Multidimensional Data Analysis

1. Goal of Analysis

The goal of this project is to analyze the provided credit card approval data to identify patterns and relationships between the applicant's information and the likelihood of their credit card application being approved. This analysis will aim to develop predictive models that assess the probability of approval based on various factors.

2. Description of Input Data

The input data consists of a file containing information about credit card applicants. The dataset includes variables such as applicant's age, income, employment length, education, marital status, home ownership, credit score, and loan amount. The target variable is the binary response of whether the applicant's credit card application was approved or not.

ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_HOUSING_TYPE	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL	JOB	BEGIN_MONTHS	STATUS	TARGET
id	*x1*	*x2*	*x3*	*x4*	*x5*	*x6*	*x7*	*x8*	*x9*	*x10*	*x11*	*x12*	*x13*	*x14*	*x15*	*x16*	*x17*	*y1*
5065438	F	Y	N	2+ children	270000	Secondary / secondary special	Married	With parents	-13258	-2300	1	0	0	0	Managers	-6	C	0
5142753	F	N	N	No children	81000	Secondary / secondary special	Single / not married	House / apartment	-17876	-377	1	1	1	0	Private service staff	-4	0	0
5111146	M	Y	Y	No children	270000	Higher education	Married	House / apartment	-19579	-1028	1	0	1	0	Laborers	0	C	0
5010310	F	Y	Y	1 children	112500	Secondary / secondary special	Married	House / apartment	-15109	-1956	1	0	0	0	Core staff	-3	0	0
5010835	M	Y	Y	2+ children	139500	Secondary / secondary special	Married	House / apartment	-17281	-5578	1	1	0	0	Drivers	-29	0	0
5067057	F	Y	Y	No children	144000	Secondary / secondary special	Married	House / apartment	-15394	-2959	1	0	1	0	Core staff	-25	0	0
5095635	M	Y	N	1 children	180000	Higher education	Married	House / apartment	-11178	-219	1	0	0	0	Drivers	-19	X	0
5096402	M	Y	N	No children	405000	Higher education	Married	House / apartment	-18655	-3200	1	1	1	0	High skill tech staff	-18	X	0
5061372	F	N	Y	1 children	135000	Secondary / secondary special	Single / not married	House / apartment	-17068	-8325	1	0	0	0	Laborers	-43	0	0
5026464	F	N	Y	1 children	270000	Secondary / secondary special	Married	House / apartment	-16616	-2722	1	1	1	0	Realty agents	-38	0	0
5026032	M	Y	Y	No children	99000	Secondary / secondary special	Married	Rented apartment	-9928	-1531	1	0	0	0	Managers	-15	0	0
5096494	F	N	N	2+ children	103500	Higher education	Married	House / apartment	-12987	-3537	1	0	1	0	High skill tech staff	-30	X	0
5058466	F	N	Y	2+ children	225000	Secondary / secondary special	Civil marriage	House / apartment	-12486	-1816	1	0	0	0	Laborers	-26	0	0
5088843	F	N	Y	No children	171000	Higher education	Single / not married	House / apartment	-10224	-2073	1	0	0	0	Laborers	-8	X	0
5149050	F	N	N	2+ children	135000	Secondary / secondary special	Married	House / apartment	-15050	-4977	1	0	0	0	Laborers	-39	0	0
5023566	M	Y	Y	2+ children	270000	Secondary / secondary special	Married	House / apartment	-12323	-1117	1	0	1	0	Laborers	-12	C	0
5095306	F	Y	Y	No children	225000	Higher education	Married	House / apartment	-12322	-3717	1	0	0	0	Core staff	-52	X	0
5022248	F	N	Y	No children	202500	Secondary / secondary special	Married	House / apartment	-15231	-8375	1	0	0	1	High skill tech staff	-7	C	0
5023934	F	N	Y	2+ children	135000	Secondary / secondary special	Separated	House / apartment	-14874	-2407	1	0	0	0	Core staff	-20	0	0
5105886	F	N	N	2+ children	67500	Higher education	Married	House / apartment	-12231	-3072	1	1	1	0	Secretaries	-35	C	0
5022044	F	Y	N	1 children	225000	Secondary / secondary special	Separated	House / apartment	-11714	-1740	1	0	0	1	Core staff	-25	C	0
5095735	M	N	N	No children	405000	Higher education	Married	House / apartment	-18267	-2045	1	1	0	0	Managers	-18	C	0
5021445	F	N	Y	No children	121500	Secondary / secondary special	Married	With parents	-13315	-5204	1	1	1	0	Core staff	-15	C	0
5095851	F	N	N	No children	225000	Secondary / secondary special	Widow	House / apartment	-20657	-5637	1	0	0	0	Accountants	-32	X	0
5050878	F	N	N	No children	270000	Secondary / secondary special	Married	House / apartment	-13101	-2204	1	1	1	0	Accountants	-2	C	0
5116121	F	Y	Y	1 children	560250	Secondary / secondary special	Single / not married	House / apartment	-12987	-2330	1	0	0	0	Sales staff	-20	0	0
5117760	F	N	N	No children	74250	Secondary / secondary special	Married	House / apartment	-19931	-3141	1	1	0	0	Core staff	-11	0	0
5135918	M	N	Y	2+ children	90000	Secondary / secondary special	Married	House / apartment	-15988	-566	1	1	1	0	Laborers	-30	0	0

X's attributes:

ID:	Client Number
CODE_GENDER:	Gender
FLAG_OWN_CAR:	Is there a car
FLAG_OWN_REALTY:	Is there a property
CNT_CHILDREN:	Number of Children
AMT_INCOME_TOTAL:	Annual Income

NAME_EDUCATION_TYPE:	Education Level
NAME_FAMILY_STATUS:	Marital Status
NAME_HOUSING_TYPE:	Way of Living
DAYS_BIRTH:	Age in days
DAYS_EMPLOYED:	Duration of work in days
FLAG_MOBIL:	Is there a mobile phone
FLAG_WORK_PHONE:	Is there a work phone
FLAG_PHONE:	Is there a phone
FLAG_EMAIL:	Is there an email
JOB:	Job
BEGIN_MONTHS:	Record month (The month of the extracted data is the starting point, backwards, 0 is the current month, -1 is the previous month, and so on)
STATUS:	Status (0: 1-29 days past due 1: 30-59 days past due 2: 60-89 days overdue 3: 90-119 days overdue 4: 120-149 days overdue 5: Overdue or bad debts, write-offs for more than 150 days C: paid off that month X: No loan for the month)

Y's attributes:

TARGET: Target (Risk user are marked as '1', else are '0')

