Note - Whole file can be runned at once, so request to run the whole all in once.

Capstone Project 1 - Credit Card Approval - Pranay B Shah - DS38B

The proposed approach enhances credit risk assessment by leveraging machine learning techniques, ensuring loans are granted to reliable individuals. It improves decision-making through in-depth analysis of applicant data, reducing the risk of financial losses for banks. This method streamlines processes, cutting down on manual tasks and operational costs. By automating tasks, it boosts efficiency and reduces processing time, benefiting both banks and customers. Additionally, it provides a competitive advantage by offering better terms to low-risk borrowers, attracting more clients. Overall, this approach revolutionizes credit assessment in banking, fostering reliability, efficiency, and competitiveness.

Data Analysis (DA) Track:

Hypothesis:

1. **Pattern Identification**: We hypothesize that certain demographic factors such as age, gender, education level, and income may exhibit patterns that correlate with credit card approval decisions.
2. **Feature Importance**: We believe that features like annual income, age, and employment duration will be crucial in determining creditworthiness. Additionally, factors like gender and marital status may also play a role, but There could be some more information which talks about credit_score, previous credit history, remaining_payments_till_date, and missed_installments/bills information could have helped much more to get better analysis and prediction.
3. **Data Quality Impact**: We found several missing values in certain columns, such as education and occupation, may impact the analysis and need to be addressed through imputation or removal.

Machine Learning (ML) Track:

Hypothesis:

1. **Model Performance**: We hypothesize that machine learning models trained on the dataset will outperform traditional rule-based approaches in predicting credit card approval decisions.
2. **Feature Importance**: Features such as annual income, age, and education level will be significant predictors of credit card approval. Other factors like gender and employment duration may also contribute to the model's predictive power.
3. **Model Justification**: We expect that the Random Forest model, due to its ability to capture complex relationships and handle non-linearities, will outperform other models like logistic

regression and decision trees in terms of accuracy and generalization.

Justification:

1. **Cost Functions**: We have evaluated the performance of the ML models using relevant cost functions such as accuracy, and confusion matrix, F1-score to assess their predictive capability and ability to minimize misclassifications.
2. **Cross-Validation**: To ensure the robustness of our models, we will employ techniques like cross-validation to assess their performance on unseen data and avoid overfitting.
3. **Feature Importance Analysis**: We will analyze the importance of features in predicting credit card approval using techniques like feature importance plots provided by tree-based models.

Section 3: Data analysis approach

- What approach are you going to take in order to prove or disprove your hypothesis?
- What feature engineering techniques will be relevant to your project?

Data Analysis Approach:

1. **EDA (Exploratory Data Analysis):**

- **Univariate Analysis:**
  - **Gender, Car Owner, Property Owner, and Credit Card Approval:**
    - Explored the distribution of gender, car ownership, property ownership, and credit card approval using pie charts.
    - Identified the percentage share of each category in these columns.
  - **Education and Children:**
    - Analyzed the distribution of education levels and the number of children using bar charts.
    - Provided insights into the education levels and the presence of children in credit card applications.
  - **Income Type and Housing Type:**
    - Examined the distribution of income types and housing types using bar charts.
    - Highlighted the proportions of different income and housing categories.
  - **Age and Employment Year Relationship:**
    - Investigated the relationship between age and employment year using a line plot.
    - Calculated statistics related to age and employment year groups.
- **Bivariate Analysis:**
  - **Gender, Age, and Car Ownership:**
    - Utilized boxplots to show the relationship between gender, age, and car ownership.
    - Identified age ranges where car ownership is prevalent for different genders.
  - **Gender, Annual Income, and Property Ownership:**
    - Visualized the distribution of annual income based on gender and property ownership using violin plots.
  - **Age vs. Annual Income Scatter Plot:**

        - Explored the scatter plot of age vs. annual income, colored by gender.
        - Investigated the general trend and patterns in the relationship between age and income.
    - **Children, Age, and Gender:**
        - Plotted a violin plot to analyze the relationship between the number of children, age, and gender.

2. **Feature Engineering Techniques:**

- **Renaming Columns:**
    - Renamed columns for better readability and usage in further analysis.
- **Conversion of Object Type Columns:**
    - Utilized label encoding and one-hot encoding for converting object-type columns to numerical format.
    - Applied ordinal encoding for certain columns like 'Type_Occupation', 'Car_Owner', 'Property_Owner', 'Housing_type', 'EDUCATION', and 'Type_Income'.
    - Used one-hot encoding for nominal columns like 'GENDER' and 'Marital_status'.
- **Imputation:**
    - Employed KNN imputation to fill null values in 'Age' and 'Annual_income' columns.
    - Imputed missing values in 'Type_Occupation' based on the type of income.
    - Filled missing values in 'GENDER' with the mode value.
- **Exploratory Data Visualization:**
    - Visualized the distribution of various categorical and numerical features.
    - Used pie charts, bar charts, line plots, boxplots, violin plots, and scatter plots to analyze relationships and patterns.
    - Calculated mean and median values for numeric columns and represented them in histograms.

Justification of Data Analysis Approach:

1. **Visualization Techniques:**

    - Utilized a variety of visualization techniques to explore both univariate and bivariate relationships in the data.
    - Visualization methods such as pie charts, bar charts, boxplots, violin plots, and scatter plots provide a comprehensive understanding of the data distribution and relationships.
2. **Feature Engineering:**

    - Renamed columns to enhance clarity and ease of use.
    - Employed appropriate encoding techniques for converting object-type columns to numerical format.
    - Handled null values using KNN imputation and strategic imputation based on relevant columns.
3. **Insight Generation:**

    - Extracted meaningful insights from the distribution of gender, car ownership, property ownership, education, income types, housing types, and their relationships.

- Investigated the impact of age on employment year and explored patterns related to gender, car ownership, and property ownership.

4. **Statistical Analysis:**
   - Calculated statistics such as mean, median, minimum, maximum, and counts for numeric columns, aiding in understanding central tendencies and variations.
5. **Relationship Exploration:**
   - Explored relationships between various features, providing a basis for understanding potential correlations and dependencies.
6. **Data Quality Checks:**
   - Checked and handled duplicates and missing values to ensure data quality.
   - Ensured that the dataset is suitable for further analysis and modeling.

Section 4: Machine learning approach

- What method will you use for machine learning based predictions for credit card approval?
- Please justify the most appropriate model.
- Please compare all models (at least 4 models).

1. Method for Machine Learning Predictions for Credit Card Approval:

The provided code implements the following machine learning models for predicting credit card approval:

- **RandomForest Classifier**
- **XGBoost**
- **Logistic Regression**
- **Support Vector Machines (SVM)**
- **Decision Tree Classifier**

2. Justification for RandomForest Classifier:

The RandomForest Classifier is chosen as the primary model for credit card approval predictions for the following reasons:

- **Ensemble Learning:** RandomForest is an ensemble learning method that builds multiple decision trees and merges them together. This helps reduce overfitting and improves the model's generalization.
- **Feature Importance:** The model considers feature importance, providing insights into which features contribute the most to the predictions. This is crucial for understanding the factors influencing credit card approval.
- **Handling Non-Linearity:** RandomForest can capture non-linear relationships between features and the target variable, which is valuable when dealing with complex patterns in

credit card approval.

3. Steps to Improve Model Accuracy:

Several steps are taken to enhance the accuracy of the RandomForest model:

- **Feature Selection:** The top 10 important features are selected to focus on the most relevant information for prediction.
- **Standardization:** MinMaxScaler is applied to standardize the selected features, ensuring all features contribute equally to the model.
- **Cross-Validation:** Cross-validation is performed to assess the model's performance on different subsets of the data, reducing the risk of overfitting.

4. Model Comparison:

Four different machine learning models are compared based on their accuracy scores:

- **RandomForest Classifier**
- **XGBoost**
- **Logistic Regression**
- **Support Vector Machines (SVM)**
- **Decision Tree Classifier**

5. Model Accuracy Comparison:

A bar graph is plotted to visualize the accuracy comparison of different models. The accuracy scores are as follows:

- **Decision Tree: 87.74%**
- **Support Vector Machine: 90.32%**
- **Logistic Regression: 90.32%**
- **XGBoost: 92.58%**
- **Random Forest: 95.16%**

Insights:

- The RandomForest model outperforms other models in terms of accuracy, making it the preferred choice for credit card approval predictions.
- Cross-validation helps evaluate the model's generalization performance and identify potential issues such as overfitting.

In [ ]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, con
```

```
from sklearn.impute import SimpleImputer
from sklearn import tree
```

A. Overview of Dataset

1. Loading dataset

In [ ]:
```
# Load the dataset
cd = pd.read_csv('Credit_card.csv')              #Dataset I
cl = pd.read_csv('Credit_card_label.csv')        #Dataset II
```

In [ ]:
```
# Information of Dataset I
cd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Birthday_count   1526 non-null   float64
 11  Employed_days    1548 non-null   int64  
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
dtypes: float64(2), int64(8), object(8)
memory usage: 217.8+ KB
```

In [ ]:
```
# Information of Dataset I
cl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Ind_ID  1548 non-null   int64
 1   label   1548 non-null   int64
dtypes: int64(2)
memory usage: 24.3 KB
```

2. Merging 2 diff table

The `Credit_card_merged` DataFrame represents a merged dataset combining information from two tables based on the common column 'Ind_ID'. This merged dataset contains details about credit card applicants (cd) and their corresponding credit limits (cl). Each row corresponds to a unique individual with their associated credit card and credit limit information.

In [ ]:
```
# Considering 'Ind_ID' being the common column between the two tables
Credit_card_merged = pd.merge(cd, cl, left_on='Ind_ID', right_on='Ind_ID')

print(Credit_card_merged.head())
```

```
    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0  5008827      M         Y             Y         0       180000.0
1  5009744      F         Y             N         0       315000.0
2  5009746      F         Y             N         0       315000.0
3  5009749      F         Y             N         0            NaN
4  5009752      F         Y             N         0       315000.0

             Type_Income          EDUCATION Marital_status       Housing_type  \
0              Pensioner  Higher education        Married  House / apartment
1  Commercial associate  Higher education        Married  House / apartment
2  Commercial associate  Higher education        Married  House / apartment
3  Commercial associate  Higher education        Married  House / apartment
4  Commercial associate  Higher education        Married  House / apartment

   Birthday_count  Employed_days  Mobile_phone  Work_Phone  Phone  EMAIL_ID  \
0        -18772.0         365243             1           0      0         0
1        -13557.0           -586             1           1      1         0
2             NaN           -586             1           1      1         0
3        -13557.0           -586             1           1      1         0
4        -13557.0           -586             1           1      1         0

  Type_Occupation  Family_Members  label
0             NaN               2      1
1             NaN               2      1
2             NaN               2      1
3             NaN               2      1
4             NaN               2      1
```

3. Saving merged dataset

In [ ]:
```
Credit_card_merged.to_csv("1.Credit_card_merged.csv", index = False)
```

4. Operations on Merged dataset

In [ ]:
```
df = pd.read_csv("1.Credit_card_merged.csv")
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Birthday_count   1526 non-null   float64
 11  Employed_days    1548 non-null   int64  
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(2), int64(9), object(8)
memory usage: 229.9+ KB
```

In [ ]:
```
df.head(3)
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education

In [ ]:
```
# Checking for null values into the dataset.
df.isnull().sum()
```

Out[ ]:
```
Ind_ID               0
GENDER               7
Car_Owner            0
Propert_Owner        0
CHILDREN             0
Annual_income       23
Type_Income          0
EDUCATION            0
Marital_status       0
Housing_type         0
Birthday_count      22
Employed_days        0
Mobile_phone         0
Work_Phone           0
Phone                0
EMAIL_ID             0
Type_Occupation    488
Family_Members       0
label                0
dtype: int64
```

In [ ]:
```
df.head()
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education
3	5009749	F	Y	N	0	NaN	Commercial associate	Higher education
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education

4.1. Converting "Employees_days" to "Employees_year"

The column named Employees_days is the column with negative values and is something which can't be used directly hence we change that to year.

Counting 'Employed days' value counts below 0 or in -ve, since we have major values in Employed Days in -ve which dates backward, and explains about employed days till date.

In [ ]:
```
df[df['Employed_days'] < 0].value_counts()
```

Out[ ]:
```
Ind_ID   GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income  Type_Income
EDUCATION                     Marital_status        Housing_type       Birthday_count
Employed_days  Mobile_phone  Work_Phone  Phone  EMAIL_ID  Type_Occupation  Family_Member
s  label
5008865  F       Y          Y              2         135000.0       Working
Secondary / secondary special  Married              House / apartment  -15761.0
-3173          1             0           0      0         Laborers         4
0        1
5088687  M       N          Y              1         135000.0       Working
Secondary / secondary special  Married              House / apartment  -18719.0
-1473          1             1           1      0         Laborers         3
0        1
5105291  F       N          N              0         112500.0       Working
Secondary / secondary special  Widow                With parents       -10240.0
-818           1             1           1      0         Core staff       1
0        1
5105395  F       N          Y              0         135000.0       Working
Incomplete higher              Single / not married  House / apartment  -10932.0
-575           1             0           0      0         Secretaries      1
0        1
5105550  F       N          Y              0         112500.0       Working
Higher education               Married              With parents       -9310.0
-825           1             0           0      0         Core staff       2
0        1

..
5054348  F       N          N              0         103500.0       Working
Secondary / secondary special  Single / not married  House / apartment  -12348.0
-424           1             1           1      0         Sales staff      1
0        1
5054384  M       Y          N              0         292500.0       Commercial associate
Higher education               Single / not married  House / apartment  -14050.0
-3680          1             1           1      0         Laborers         1
0        1
5054407  M       N          N              0         225000.0       Commercial associate
Higher education               Married              With parents       -13021.0
-5082          1             1           0      0         Managers         2
0        1
5054414  M       N          N              0         225000.0       Commercial associate
Higher education               Married              With parents       -13021.0
-5082          1             1           0      0         Managers         2
0        1
5150221  M       Y          Y              0         116100.0       Working
Secondary / secondary special  Married              House / apartment  -10136.0
-2441          1             0           0      0         Laborers         2
0        1
Length: 1025, dtype: int64
```

Converting -ve values to +ve ones

In [ ]:
```
# Finding negative values in 'Employed_days' column
negative_values_mask = df['Employed_days'] < 0

# Converting negative values to positive
df.loc[negative_values_mask, 'Employed_days'] *= -1
```

In [ ]:
```
# Confirming the above code.
df[df['Employed_days'] < 0].value_counts()
```

Out[ ]:
```
Series([], dtype: int64)
```

In [ ]:
```
# Conversion to +ve value Confirmed
df['Employed_days'].sample(3)
```

Out[ ]:
```
1333    3574
964      795
264     3309
Name: Employed_days, dtype: int64
```

Converting days to year

In [ ]:
```
# Calculating employment duration from Employed_days and rounding it.
df['Employment_Year'] = round(df['Employed_days'] / 365, 0)

# Removing the original Employed_days column.
df = df.drop('Employed_days', axis=1)

# Inserting 'Employment_Duration' column at the 12th position
df.insert(11, 'Employment_Year', df.pop('Employment_Year'))
```

In [ ]:
```
print(df['Employment_Year'].sample(3))
```

```
1400     4.0
1049     4.0
1188    15.0
Name: Employment_Year, dtype: float64
```

4.2. Converting "Birthday_count" to "Age"

Converting -ve values to +ve ones

In [ ]:
```
df.head(3)
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education

In [ ]:
```
# Finding negative values in 'Employed_days' column
negative_values_mask = df['Birthday_count'] < 0

# Converting negative values to positive
df.loc[negative_values_mask, 'Birthday_count'] *= -1
```

In [ ]:
```
# Confirming the above code.
df[df['Birthday_count'] < 0].value_counts()
```

Out[ ]:
```
Series([], dtype: int64)
```

In [ ]:
```
df['Birthday_count'].sample(3)
```

Out[ ]:
```
1332    13483.0
562     16768.0
1152    21117.0
Name: Birthday_count, dtype: float64
```

Converting birth days to Age

In [ ]:
```
# Calculating age from Birthday_count and round
df['Age'] = round(df['Birthday_count']/365, 0)
```

In [ ]:
```
# Removing the original Birthday_count column
df = df.drop('Birthday_count', axis=1)
```

In [ ]:
```
# Inserting 'Age' column at the 11th position
df.insert(10, 'Age', df.pop('Age'))
```

In [ ]:
```
df['Age'].sample(3)
```

Out[ ]:
```
911    31.0
338    34.0
430    60.0
Name: Age, dtype: float64
```

Saving new dataset

In [ ]:
```
df.to_csv("2.Credit_card_changed_Bday&empyear.csv", index = False)
```

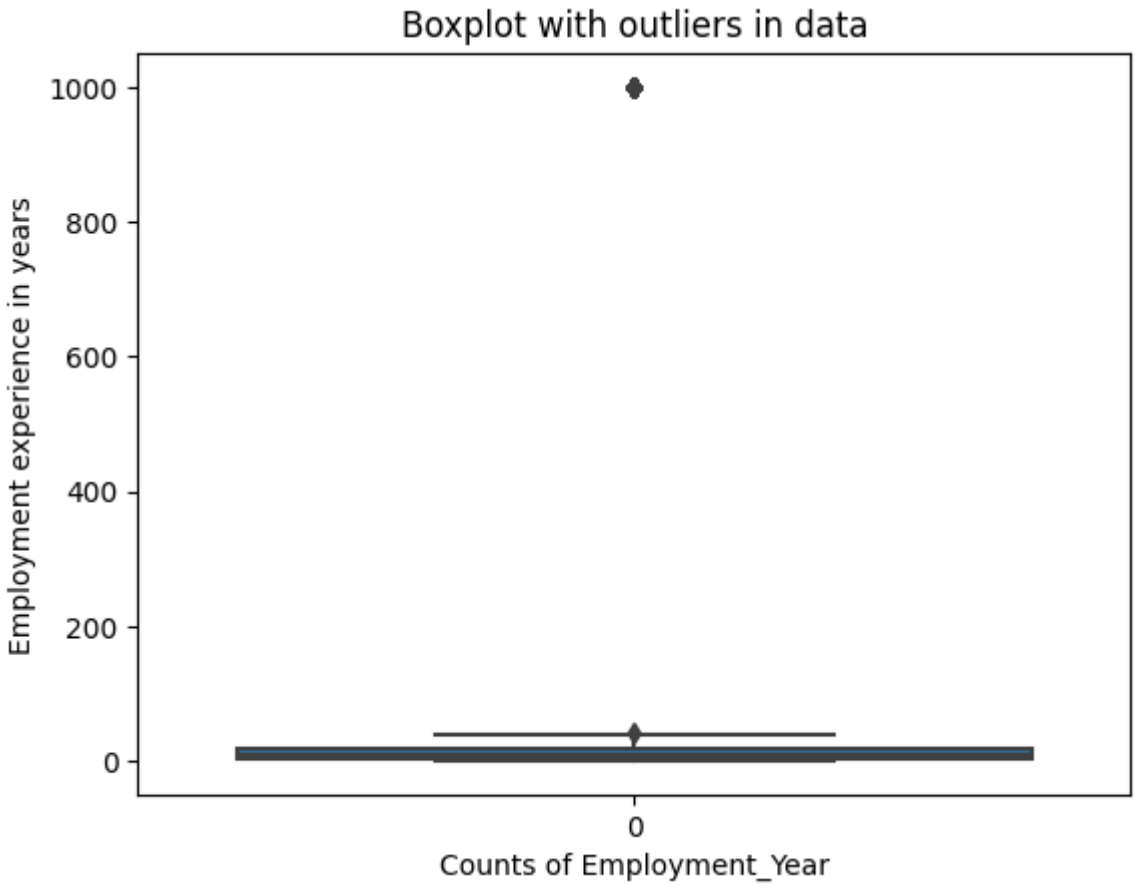
1 - New Dataset - Savepoint 1

B. Treating Outliers in "Employement_in_years" column

1. Outliers Overview

In [ ]:
```
df1 = pd.read_csv("2.Credit_card_changed_Bday&empyear.csv")     # Reading dataset
```

In [ ]:
```
sns.boxplot(df['Employment_Year'])          # Using seaborn library for plotting boxplot
plt.xlabel("Counts of Employment_Year")     # applying x label
plt.ylabel("Employment experience in years")    # applying y label
plt.title("Boxplot with outliers in data")  # applying title
plt.show()
```



Insights:

The above boxplot explains that there outliers at 1000, and major lies below 60

In [ ]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Age              1526 non-null   float64
 11  Employment_Year  1548 non-null   float64
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(3), int64(8), object(8)
memory usage: 229.9+ KB
```

In [ ]:
```
df1.sample(4)
```

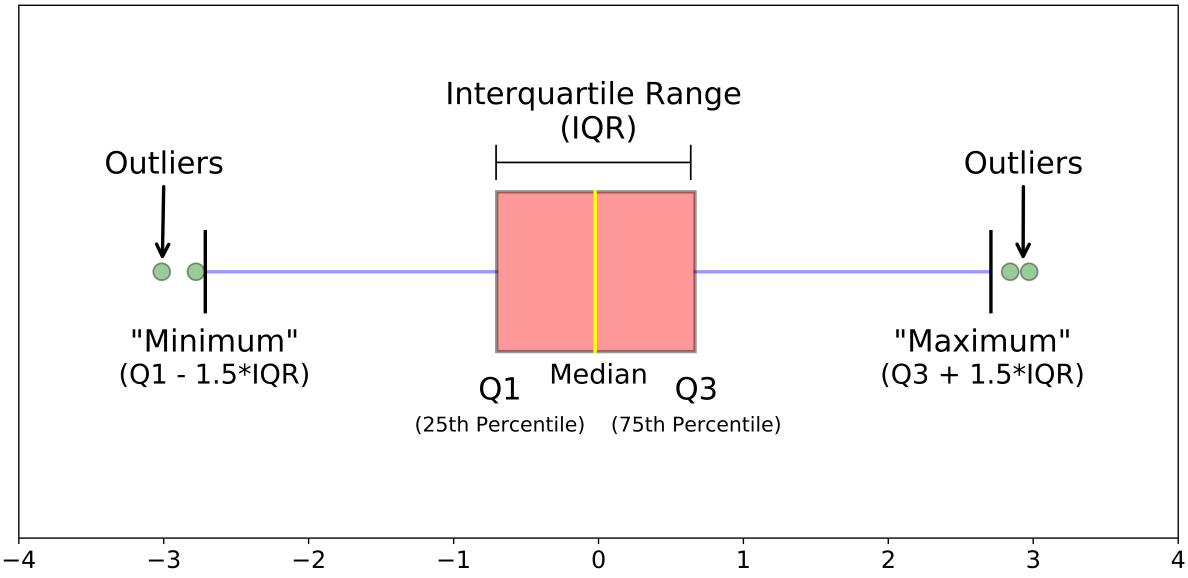
Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATI
339	5067982	M	Y	Y	0	153000.0	Working	Hig educa
1442	5061161	M	Y	Y	3	202500.0	Working	Seconda secon spe
1400	5089497	F	N	N	1	112500.0	Working	Seconda secon spe
1246	5054228	F	N	N	2	180000.0	Working	Seconda secon spe

In [ ]:
```
df1.isnull().sum()      # Counting null values
```

Out[ ]:
```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income      23
Type_Income         0
EDUCATION           0
Marital_status      0
Housing_type        0
Age                22
Employment_Year     0
Mobile_phone        0
Work_Phone          0
Phone               0
EMAIL_ID            0
Type_Occupation   488
Family_Members      0
label               0
dtype: int64
```

1.1 Inter Quartile Range - Detecting & treating Outliers



In [ ]:
```
df1.describe()["Employment_Year"]
```

Out[ ]:
```
count    1548.000000
mean      174.844315
std       372.211315
min         0.000000
25%         3.000000
50%         7.000000
75%        17.000000
max      1001.000000
Name: Employment_Year, dtype: float64
```

`df1.describe()["Employment_Year"]` generates summary statistics for the 'Employment_Year' column in the DataFrame df1. The statistics include count, mean, standard deviation, minimum, 25th percentile (Q1), 50th percentile (median), 75th percentile (Q3), and maximum values.

The 25th percentile (Q1) represents the value below which 25% of the data falls. It indicates the first quartile of the data distribution.

The 75th percentile (Q3) represents the value below which 75% of the data falls. It indicates the third quartile of the data distribution.

These percentiles, along with the median (50th percentile), provide insights into the spread and distribution of the 'Employment_Year' data.

In [ ]:
```
Q1 = df1.describe()["Employment_Year"]['25%']
Q3 = df1.describe()["Employment_Year"]['75%']

print("Q1 = ", Q1)
print("Q3 = ", Q3)
```

```
Q1 =  3.0
Q3 =  17.0
```

In [ ]:
```
IQR = Q3-Q1

print("IQR = ", IQR)
```

```
IQR =  14.0
```

In [ ]:
```
LL = Q1 - 1.5 * IQR

UL = Q3 + 1.5 * IQR

print("Lower Limit = ",LL)
print("Upper Limit = ", UL)
```

```
Lower Limit =  -18.0
Upper Limit =  38.0
```

1.2 Clipping Upper_Limit vallues to Outliers in dataset

Using function named "clip()" we here cap upper outliers with upper limit which we've obtained from using statistical method IQR, and since our lower limit values are major near 1-3 , hence we don't cap lower values with Lower_Limit from IQR.

In [ ]:
```
df1['Employment_Year'] = df1['Employment_Year'].clip(lower=None, upper = UL)
```

In [ ]:
```
df1['Employment_Year'].value_counts(bins = 10)
```

Out[ ]:
```
(-0.039, 3.8]    424
(3.8, 7.6]       403
(34.2, 38.0]     267
(7.6, 11.4]      219
(11.4, 15.2]     103
(15.2, 19.0]      53
(19.0, 22.8]      37
(22.8, 26.6]      21
(26.6, 30.4]      11
(30.4, 34.2]      10
Name: Employment_Year, dtype: int64
```

In [ ]:
```
# Verifying that above mentioned (-0.039) values is not present in dataset, and there ar
df1[df1['Employment_Year'] == -0.039]
```

Out[ ]:

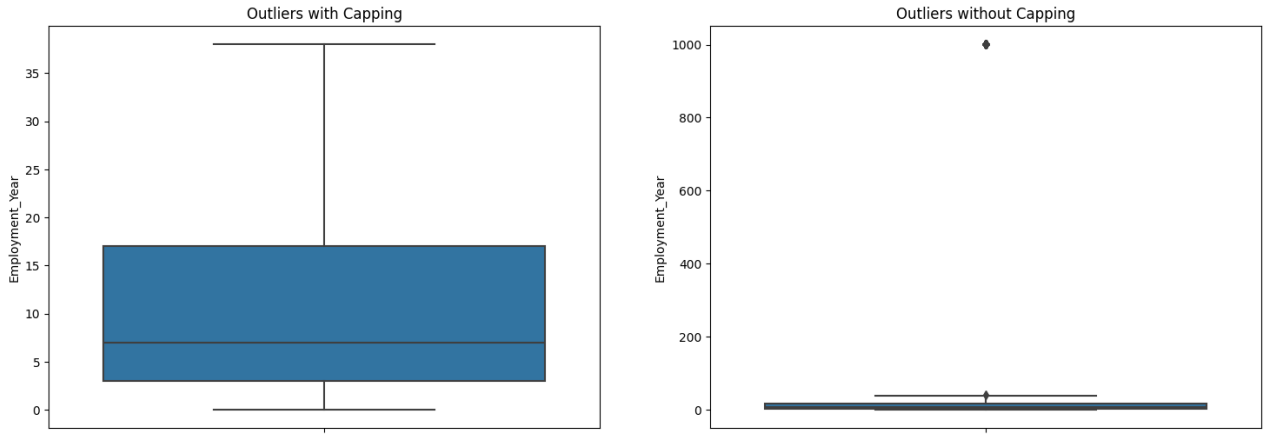
Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	N

In [ ]:
```
df1['Employment_Year'].unique()
```

Out[ ]:
```
array([38.,  2.,  3.,  1.,  7.,  5., 12.,  9., 13., 10.,  4., 20.,  6.,
       11.,  0., 23.,  8., 14., 19., 22., 21., 17., 26., 24., 15., 16.,
       34., 37., 27., 35., 18., 31., 32., 25., 29., 33., 28., 36.])
```

In [ ]:
```
plt.figure(figsize=(18,6))

# Plot for MICE vs Without MICE
plt.subplot(1,2,1)
sns.boxplot(data = df1, y = 'Employment_Year')
plt.title("Outliers with Capping")

plt.subplot(1,2,2)
sns.boxplot(data = df, y = 'Employment_Year')
plt.title("Outliers without Capping")
```

Out[ ]:
```
Text(0.5, 1.0, 'Outliers without Capping')
```



From above code we can clearly see that maximum upper_limit for the dataset is 1001 , which is technically incorrect, and its not possible to have an experience of 1001 years, since maximum life of an human as per current data is around 89, since those are the real outliers (Type_occupation is also missing for same individuals) which can't be predicted with help of KNN and MICE, hence we cap those values with UPPER_LIMIT of 38.

In [ ]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Age              1526 non-null   float64
 11  Employment_Year  1548 non-null   float64
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(3), int64(8), object(8)
memory usage: 229.9+ KB
```

From above analysis we conclude that the individuals with "Employement_in_years" having "1001" of exp could be an error taken place during entry of data, further it is observed that major of the these individual are above 52 years of age and also having Type_occupation missing, which further makes it easy to drop them out of dataset, as missing "Type_occupation" can't be imputed with using "mode" into that column as Type_occupation could be very different for individual and can't be guessed easily, further to use 'MICE Imputer' we need an similar range of data for predicting the values for those individual, but since all of those values for those individuals are "1001", hence there's no room for MICE Imputation as well, hence we capped UPPERLIMIT to all those (1001)

values and will be creating another category in "Type_occupation" for "pensioners" as "others" or "pensioners"

In [ ]:
```
df1.to_csv("3.Credit_card_Capped(1001).csv", index = False)
```

2 - New Dataset - Savepoint 2

C. Data Cleaning & Imputation

1. KNN-Imputation for null values in ['Age', 'Annual_Income'] columns, general imputation for "Type_occupation", and mode imputation for "gender"

