Credit Card Approval

Project Details

A bank's credit card department is one of the top adopters of data science. A top focus for the bank has always been acquiring new credit card customers. Giving out credit cards without doing proper research or evaluating applicants' creditworthiness is quite risky. The credit card department has been using a data-driven system for credit assessment called Credit Scoring for many years, and the model is known as an application scorecard. A credit card application's cutoff value is determined using the application scorecard, which also aids in estimating the applicant's level of risk. This decision is made based on strategic priority at a given time.

Customers must fill out a form, either physically or online, to apply for a credit card. The application data is used to evaluate the applicant's creditworthiness. The decision is made using the application data in addition to the Credit Bureau Score, such as the FICO Score in the US or the CIBIL Score in India, and other internal information on the applicants. Additionally, the banks are rapidly taking a lot of outside data into account to enhance the caliber of credit judgements.

Features name: (Credit_Card.csv)

Ind_ID: Client ID

Gender: Gender information

Car_owner: Having car or not

Propert_owner: Having property or not

Children: Count of children

Annual_income: Annual income

Type_Income: Income type

Education: Education level

Marital_status: Marital_status

Housing_type: Living style

Birthday_count: Use backward count from current day (0), -1 means yesterday.

Employed_days: Start date of employment. Use backward count from current day (0). Positive value means, individual is currently unemployed.

Mobile_phone: Any mobile phone

Work_phone: Any work phone

Phone: Any phone number

EMAIL_ID: Any email ID

Type_Occupation: Occupation

Family_Members: Family size

Another data set (Credit_card_label.csv) contains two key pieces of information

ID: The joining key between application data and credit status data, same is Ind_ID

Label: 0 is application approved and 1 is application rejected.

Section 1:

Importance of Proposal

In today's data-centric world, leveraging advanced analytics and machine learning for credit card approval enables banks to make more informed decisions. By predicting the good client, bank can make more informed decisions to approved the applications and make them safe from client having high risk of default. Implementing an efficient credit card approval system can reduce the time and resource required for manual evaluation of applications.

Impact on the Banking Sector

Implementing the credit card approval system is likely to result in reduced default rates,thereby safegaurding the finanacial health of the bank. Automation of credit card approval system can speed up the approval process, improving efficiency and customer satisfaction. It helps the bank to take the quick decision wheather to approved or reject the application on the basis of data analysis and prediction.

Knowledge Gap and Future importance in India

The gap in knowledge that our proposed method addresses lies in predicting whether a credit card application will be approved or rejected. Our method helps by analyzing various factors, providing banks with a clearer understanding of the risks involved in approving someone's application.

In the future, if any bank in India needs to enhance its credit card approval process, our method can be quite beneficial. It allows banks to more accurately assess the risks associated with approving a credit card application. Additionally, the method can be customized to suit the specific needs of individual banks, making it a valuable tool for effectively managing credit risk

Section 2:

Initial Hypothesis (or hypotheses)

T-Test for Annual Income

- Null Hypothesis (H0): There is no significant difference in mean annual income between approved and rejected credit card applications.

ANOVA for Education Level

- Null Hypothesis (H0): There is no significant difference in mean annual income among different education levels.

After completing the univariate and bivariate analysis, I will proceed to test all this hypotheses in order to reject or accept the Null hypotheses.

Section 3:

Data Exploration

In [1]:
```
# Importing the important libraries for data preprocessing and data visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import missingno as msno
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Importing Credit_card dataset
df1=pd.read_csv("Credit_card.csv")
df1
```

Out[2]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EI
0	5008827	M	Y	Y	0	180000.0	Pensioner	
1	5009744	F	Y	N	0	315000.0	Commercial associate	
2	5009746	F	Y	N	0	315000.0	Commercial associate	
3	5009749	F	Y	N	0	NaN	Commercial associate	
4	5009752	F	Y	N	0	315000.0	Commercial associate	
...	...	...	...	...	...	...	...	
1543	5028645	F	N	Y	0	NaN	Commercial associate	
1544	5023655	F	N	N	0	225000.0	Commercial associate	
1545	5115992	M	Y	Y	2	180000.0	Working	
1546	5118219	M	Y	N	0	270000.0	Working	S
1547	5053790	F	Y	Y	0	225000.0	Working	

1548 rows × 18 columns

In [3]:

```
# shape of df1
df1.shape
```

Out[3]:

```
(1548, 18)
```

In [4]:

```
# Importing Credit_Card_Label Dataset
df2=pd.read_csv("Credit_card_label.csv")
df2
```

Out[4]:

	Ind_ID	label
0	5008827	1
1	5009744	1
2	5009746	1
3	5009749	1
4	5009752	1
...	...	...
1543	5028645	0
1544	5023655	0
1545	5115992	0
1546	5118219	0
1547	5053790	0

1548 rows × 2 columns

In [5]:

```
# Merging df1 and df2
dataset=pd.merge(df1,df2,on='Ind_ID',how='inner')
dataset.head()
```

Out[5]:

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

In the dataset, there are some missing values that need to be handled in further data analysis approach.

In [6]:

```
# Info about dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 241.9+ KB
```

The dataset has a total of 1548 rows and 19 columns.

In [7]:
```
# Checking duplicates in dataset
dataset[dataset.duplicated()].sum()
```

Out[7]:
```
Ind_ID             0.0
GENDER             0.0
Car_Owner          0.0
Propert_Owner      0.0
CHILDREN           0.0
Annual_income      0.0
Type_Income        0.0
EDUCATION          0.0
Marital_status     0.0
Housing_type       0.0
Birthday_count     0.0
Employed_days      0.0
Mobile_phone       0.0
Work_Phone         0.0
Phone              0.0
EMAIL_ID           0.0
Type_Occupation    0.0
Family_Members     0.0
label              0.0
dtype: float64
```

In [8]:
```
# Missing Values
dataset.isnull().sum()
```

Out[8]:
```
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
```

There are 7 null values in the GENDER column, 23 null values in the Annual_income column, 22 null values in the Birthday_count column, and 488 null values in the Type_Occupation column. It is important to handle these missing values during exploratory data analysis (EDA) to avoid any bias or errors in the results.

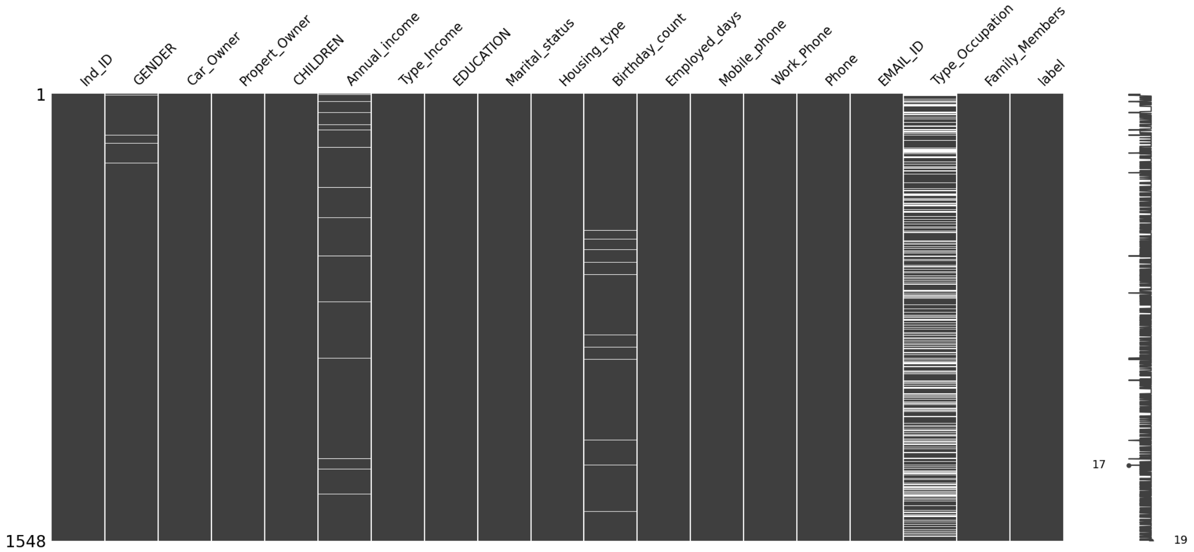
In [9]:
```
# Missing_values in Percentages
missing_values = dataset.isnull().sum()
total_values = dataset.count() + missing_values
missing_percentage = (missing_values / total_values) * 100
missing_percentage
```

Out[9]:
```
Ind_ID              0.000000
GENDER              0.452196
Car_Owner           0.000000
Propert_Owner       0.000000
CHILDREN            0.000000
Annual_income       1.485788
Type_Income         0.000000
EDUCATION           0.000000
Marital_status      0.000000
Housing_type        0.000000
Birthday_count      1.421189
Employed_days       0.000000
Mobile_phone        0.000000
Work_Phone          0.000000
Phone               0.000000
EMAIL_ID            0.000000
Type_Occupation    31.524548
Family_Members      0.000000
label               0.000000
dtype: float64
```

- The Gender column has 0.5% missing values. Annual income and Birthday count each have approximately 1.5% missing data.
- The Type_Occupation has the highest percentage of missing values, specifically 32% of the overall data.