3. Data Preparation

```
[2] # Check for missing values
    missing_values = data.isna().sum()
    print("Missing values:\n",missing_values)

    Missing values:
     ID                     0
    CODE_GENDER            0
    FLAG_OWN_CAR           0
    FLAG_OWN_REALTY        0
    CNT_CHILDREN           0
    AMT_INCOME_TOTAL       0
    NAME_EDUCATION_TYPE    0
    NAME_FAMILY_STATUS     0
    NAME_HOUSING_TYPE      0
    DAYS_BIRTH             0
    DAYS_EMPLOYED          0
    FLAG_MOBIL             0
    FLAG_WORK_PHONE        0
    FLAG_PHONE             0
    FLAG_EMAIL             0
    JOB                    0
    BEGIN_MONTHS           0
    STATUS                 0
    TARGET                 0
    dtype: int64
```

```
[3] # Handle missing numerical values
    imputer = SimpleImputer(strategy="median")
    numerical_data = data[["ID", "DAYS_BIRTH", "DAYS_EMPLOYED", "AMT_INCOME_TOTAL"]]
    numerical_imputed_data = imputer.fit_transform(numerical_data)
    numerical_imputed_df = pd.DataFrame(numerical_imputed_data, columns=["ID","DAYS_BIRTH", "DAYS_EMPLOYED", "AMT_INCOME_TOTAL"])

    data[["ID","DAYS_BIRTH", "DAYS_EMPLOYED", "AMT_INCOME_TOTAL"]] = numerical_imputed_df

    # Handle missing categorical values
    categorical_data = data[["CODE_GENDER", "FLAG_OWN_CAR", "FLAG_OWN_REALTY", "CNT_CHILDREN", "NAME_EDUCATION_TYPE", "NAME_FAMILY_STATUS", "NAME_HOUSING_TYPE", "FLAG_MC
    categorical_imputer = SimpleImputer(strategy="most_frequent")
    categorical_imputed_data = categorical_imputer.fit_transform(categorical_data)
    categorical_imputed_df = pd.DataFrame(np.array(categorical_imputed_data), columns=["CODE_GENDER", "FLAG_OWN_CAR", "FLAG_OWN_REALTY", "CNT_CHILDREN", "NAME_EDUCATION_

    data[["CODE_GENDER", "FLAG_OWN_CAR", "FLAG_OWN_REALTY", "CNT_CHILDREN", "NAME_EDUCATION_TYPE", "NAME_FAMILY_STATUS", "NAME_HOUSING_TYPE", "FLAG_MOBIL", "FLAG_WORK_PH
```

This section is for preparing the credit card approval data for further analysis. Initially we identifying missing values and using appropriate techniques to impute them (There is no missing data but I still implemented the code for Handling missing data just in case). For numerical variables, the median is used, while for categorical variables, the most frequent category is imputed.

```
[4] # Creating new dataset to be modified
    data_cpy = data.copy()
```

```
[5] data_cpy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537667 entries, 0 to 537666
Data columns (total 19 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   ID                   537667 non-null  float64
 1   CODE_GENDER          537667 non-null  object 
 2   FLAG_OWN_CAR         537667 non-null  object 
 3   FLAG_OWN_REALTY      537667 non-null  object 
 4   CNT_CHILDREN         537667 non-null  object 
 5   AMT_INCOME_TOTAL     537667 non-null  float64
 6   NAME_EDUCATION_TYPE  537667 non-null  object 
 7   NAME_FAMILY_STATUS   537667 non-null  object 
 8   NAME_HOUSING_TYPE    537667 non-null  object 
 9   DAYS_BIRTH           537667 non-null  float64
 10  DAYS_EMPLOYED        537667 non-null  float64
 11  FLAG_MOBIL           537667 non-null  object 
 12  FLAG_WORK_PHONE      537667 non-null  object 
 13  FLAG_PHONE           537667 non-null  object 
 14  FLAG_EMAIL           537667 non-null  object 
 15  JOB                  537667 non-null  object 
 16  BEGIN_MONTHS         537667 non-null  object 
 17  STATUS               537667 non-null  object 
 18  TARGET               537667 non-null  object 
dtypes: float64(4), object(15)
memory usage: 77.9+ MB
```

At this step, I created a new dataset to be used for further analysis so the initial one to remains untouched.

```
[6] # Convert the "DAYS_BIRTH" column to positive values
    data_cpy["DAYS_BIRTH"] = data_cpy["DAYS_BIRTH"].apply(abs)
```

```
[7] # If the values greater than zero, that means that the person does not work
    # Convert the "DAYS_EMPLOYED" column to positive values
    data_cpy.loc[(data_cpy['DAYS_EMPLOYED'] > 0), 'DAYS_EMPLOYED'] = 0
    data_cpy["DAYS_EMPLOYED"] = data_cpy["DAYS_EMPLOYED"].apply(abs)
```

```
[8] # Convert the "BEGIN_MONTHS" column to positive values
    data_cpy["BEGIN_MONTHS"] = data_cpy["BEGIN_MONTHS"].apply(abs)
```

```
[9] # Converting categorical values to 1 and 0
    data_cpy = data_cpy.replace({'CODE_GENDER' :{'M' : 1,'F' : 0}})
    data_cpy = data_cpy.replace({'FLAG_OWN_CAR' : {'Y' : 1, 'N' : 0}})
    data_cpy = data_cpy.replace({'FLAG_OWN_REALTY' : {'Y' : 1, 'N' : 0}})
    data_cpy.FLAG_MOBIL = data_cpy.FLAG_MOBIL.astype('int')
    data_cpy.FLAG_WORK_PHONE = data_cpy.FLAG_WORK_PHONE.astype('int')
    data_cpy.FLAG_EMAIL = data_cpy.FLAG_EMAIL.astype('int')
    data_cpy.FLAG_PHONE = data_cpy.FLAG_PHONE.astype('int')
    data_cpy.TARGET = data_cpy.TARGET.astype('int')
    data_cpy.STATUS.replace('X', 0, inplace=True)
    data_cpy.STATUS.replace('C', 0, inplace=True)
    data_cpy.STATUS = data_cpy.STATUS.astype('int')
```

This code focuses on further preprocessing the data after handling missing values. It aims to transform categorical variables into numerical representations that can be used for analysis.