In [ ]:
```
df2 = pd.read_csv("3.Credit_card_Capped(1001).csv")
```

In [ ]:
```
df2.head()
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education
3	5009749	F	Y	N	0	NaN	Commercial associate	Higher education
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education

In [ ]:
```
df2.isnull().sum()
```

Out[ ]:
```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income      23
Type_Income         0
EDUCATION           0
Marital_status      0
Housing_type        0
Age                22
Employment_Year     0
Mobile_phone        0
Work_Phone          0
Phone               0
EMAIL_ID            0
Type_Occupation   488
Family_Members      0
label               0
dtype: int64
```

1.1 KNN Imputation for "Age" and "Annual_income" columns

In [ ]:
```
# Importing KNNImputer from sklearn library
from sklearn.impute import KNNImputer

columns_to_impute = ['Annual_income','Age']

# Creating a KNN imputer
knn_imputer = KNNImputer(n_neighbors=3)

# Applying KNN imputation to the columns
df2[columns_to_impute] = knn_imputer.fit_transform(df2[columns_to_impute])
```

This code utilizes the `KNNImputer` class from the `sklearn.impute` module to impute missing values in specified columns ('Annual_income' and 'Age') using the K-Nearest Neighbors algorithm with `n_neighbors` set to 3. It fits the imputer to the data (`df2`) and transforms the specified columns, replacing missing values with values inferred from neighboring data points.

In [ ]:
```
# Verifying the above code
df2.isna().sum()
```

Out[ ]:
```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income       0
Type_Income         0
EDUCATION           0
Marital_status      0
Housing_type        0
Age                 0
Employment_Year     0
Mobile_phone        0
Work_Phone          0
Phone               0
EMAIL_ID            0
Type_Occupation   488
Family_Members      0
label               0
dtype: int64
```

In [ ]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1548 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Age              1548 non-null   float64
 11  Employment_Year  1548 non-null   float64
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(3), int64(8), object(8)
memory usage: 229.9+ KB
```

In [ ]:
```
df2['Age'] = df2['Age'].astype('int64') #Converting 'Age' column to "int" type
```

In [ ]:
```
df2['Annual_income'] = df2['Annual_income'].astype('int64')      #Converting 'Annual_inc
```

In [ ]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1548 non-null   int64  
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Age              1548 non-null   int64  
 11  Employment_Year  1548 non-null   float64
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(1), int64(10), object(8)
memory usage: 229.9+ KB
```

1.1.1 Filling null values in Type_occupation column with *"Unknown" *for non-pensioners and* "pensioner"* for pensioners

In this case we could have deleted thode columns but while going through data set I found that Type_occupations column has null values majorly for Pendioners from Type_income and some random ones, so instead of removing those I filled those "pensioners" with "pensioners" and others with as "unknown" category.

In [ ]:
```
df2['Type_Occupation'].isnull().value_counts()
```

Out[ ]:
```
False    1060
True      488
Name: Type_Occupation, dtype: int64
```

In [ ]:
```
# from sklearn.preprocessing import OrdinalEncoder
# rank = ['Core staff', 'Cooking staff', 'Laborers', 'Sales staff',
#         'Accountants', 'High skill tech staff', 'Managers',
#         'Cleaning staff', 'Drivers', 'Low-skill Laborers', 'IT staff',
#         'Waiters/barmen staff', 'Security staff', 'Medicine staff',
#         'Private service staff', 'HR staff', 'Secretaries',
#         'Realty agents']
# oe = OrdinalEncoder(categories=[rank])
# df3['Type_Occupation'] = oe.fit_transform(df3[['Type_Occupation']])
# df3.head()
```

Since there are nan values present as per above columns we'll impute those values by 'unknown' and then encode them to go for MICE prediction

1.1.1.1 Filling Type_occupation column with *"Unknown" *for non-pensioners and* "pensioner"* for pensioners

In [ ]:
```
# Fill missing values in 'Type_Occupation' column where 'Type_Income' is not 'Pensioner'
df2.loc[df2['Type_Income'] != 'Pensioner', 'Type_Occupation'] = df2.loc[df2['Type_Income
```

In [ ]:
```
# Fill missing values in 'Type_Occupation' column where 'Type_Income' is 'Pensioner'
df2.loc[df2['Type_Income'] == 'Pensioner', 'Type_Occupation'] = df2.loc[df2['Type_Income
```

In [ ]:
```
df2['Type_Occupation'].isnull().value_counts()
```

Out[ ]:
```
False    1548
Name: Type_Occupation, dtype: int64
```

In [ ]:
```
df2['Type_Occupation'].unique()
```

Out[ ]:
```
array(['Pensioner', 'Unknown', 'Core staff', 'Cooking staff', 'Laborers',
       'Sales staff', 'Accountants', 'High skill tech staff', 'Managers',
       'Cleaning staff', 'Drivers', 'Low-skill Laborers', 'IT staff',
       'Waiters/barmen staff', 'Security staff', 'Medicine staff',
       'Private service staff', 'HR staff', 'Secretaries',
       'Realty agents'], dtype=object)
```

In [ ]:
```
df2.isna().sum()
```

Out[ ]:
```
Ind_ID             0
GENDER             7
Car_Owner          0
Propert_Owner      0
CHILDREN           0
Annual_income      0
Type_Income        0
EDUCATION          0
Marital_status     0
Housing_type       0
Age                0
Employment_Year    0
Mobile_phone       0
Work_Phone         0
Phone              0
EMAIL_ID           0
Type_Occupation    0
Family_Members     0
label              0
dtype: int64
```

1.1.2 Filling GENDER column with *"Mode"* values

In [ ]:
```
column_to_impute = 'GENDER'

# Finding the most frequent gender in the dataset
most_frequent_gender = df2[column_to_impute].mode()[0]

# Imputing missing values with the most frequent gender
df2[column_to_impute].fillna(most_frequent_gender, inplace=True)

# Checking if there are any remaining missing values
print(df2.isnull().sum())
```

```
Ind_ID             0
GENDER             0
Car_Owner          0
Propert_Owner      0
CHILDREN           0
Annual_income      0
Type_Income        0
EDUCATION          0
Marital_status     0
Housing_type       0
Age                0
Employment_Year    0
Mobile_phone       0
Work_Phone         0
Phone              0
EMAIL_ID           0
Type_Occupation    0
Family_Members     0
label              0
dtype: int64
```

In [ ]:
```
# Before imputation of mean Gender in previous dataset
df1['GENDER'].value_counts()
```

Out[ ]:
```
F    973
M    568
Name: GENDER, dtype: int64
```

Form above code we can understand that "Females" are the highest number of applicants, hence we fill 7 missing values with most_frequent occuring ones, i.e "female".

In [ ]:
```
# After imputation of mean Gender in df2 dataset
df2['GENDER'].value_counts()
```

Out[ ]:
```
F    980
M    568
Name: GENDER, dtype: int64
```

In [ ]:
```
df2.head()
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION
0	5008827	M	Y	Y	0	180000	Pensioner	Higher education
1	5009744	F	Y	N	0	315000	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000	Commercial associate	Higher education
3	5009749	F	Y	N	0	315000	Commercial associate	Higher education
4	5009752	F	Y	N	0	315000	Commercial associate	Higher education

In [ ]:
```
df2.to_csv("4.Credit_card_clean_Data_Analysis.csv", index = False)
```

The above dataset now could be used for EDA and further data analysis

D. EDA

In [ ]:
```
import pandas as pd
dv = pd.read_csv("4.Credit_card_clean_Data_Analysis.csv")
```

In [ ]:
```
dv.head()
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION
0	5008827	M	Y	Y	0	180000	Pensioner	Higher education
1	5009744	F	Y	N	0	315000	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000	Commercial associate	Higher education
3	5009749	F	Y	N	0	315000	Commercial associate	Higher education
4	5009752	F	Y	N	0	315000	Commercial associate	Higher education

In [ ]:
```
# Renaming the Columns for easy usage.

# (inplace = True) helps renaming the data permanently in main file.
dv.rename(columns= {"GENDER": "Gender"}, inplace = True)

dv.rename(columns= {"Propert_Owner": "Property_Owner"}, inplace = True)

dv.rename(columns= {"CHILDREN": "Children"}, inplace = True)

dv.rename(columns= {"EDUCATION": "Education"}, inplace = True)

dv.rename(columns= {"EMAIL_ID": "Email_ID"}, inplace = True)
```

In [ ]:
```
dv.to_csv("SQL_Credit_card_clean_Data_Analysis.csv", index = False)
```

In [ ]:
```
dv.head(4)
```

Out[ ]:

	Ind_ID	Gender	Car_Owner	Property_Owner	Children	Annual_income	Type_Income	Education	M
0	5008827	M	Y	Y	0	180000	Pensioner	Higher education	
1	5009744	F	Y	N	0	315000	Commercial associate	Higher education	
2	5009746	F	Y	N	0	315000	Commercial associate	Higher education	
3	5009749	F	Y	N	0	315000	Commercial associate	Higher education	

In [ ]:
```
dv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   Gender           1548 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Property_Owner   1548 non-null   object 
 4   Children         1548 non-null   int64  
 5   Annual_income    1548 non-null   int64  
 6   Type_Income      1548 non-null   object 
 7   Education        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Age              1548 non-null   int64  
 11  Employment_Year  1548 non-null   float64
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  Email_ID         1548 non-null   int64  
 16  Type_Occupation  1548 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(1), int64(10), object(8)
memory usage: 229.9+ KB
```

In [ ]:
```
dv.shape
```

Out[ ]: (1548, 19)

In [ ]:
```
# Checking for any duplicate rows present in data.

dv.duplicated().sum()
```

Out[ ]: 0

In [ ]:
```
dv.head()
```

Out[ ]:

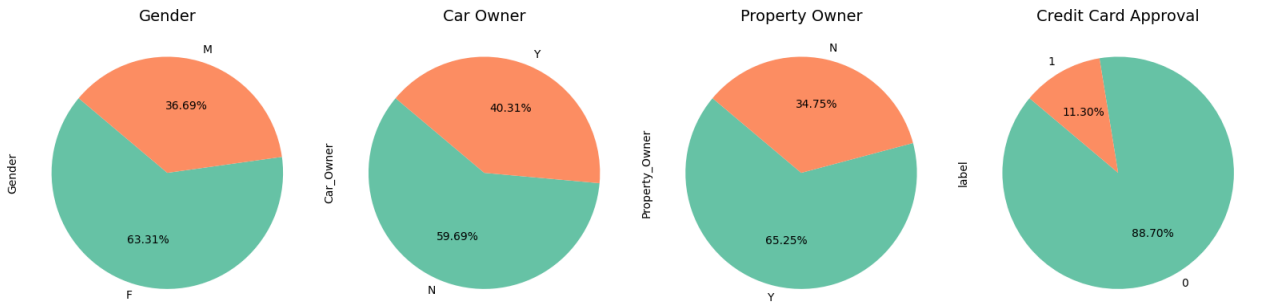
	Ind_ID	Gender	Car_Owner	Property_Owner	Children	Annual_income	Type_Income	Education	M
0	5008827	M	Y	Y	0	180000	Pensioner	Higher education	
1	5009744	F	Y	N	0	315000	Commercial associate	Higher education	
2	5009746	F	Y	N	0	315000	Commercial associate	Higher education	
3	5009749	F	Y	N	0	315000	Commercial associate	Higher education	
4	5009752	F	Y	N	0	315000	Commercial associate	Higher education	

Univariate Analysis

What is the percent share of Gender, Car Owner, Property Owner and How much applications are approved till date?

In [ ]:
```
# Applying value_counts() separately to each column
value_counts_gender = dv['Gender'].value_counts()
value_counts_car_owner = dv['Car_Owner'].value_counts()
value_counts_property_owner = dv['Property_Owner'].value_counts()
value_counts_label = dv['label'].value_counts()

# Plotting pie chart for each column
plt.figure(figsize=(16, 10))

# Color palette for pie charts
colors = ['#66c2a5', '#fc8d62']

plt.subplot(1, 4, 1)
value_counts_gender.plot(kind='pie', autopct='%.2f%%', colors=colors, startangle=140)
plt.title('Gender', fontsize=14)

plt.subplot(1, 4, 2)
value_counts_car_owner.plot(kind='pie', autopct='%.2f%%', colors=colors, startangle=140)
plt.title('Car Owner', fontsize=14)

plt.subplot(1, 4, 3)
value_counts_property_owner.plot(kind='pie', autopct='%.2f%%', colors=colors, startangle
plt.title('Property Owner', fontsize=14)

plt.subplot(1, 4, 4)
value_counts_label.plot(kind='pie', autopct='%.2f%%', colors=colors, startangle=140)
plt.title('Credit Card Approval', fontsize=14)
```

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



Insights :

Gender:

- The above chart explains that there are total 63% females and 36% males applting for Credit card

Car Owner:

- The above chart explains that there are total 59% of applicants who does not owns the car and 40% applicants owns the car.

Property_owner:

- The above chart explains that there are total 65% of applicants who owns the property and 34% of applicants does not owns the property.

Credit Card Approvals:

- The above chart explains that there are total 88% of applications who are passed and 11% of applicants got rejected for application of credit card.

In [ ]:
```
# Below code explains median and maximum salary as per educational background in our sys
dv.groupby(['Gender', "Car_Owner", "Property_Owner"])['Ind_ID'].count()
```

Out[ ]:
```
Gender  Car_Owner  Property_Owner
F       N          N                245
                   Y                474
        Y          N                 82
                   Y                179
M       N          N                 77
                   Y                128
        Y          N                134
                   Y                229
Name: Ind_ID, dtype: int64
```

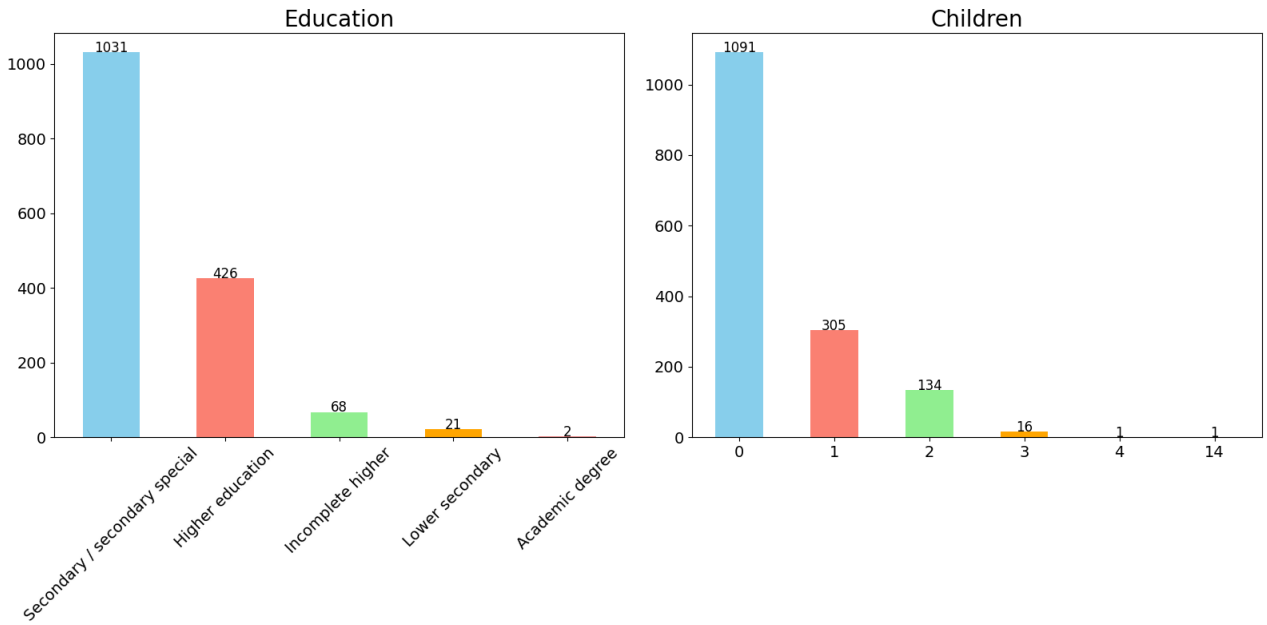
Insights:

The code groups the dataset by gender, car ownership status, and property ownership status. It then calculates the count of individuals (Ind_ID) in each group. This analysis provides insights into

the distribution of individuals based on gender, car ownership, and property ownership.

Show Unique values and their counts in Education and Children Column

In [ ]:
```
# Applying value_counts() separately to each column
value_counts_education = dv['Education'].value_counts()
value_counts_children = dv['Children'].value_counts()

# Plotting pie chart for each column
plt.figure(figsize=(16, 8))

# Color palette for pie charts
custom_colors = ['skyblue', 'salmon', 'lightgreen', 'orange', 'lightcoral']

plt.subplot(1, 2, 1)
ax = value_counts_education.plot(kind='bar', color=custom_colors)
plt.title('Education', fontsize=20)
plt.xticks(rotation=45, fontsize=14)  # Rotate x-axis labels and set font size for bette
plt.yticks(fontsize=14)

# Add count above each bar
for i in ax.patches:
    plt.text(i.get_x() + i.get_width() / 2, i.get_height() + 0.1, str(i.get_height()), h

plt.subplot(1, 2, 2)
ax = value_counts_children.plot(kind='bar', color=custom_colors)
plt.title('Children', fontsize=20)
plt.xticks(rotation=0, fontsize=14)  # Rotate x-axis labels and set font size for better
plt.yticks(fontsize=14)

# Add count above each bar
for i in ax.patches:
    plt.text(i.get_x() + i.get_width() / 2, i.get_height() + 0.1, str(i.get_height()), h

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



Insights :

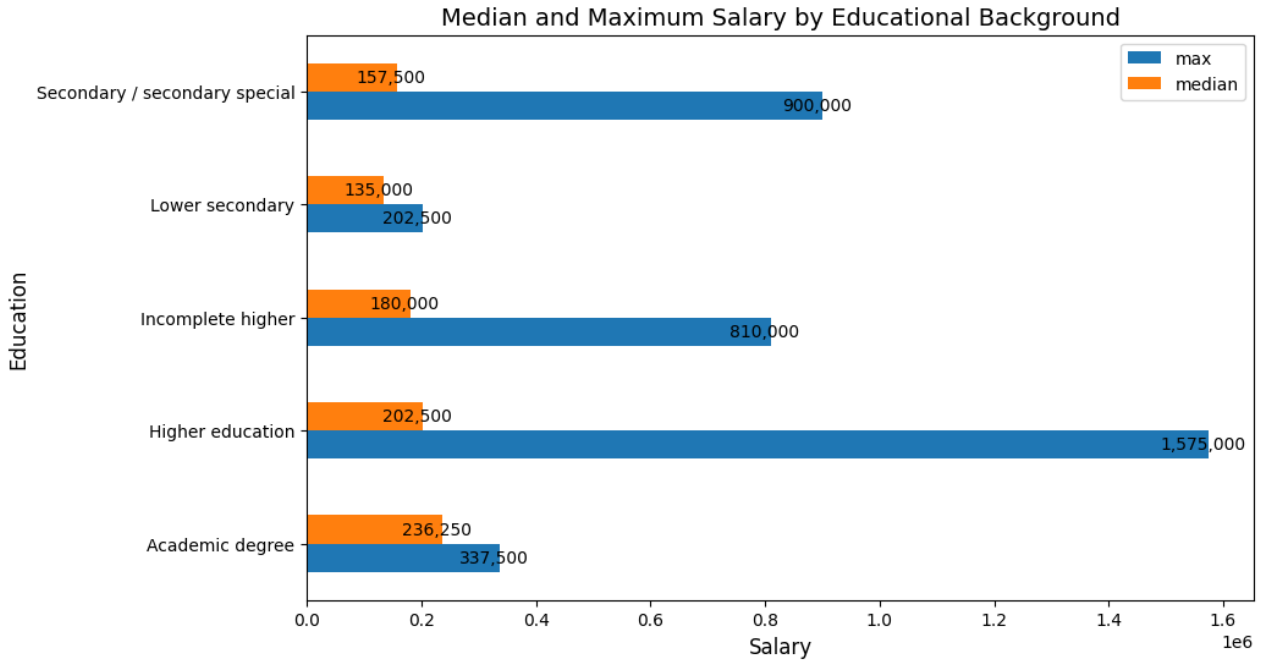
Education:

- The application for credit cards are majorly educated from Secondary academics with 1031 applicants, followed by Higher education with 426 applications and least are from Academic Degree with 2 applicants.

Children:

- The major number of applicants have no childrens with counts of 1091 applications and having 1 children being the 30% of applications from previous one.

In [ ]:
```
import matplotlib.pyplot as plt

ax = dv.groupby('Education')['Annual_income'].agg(["max", "median"]).plot(kind='barh', f

for p in ax.patches:
    ax.annotate(f"{p.get_width():,.0f}", (p.get_width() - 10000, p.get_y() + p.get_heigh
                ha='center', va='center', color='black', fontsize=10)

plt.xlabel('Salary', fontsize=12)
plt.ylabel('Education', fontsize=12)
plt.title('Median and Maximum Salary by Educational Background', fontsize=14)

plt.show()
```



In [ ]:
```
# Below code explains median and maximum salary as per educational background in our sys
dv.groupby(['Education'])['Annual_income'].agg(["max", "median"])
```

Out[ ]:

Education	max	median
Academic degree	337500	236250.0
Higher education	1575000	202500.0
Incomplete higher	810000	180000.0
Lower secondary	202500	135000.0
Secondary / secondary special	900000	157500.0

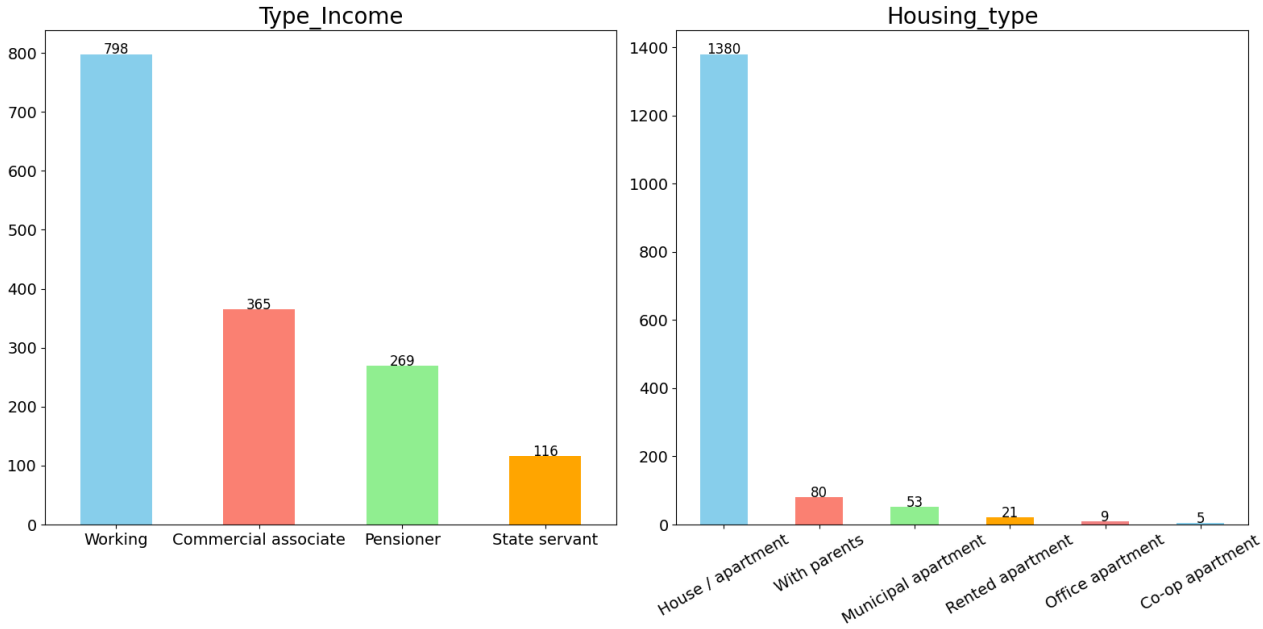
Insights:

The code groups the dataset by educational background and calculates the maximum and median annual income for each group. This helps to understand the salary distribution based on different levels of education.

Show Unique values and their counts in Income Type and Type of Housing

In [ ]:

```
value_counts_income = dv['Type_Income'].value_counts()
value_counts_housing = dv['Housing_type'].value_counts()


plt.figure(figsize=(16, 8))

# Color palette for pie charts
custom_colors = ['skyblue', 'salmon', 'lightgreen', 'orange', 'lightcoral']

plt.subplot(1, 2, 1)
ax = value_counts_income.plot(kind='bar', color = custom_colors)
plt.title('Type_Income', fontsize=20)
plt.xticks(rotation=0)  # Rotate x-axis labels by 0 degrees instead of 90 in general set
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

for i in ax.patches:
    plt.text(i.get_x() + i.get_width() / 2, i.get_height() + 0.1, str(i.get_height()), h

plt.subplot(1, 2, 2)
ax = value_counts_housing.plot(kind='bar', color = custom_colors)
plt.title('Housing_type', fontsize=20)
plt.xticks(rotation=30)  # Rotate x-axis labels by 0 degrees instead of 90 in general se
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

for i in ax.patches:
    plt.text(i.get_x() + i.get_width() / 2, i.get_height() + 0.1, str(i.get_height()), h


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



Insights :

Income Type:

- The major amount of applicants belongs to Working Class, where their average salary being the lowest in all Income groups, followed by Commercial associate, who have highest average salary and then by pensioners, where state servants being the least one.

In [ ]:
```
dv.groupby(["Type_Income"])['Type_Income'].count()
```

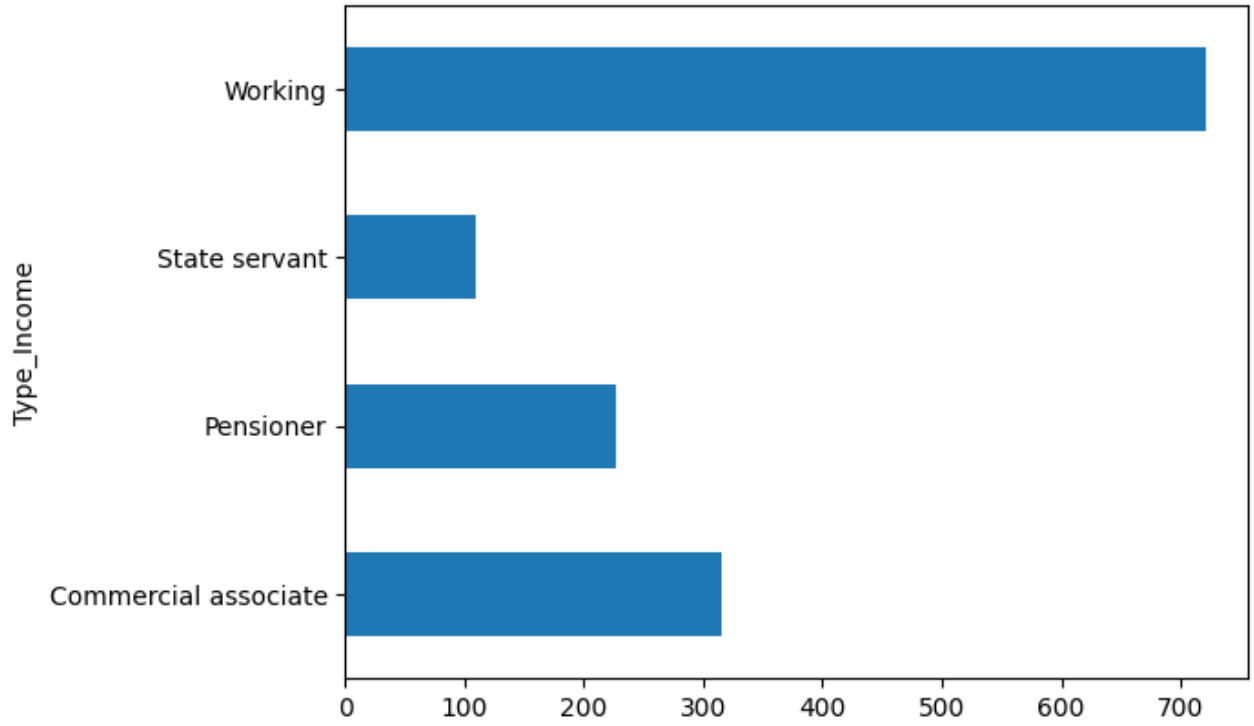
Out[ ]:
```
Type_Income
Commercial associate    365
Pensioner               269
State servant           116
Working                 798
Name: Type_Income, dtype: int64
```

Insights:

The code groups the dataset by the type of income and then counts the number of applications .This analysis helps understand the distribution of application across different types of income.

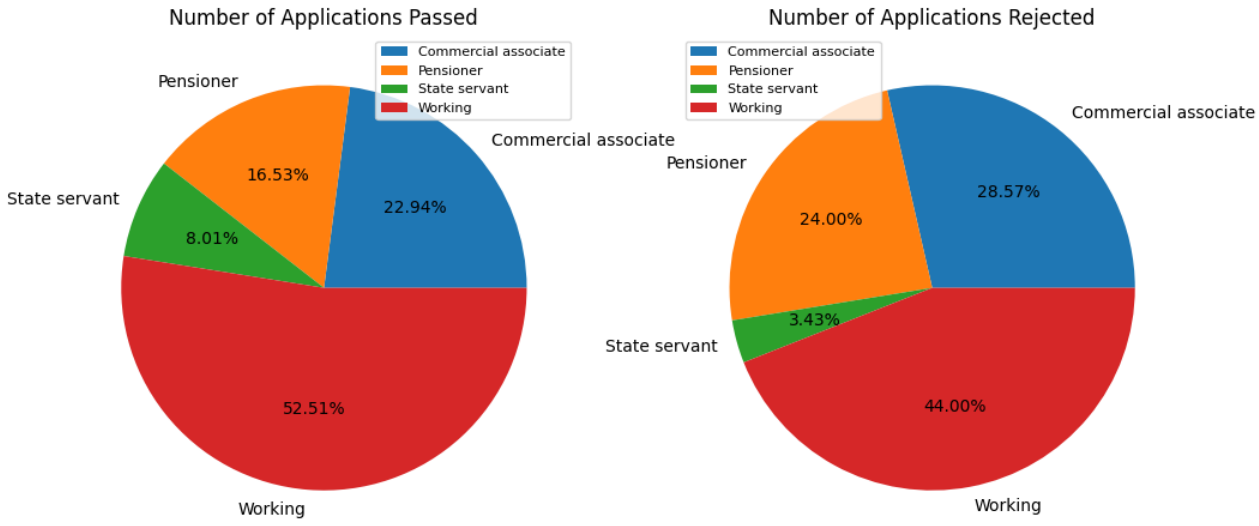
In [ ]:
```
dv[dv['label'] == 0].groupby('Type_Income').size().plot(kind = 'barh')
```

Out[ ]:
```
<Axes: ylabel='Type_Income'>
```



In [ ]:
```
plt.figure(figsize=(12,9))

plt.subplot(1,2,1)
dv[dv['label'] == 0].groupby('Type_Income').size().plot(kind = 'pie', autopct="%.2f%%")
plt.title("Number of Applications Passed")
plt.legend( loc="best", fontsize=8)

plt.subplot(1,2,2)
dv[dv['label'] == 1].groupby('Type_Income').size().plot(kind = 'pie', autopct="%.2f%%")
plt.title("Number of Applications Rejected")
plt.legend( loc="best", fontsize=8)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x26ca3e78450>
```



This code filters the DataFrame dv where the 'label' column is equal to 0(passed applications), then groups the filtered DataFrame by the 'Type_Income' column, and finally returns the count of occurrences within each group.

We can cleary see that Working class has major applications approved irrespective of number of applications.

In [ ]:
```
dv.groupby(['Type_Occupation'])['label'].count()
```

Out[ ]:
```
Type_Occupation
Accountants               44
Cleaning staff            22
Cooking staff             21
Core staff               174
Drivers                   86
HR staff                   3
High skill tech staff     65
IT staff                   2
Laborers                 268
Low-skill Laborers         9
Managers                 136
Medicine staff            50
Pensioner                264
Private service staff     17
Realty agents              2
Sales staff              122
Secretaries                9
Security staff            25
Unknown                  224
Waiters/barmen staff       5
Name: label, dtype: int64
```

Insights:

The code groups the dataset by the type of income and then counts the number of applications .This analysis helps understand the distribution of application across different types of occupation.

In [ ]:
```
income_stats = dv.groupby(['Type_Income'])['Annual_income'].agg(["median", "max"])

