/17/1944 0:00	8/30/1930 0:00
lastName	Arnault	Musk	Bezos	Ellison	Buffett
firstName	Bernard	Elon	Jeff	Larry	Warren
title	Chairman and CEO	CEO	Chairman and Founder	CTO and Founder	CEO
date	4/4/2023 5:01	4/4/2023 5:01	4/4/2023 5:01	4/4/2023 5:01	4/4/2023 5:01
state	NaN	Texas	Washington	Hawaii	Nebraska
residenceStateRegion	NaN	South	West	West	Midwest
birthYear	1,949.00	1,971.00	1,964.00	1,944.00	1,930.00

	0	1	2	3	4
birthMonth	3.00	6.00	1.00	8.00	8.00
birthDay	5.00	28.00	12.00	17.00	30.00
cpi_country	110.05	117.24	117.24	117.24	117.24
cpi_change_country	1.10	7.50	7.50	7.50	7.50
gdp_country	$2,715,518,274,227	$21,427,700,000,000	$21,427,700,000,000	$21,427,700,000,000	$21,427,700,000,000
gross_tertiary_education_enrollment	65.60	88.20	88.20	88.20	88.20
gross_primary_education_enrollment_country	102.50	101.80	101.80	101.80	101.80
life_expectancy_country	82.50	78.50	78.50	78.50	78.50
tax_revenue_country_country	24.20	9.60	9.60	9.60	9.60
total_tax_rate_country	60.70	36.60	36.60	36.60	36.60
population_country	67,059,887.00	328,239,523.00	328,239,523.00	328,239,523.00	328,239,523.00
latitude_country	46.23	37.09	37.09	37.09	37.09
longitude_country	2.21	-95.71	-95.71	-95.71	-95.71

In [7]:

```
df_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2640 entries, 0 to 2639
Data columns (total 35 columns):
 #   Column                                    Non-Null Count  Dtype  
---  ------                                    --------------  -----  
 0   rank                                      2640 non-null   int64  
 1   finalWorth                                2640 non-null   int64  
 2   category                                  2640 non-null   object 
 3   personName                                2640 non-null   object 
 4   age                                       2575 non-null   float64
 5   country                                   2602 non-null   object 
 6   city                                      2568 non-null   object 
 7   source                                    2640 non-null   object 
 8   industries                                2640 non-null   object 
 9   countryOfCitizenship                      2640 non-null   object 
 10  organization                              325 non-null    object 
 11  selfMade                                  2640 non-null   bool   
 12  status                                    2640 non-null   object 
 13  gender                                    2640 non-null   object 
 14  birthDate                                 2564 non-null   object 
 15  lastName                                  2640 non-null   object 
 16  firstName                                 2637 non-null   object 
 17  title                                     339 non-null    object 
 18  date                                      2640 non-null   object 
 19  state                                     753 non-null    object 
 20  residenceStateRegion                      747 non-null    object 
 21  birthYear                                 2564 non-null   float64
 22  birthMonth                                2564 non-null   float64
 23  birthDay                                  2564 non-null   float64
 24  cpi_country                               2456 non-null   float64
 25  cpi_change_country                        2456 non-null   float64
 26  gdp_country                               2476 non-null   object 
 27  gross_tertiary_education_enrollment       2458 non-null   float64
 28  gross_primary_education_enrollment_country  2459 non-null   float64
 29  life_expectancy_country                   2458 non-null   float64
 30  tax_revenue_country_country               2457 non-null   float64
 31  total_tax_rate_country                    2458 non-null   float64
 32  population_country                        2476 non-null   float64
 33  latitude_country                          2476 non-null   float64
 34  longitude_country                         2476 non-null   float64
```

```
dtypes: bool(1), float64(14), int64(2), object(18)
memory usage: 704.0+ KB
```

In [13]:

```
#Data Statistics

pd.options.display.float_format = '{:,.2f}'.format
def dataStat(df):
    sel_cols = df.select_dtypes(include=['float64','int64'])
    data_box = pd.DataFrame(sel_cols.dtypes, columns=['data type'])
    data_box['missing_val'] = sel_cols.isnull().sum().values
    data_box['missing_perc'] = sel_cols.isnull().sum().values / len(df) * 100
    data_box['unique_val'] = sel_cols.nunique().values
    desc = pd.DataFrame(sel_cols.describe(include='number').transpose())
    data_box['min'] = desc['min'].values
    data_box['max'] = desc['max'].values
    data_box['avg'] = desc['mean'].values
    data_box['std_dev'] = desc['std'].values

    return data_box


dataStat(df_full)
```

Out[13]:

	data type	missing_val	missing_perc	unique_val	min	max	avg	std_
rank	int64	0	0.00	219	1.00	2,540.00	1,289.16	739
finalWorth	int64	0	0.00	219	1,000.00	211,000.00	4,623.79	9,834
age	float64	65	2.46	79	18.00	101.00	65.14	13
birthYear	float64	76	2.88	77	1,921.00	2,004.00	1,957.18	13
birthMonth	float64	76	2.88	12	1.00	12.00	5.74	3
birthDay	float64	76	2.88	31	1.00	31.00	12.10	9
cpi_country	float64	184	6.97	63	99.55	288.57	127.76	26
cpi_change_country	float64	184	6.97	44	-1.90	53.50	4.36	3
gross_tertiary_education_enrollment	float64	182	6.89	63	4.00	136.60	67.23	21
gross_primary_education_enrollment_country	float64	181	6.86	60	84.70	142.10	102.86	4
life_expectancy_country	float64	182	6.89	54	54.30	84.20	78.12	3
tax_revenue_country_country	float64	183	6.93	57	0.10	37.20	12.55	5
total_tax_rate_country	float64	182	6.89	63	9.90	106.30	43.96	12
population_country	float64	164	6.21	68	38,019.00	1,397,715,000.00	510,205,317.84	554,244,732
latitude_country	float64	164	6.21	68	-40.90	61.92	34.90	17
longitude_country	float64	164	6.21	68	-106.35	174.89	12.58	86

In [14]:

```python
#Check the data which have object and boolean datatype

df_full.select_dtypes(include=['object','bool']).describe().T
```

Out[14]:

	count	unique	top	freq
category	2640	18	Finance & Investments	372
personName	2640	2638	Wang Yanqing & family	2
country	2602	78	United States	754
city	2568	741	New York	99
source	2640	906	Real estate	151
industries	2640	18	Finance & Investments	372
countryOfCitizenship	2640	77	United States	735
organization	325	294	Meta Platforms	4
selfMade	2640	2	True	1812
status	2640	6	D	1223
gender	2640	2	M	2303
birthDate	2564	2060	1/1/1965 0:00	19
lastName	2640	1736	Li	44
firstName	2637	1770	John	40
title	339	97	Investor	44
date	2640	2	4/4/2023 5:01	2638
state	753	45	California	178
residenceStateRegion	747	5	West	248
gdp_country	2476	68	$21,427,700,000,000	754

Observations

Date has only two values , hence we can drop

gdp_country has object data type , has $ needs conversion

birth date has object type datatype , need to convert to timestamp
Country and country of Citizenship same data , hence we can drop country of Citizenship

Irrevelant Data
date
personName
city
source
industries
countryOfCitizenship
organization
lastName
firstName
title

DATA CLEANING

In [15]:
```
#Remove Irrevelant columns and data
df_full.drop(columns=['date','personName','city','source','industries','countryOfCitizenship','organization','las
                  'firstName', 'title','residenceStateRegion'],inplace=True)
```

In [16]:
```
#Find the Missing Data in the columns

print("There are {} missing values in the processed dataset.".format(df_full.isna().sum().sum()))
```

```
There are 4228 missing values in the processed dataset.
```

In [17]:

```
# The amount of data missing is huge i.e 4228 , hence deletion cannot be an option
#Since the data is ranked reguarly in order , we can fill the data by the last valid non missing data
# i.e. using forward fill method




for col in df_full:
    df_full[col]=df_full[col].fillna(method='ffill')

print("There are still {} missing values in the processed dataset.".format(df_full.isna().sum().sum()))
```

```
There are still 1 missing values in the processed dataset.
```

In [18]:

```
#The missing value is because , we may hace a situation where there is no last valid non missing data
#We can use the mode for filling the missing data

for col in df_full:
    mode_value=df_full[col].mode()[0]
    df_full[col]=df_full[col].fillna(mode_value)

print("There are still {} missing values in the processed dataset.".format(df_full.isna().sum().sum()))
```

```
There are still 0 missing values in the processed dataset.
```

In [19]:

```
#Check for Duplicate data
print("There are {} duplicate in the processed dataset".format(df_full.duplicated().sum()))

```

```
There are 3 duplicate in the processed dataset
```

In [20]:

```
#Drop Duplicated data

df_full.drop_duplicates(inplace=True)
print("There are still {} duplicate in the processed dataset".format(df_full.duplicated().sum()))

```

```
There are still 0 duplicate in the processed dataset
```

In [21]:
```
df_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2637 entries, 0 to 2639
Data columns (total 24 columns):
 #   Column                                      Non-Null Count  Dtype  
---  ------                                      --------------  -----  
 0   rank                                        2637 non-null   int64  
 1   finalWorth                                  2637 non-null   int64  
 2   category                                    2637 non-null   object 
 3   age                                         2637 non-null   float64
 4   country                                     2637 non-null   object 
 5   selfMade                                    2637 non-null   bool   
 6   status                                      2637 non-null   object 
 7   gender                                      2637 non-null   object 
 8   birthDate                                   2637 non-null   object 
 9   state                                       2637 non-null   object 
 10  birthYear                                   2637 non-null   float64
 11  birthMonth                                  2637 non-null   float64
 12  birthDay                                    2637 non-null   float64
 13  cpi_country                                 2637 non-null   float64
 14  cpi_change_country                          2637 non-null   float64
 15  gdp_country                                 2637 non-null   object 
 16  gross_tertiary_education_enrollment         2637 non-null   float64
 17  gross_primary_education_enrollment_country  2637 non-null   float64
 18  life_expectancy_country                     2637 non-null   float64
 19  tax_revenue_country_country                 2637 non-null   float64
 20  total_tax_rate_country                      2637 non-null   float64
 21  population_country                          2637 non-null   float64
 22  latitude_country                            2637 non-null   float64
 23  longitude_country                           2637 non-null   float64
dtypes: bool(1), float64(14), int64(2), object(7)
memory usage: 497.0+ KB
```

In [22]:

```
#Validate Data and update datatypes

df_full['birthYear']=df_full['birthYear'].astype(int)
df_full['birthMonth']=df_full['birthMonth'].astype(int)
df_full['birthDay']=df_full['birthDay'].astype(int)

# BirthDate converted to datetime
df_full['birthDate']=pd.to_datetime(df_full['birthDate'])

#Gdp country has object type data with "$", need to remove dollar sign and convert to float
df_full['gdp_country']=df_full['gdp_country'].str.replace('$', '', regex=False)
df_full['gdp_country']=df_full['gdp_country'].str.replace(',', '', regex=False).astype(float)
```

In [23]:

```
df_full.head().T
```

Out[23]:

	0	1	2	3	
rank	1	2	3	4	
finalWorth	211000	180000	114000	107000	1
category	Fashion & Retail	Automotive	Technology	Technology	Finance & Inves
age	74.00	51.00	59.00	78.00	
country	France	United States	United States	United States	United
selfMade	False	True	True	True	
status	U	D	D	U	
gender	M	M	M	M	
birthDate	1949-03-05 00:00:00	1971-06-28 00:00:00	1964-01-12 00:00:00	1944-08-17 00:00:00	1930-08-30 00
state	California	Texas	Washington	Hawaii	Ne
birthYear	1949	1971	1964	1944	
birthMonth	3	6	1	8	
birthDay	5	28	12	17	
cpi_country	110.05	117.24	117.24	117.24	
cpi_change_country	1.10	7.50	7.50	7.50	
gdp_country	2,715,518,274,227.00	21,427,700,000,000.00	21,427,700,000,000.00	21,427,700,000,000.00	21,427,700,000,
gross_tertiary_education_enrollment	65.60	88.20	88.20	88.20	
gross_primary_education_enrollment_country	102.50	101.80	101.80	101.80	
life_expectancy_country	82.50	78.50	78.50	78.50	
tax_revenue_country_country	24.20	9.60	9.60	9.60	
total_tax_rate_country	60.70	36.60	36.60	36.60	
population_country	67,059,887.00	328,239,523.00	328,239,523.00	328,239,523.00	328,239,
latitude_country	46.23	37.09	37.09	37.09	
longitude_country	2.21	-95.71	-95.71	-95.71	