4: Initial Data Analysis

```
[12] # Descriptive statistics
     print('Descriptive statistics:')
     data_cpy.describe().T
```

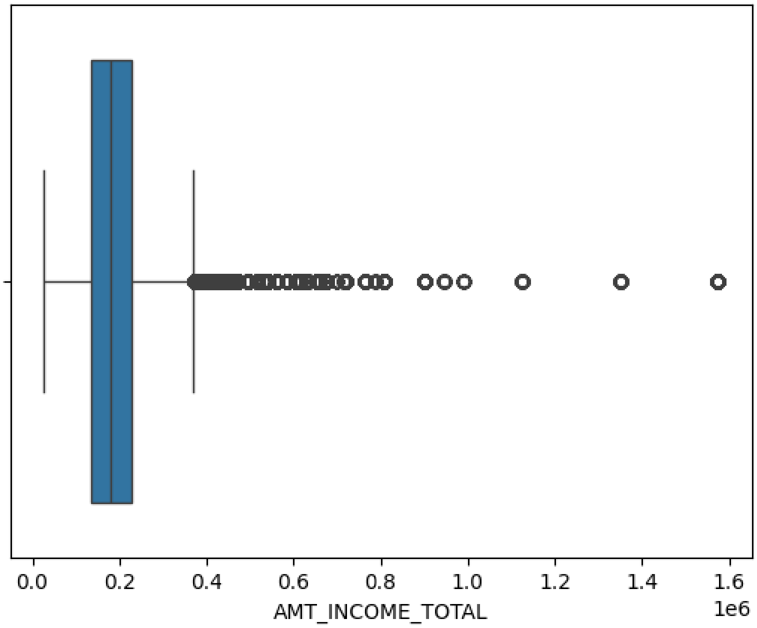
Descriptive statistics:

	count	mean	std	min	25%	50%	75%	max
ID	537667.0	5.079231e+06	42001.999788	5008806.0	5044925.0	5079091.0	5115755.0	5150487.0
CODE_GENDER	537667.0	3.791101e-01	0.485166	0.0	0.0	0.0	1.0	1.0
FLAG_OWN_CAR	537667.0	4.304895e-01	0.495145	0.0	0.0	0.0	1.0	1.0
FLAG_OWN_REALTY	537667.0	6.425371e-01	0.479253	0.0	0.0	1.0	1.0	1.0
AMT_INCOME_TOTAL	537667.0	1.971171e+05	104138.963465	27000.0	135000.0	180000.0	229500.0	1575000.0
DAYS_BIRTH	537667.0	1.501096e+04	3416.418092	7489.0	12239.0	14785.0	17594.0	24611.0
DAYS_EMPLOYED	537667.0	2.762030e+03	2393.919456	17.0	1050.0	2147.0	3661.0	15713.0
FLAG_MOBIL	537667.0	1.000000e+00	0.000000	1.0	1.0	1.0	1.0	1.0
FLAG_WORK_PHONE	537667.0	2.816148e-01	0.449787	0.0	0.0	0.0	1.0	1.0
FLAG_PHONE	537667.0	2.988932e-01	0.457773	0.0	0.0	0.0	1.0	1.0
FLAG_EMAIL	537667.0	1.007296e-01	0.300971	0.0	0.0	0.0	0.0	1.0
BEGIN_MONTHS	537667.0	1.930524e+01	14.037827	0.0	8.0	17.0	29.0	60.0
STATUS	537667.0	2.621139e-02	0.270900	0.0	0.0	0.0	0.0	5.0
TARGET	537667.0	3.649099e-03	0.060298	0.0	0.0	0.0	0.0	1.0

This part of the program is displaying the descriptive statistics for the numerical variables in the dataset. This includes the count, mean, standard deviation, minimum, 25th percentile, 50th percentile, 75th percentile, and maximum for each numerical variable.

At the beginning step of the data analysis, we filter out the data by detecting the outliers. Outliers can distort the distribution of the data and can affect the accuracy of the machine learning models. By removing the outliers, we have more suitable data for analysis and modeling.

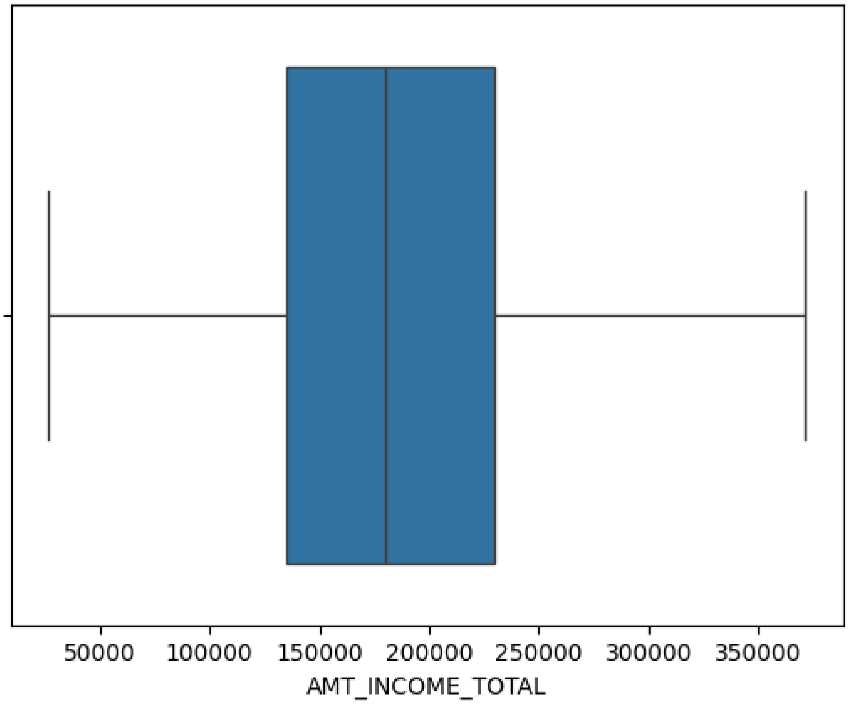
```
[14] # Outlier anlysis for Income (AMT_INCOME_TOTAL)
     outlier_income = data_cpy['AMT_INCOME_TOTAL']
     sns.boxplot(x = outlier_income);
```



This part of the code displays a box plot showing that there are outliers in the AMT_INCOME_TOTAL variable, which is the Annual income. The box plot shows the median, 25th percentile, 75th percentile, and minimum and maximum values for the variable. The outliers are the points that fall outside the whiskers, which extend from the 25th percentile to the 75th percentile plus 1.5 times the interquartile range (IQR). In this case, the outliers are the points that are below 0.2 and above 1.6. This suggests that there may be some data errors or anomalies in the AMT_INCOME_TOTAL variable.