# Sort the income statistics by descending order of median income
income_stats_sorted = income_stats.sort_values(by='median', ascending=False)

print(income_stats_sorted)
```

```
                      median      max
Type_Income
Commercial associate  202500.0  1575000
State servant         180000.0   787500
Working               157500.0   900000
Pensioner             135000.0   630000
```

Insights:

The code calculates the median and maximum annual income for each type of income listed in the dataset. It then sorts the results based on the median income in descending order, showcasing which type of income has the highest median earnings.

In [ ]:
```
# Median and maximum salary by type of occupation
occupation_income_stats = dv.groupby(['Type_Occupation'])['Annual_income'].agg(["median"

occupation_income_stats_sorted = occupation_income_stats.sort_values(by='median', ascend

print(occupation_income_stats_sorted)
```

```
                        median      max
Type_Occupation
Managers                225000.0  1575000
Drivers                 213750.0   585000
Realty agents           202500.0   225000
High skill tech staff   180000.0   900000
Unknown                 180000.0   540000
Accountants             180000.0   612000
Laborers                166500.0   900000
Medicine staff          157500.0   306000
Private service staff   157500.0   423000
Core staff              157500.0   450000
Sales staff             157500.0   450000
HR staff                144000.0   180000
Low-skill Laborers      135000.0   225000
Cleaning staff          135000.0   266700
Pensioner               135000.0   630000
Secretaries             135000.0   270000
Security staff          112500.0   810000
Cooking staff           112500.0   495000
IT staff                103500.0   103500
Waiters/barmen staff     94500.0   225000
```

Insights :

The provided code computes two statistics, median and maximum salary, for different types of occupations listed in the dataset:

1. **Median Salary by Occupation**: The code calculates the median (middle value) of annual income for each type of occupation. This median value represents the income level at which half of the individuals in a particular occupation earn more, and the other half earn less.

2. **Maximum Salary by Occupation**: Additionally, the code determines the maximum annual income earned by individuals in each type of occupation. This value reflects the highest income earned by any individual within a specific occupation category.

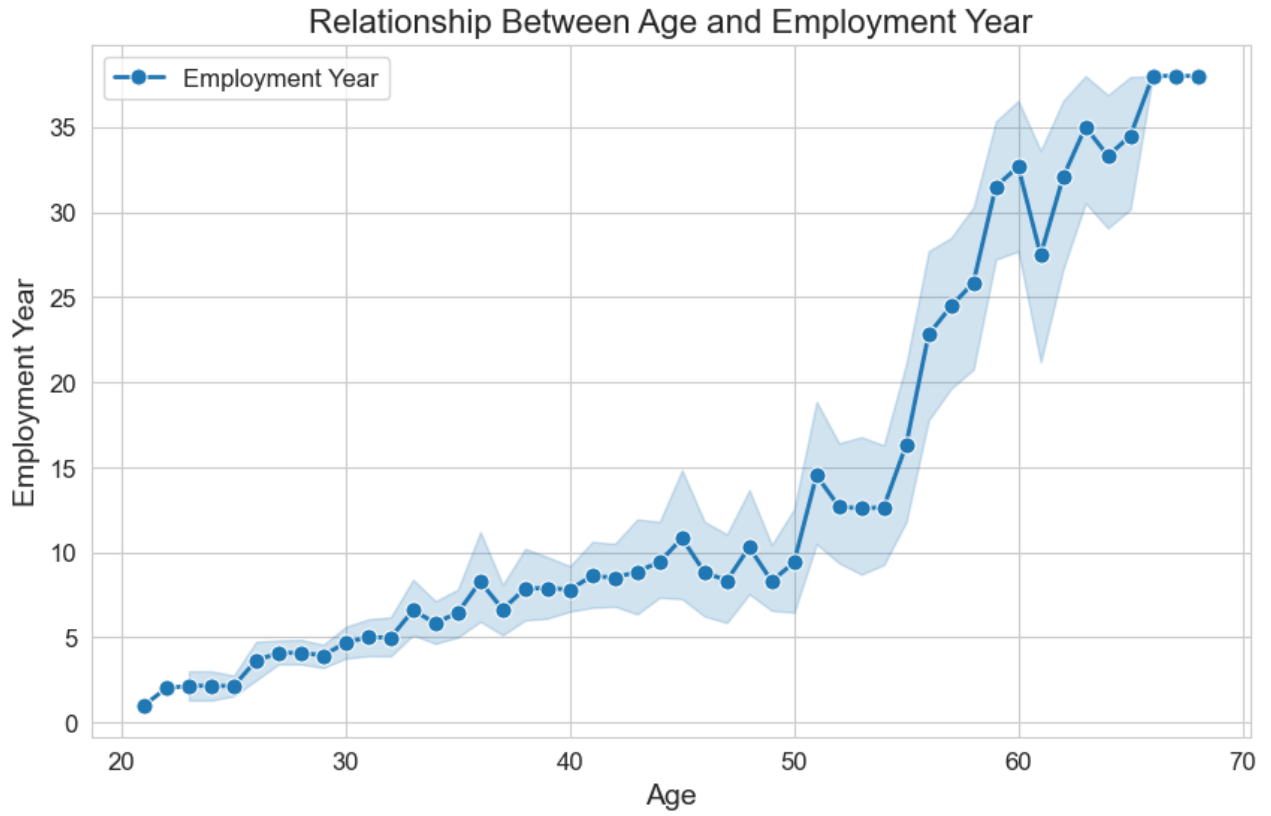
After calculating these statistics, the code sorts the results based on the median salary in descending order (from highest to lowest) and prints the sorted statistics.

In summary, the code provides insights into the income distribution across different types of occupations, highlighting the median income levels and the maximum income potential within each occupation category.

Does Age relates with Experience in Years?

In [ ]:
```python
# Setting the style of the plot
sns.set_style("whitegrid")

# Defining custom color palette
colors = ['#1f77b4']  # Blue color

# Plotting the line plot
plt.figure(figsize=(10, 6))
sns.lineplot(data=dv, x='Age', y='Employment_Year', color=colors[0], marker='o', markers

# Setting the title and labels
plt.title('Relationship Between Age and Employment Year', fontsize=16)
plt.xlabel('Age', fontsize=14)
plt.ylabel('Employment Year', fontsize=14)
```

```
# Setting the font size of ticks
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add legend with appropriate label
plt.legend(['Employment Year'], loc='best', fontsize=12)

# Display the plot
plt.show()
```



In [ ]:
```
# Median and maximum salary by type of occupation
occupation_income_stats = dv.groupby(['Age'])['Employment_Year'].agg(["median", "min", "

occupation_income_stats_sorted = occupation_income_stats.sort_values(by='Age', ascending

print(occupation_income_stats_sorted)
```

Age	median	min	max	count
21	1.0	1.0	1.0	1
22	2.0	2.0	2.0	1
23	2.0	1.0	4.0	7
24	2.0	1.0	4.0	7
25	2.0	0.0	5.0	24
26	4.0	0.0	8.0	19
27	5.0	0.0	8.0	43
28	4.0	1.0	9.0	46
29	4.0	0.0	9.0	39
30	4.0	0.0	11.0	42
31	5.0	0.0	13.0	37
32	3.0	0.0	13.0	44
33	5.0	0.0	38.0	47
34	5.0	0.0	16.0	48
35	6.0	0.0	17.0	43
36	7.0	1.0	38.0	37
37	6.0	0.0	19.0	47
38	5.0	0.0	38.0	43
39	6.0	0.0	20.0	45
40	7.0	0.0	20.0	50
41	7.0	0.0	22.0	39
42	7.0	1.0	38.0	57
43	7.5	1.0	38.0	28
44	7.0	0.0	38.0	53
45	7.0	1.0	38.0	28
46	7.0	0.0	38.0	34
47	5.0	0.0	38.0	40
48	6.0	1.0	38.0	41
49	8.0	1.0	29.0	37
50	8.0	0.0	38.0	35
51	10.0	0.0	38.0	27
52	9.0	0.0	38.0	44
53	7.5	1.0	38.0	30
54	9.0	1.0	38.0	31
55	9.5	1.0	38.0	36
56	22.5	3.0	38.0	34
57	31.0	3.0	38.0	37
58	38.0	1.0	38.0	32
59	38.0	2.0	38.0	36
60	38.0	2.0	38.0	33
61	38.0	1.0	38.0	23
62	38.0	0.0	38.0	22
63	38.0	6.0	38.0	21
64	38.0	1.0	38.0	33
65	38.0	6.0	38.0	19
66	38.0	38.0	38.0	10
67	38.0	38.0	38.0	16
68	38.0	38.0	38.0	2

Insights:

The above provided code calculates various statistics related to the employment years based on age groups in the dataset:

1. **Median Employment Year**: For each age group, it calculates the median number of years of employment, which represents the midpoint value of the distribution of employment years. This gives an idea of the typical duration of employment for individuals in different age brackets.

2. **Minimum Employment Year**: It determines the minimum number of years of employment within each age group, indicating the shortest duration of employment among individuals in that age category.

3. **Maximum Employment Year**: This calculates the maximum number of years of employment within each age group, representing the longest duration of employment among individuals in that age bracket.

4. **Count**: The count indicates the number of individuals in each age group, providing insight into the sample size or population size for each age category.

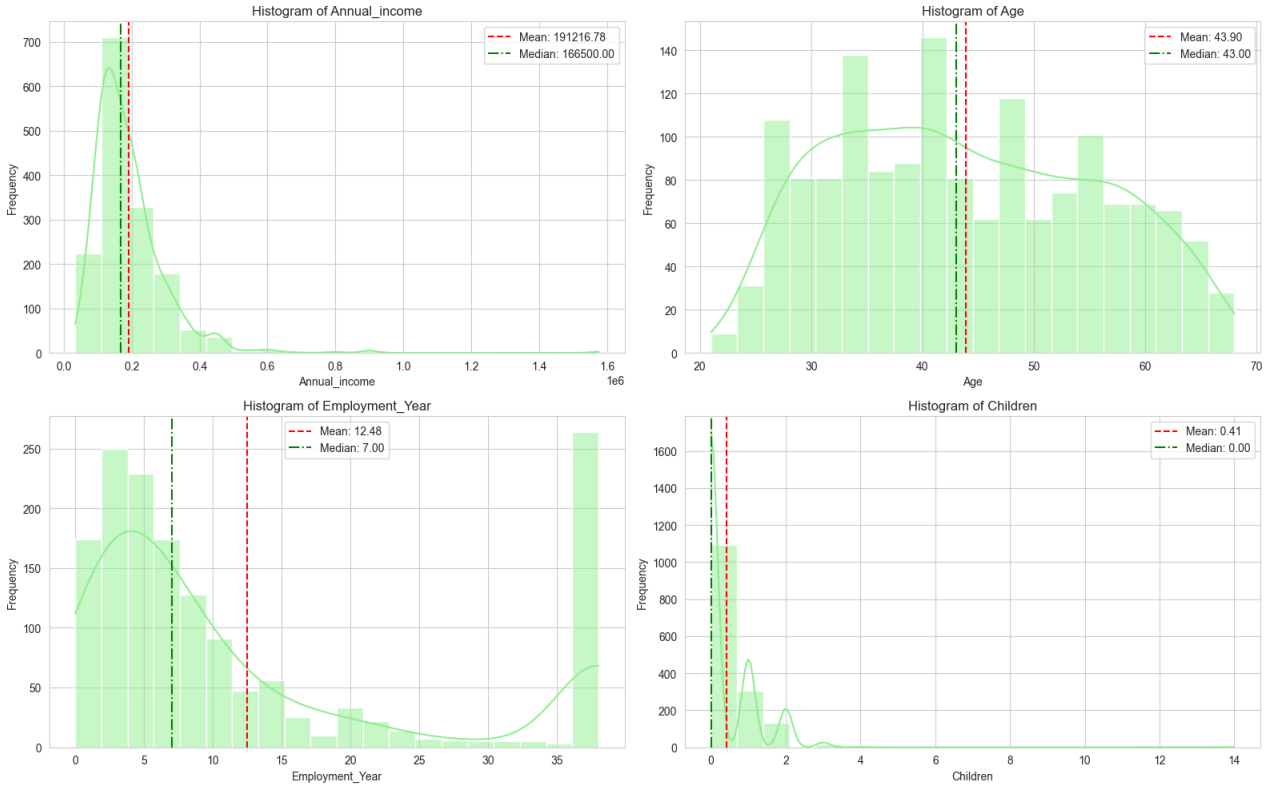
The code then sorts these statistics based on age in ascending order and displays the results. Overall, it helps in understanding the distribution of employment years across different age groups in the dataset.

In [ ]:
```
dv.select_dtypes(include=['int64', 'float64']).columns
```

Out[ ]:
```
Index(['Ind_ID', 'Children', 'Annual_income', 'Age', 'Employment_Year',
       'Mobile_phone', 'Work_Phone', 'Phone', 'Email_ID', 'Family_Members',
       'label'],
      dtype='object')
```

Provide the Distribution plot for 'Children', 'Annual_income', 'Age', 'Employment_Year' and show 'Mean', and 'Median' values of same

In [ ]:
```
# Plot histograms for each numeric column
numeric_columns = ['Annual_income', 'Age', 'Employment_Year','Children']

plt.figure(figsize=(16, 10))

for i, column in enumerate(numeric_columns, start=1):
    # Plot histogram
    plt.subplot(2, 2, i)
    sns.histplot(dv[column], bins=20, color='lightgreen', kde=True)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.title(f'Histogram of {column}')
    plt.grid(True)

    # Calculate mean and median
    mean_value = dv[column].mean()
    median_value = dv[column].median()

    # Plot mean and median lines
    plt.axvline(x=mean_value, color='red', linestyle='--', label=f'Mean: {mean_value:.2f
    plt.axvline(x=median_value, color='green', linestyle='-.', label=f'Median: {median_v
    plt.legend()

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

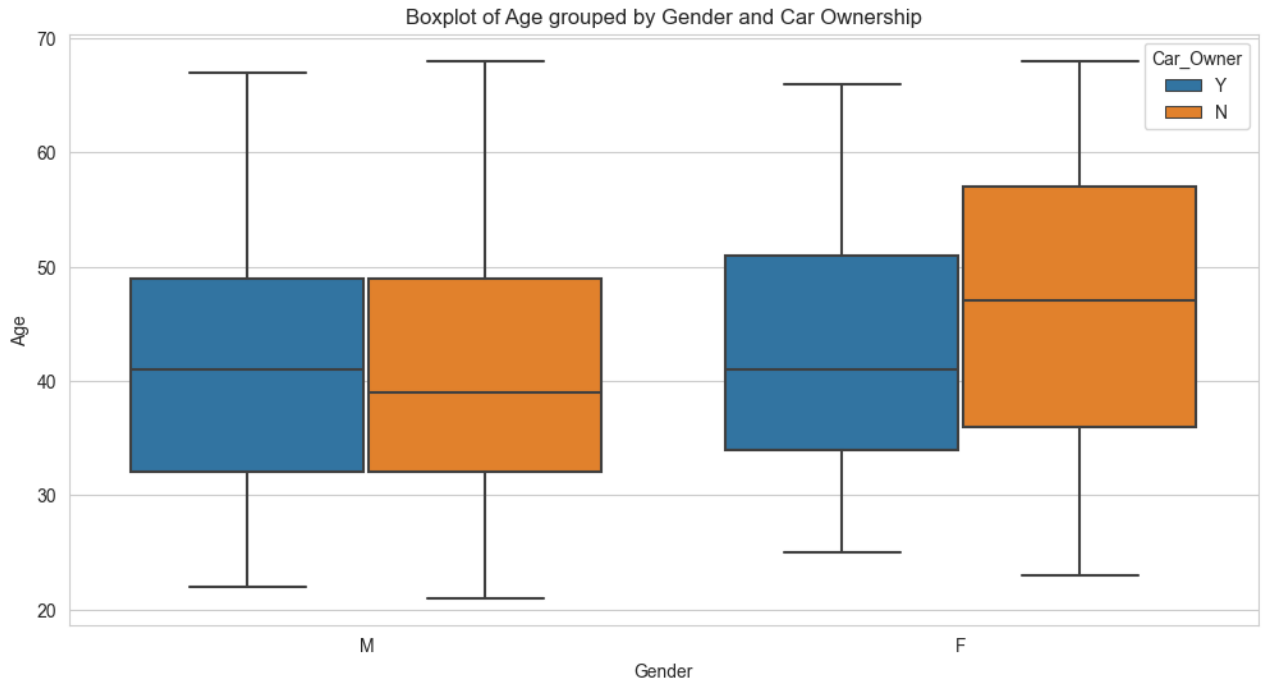


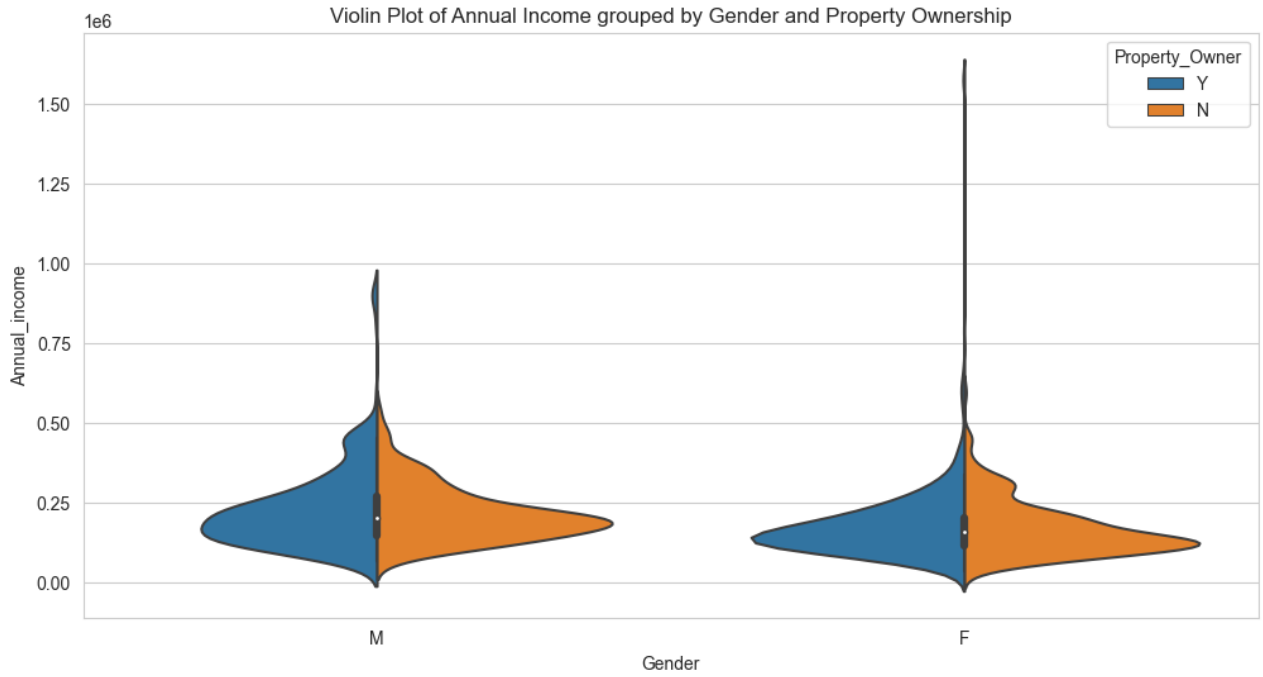
Bivariate Analysis

Provide details of gender who purchased car and what is the major range of age for the same?

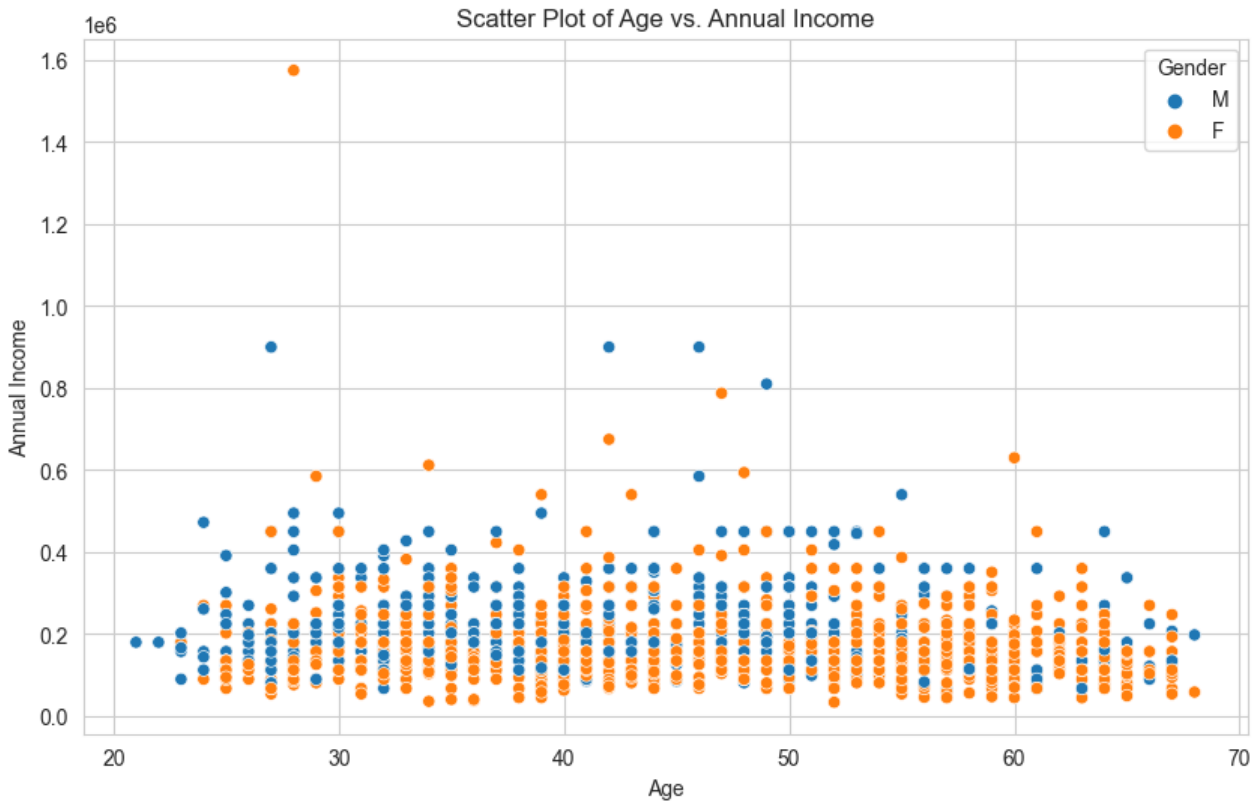
In [ ]:

```
# Boxplot with Categorical Variables
plt.figure(figsize=(12, 6))
sns.boxplot(data=dv, x='Gender', y='Age', hue='Car_Owner')
plt.title('Boxplot of Age grouped by Gender and Car Ownership')
plt.show()

# Violin Plot
plt.figure(figsize=(12, 6))
sns.violinplot(data=dv, x='Gender', y='Annual_income', hue='Property_Owner', split=True)
plt.title('Violin Plot of Annual Income grouped by Gender and Property Ownership')
plt.show()
```



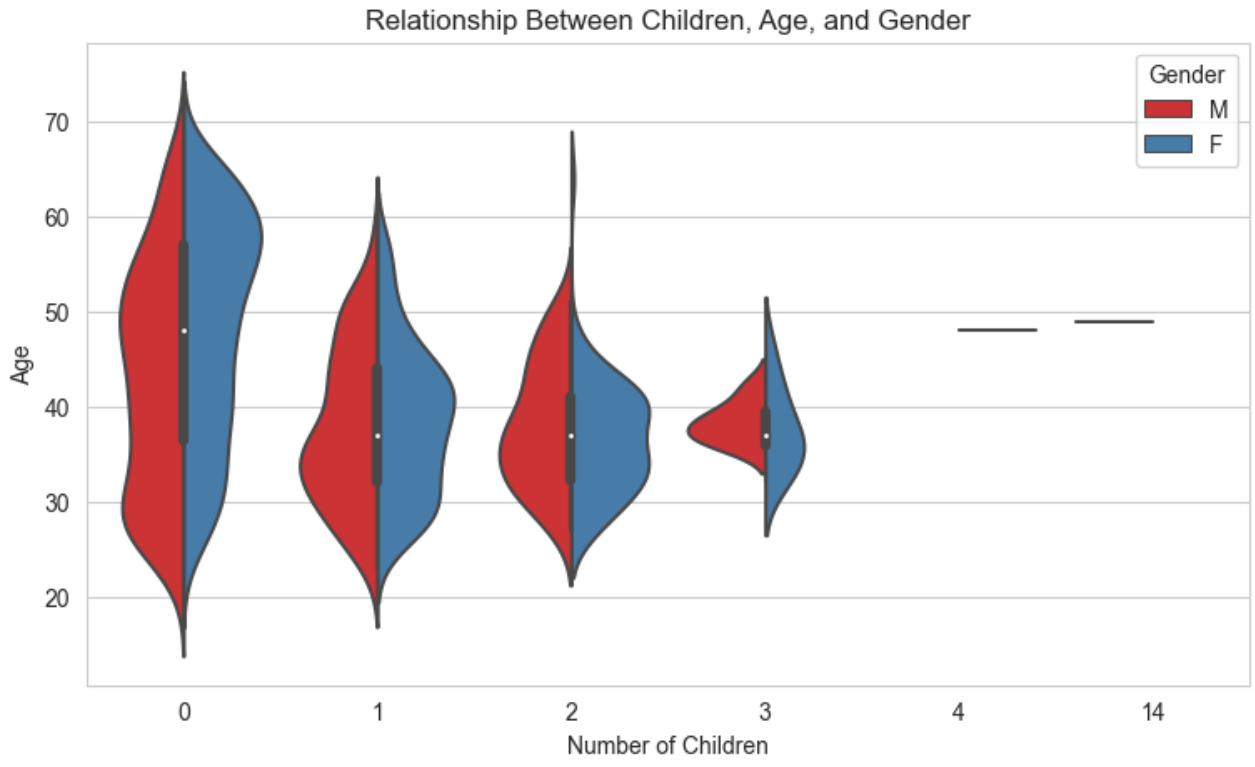


In [ ]:
```
# Scatter plot for Age vs. Annual_income
plt.figure(figsize=(10, 6))
sns.scatterplot(data=dv, x='Age', y='Annual_income', hue='Gender')
plt.title('Scatter Plot of Age vs. Annual Income')
plt.xlabel('Age')
plt.ylabel('Annual Income')
plt.show()
```



In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting a violin plot
plt.figure(figsize=(9, 5))
sns.violinplot(data=dv, x='Children', y='Age', hue='Gender', palette='Set1', split=True)
plt.title('Relationship Between Children, Age, and Gender')
plt.xlabel('Number of Children')
plt.ylabel('Age')
plt.legend(title='Gender')
plt.show()
```



3 - New Dataset - Savepoint 3

E. Graphical Representation of KNN and MICE Imputation

In [ ]:
```
df3 = pd.read_csv("4.Credit_card_clean_Data_Analysis.csv")
```

In [ ]:
```
print(df3.isna().sum())
```