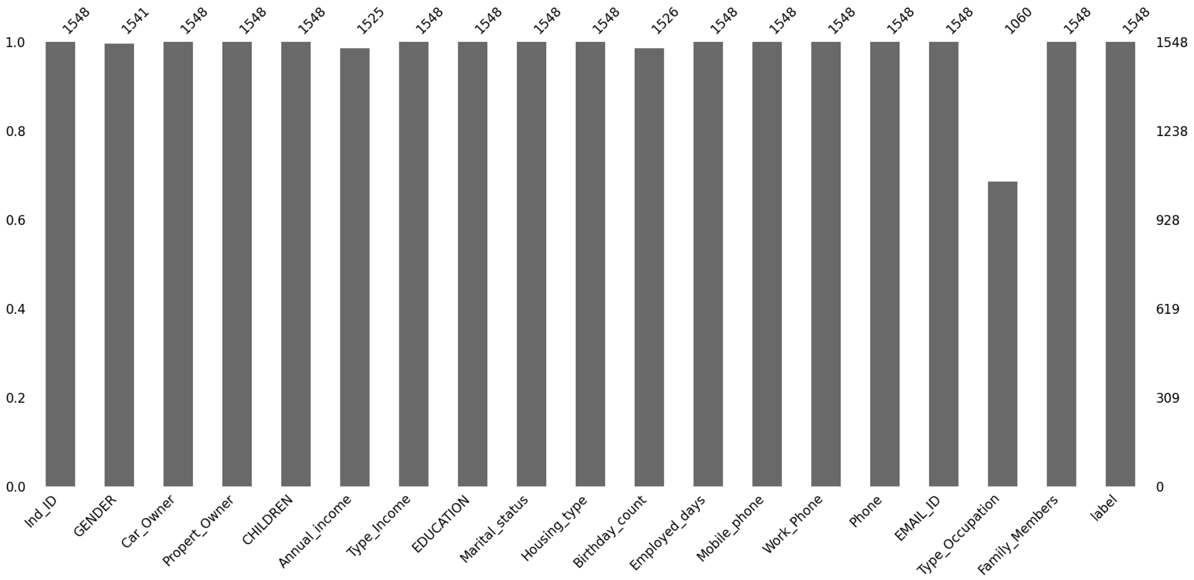
In [10]:
```
# Check missing values using matrix
msno.matrix(dataset)
```

Out[10]: <Axes: >



In [11]:
```
msno.bar(dataset)
```

Out[11]: <Axes: >



In [12]:
```
# Summary of dataset
dataset.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75
Ind_ID	1548.0	5.078920e+06	41717.587742	5008827.0	5045069.75	5078841.5	5115673.
CHILDREN	1548.0	4.127907e-01	0.776691	0.0	0.00	0.0	1.
Annual_income	1525.0	1.913993e+05	113252.997656	33750.0	121500.00	166500.0	225000.
Birthday_count	1526.0	-1.604034e+04	4229.503202	-24946.0	-19553.00	-15661.5	-12417.
Employed_days	1548.0	5.936469e+04	137808.062701	-14887.0	-3174.50	-1565.0	-431.
Mobile_phone	1548.0	1.000000e+00	0.000000	1.0	1.00	1.0	1.
Work_Phone	1548.0	2.080103e-01	0.406015	0.0	0.00	0.0	0.
Phone	1548.0	3.094315e-01	0.462409	0.0	0.00	0.0	1.
EMAIL_ID	1548.0	9.237726e-02	0.289651	0.0	0.00	0.0	0.
Family_Members	1548.0	2.161499e+00	0.947772	1.0	2.00	2.0	3.
label	1548.0	1.130491e-01	0.316755	0.0	0.00	0.0	0.

Imputation

In [13]:
```
# Fillna with mode of Type_Occupation
dataset['Type_Occupation'].fillna(dataset['Type_Occupation'].mode()[0],inplace=True
```

In [14]:
```
dataset['Type_Occupation'].value_counts()
```

Out[14]:
```
Laborers                 756
Core staff               174
Managers                 136
Sales staff              122
Drivers                   86
High skill tech staff     65
Medicine staff            50
Accountants               44
Security staff            25
Cleaning staff            22
Cooking staff             21
Private service staff     17
Secretaries                9
Low-skill Laborers         9
Waiters/barmen staff       5
HR staff                   3
IT staff                   2
Realty agents              2
Name: Type_Occupation, dtype: int64
```

In [15]:
```
# Filling NA with mode of GENDER Column
dataset['GENDER'].fillna(dataset['GENDER'].mode()[0],inplace=True)
```

In [16]:
```
# Imputing the Annul_income with Median
dataset['Annual_income'].fillna(dataset['Annual_income'].median(),inplace=True)
```

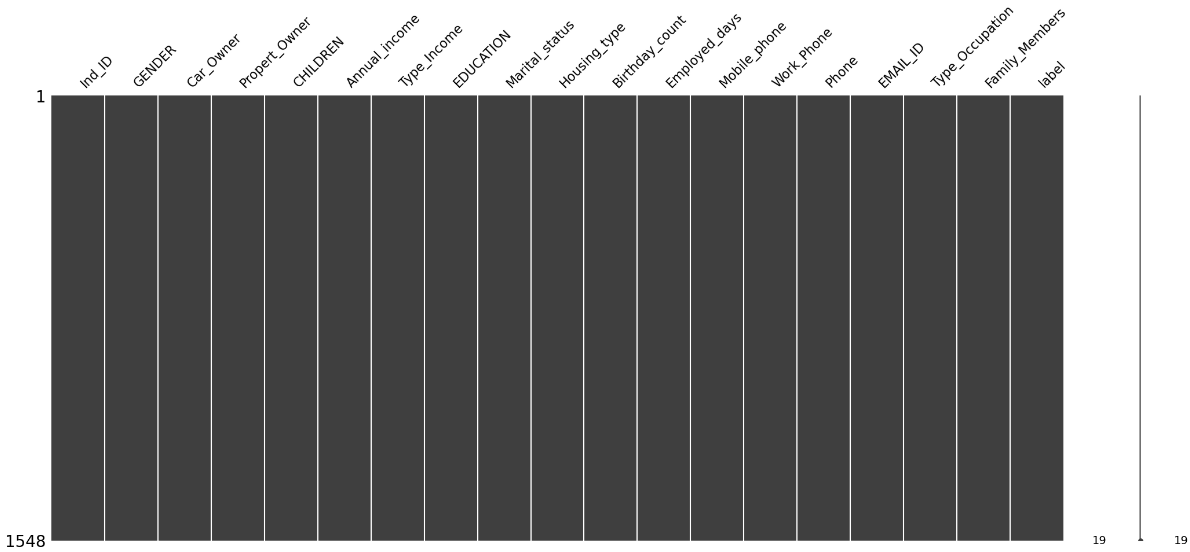
In [17]:
```
# Imputing the Birthday_count with Mean
dataset['Birthday_count'].fillna(dataset['Birthday_count'].mean(),inplace=True)
```

In [18]:
```
# Verifying null values again after imputation
dataset.isnull().sum()
```

Out[18]:
```
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
Birthday_count    0
Employed_days     0
Mobile_phone      0
Work_Phone        0
Phone             0
EMAIL_ID          0
Type_Occupation   0
Family_Members    0
label             0
dtype: int64
```

In [19]:
```
# Check missing values in matrix
msno.matrix(dataset)
```

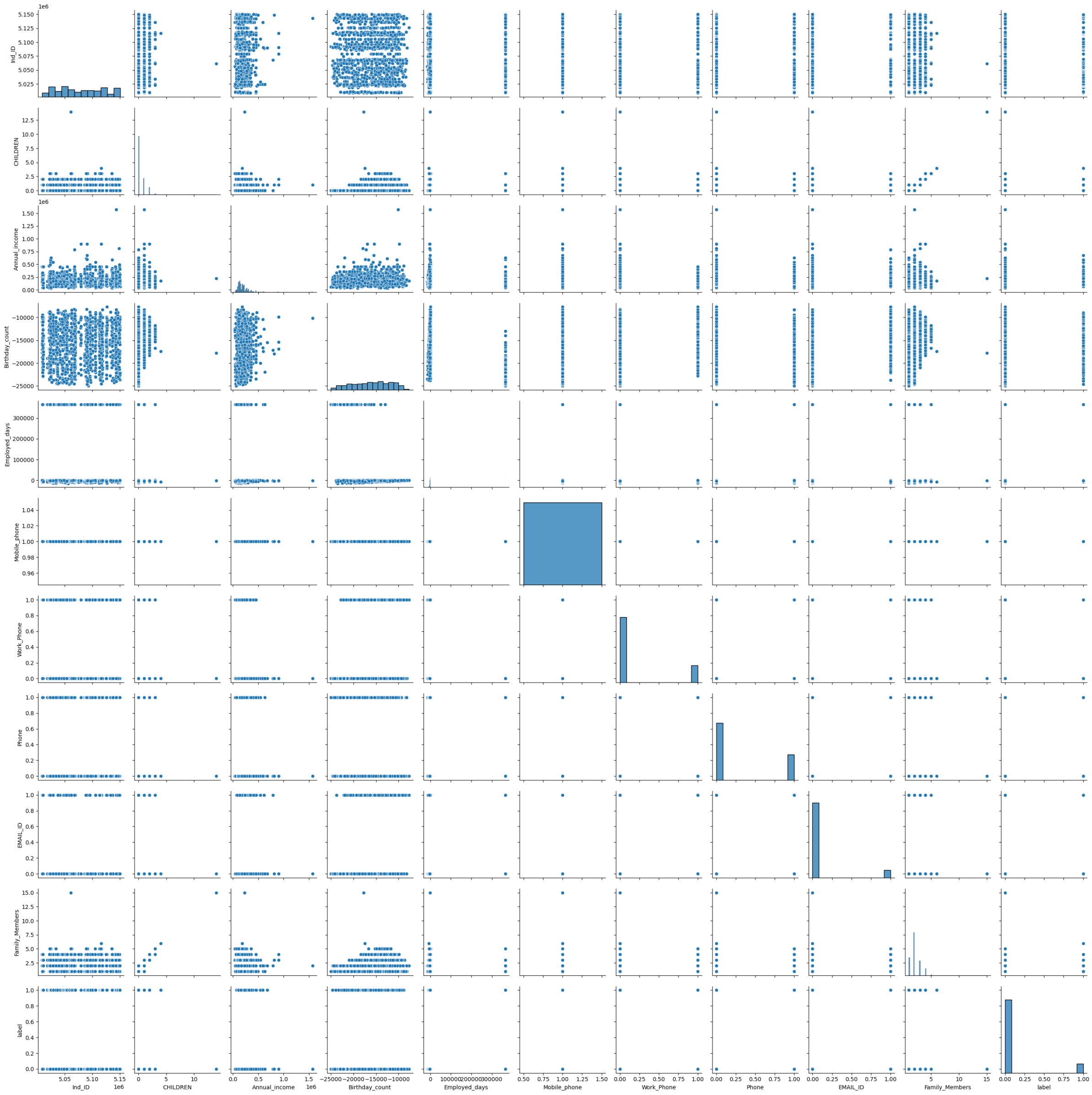
Out[19]:
```
<Axes: >
```



Univariate and Bivariate Analysis

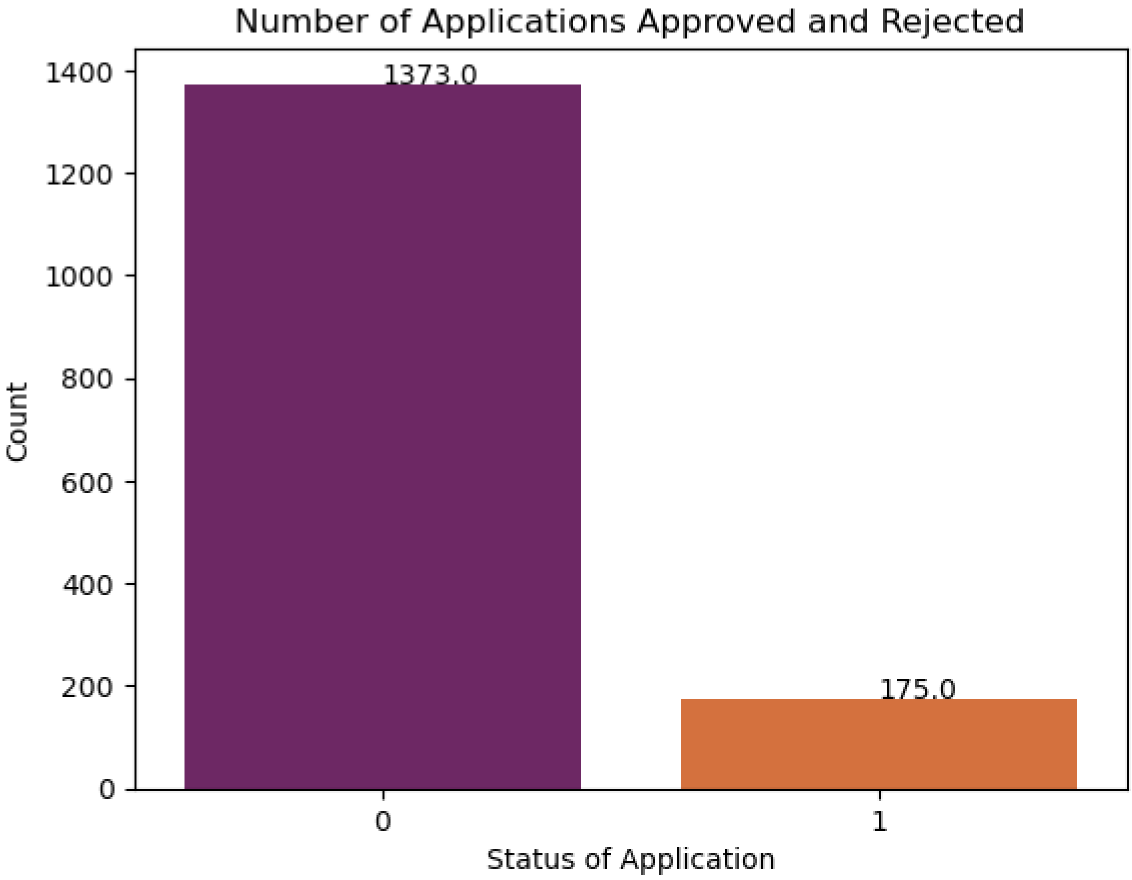
In [20]:
```
# Plot relationship between each variables
plt.figure(figsize=(30, 20))
sns.pairplot(dataset)
plt.show()
```