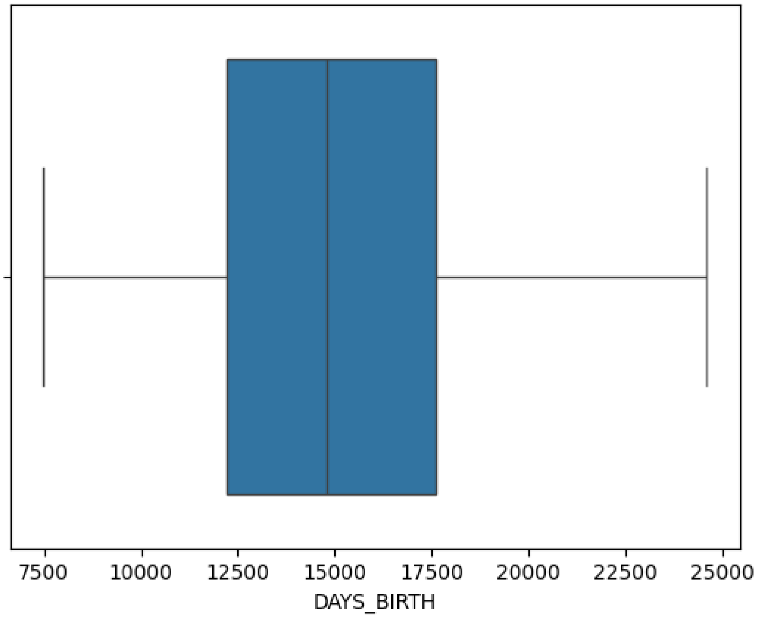
Same principle applies also to the ones from below.

```
[15] Q1 = outlier_income.quantile(0.25)
     Q3 = outlier_income.quantile(0.75)
     IQR = Q3 - Q1
     lower = Q1 - 1.5 * IQR
     upper = Q3 + 1.5 * IQR
     outlier = (outlier_income < lower) | (outlier_income > upper)
     outlier

     0         False
     1         False
     2         False
     3         False
     4         False
               ...
     537662    False
     537663    False
     537664    False
     537665    False
     537666     True
     Name: AMT_INCOME_TOTAL, Length: 537667, dtype: bool

[16] outlier_income[outlier] = upper
     sns.boxplot(x = outlier_income);
```



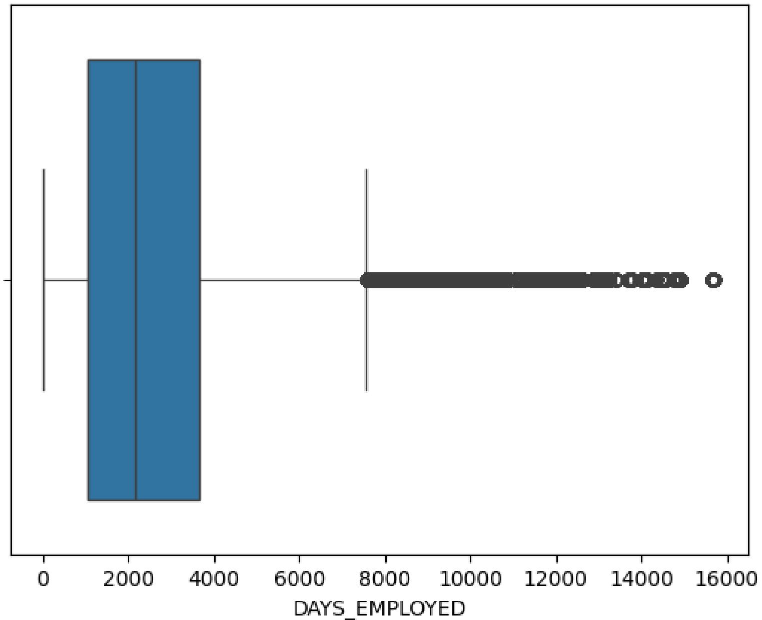
The code snippet identifies and removes outliers from the AMT_INCOME_TOTAL variable using the Interquartile Range (IQR) method. After performing the removal of the outliers, the program shows the updated plot box.

```
[17] #Outlier analysis for Age (DAYS_BIRTH)
     outlier_Age = data_cpy["DAYS_BIRTH"]
     sns.boxplot(x = outlier_Age);
```



There are no outliers for the DAYS_BIRTH variable.

```
[18] #Outlier analysis for Days beign employed (DAYS_EMPLOYED)
     outlier_Days_Employed = data_cpy["DAYS_EMPLOYED"]
     sns.boxplot(x = outlier_Days_Employed)
```

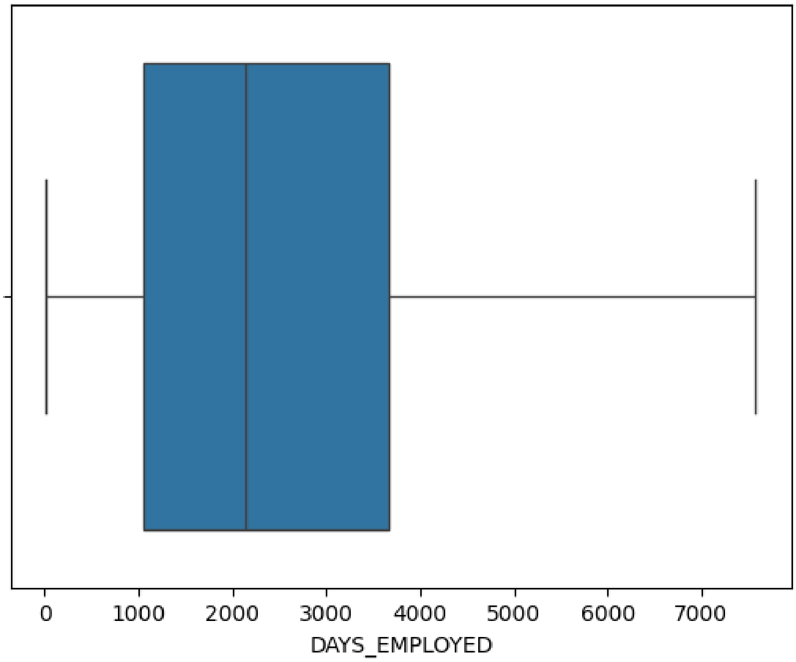


The box plot with the outliers for the variable DAYS_EMPLOYED.

```
[19] Q1 = outlier_Days_Employed.quantile(0.25)
     Q3 = outlier_Days_Employed.quantile(0.75)
     IQR = Q3 - Q1
     lower = Q1 - 1.5 * IQR
     upper = Q3 + 1.5 * IQR
     outlier = (outlier_Days_Employed < lower) | (outlier_Days_Employed > upper)
     outlier
```