```
Ind_ID             0
GENDER             0
Car_Owner          0
Propert_Owner      0
CHILDREN           0
Annual_income      0
Type_Income        0
EDUCATION          0
Marital_status     0
Housing_type       0
Age                0
Employment_Year    0
Mobile_phone       0
Work_Phone         0
Phone              0
EMAIL_ID           0
Type_Occupation    0
Family_Members     0
label              0
dtype: int64
```

Creating another copy to plot graph for comparing distribution before imputation and after imputation and with KNN & MICE

"dfmice" dataset is created after Savepoint 4, but just for compairng KNN-Imputed values with MICE-Imputation, have shown below plots. Here for this data KNN-Imputation suits the best for Imputating values, and hence we go for further analysis using the KNN-Imputed values. MICE-Imputation is performed below with another dataset (categorical to - numerical converted dataset) just for information and to fetch plots to compare in this case.

In [ ]:
```
dfmice = pd.read_csv("dfmice.csv")
```

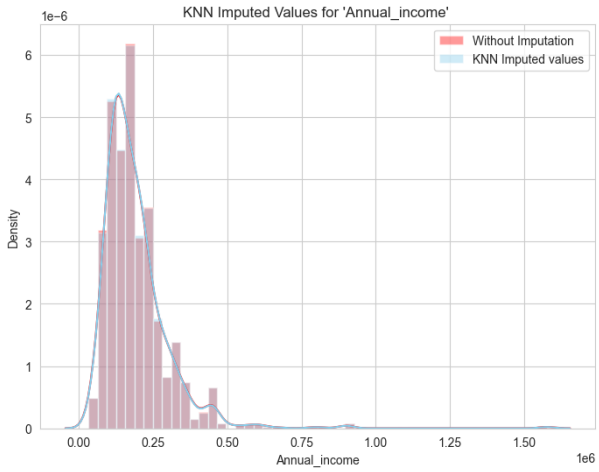
In [ ]:
```
dfmice1 = dfmice.copy()
```

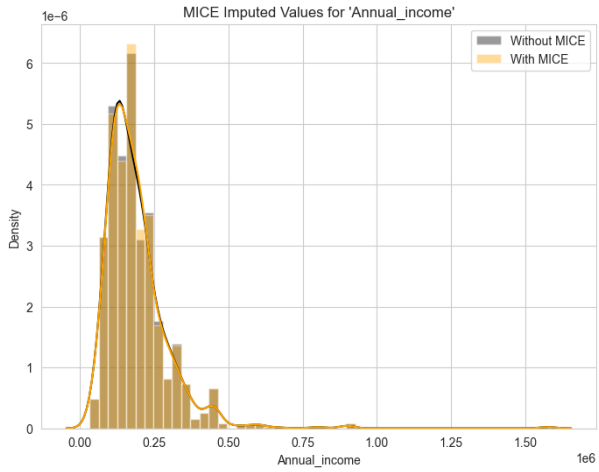
Annual_income comparison

In [ ]:
```
plt.figure(figsize=(18,6))

# Plot for MICE vs Without MICE
plt.subplot(1,2,2)
sns.distplot(df3['Annual_income'], color = 'black',  label = "Without MICE")
sns.distplot(dfmice1['Annual_income'], color = 'orange', label = "With MICE")
plt.legend()
plt.title("MICE Imputed Values for 'Annual_income'")
plt.grid(True)  # Adding gridlines

# Plot for KNN Imputed vs Without Imputed
```

```
plt.subplot(1,2,1)
sns.distplot(df["Annual_income"], kde=True, color = 'red', label = 'Without Imputation')
sns.distplot(df3["Annual_income"], kde=True, color='skyblue', label = 'KNN Imputed value
plt.title("KNN Imputed Values for 'Annual_income' ")
plt.grid(True)  # Adding gridlines
plt.legend()

plt.show()
```

```
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\733304567.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df3['Annual_income'], color = 'black',  label = "Without MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\733304567.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dfmice1['Annual_income'], color = 'orange', label = "With MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\733304567.py:13: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Annual_income"], kde=True, color = 'red', label = 'Without Imputatio
n')
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\733304567.py:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df3["Annual_income"], kde=True, color='skyblue', label = 'KNN Imputed val
ues')
```



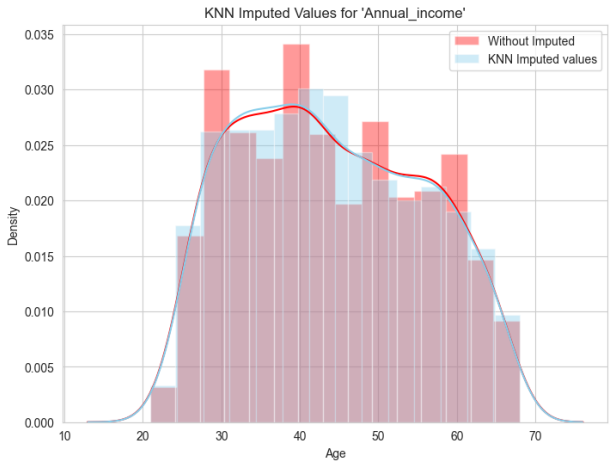


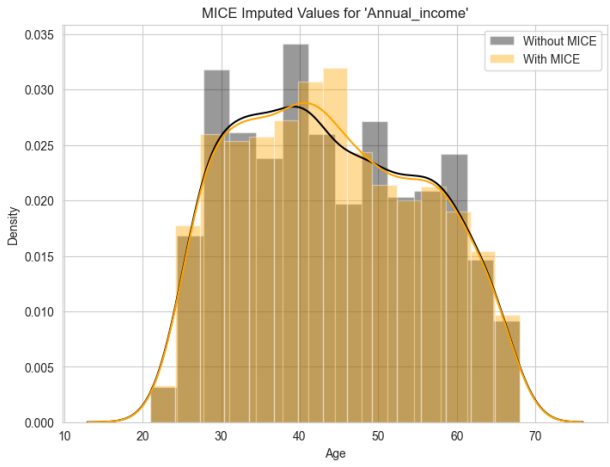
Age comparison

In [ ]:

```
plt.figure(figsize=(18,6))

# Plot for MICE vs Without MICE
plt.subplot(1,2,2)
sns.distplot(df1['Age'], color='black', label="Without MICE")
sns.distplot(dfmice1['Age'], color='orange', label="With MICE")
plt.legend()
plt.title("MICE Imputed Values for 'Annual_income'")
plt.grid(True)  # Adding gridlines

# Plot for KNN Imputed vs Without Imputed
plt.subplot(1,2,1)
sns.distplot(df["Age"], kde=True, color='red', label="Without Imputed")
sns.distplot(df3["Age"], kde=True, color='skyblue', label="KNN Imputed values")
plt.title("KNN Imputed Values for 'Annual_income'")
plt.legend()
plt.grid(True)  # Adding gridlines

plt.show()
```

```
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\259428918.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df1['Age'], color='black', label="Without MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\259428918.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dfmice1['Age'], color='orange', label="With MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\259428918.py:13: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Age"], kde=True, color='red', label="Without Imputed")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\259428918.py:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df3["Age"], kde=True, color='skyblue', label="KNN Imputed values")
```





From above graph we can say that distribution is quite normal and an use the imputed values by KNN method for further analysis

In [ ]:
```
df3.sample(3)
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATI
189	5022326	M	Y	Y	0	256500	Commercial associate	Hig educa
1131	5100464	M	Y	Y	1	315000	Working	Hig educa
622	5023724	M	Y	N	1	495000	Working	Seconda secon spe

F. Feature Engineering : Label Encodings

Details about "object" type columns

In [ ]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1548 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1548 non-null   int64  
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Age              1548 non-null   int64  
 11  Employment_Year  1548 non-null   float64
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1548 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(1), int64(10), object(8)
memory usage: 229.9+ KB
```

In [ ]:
```
df3.isna().sum()
```

Out[ ]:
```
Ind_ID             0
GENDER             0
Car_Owner          0
Propert_Owner      0
CHILDREN           0
Annual_income      0
Type_Income        0
EDUCATION          0
Marital_status     0
Housing_type       0
Age                0
Employment_Year    0
Mobile_phone       0
Work_Phone         0
Phone              0
EMAIL_ID           0
Type_Occupation    0
Family_Members     0
label              0
dtype: int64
```

In [ ]:
```
df3.select_dtypes(include='object').columns
```

Out[ ]:
```
Index(['GENDER', 'Car_Owner', 'Propert_Owner', 'Type_Income', 'EDUCATION',
       'Marital_status', 'Housing_type', 'Type_Occupation'],
      dtype='object')
```

In [ ]:
```
cols = ['GENDER', 'Car_Owner', 'Propert_Owner', 'Type_Income', 'EDUCATION', 'Marital_sta

for col in cols:
    unique_values = df3[col].unique()
    print(f"Column: {col}")
    for val in unique_values:
        ct = (df3[col] == val).sum()
        print(f"Value: {val}", "-->", f"Count: {ct}")
    print("-------" * 7)
```

```
Column: GENDER
Value: M --> Count: 568
Value: F --> Count: 980
-----------------------------------------------
Column: Car_Owner
Value: Y --> Count: 624
Value: N --> Count: 924
-----------------------------------------------
Column: Propert_Owner
Value: Y --> Count: 1010
Value: N --> Count: 538
-----------------------------------------------
Column: Type_Income
Value: Pensioner --> Count: 269
Value: Commercial associate --> Count: 365
Value: Working --> Count: 798
Value: State servant --> Count: 116
-----------------------------------------------
Column: EDUCATION
Value: Higher education --> Count: 426
Value: Secondary / secondary special --> Count: 1031
Value: Lower secondary --> Count: 21
Value: Incomplete higher --> Count: 68
Value: Academic degree --> Count: 2
-----------------------------------------------
Column: Marital_status
Value: Married --> Count: 1049
Value: Single / not married --> Count: 227
Value: Civil marriage --> Count: 101
Value: Separated --> Count: 96
Value: Widow --> Count: 75
-----------------------------------------------
Column: Housing_type
Value: House / apartment --> Count: 1380
Value: With parents --> Count: 80
Value: Rented apartment --> Count: 21
Value: Municipal apartment --> Count: 53
Value: Co-op apartment --> Count: 5
Value: Office apartment --> Count: 9
-----------------------------------------------
Column: Type_Occupation
Value: Pensioner --> Count: 264
Value: Unknown --> Count: 224
Value: Core staff --> Count: 174
Value: Cooking staff --> Count: 21
Value: Laborers --> Count: 268
Value: Sales staff --> Count: 122
Value: Accountants --> Count: 44
Value: High skill tech staff --> Count: 65
Value: Managers --> Count: 136
Value: Cleaning staff --> Count: 22
Value: Drivers --> Count: 86
Value: Low-skill Laborers --> Count: 9
Value: IT staff --> Count: 2
Value: Waiters/barmen staff --> Count: 5
Value: Security staff --> Count: 25
Value: Medicine staff --> Count: 50
Value: Private service staff --> Count: 17
Value: HR staff --> Count: 3
Value: Secretaries --> Count: 9
Value: Realty agents --> Count: 2
-----------------------------------------------
```

Label Encoding

Ordinal data : Ordinal Encoding (using map / Encoder())

```
Type_Occupation
Car_Owner
Propert_Owner
Housing_type
EDUCATION
Type_Income
```

Nomial Data : get_dummies

```
GENDER
Marital_status
```

Nominal Data Conversion

GENDER & Martial_status

In [ ]:
```
df3 = pd.get_dummies(df3, columns = ["GENDER", "Marital_status",], drop_first=True)
```

In [ ]:
```
df3.sample(3)
```

Out[ ]:

	Ind_ID	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Hous
441	5137022	N	N	0	180000	State servant	Secondary / secondary special	a
93	5089162	Y	N	0	225000	Working	Higher education	a
1385	5069141	N	N	0	135000	Commercial associate	Higher education	Wit

3 rows × 22 columns

Ordinal Data Conversion - get_dummies

Conversion of "Car_Owner" & "Propert_Owner" column

In [ ]:
```
df3 = pd.get_dummies(df3, columns = ["Car_Owner", "Propert_Owner",], drop_first=True)
```

Conversion of "Type_Income" column

Here we can go for ordinal encoder but, analyzing the data we understood that income ranges are highly variable in all of the income types and placing them in a ranking is bit difficult hence we go for get_dummies for this case, even though its an ordinal data.

In [ ]:
```
df3['Type_Income'].unique()
```

Out[ ]:
```
array(['Pensioner', 'Commercial associate', 'Working', 'State servant'],
      dtype=object)
```

In [ ]:
```
df3 = pd.get_dummies(df3, columns = ["Type_Income"], drop_first=True)
```

In [ ]:
```
df3.head()
```

Out[ ]:

	Ind_ID	CHILDREN	Annual_income	EDUCATION	Housing_type	Age	Employment_Year	Mobile_pho
0	5008827	0	180000	Higher education	House / apartment	51	38.0	
1	5009744	0	315000	Higher education	House / apartment	37	2.0	
2	5009746	0	315000	Higher education	House / apartment	37	2.0	
3	5009749	0	315000	Higher education	House / apartment	37	2.0	
4	5009752	0	315000	Higher education	House / apartment	37	2.0	

5 rows × 24 columns

Ordinal Data Conversion - ordinal_encoder

Conversion of "Type_Occupation" column

In [ ]:
```
df3['Type_Occupation'].unique()
```

Out[ ]:
```
array(['Pensioner', 'Unknown', 'Core staff', 'Cooking staff', 'Laborers',
       'Sales staff', 'Accountants', 'High skill tech staff', 'Managers',
       'Cleaning staff', 'Drivers', 'Low-skill Laborers', 'IT staff',
       'Waiters/barmen staff', 'Security staff', 'Medicine staff',
       'Private service staff', 'HR staff', 'Secretaries',
       'Realty agents'], dtype=object)
```

In [ ]:
```
from sklearn.preprocessing import OrdinalEncoder

rank = ['Pensioner', 'Unknown', 'Core staff', 'Cooking staff', 'Laborers',
       'Sales staff', 'Accountants', 'High skill tech staff', 'Managers',
       'Cleaning staff', 'Drivers', 'Low-skill Laborers', 'IT staff',
       'Waiters/barmen staff', 'Security staff', 'Medicine staff',
       'Private service staff', 'HR staff', 'Secretaries',
       'Realty agents']

oe = OrdinalEncoder(categories=[rank])
df3['Type_Occupation'] = oe.fit_transform(df3[['Type_Occupation']])
df3.sample(4)
```

Out[ ]:

	Ind_ID	CHILDREN	Annual_income	EDUCATION	Housing_type	Age	Employment_Year	Mobile_
271	5066843	0	135000	Secondary / secondary special	House / apartment	47	38.0	
89	5079166	0	202500	Secondary / secondary special	With parents	38	4.0	
335	5117047	0	180000	Secondary / secondary special	House / apartment	63	38.0	
1228	5078805	0	306000	Secondary / secondary special	House / apartment	52	8.0	

4 rows × 24 columns

In [ ]:
```
df3['Type_Occupation'].unique()
```

Out[ ]:
```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
       13., 14., 15., 16., 17., 18., 19.])
```

Conversion of "Housing_type" column - Ordinal Encode (map)

In [ ]:
```
df3['Housing_type'].unique()
```

Out[ ]:
```
array(['House / apartment', 'With parents', 'Rented apartment',
       'Municipal apartment', 'Co-op apartment', 'Office apartment'],
      dtype=object)
```

In [ ]:
```
mappings = {"Housing_type" :
    {
        'With parents':0,
        'Rented apartment':1,
        'Municipal apartment':2,
        'Co-op apartment':3,
        'Office apartment':4,
        'House / apartment':5
    }
}

for i in df1:
    if i in mappings:
        df3[i] = df3[i].map(mappings[i])

df3['Housing_type'].head()
```

Out[ ]:
```
0    5
1    5
2    5
3    5
4    5
Name: Housing_type, dtype: int64
```

In [ ]:
```
df3.sample(4)
```

Out[ ]:

	Ind_ID	CHILDREN	Annual_income	EDUCATION	Housing_type	Age	Employment_Year	Mobile_
90	5079167	0	202500	Secondary / secondary special	0	38	4.0	
669	5135938	2	315000	Secondary / secondary special	5	43	10.0	
803	5037125	0	180000	Higher education	0	32	6.0	
1003	5028569	0	315000	Secondary / secondary special	5	43	15.0	

4 rows × 24 columns

Conversion of "EDUCATION" column - Ordinal Encode (map)

In [ ]:
```
df3['EDUCATION'].unique()
```

Out[ ]:
```
array(['Higher education', 'Secondary / secondary special',
       'Lower secondary', 'Incomplete higher', 'Academic degree'],
      dtype=object)
```

In [ ]:
```
mappings = {"EDUCATION" :
    {
        'Lower secondary':0,
        'Secondary / secondary special':1,
        'Incomplete higher':2,
        'Higher education':3,
        'Academic degree':4,
    }
}

for i in df3:
    if i in mappings:
        df3[i] = df3[i].map(mappings[i])

df3['EDUCATION'].head()
```

Out[ ]:
```
0    3
1    3
2    3
3    3
4    3
Name: EDUCATION, dtype: int64
```

In [ ]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 24 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Ind_ID                                1548 non-null   int64
 1   CHILDREN                              1548 non-null   int64
 2   Annual_income                         1548 non-null   int64
 3   EDUCATION                             1548 non-null   int64
 4   Housing_type                          1548 non-null   int64
 5   Age                                   1548 non-null   int64
 6   Employment_Year                       1548 non-null   float64
 7   Mobile_phone                          1548 non-null   int64
 8   Work_Phone                            1548 non-null   int64
 9   Phone                                 1548 non-null   int64
 10  EMAIL_ID                              1548 non-null   int64
 11  Type_Occupation                       1548 non-null   float64
 12  Family_Members                        1548 non-null   int64
 13  label                                 1548 non-null   int64
 14  GENDER_M                              1548 non-null   uint8
 15  Marital_status_Married                1548 non-null   uint8
 16  Marital_status_Separated              1548 non-null   uint8
 17  Marital_status_Single / not married   1548 non-null   uint8
 18  Marital_status_Widow                  1548 non-null   uint8
 19  Car_Owner_Y                           1548 non-null   uint8
 20  Propert_Owner_Y                       1548 non-null   uint8
 21  Type_Income_Pensioner                 1548 non-null   uint8
 22  Type_Income_State servant             1548 non-null   uint8
 23  Type_Income_Working                   1548 non-null   uint8
dtypes: float64(2), int64(12), uint8(10)
memory usage: 184.6 KB
```

from above we confirm that whole dataset is converted to numerical data type and have replaced all the object type data to numerical ones, further we can proceed for prediction values using MICE for filling null values in those converted categorical columns

In [ ]:
```
df3.isnull().sum()
```

Out[ ]:
```
Ind_ID                                0
CHILDREN                              0
Annual_income                         0
EDUCATION                             0
Housing_type                          0
Age                                   0
Employment_Year                       0
Mobile_phone                          0
Work_Phone                            0
Phone                                 0
EMAIL_ID                              0
Type_Occupation                       0
Family_Members                        0
label                                 0
GENDER_M                              0
Marital_status_Married                0
Marital_status_Separated              0
Marital_status_Single / not married   0
Marital_status_Widow                  0
Car_Owner_Y                           0
Propert_Owner_Y                       0
Type_Income_Pensioner                 0
Type_Income_State servant             0
Type_Income_Working                   0
dtype: int64
```

In [ ]:
```
df3.head()
```

Out[ ]:

	Ind_ID	CHILDREN	Annual_income	EDUCATION	Housing_type	Age	Employment_Year	Mobile_pho
0	5008827	0	180000	3	5	51	38.0	
1	5009744	0	315000	3	5	37	2.0	
2	5009746	0	315000	3	5	37	2.0	
3	5009749	0	315000	3	5	37	2.0	
4	5009752	0	315000	3	5	37	2.0	

5 rows × 24 columns

In [ ]:
```
df3.to_csv("5.Credit_card_Cleaned_ML.csv", index = False)
```

The above created dataset could be used for model training in Machine Learning further.

4 - New Dataset - Savepoint 4

```
df4 = pd.read_csv("5.Credit_card_Cleaned.csv")
```

```
#
```

CAUTION BEFORE RUNNING CODES FOR MICE IMPUTED VALUES THEY'RE JUST FOR COMPARISON PURPOSE with KNN iMPUTATION

MICE Imputation for Age, Income

In [ ]:
```
dfmice = df1.copy()
```

We're using df1 dataset, as we need empty/null values in columns to be filled.

In [ ]:
```
dfmice.isna().sum()
```

Out[ ]:
```
Ind_ID               0
GENDER               7
Car_Owner            0
Propert_Owner        0
CHILDREN             0
Annual_income       23
Type_Income          0
EDUCATION            0
Marital_status       0
Housing_type         0
Age                 22
Employment_Year      0
Mobile_phone         0
Work_Phone           0
Phone                0
EMAIL_ID             0
Type_Occupation    488
Family_Members       0
label                0
dtype: int64
```

In [ ]:
```
dfmice.head()
```

Out[ ]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education
3	5009749	F	Y	N	0	NaN	Commercial associate	Higher education
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education

In [ ]:
```
import pandas as pd
from fancyimpute import IterativeImputer

# Assuming Capped_values_2_CCB2 is your DataFrame
# Select only the relevant columns for imputation
columns_to_impute = ["Age", "Annual_income"]

# Perform MICE imputation
imputer = IterativeImputer()
dfmice[columns_to_impute] = imputer.fit_transform(dfmice[columns_to_impute])

# Check if null values have been filled
print(dfmice[columns_to_impute].isnull().sum())
```

```
Age              0
Annual_income    0
dtype: int64
```

We've imputed values for "Age" and "Annual_income" columns in "dfmice" dataset.

In [ ]:
```
dfmice.isnull().sum()
```

Out[ ]:
```
Ind_ID               0
GENDER               7
Car_Owner            0
Propert_Owner        0
CHILDREN             0
Annual_income        0
Type_Income          0
EDUCATION            0
Marital_status       0
Housing_type         0
Age                  0
Employment_Year      0
Mobile_phone         0
Work_Phone           0
Phone                0
EMAIL_ID             0
Type_Occupation    488
Family_Members       0
label                0
dtype: int64
```

In [ ]:
```
dfmice.to_csv("dfmice.csv", index = False)
```

In [ ]:
```
dfmice.loc[:, 'Annual_income':"Age"]
```

Out[ ]:

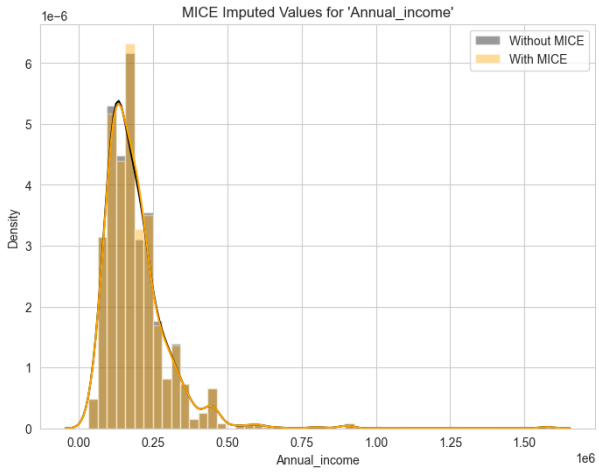
	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Age
0	180000.000000	Pensioner	Higher education	Married	House / apartment	51.000000
1	315000.000000	Commercial associate	Higher education	Married	House / apartment	37.000000
2	315000.000000	Commercial associate	Higher education	Married	House / apartment	42.524097
3	198626.375557	Commercial associate	Higher education	Married	House / apartment	37.000000
4	315000.000000	Commercial associate	Higher education	Married	House / apartment	37.000000
...	...	...	...	...	...	...
1543	202802.780917	Commercial associate	Higher education	Married	House / apartment	33.000000
1544	225000.000000	Commercial associate	Incomplete higher	Single / not married	House / apartment	28.000000
1545	180000.000000	Working	Higher education	Married	House / apartment	36.000000
1546	270000.000000	Working	Secondary / secondary special	Civil marriage	House / apartment	42.000000
1547	225000.000000	Working	Higher education	Married	House / apartment	45.000000

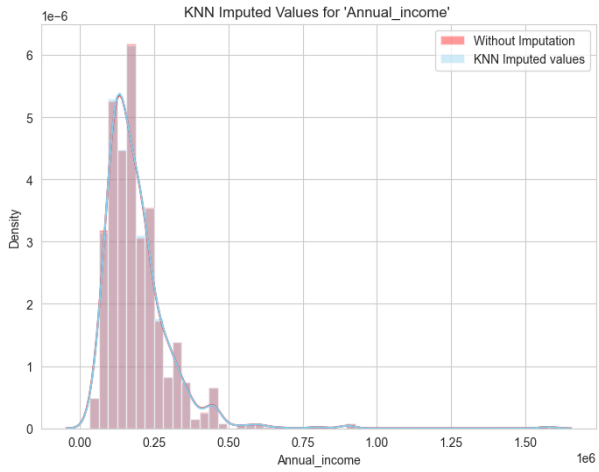
1548 rows × 6 columns

In [ ]:
```
plt.figure(figsize=(18,6))
plt.subplot(1,2,1)
sns.distplot(df3['Annual_income'], color = 'black',  label = "Without MICE")
sns.distplot(dfmice['Annual_income'], color = 'orange', label = "With MICE")
```

```
plt.legend()
plt.title("MICE Imputed Values for 'Annual_income'")

plt.subplot(1,2,2)
sns.distplot(df["Annual_income"], kde=True, color = 'red', label = 'Without Imputation')
sns.distplot(df3["Annual_income"], kde=True, color='skyblue', label = 'KNN Imputed value
plt.title("KNN Imputed Values for 'Annual_income' ")
plt.legend()
plt.show()
```

```
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\873382007.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df3['Annual_income'], color = 'black',  label = "Without MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\873382007.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dfmice['Annual_income'], color = 'orange', label = "With MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\873382007.py:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Annual_income"], kde=True, color = 'red', label = 'Without Imputatio
n')
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\873382007.py:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df3["Annual_income"], kde=True, color='skyblue', label = 'KNN Imputed val
ues')
```

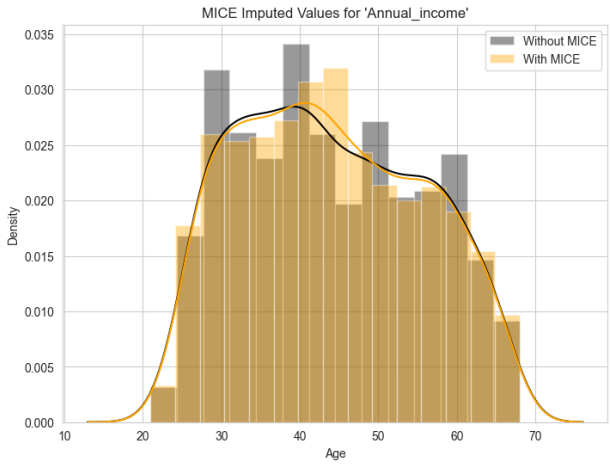


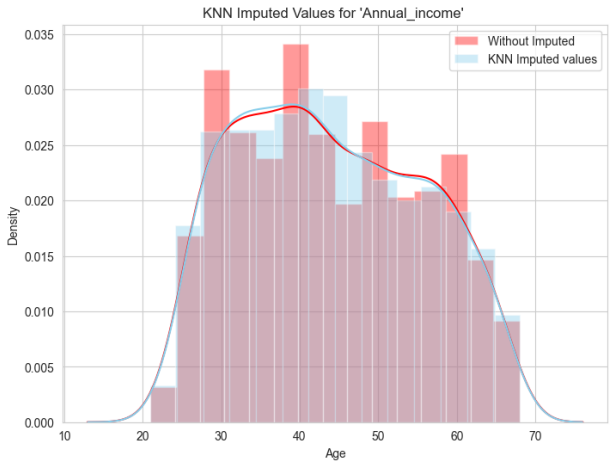


In [ ]:

```
plt.figure(figsize=(18,6))
plt.subplot(1,2,1)
sns.distplot(df1['Age'], color = 'black', label = "Without MICE")
sns.distplot(dfmice['Age'], color = 'orange', label = "With MICE")
plt.legend()
plt.title("MICE Imputed Values for 'Annual_income'")

plt.subplot(1,2,2)
sns.distplot(df["Age"], kde=True, color = 'red', label="Without Imputed")
sns.distplot(df3["Age"], kde=True, color='skyblue', label = "KNN Imputed values")
plt.title("KNN Imputed Values for 'Annual_income' ")
plt.legend()
plt.show()
```

```
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\1594115151.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df1['Age'], color = 'black', label = "Without MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\1594115151.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dfmice['Age'], color = 'orange', label = "With MICE")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\1594115151.py:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Age"], kde=True, color = 'red', label="Without Imputed")
C:\Users\RKC\AppData\Local\Temp\ipykernel_13236\1594115151.py:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df3["Age"], kde=True, color='skyblue', label = "KNN Imputed values")
```





#

G. Feature Selection & Feature Scaling

In [ ]:
```
df4 = pd.read_csv("5.Credit_card_Cleaned_ML.csv")
```

In [ ]:
```
df4.head()
```

Out[ ]:

	Ind_ID	CHILDREN	Annual_income	EDUCATION	Housing_type	Age	Employment_Year	Mobile_pho
0	5008827	0	180000	3	5	51	38.0	
1	5009744	0	315000	3	5	37	2.0	
2	5009746	0	315000	3	5	37	2.0	
3	5009749	0	315000	3	5	37	2.0	
4	5009752	0	315000	3	5	37	2.0	

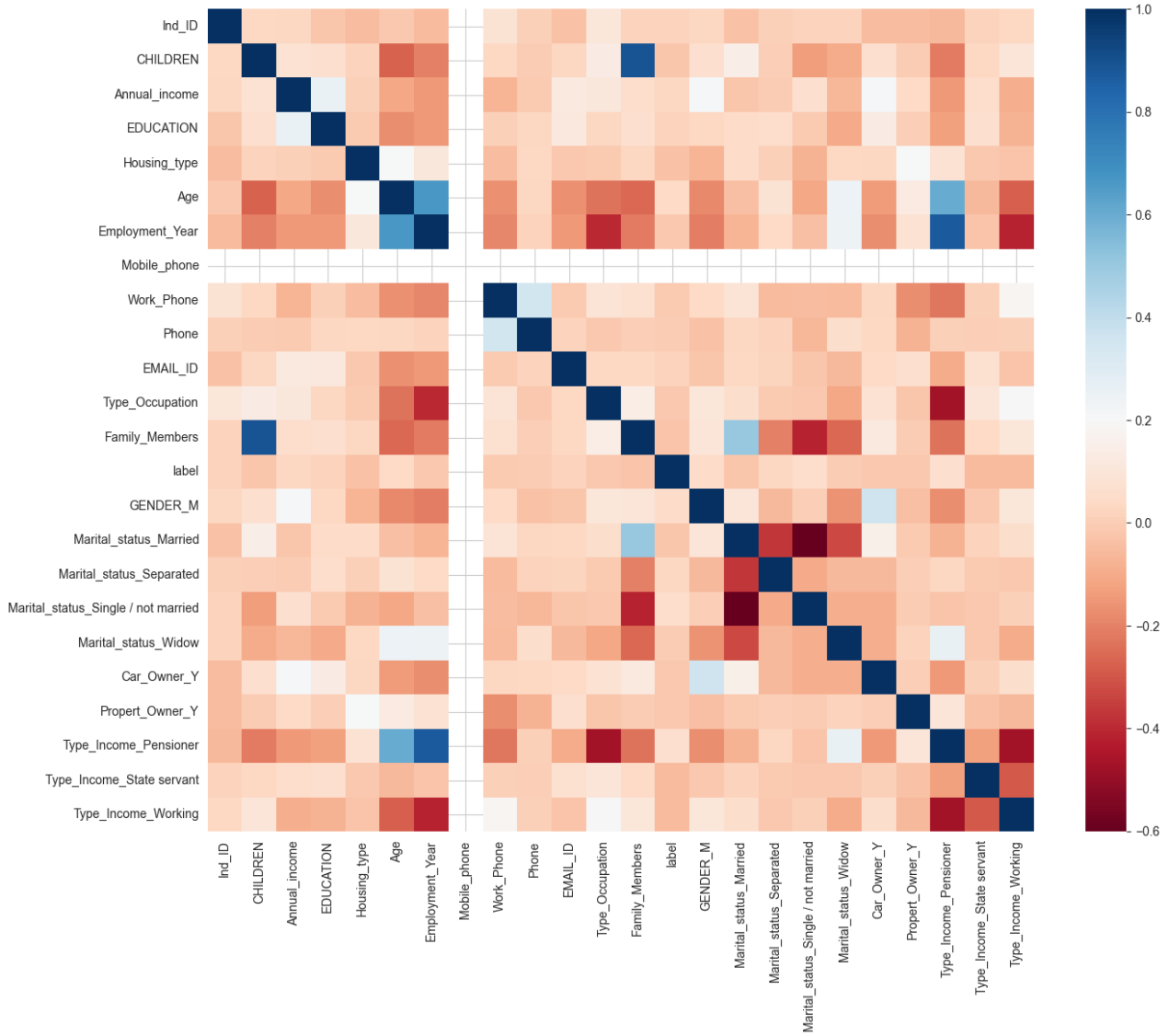
5 rows × 24 columns

In [ ]:
```
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 24 columns):
 #   Column                               Non-Null Count  Dtype  
---  ------                               --------------  -----  
 0   Ind_ID                               1548 non-null   int64  
 1   CHILDREN                             1548 non-null   int64  
 2   Annual_income                        1548 non-null   int64  
 3   EDUCATION                            1548 non-null   int64  
 4   Housing_type                         1548 non-null   int64  
 5   Age                                  1548 non-null   int64  
 6   Employment_Year                      1548 non-null   float64
 7   Mobile_phone                         1548 non-null   int64  
 8   Work_Phone                           1548 non-null   int64  
 9   Phone                                1548 non-null   int64  
 10  EMAIL_ID                             1548 non-null   int64  
 11  Type_Occupation                      1548 non-null   float64
 12  Family_Members                       1548 non-null   int64  
 13  label                                1548 non-null   int64  
 14  GENDER_M                             1548 non-null   int64  
 15  Marital_status_Married               1548 non-null   int64  
 16  Marital_status_Separated             1548 non-null   int64  
 17  Marital_status_Single / not married  1548 non-null   int64  
 18  Marital_status_Widow                 1548 non-null   int64  
 19  Car_Owner_Y                          1548 non-null   int64  
 20  Propert_Owner_Y                      1548 non-null   int64  
 21  Type_Income_Pensioner                1548 non-null   int64  
 22  Type_Income_State servant            1548 non-null   int64  
 23  Type_Income_Working                  1548 non-null   int64  
dtypes: float64(2), int64(22)
memory usage: 290.4 KB
```

In [ ]:
```
# Correlation between each rows.

plt.figure(figsize=(15,12))
sns.heatmap(df4.corr(), annot=False, cmap = "RdBu")
```

Out[ ]:
```
<Axes: >
```



Split the data

In [ ]:
```
X = df4.drop(columns = ['label'], axis = 1)
y = df4['label']
```

Feature Selection with Extra Tree Classifier - Selecting Most important Features

The ExtraTrees classifier, short for Extremely Randomized Trees, works similarly to Random Forests but with a couple of key differences:

1. **Randomization of Split Points**: In ExtraTrees, split points for each feature are selected randomly rather than based on the best split points, as in Random Forests. This randomness helps to further decorrelate the individual trees in the ensemble, reducing variance and potentially improving generalization.

