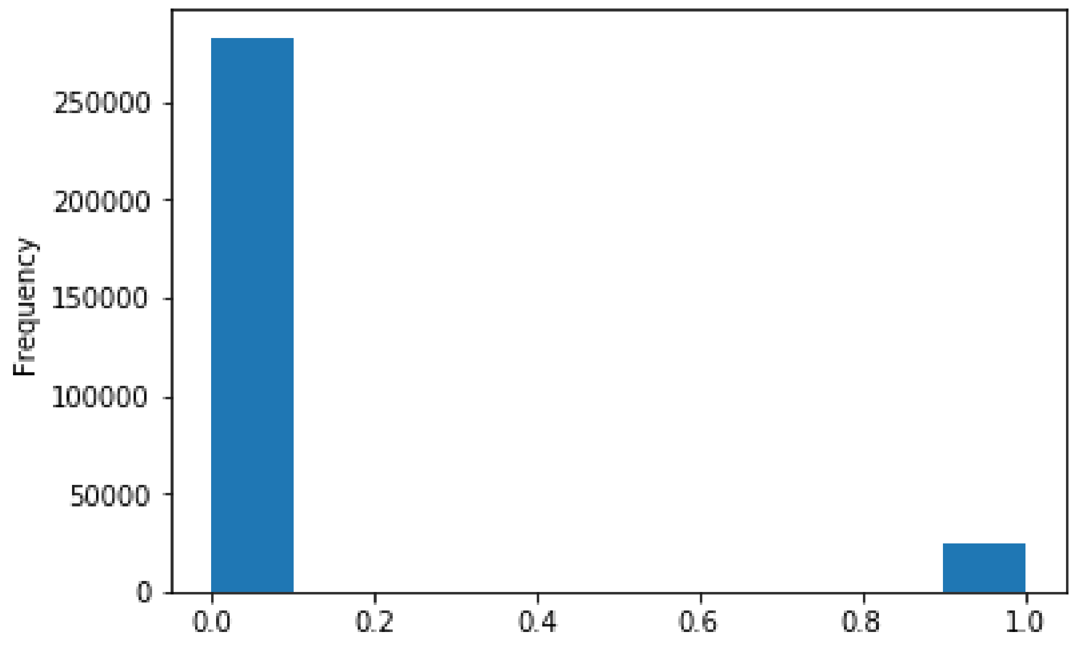
```
#Import numpy and pandas for data manipulation

import numpy as np

import pandas as pd


#File system manangement

import os


#matplotlib and seaborn for plotting

import matplotlib.pyplot as plt

import seaborn as sns


#sklearn preprocessing for dealing with categorical variables

from sklearn.preprocessing import LabelEncoder


#Suppress warnings

import warnings

warnings.filterwarnings('ignore')


pd.options.display.max_columns = 125

pd.options.display.max_rows = 125


#Import Application Data

app_data = pd.read_csv('Input/application_data.csv')

print('Input data shape: ', app_data.shape)

app_data.head()


#The target is what we are asked to predict: either a 0 for the loan was repaid on time, or a 1
indicating the client had payment difficulties.

#We can first examine the number of loans falling into each category.

app_data['TARGET'].value_counts()
```

```
#Plotting histogram to display TARGET value - loan was repaid on time or not

app_data['TARGET'].astype(int).plot.hist();

#There are far more loans that were repaid on time than loans that were not repaid.

#From this information, we see this is an imbalanced class problem.

print("Imbalance percentage is:"+str(app_data['TARGET'].eq(1).sum()*100/len(app_data)))
```