In [24]:

```
#Dataset Shape
df_full.shape[1]
```

Out[24]: 24

In [25]:

```python
#Using Z-score to remove the outliers


from scipy import stats
count_outlier=[]
num_cols = df_full.select_dtypes(include=['float64','int64'])
print("There are {:,} records in the data before deleting the outliers.".format(df_full.shape[0]))

for column in num_cols.columns:
    # Calculate the z-score for each column
    z = np.abs(stats.zscore(df_full[column]))

    # Identify outlier data with a z-score greater than 3
    threshold = 3
    outliers = df_full[z > threshold]
    count_outlier.append(len(outliers))
    # drop rows containing outliers
    df_full.drop(outliers.index,inplace=True)

print("There are {:,} records in the data after delete the outliers.".format(df_full.shape[0]))
df_outlier=pd.DataFrame({"column":num_cols.columns,"number_of_outliers":count_outlier})
df_outlier
```

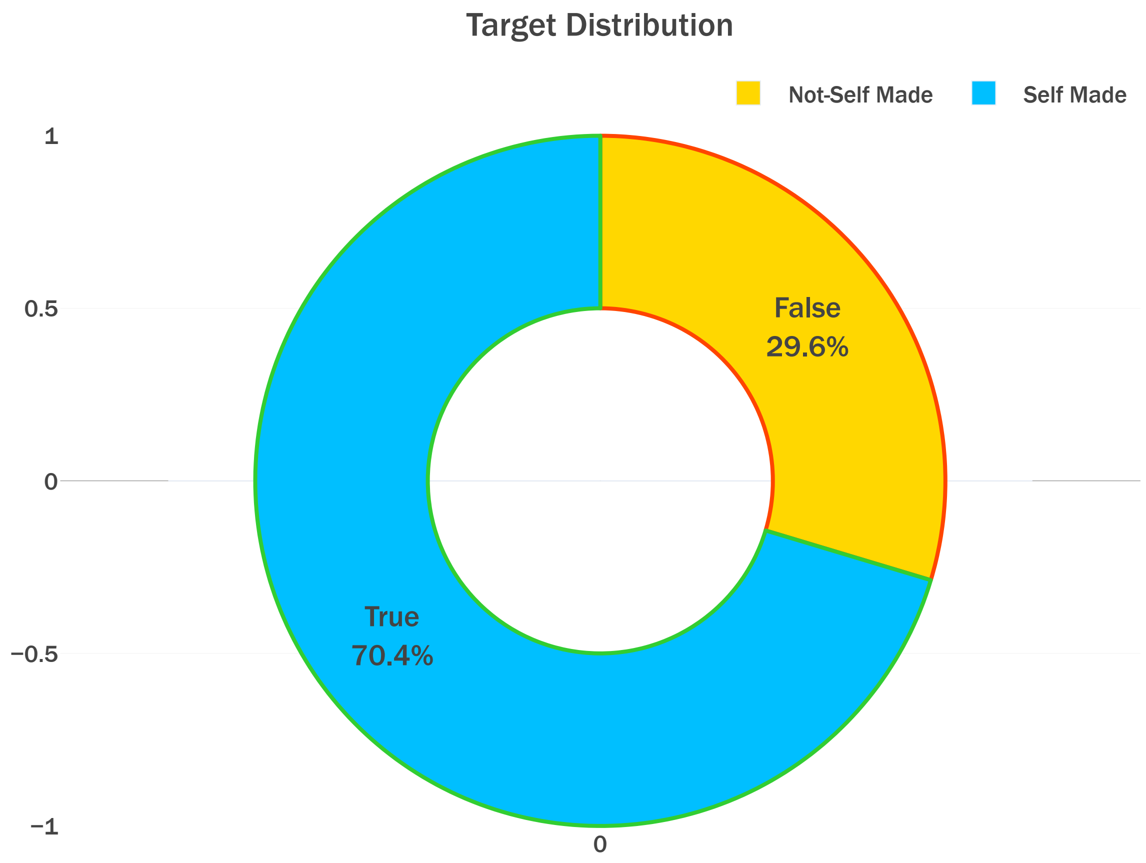
```
There are 2,637 records in the data before deleting the outliers.
There are 2,395 records in the data after delete the outliers.
```

Out[25]:

	column	number_of_outliers
0	rank	0
1	finalWorth	37
2	age	6
3	cpi_country	46
4	cpi_change_country	0
5	gdp_country	0
6	gross_tertiary_education_enrollment	4
7	gross_primary_education_enrollment_country	34
8	life_expectancy_country	8
9	tax_revenue_country_country	10
10	total_tax_rate_country	0
11	population_country	0
12	latitude_country	97
13	longitude_country	0

In [26]:

```
%matplotlib inline
fig=go.Figure()


target=df_full.selfMade.value_counts(normalize=True)[::-1]
text=['{}'.format(i) for i in target.index]
color, pal = ['#32CD32', '#FF4500'], ['#00BFFF', '#FFD700']
if text[0]=='State 0':
    color,pal=color,pal
else:
    color,pal=color[::-1],pal[::-1]


fig.add_trace(go.Pie(labels=target.index, values=target*100, hole=.5,
                     text=text, sort=False, showlegend=False,
                     marker=dict(colors=pal,line=dict(color=color,width=2)),
                     hovertemplate = "Self-Made %{label}: %{value:.2f}%<extra></extra>"))

temp = dict(layout=go.Layout(font=dict(family="Franklin Gothic", size=12)))
fig.update_layout(template=temp, title='Target Distribution',
                  uniformtext_minsize=15, uniformtext_mode='hide',width=700)


#for Legend
for i, c in enumerate(pal):
    fig.add_trace(go.Bar(x=[0], y=[0], marker_color=c, showlegend=True, name=['Not-Self Made', 'Self Made'][i]))

# Update layout
temp = dict(layout=go.Layout(font=dict(family="Franklin Gothic", size=12)))
fig.update_layout(template=temp, title='Target Distribution',
                  uniformtext_minsize=15, uniformtext_mode='hide', width=700,
                  legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1))


fig.show()
```



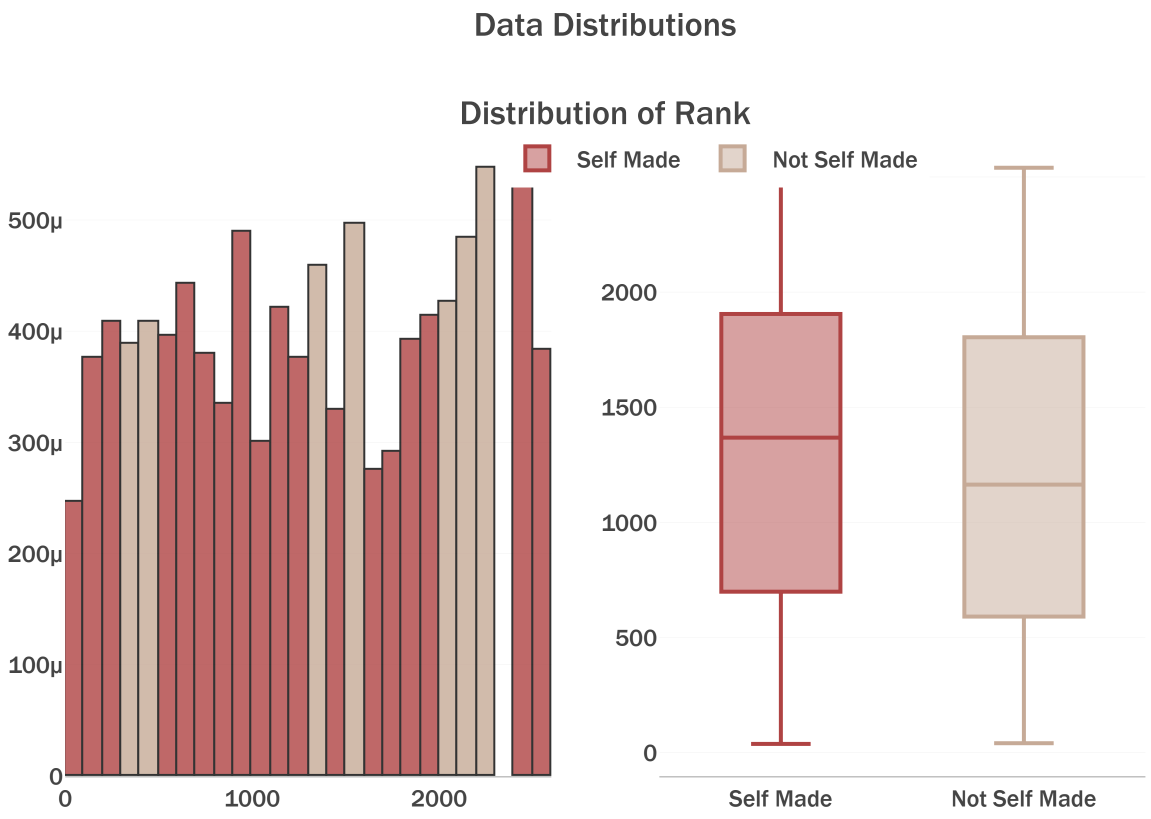
Target Distribution
Not-Self Made
Self Made
1
0.5
0
−0.5
−1
0
False
29.6%
True
70.4%

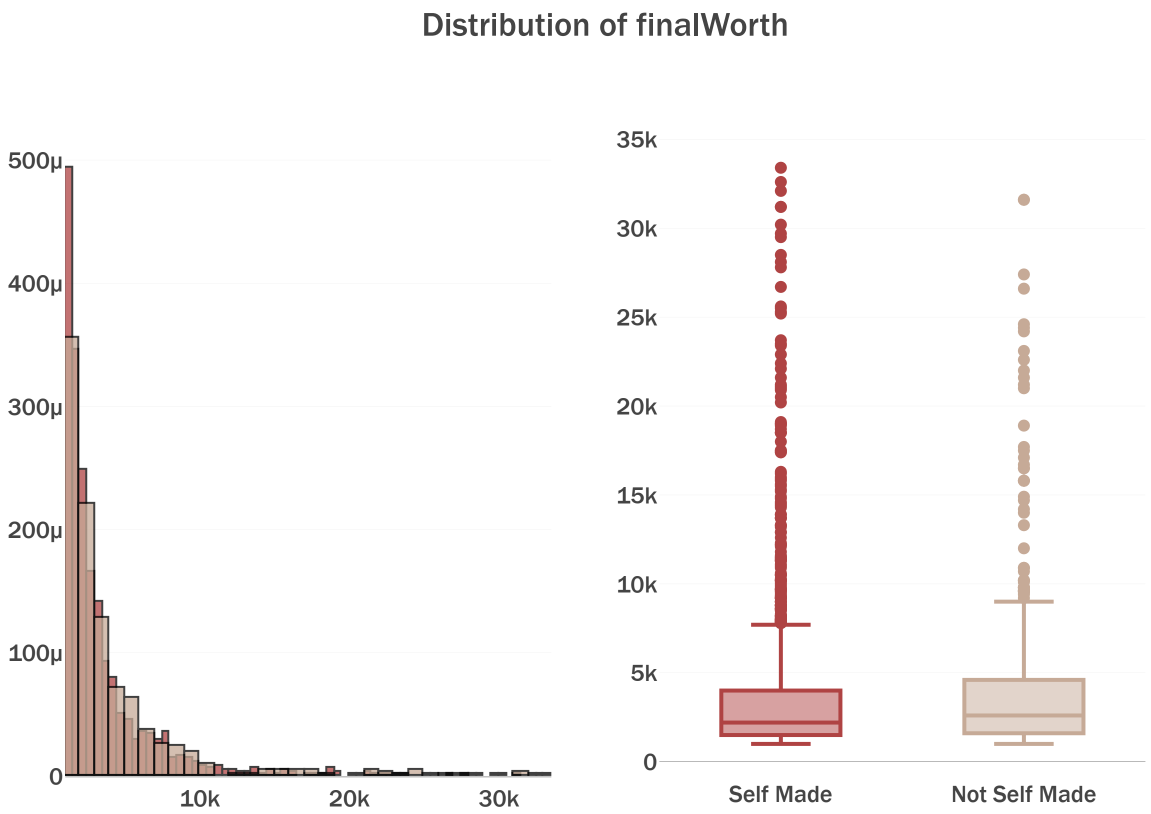
In [27]:

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd

# Assuming df_full is your DataFrame
# df_full = pd.read_csv('your_data.csv')  # Replace with your actual data loading method

# Define a custom layout template for consistent styling across all plots
temp = dict(layout=go.Layout(font=dict(family="Franklin Gothic", size=12)))