```
<Figure size 3000x2000 with 0 Axes>
```



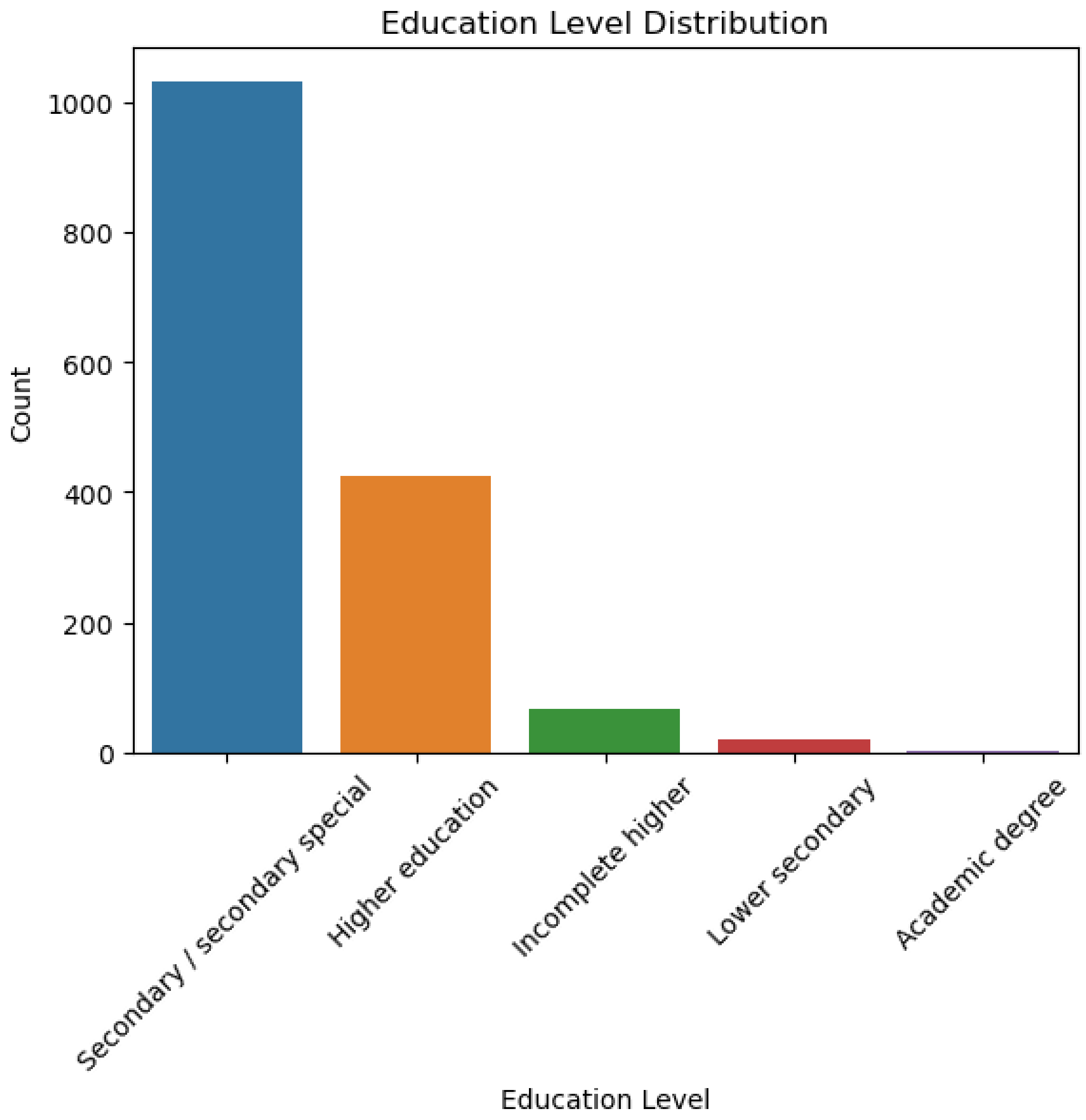
- There is strong correlation between family_Members and CHILDREN

In [21]:
```
# Number of applications Approved and Rejected
ax = sns.countplot(data=dataset,x='label',palette='inferno')