Imbalance percentage is:8.072881945686495

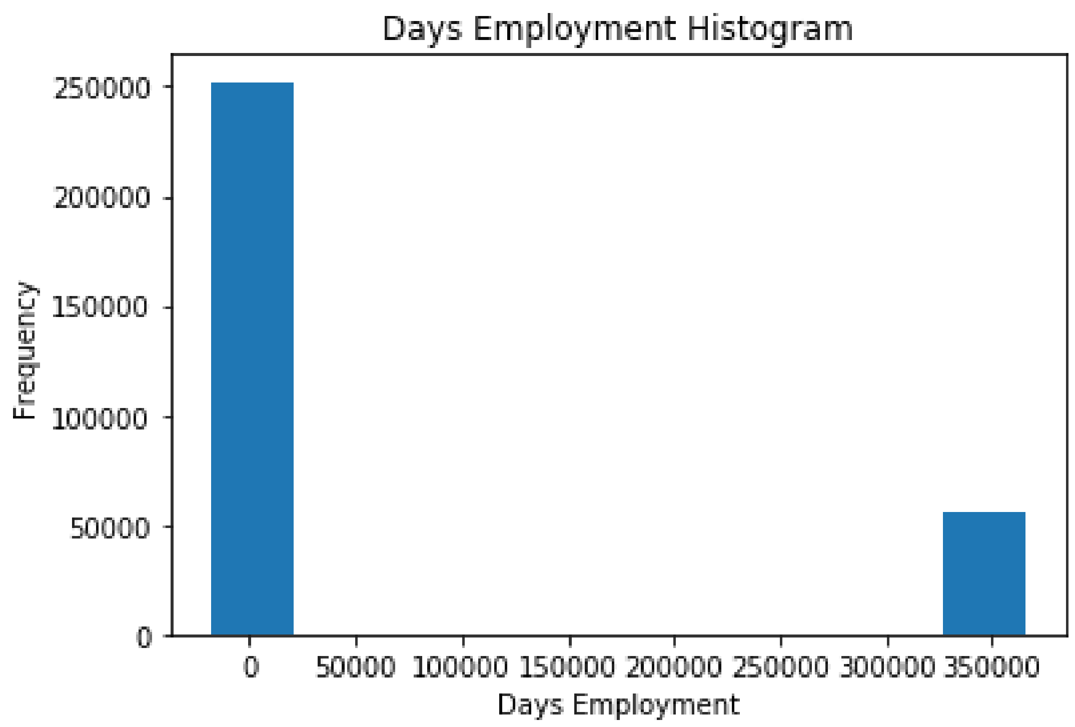


```
#Identify the missing data and use appropriate method to deal with it. (Remove columns/or replace it with an appropriate value)

#Find missing values in each column

#Total missing values

mis_val = app_data.isnull().sum()

#Percentage of missing values

mis_val_percent = app_data.isnull().sum() / len(app_data)

print(mis_val_percent*100)

#Remove columns with high missing percentage say 30%

mis_val_percent = list(mis_val_percent[mis_val_percent.values>=0.30].index)

app_data.drop(labels = mis_val_percent,axis =1,inplace=True)

print('Input data shape after removing missing values greater than 30%: ', app_data.shape)
```

Input data shape after removing missing values greater than 30%: (307511, 72)