# Plotting the 'rank' column with a subplot layout (1 row, 2 columns)
fig = make_subplots(rows=1, cols=2, vertical_spacing=0.0625)

################################ Histogram ################################
# Histogram for the 'rank' column
fig.add_trace(
    go.Histogram(
        x=df_full['rank'],
        # Probability density histogram shows distribution in terms of probabilities
        histnorm='probability density',
        marker=dict(
            # Map 'selfMade' values to specific colors
            color=df_full['selfMade'].map({True: '#AF4343', False: '#C6AA97'}),
            line=dict(width=1, color='#000000'),  # Black outline for bars
        ),
        opacity=0.8,
        name='Rank',  # Name for the legend
        showlegend=False
    ), row=1, col=1
)

################################ Box ################################
# Box plot for 'Self Made' group in the 'rank' column
fig.add_trace(go.Box(y=df_full[df_full.selfMade==True]['rank'], name="Self Made",
                     marker_color='#AF4343', showlegend=True), row=1, col=2)

# Box plot for 'Not Self Made' group in the 'rank' column
fig.add_trace(go.Box(y=df_full[df_full.selfMade==False]['rank'], name="Not Self Made",
                     marker_color='#C6AA97', showlegend=True), row=1, col=2)

# Layout settings for the subplot
```

```
fig.update_layout(
    title="Data Distributions<br><br>Distribution of Rank<br>",
    barmode='overlay',
    showlegend=True,
    template=temp,
    legend=dict(orientation="h", yanchor="top", y=1.01, xanchor="left", x=.4),
    height=500,
    width=700
)
# Axis settings for the subplot
fig.update_xaxes(showline=True, linewidth=1, linecolor='black', zeroline=False)
fig.update_yaxes(showline=False, zeroline=False)

# Display the figure
fig.show()

# Plotting other numerical columns
num_cols = df_full.select_dtypes(include=['float64','int64'])

# Loop through each numerical column (excluding the first one)
for i, column in enumerate(num_cols.columns[1:], 1):
    # Creating a subplot for each numerical column
    fig = make_subplots(rows=1, cols=2, vertical_spacing=0.0625)
    fig.update_layout(barmode='overlay', template=temp, height=500, width=700)

    # Histograms for 'Self Made' and 'Not Self Made' groups
    fig.add_trace(go.Histogram(x=df_full[df_full.selfMade == True][column],
                               histnorm='probability density',
                               marker=dict(color='#AF4343', line=dict(width=1, color='#000000')),
                               opacity=0.75,
                               showlegend=False), row=1, col=1)
    fig.add_trace(go.Histogram(x=df_full[df_full.selfMade == False][column],
                               histnorm='probability density',
                               marker=dict(color='#C6AA97', line=dict(width=1, color='#000000')),
                               opacity=0.75,
                               showlegend=False), row=1, col=1)

    # Box plots for 'Self Made' and 'Not Self Made' groups
    fig.add_trace(go.Box(y=df_full[df_full.selfMade == True][column],
                         name="Self Made",
                         marker_color='#AF4343',
                         showlegend=False), row=1, col=2)
```

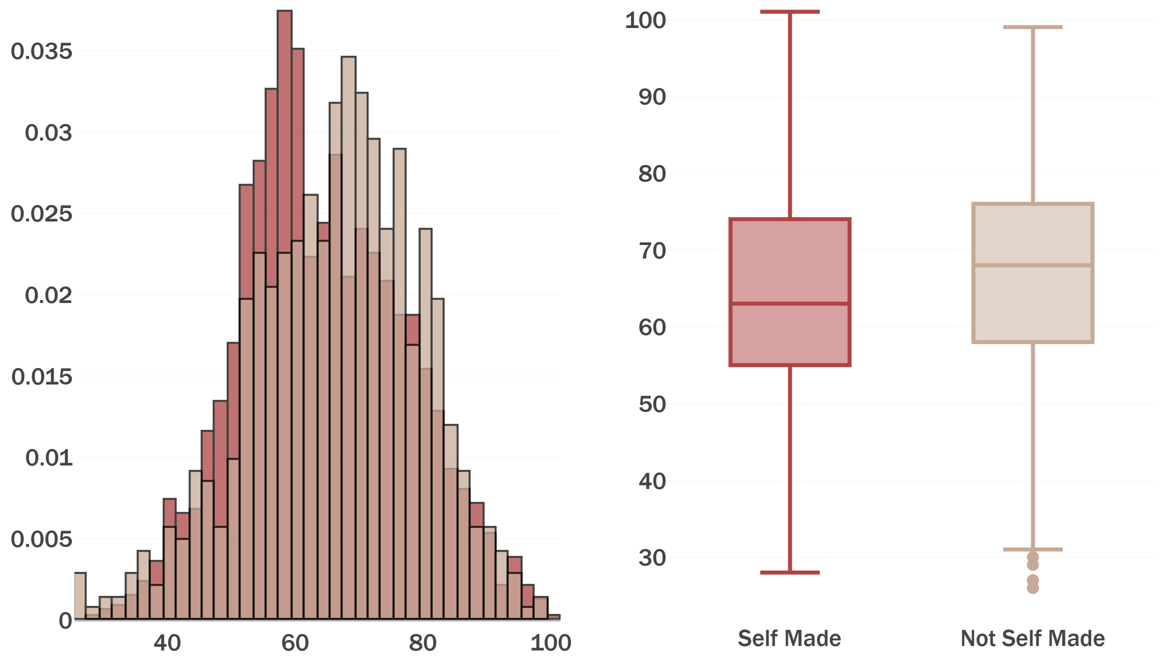
```
    fig.add_trace(go.Box(y=df_full[df_full.selfMade == False][column],
                         name="Not Self Made",
                         marker_color='#C6AA97',
                         showlegend=False), row=1, col=2)

    # Update layout for each subplot
    fig.update_layout(
        title=f'Distribution of {column}',
        xaxis=dict(showline=True, linewidth=1, linecolor='black', zeroline=False),
        yaxis=dict(showline=False, zeroline=False)
    )