for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x()+0.4, p.get_height()+1))

plt.title('Number of Applications Approved and Rejected')
plt.xlabel('Status of Application')
plt.ylabel('Count')
plt.show()
```



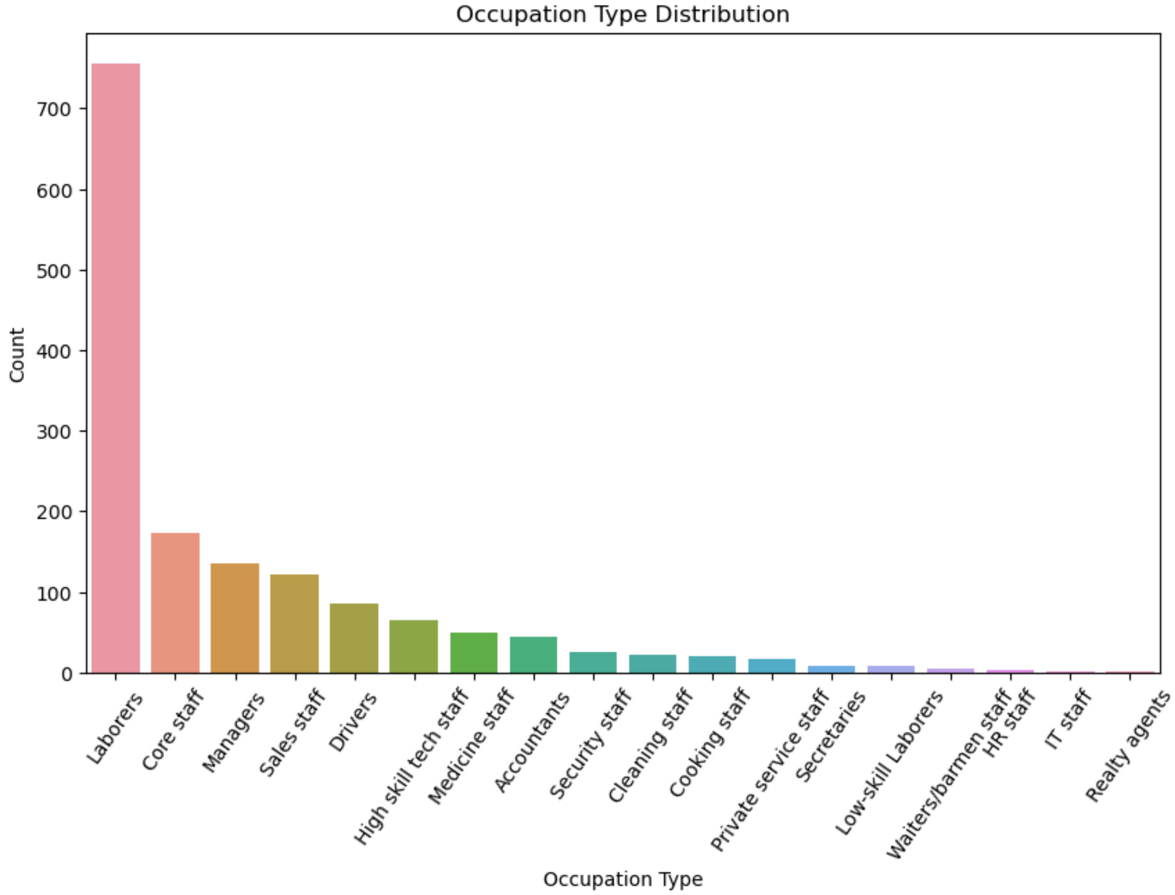
- Here, 0 means: Application is approved and 1 means: Application is rejected.
- Out of a total of 1548 applications, 1373 were Approved and 175 were rejected. The graph indicates that the majority (1373 applications) received approval, while a smaller number (175 applications) faced rejection.

In [22]:
```
# Education level distribution
sns.barplot(x=dataset['EDUCATION'].value_counts().index,y=dataset['EDUCATION'].valu
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.title('Education Level Distribution')
plt.xticks(rotation=45)
plt.show()
```



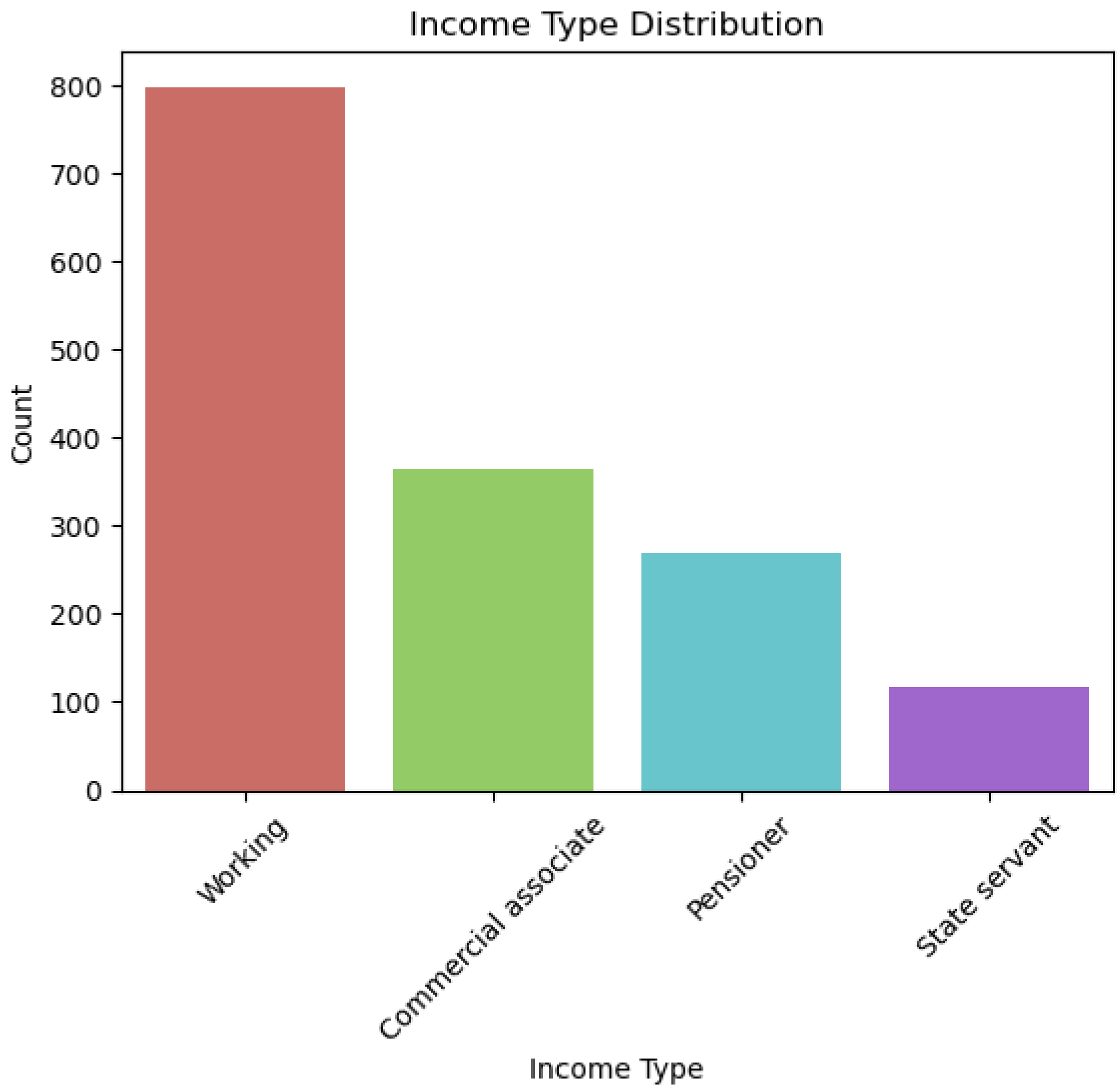
- The highest count in education levels is observed in the secondary/secondary special category, followed by the higher education category.
- The education levels with fewer instances are Incomplete Higher, Lower Secondary, and Academic Degree.

In [23]:
```
# Distribution of Occupation
plt.figure(figsize=(10, 6))