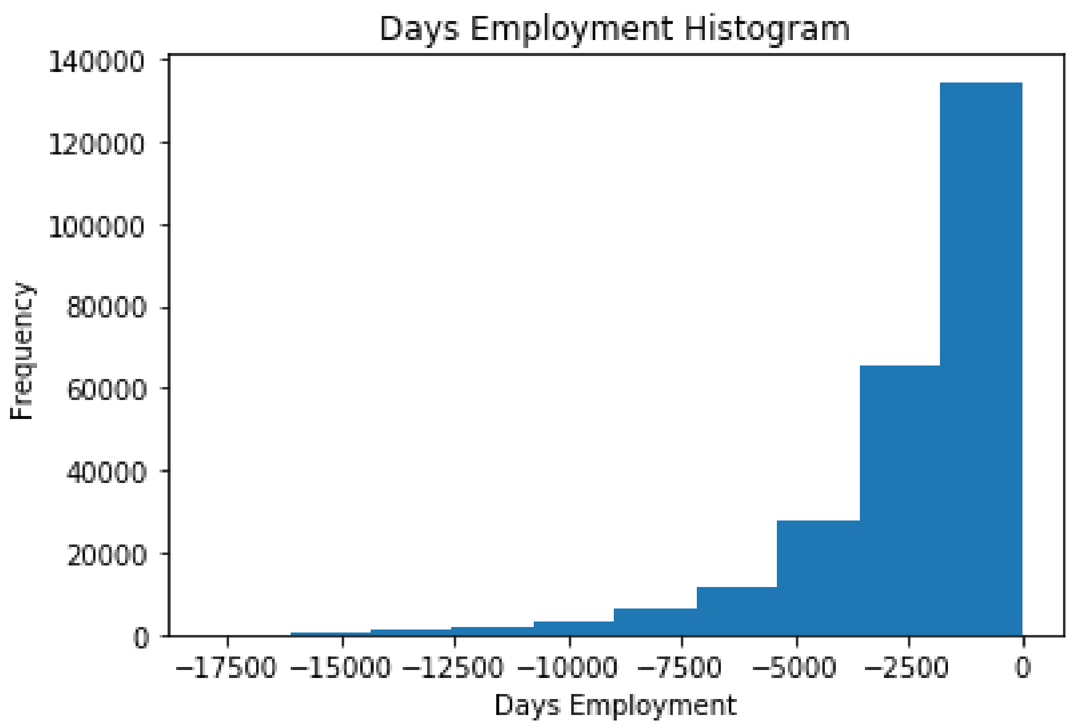
```
#The below code replaces the rows with missing values  for the column
AMT_REQ_CREDIT_BUREAU_DAY with the median value of that column.

app_data.loc[pd.isnull(app_data['AMT_REQ_CREDIT_BUREAU_DAY']),'AMT_REQ_CREDIT_BUREAU_
DAY']=app_data['AMT_REQ_CREDIT_BUREAU_DAY'].median()


#The below code replaces the rows with missing values  for the column
AMT_REQ_CREDIT_BUREAU_WEEK with the median value of that column.

app_data.loc[pd.isnull(app_data['AMT_REQ_CREDIT_BUREAU_WEEK']),'AMT_REQ_CREDIT_BUREAU
_WEEK']=app_data['AMT_REQ_CREDIT_BUREAU_WEEK'].median()


#The below code replaces the rows with missing values  for the column
AMT_REQ_CREDIT_BUREAU_YEAR with the median value of that column.


app_data.loc[pd.isnull(app_data['AMT_REQ_CREDIT_BUREAU_YEAR']),'AMT_REQ_CREDIT_BUREAU
_YEAR']=app_data['AMT_REQ_CREDIT_BUREAU_YEAR'].median()


#Check the datatypes of all the columns and change the datatype if required.

# Number of each type of column

app_data.dtypes.value_counts()


#In the provided dataset we have negative vlues provided for
DAYS_BIRTH,DAYS_EMPLOYED,DAYS_REGISTRATION,DAYS_ID_PUBLISH etc

#as they are recorded relative to the current loan application

#To see these stats in years, we can mutliple by -1 and divide by the number of days in a year:

(app_data['DAYS_BIRTH'] / -365).describe()


#Those ages look reasonable. There are no outliers for the age on either the high or low end. How
about the days of employment

(app_data['DAYS_EMPLOYED']).describe()


#That doesn't look right..The maximum value besides being positive is about 1000 years!!

app_data['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
```

```
plt.xlabel('Days Employment');
```



```
#Just out of curiousity, let's subset the anomalous clients and see if they tend to have higher or low rates of default than

#the rest of the clients.

anom = app_data[app_data['DAYS_EMPLOYED'] == 365243]

non_anom = app_data[app_data['DAYS_EMPLOYED'] != 365243]

print('The non-anomalies default on %0.2f%% of loans' % (100 * non_anom['TARGET'].mean()))

print('The anomalies default on %0.2f%% of loans' % (100 * anom['TARGET'].mean()))

print('There are %d anomalous days of employment' % len(anom))


#From above calculation It turns out that the anomalies have a lower rate of default.

#To find anamolies we will fill in the anomalous values with not a number (np.nan) and then create a new boolean column indicating whether or not the value was anomalous.

#Create an anomalous flag column

app_data['DAYS_EMPLOYED_ANOM'] = app_data["DAYS_EMPLOYED"] == 365243


#Replace the anomalous values with nan

app_data['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
```

```
app_data['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram')

plt.xlabel('Days Employment')
```



#The distribution looks to be much more in line with what we would expect, and we also have created a new column to tell

#the model that these values were originally anomalous (becuase we will have to fill in the nans with some value,

#probably the median of the column). The other columns with DAYS in the dataframe look to be about what we expect with no obvious outliers.

#Now that we have dealt with the categorical variables and the outliers, let's continue with the EDA. One way to try and understand the data is by looking for correlations between the features and the target. We can calculate the Pearson correlation coefficient between every variable and the target using the .corr dataframe method.

#The correlation coefficient is not the greatest method to represent "relevance" of a feature, but it does give us an idea of possible relationships within the data. Some general interpretations of the absolute value of the correlation coefficent are:

#.00-.19 "very weak"

#.20-.39 “weak”

```
#.40-.59 “moderate”
#.60-.79 “strong”
#.80-1.0 “very strong”

# Find correlations with the target and sort
correlations = app_data.corr()['TARGET'].sort_values()

# Display correlations
print('Most Positive Correlations:\n', correlations.tail(15))
print('\nMost Negative Correlations:\n', correlations.head(15))
#The 3 variables with the strongest negative correlations with the target are EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3.
#According to the documentation, these features represent a "normalized score from external data source".



#First, we can show the correlations of the EXT_SOURCE features with the target and with each other.

# Extract the EXT_SOURCE variables and show correlations
ext_data = app_data[['TARGET', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH']]
ext_data_corrs = ext_data.corr()
ext_data_corrs

plt.figure(figsize = (8, 6))

# Heatmap of correlations
sns.heatmap(ext_data_corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = True, vmax = 0.6)
plt.title('Correlation Heatmap')

#Let's take a look at some of more significant correlations: the DAYS_BIRTH is the most positive correlation.
```