    # Display the figure
    fig.show()

```


Data Distributions
Distribution of Rank
Self Made
Not Self Made
500µ
400µ
300µ
200µ
100µ
0
0
1000
2000
2000
1500
1000
500
0
Self Made
Not Self Made

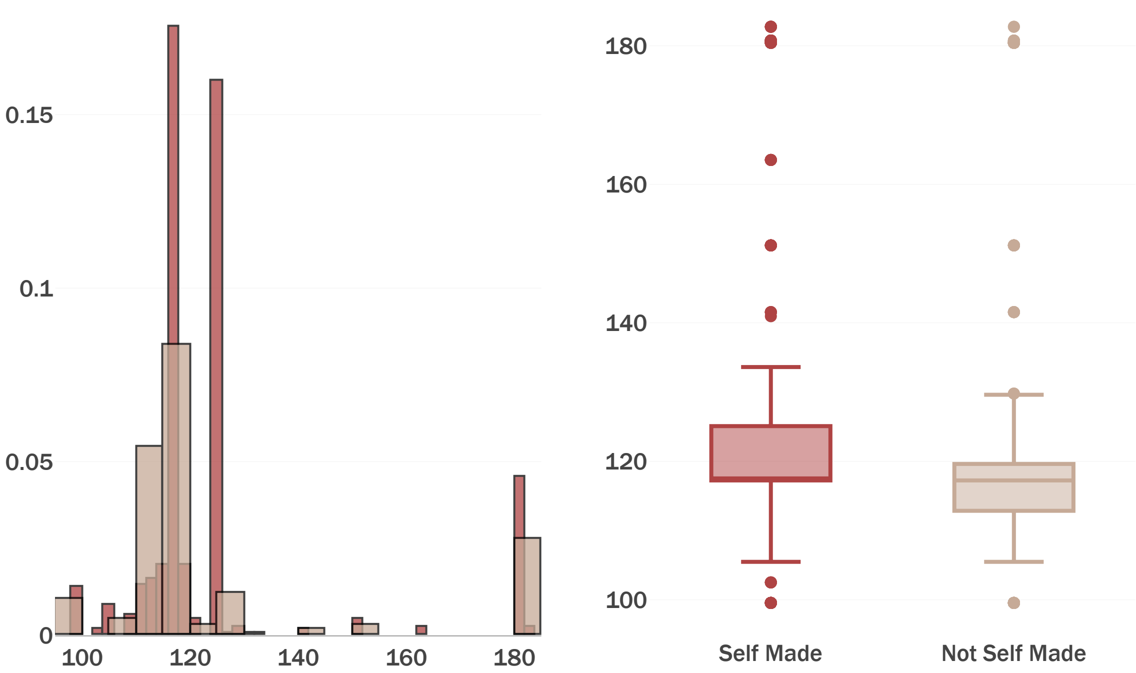


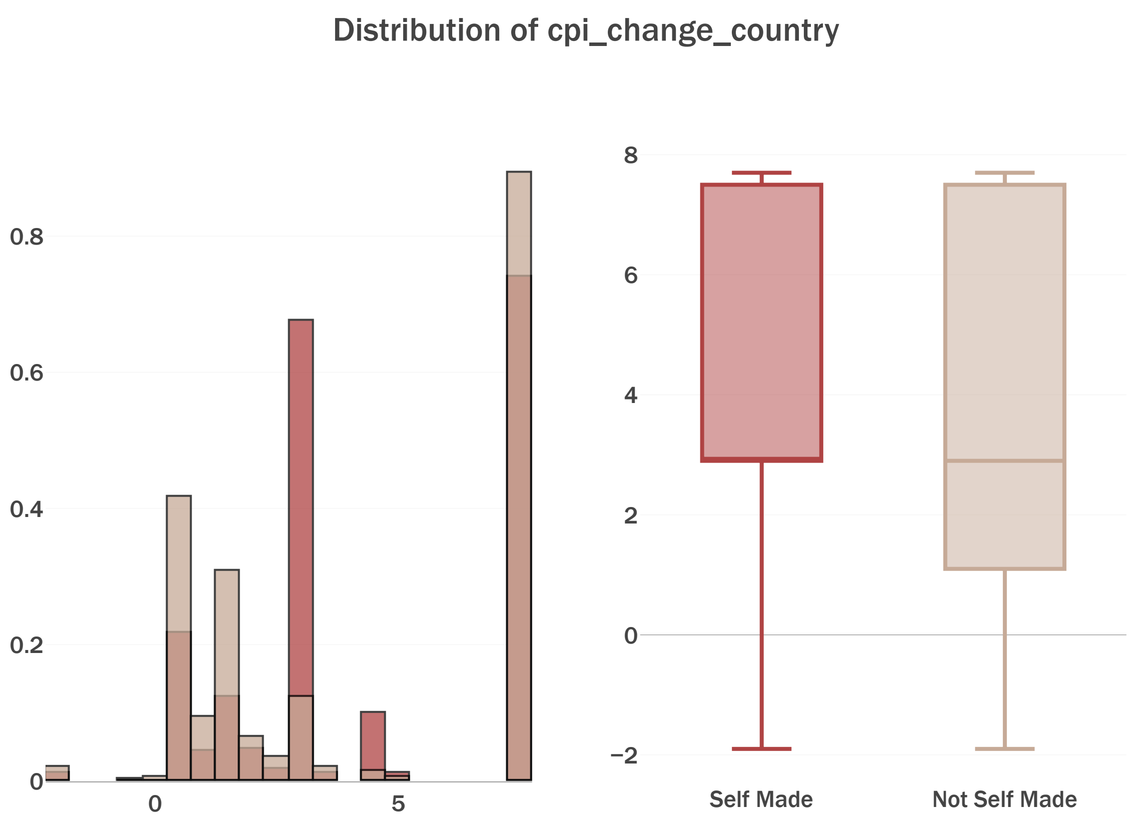
Distribution of finalWorth
500μ
400μ
300μ
200μ
100μ
0
10k
20k
30k
35k
30k
25k
20k
15k
10k
5k
0
Self Made
Not Self Made

Distribution of age

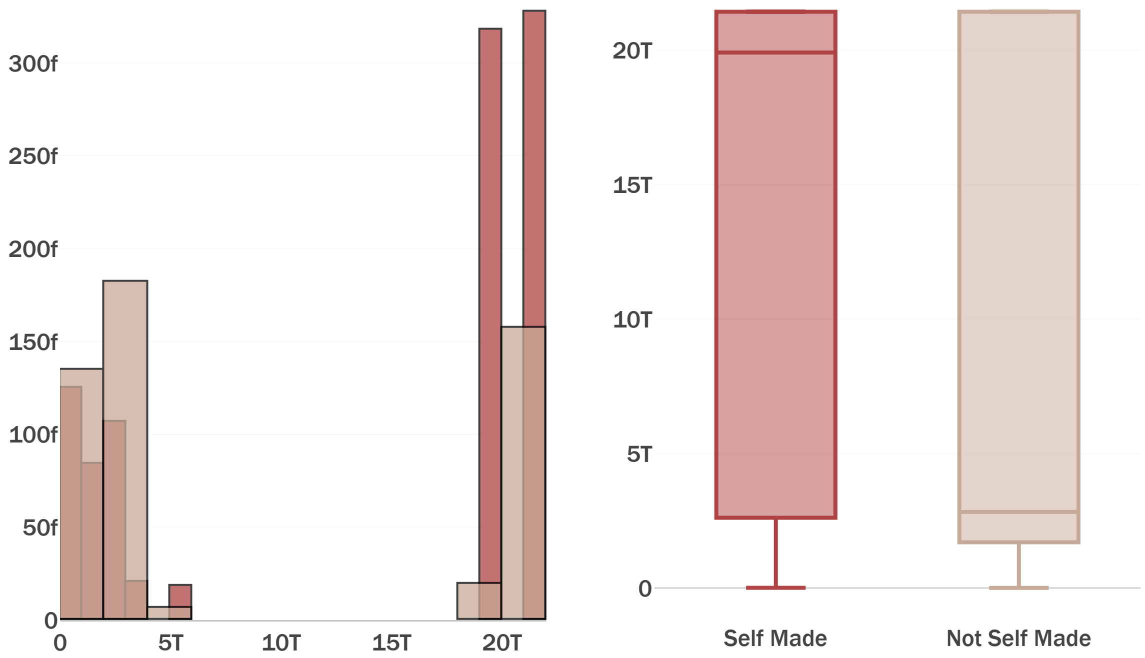


Distribution of cpi_country

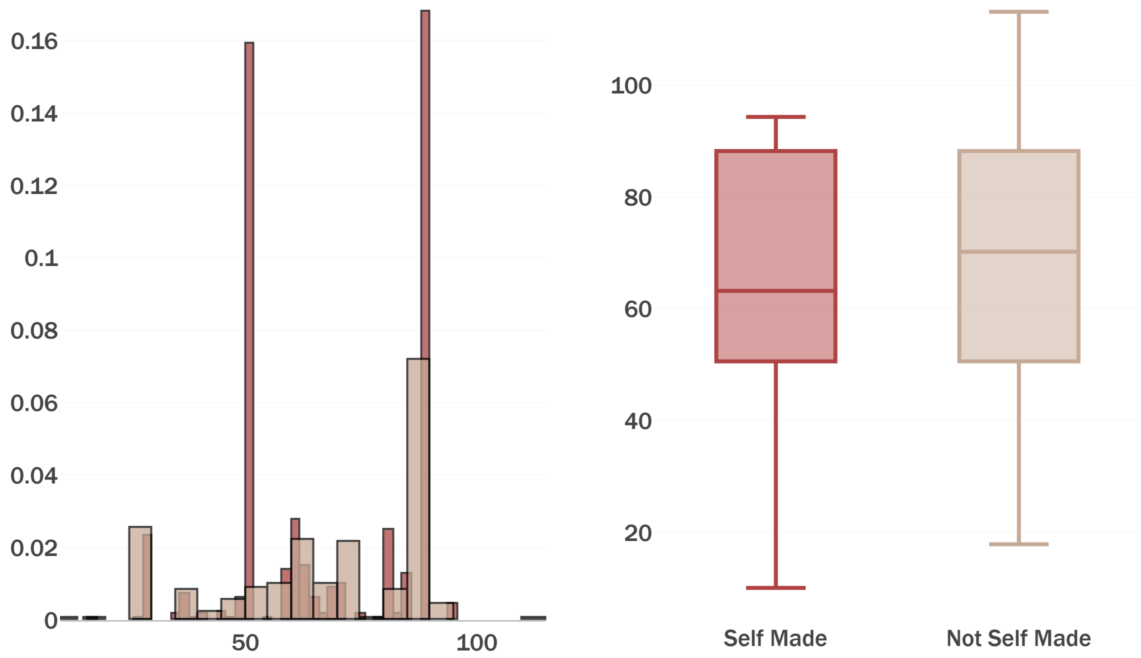



Distribution of cpi_change_country
0.8
0.6
0.4
0.2
0
0
5
8
6
4
2
0
−2
Self Made
Not Self Made

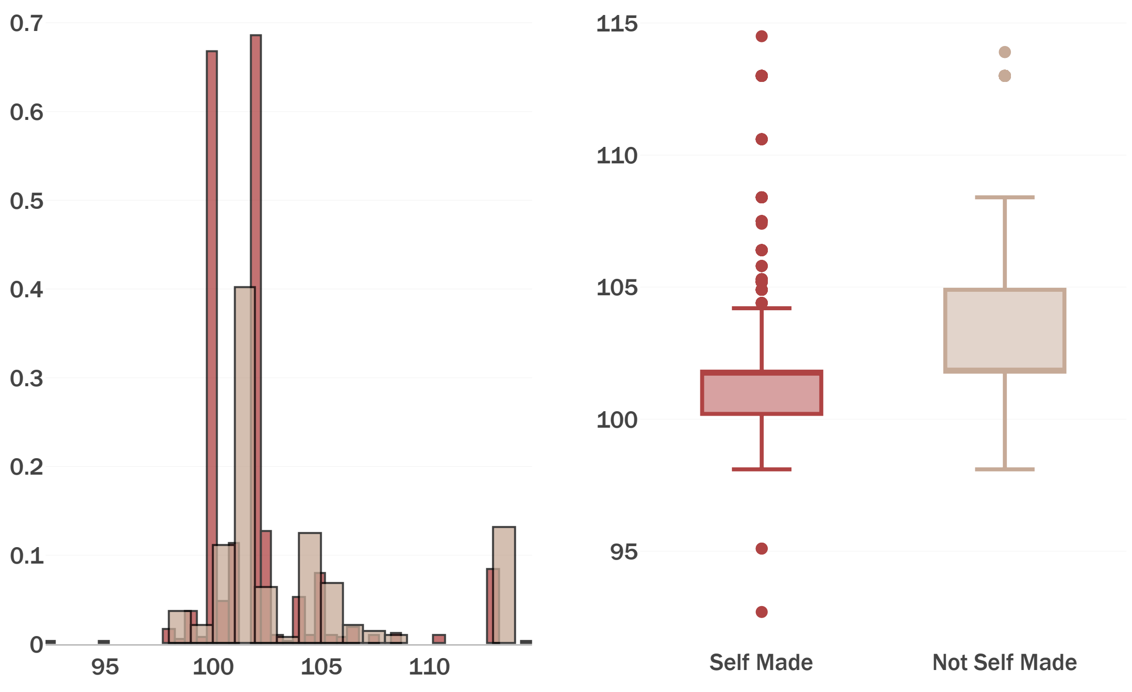
Distribution of gdp_country



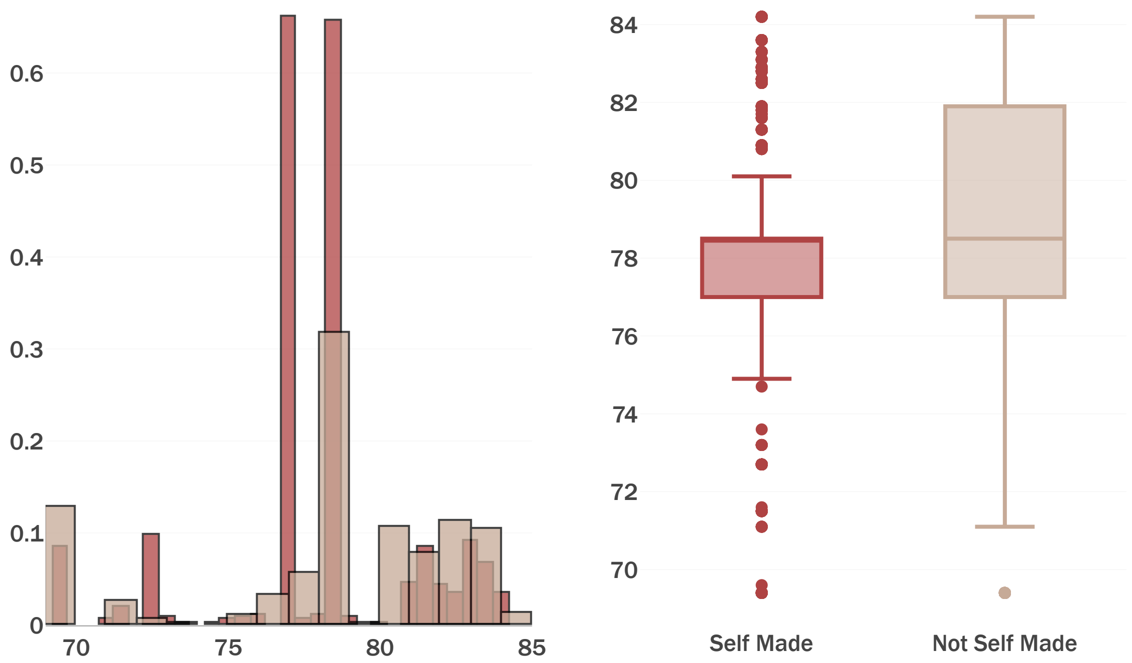