2. **Aggregation of Predictions**: Like Random Forests, ExtraTrees aggregates predictions from multiple decision trees to make the final prediction. Each tree gets a vote, and the majority vote (for classification) or the average (for regression) determines the final prediction.

In summary, ExtraTrees aims to increase the randomness in the construction of individual trees, which can lead to a more diverse ensemble and potentially better performance, especially when dealing with noisy or high-dimensional data.

In [ ]:
```
from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()

model.fit(X, y)
```

Out[ ]:
```
▾ ExtraTreesClassifier
ExtraTreesClassifier()
```

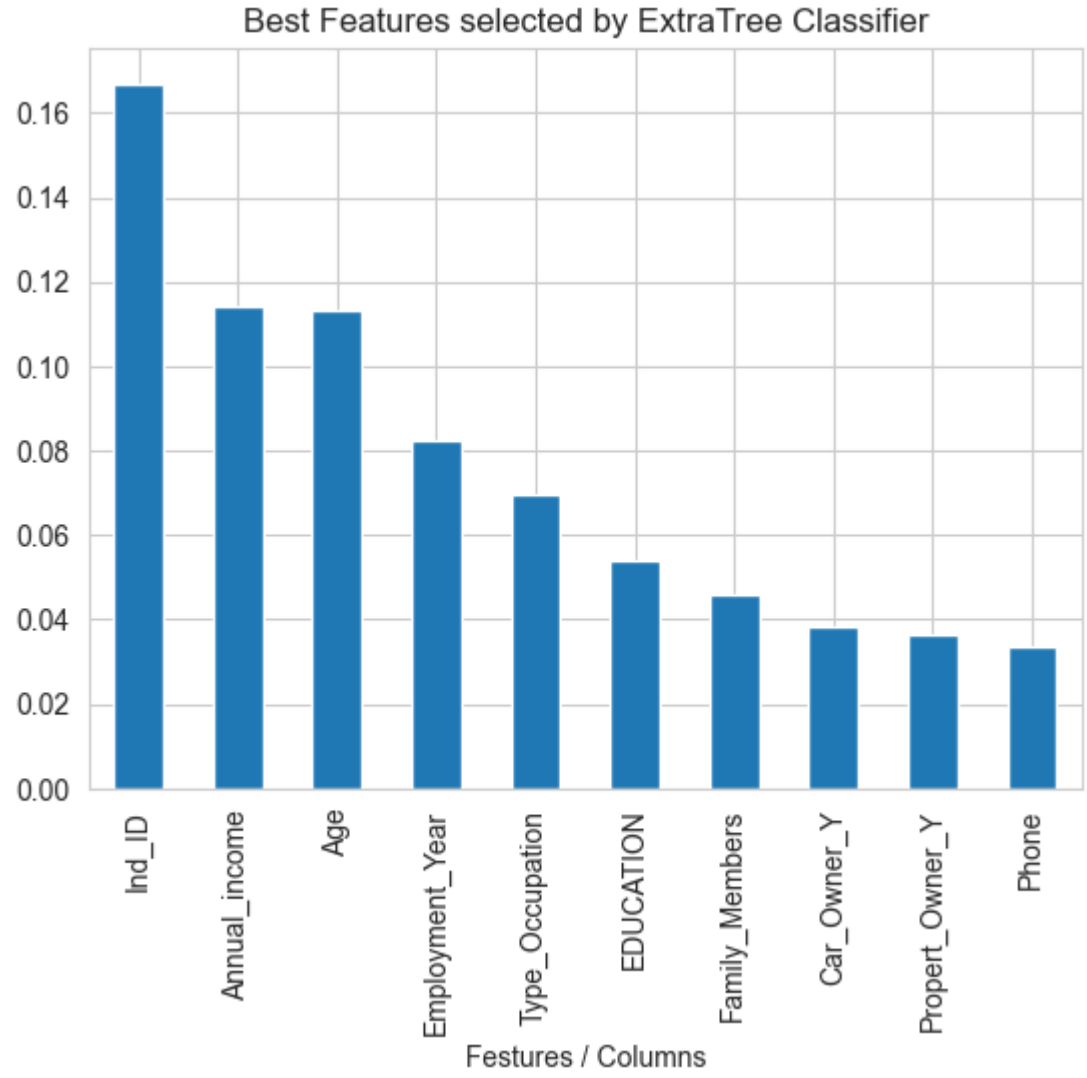
In [ ]:
```
df4.columns
```

Out[ ]:
```
Index(['Ind_ID', 'CHILDREN', 'Annual_income', 'EDUCATION', 'Housing_type',
       'Age', 'Employment_Year', 'Mobile_phone', 'Work_Phone', 'Phone',
       'EMAIL_ID', 'Type_Occupation', 'Family_Members', 'label', 'GENDER_M',
       'Marital_status_Married', 'Marital_status_Separated',
       'Marital_status_Single / not married', 'Marital_status_Widow',
       'Car_Owner_Y', 'Propert_Owner_Y', 'Type_Income_Pensioner',
       'Type_Income_State servant', 'Type_Income_Working'],
      dtype='object')
```

In [ ]:
```
cols=[ 'Ind_ID','CHILDREN', 'Annual_income', 'EDUCATION', 'Housing_type',
       'Age', 'Employment_Year', 'Mobile_phone', 'Work_Phone', 'Phone',
       'EMAIL_ID', 'Type_Occupation', 'Family_Members', 'GENDER_M',
       'Marital_status_Married', 'Marital_status_Separated',
       'Marital_status_Single / not married', 'Marital_status_Widow',
       'Car_Owner_Y', 'Propert_Owner_Y', 'Type_Income_Pensioner',
       'Type_Income_State servant', 'Type_Income_Working']

feature_imp = pd.Series(model.feature_importances_, index=cols)

feature_imp
```

Out[ ]:
```
Ind_ID                                0.166927
CHILDREN                              0.033232
Annual_income                         0.114250
EDUCATION                             0.053744
Housing_type                          0.032530
Age                                   0.113143
Employment_Year                       0.082354
Mobile_phone                          0.000000
Work_Phone                            0.028480
Phone                                 0.033734
EMAIL_ID                              0.022665
Type_Occupation                       0.069769
Family_Members                        0.045638
GENDER_M                              0.030120
Marital_status_Married                0.017009
Marital_status_Separated              0.011922
Marital_status_Single / not married   0.015461
Marital_status_Widow                  0.009308
Car_Owner_Y                           0.038272
Propert_Owner_Y                       0.036558
Type_Income_Pensioner                 0.013756
Type_Income_State servant             0.007451
Type_Income_Working                   0.023679
dtype: float64
```

In [ ]:
```
feature_imp.nlargest(10).plot(kind = 'bar')
plt.title("Best Features selected by ExtraTree Classifier")
plt.xlabel("Festures / Columns")
plt.show()
```



Here "Ind_ID" should not be used ideally, as its just an random uniue number assigned to every individual, but for current scenario , I have continued here for demonstration purpose, and have also tried removing that column for prediction, where prediction rate gets down by 1-1.7 %

Common Standardizarion before prediction

In [ ]:
```
X = df4.drop('label', axis = 1)
y = df4['label']
```

In [ ]:
```
# Extract the top 10 important features
top_features = feature_imp.nlargest(10)
```

In [ ]:
```
# Selecting only the top important features from your original dataset
X_selected = X[top_features.index]
```

In [ ]:
```
X_selected.head(4)
```

Out[ ]:

	Ind_ID	Annual_income	Age	Employment_Year	Type_Occupation	EDUCATION	Family_Members	Ca
0	5008827	180000	51	38.0	0.0	3	2	
1	5009744	315000	37	2.0	1.0	3	2	
2	5009746	315000	37	2.0	1.0	3	2	
3	5009749	315000	37	2.0	1.0	3	2	

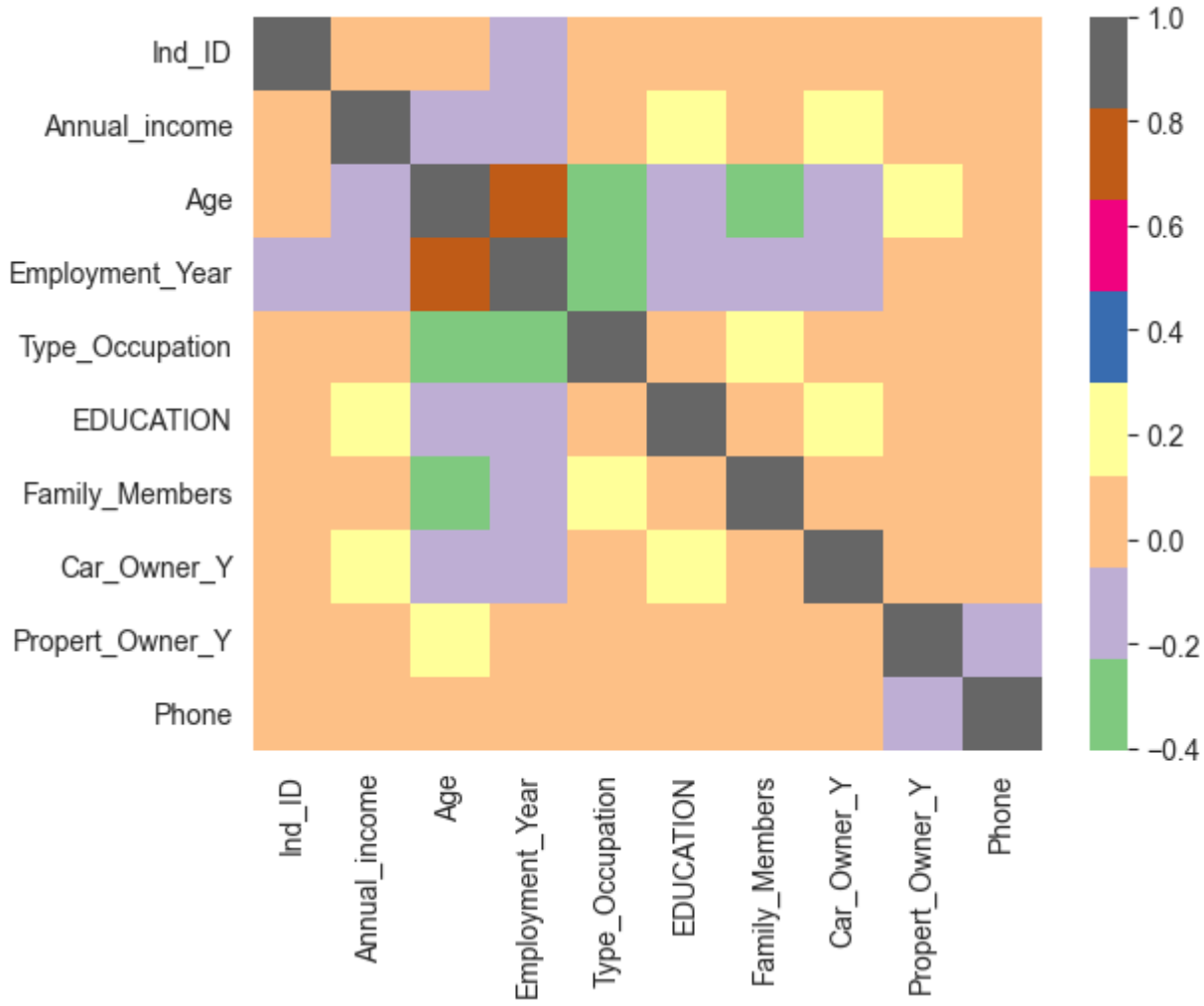
In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

x = pd.DataFrame(MinMaxScaler().fit_transform(X_selected))
```

In [ ]:
```
x.head(3)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9
0	0.000000	0.094891	0.638298	1.000000	0.000000	0.75	0.071429	1.0	1.0	0.0
1	0.006477	0.182482	0.340426	0.052632	0.052632	0.75	0.071429	1.0	0.0	1.0
2	0.006491	0.182482	0.340426	0.052632	0.052632	0.75	0.071429	1.0	0.0	1.0

In [ ]:
```
# Correlation between top selected features by ExtraTree Classifier
sns.heatmap(X_selected.corr(), cmap='Accent')
plt.show()
```



In [ ]:
```
X_selected.shape
```

Out[ ]:
```
(1548, 10)
```

#

H. Machine Learning - Models

1. RandomForest Classifier (Best Fit Model)

In [ ]:
```
X = df4.drop(columns = ['label'], axis = 1)
y = df4['label']

# Extract the top 10 important features
top_features = feature_imp.nlargest(10)

# Selecting only the top important features from original dataset
X_selected = X[top_features.index]

# Performing Standardization
from sklearn.preprocessing import MinMaxScaler

x = pd.DataFrame(MinMaxScaler().fit_transform(X_selected))
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Splitting the dataset into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42
```

In [ ]:
```
x_train.shape
```

Out[ ]:
```
(1238, 10)
```

In [ ]:
```
x_test.shape
```

Out[ ]:
```
(310, 10)
```

In [ ]:
```
y_train.shape
```

Out[ ]:
```
(1238,)
```

In [ ]:
```
y_test.shape
```

Out[ ]:
```
(310,)
```

In [ ]:
```
x_train.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9
680	0.256284	0.109489	0.787234	1.000000	0.000000	0.25	0.000000	0.0	1.0	0.0
1079	0.946103	0.080292	0.361702	0.105263	0.210526	0.25	0.142857	1.0	0.0	0.0
1190	0.740785	0.240876	0.531915	0.026316	0.210526	0.25	0.142857	1.0	1.0	1.0
864	0.425822	0.094891	0.744681	1.000000	0.000000	0.25	0.000000	0.0	1.0	0.0
743	0.767977	0.065693	0.255319	0.052632	0.210526	0.25	0.071429	0.0	0.0	0.0

In [ ]:
```
# Initializing and training your machine learning model (Random Forest Classifier used a
rdf = RandomForestClassifier()
rdf.fit(x_train, y_train)
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [ ]:
```
y_pred = rdf.predict(x_test)
```

In [ ]:
```
y_pred[10:20]
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=int64)
```

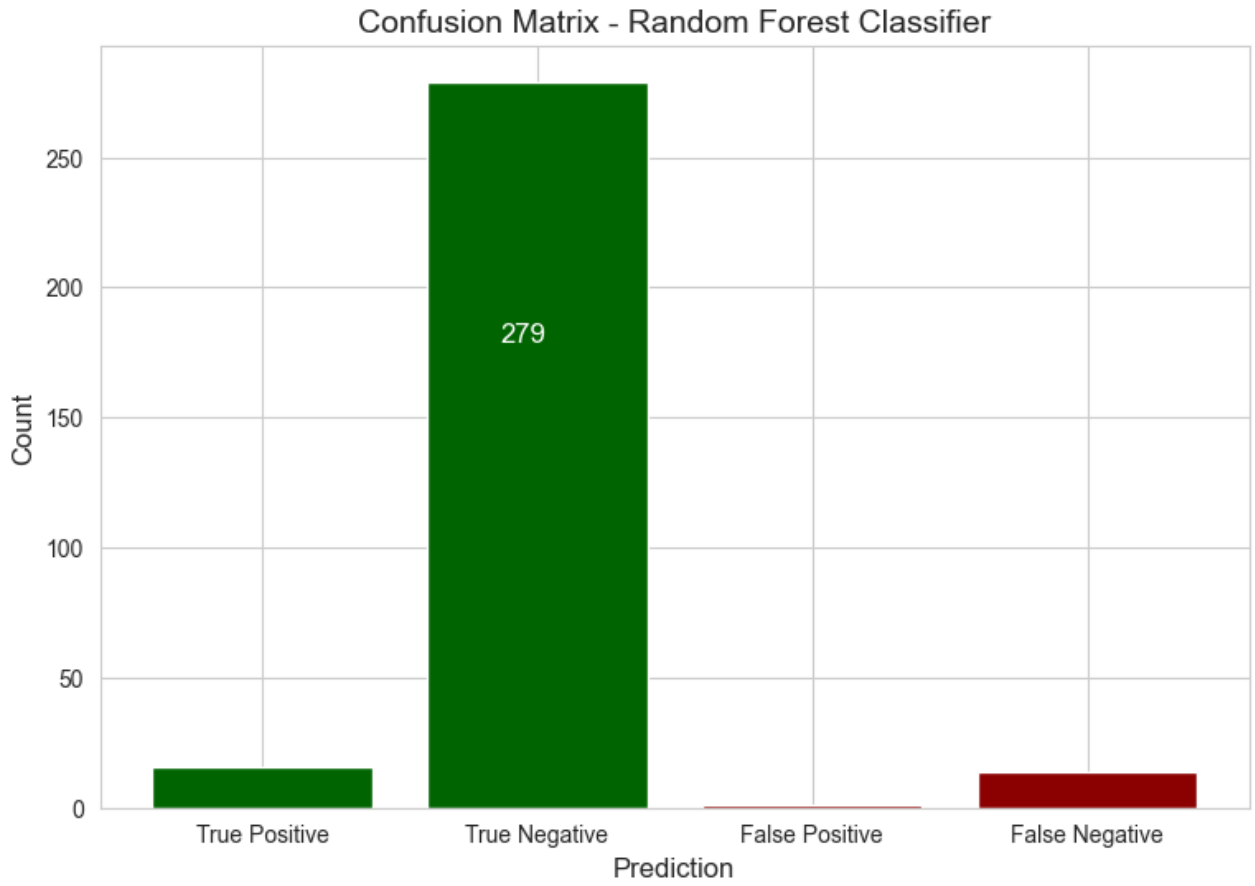
In [ ]:
```
print(list(y_test[10:20]), end = " ")
```

```
[0, 0, 0, 0, 0, 1, 0, 0, 1, 0]
```

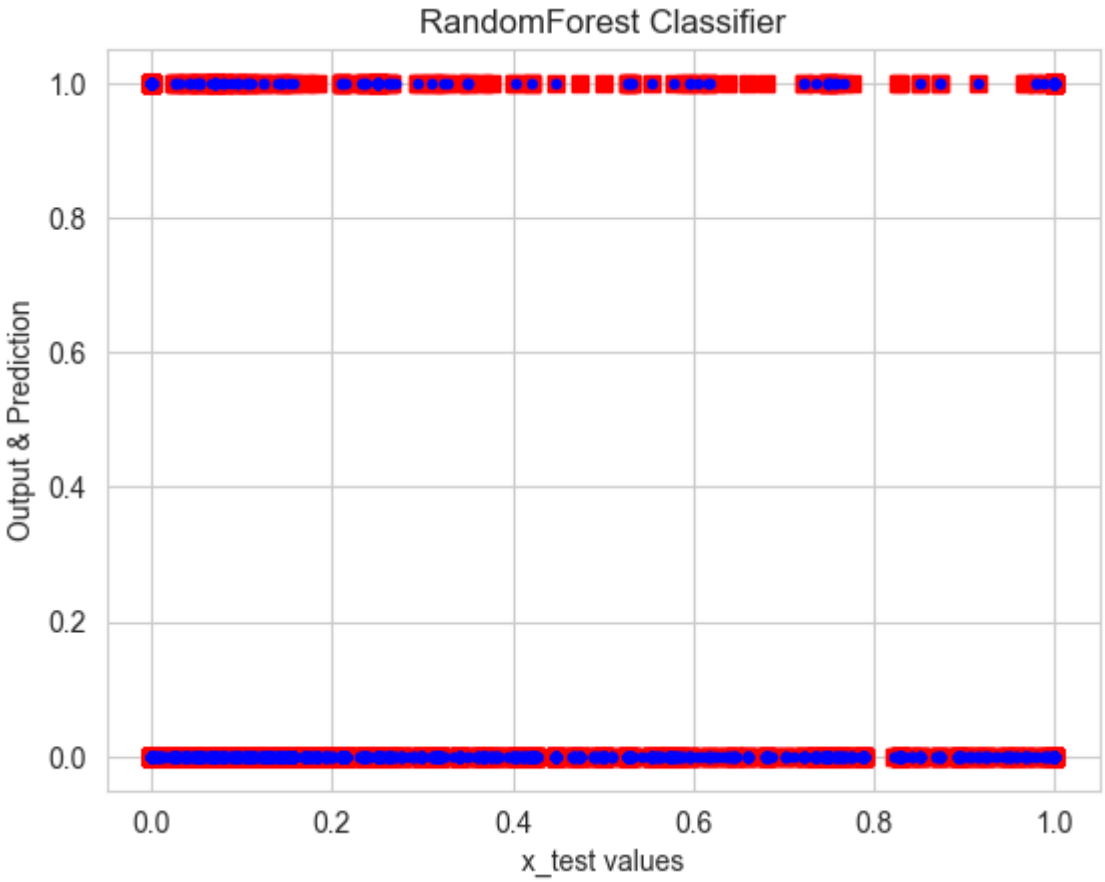
In [ ]:
```
print("Accuracy Score = ",accuracy_score(y_test, y_pred))

rdf_conf_matrix = confusion_matrix(y_test, y_pred)
```

```
Accuracy Score =  0.9516129032258065
```

In [ ]:
```
# Calculate True Positives (TP), True Negatives (TN), False Positives (FP), and False Ne
TP = rdf_conf_matrix[1, 1]
TN = rdf_conf_matrix[0, 0]
FP = rdf_conf_matrix[0, 1]
FN = rdf_conf_matrix[1, 0]

# Plotting the confusion matrix using bar plots
plt.figure(figsize=(9, 6))
bars = plt.bar(x=['True Positive', 'True Negative', 'False Positive', 'False Negative'],

# Adding text labels on top of each bar
for bar in bars:
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.05, bar.get_height() - 100, f'{int(ba

plt.xlabel('Prediction', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Confusion Matrix - Random Forest Classifier', fontsize=14)
plt.show()
```

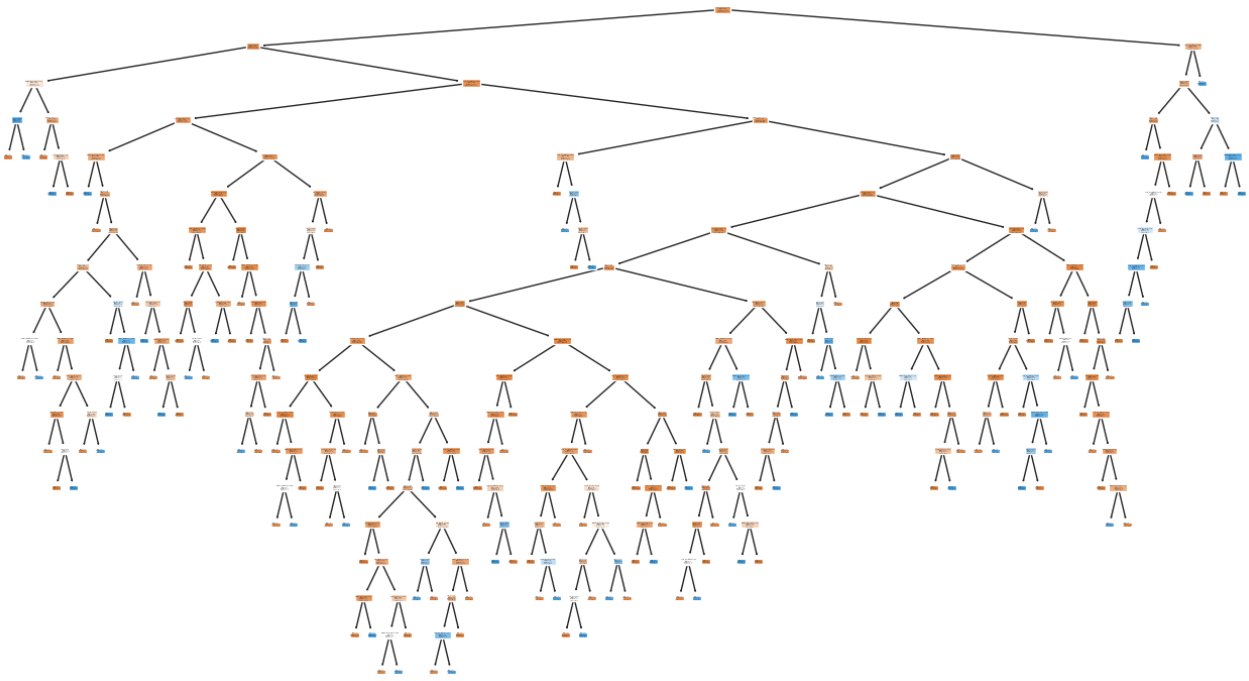


In [ ]:
```
plt.plot(x_test, y_test, "s", color = "red")
plt.plot(x_test, y_pred, ".", color = "blue")
plt.title("RandomForest Classifier")
plt.ylabel("Output & Prediction")
plt.xlabel("x_test values")
plt.show()
```



In [ ]:

```
from sklearn import tree

# Extracting individual trees from the random forest
estimator = rdf.estimators_[0]

# Visualizing the first tree
plt.figure(figsize=(16, 9))
tree.plot_tree(estimator, filled=True, feature_names=X_selected.columns)
plt.show()
```



The above visualizes the tree model, showing the hierarchical structure of decisions made by the classifier based on the features in the dataset. Each node represents a decision based on a feature, and the branches represent the possible outcomes of that decision.