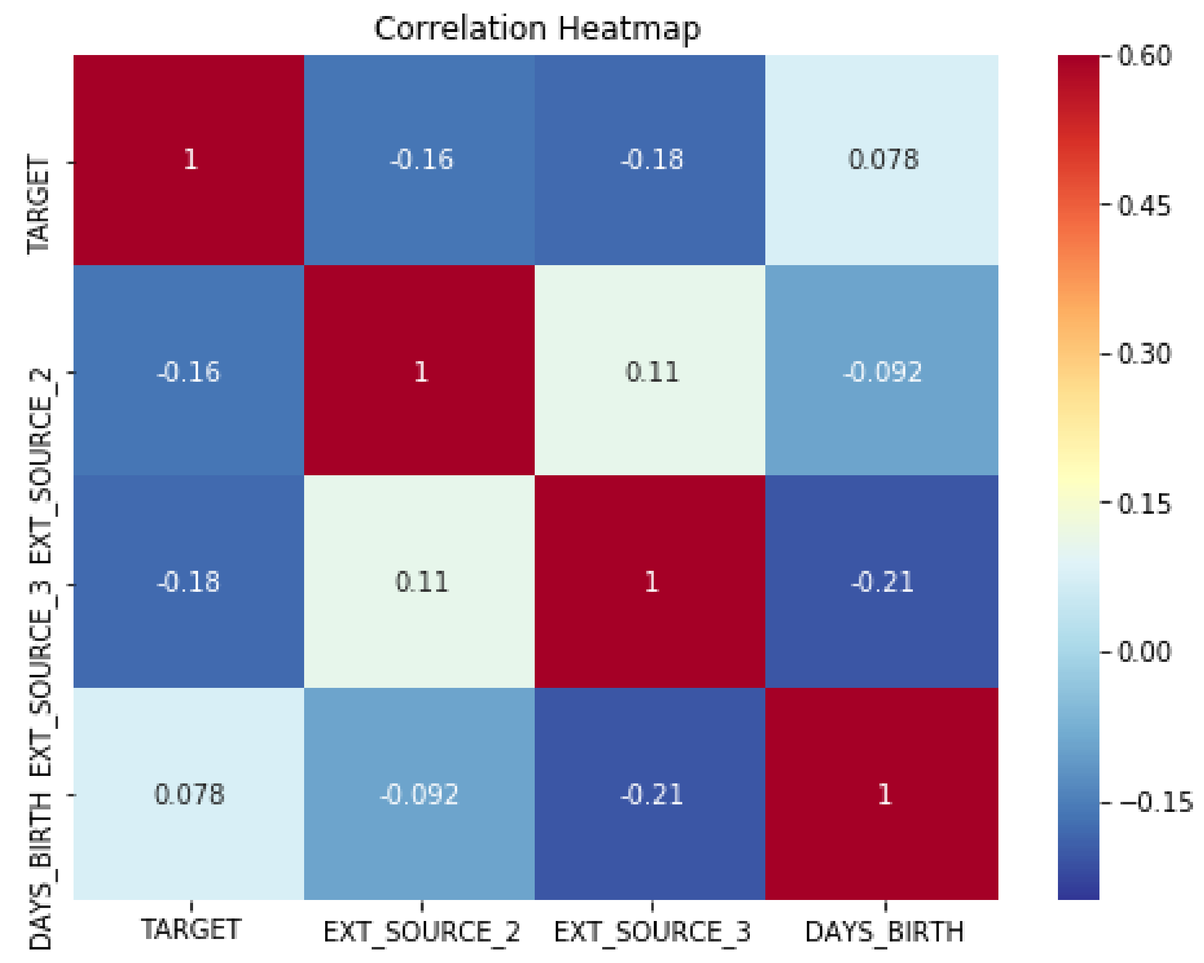
#(except for TARGET because the correlation of a variable with itself is always 1!) Looking at the documentation,

#DAYS_BIRTH is the age in days of the client at the time of the loan in negative days (for whatever reason!). The correlation is

#positive, but the value of this feature is actually negative, meaning that as the client gets older, they are less likely to

#default on their loan (ie the target == 0). That's a little confusing, so we will take the absolute value of the feature and then

#the correlation will be negative.