Distribution of gross_tertiary_education_enrollment



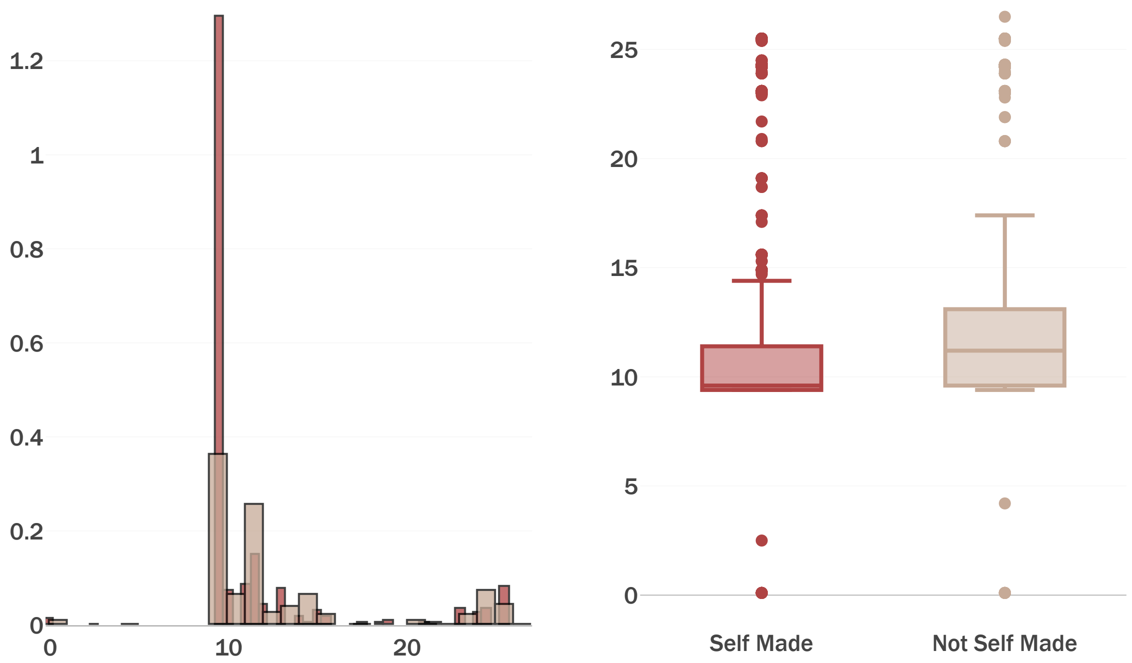
Distribution of gross_primary_education_enrollment_country



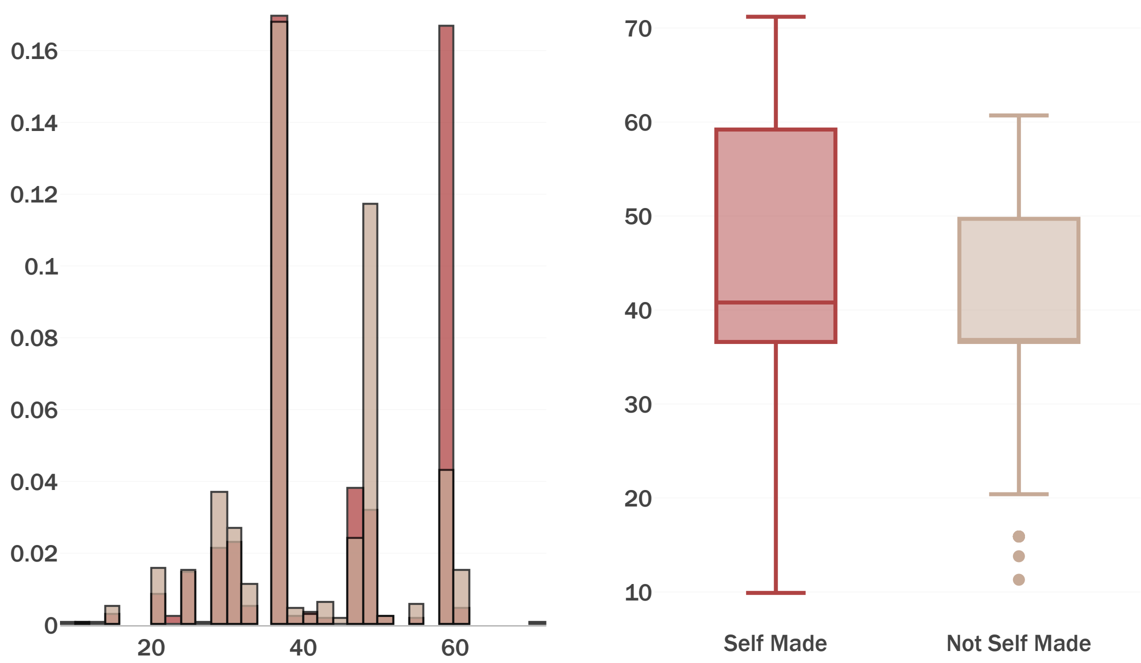
Distribution of life_expectancy_country

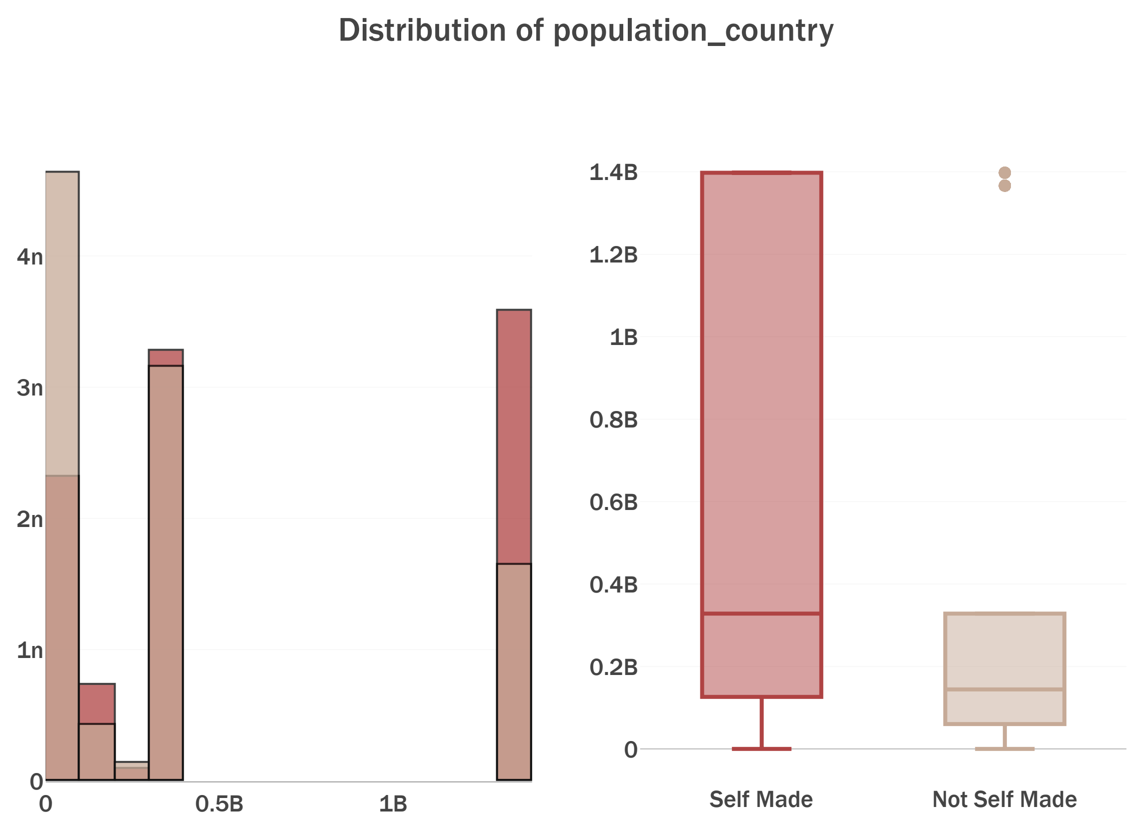


Distribution of tax_revenue_country_country



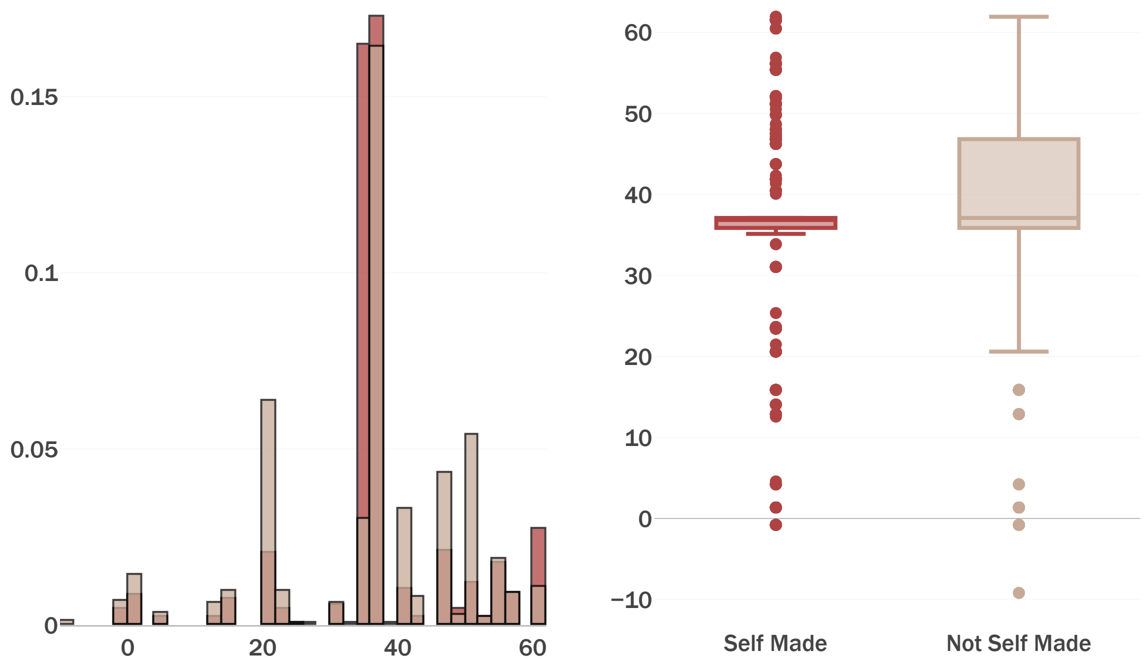
Distribution of total_tax_rate_country



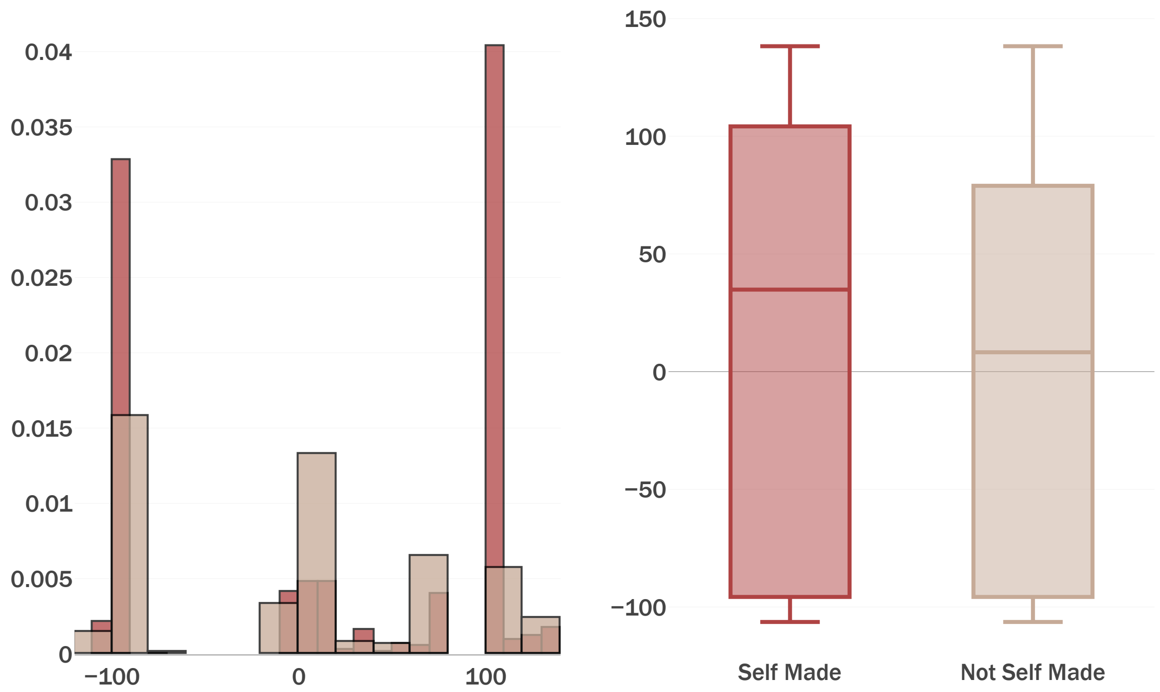


Distribution of population_country
4n
3n
2n
1n
0
0
0.5B
1B
1.4B
1.2B
1B
0.8B
0.6B
0.4B
0.2B
0
Self Made
Not Self Made

Distribution of latitude_country



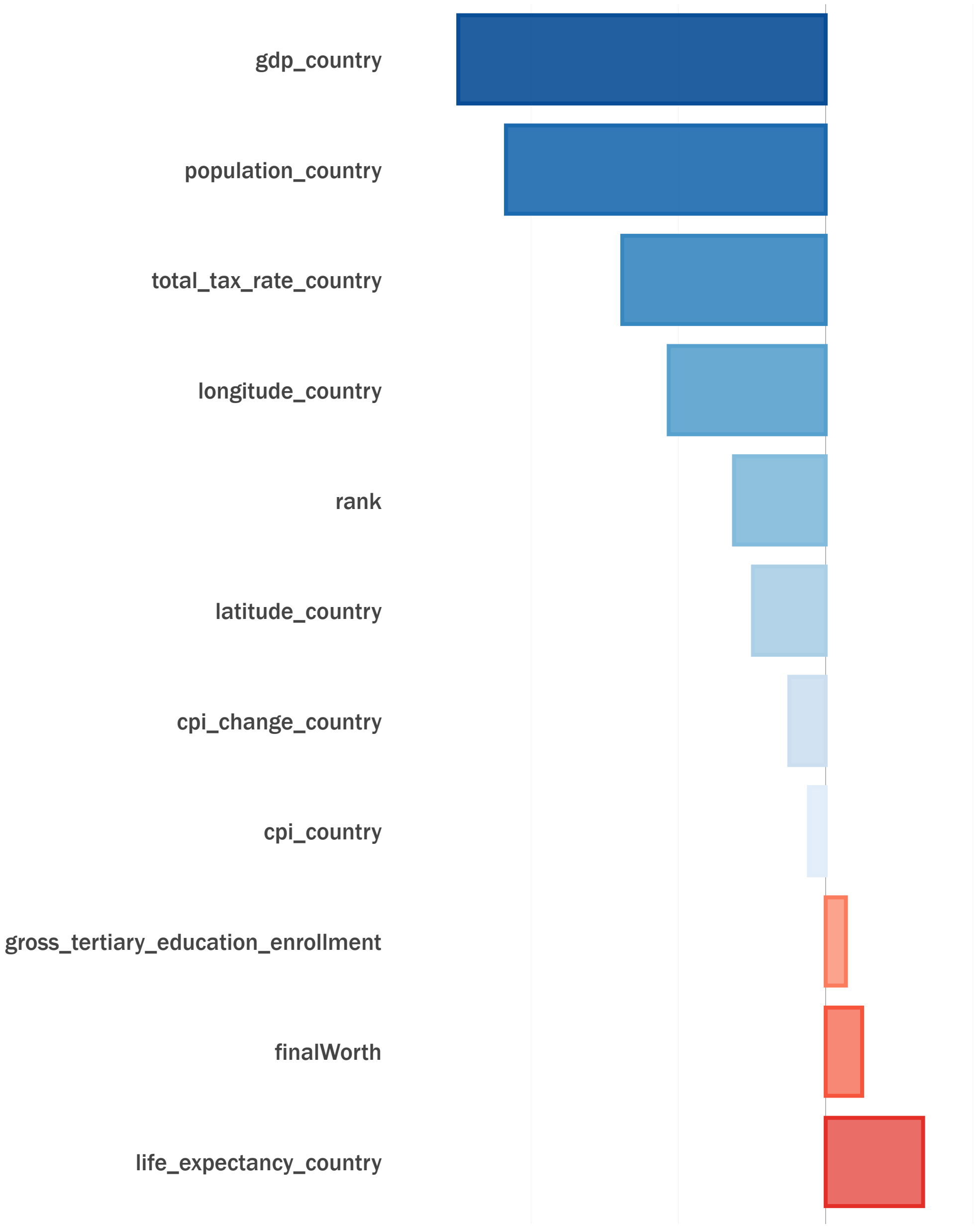
Distribution of longitude_country

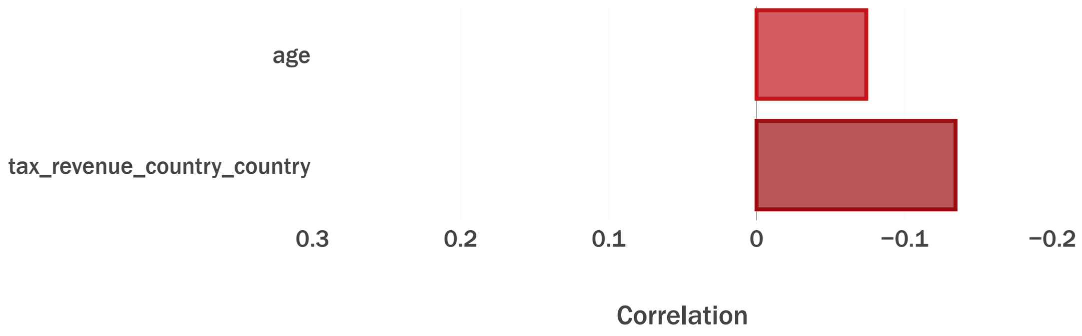


In [28]:

```
corr=df_full.select_dtypes(include=['float64','int64','bool']).corr()
corr=corr['selfMade'].sort_values(ascending=True)[1:-1]
fig = go.Figure()

pal=sns.color_palette("Reds_r",8).as_hex()
rgb=['rgba'+str(matplotlib.colors.to_rgba(i,0.7)) for i in pal]
fig.add_trace(go.Bar(x=corr[corr<0], y=corr[corr<0].index,
                     marker_color=rgb, orientation='h',
                     marker_line=dict(color=pal,width=2), name='',
                     hovertemplate='%{y} correlation with target: %{x:.3f}',
                     showlegend=False))

pal=sns.color_palette("Blues",8).as_hex()
rgb=['rgba'+str(matplotlib.colors.to_rgba(i,0.9)) for i in pal]
fig.add_trace(go.Bar(x=corr[corr>=0], y=corr[corr>=0].index,
                     marker_color=rgb, orientation='h',
                     marker_line=dict(color=pal,width=2), name='',
                     hovertemplate='%{y} correlation with target: %{x:.3f}',
                     showlegend=False))


temp = dict(layout=go.Layout(font=dict(family="Franklin Gothic", size=12)))
fig.update_layout(template=temp,title="Feature Correlations with Target (Self Made)",
                  xaxis_title="Correlation", margin=dict(l=250),
                  height=900, width=700, hovermode='closest')
fig.update_xaxes(range=(0.3,-0.2))
fig.show()
```

Feature Correlations with Target (Self Made)





In [29]:

```
#Remove irrevelant columns (Feature Engineering)

df_full.drop(columns=['category','country','state','birthDate','birthDay','birthMonth','birthYear'],inplace=True)

```

In [30]:

```python
print("current Shape",df_full.shape)
df_full.head().T
```

```
current Shape (2395, 17)
```

Out[30]:

	37	38	39	40	
rank	38	39	40	41	
finalWorth	33400	32600	32100	31600	3
age	54.00	74.00	65.00	74.00	
selfMade	True	True	True	False	
status	D	U	D	U	
gender	M	M	M	M	
cpi_country	125.08	105.48	119.62	117.24	
cpi_change_country	2.90	0.50	1.70	7.50	
gdp_country	19,910,000,000,000.00	5,081,769,542,380.00	2,827,113,184,696.00	21,427,700,000,000.00	21,427,700,000,0
gross_tertiary_education_enrollment	50.60	63.20	60.00	88.20	
gross_primary_education_enrollment_country	100.20	98.80	101.20	101.80	
life_expectancy_country	77.00	84.20	81.30	78.50	
tax_revenue_country_country	9.40	11.90	25.50	9.60	
total_tax_rate_country	59.20	46.70	30.60	36.60	
population_country	1,397,715,000.00	126,226,568.00	66,834,405.00	328,239,523.00	328,239,5
latitude_country	35.86	36.20	55.38	37.09	
longitude_country	104.20	138.25	-3.44	-95.71	

In [31]:
```
df_full.describe().T
```

Out[31]:

	count	mean	std	min	25%	
rank	2,395.00	1,298.97	729.71	38.00	659.00	
finalWorth	2,395.00	3,730.94	4,132.17	1,000.00	1,500.00	
age	2,395.00	65.02	12.98	26.00	56.00	
cpi_country	2,395.00	125.19	20.72	99.55	117.24	
cpi_change_country	2,395.00	4.22	2.84	-1.90	1.70	
gdp_country	2,395.00	12,456,897,309,555.13	9,441,190,639,529.93	3,154,057,987.00	2,029,000,000,000.00	19,910,000,(
gross_tertiary_education_enrollment	2,395.00	67.04	20.00	10.10	50.60	
gross_primary_education_enrollment_country	2,395.00	102.51	3.37	92.70	100.20	
life_expectancy_country	2,395.00	78.11	3.56	69.40	77.00	
tax_revenue_country_country	2,395.00	11.90	4.73	0.10	9.60	
total_tax_rate_country	2,395.00	43.58	11.86	9.90	36.60	
population_country	2,395.00	549,353,321.82	564,291,820.23	38,019.00	69,625,582.00	328,2
latitude_country	2,395.00	37.25	12.02	-9.19	35.86	
longitude_country	2,395.00	12.75	87.82	-106.35	-95.71	

**Feature Selection **

```
#PCA Exploration Begins
```

In [35]:

```
#For Testing with PCA
df_full_t=df_full


def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
import numpy as np
import pandas as pd
from itertools import accumulate
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
warnings.filterwarnings('ignore')
sns.set_context('notebook')
sns.set_style('white')



df_full_t['selfMade'] = df_full_t['selfMade'].astype(int)

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
one_hot = ColumnTransformer(transformers=[("one_hot", OneHotEncoder(), ['status','gender']) ],remainder="passthrou
data=one_hot.fit_transform(df_full_t)

names=one_hot.get_feature_names_out()
column_names=[name[name.find("_")+1:] for name in  [name[name.find("__")+2:] for name in names]]
new_data=pd.DataFrame(data,columns=column_names)
new_data.head()
```

Out[35]:

	D	E	N	R	Split Family Fortune	U	F	M	rank	finalWorth	...	change_country	country	tertiary_education_enrollment	prim
0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	33,400.00	...	2.90	19,910,000,000,000.00	50.60	
1	0.00	0.00	0.00	0.00	0.00	1.00	0.00	1.00	1.00	32,600.00	...	0.50	5,081,769,542,380.00	63.20	
2	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	32,100.00	...	1.70	2,827,113,184,696.00	60.00	
3	0.00	0.00	0.00	0.00	0.00	1.00	0.00	1.00	1.00	31,600.00	...	7.50	21,427,700,000,000.00	88.20	
4	0.00	0.00	0.00	0.00	0.00	1.00	0.00	1.00	1.00	31,600.00	...	7.50	21,427,700,000,000.00	88.20	

5 rows × 23 columns

In [37]:

```python
features = new_data.columns
from sklearn.preprocessing import StandardScaler
x = new_data.loc[:, features].values
x = StandardScaler().fit_transform(x) # normalizing the features
```

In [38]:

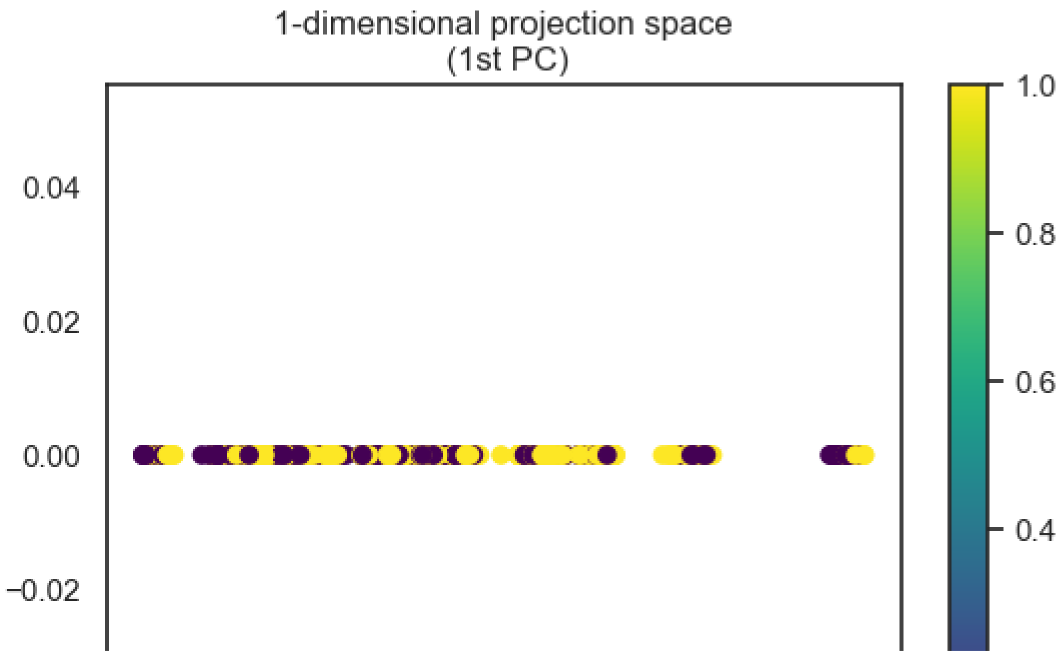
```python
feat_cols = ['feature'+str(i) for i in range(x.shape[1])]
from sklearn.decomposition import PCA
pca_result = PCA(n_components=3)
principalComponents = pca_result.fit_transform(x)
y=new_data['selfMade']
y.head()
```

Out[38]:

```
0    1.00
1    1.00
2    1.00
3    0.00
4    0.00
Name: selfMade, dtype: float64
```

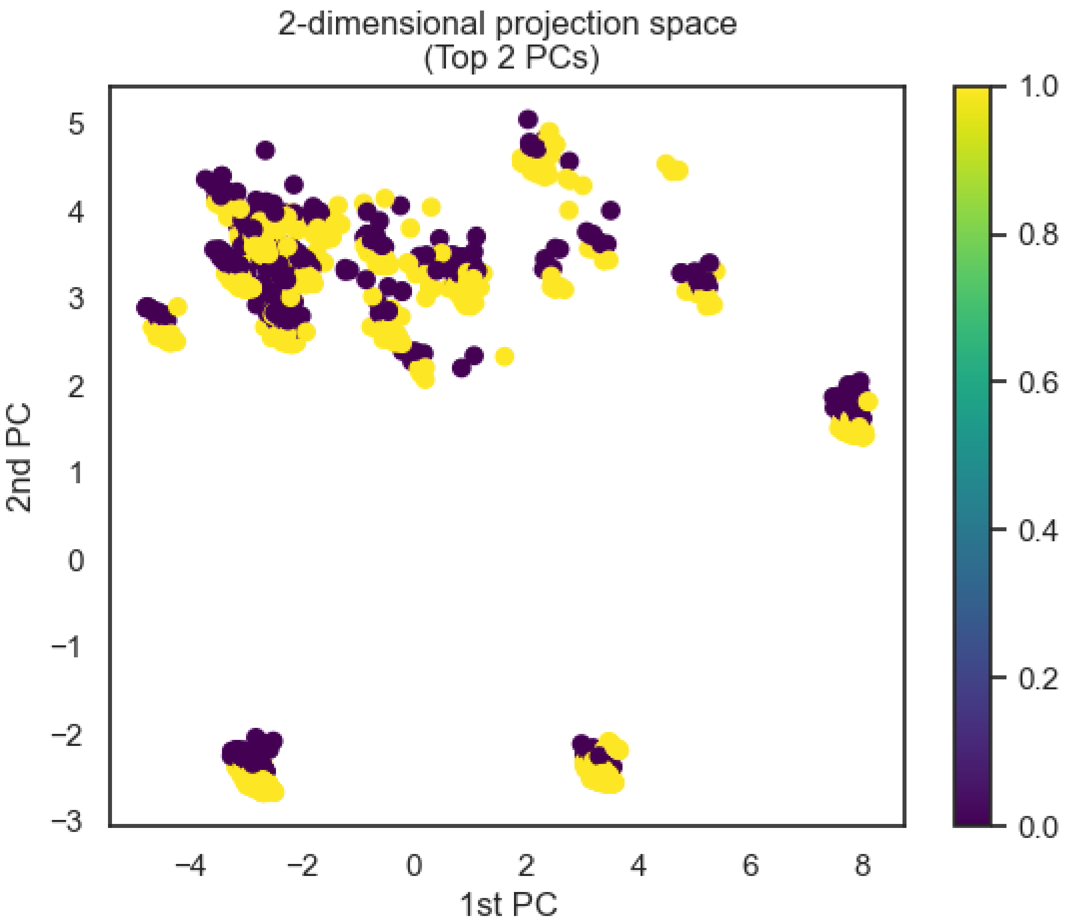
In [39]:

```
fig, ax = plt.subplots()
sc=ax.scatter(principalComponents[:,0],np.zeros(principalComponents[:,1].shape ),c=y,cmap='viridis')
ax.set_title('1-dimensional projection space\n (1st PC)')
fig.colorbar(sc, orientation='vertical')
plt.savefig("1d_projection_space.png")
plt.show()

```



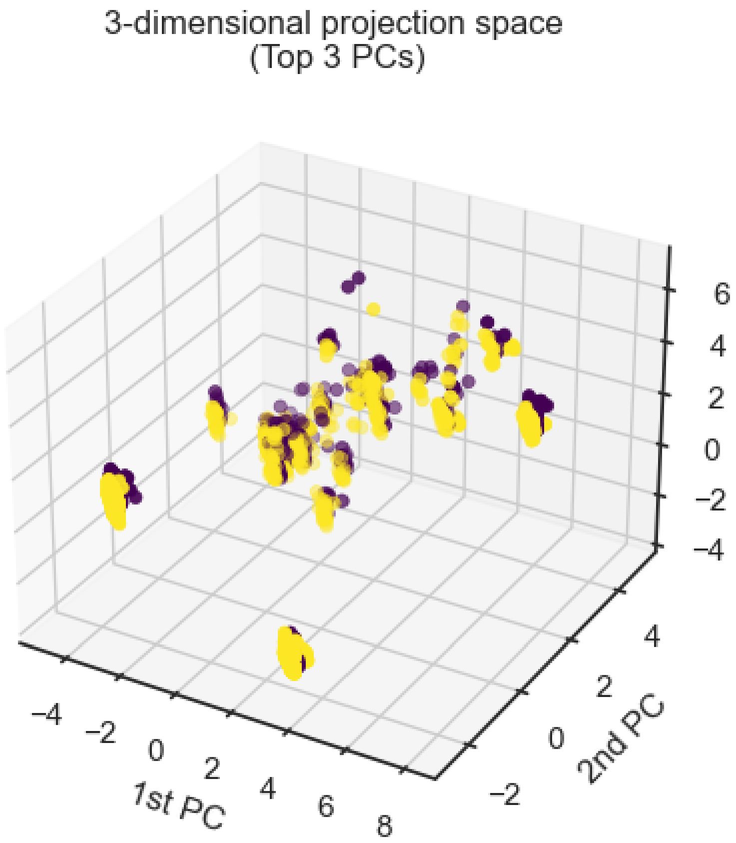
In [40]:

```
fig, ax = plt.subplots()
sc=ax.scatter(principalComponents[:,0],principalComponents[:,1] ,c=y,cmap='viridis')
fig.colorbar(sc, orientation='vertical')
ax.set_title('2-dimensional projection space\n (Top 2 PCs)')
plt.xlabel("1st PC")
plt.ylabel("2nd PC")
plt.savefig("2d_projection_space.png")
plt.show()
```



In [41]:

```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
sc=ax.scatter(principalComponents[:,0], principalComponents[:,1],  principalComponents[:,2], c=y, cmap='viridis',
ax.set_title('3-dimensional projection space\n (Top 3 PCs)')
ax.set_xlabel('1st PC')
ax.set_ylabel('2nd PC')
ax.set_zlabel('3rd PC')
plt.savefig("3d_projection_space.png")
plt.show()
```



In [42]:
```
print('Explained variation per principal component: {}'.format(pca_result.explained_variance_ratio_))
```

```
Explained variation per principal component: [0.27482737 0.18858704 0.11537287]
```

#PCA exploration ends

In [43]:
```
#Since the min rank = 38 and the max rank=2540, we need to group this ranks into ranges , so that the model
#does not get confused ,5 labels have been produced each having a range of 500 , so rank suppose 38 will fall
#under label -->1

df_full['rank']=pd.cut(df_full['rank'],[0,500,1000,1500,2000,2600],labels=[1,2,3,4,5])
df_full.head()
```

Out[43]:

	rank	finalWorth	age	selfMade	status	gender	cpi_country	cpi_change_country	gdp_country	gross_tertiary_education_enrollment
37	1	33400	54.00	1	D	M	125.08	2.90	19,910,000,000,000.00	50.60
38	1	32600	74.00	1	U	M	105.48	0.50	5,081,769,542,380.00	63.20
39	1	32100	65.00	1	D	M	119.62	1.70	2,827,113,184,696.00	60.00
40	1	31600	74.00	0	U	M	117.24	7.50	21,427,700,000,000.00	88.20
41	1	31600	72.00	0	U	M	117.24	7.50	21,427,700,000,000.00	88.20

In [44]:

```
#Data Transformation
#Separation of target and feature variable
y=df_full.selfMade
X=df_full.drop('selfMade', axis=1)
#Drop_first avoids multicollinearity
X=pd.get_dummies(X, columns=['status','gender'], drop_first=True)
# create new features
#Need only interaction terms not polynomial terms , bias false does not create additional column of ones
pf = PolynomialFeatures(interaction_only=True, include_bias=False)
X_pf = pd.DataFrame(data=pf.fit_transform(X), columns=pf.get_feature_names_out(X.columns))
print("New data shape with interaction terms:",X_pf.shape)
X_pf.head()

```

```
New data shape with interaction terms: (2395, 210)
```

Out[44]:

	rank	finalWorth	age	cpi_country	cpi_change_country	gdp_country	gross_tertiary_education_enrollment	gross_primary_education_er
0	1.00	33,400.00	54.00	125.08	2.90	19,910,000,000,000.00	50.60	
1	1.00	32,600.00	74.00	105.48	0.50	5,081,769,542,380.00	63.20	
2	1.00	32,100.00	65.00	119.62	1.70	2,827,113,184,696.00	60.00	
3	1.00	31,600.00	74.00	117.24	7.50	21,427,700,000,000.00	88.20	
4	1.00	31,600.00	72.00	117.24	7.50	21,427,700,000,000.00	88.20	

5 rows × 210 columns

In [78]:

```python
#There will be quasi-constant features , i,e features having not much of variation
#Lets put a threshold for .25 , so that all the features having vartions< .25 will be removed
remove_quasi_costant = VarianceThreshold(threshold = 0.25) #Removing both constant and quasi-constant
remove_quasi_costant.fit(X_pf)
#Items having True are features will be retained
remove_quasi_costant.get_support()
```

Out[78]:

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False, False, False, False,
       False, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False])
```

In [79]:
```
#Lets remove the low variance features

# Picking Up the low Variance Columns
col_low_variance = [column for column in X_pf.columns
          if column not in X_pf.columns[remove_quasi_costant.get_support()]]

for features in col_low_variance:
    print(features)
```

```
status_E
status_N
status_R
status_Split Family Fortune
status_U
gender_M
status_E status_N
status_E status_R
status_E status_Split Family Fortune
status_E status_U
status_E gender_M
status_N status_R
status_N status_Split Family Fortune
status_N status_U
status_N gender_M
status_R status_Split Family Fortune
status_R status_U
status_R gender_M
status_Split Family Fortune status_U
status_Split Family Fortune gender_M
status_U gender_M
```

In [80]:
```
# Drop the low variance columns
X_pf.drop(col_low_variance,axis=1,inplace=True)
print("New data shape with interaction terms and reduced features:",X_pf.shape)

```

```
New data shape with interaction terms and reduced features: (2395, 189)
```

In [81]:
```
X_pf.describe()
```

Out[81]:

	rank	finalWorth	age	cpi_country	cpi_change_country	gdp_country	gross_tertiary_education_enrollment	gross_primary_ec
count	2,395.00	2,395.00	2,395.00	2,395.00	2,395.00	2,395.00	2,395.00	
mean	3.09	3,730.94	65.02	125.19	4.22	12,456,897,309,555.13	67.04	
std	1.44	4,132.17	12.98	20.72	2.84	9,441,190,639,529.93	20.00	
min	1.00	1,000.00	26.00	99.55	-1.90	3,154,057,987.00	10.10	
25%	2.00	1,500.00	56.00	117.24	1.70	2,029,000,000,000.00	50.60	
50%	3.00	2,300.00	65.00	117.24	2.90	19,910,000,000,000.00	65.60	
75%	4.00	4,200.00	74.00	125.08	7.50	21,427,700,000,000.00	88.20	
max	5.00	33,400.00	101.00	182.75	7.70	21,427,700,000,000.00	113.10	

8 rows × 189 columns

In [82]:

```
#Data Transformation
# Scale features
scaler=StandardScaler()
X_pf=pd.DataFrame(scaler.fit_transform(X_pf), columns=X_pf.columns)
X_pf.describe()
```

Out[82]:

	rank	finalWorth	age	cpi_country	cpi_change_country	gdp_country	gross_tertiary_education_enrollment	gross_primary_education_
count	2,395.00	2,395.00	2,395.00	2,395.00	2,395.00	2,395.00	2,395.00	
mean	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	
std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	
min	-1.46	-0.66	-3.01	-1.24	-2.16	-1.32	-2.85	
25%	-0.76	-0.54	-0.70	-0.38	-0.89	-1.10	-0.82	
50%	-0.06	-0.35	-0.00	-0.38	-0.47	0.79	-0.07	
75%	0.63	0.11	0.69	-0.01	1.15	0.95	1.06	
max	1.33	7.18	2.77	2.78	1.22	0.95	2.30	

8 rows × 189 columns

In [83]:

```python
#Lets run few models and check which is the best one
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
#Support vector classification
from sklearn.svm import SVC
from sklearn.metrics import make_scorer, f1_score



# Initialize the model#Increased max_iter from 100 to 1000 for convergence issue
model_log_reg = LogisticRegression(max_iter=1000)
model_random_for_classifier = RandomForestClassifier()
model_svc = SVC()


# Initialize KFold
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Perform k-fold cross-validation with F1 scoring
f1_scorer = make_scorer(f1_score)

#Logistic Regression
cv_f1_scores_log_reg = cross_val_score(model_log_reg, X_pf, y, cv=kf, scoring=f1_scorer)
#Random Forest Classifier
cv_f1_scores_for_classifier = cross_val_score(model_random_for_classifier, X_pf, y, cv=kf, scoring=f1_scorer)
#SVC or Support Vector Classifier
cv_f1_scores_svc = cross_val_score(model_svc, X_pf, y, cv=kf, scoring=f1_scorer)



# Display the cross-validation F1 scores
print("Logistic Regression results  \n")
print("Cross-validation F1 scores:", cv_f1_scores_log_reg)
print("Average F1 Score:", cv_f1_scores_log_reg.mean())
print("\n\nRandom Forest Classifier results \n")
print("Cross-validation F1 scores:", cv_f1_scores_for_classifier)
print("Average F1 Score:", cv_f1_scores_for_classifier.mean())
print("\n\nSVC results \n")
print("Cross-validation F1 scores:", cv_f1_scores_svc)
print("Average F1 Score:", cv_f1_scores_svc.mean())
```

```
Logistic Regression results

Cross-validation F1 scores: [0.87569061 0.83844011 0.82576867 0.864      0.84722222]
Average F1 Score: 0.8502243218040787


Random Forest Classifier results

Cross-validation F1 scores: [0.84813754 0.82708934 0.78593272 0.85714286 0.81779053]
Average F1 Score: 0.8272185965388583


SVC results

Cross-validation F1 scores: [0.86909582 0.85479452 0.83168317 0.86410256 0.85054348]
Average F1 Score: 0.8540439095384895
```