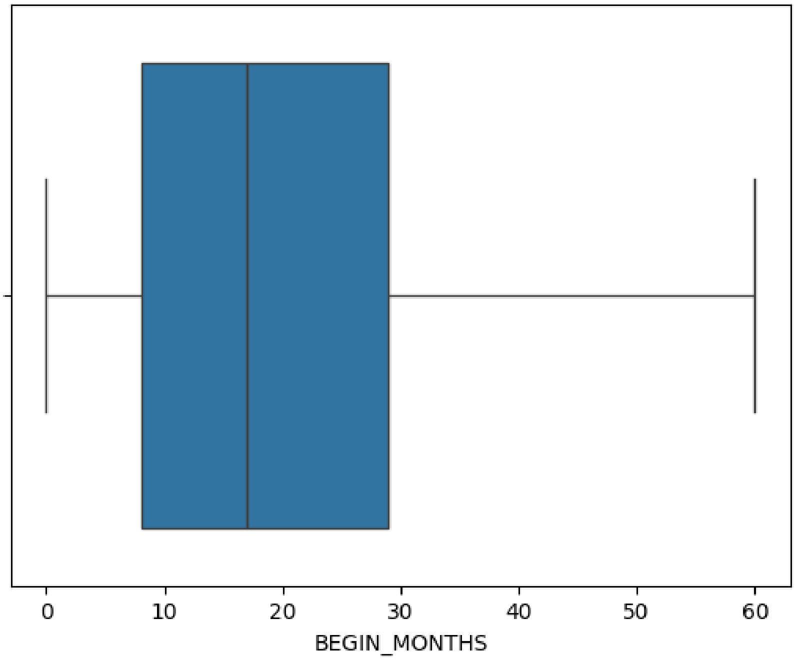
```
0         False
1         False
2         False
3         False
4         False
          ...
537662    False
537663    False
537664    False
537665    False
537666    False
Name: DAYS_EMPLOYED, Length: 537667, dtype: bool
```

```
[20] outlier_Days_Employed[outlier] = upper
     sns.boxplot(x = outlier_Days_Employed);
```



The box plot with the outliers removed for the variable DAYS_EMPLOYED.

```
[21] #Outlier analysis for begin months
     outlier_Begin_Months = data_cpy["BEGIN_MONTHS"]
     sns.boxplot(x = outlier_Begin_Months)
```

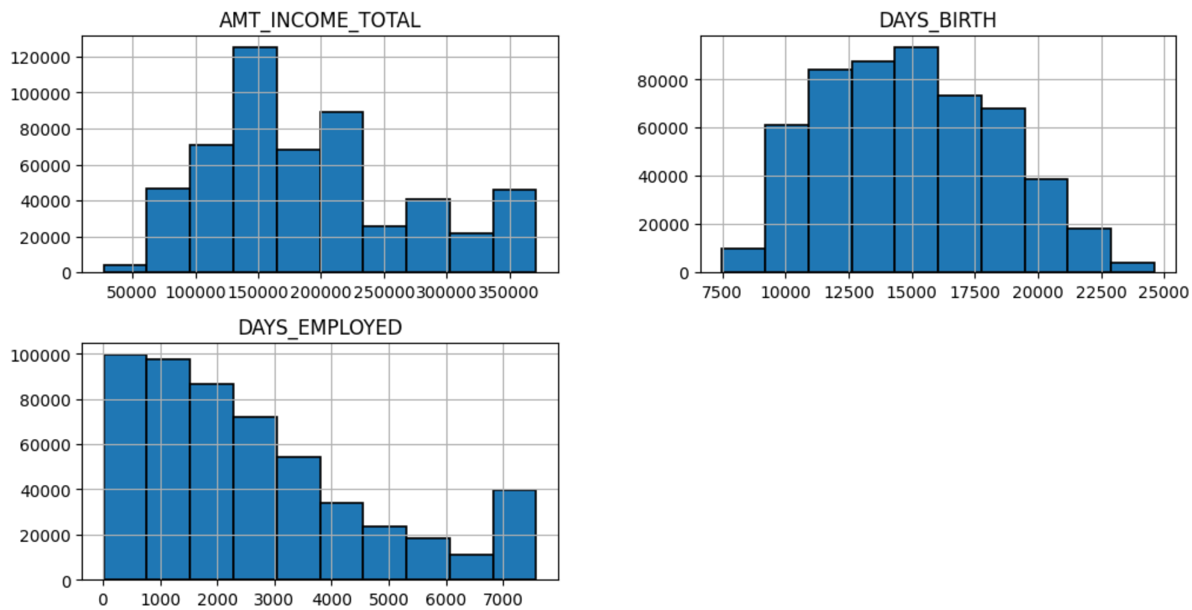


There are no outliers for the BEGIN_MONTHS variable.