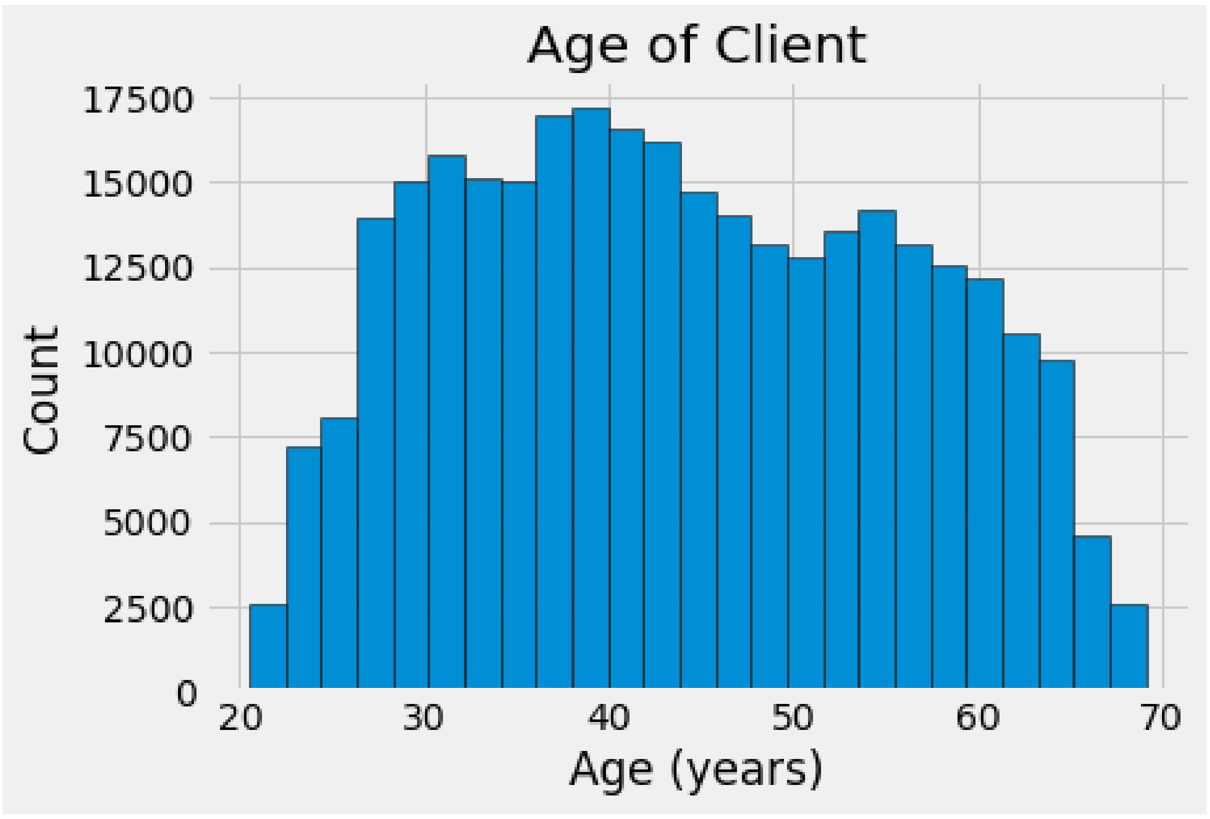
# Find the correlation of the positive days since birth and target

app_data['DAYS_BIRTH'] = abs(app_data['DAYS_BIRTH'])

app_data['DAYS_BIRTH'].corr(app_data['TARGET'])