sns.barplot(x=dataset['Type_Occupation'].value_counts().index,y=dataset['Type_Occup
plt.xlabel('Occupation Type')
plt.ylabel('Count')
plt.title('Occupation Type Distribution')
plt.xticks(rotation=55)
plt.show()
```



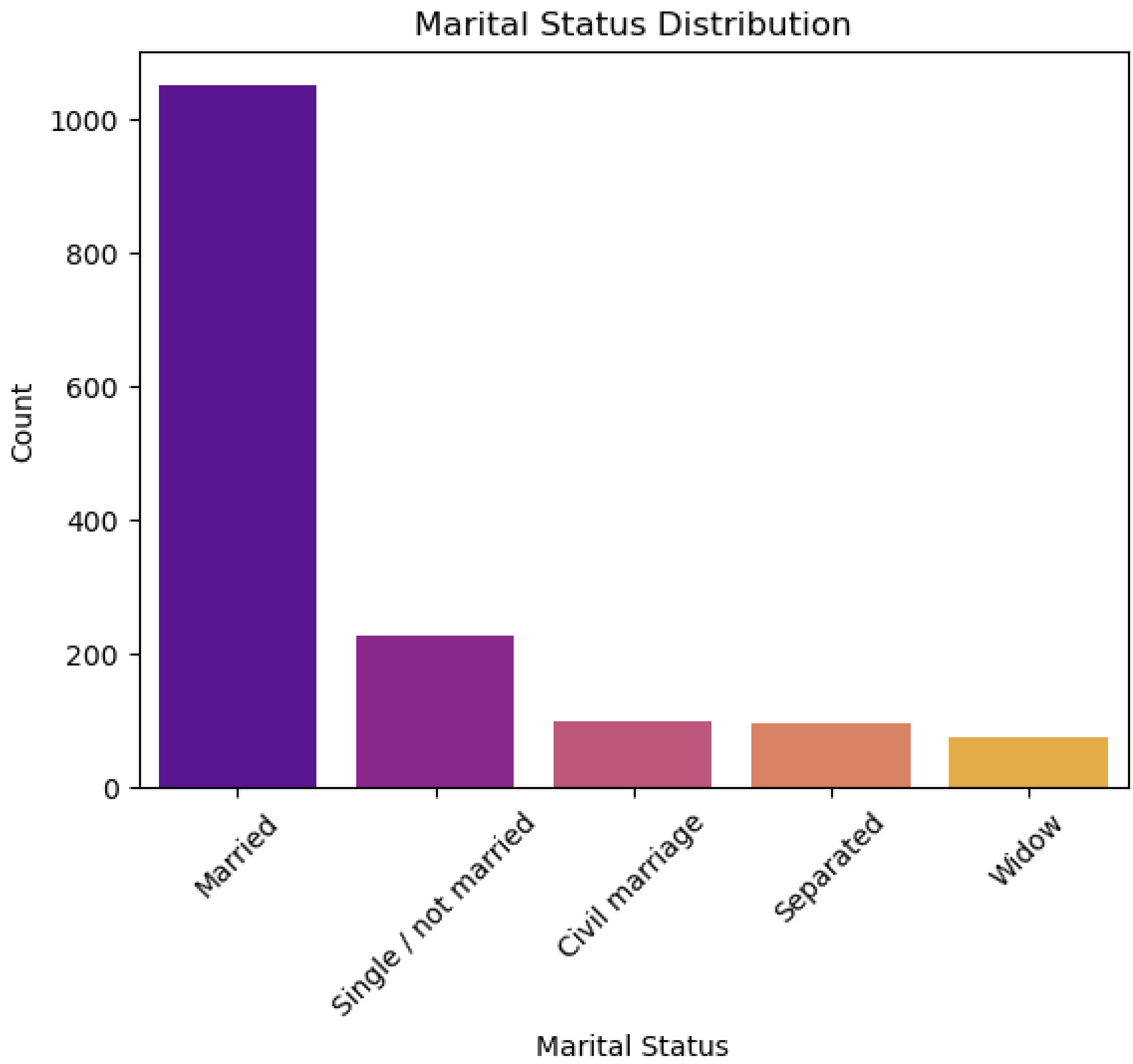
- Applicants from various occupational backgrounds, such as Laborers, Core Staff, Managers, Sales Staff, High-Skill Tech Staff, Drivers, Medicine Staff, and Accountants, show a substantial count.
- 'Laborers' have the highest count at around 700. There is a significant drop to 'Core Staff' and 'Managers' which are around 150 and 100 respectively. All other occupations listed have counts below 100.

In [24]:
```
# Income Type Distribution
sns.countplot(data=dataset,x='Type_Income', order=dataset['Type_Income'].value_cour
plt.xlabel('Income Type')
plt.ylabel('Count')
plt.title('Income Type Distribution')
plt.xticks(rotation=45)
plt.show()
```



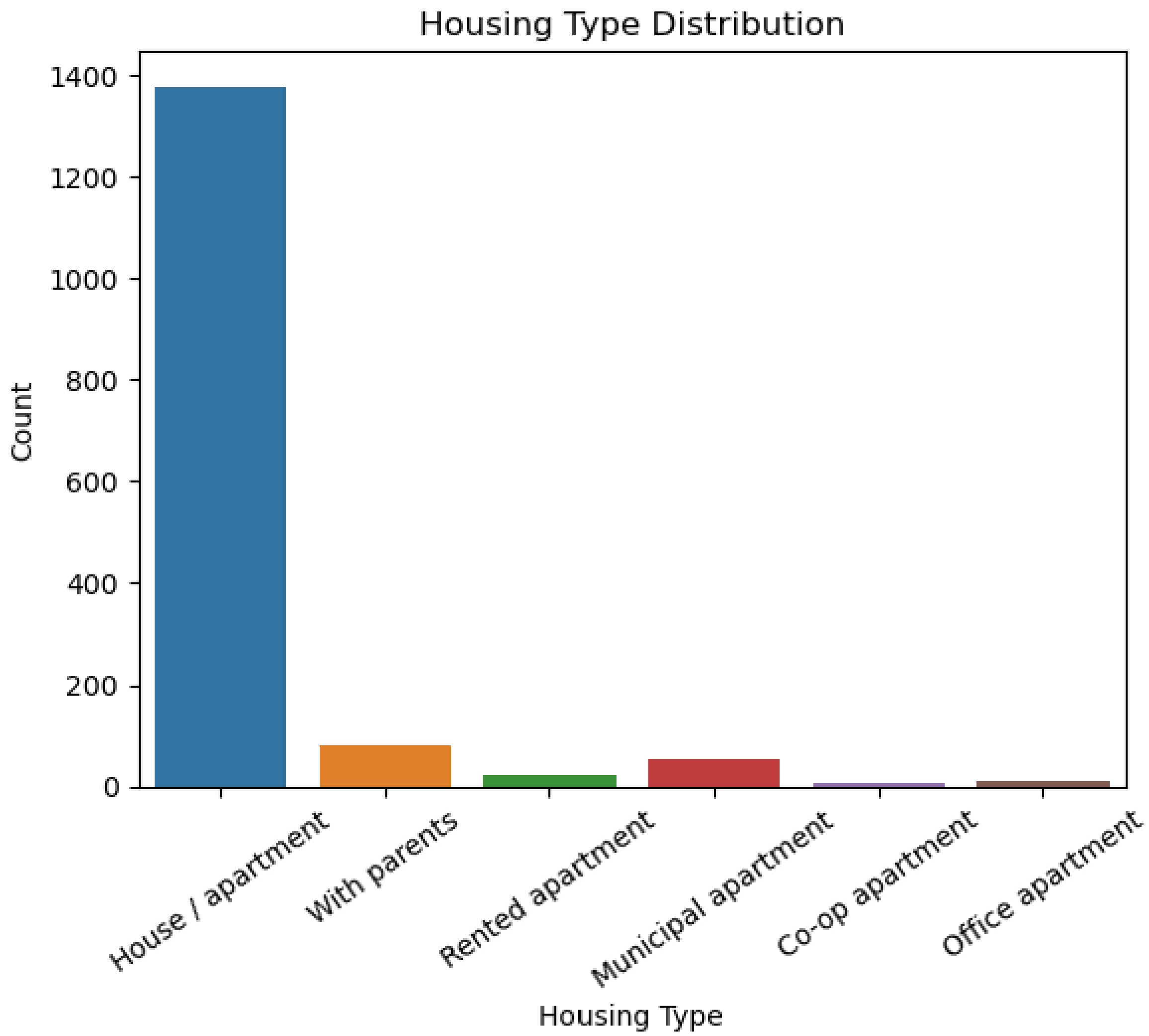
- Majority of the population applied for applicaton are from Working class followed by commercial associate, Pensioner and State servant

In [25]:
```
# Marital Status Distribution
sns.countplot(data=dataset,x='Marital_status',palette='plasma')