RandomForest Classifier - Cross Validation

In [ ]:
```
from sklearn.model_selection import cross_val_score
cross_val_randomforest = cross_val_score(rdf, X_selected, y, scoring='accuracy', cv = 5)
```

In [ ]:
```
cross_val_randomforest
```

Out[ ]:
```
array([0.88387097, 0.88064516, 0.88064516, 0.8802589 , 0.87702265])
```

2. XGBoost

In [ ]:
```
X = df4.drop('label', axis = 1)
y = df4['label']

# Extract the top 10 important features
top_features = feature_imp.nlargest(10)

# Selecting only the top important features from your original dataset
X_selected = X[top_features.index]

from sklearn.preprocessing import MinMaxScaler

x = pd.DataFrame(MinMaxScaler().fit_transform(X_selected))
```

In [ ]:
```
x.head(2)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9
0	0.000000	0.094891	0.638298	1.000000	0.000000	0.75	0.071429	1.0	1.0	0.0
1	0.006477	0.182482	0.340426	0.052632	0.052632	0.75	0.071429	1.0	0.0	1.0

In [ ]:
```
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

# Splitting the dataset into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42
```

In [ ]:
```
x.head(2)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9
0	0.000000	0.094891	0.638298	1.000000	0.000000	0.75	0.071429	1.0	1.0	0.0
1	0.006477	0.182482	0.340426	0.052632	0.052632	0.75	0.071429	1.0	0.0	1.0

In [ ]:
```
x_train.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9
680	0.256284	0.109489	0.787234	1.000000	0.000000	0.25	0.000000	0.0	1.0	0.0
1079	0.946103	0.080292	0.361702	0.105263	0.210526	0.25	0.142857	1.0	0.0	0.0
1190	0.740785	0.240876	0.531915	0.026316	0.210526	0.25	0.142857	1.0	1.0	1.0
864	0.425822	0.094891	0.744681	1.000000	0.000000	0.25	0.000000	0.0	1.0	0.0
743	0.767977	0.065693	0.255319	0.052632	0.210526	0.25	0.071429	0.0	0.0	0.0

In [ ]:
```
xgb = XGBClassifier()
xgb.fit(x_train, y_train)
```

Out[ ]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
              enable_categorical=False, eval_metric=None, feature_types=Non
e,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=Non
e,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
```

In [ ]:
```
y_pred = xgb.predict(x_test)
```

In [ ]:
```
y_pred[10:20]
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

In [ ]:
```
print(list(y_test[10:20]), end = " ")
```

```
[0, 0, 0, 0, 0, 1, 0, 0, 1, 0]
```

In [ ]:
```
print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

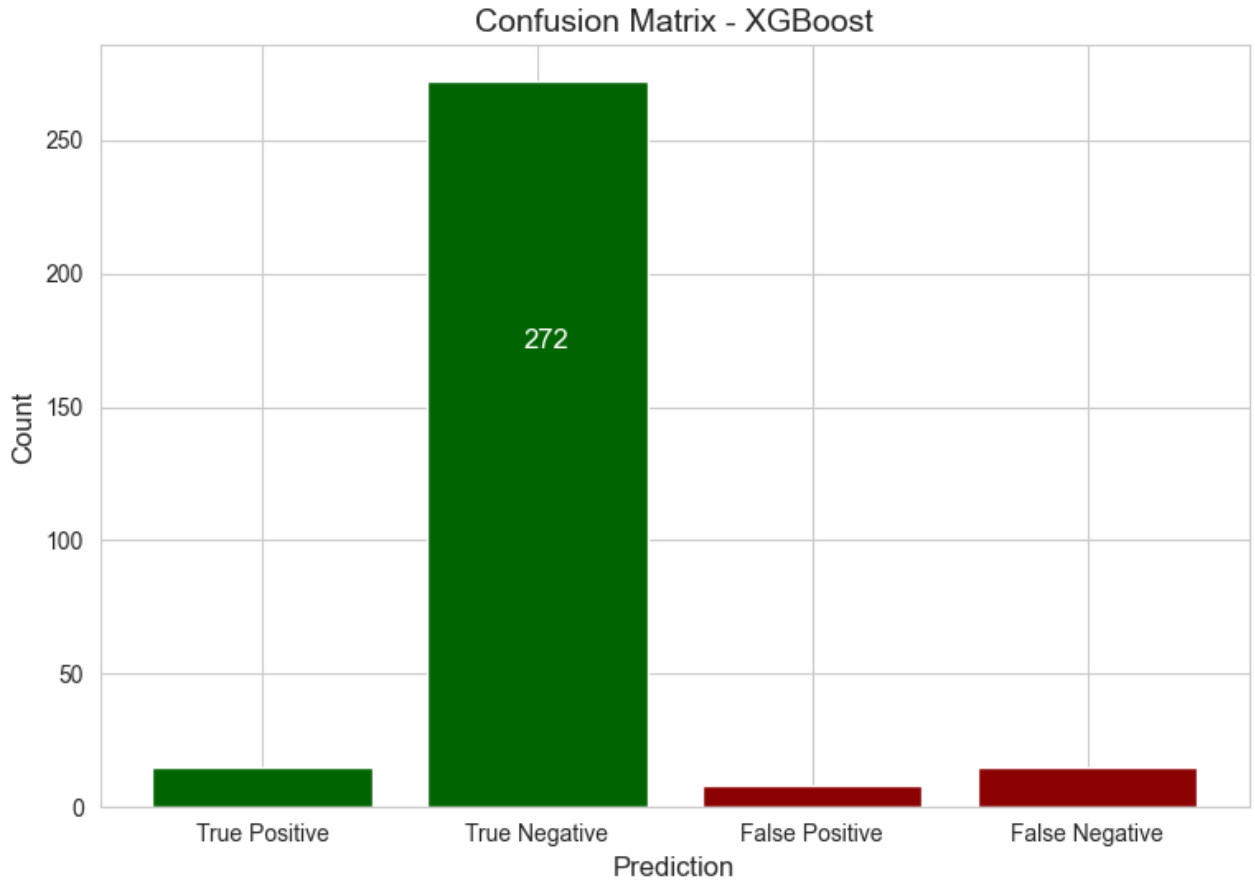
```
Confusion Matrix =
 [[272   8]
 [ 15  15]]
```

In [ ]:
```
print("Accuracy Score = ",accuracy_score(y_test, y_pred))

xgb_conf_matrix = confusion_matrix(y_test, y_pred)
```

```
Accuracy Score =  0.9258064516129032
```

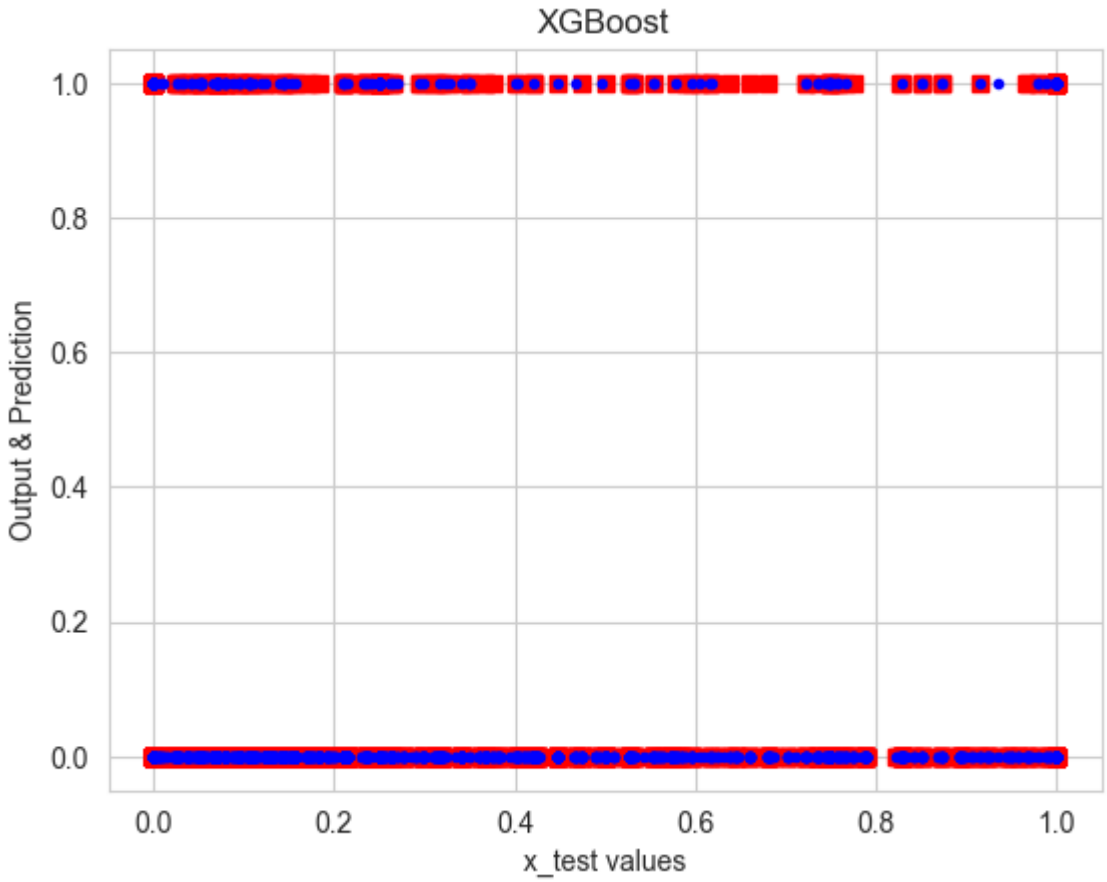
In [ ]:
```
# Calculate True Positives (TP), True Negatives (TN), False Positives (FP), and False Ne
TP = xgb_conf_matrix[1, 1]
TN = xgb_conf_matrix[0, 0]
FP = xgb_conf_matrix[0, 1]
FN = xgb_conf_matrix[1, 0]

# Plotting the confusion matrix using bar plots
plt.figure(figsize=(9, 6))
bars = plt.bar(x=['True Positive', 'True Negative', 'False Positive', 'False Negative'],

# Adding text labels on top of each bar
```

```
for bar in bars:
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.05, bar.get_height() - 100, f'{int(ba

plt.xlabel('Prediction', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Confusion Matrix - XGBoost', fontsize=14)
plt.show()
```



In [ ]:

```
plt.plot(x_test, y_test, "s", color = "red")
plt.plot(x_test, y_pred, ".", color = "blue")
plt.title("XGBoost")
plt.ylabel("Output & Prediction")
plt.xlabel("x_test values")
plt.show()
```



XGBoost Cross-Validation

In [ ]:
```
from sklearn.model_selection import cross_val_score
cross_val_xgb = cross_val_score(xgb, x, y, scoring='accuracy', cv = 5)
```

In [ ]:
```
cross_val_xgb
```

Out[ ]:
```
array([0.86451613, 0.87096774, 0.88064516, 0.87378641, 0.86731392])
```

3. Logistic Regression

In [ ]:
```
X = df4.drop('label', axis = 1)
y = df4['label']

# Extract the top 10 important features
top_features = feature_imp.nlargest(10)

# Selecting only the top important features from your original dataset
X_selected = X[top_features.index]

from sklearn.preprocessing import MinMaxScaler

x = pd.DataFrame(MinMaxScaler().fit_transform(X_selected))
```

In [ ]:
```
from sklearn.model_selection import train_test_split

# Splitting the dataset into train and test sets
x_train, x_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
log_r = LogisticRegression()
```

```
log_r.fit(x_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression()
```

In [ ]:
```
x_train.shape
```

Out[ ]:
```
(1238, 10)
```

In [ ]:
```
y_train.shape
```

Out[ ]:
```
(1238,)
```

In [ ]:
```
y_pred = log_r.predict(x_test)
```

In [ ]:
```
y_pred[:12]
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

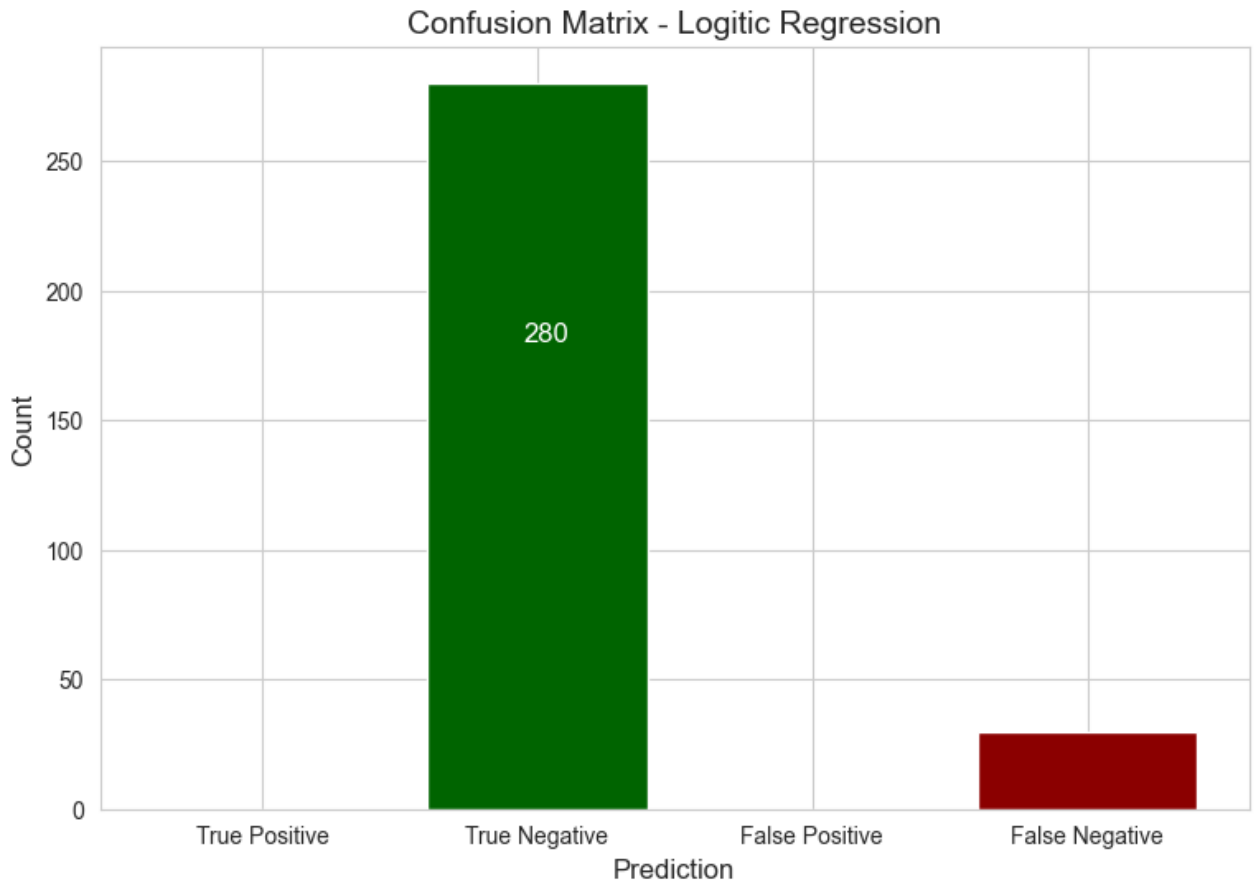
In [ ]:
```
print(list(y_test[:10]), end = " ")
```

```
[1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
```

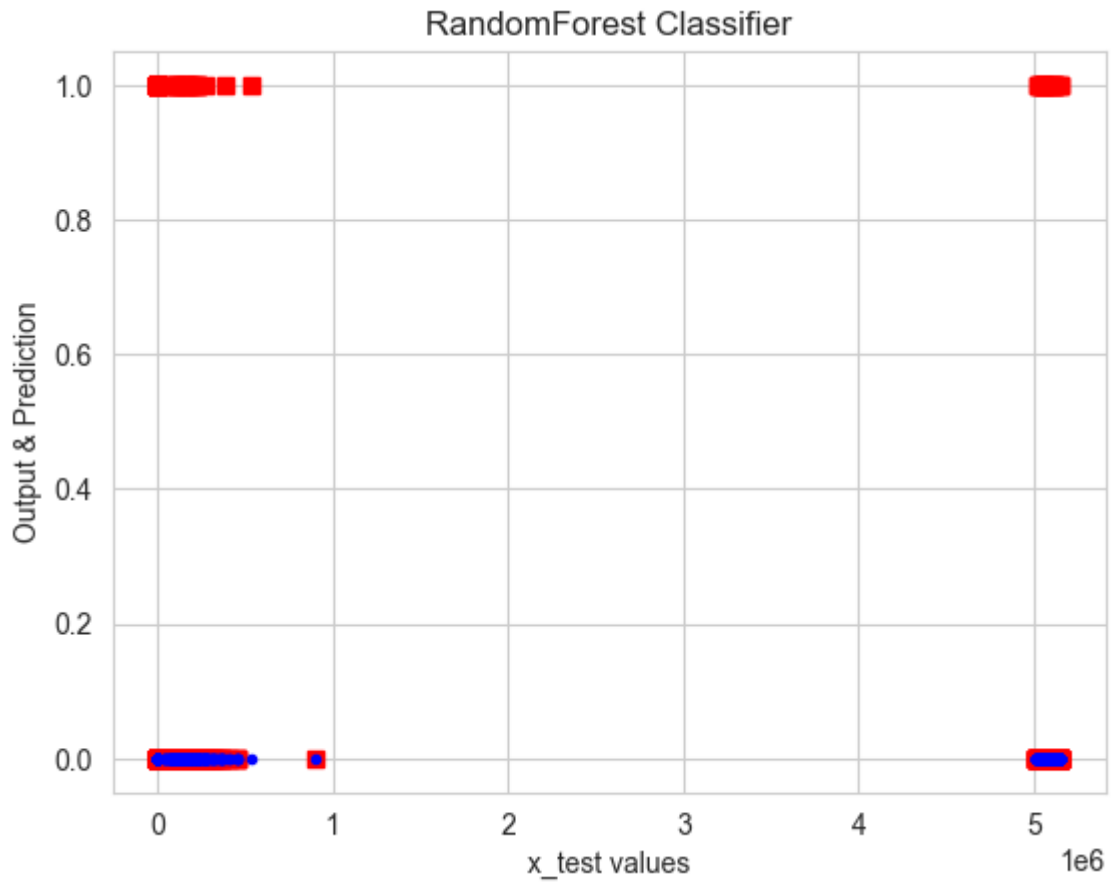
In [ ]:
```
print("Accuracy Score = ",accuracy_score(y_test, y_pred))

logr_conf_matrix = confusion_matrix(y_test, y_pred)

print(logr_conf_matrix)
```

```
Accuracy Score =  0.9032258064516129
[[280   0]
 [ 30   0]]
```

In [ ]:
```
# Calculate True Positives (TP), True Negatives (TN), False Positives (FP), and False Ne
TP = logr_conf_matrix[1, 1]
TN = logr_conf_matrix[0, 0]
FP = logr_conf_matrix[0, 1]
FN = logr_conf_matrix[1, 0]

# Plotting the confusion matrix using bar plots
plt.figure(figsize=(9, 6))
bars = plt.bar(x=['True Positive', 'True Negative', 'False Positive', 'False Negative'],

# Adding text labels on top of each bar
for bar in bars:
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.05, bar.get_height() - 100, f'{int(ba

plt.xlabel('Prediction', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Confusion Matrix - Logitic Regression', fontsize=14)
plt.show()
```



In [ ]:
```
plt.plot(x_test, y_test, "s", color = "red")
plt.plot(x_test, y_pred, ".", color = "blue")
plt.title("RandomForest Classifier")
plt.ylabel("Output & Prediction")
plt.xlabel("x_test values")
plt.show()
```



Logistic Regression

In [ ]:
```
from sklearn.model_selection import cross_val_score
cross_val_log_r = cross_val_score(log_r, X_selected, y, scoring='accuracy', cv = 5)
```

In [ ]:
```
cross_val_log_r
```

Out[ ]:
```
array([0.88709677, 0.88709677, 0.88709677, 0.88673139, 0.88673139])
```

4. Support vector Machines

In [ ]:
```
X = df4.drop('label', axis = 1)
y = df4['label']

# Extract the top 10 important features
top_features = feature_imp.nlargest(10)

# Selecting only the top important features from your original dataset
X_selected = X[top_features.index]

from sklearn.preprocessing import MinMaxScaler

x = pd.DataFrame(MinMaxScaler().fit_transform(X_selected))
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Splitting the dataset into train and test sets
x_train, x_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random
```

In [ ]:
```
# Initializing and training your machine learning model (Random Forest Classifier used a
svc = SVC()
svc.fit(x_train, y_train)
```

Out[ ]:
```
▾ SVC

SVC()
```

In [ ]:
```
y_pred = svc.predict(x_test)
```

In [ ]:
```
y_pred[10:20]
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [ ]:
```
print(list(y_test[10:20]), end = " ")
```

```
[0, 0, 0, 0, 0, 1, 0, 0, 1, 0]
```

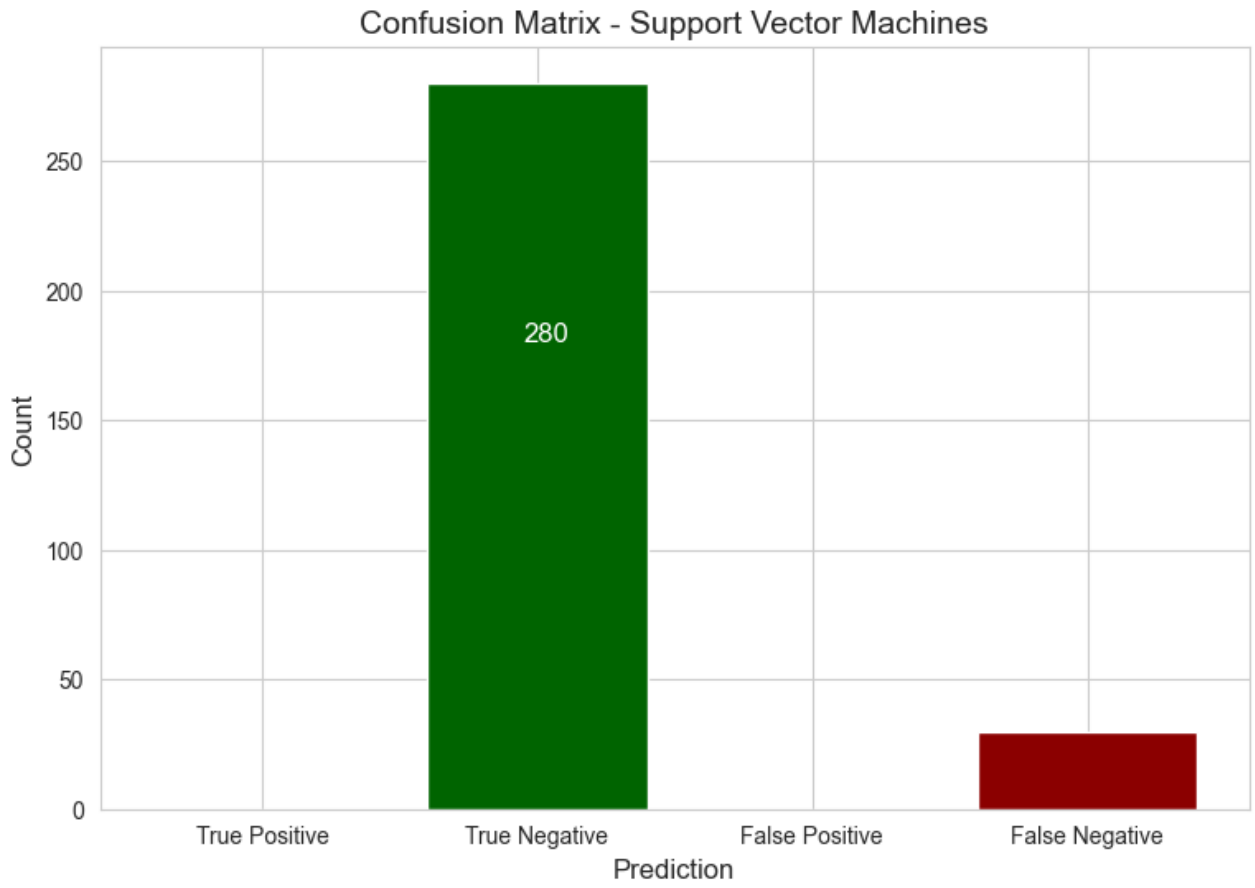
In [ ]:
```
print("Confusion Matrix = \n",confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix =
 [[280   0]
 [ 30   0]]
```

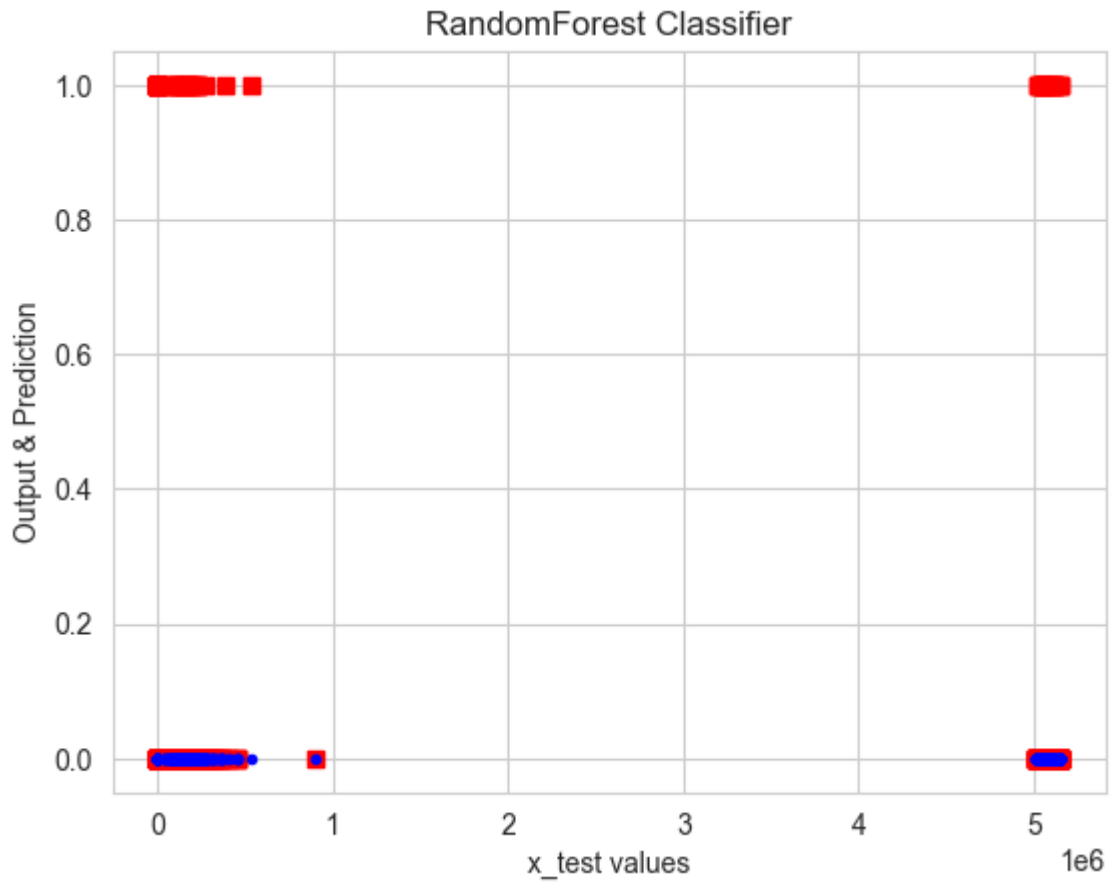
In [ ]:
```
print("Accuracy Score = ",accuracy_score(y_test, y_pred))

svm_conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix : \n",svm_conf_matrix)
```

```
Accuracy Score =  0.9032258064516129
Confusion Matrix :
 [[280   0]
 [ 30   0]]
```

In [ ]:
```
# Calculate True Positives (TP), True Negatives (TN), False Positives (FP), and False Ne
TP = svm_conf_matrix[1, 1]
TN = svm_conf_matrix[0, 0]
FP = svm_conf_matrix[0, 1]
FN = svm_conf_matrix[1, 0]

# Plotting the confusion matrix using bar plots
plt.figure(figsize=(9, 6))
bars = plt.bar(x=['True Positive', 'True Negative', 'False Positive', 'False Negative'],

# Adding text labels on top of each bar
for bar in bars:
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.05, bar.get_height() - 100, f'{int(ba

plt.xlabel('Prediction', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Confusion Matrix - Support Vector Machines', fontsize=14)
plt.show()
```



In [ ]:
```
plt.plot(x_test, y_test, "s", color = "red")
plt.plot(x_test, y_pred, ".", color = "blue")
plt.title("RandomForest Classifier")
plt.ylabel("Output & Prediction")
plt.xlabel("x_test values")
plt.show()
```



5. Decision Tree Classifier

In [ ]:
```
X = df4.drop('label', axis = 1)
y = df4['label']

# Extract the top 10 important features
top_features = feature_imp.nlargest(10)

# Selecting only the top important features from your original dataset
X_selected = X[top_features.index]

from sklearn.preprocessing import MinMaxScaler

x = pd.DataFrame(MinMaxScaler().fit_transform(X_selected))
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Splitting the dataset into train and test sets
x_train, x_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random
```

In [ ]:
```
# Initializing and training your machine learning model (Random Forest Classifier used a
dct = DecisionTreeClassifier()
dct.fit(x_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [ ]:
```
y_pred = dct.predict(x_test)
```

In [ ]:
```
y_pred[10:20]
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=int64)
```

In [ ]:
```
print(list(y_test[10:20]), end = " ")
```