In [85]:

```
#Selection of best model
#The model which been selected is SVC
#We need to perform hyper parameter tuning

#Lets Split the dataset into  Training and test sets , 80% training and 20% test
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_pf, y, test_size=0.2, random_state=42)


```

In [86]:

```
#Hyperparameter tuning using GridCV
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'],  # Type of the kernel
    'gamma': ['scale', 'auto'],  # Kernel coefficient
    'class_weight': [None, 'balanced']  # Handling class imbalance
}


# Set the random seed for reproducibility
random_state = 25
np.random.seed(random_state)

# Create the SVC model
model = SVC(probability=True,random_state=random_state)

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='f1')
grid_search.fit(X_train, y_train)
```

Out[86]:

```
GridSearchCV(cv=5, estimator=SVC(probability=True, random_state=25),
             param_grid={'C': [0.1, 1, 10], 'class_weight': [None, 'balanced'],
                         'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'rbf', 'poly']},
             scoring='f1')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [87]:

```python
#Get the best model and parameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
#Print best model and best_params
print("Best Model Parameters:", grid_search.best_estimator_)
print("\nBest Model:", grid_search.best_params_)
# Retrain the best model
best_model.fit(X_train, y_train)

# Evaluate on the training set
y_train_pred = best_model.predict(X_train)
f1_train = f1_score(y_train_pred, y_train)

# Evaluate on the test set
y_test_pred = best_model.predict(X_test)
f1_test = f1_score(y_test_pred, y_test)

# Print the F1 scores
print('F1-score on the train set:', f1_train)
print('F1-score on the test set:', f1_test)

```

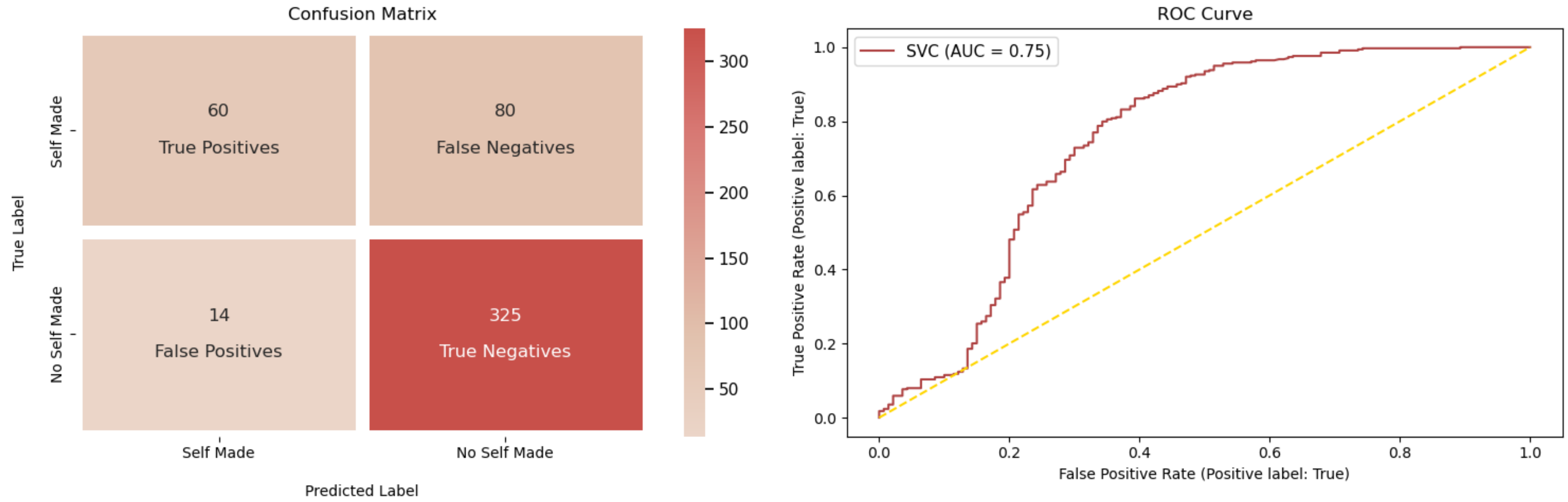
```
Best Model Parameters: SVC(C=0.1, kernel='linear', probability=True, random_state=25)

Best Model: {'C': 0.1, 'class_weight': None, 'gamma': 'scale', 'kernel': 'linear'}
F1-score on the train set: 0.8646641916976037
F1-score on the test set: 0.8736559139784946
```

In [88]:

```
from sklearn.metrics import confusion_matrix, roc_curve, classification_report
from sklearn.metrics import f1_score, precision_score, recall_score, roc_auc_score,auc,RocCurveDisplay
# Subplot 1: Confusion Matrix
from matplotlib.colors import LinearSegmentedColormap

cm = confusion_matrix(y_test,y_test_pred ,labels=[0,1])
plt.subplots_adjust(hspace=0.5)
warm=LinearSegmentedColormap.from_list('warm',
                                       [(0, '#EBD5C8'),
                                        (0.25, '#E1C1AD'),
                                        (.75, '#D77873'),
                                        (1, '#C8504A')], N=256)

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(15,5))

classes=['Self Made','No Self Made']
label = ['True Positives', 'False Negatives', 'False Positives', 'True Negatives']
annot = [f'{i}\n\n{j}' for i, j in zip(cm.flatten(), label)]
annot = np.asarray(annot).reshape(2,2)

sns.set_context('notebook')
sns.heatmap(cm, annot=annot, fmt='', cmap=warm, linecolor='white', linewidths=8, ax=ax[0])
ax[0].set_title('Confusion Matrix')
ax[0].set_xlabel('\nPredicted Label')
ax[0].set_ylabel('True Label\n')
ax[0].xaxis.set_ticklabels(classes)
ax[0].yaxis.set_ticklabels(classes)

# Subplot 2: AUC

y_proba= best_model.predict_proba(X_test)
fpr, tpr, _ = roc_curve(y_test, y_proba[:,1])
roc_auc = auc(fpr, tpr)
RocCurveDisplay.from_estimator(best_model, X_test, y_test,ax=ax[1],color="#AF4343")
sns.lineplot(x = [0, 1], y = [0, 1], color = 'gold', linestyle="dashed",ax=ax[1])
ax[1].set_title('ROC Curve')

plt.tight_layout()
```

```
39  plt.show();
```

<Figure size 640x480 with 0 Axes>



In [89]:

```python
# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50,100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy'],
    'class_weight': [None, 'balanced']
}

# Set the random seed for reproducibility
random_state = 25
np.random.seed(random_state)


# Create the RandomForestClassifier model
model = RandomForestClassifier(class_weight='balanced',random_state=random_state)

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Retrieve the best model and best parameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# Retrain the best model on the entire training set
best_model.fit(X_train, y_train)

# Evaluate on training set
y_train_val_pred = best_model.predict(X_train)
f1_train = f1_score(y_train_val_pred, y_train)

#Calculate the model results to the data points in the testing data set.
y_test_pred = best_model.predict(X_test)
f1_test = f1_score(y_test_pred, y_test)



print('F1-score on the train set:', f1_train)
```

```
print('F1-score on the test set:', f1_test)
```

```
F1-score on the train set: 0.9018789144050104
F1-score on the test set: 0.8698060941828255
```

In [91]:

```
print(best_model)
```

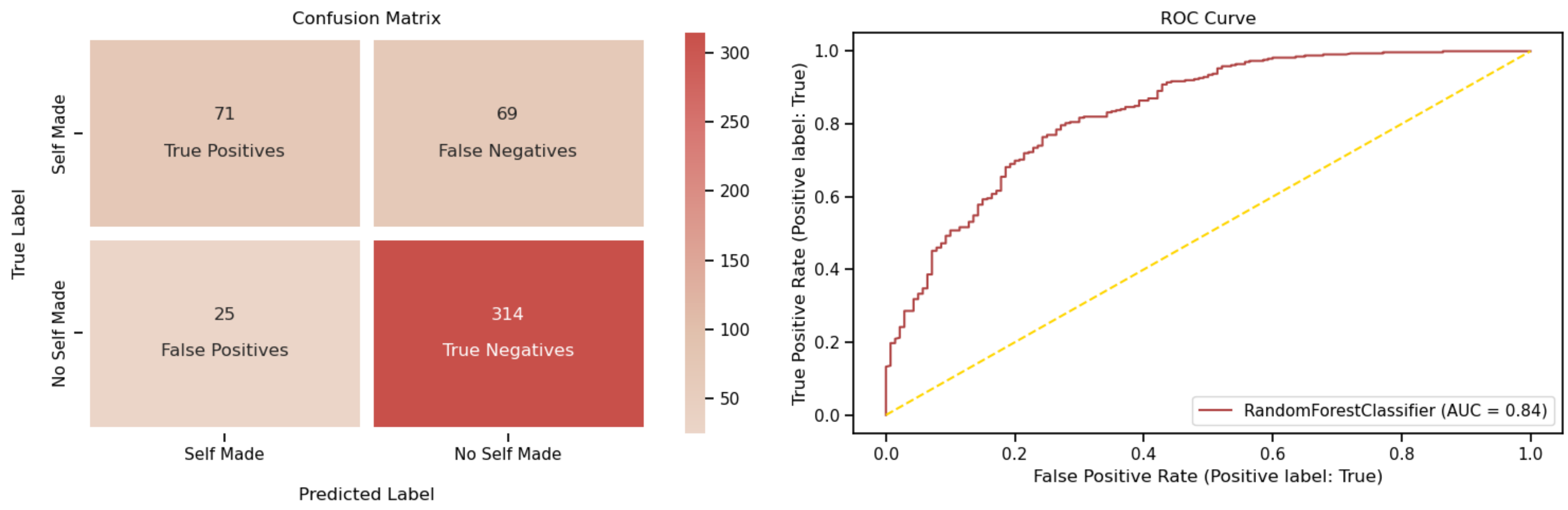
```
RandomForestClassifier(criterion='entropy', max_depth=10, max_features='log2',
                       min_samples_split=10, n_estimators=50, random_state=25)
```

In [90]:

```
from sklearn.metrics import confusion_matrix, roc_curve, classification_report
from sklearn.metrics import f1_score, precision_score, recall_score, roc_auc_score,auc,RocCurveDisplay
# Subplot 1: Confusion Matrix
from matplotlib.colors import LinearSegmentedColormap

cm = confusion_matrix(y_test,y_test_pred ,labels=[0,1])
plt.subplots_adjust(hspace=0.5)
warm=LinearSegmentedColormap.from_list('warm',
                                       [(0, '#EBD5C8'),
                                        (0.25, '#E1C1AD'),
                                        (.75, '#D77873'),
                                        (1, '#C8504A')], N=256)

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(15,5))

classes=['Self Made','No Self Made']
label = ['True Positives', 'False Negatives', 'False Positives', 'True Negatives']
annot = [f'{i}\n\n{j}' for i, j in zip(cm.flatten(), label)]
annot = np.asarray(annot).reshape(2,2)

sns.set_context('notebook')
sns.heatmap(cm, annot=annot, fmt='', cmap=warm, linecolor='white', linewidths=8, ax=ax[0])
ax[0].set_title('Confusion Matrix')
ax[0].set_xlabel('\nPredicted Label')
ax[0].set_ylabel('True Label\n')
ax[0].xaxis.set_ticklabels(classes)
ax[0].yaxis.set_ticklabels(classes)

# Subplot 2: AUC

y_proba= best_model.predict_proba(X_test)
fpr, tpr, _ = roc_curve(y_test, y_proba[:,1])
roc_auc = auc(fpr, tpr)
RocCurveDisplay.from_estimator(best_model, X_test, y_test,ax=ax[1],color="#AF4343")
sns.lineplot(x = [0, 1], y = [0, 1], color = 'gold', linestyle="dashed",ax=ax[1])
ax[1].set_title('ROC Curve')

plt.tight_layout()
plt.show();
```

```
40
```

```
<Figure size 640x480 with 0 Axes>
```



In [ ]:

```
1
```