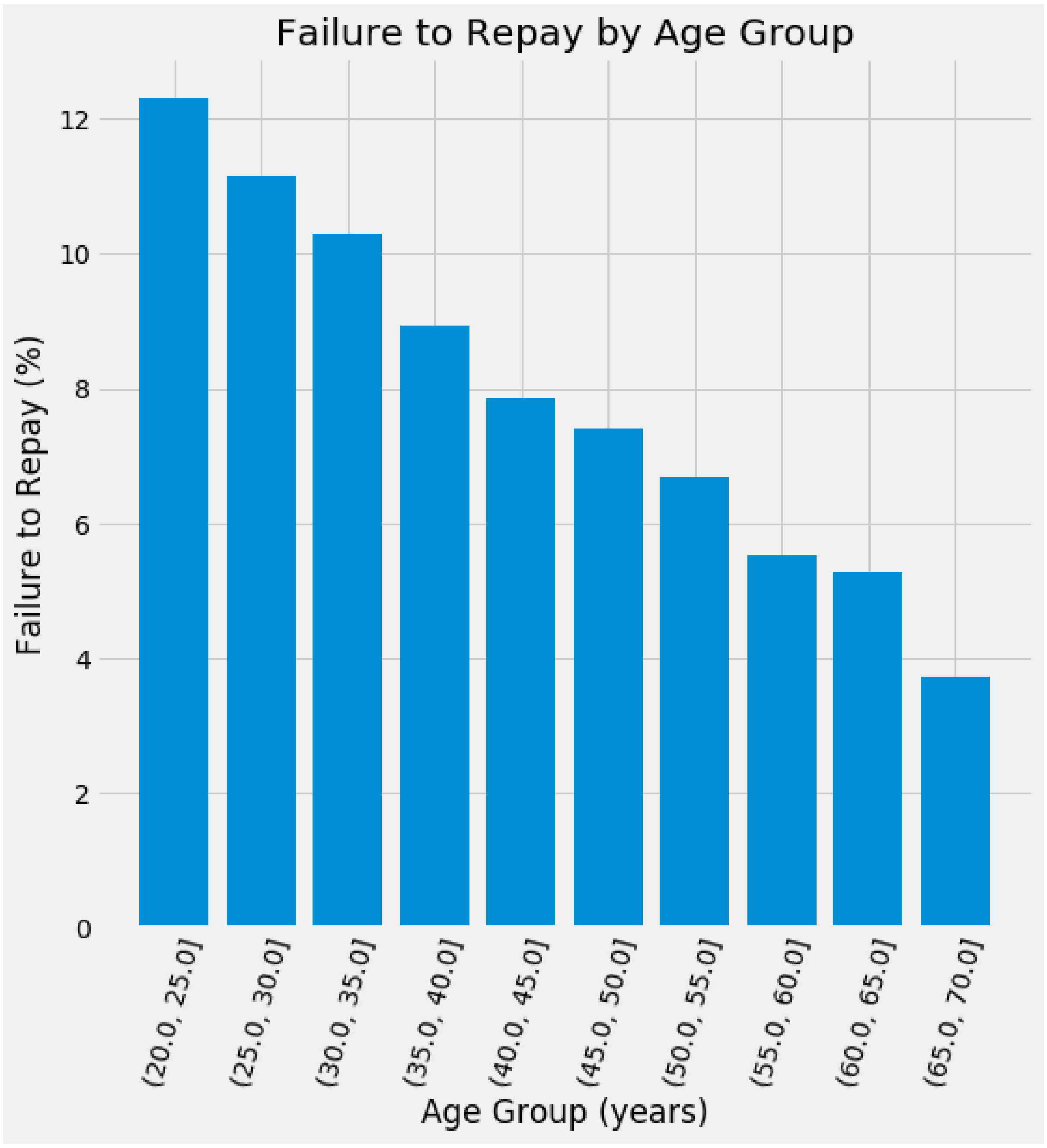
#As the client gets older, there is a negative linear relationship with the target meaning that as clients get older,

```
#they tend to repay their loans on time more often.


#Let's start looking at this variable. First, we can make a histogram of the age.

#We will put the x axis in years to make the plot a little more understandable

# Set the style of plots

plt.style.use('fivethirtyeight')


# Plot the distribution of ages in years

plt.hist(app_data['DAYS_BIRTH'] / 365, edgecolor = 'k', bins = 25)

plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```



```
#By itself, the distribution of age does not tell us much other than that there are no outliers

#as all the ages are reasonable. To visualize the effect of the age on the target

#Now we will first cut the age category into bins of 5 years each. Then, for each bin, we calculate

#the average value of the target, which tells us the ratio of loans that were not repaid in each age
category.
```

```
# Age information into a separate dataframe

age_data = app_data[['TARGET', 'DAYS_BIRTH']]

age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365


# Bin the age data

age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20, 70, num = 11))

age_data.head(10)


age_groups  = age_data.groupby('YEARS_BINNED').mean()

age_groups


plt.figure(figsize = (8, 8))


# Graph the age bins and the average of the target as a bar plot

plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])


# Plot labeling

plt.xticks(rotation = 75); plt.xlabel('Age Group (years)'); plt.ylabel('Failure to Repay (%)')