plt.xlabel('Marital Status')
plt.ylabel('Count')
plt.title('Marital Status Distribution')
plt.xticks(rotation=45)
plt.show()
```

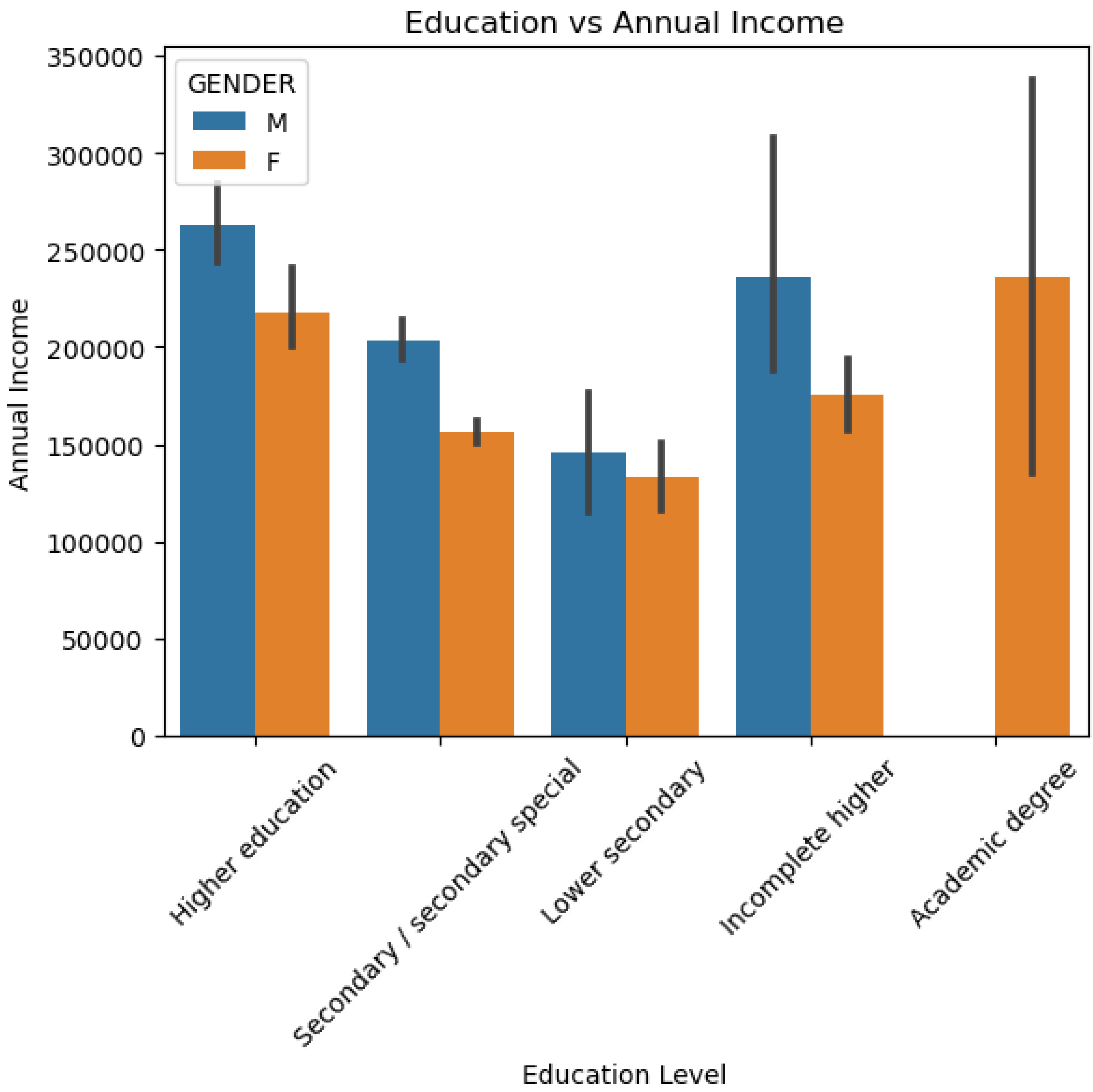


- Person who is married has a highest count approx 1000 applied for the application.

In [26]:
```
# House Type Distribution
sns.countplot(data=dataset,x='Housing_type')
plt.xlabel('Housing Type')
plt.ylabel('Count')
plt.title('Housing Type Distribution')
plt.xticks(rotation=35)
plt.show()
```

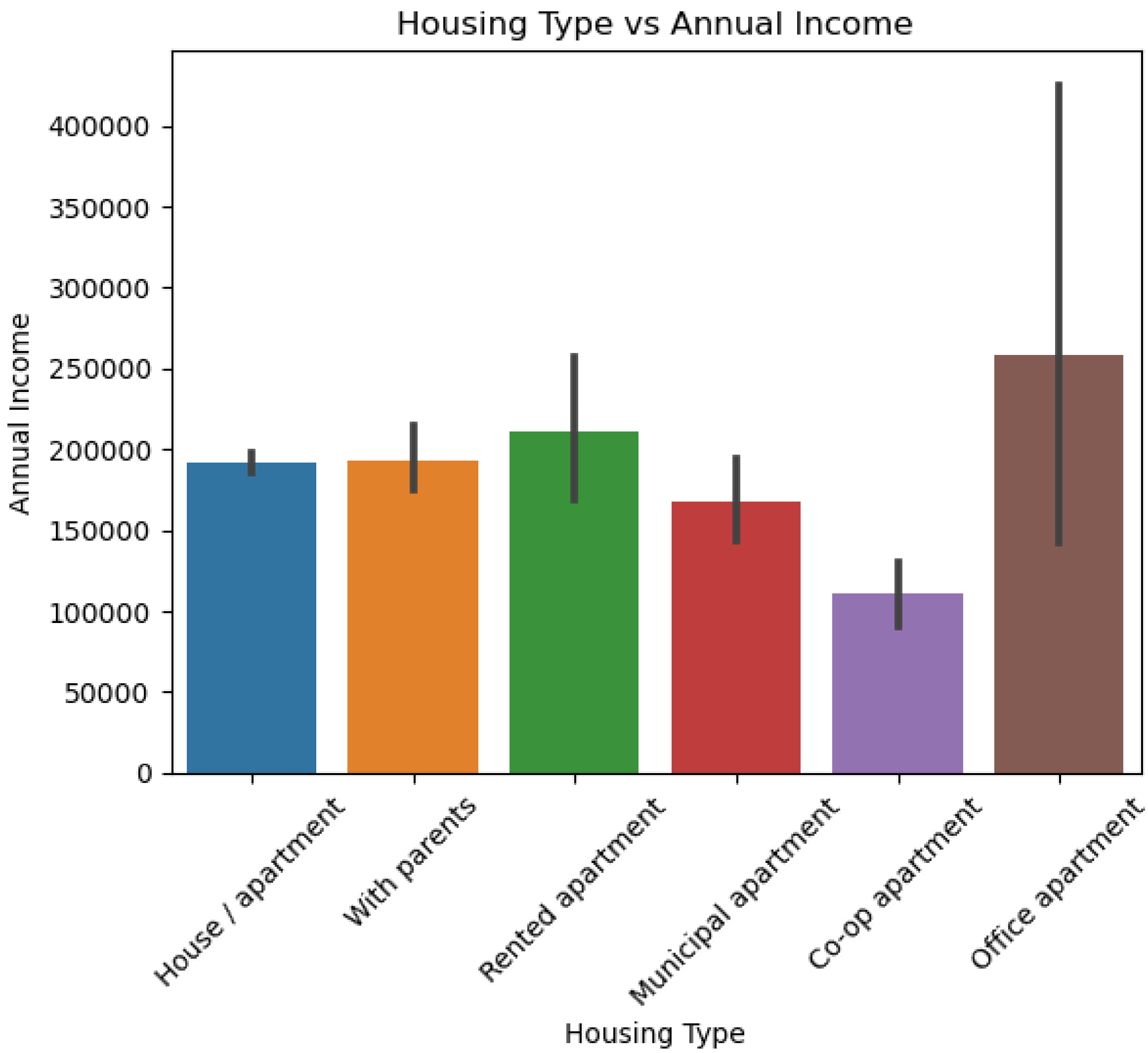


In [27]:
```
# Relationship between Education vs Annual_income
sns.barplot(data=dataset,x='EDUCATION',y='Annual_income',hue='GENDER')
plt.xlabel('Education Level')
plt.ylabel('Annual Income')
plt.title('Education vs Annual Income')
plt.xticks(rotation=45)
plt.show()
```



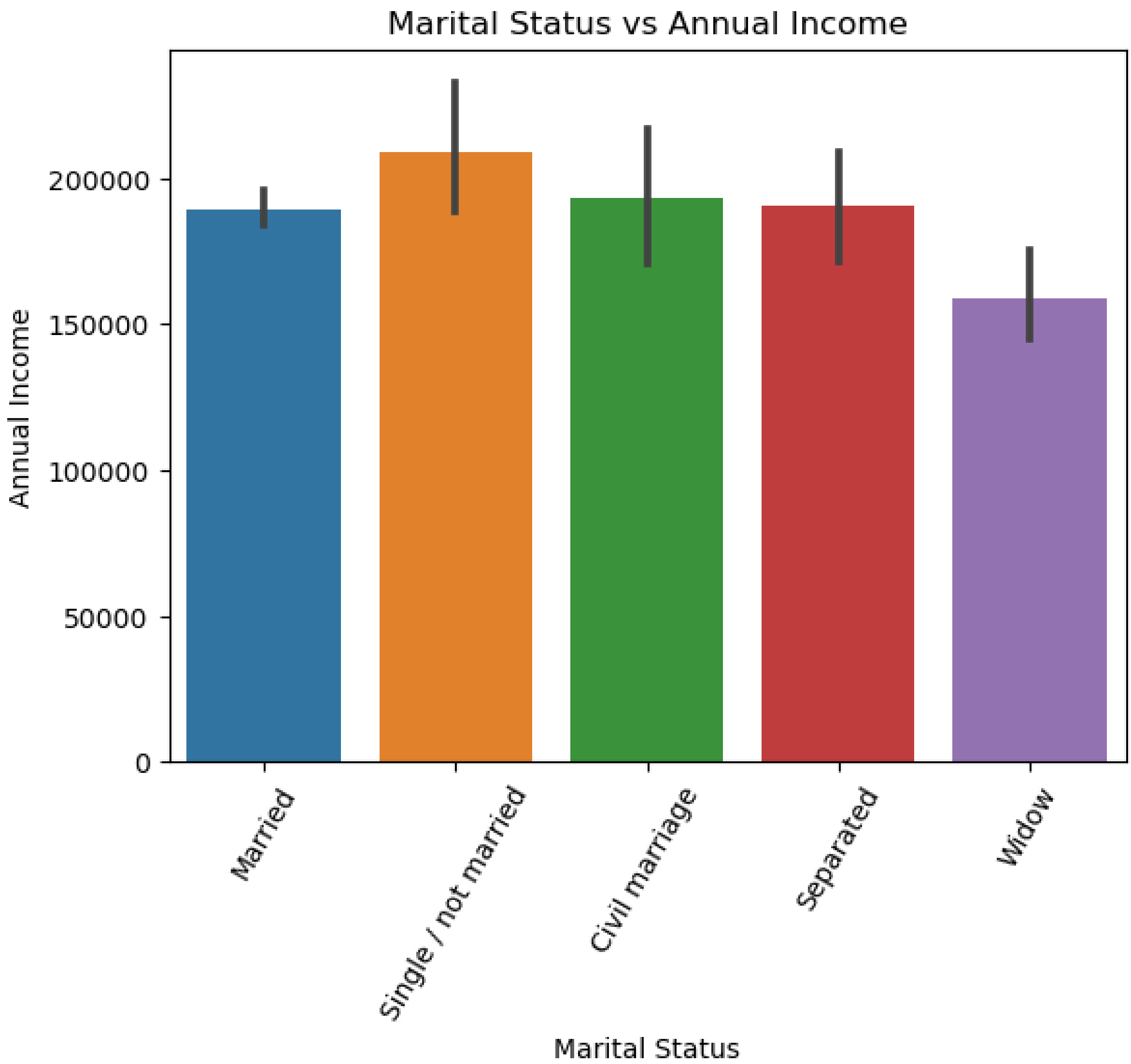
- It shows that across all education levels, males tend to have a higher income than females.

In [28]:
```
# Relationship between Housing_type vs Annual_income
sns.barplot(data=dataset,x='Housing_type',y='Annual_income')
plt.xlabel('Housing Type')
plt.ylabel('Annual Income')