```
[0, 0, 0, 0, 0, 1, 0, 0, 1, 0]
```

In [ ]:
```
from sklearn import tree

reg = DecisionTreeClassifier(random_state=42)
reg.fit(x_train,y_train)
plt.figure(figsize=(16,9))
tree.plot_tree(reg,filled=True,feature_names=X_selected.columns)
```

Out[ ]:
```
[Text(0.45742126423944607, 0.9782608695652174, 'Employment_Year <= 3.5\ngini = 0.207\nsa
mples = 1238\nvalue = [1093, 145]'),
 Text(0.14465601965601965, 0.9347826086956522, 'Ind_ID <= 5021343.5\ngini = 0.293\nsampl
es = 336\nvalue = [276, 60]'),
 Text(0.0285905740451195, 0.8913043478260869, 'Annual_income <= 236250.0\ngini = 0.476\n
samples = 23\nvalue = [9, 14]'),
 Text(0.01429528702255975, 0.8478260869565217, 'Ind_ID <= 5014746.5\ngini = 0.397\nsampl
es = 11\nvalue = [8, 3]'),
 Text(0.007147643511279875, 0.8043478260869565, 'gini = 0.0\nsamples = 8\nvalue = [8,
0]'),
 Text(0.021442930533839624, 0.8043478260869565, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.04288586106767925, 0.8478260869565217, 'Age <= 48.0\ngini = 0.153\nsamples = 12
\nvalue = [1, 11]'),
 Text(0.035738217556399374, 0.8043478260869565, 'gini = 0.0\nsamples = 8\nvalue = [0,
8]'),
 Text(0.05003350457895912, 0.8043478260869565, 'Ind_ID <= 5010866.5\ngini = 0.375\nsampl
es = 4\nvalue = [1, 3]'),
 Text(0.04288586106767925, 0.7608695652173914, 'Ind_ID <= 5010864.5\ngini = 0.5\nsamples
= 2\nvalue = [1, 1]'),
 Text(0.035738217556399374, 0.717391304347826, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.05003350457895912, 0.717391304347826, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.057181148090239, 0.7608695652173914, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.2607214652669198, 0.8913043478260869, 'Ind_ID <= 5134064.0\ngini = 0.251\nsample
s = 313\nvalue = [267, 46]'),
 Text(0.19265132901496537, 0.8478260869565217, 'Annual_income <= 389250.0\ngini = 0.194
\nsamples = 267\nvalue = [238, 29]'),
 Text(0.1136922046012955, 0.8043478260869565, 'Type_Occupation <= 3.5\ngini = 0.172\nsam
ples = 252\nvalue = [228, 24]'),
 Text(0.07147643511279875, 0.7608695652173914, 'Age <= 54.5\ngini = 0.024\nsamples = 82
\nvalue = [81, 1]'),
 Text(0.06432879160151887, 0.717391304347826, 'gini = 0.0\nsamples = 74\nvalue = [74,
0]'),
 Text(0.07862407862407862, 0.717391304347826, 'Family_Members <= 1.5\ngini = 0.219\nsamp
les = 8\nvalue = [7, 1]'),
 Text(0.07147643511279875, 0.6739130434782609, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.0857717221353585, 0.6739130434782609, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.15590797408979226, 0.7608695652173914, 'EDUCATION <= 0.5\ngini = 0.234\nsamples
= 170\nvalue = [147, 23]'),
 Text(0.10721465266919812, 0.717391304347826, 'Ind_ID <= 5125709.0\ngini = 0.444\nsample
s = 3\nvalue = [1, 2]'),
 Text(0.10006700915791825, 0.6739130434782609, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.114362296180478, 0.6739130434782609, 'Ind_ID <= 5125713.0\ngini = 0.5\nsamples =
2\nvalue = [1, 1]'),
 Text(0.10721465266919812, 0.6304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.12150993969175787, 0.6304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.20460129551038642, 0.717391304347826, 'Phone <= 0.5\ngini = 0.22\nsamples = 167
\nvalue = [146, 21]'),
 Text(0.15903506812597723, 0.6739130434782609, 'Annual_income <= 123750.0\ngini = 0.267
\nsamples = 126\nvalue = [106, 20]'),
 Text(0.13580522671431763, 0.6304347826086957, 'Type_Occupation <= 11.5\ngini = 0.102\ns
amples = 37\nvalue = [35, 2]'),
 Text(0.12865758320303775, 0.5869565217391305, 'gini = 0.0\nsamples = 30\nvalue = [30,
0]'),
 Text(0.1429528702255975, 0.5869565217391305, 'Type_Occupation <= 13.0\ngini = 0.408\nsa
mples = 7\nvalue = [5, 2]'),
 Text(0.13580522671431763, 0.5434782608695652, 'gini = 0.0\nsamples = 2\nvalue = [0,
```

```
2]'),
 Text(0.15010051373687738, 0.5434782608695652, 'gini = 0.0\nsamples = 5\nvalue = [5,
0]'),
 Text(0.1822649095376368, 0.6304347826086957, 'Annual_income <= 130500.0\ngini = 0.323\n
samples = 89\nvalue = [71, 18]'),
 Text(0.17511726602635694, 0.5869565217391305, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.1894125530489167, 0.5869565217391305, 'Type_Occupation <= 4.5\ngini = 0.3\nsampl
es = 87\nvalue = [71, 16]'),
 Text(0.16439580075943713, 0.5434782608695652, 'Propert_Owner_Y <= 0.5\ngini = 0.458\nsa
mples = 31\nvalue = [20, 11]'),
 Text(0.13580522671431763, 0.5, 'Ind_ID <= 5052257.5\ngini = 0.42\nsamples = 10\nvalue =
[3, 7]'),
 Text(0.12865758320303775, 0.45652173913043476, 'gini = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.1429528702255975, 0.45652173913043476, 'Age <= 34.0\ngini = 0.5\nsamples = 6\nva
lue = [3, 3]'),
 Text(0.13580522671431763, 0.41304347826086957, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.15010051373687738, 0.41304347826086957, 'Annual_income <= 213750.0\ngini = 0.375
\nsamples = 4\nvalue = [1, 3]'),
 Text(0.1429528702255975, 0.3695652173913043, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.15724815724815724, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.19298637480455663, 0.5, 'Employment_Year <= 2.5\ngini = 0.308\nsamples = 21\nval
ue = [17, 4]'),
 Text(0.18583873129327674, 0.45652173913043476, 'Ind_ID <= 5098636.0\ngini = 0.408\nsamp
les = 14\nvalue = [10, 4]'),
 Text(0.17869108778199688, 0.41304347826086957, 'Ind_ID <= 5079368.5\ngini = 0.5\nsample
s = 8\nvalue = [4, 4]'),
 Text(0.171543444270717, 0.3695652173913043, 'Age <= 39.0\ngini = 0.32\nsamples = 5\nval
ue = [4, 1]'),
 Text(0.16439580075943713, 0.32608695652173914, 'gini = 0.0\nsamples = 4\nvalue = [4,
0]'),
 Text(0.17869108778199688, 0.32608695652173914, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.18583873129327674, 0.3695652173913043, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.19298637480455663, 0.41304347826086957, 'gini = 0.0\nsamples = 6\nvalue = [6,
0]'),
 Text(0.2001340183158365, 0.45652173913043476, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.21442930533839624, 0.5434782608695652, 'Age <= 43.0\ngini = 0.163\nsamples = 56
\nvalue = [51, 5]'),
 Text(0.20728166182711638, 0.5, 'gini = 0.0\nsamples = 43\nvalue = [43, 0]'),
 Text(0.22157694884967613, 0.5, 'Annual_income <= 166500.0\ngini = 0.473\nsamples = 13\n
value = [8, 5]'),
 Text(0.21442930533839624, 0.45652173913043476, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.228724592360956, 0.45652173913043476, 'EDUCATION <= 2.5\ngini = 0.397\nsamples =
11\nvalue = [8, 3]'),
 Text(0.21442930533839624, 0.41304347826086957, 'Type_Occupation <= 12.0\ngini = 0.219\n
samples = 8\nvalue = [7, 1]'),
 Text(0.20728166182711638, 0.3695652173913043, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.22157694884967613, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.24301987938351574, 0.41304347826086957, 'Age <= 59.5\ngini = 0.444\nsamples = 3
\nvalue = [1, 2]'),
 Text(0.23587223587223588, 0.3695652173913043, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.2501675228947956, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
```

```
 Text(0.2501675228947956, 0.6739130434782609, 'Employment_Year <= 0.5\ngini = 0.048\nsam
ples = 41\nvalue = [40, 1]'),
 Text(0.24301987938351574, 0.6304347826086957, 'Ind_ID <= 5068389.5\ngini = 0.32\nsample
s = 5\nvalue = [4, 1]'),
 Text(0.23587223587223588, 0.5869565217391305, 'Annual_income <= 180000.0\ngini = 0.5\ns
amples = 2\nvalue = [1, 1]'),
 Text(0.228724592360956, 0.5434782608695652, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.24301987938351574, 0.5434782608695652, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.2501675228947956, 0.5869565217391305, 'gini = 0.0\nsamples = 3\nvalue = [3,
0]'),
 Text(0.2573151664060755, 0.6304347826086957, 'gini = 0.0\nsamples = 36\nvalue = [36,
0]'),
 Text(0.27161045342863527, 0.8043478260869565, 'Car_Owner_Y <= 0.5\ngini = 0.444\nsample
s = 15\nvalue = [10, 5]'),
 Text(0.2644628099173554, 0.7608695652173914, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.2787580969399151, 0.7608695652173914, 'Type_Occupation <= 2.5\ngini = 0.278\nsam
ples = 12\nvalue = [10, 2]'),
 Text(0.27161045342863527, 0.717391304347826, 'Age <= 41.0\ngini = 0.5\nsamples = 4\nval
ue = [2, 2]'),
 Text(0.2644628099173554, 0.6739130434782609, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.2787580969399151, 0.6739130434782609, 'Age <= 43.5\ngini = 0.444\nsamples = 3\nv
alue = [1, 2]'),
 Text(0.27161045342863527, 0.6304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.285905740451195, 0.6304347826086957, 'Ind_ID <= 5041521.0\ngini = 0.5\nsamples =
2\nvalue = [1, 1]'),
 Text(0.2787580969399151, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.2930533839624749, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.285905740451195, 0.717391304347826, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.32879160151887427, 0.8478260869565217, 'Age <= 51.0\ngini = 0.466\nsamples = 46
\nvalue = [29, 17]'),
 Text(0.3144963144963145, 0.8043478260869565, 'Annual_income <= 405000.0\ngini = 0.388\n
samples = 38\nvalue = [28, 10]'),
 Text(0.3073486709850346, 0.7608695652173914, 'Annual_income <= 146250.0\ngini = 0.257\n
samples = 33\nvalue = [28, 5]'),
 Text(0.30020102747375477, 0.717391304347826, 'Employment_Year <= 2.5\ngini = 0.486\nsam
ples = 12\nvalue = [7, 5]'),
 Text(0.2930533839624749, 0.6739130434782609, 'gini = 0.0\nsamples = 5\nvalue = [5,
0]'),
 Text(0.3073486709850346, 0.6739130434782609, 'Ind_ID <= 5146445.5\ngini = 0.408\nsample
s = 7\nvalue = [2, 5]'),
 Text(0.30020102747375477, 0.6304347826086957, 'gini = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.3144963144963145, 0.6304347826086957, 'Age <= 44.5\ngini = 0.444\nsamples = 3\nv
alue = [2, 1]'),
 Text(0.3073486709850346, 0.5869565217391305, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.3216439580075944, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.3144963144963145, 0.717391304347826, 'gini = 0.0\nsamples = 21\nvalue = [21,
0]'),
 Text(0.3216439580075944, 0.7608695652173914, 'gini = 0.0\nsamples = 5\nvalue = [0,
5]'),
 Text(0.343086888541434, 0.8043478260869565, 'Employment_Year <= 0.5\ngini = 0.219\nsamp
les = 8\nvalue = [1, 7]'),
 Text(0.3359392450301541, 0.7608695652173914, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.3502345320527139, 0.7608695652173914, 'gini = 0.0\nsamples = 7\nvalue = [0,
7]'),
```

```
 Text(0.7701865088228724, 0.9347826086956522, 'Ind_ID <= 5008846.0\ngini = 0.171\nsample
s = 902\nvalue = [817, 85]'),
 Text(0.7630388653115926, 0.8913043478260869, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.7773341523341524, 0.8913043478260869, 'Ind_ID <= 5113927.5\ngini = 0.169\nsample
s = 901\nvalue = [817, 84]'),
 Text(0.6600960464596828, 0.8478260869565217, 'Ind_ID <= 5079174.5\ngini = 0.14\nsamples
= 661\nvalue = [611, 50]'),
 Text(0.5518204154567791, 0.8043478260869565, 'Ind_ID <= 5079153.5\ngini = 0.171\nsample
s = 455\nvalue = [412, 43]'),
 Text(0.5446727719454992, 0.7608695652173914, 'Ind_ID <= 5069207.5\ngini = 0.161\nsample
s = 452\nvalue = [412, 40]'),
 Text(0.4121063211972303, 0.717391304347826, 'Annual_income <= 78750.0\ngini = 0.144\nsa
mples = 434\nvalue = [400, 34]'),
 Text(0.35738217556399376, 0.6739130434782609, 'Ind_ID <= 5061247.5\ngini = 0.375\nsampl
es = 20\nvalue = [15, 5]'),
 Text(0.343086888541434, 0.6304347826086957, 'Car_Owner_Y <= 0.5\ngini = 0.219\nsamples
= 16\nvalue = [14, 2]'),
 Text(0.3359392450301541, 0.5869565217391305, 'Employment_Year <= 13.0\ngini = 0.124\nsa
mples = 15\nvalue = [14, 1]'),
 Text(0.32879160151887427, 0.5434782608695652, 'Ind_ID <= 5038285.5\ngini = 0.5\nsamples
= 2\nvalue = [1, 1]'),
 Text(0.3216439580075944, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3359392450301541, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.343086888541434, 0.5434782608695652, 'gini = 0.0\nsamples = 13\nvalue = [13,
0]'),
 Text(0.3502345320527139, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.3716774625865535, 0.6304347826086957, 'Annual_income <= 64125.0\ngini = 0.375\ns
amples = 4\nvalue = [1, 3]'),
 Text(0.3645298190752736, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.3788251060978334, 0.5869565217391305, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.4668304668304668, 0.6739130434782609, 'Age <= 41.5\ngini = 0.13\nsamples = 414\n
value = [385, 29]'),
 Text(0.400268036631673, 0.6304347826086957, 'Phone <= 0.5\ngini = 0.074\nsamples = 155
\nvalue = [149, 6]'),
 Text(0.3931203931203931, 0.5869565217391305, 'Ind_ID <= 5022686.5\ngini = 0.11\nsamples
= 103\nvalue = [97, 6]'),
 Text(0.36989055170873353, 0.5434782608695652, 'Ind_ID <= 5022047.0\ngini = 0.245\nsampl
es = 14\nvalue = [12, 2]'),
 Text(0.36274290819745364, 0.5, 'gini = 0.0\nsamples = 12\nvalue = [12, 0]'),
 Text(0.3770381952200134, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.4163502345320527, 0.5434782608695652, 'Ind_ID <= 5064133.5\ngini = 0.086\nsample
s = 89\nvalue = [85, 4]'),
 Text(0.39133348224257314, 0.5, 'Employment_Year <= 8.5\ngini = 0.052\nsamples = 75\nval
ue = [73, 2]'),
 Text(0.38418583873129325, 0.45652173913043476, 'gini = 0.0\nsamples = 47\nvalue = [47,
0]'),
 Text(0.398481125753853, 0.45652173913043476, 'Age <= 30.5\ngini = 0.133\nsamples = 28\n
value = [26, 2]'),
 Text(0.38418583873129325, 0.41304347826086957, 'Propert_Owner_Y <= 0.5\ngini = 0.444\ns
amples = 3\nvalue = [2, 1]'),
 Text(0.3770381952200134, 0.3695652173913043, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.39133348224257314, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.41277641277641275, 0.41304347826086957, 'Type_Occupation <= 1.5\ngini = 0.077\ns
amples = 25\nvalue = [24, 1]'),
 Text(0.4056287692651329, 0.3695652173913043, 'Ind_ID <= 5035239.0\ngini = 0.245\nsample
s = 7\nvalue = [6, 1]'),
 Text(0.398481125753853, 0.32608695652173914, 'Employment_Year <= 9.5\ngini = 0.5\nsampl
es = 2\nvalue = [1, 1]'),
```

```
 Text(0.39133348224257314, 0.2826086956521739, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.4056287692651329, 0.2826086956521739, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.41277641277641275, 0.32608695652173914, 'gini = 0.0\nsamples = 5\nvalue = [5,
0]'),
 Text(0.41992405628769264, 0.3695652173913043, 'gini = 0.0\nsamples = 18\nvalue = [18,
0]'),
 Text(0.4413669868215323, 0.5, 'Age <= 26.5\ngini = 0.245\nsamples = 14\nvalue = [12,
2]'),
 Text(0.4342193433102524, 0.45652173913043476, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.44851463033281214, 0.45652173913043476, 'Annual_income <= 146250.0\ngini = 0.142
\nsamples = 13\nvalue = [12, 1]'),
 Text(0.4413669868215323, 0.41304347826086957, 'Ind_ID <= 5067868.5\ngini = 0.375\nsampl
es = 4\nvalue = [3, 1]'),
 Text(0.4342193433102524, 0.3695652173913043, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.44851463033281214, 0.3695652173913043, 'Annual_income <= 112500.0\ngini = 0.5\ns
amples = 2\nvalue = [1, 1]'),
 Text(0.4413669868215323, 0.32608695652173914, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.455662273844092, 0.32608695652173914, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.455662273844092, 0.41304347826086957, 'gini = 0.0\nsamples = 9\nvalue = [9,
0]'),
 Text(0.4074156801429529, 0.5869565217391305, 'gini = 0.0\nsamples = 52\nvalue = [52,
0]'),
 Text(0.5333928970292606, 0.6304347826086957, 'Employment_Year <= 4.5\ngini = 0.162\nsam
ples = 259\nvalue = [236, 23]'),
 Text(0.47710520437793164, 0.5869565217391305, 'Propert_Owner_Y <= 0.5\ngini = 0.308\nsa
mples = 21\nvalue = [17, 4]'),
 Text(0.4628099173553719, 0.5434782608695652, 'Annual_income <= 108750.0\ngini = 0.5\nsa
mples = 6\nvalue = [3, 3]'),
 Text(0.455662273844092, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.4699575608666518, 0.5, 'Family_Members <= 2.5\ngini = 0.375\nsamples = 4\nvalue
= [1, 3]'),
 Text(0.4628099173553719, 0.45652173913043476, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.47710520437793164, 0.45652173913043476, 'Ind_ID <= 5054063.5\ngini = 0.5\nsample
s = 2\nvalue = [1, 1]'),
 Text(0.4699575608666518, 0.41304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.4842528478892115, 0.41304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.4914004914004914, 0.5434782608695652, 'Age <= 52.0\ngini = 0.124\nsamples = 15\n
value = [14, 1]'),
 Text(0.4842528478892115, 0.5, 'gini = 0.0\nsamples = 12\nvalue = [12, 0]'),
 Text(0.4985481349117713, 0.5, 'Phone <= 0.5\ngini = 0.444\nsamples = 3\nvalue = [2,
1]'),
 Text(0.4914004914004914, 0.45652173913043476, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.5056957784230511, 0.45652173913043476, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.5896805896805897, 0.5869565217391305, 'Age <= 59.5\ngini = 0.147\nsamples = 238
\nvalue = [219, 19]'),
 Text(0.5521554612463704, 0.5434782608695652, 'Age <= 53.5\ngini = 0.117\nsamples = 176
\nvalue = [165, 11]'),
 Text(0.5342863524681707, 0.5, 'Ind_ID <= 5022093.5\ngini = 0.171\nsamples = 106\nvalue
= [96, 10]'),
 Text(0.5199910654456109, 0.45652173913043476, 'Age <= 51.5\ngini = 0.337\nsamples = 14
\nvalue = [11, 3]'),
 Text(0.5128434219343311, 0.41304347826086957, 'gini = 0.0\nsamples = 10\nvalue = [10,
0]'),
```

```
 Text(0.5271387089568907, 0.41304347826086957, 'Car_Owner_Y <= 0.5\ngini = 0.375\nsample
s = 4\nvalue = [1, 3]'),
 Text(0.5199910654456109, 0.3695652173913043, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.5342863524681707, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.5485816394907304, 0.45652173913043476, 'Ind_ID <= 5041872.5\ngini = 0.141\nsampl
es = 92\nvalue = [85, 7]'),
 Text(0.5414339959794505, 0.41304347826086957, 'gini = 0.0\nsamples = 39\nvalue = [39,
0]'),
 Text(0.5557292830020103, 0.41304347826086957, 'Ind_ID <= 5042073.0\ngini = 0.229\nsampl
es = 53\nvalue = [46, 7]'),
 Text(0.5485816394907304, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.5628769265132901, 0.3695652173913043, 'Age <= 42.5\ngini = 0.204\nsamples = 52\n
value = [46, 6]'),
 Text(0.5485816394907304, 0.32608695652173914, 'Type_Occupation <= 5.5\ngini = 0.444\nsa
mples = 6\nvalue = [4, 2]'),
 Text(0.5414339959794505, 0.2826086956521739, 'gini = 0.0\nsamples = 3\nvalue = [3,
0]'),
 Text(0.5557292830020103, 0.2826086956521739, 'Ind_ID <= 5044312.0\ngini = 0.444\nsample
s = 3\nvalue = [1, 2]'),
 Text(0.5485816394907304, 0.2391304347826087, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.5628769265132901, 0.2391304347826087, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.5771722135358499, 0.32608695652173914, 'Age <= 45.5\ngini = 0.159\nsamples = 46
\nvalue = [42, 4]'),
 Text(0.5700245700245701, 0.2826086956521739, 'gini = 0.0\nsamples = 17\nvalue = [17,
0]'),
 Text(0.5843198570471297, 0.2826086956521739, 'Phone <= 0.5\ngini = 0.238\nsamples = 29
\nvalue = [25, 4]'),
 Text(0.5771722135358499, 0.2391304347826087, 'Age <= 52.5\ngini = 0.308\nsamples = 21\n
value = [17, 4]'),
 Text(0.5700245700245701, 0.1956521739130435, 'Ind_ID <= 5057865.5\ngini = 0.255\nsample
s = 20\nvalue = [17, 3]'),
 Text(0.5628769265132901, 0.15217391304347827, 'Ind_ID <= 5052845.0\ngini = 0.375\nsampl
es = 12\nvalue = [9, 3]'),
 Text(0.5485816394907304, 0.10869565217391304, 'Annual_income <= 162450.0\ngini = 0.198
\nsamples = 9\nvalue = [8, 1]'),
 Text(0.5414339959794505, 0.06521739130434782, 'Age <= 48.5\ngini = 0.5\nsamples = 2\nva
lue = [1, 1]'),
 Text(0.5342863524681707, 0.021739130434782608, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.5485816394907304, 0.021739130434782608, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.5557292830020103, 0.06521739130434782, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.5771722135358499, 0.10869565217391304, 'Employment_Year <= 16.0\ngini = 0.444\ns
amples = 3\nvalue = [1, 2]'),
 Text(0.5700245700245701, 0.06521739130434782, 'Ind_ID <= 5052881.0\ngini = 0.5\nsamples
= 2\nvalue = [1, 1]'),
 Text(0.5628769265132901, 0.021739130434782608, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.5771722135358499, 0.021739130434782608, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.5843198570471297, 0.06521739130434782, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.5771722135358499, 0.15217391304347827, 'gini = 0.0\nsamples = 8\nvalue = [8,
0]'),
 Text(0.5843198570471297, 0.1956521739130435, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.5914675005584097, 0.2391304347826087, 'gini = 0.0\nsamples = 8\nvalue = [8,
0]'),
```

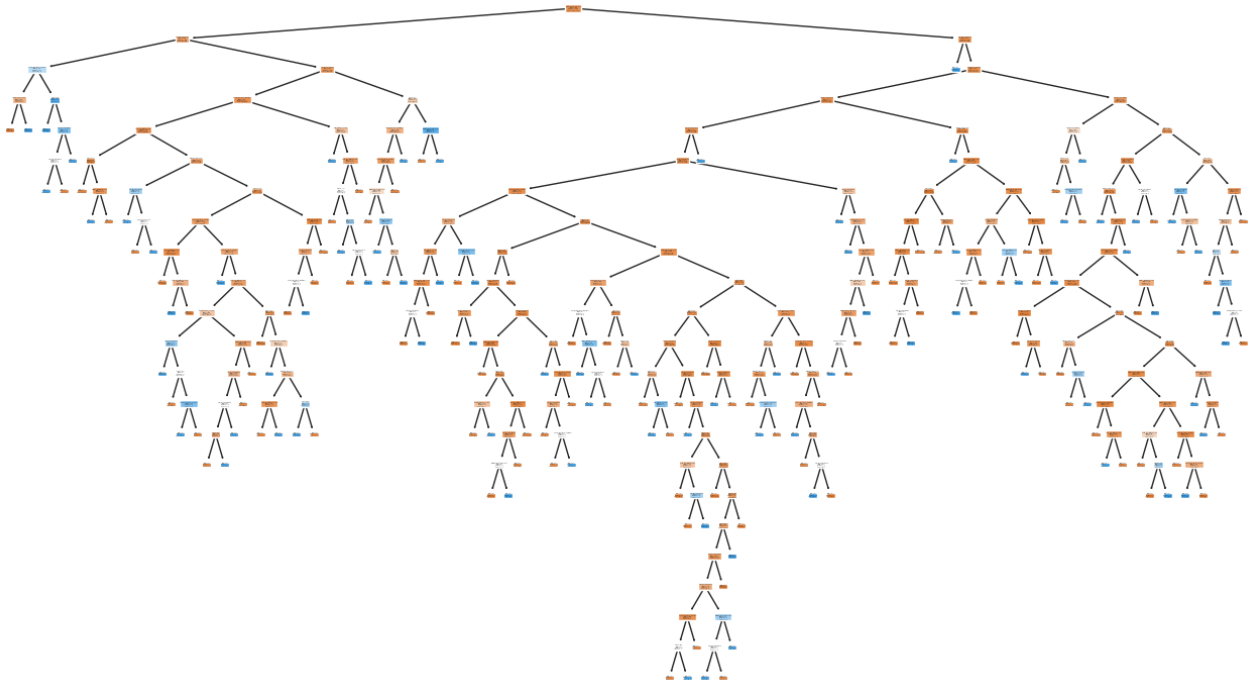
```
 Text(0.5700245700245701, 0.5, 'Ind_ID <= 5061390.5\ngini = 0.028\nsamples = 70\nvalue =
[69, 1]'),
 Text(0.5628769265132901, 0.45652173913043476, 'gini = 0.0\nsamples = 58\nvalue = [58,
0]'),
 Text(0.5771722135358499, 0.45652173913043476, 'Ind_ID <= 5061855.5\ngini = 0.153\nsampl
es = 12\nvalue = [11, 1]'),
 Text(0.5700245700245701, 0.41304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.5843198570471297, 0.41304347826086957, 'gini = 0.0\nsamples = 11\nvalue = [11,
0]'),
 Text(0.6272057181148091, 0.5434782608695652, 'Annual_income <= 117000.0\ngini = 0.225\n
samples = 62\nvalue = [54, 8]'),
 Text(0.6129104310922493, 0.5, 'Phone <= 0.5\ngini = 0.388\nsamples = 19\nvalue = [14,
5]'),
 Text(0.6057627875809694, 0.45652173913043476, 'EDUCATION <= 2.0\ngini = 0.219\nsamples
= 16\nvalue = [14, 2]'),
 Text(0.5986151440696895, 0.41304347826086957, 'gini = 0.0\nsamples = 13\nvalue = [13,
0]'),
 Text(0.6129104310922493, 0.41304347826086957, 'Annual_income <= 105750.0\ngini = 0.444
\nsamples = 3\nvalue = [1, 2]'),
 Text(0.6057627875809694, 0.3695652173913043, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.6200580746035291, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.6200580746035291, 0.45652173913043476, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.6415010051373687, 0.5, 'Annual_income <= 173250.0\ngini = 0.13\nsamples = 43\nva
lue = [40, 3]'),
 Text(0.6343533616260889, 0.45652173913043476, 'gini = 0.0\nsamples = 21\nvalue = [21,
0]'),
 Text(0.6486486486486487, 0.45652173913043476, 'EDUCATION <= 1.5\ngini = 0.236\nsamples
= 22\nvalue = [19, 3]'),
 Text(0.6415010051373687, 0.41304347826086957, 'Annual_income <= 189000.0\ngini = 0.375
\nsamples = 12\nvalue = [9, 3]'),
 Text(0.6343533616260889, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.6486486486486487, 0.3695652173913043, 'Age <= 63.5\ngini = 0.298\nsamples = 11\n
value = [9, 2]'),
 Text(0.6415010051373687, 0.32608695652173914, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.6557962921599285, 0.32608695652173914, 'Ind_ID <= 5056301.5\ngini = 0.5\nsamples
= 4\nvalue = [2, 2]'),
 Text(0.6486486486486487, 0.2826086956521739, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.6629439356712084, 0.2826086956521739, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.6557962921599285, 0.41304347826086957, 'gini = 0.0\nsamples = 10\nvalue = [10,
0]'),
 Text(0.6772392226937681, 0.717391304347826, 'Ind_ID <= 5069253.5\ngini = 0.444\nsamples
= 18\nvalue = [12, 6]'),
 Text(0.6700915791824883, 0.6739130434782609, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.6843868662050481, 0.6739130434782609, 'Propert_Owner_Y <= 0.5\ngini = 0.375\nsam
ples = 16\nvalue = [12, 4]'),
 Text(0.6772392226937681, 0.6304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.6915345097163279, 0.6304347826086957, 'Type_Occupation <= 3.0\ngini = 0.32\nsamp
les = 15\nvalue = [12, 3]'),
 Text(0.6843868662050481, 0.5869565217391305, 'Family_Members <= 2.5\ngini = 0.469\nsamp
les = 8\nvalue = [5, 3]'),
 Text(0.6772392226937681, 0.5434782608695652, 'Employment_Year <= 6.5\ngini = 0.278\nsam
ples = 6\nvalue = [5, 1]'),
 Text(0.6700915791824883, 0.5, 'Ind_ID <= 5069353.0\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
```