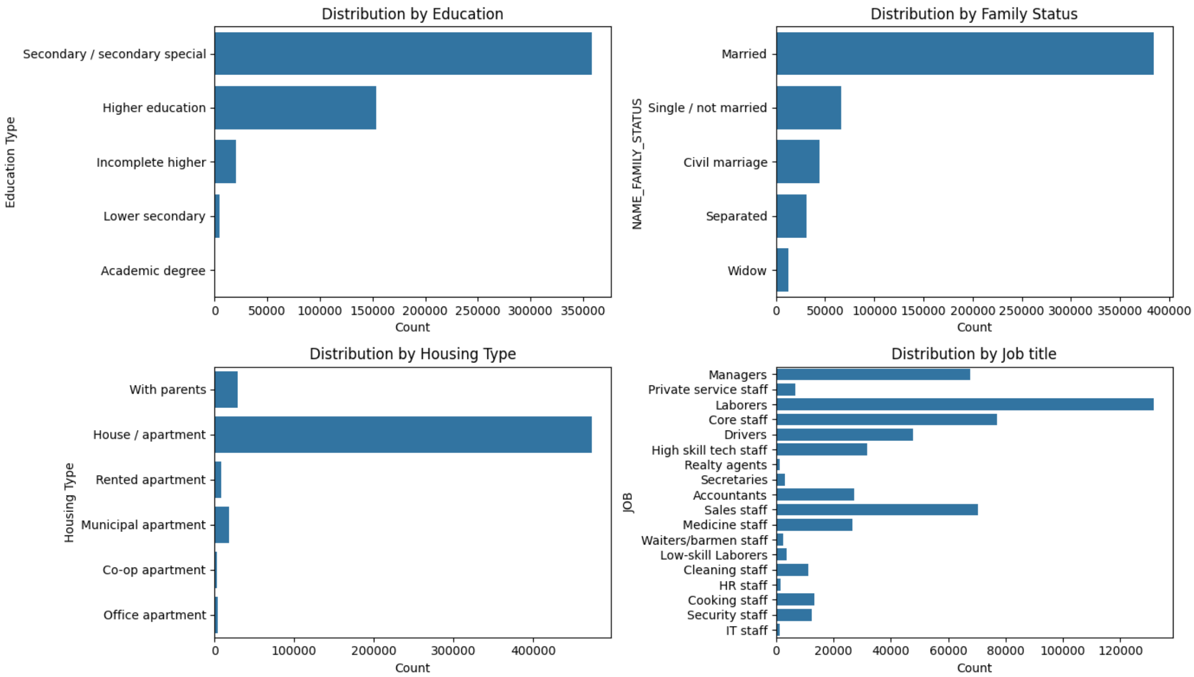
5: Data Visualization

Under this section I start displaying the data visualization of the variables in the dataset.

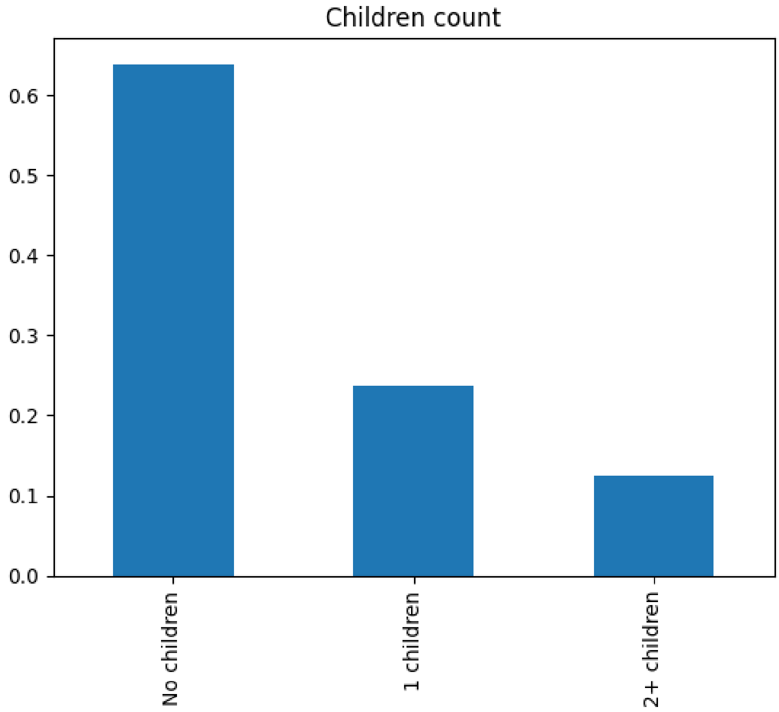
```
[22] plt.figure(figsize=(10,10))

     columns_to_plot = ["AMT_INCOME_TOTAL","DAYS_BIRTH","DAYS_EMPLOYED"]
     data_cpy[columns_to_plot].hist(edgecolor='black', linewidth=1.2)
     fig=plt.gcf()
     fig.set_size_inches(12,6)
```



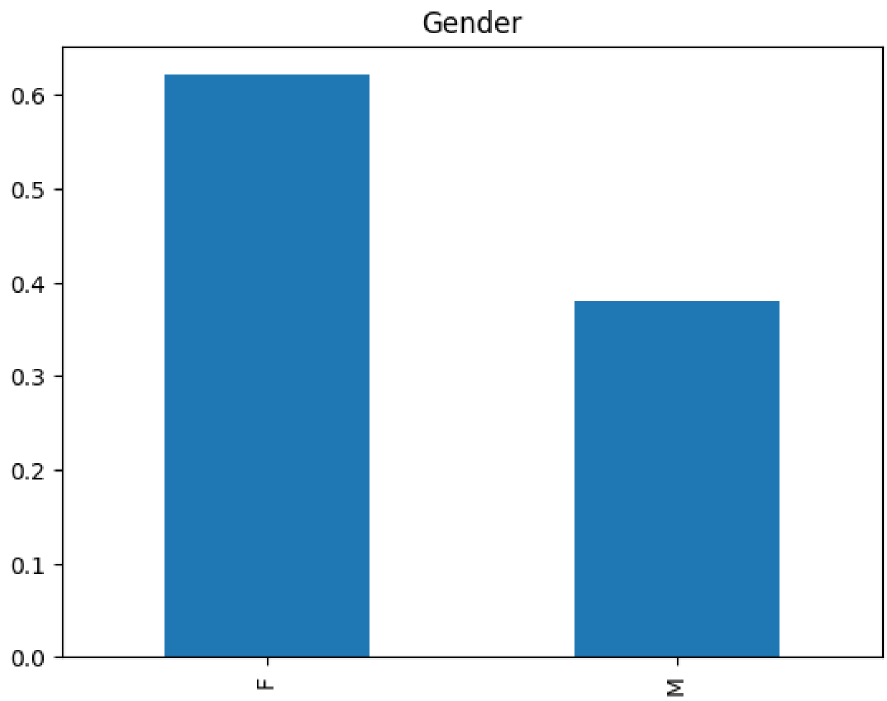
[23]
```
# Distribution plots
fig, axes = plt.subplots(2,2)

g1= sns.countplot(y=data_cpy.NAME_EDUCATION_TYPE, ax=axes[0,0])
g1.set_title("Distribution by Education")
g1.set_xlabel("Count")
g1.set_ylabel("Education Type")

g2=sns.countplot(y=data_cpy.NAME_FAMILY_STATUS,linewidth=1.2, ax=axes[0,1])
g2.set_title("Distribution by Family Status")
g2.set_xlabel("Count")

g3= sns.countplot(y=data_cpy.NAME_HOUSING_TYPE,linewidth=1.2, ax=axes[1,0])
g3.set_title("Distribution by Housing Type")
g3.set_xlabel("Count")
g3.set_ylabel("Housing Type")

g4=sns.countplot(y=data_cpy.JOB,linewidth=1.2, ax=axes[1,1])
g4.set_title("Distribution by Job title")
g4.set_xlabel("Count")

fig.set_size_inches(14,8)

plt.tight_layout()

plt.show()
```