plt.title('Housing Type vs Annual Income')
plt.xticks(rotation=45)
plt.show()
```



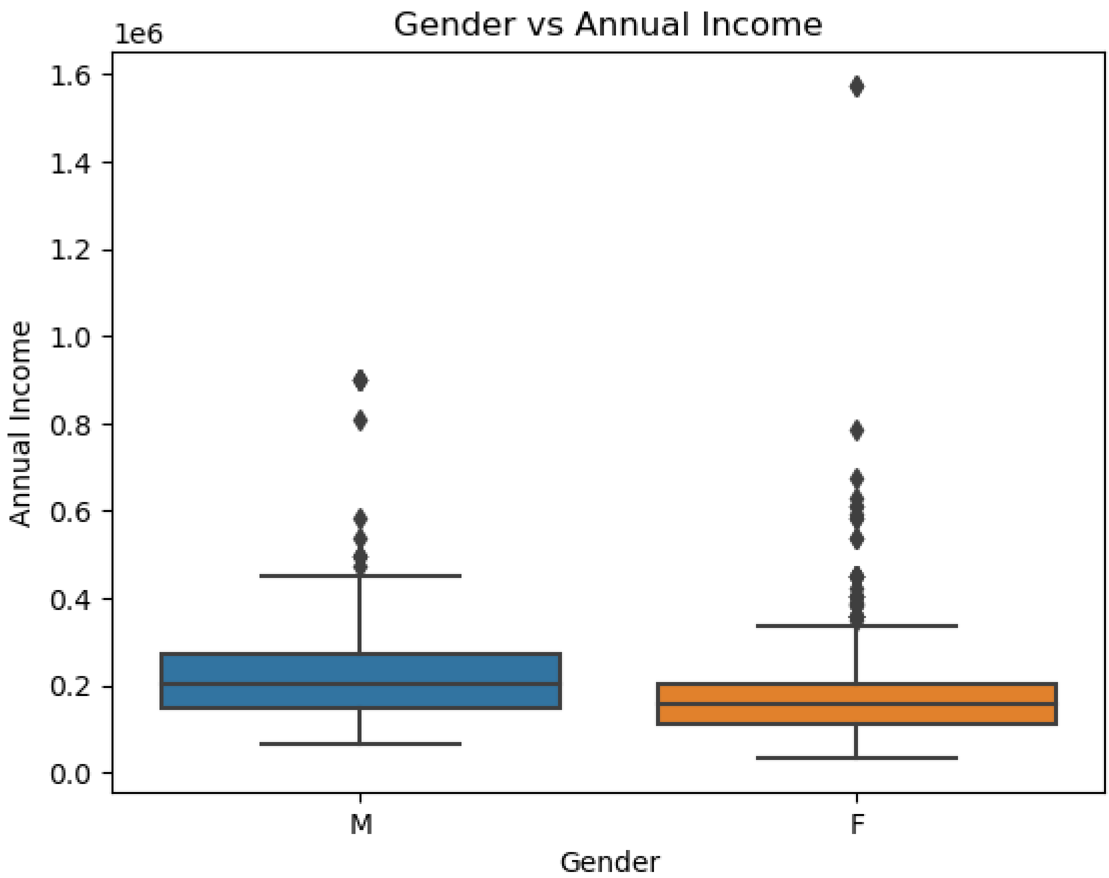
- Individuals living in an office apartment have the highest annual income, followed by those living with parents, in a house/apartment, rented apartment, and municipal apartment respectively. People living in co-op apartments have the lowest annual income.

In [29]:
```
# Relationship between Marital_Status vs Annual_income
sns.barplot(data=dataset,x='Marital_status',y='Annual_income')
plt.xlabel('Marital Status')
plt.ylabel('Annual Income')
plt.title('Marital Status vs Annual Income')
plt.xticks(rotation=60)
plt.show()
```

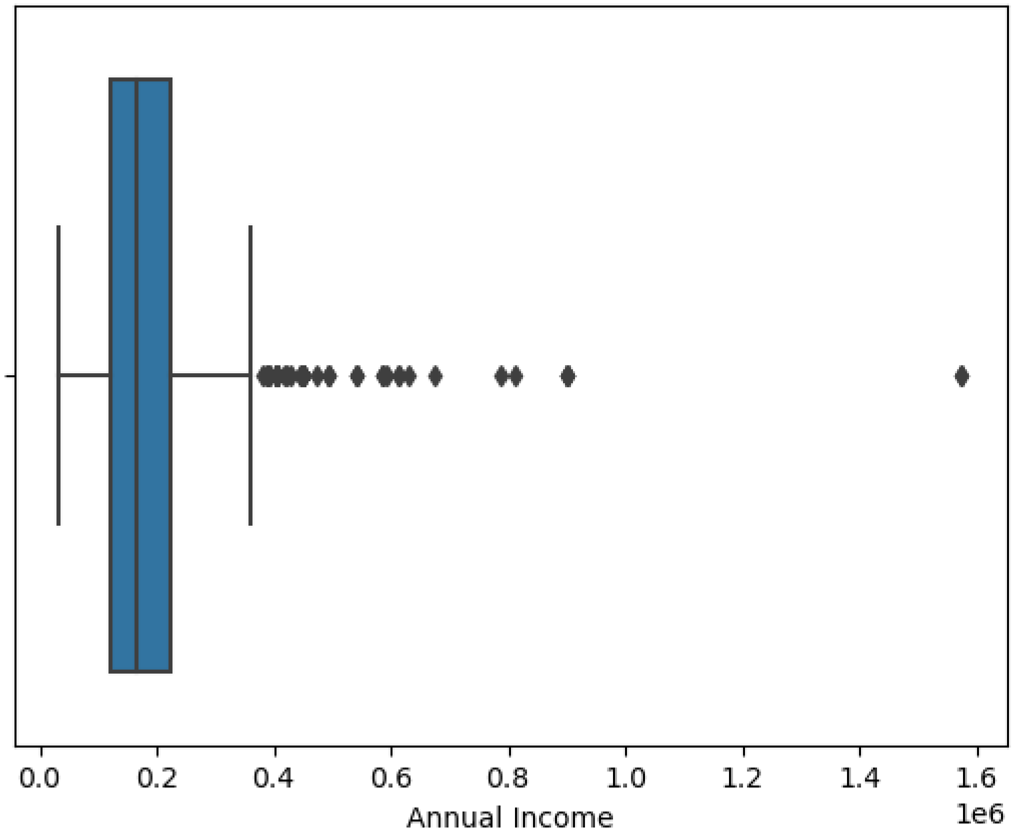


- All categories have similar income levels, with slight variations.

In [30]:
```
# Box plot between Gender and Annual income
sns.boxplot(data=dataset,x='GENDER',y='Annual_income')
plt.xlabel('Gender')
plt.ylabel('Annual Income')
plt.title('Gender vs Annual Income')
plt.show()
```

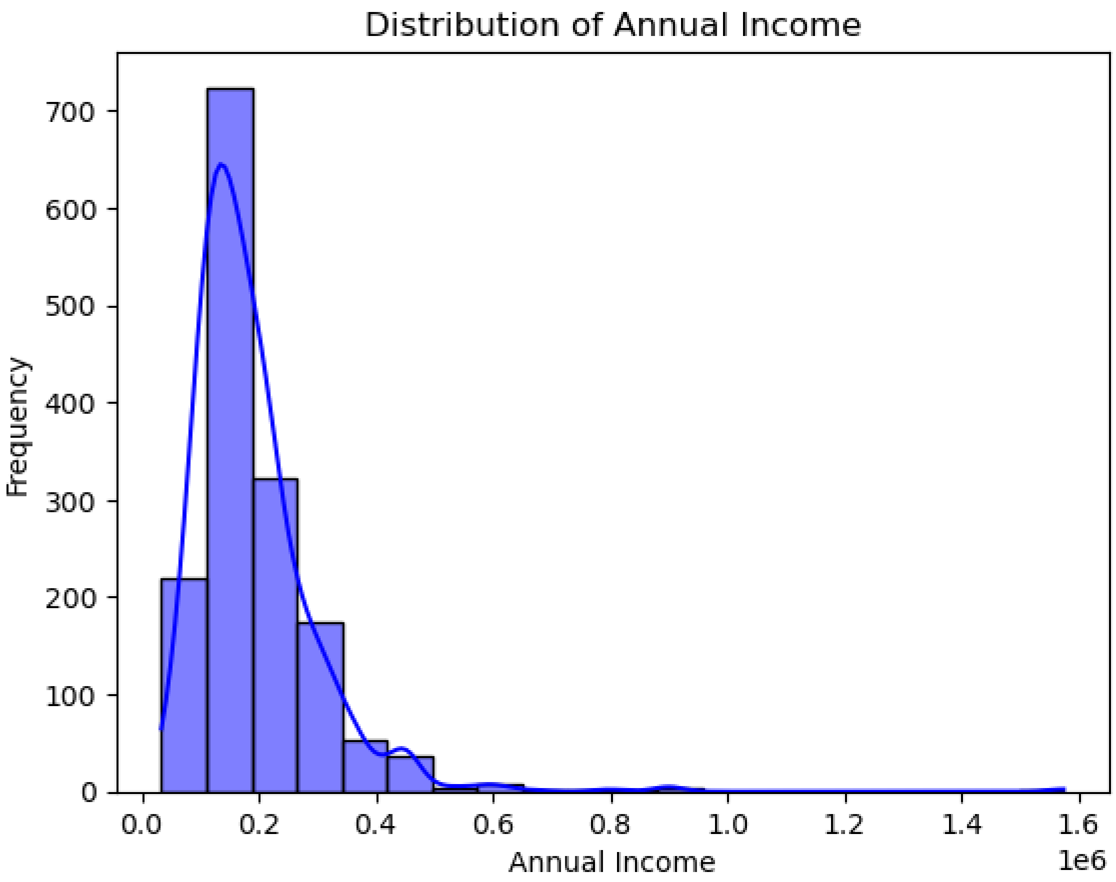


In [31]:
```
# Box plot to check outliers in Annual_income
sns.boxplot(data=dataset,x='Annual_income')
plt.xlabel('Annual Income')
plt.show()
```