```
 Text(0.6629439356712084, 0.45652173913043476, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.6772392226937681, 0.45652173913043476, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.6843868662050481, 0.5, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.6915345097163279, 0.5434782608695652, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.6986821532276077, 0.5869565217391305, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.558968058968059, 0.7608695652173914, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.7683716774625865, 0.8043478260869565, 'EDUCATION <= 0.5\ngini = 0.066\nsamples =
206\nvalue = [199, 7]'),
 Text(0.7612240339513067, 0.7608695652173914, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.7755193209738664, 0.7608695652173914, 'Employment_Year <= 18.5\ngini = 0.057\nsa
mples = 205\nvalue = [199, 6]'),
 Text(0.7415680142952871, 0.717391304347826, 'Age <= 55.5\ngini = 0.028\nsamples = 143\n
value = [141, 2]'),
 Text(0.7272727272727273, 0.6739130434782609, 'Type_Occupation <= 1.5\ngini = 0.014\nsam
ples = 137\nvalue = [136, 1]'),
 Text(0.7201250837614473, 0.6304347826086957, 'Family_Members <= 2.5\ngini = 0.091\nsamp
les = 21\nvalue = [20, 1]'),
 Text(0.7129774402501675, 0.5869565217391305, 'gini = 0.0\nsamples = 15\nvalue = [15,
0]'),
 Text(0.7272727272727273, 0.5869565217391305, 'Ind_ID <= 5097685.5\ngini = 0.278\nsample
s = 6\nvalue = [5, 1]'),
 Text(0.7201250837614473, 0.5434782608695652, 'Ind_ID <= 5089956.5\ngini = 0.5\nsamples
= 2\nvalue = [1, 1]'),
 Text(0.7129774402501675, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7272727272727273, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.7344203707840071, 0.5434782608695652, 'gini = 0.0\nsamples = 4\nvalue = [4,
0]'),
 Text(0.7344203707840071, 0.6304347826086957, 'gini = 0.0\nsamples = 116\nvalue = [116,
0]'),
 Text(0.7558633013178467, 0.6739130434782609, 'EDUCATION <= 2.0\ngini = 0.278\nsamples =
6\nvalue = [5, 1]'),
 Text(0.7487156578065669, 0.6304347826086957, 'gini = 0.0\nsamples = 5\nvalue = [5,
0]'),
 Text(0.7630109448291267, 0.6304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.8094706276524458, 0.717391304347826, 'Employment_Year <= 22.5\ngini = 0.121\nsam
ples = 62\nvalue = [58, 4]'),
 Text(0.7916015188742461, 0.6739130434782609, 'Ind_ID <= 5096649.5\ngini = 0.375\nsample
s = 12\nvalue = [9, 3]'),
 Text(0.7773062318516863, 0.6304347826086957, 'Type_Occupation <= 3.5\ngini = 0.198\nsam
ples = 9\nvalue = [8, 1]'),
 Text(0.7701585883404065, 0.5869565217391305, 'Annual_income <= 139500.0\ngini = 0.5\nsa
mples = 2\nvalue = [1, 1]'),
 Text(0.7630109448291267, 0.5434782608695652, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.7773062318516863, 0.5434782608695652, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.7844538753629663, 0.5869565217391305, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.8058968058968059, 0.6304347826086957, 'Type_Occupation <= 6.0\ngini = 0.444\nsam
ples = 3\nvalue = [1, 2]'),
 Text(0.7987491623855261, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.8130444494080857, 0.5869565217391305, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.8273397364306455, 0.6739130434782609, 'Annual_income <= 220500.0\ngini = 0.039\n
samples = 50\nvalue = [49, 1]'),
 Text(0.8201920929193657, 0.6304347826086957, 'gini = 0.0\nsamples = 39\nvalue = [39,
0]'),
```

```
 Text(0.8344873799419253, 0.6304347826086957, 'Ind_ID <= 5102608.0\ngini = 0.165\nsample
s = 11\nvalue = [10, 1]'),
 Text(0.8273397364306455, 0.5869565217391305, 'gini = 0.0\nsamples = 10\nvalue = [10,
0]'),
 Text(0.8416350234532053, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.8945722582086219, 0.8478260869565217, 'Ind_ID <= 5115636.5\ngini = 0.243\nsample
s = 240\nvalue = [206, 34]'),
 Text(0.8570471297744024, 0.8043478260869565, 'Ind_ID <= 5115585.0\ngini = 0.488\nsample
s = 19\nvalue = [11, 8]'),
 Text(0.8498994862631226, 0.7608695652173914, 'Phone <= 0.5\ngini = 0.391\nsamples = 15
\nvalue = [11, 4]'),
 Text(0.8427518427518428, 0.717391304347826, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.8570471297744024, 0.717391304347826, 'Annual_income <= 164250.0\ngini = 0.444\ns
amples = 6\nvalue = [2, 4]'),
 Text(0.8498994862631226, 0.6739130434782609, 'gini = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.8641947732856824, 0.6739130434782609, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.8641947732856824, 0.7608695652173914, 'gini = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.9320973866428411, 0.8043478260869565, 'Age <= 57.5\ngini = 0.208\nsamples = 221
\nvalue = [195, 26]'),
 Text(0.8999329908420818, 0.7608695652173914, 'Ind_ID <= 5149816.5\ngini = 0.159\nsample
s = 184\nvalue = [168, 16]'),
 Text(0.885637703819522, 0.717391304347826, 'EDUCATION <= 0.5\ngini = 0.135\nsamples = 1
78\nvalue = [165, 13]'),
 Text(0.8784900603082422, 0.6739130434782609, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.8927853473308018, 0.6739130434782609, 'Family_Members <= 5.5\ngini = 0.126\nsamp
les = 177\nvalue = [165, 12]'),
 Text(0.885637703819522, 0.6304347826086957, 'Annual_income <= 436500.0\ngini = 0.117\ns
amples = 176\nvalue = [165, 11]'),
 Text(0.8559303104757651, 0.5869565217391305, 'Propert_Owner_Y <= 0.5\ngini = 0.101\nsam
ples = 168\nvalue = [159, 9]'),
 Text(0.8179584543220907, 0.5434782608695652, 'Ind_ID <= 5143258.5\ngini = 0.028\nsample
s = 70\nvalue = [69, 1]'),
 Text(0.8108108108108109, 0.5, 'gini = 0.0\nsamples = 61\nvalue = [61, 0]'),
 Text(0.8251060978333705, 0.5, 'Ind_ID <= 5143347.5\ngini = 0.198\nsamples = 9\nvalue =
[8, 1]'),
 Text(0.8179584543220907, 0.45652173913043476, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.8322537413446505, 0.45652173913043476, 'gini = 0.0\nsamples = 8\nvalue = [8,
0]'),
 Text(0.8939021666294393, 0.5434782608695652, 'Age <= 29.5\ngini = 0.15\nsamples = 98\nv
alue = [90, 8]'),
 Text(0.8536966718784901, 0.5, 'EDUCATION <= 2.5\ngini = 0.375\nsamples = 8\nvalue = [6,
2]'),
 Text(0.8465490283672101, 0.45652173913043476, 'gini = 0.0\nsamples = 5\nvalue = [5,
0]'),
 Text(0.8608443153897699, 0.45652173913043476, 'Ind_ID <= 5135484.5\ngini = 0.444\nsampl
es = 3\nvalue = [1, 2]'),
 Text(0.8536966718784901, 0.41304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.8679919589010499, 0.41304347826086957, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.9341076613803887, 0.5, 'Age <= 55.5\ngini = 0.124\nsamples = 90\nvalue = [84,
6]'),
 Text(0.9073039982130892, 0.45652173913043476, 'Employment_Year <= 10.5\ngini = 0.094\ns
amples = 81\nvalue = [77, 4]'),
 Text(0.8822872459236095, 0.41304347826086957, 'Annual_income <= 265500.0\ngini = 0.036
\nsamples = 55\nvalue = [54, 1]'),
 Text(0.8751396024123297, 0.3695652173913043, 'gini = 0.0\nsamples = 47\nvalue = [47,
0]'),
```

```
 Text(0.8894348894348895, 0.3695652173913043, 'Type_Occupation <= 3.5\ngini = 0.219\nsam
ples = 8\nvalue = [7, 1]'),
 Text(0.8822872459236095, 0.32608695652173914, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.8965825329461693, 0.32608695652173914, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.9323207505025687, 0.41304347826086957, 'Employment_Year <= 13.0\ngini = 0.204\ns
amples = 26\nvalue = [23, 3]'),
 Text(0.9180254634800089, 0.3695652173913043, 'Type_Occupation <= 3.5\ngini = 0.48\nsamp
les = 5\nvalue = [3, 2]'),
 Text(0.9108778199687291, 0.32608695652173914, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.9251731069912889, 0.32608695652173914, 'Phone <= 0.5\ngini = 0.444\nsamples = 3
\nvalue = [1, 2]'),
 Text(0.9180254634800089, 0.2826086956521739, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.9323207505025687, 0.2826086956521739, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.9466160375251285, 0.3695652173913043, 'Annual_income <= 207000.0\ngini = 0.091\n
samples = 21\nvalue = [20, 1]'),
 Text(0.9394683940138485, 0.32608695652173914, 'gini = 0.0\nsamples = 17\nvalue = [17,
0]'),
 Text(0.9537636810364083, 0.32608695652173914, 'Annual_income <= 240750.0\ngini = 0.375
\nsamples = 4\nvalue = [3, 1]'),
 Text(0.9466160375251285, 0.2826086956521739, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.9609113245476882, 0.2826086956521739, 'gini = 0.0\nsamples = 3\nvalue = [3,
0]'),
 Text(0.9609113245476882, 0.45652173913043476, 'Employment_Year <= 6.5\ngini = 0.346\nsa
mples = 9\nvalue = [7, 2]'),
 Text(0.9537636810364083, 0.41304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.9680589680589681, 0.41304347826086957, 'Ind_ID <= 5125388.5\ngini = 0.219\nsampl
es = 8\nvalue = [7, 1]'),
 Text(0.9609113245476882, 0.3695652173913043, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.9752066115702479, 0.3695652173913043, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.9153450971632789, 0.5869565217391305, 'Family_Members <= 3.5\ngini = 0.375\nsamp
les = 8\nvalue = [6, 2]'),
 Text(0.9081974536519991, 0.5434782608695652, 'gini = 0.0\nsamples = 6\nvalue = [6,
0]'),
 Text(0.9224927406745589, 0.5434782608695652, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.8999329908420818, 0.6304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.9142282778646414, 0.717391304347826, 'Ind_ID <= 5149862.5\ngini = 0.5\nsamples =
6\nvalue = [3, 3]'),
 Text(0.9070806343533616, 0.6739130434782609, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.9213759213759214, 0.6739130434782609, 'gini = 0.0\nsamples = 3\nvalue = [3,
0]'),
 Text(0.9642617824436006, 0.7608695652173914, 'Age <= 60.0\ngini = 0.394\nsamples = 37\n
value = [27, 10]'),
 Text(0.942818851909761, 0.717391304347826, 'Ind_ID <= 5134604.0\ngini = 0.346\nsamples
= 9\nvalue = [2, 7]'),
 Text(0.9356712083984812, 0.6739130434782609, 'gini = 0.0\nsamples = 6\nvalue = [0,
6]'),
 Text(0.9499664954210408, 0.6739130434782609, 'Annual_income <= 243000.0\ngini = 0.444\n
samples = 3\nvalue = [2, 1]'),
 Text(0.942818851909761, 0.6304347826086957, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.9571141389323208, 0.6304347826086957, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.9857047129774402, 0.717391304347826, 'Ind_ID <= 5121815.5\ngini = 0.191\nsamples
```

```
= 28\nvalue = [25, 3]'),
 Text(0.9785570694661604, 0.6739130434782609, 'EDUCATION <= 2.0\ngini = 0.444\nsamples =
9\nvalue = [6, 3]'),
 Text(0.9714094259548804, 0.6304347826086957, 'Age <= 61.5\ngini = 0.48\nsamples = 5\nva
lue = [2, 3]'),
 Text(0.9642617824436006, 0.5869565217391305, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.9785570694661604, 0.5869565217391305, 'Ind_ID <= 5116676.0\ngini = 0.375\nsample
s = 4\nvalue = [1, 3]'),
 Text(0.9714094259548804, 0.5434782608695652, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.9857047129774402, 0.5434782608695652, 'Propert_Owner_Y <= 0.5\ngini = 0.5\nsampl
es = 2\nvalue = [1, 1]'),
 Text(0.9785570694661604, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.9928523564887202, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.9857047129774402, 0.6304347826086957, 'gini = 0.0\nsamples = 4\nvalue = [4,
0]'),
 Text(0.9928523564887202, 0.6739130434782609, 'gini = 0.0\nsamples = 19\nvalue = [19,
0]')]
```



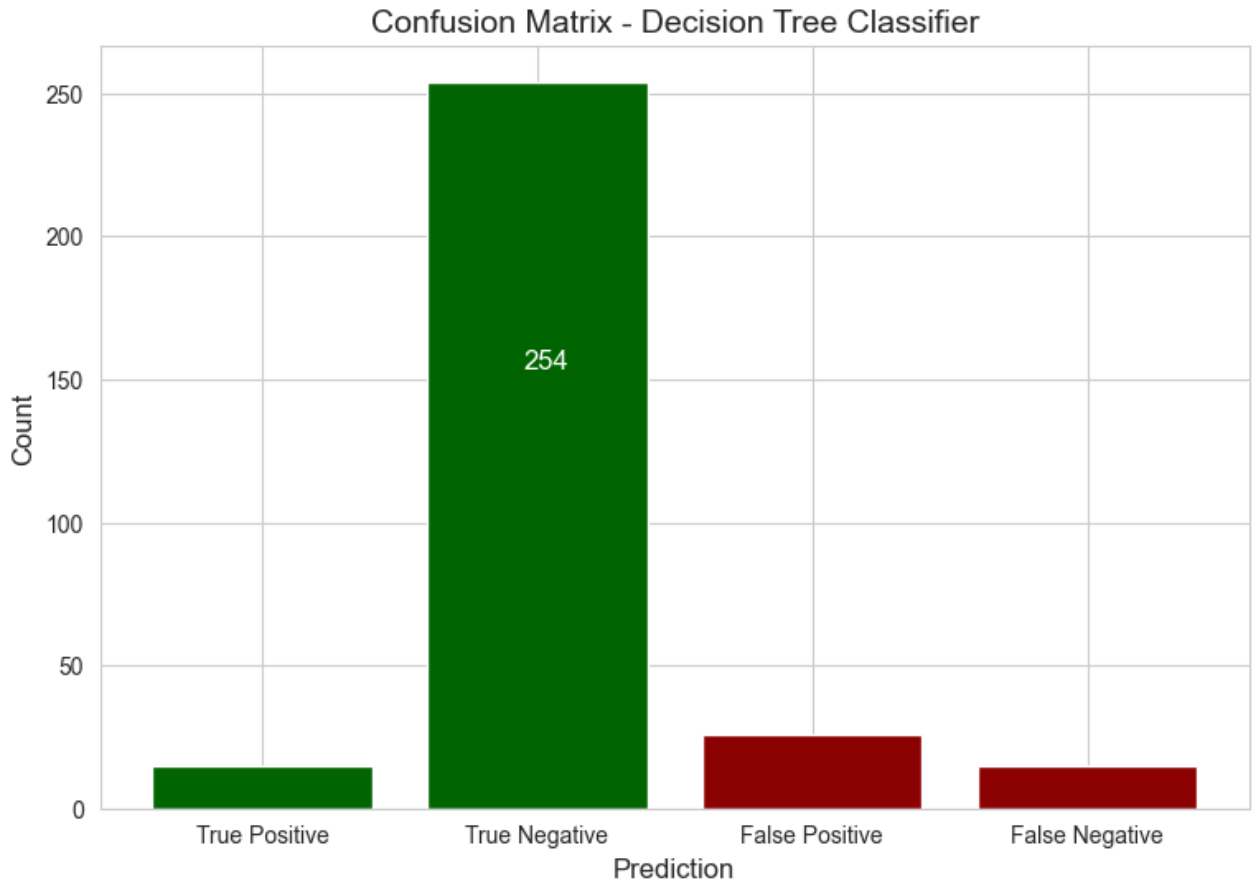
The above visualizes the tree model, showing the hierarchical structure of decisions made by the classifier based on the features in the dataset. Each node represents a decision based on a feature, and the branches represent the possible outcomes of that decision.

In [ ]:
```
dt_Accuracy_Score = accuracy_score(y_test, y_pred)

print(dt_Accuracy_Score)

dt_conf_matrix = confusion_matrix(y_test, y_pred)

print(dt_conf_matrix)
```

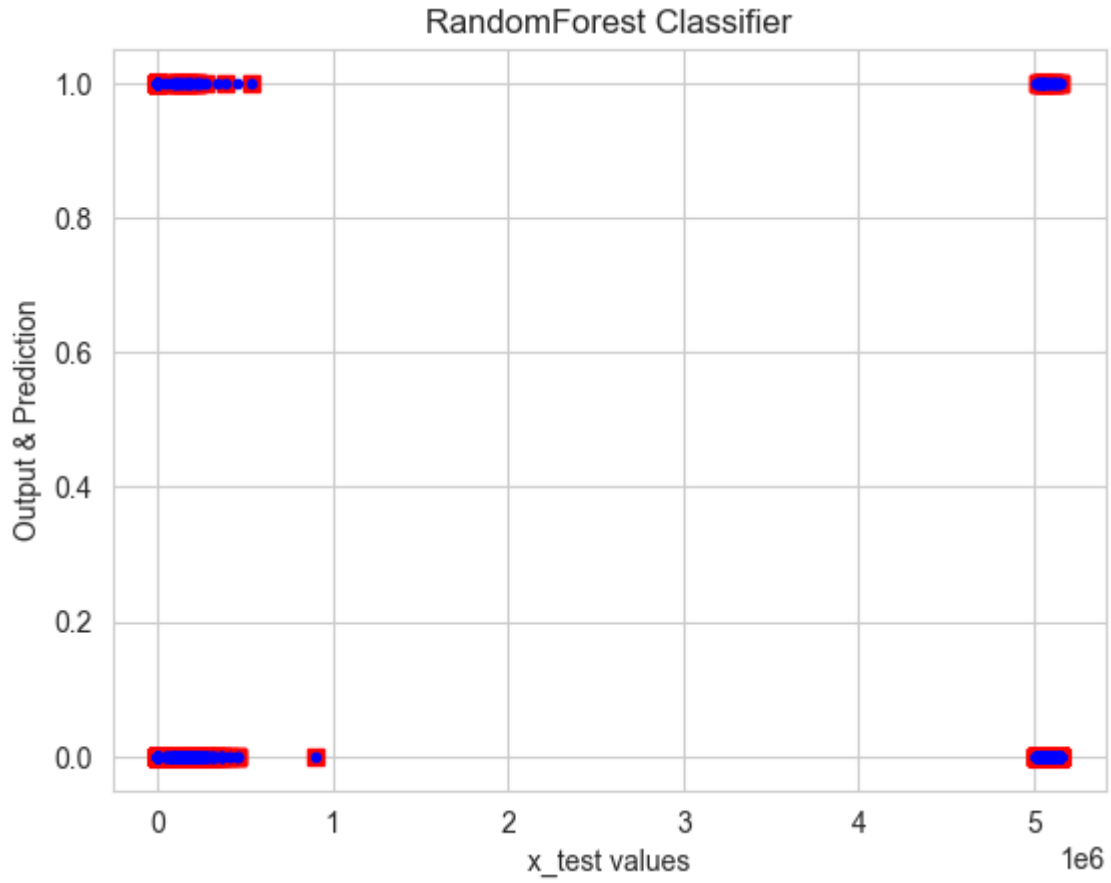
```
0.867741935483871
[[254  26]
 [ 15  15]]
```

In [ ]:
```
# Calculate True Positives (TP), True Negatives (TN), False Positives (FP), and False Ne
TP = dt_conf_matrix[1, 1]
TN = dt_conf_matrix[0, 0]
FP = dt_conf_matrix[0, 1]
FN = dt_conf_matrix[1, 0]
```

```
# Plotting the confusion matrix using bar plots
plt.figure(figsize=(9, 6))
bars = plt.bar(x=['True Positive', 'True Negative', 'False Positive', 'False Negative'],

# Adding text labels on top of each bar
for bar in bars:
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.05, bar.get_height() - 100, f'{int(ba

plt.xlabel('Prediction', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Confusion Matrix - Decision Tree Classifier', fontsize=14)
plt.show()
```



In [ ]:
```
plt.plot(x_test, y_test, "s", color = "red")
plt.plot(x_test, y_pred, ".", color = "blue")
plt.title("RandomForest Classifier")
plt.ylabel("Output & Prediction")
plt.xlabel("x_test values")
plt.show()
```



DEcision Tree Cross-Validation

In [ ]:
```
from sklearn.model_selection import cross_val_score
cross_val_decisiontree = cross_val_score(dct, X_selected, y, scoring='accuracy', cv = 5)
```

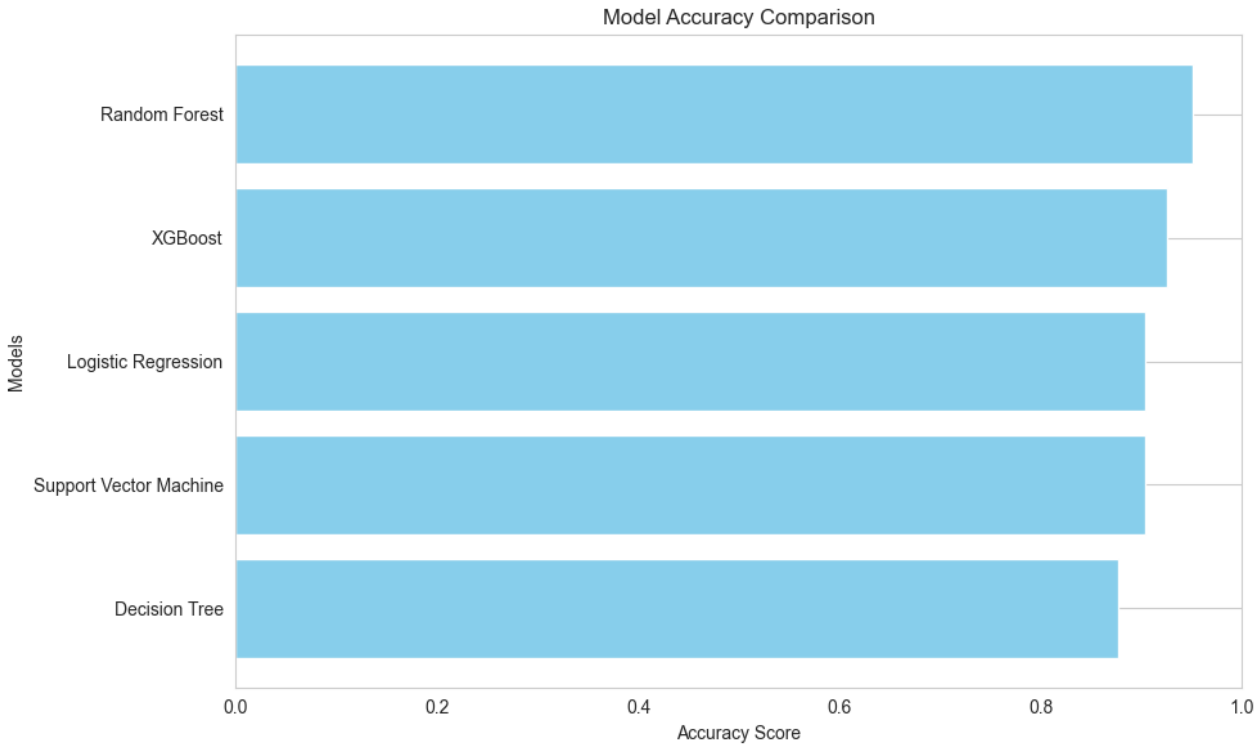
In [ ]:
```
cross_val_decisiontree
```

Out[ ]:
```
array([0.8483871 , 0.79354839, 0.83225806, 0.79288026, 0.802589  ])
```

I. Model Accuracy Comparison

In [ ]:
```
# Model accuracy data
Model_accuracy = {
    "Decision Tree": {"Accuracy_score": 0.8774193548387097},
    "Support Vector Machine": {"Accuracy_score": 0.9032258064516129},
    "Logistic Regression": {"Accuracy_score": 0.9032258064516129},
    "XGBoost": {"Accuracy_score": 0.9258064516129032},
    "Random Forest": {"Accuracy_score": 0.9516129032258065}
}

# Create DataFrame
Model_Accuracy = pd.DataFrame(Model_accuracy)

# Transpose DataFrame
Model_Accuracy = Model_Accuracy.transpose()

# Plotting the bar graph
plt.figure(figsize=(10, 6))
plt.barh(Model_Accuracy.index, Model_Accuracy['Accuracy_score'], color='skyblue')
plt.xlabel('Accuracy Score')
plt.ylabel('Models')
plt.title('Model Accuracy Comparison')
```

```
plt.xlim(0, 1)  # Set the x-axis limit to ensure accuracy scores are visible
plt.grid(axis='x')  # Add gridlines for better readability
plt.tight_layout()
plt.show()
```



J. SQL

- I have solved SQL Question in MySQL Workbench as well, I've saved file in the .zip file
- Use MySQL or PyMySQL to perform the below queries.
- Note: Use only the cleaned data for SQL part of the project

In [ ]:
```
import duckdb
conn=duckdb.connect()
conn.register('df',dv)
conn.execute("select * from df ").fetchdf()
```

Out[ ]:

	Ind_ID	Gender	Car_Owner	Property_Owner	Children	Annual_income	Type_Income	Education
0	5008827	M	Y	Y	0	180000	Pensioner	Higher education
1	5009744	F	Y	N	0	315000	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000	Commercial associate	Higher education
3	5009749	F	Y	N	0	315000	Commercial associate	Higher education
4	5009752	F	Y	N	0	315000	Commercial associate	Higher education
...	...	...	...	...	...	...	...	...
1543	5028645	F	N	Y	0	135000	Commercial associate	Higher education
1544	5023655	F	N	N	0	225000	Commercial associate	Incomplete higher
1545	5115992	M	Y	Y	2	180000	Working	Higher education
1546	5118219	M	Y	N	0	270000	Working	Secondary / secondary special
1547	5053790	F	Y	Y	0	225000	Working	Higher education

1548 rows × 19 columns

1. Group the customers based on their income type and find the average of their annual income.

In [ ]:
```
import duckdb
conn=duckdb.connect()
conn.register('df',dv)
conn.execute("SELECT Type_Income, ROUND(AVG(Annual_income),2) as Average_Annual_Income F
```

Out[ ]:

	Type_Income	Average_Annual_Income
0	Commercial associate	233932.60
1	State servant	211422.41
2	Working	180905.73
3	Pensioner	155131.60

2. Find the female owners of cars and property.

In [ ]:
```
import duckdb
conn=duckdb.connect()
conn.register('df',dv)
conn.execute("SELECT Ind_ID,Gender, Car_Owner, Property_Owner FROM dv WHERE Gender = 'F'
```

Out[ ]:

	Ind_ID	Gender	Car_Owner	Property_Owner
0	5018498	F	Y	Y
1	5018501	F	Y	Y
2	5018503	F	Y	Y
3	5024213	F	Y	Y
4	5036660	F	Y	Y
...	...	...	...	...
174	5048458	F	Y	Y
175	5023719	F	Y	Y
176	5033520	F	Y	Y
177	5024049	F	Y	Y
178	5053790	F	Y	Y

179 rows × 4 columns

3. Find the male customers who are staying with their families.

- Here we have 2 approaches to the question:
  - 1st :
    - There is an male living with his parents (could be an single parent or both)
  - 2nd
    - He could be living with his wife and childrens which would make up his family have more than 2 members or he and his wife make family of atleast 2.

In [ ]:

```
import duckdb
conn=duckdb.connect()
conn.register('df',dv)
conn.execute("SELECT Ind_ID, Gender, Children, Family_Members, Housing_Type FROM dv WHER
```

Out[ ]:

	Ind_ID	Gender	Children	Family_Members	Housing_type
0	5008827	M	0	2	House / apartment
1	5010864	M	1	3	House / apartment
2	5010868	M	1	3	House / apartment
3	5021303	M	1	3	With parents
4	5021310	M	0	2	House / apartment
...	...	...	...	...	...
473	5150038	M	0	1	With parents
474	5145694	M	0	1	With parents
475	5068648	M	0	1	With parents
476	5094884	M	0	1	With parents
477	5028612	M	0	1	With parents

478 rows × 5 columns

4. Please list the top five people having the highest income.

In [ ]:
```
import duckdb
conn=duckdb.connect()
conn.register('df',dv)
conn.execute("SELECT * FROM dv ORDER BY Annual_Income DESC LIMIT 5 ").fetchdf()
```

Out[ ]:

	Ind_ID	Gender	Car_Owner	Property_Owner	Children	Annual_income	Type_Income	Education	M
0	5143231	F	Y	Y	1	1575000	Commercial associate	Higher education	
1	5143235	F	Y	Y	1	1575000	Commercial associate	Higher education	
2	5090470	M	N	Y	1	900000	Working	Secondary / secondary special	
3	5079016	M	Y	Y	2	900000	Commercial associate	Higher education	
4	5079017	M	Y	Y	2	900000	Commercial associate	Higher education	

5. How many married people are having bad credit?

- Here since we don't have enough information on Credit limit/score, credit history, pending bills, delayed payments history etc, we won't be able to find bad credits, hence I'm using label (which is approval for credit card 0=approved, 1=rejected), hence I'm using 1 (rejected) as bad credit in this context.

In [ ]:
```
import duckdb
conn=duckdb.connect()
```

```
conn.register('df',dv)
conn.execute("SELECT * FROM dv WHERE (label = 1 AND Marital_Status = 'Married');").fetch
```

Out[ ]:

	Ind_ID	Gender	Car_Owner	Property_Owner	Children	Annual_income	Type_Income	Education
0	5008827	M	Y	Y	0	180000	Pensioner	Higher education
1	5009744	F	Y	N	0	315000	Commercial associate	Higher education
2	5009746	F	Y	N	0	315000	Commercial associate	Higher education
3	5009749	F	Y	N	0	315000	Commercial associate	Higher education
4	5009752	F	Y	N	0	315000	Commercial associate	Higher education
...	...	...	...	...	...	...	...	...
109	5149190	M	Y	N	1	450000	Working	Higher education
110	5149192	F	Y	N	1	450000	Working	Higher education
111	5149828	M	Y	Y	0	315000	Working	Secondary / secondary special
112	5149834	F	N	Y	0	157500	Commercial associate	Higher education
113	5149838	F	N	Y	0	157500	Pensioner	Higher education

114 rows × 19 columns

6. What is the highest education level and what is the total count?

In [ ]:
```
import duckdb
conn=duckdb.connect()
conn.register('df',dv)
conn.execute("SELECT Education, COUNT(Education) as Count_Education FROM dv GROUP BY Edu
```

Out[ ]:

	Education	Count_Education
0	Secondary / secondary special	1031
1	Higher education	426
2	Incomplete higher	68
3	Lower secondary	21
4	Academic degree	2

7. Between married males and females, who is having more bad credit?

In [ ]:
```
import duckdb
conn=duckdb.connect()
conn.register('df',dv)
conn.execute("SELECT Gender, label, COUNT(label) as Number_of_bad_credits FROM dv WHERE
```

Out[ ]:

	Gender	label	Number_of_bad_credits
0	F	1	100
1	M	1	75

END