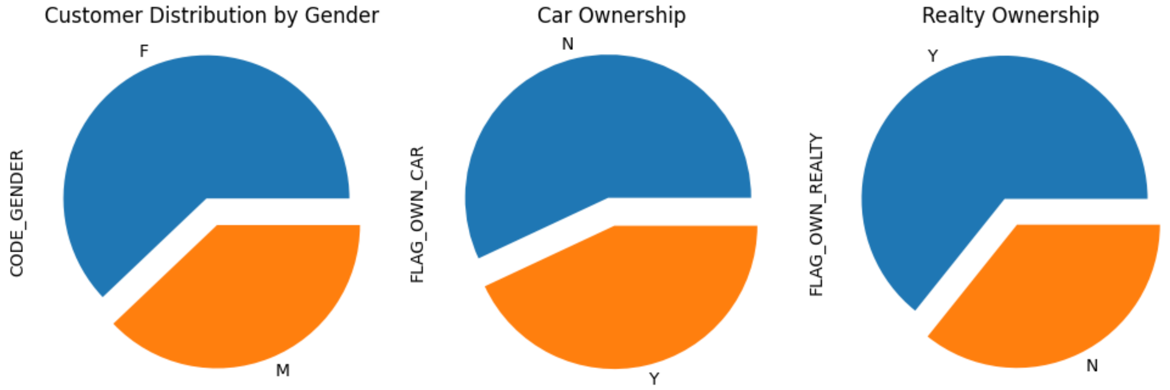
```
[24] # Other distribution plots
     data_cpy['CNT_CHILDREN'].value_counts(normalize=True).plot.bar(title='Children count')
     plt.show()
```




```
[25] data['CODE_GENDER'].value_counts(normalize=True).plot.bar(title='Gender')
     plt.show()
```



```
[26] fig, axes = plt.subplots(1,3)

     g1= data['CODE_GENDER'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[0])
     g1.set_title("Customer Distribution by Gender")

     g2= data['FLAG_OWN_CAR'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[1])
     g2.set_title("Car Ownership")

     g3= data['FLAG_OWN_REALTY'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[2])
     g3.set_title("Realty Ownership")

     fig.set_size_inches(10,5)

     plt.tight_layout()

     plt.show()
```



6: Division of the set of objects into two groups

```
[27] #OneHot Encoding
     aux1 = pd.get_dummies(data_cpy, columns = ["CNT_CHILDREN"])
     aux2 = pd.get_dummies(aux1, columns = ["NAME_EDUCATION_TYPE"])
     aux3 = pd.get_dummies(aux2, columns = ["NAME_FAMILY_STATUS"])
     aux4 = pd.get_dummies(aux3, columns = ["NAME_HOUSING_TYPE"])
     new_data = pd.get_dummies(aux4, columns = ["JOB"])
     new_data
```

In this part of the code, we transform categorical variables into numerical representations. The code utilizes the One-Hot Encoding technique, which creates a binary variable for each unique value in a categorical column.

```
[28] from sklearn.model_selection import train_test_split, cross_val_score,cross_val_predict
     X = new_data.drop("TARGET", axis = 1)
     y = new_data["TARGET"]
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
     #print(new_data.dtypes)
```

Here we perform train-test split on the dataset to evaluate the performance of the machine learning model. The code randomly splits the dataset into two parts: training and testing sets.

7: Artificial Neural Network (ANN) Model

<u>Theoretical basis of the model and an analysis performed on sample data:</u>

ANNs are powerful learning models that can effectively capture complex relationships in data. They have demonstrated remarkable performance in various classification tasks, particularly those with nonlinear relationships. In the code we have the implementation of an ANN model using the MLPClassifier class for training a binary classifier.

<u>Summary of the code's ANN-specific components:</u>

- Import MLPClassifier

- Train the ANN

- Make predictions

- Evaluate accuracy

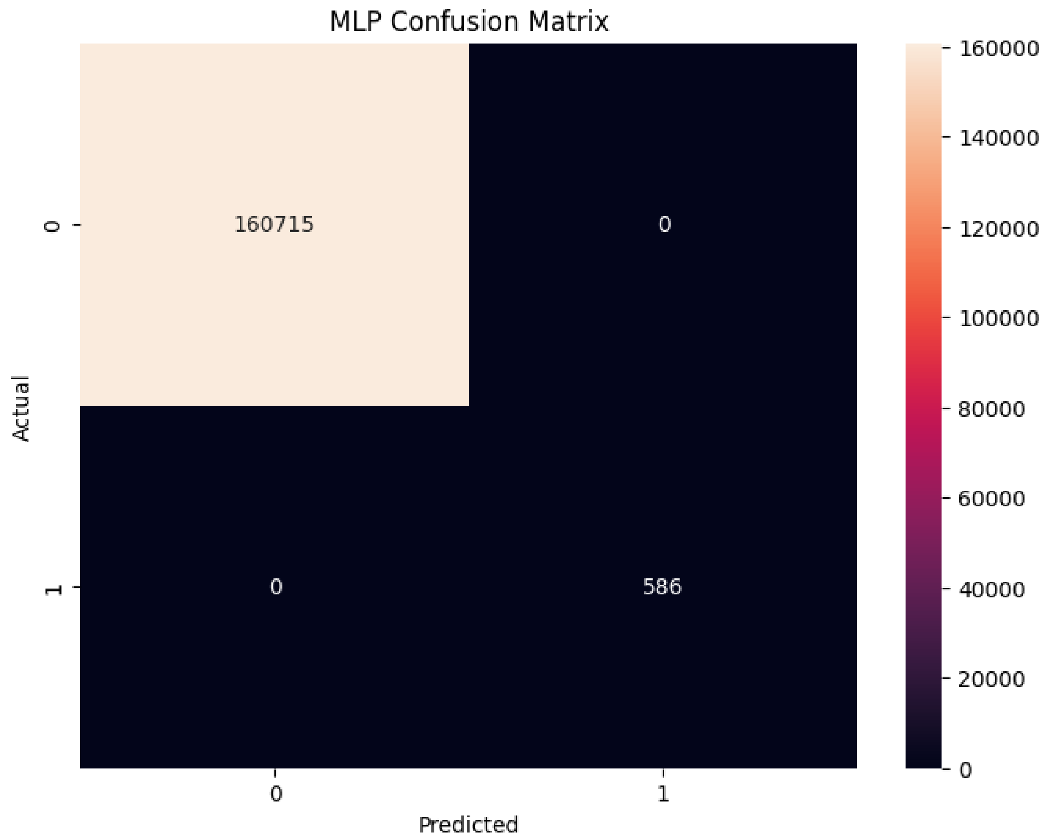
```
[29] # ANN
     from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()

     scaler.fit_transform(X_train)
     X_train_scaled = scaler.transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