plt.title('Failure to Repay by Age Group');
```



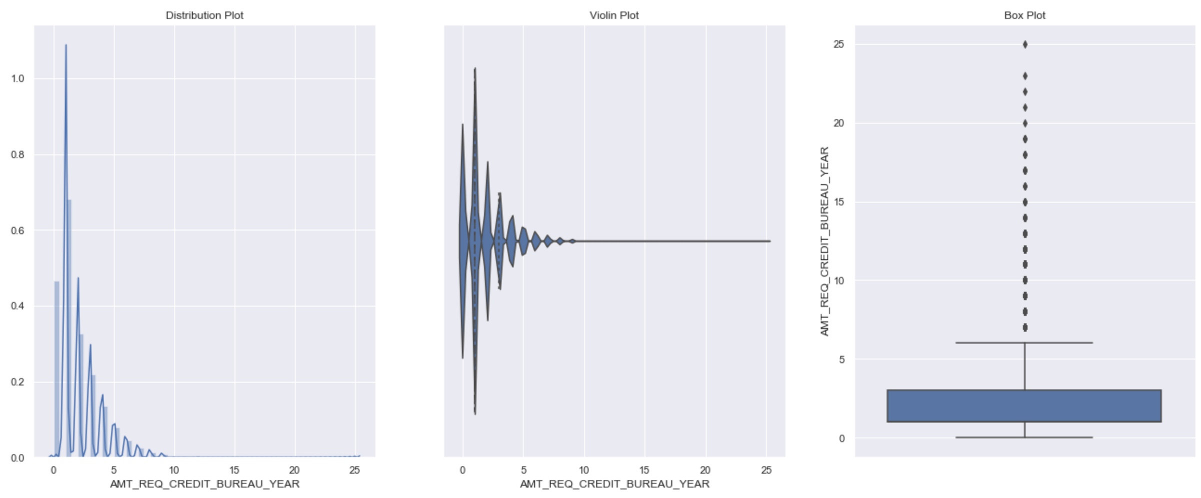
#There is a clear trend: younger applicants are more likely to not repay the loan!

#The rate of failure to repay is above 10% for the youngest three age groups and beolow 5% for the oldest age group.

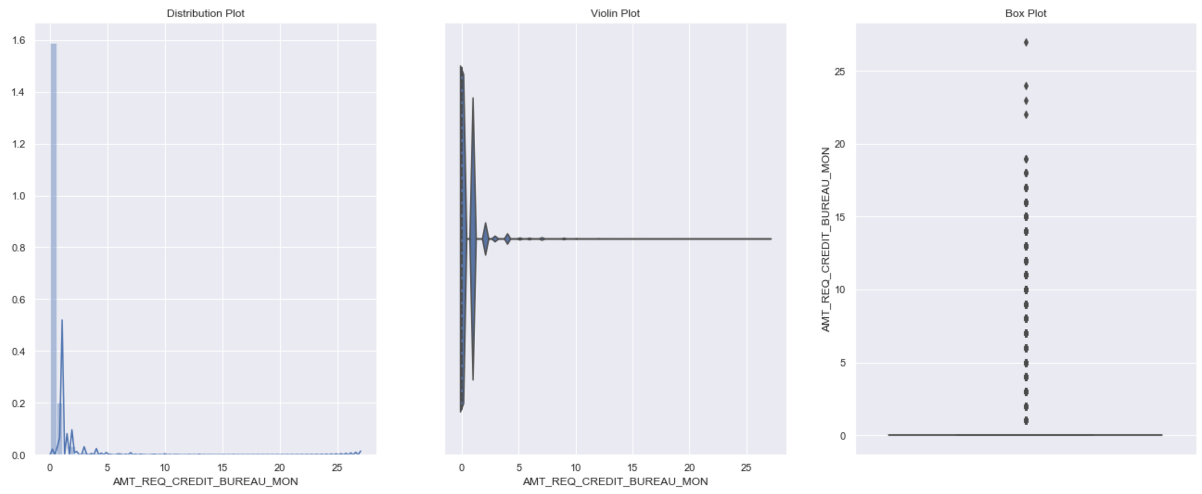
#This is information that could be directly used by the bank: because younger clients are less likely to repay the loan,

#maybe they should be provided with more guidance or financial planning tips. This does not mean the bank should discriminate

```
#against younger clients, but it would be smart to take precautionary measures to help younger
clients pay on time.

 #Univariate function will plot the graphs based on the parameter values.

def univariate(df,col,vartype,hue =None):


    '''

    Univariate function will plot the graphs based on the parameters.

    df      : dataframe name

    col     : Column name

    vartype : variable type : continuos or categorical

            Continuos(0)   : Distribution, Violin & Boxplot will be plotted.

            Categorical(1) : Countplot will be plotted.

    hue     : It's only applicable for categorical analysis.


    '''

    sns.set(style="darkgrid")


    if vartype == 0:

        fig, ax=plt.subplots(nrows =1,ncols=3,figsize=(20,8))

        ax[0].set_title("Distribution Plot")

        sns.distplot(df[col],ax=ax[0])

        ax[1].set_title("Violin Plot")

        sns.violinplot(data =df, x=col,ax=ax[1], inner="quartile")

        ax[2].set_title("Box Plot")

        sns.boxplot(data =df, x=col,ax=ax[2],orient='v')


    if vartype == 1:

        temp = pd.Series(data = hue)

        fig, ax = plt.subplots()

        width = len(df[col].unique()) + 6 + 4*len(temp.unique())

        fig.set_size_inches(width , 7)
        ax = sns.countplot(data = df, x= col, order=df[col].value_counts().index,hue = hue)
        if len(temp.unique()) > 0:
            for p in ax.patches:
                ax.annotate('{:1.1f}%'.format((p.get_height()*100)/float(len(loan))), (p.get_x()+0.05,
p.get_height()+20))
        else:
            for p in ax.patches:
                ax.annotate(p.get_height(), (p.get_x()+0.32, p.get_height()+20))
        del temp
    else:
        exit

    plt.show()

#AMT_REQ_CREDIT_BUREAU_YEAR
univariate(df=app_data,col='AMT_REQ_CREDIT_BUREAU_YEAR',vartype=0)
```



```
#AMT_REQ_CREDIT_BUREAU_MON
univariate(df=app_data,col='AMT_REQ_CREDIT_BUREAU_MON',vartype=0)
```

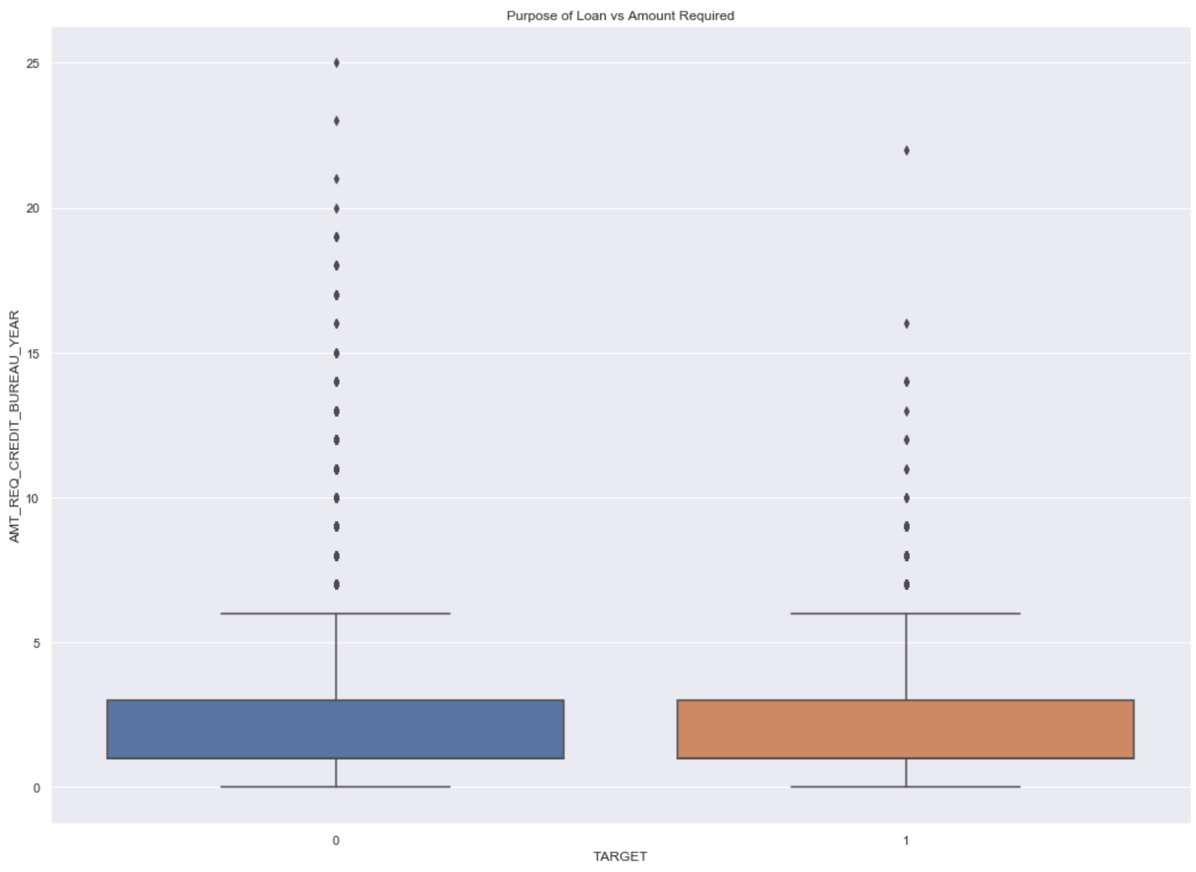


#Bivariate/Multivariate Analysis finds out the relationship between two/two or more variables.

#We can perform Bivariate/Multivariate analysis for any combination of categorical and continuous variables.

#The combination can be: Categorical & Categorical, Categorical & Continuous and Continuous & Continuous.

```
plt.figure(figsize=(16,12))

sns.boxplot(x='TARGET', y='AMT_REQ_CREDIT_BUREAU_YEAR', data=app_data)

plt.title('Loan Status vs Amount Required Yearly')

plt.show()
```



Analysis conclusion:

Target Variable

- Loan Status

Top-5 Major variables to consider for loan prediction:

1. Purpose of Loan
2. Employment Length
3. Age of applicant
4. Interest Rate
5. Term