```
[30] from sklearn.neural_network import MLPClassifier
     from sklearn.metrics import confusion_matrix
     import time
     from sklearn.metrics import accuracy_score

     start_time =time.time()

     training_start = time.perf_counter()
     mlpc = MLPClassifier().fit(X_train_scaled, y_train)
     training_end = time.perf_counter()

     prediction_start = time.perf_counter()
     preds_mlpc = mlpc.predict(X_test_scaled)
     prediction_end = time.perf_counter()

     acc_mlpc = 100*accuracy_score(y_test, preds_mlpc)

     mlpc_train_time = training_end-training_start
     mlpc_prediction_time = prediction_end-prediction_start

     print("ANN Accuracy: ", acc_mlpc)
     print("Time consumed for training: %s seconds" % (mlpc_train_time))
     print("Prediction Execution Time: %s seconds" % (mlpc_prediction_time))
```

```
     ANN Accuracy:  100.0
     Time consumed for training: 54.61916916300004 seconds
     Prediction Execution Time: 0.2100256880000302 seconds
```

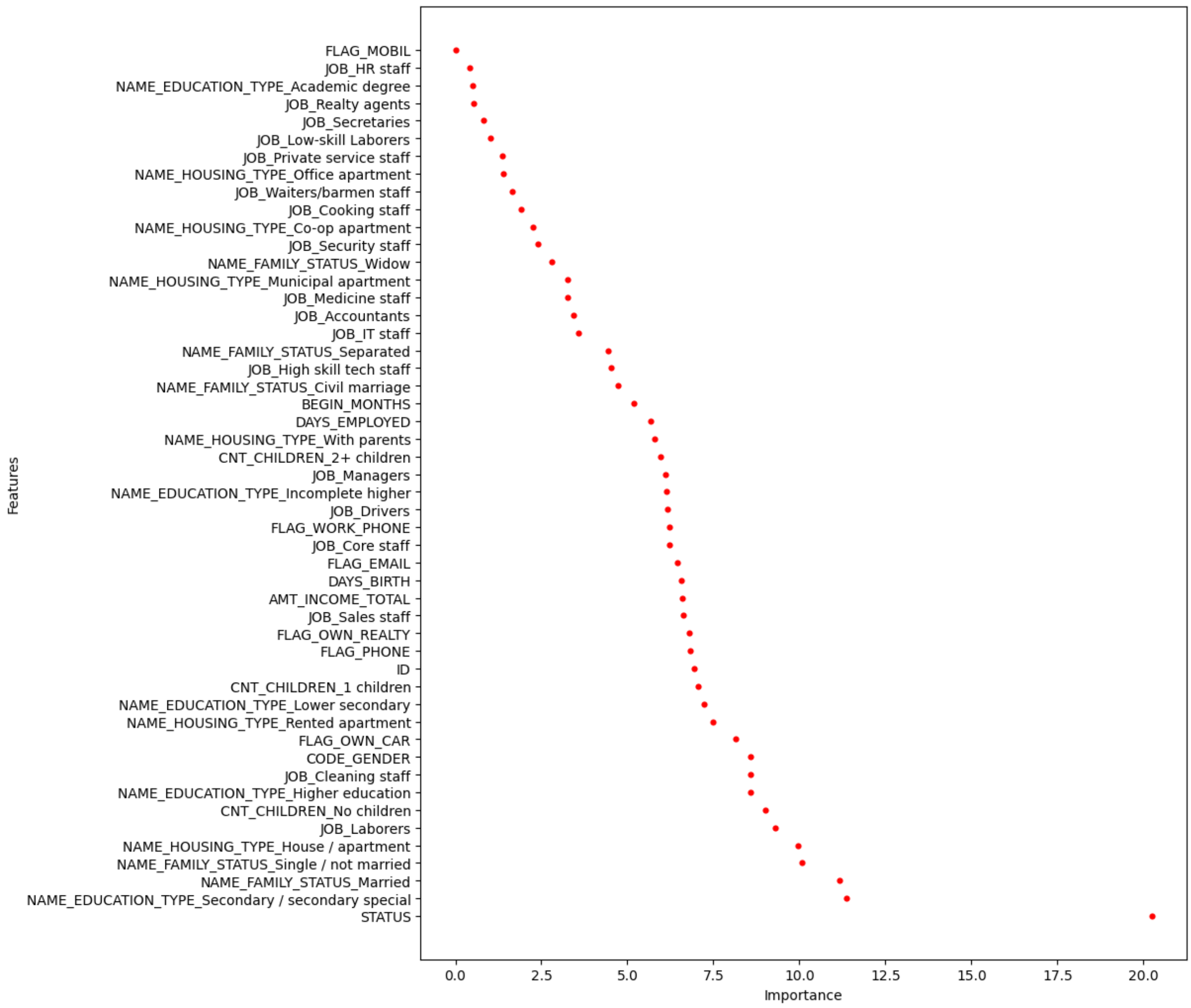
This code here demonstrates the implementation of a multilayer perceptron (MLP) classifier, a type of artificial neural network (ANN), for training a binary classifier. The output presents the accuracy, the time consumed for training and the predicted execution time.

```
[31] # Confusion matrix
     cm = confusion_matrix(y_test, preds_mlpc)
     plt.figure(figsize=(8, 6))
     sns.heatmap(cm, annot=True, fmt='d')
     plt.xlabel('Predicted')
     plt.ylabel('Actual')
     plt.title('MLP Confusion Matrix')
     plt.show()
```



The provided graph illustrates the performance of an artificial neural network (ANN) model in classifying data. It visualizes the model's ability to accurately predict the target variable based on the input features. The ANN's performance is evaluated using the accuracy metric, which represents the percentage of correctly classified instances.

```
[32] # Get the coefficients (weights) for the input layer
     weights = mlpc.coefs_[0]

     # Calculate the absolute sum of weights for each feature
     feature_importances = np.sum(np.abs(weights), axis=1)

     # Create a DataFrame to visualize feature importances
     imp = pd.DataFrame({'Name': X_train.columns, 'Score': feature_importances})
     imp.sort_values(by='Score', inplace=True)

     # Plot feature importances
     sns.scatterplot(x='Score', y='Name', linewidth=0, data=imp, s=20, color='red').set(
         xlabel='Importance',
         ylabel='Features'
     )

     plt.gcf().set_size_inches(10, 12.5)
     plt.show()
```



This part shows the feature importance for the artificial neural network (ANN) model trained to classify data. It visualizes how each input feature contributes to the model's predictions.