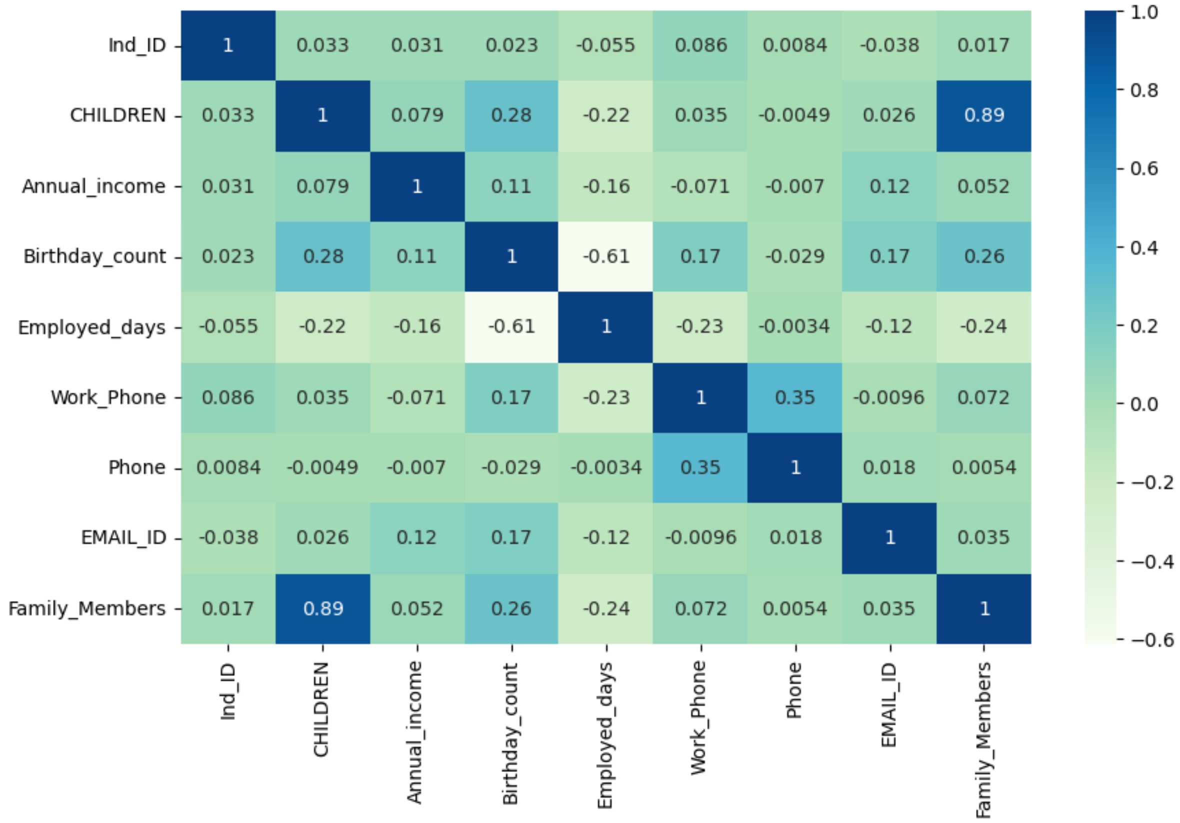
In [32]:
```
# Distribution of Annual Income
sns.histplot(data=dataset,x='Annual_income',kde=True,color='b',bins=20, edgecolor='
```

```
plt.xlabel('Annual Income')
plt.ylabel('Frequency')
plt.title('Distribution of Annual Income')
plt.show()
```



- This histogram shows that most people earn between 0 and approximately 400,000 annually.The graph also shows that there are very few people who earn more than 1 million annually.
- This leads to skewness toward right that is positive skewed.

In [33]:
```
# Heatmap
plt.figure(figsize=(10, 6))

# Drop Mobile_phone
dataset_without_mobile = dataset.drop(['Mobile_phone','label'], axis=1)

# Calculate the correlation matrix
corre = dataset_without_mobile.corr()
sns.heatmap(corre,annot=True,cmap='GnBu')
plt.show()
```



Hypotheses Testing (T-test)

Null Hypothesis (H0): There is no significant difference in mean annual income between approved and rejected credit card applications.

Alternate Hypothesis (H1): There is a significant difference in mean annual income between approved and rejected credit card applications.

In [34]:

```
import scipy.stats as stats

# Approved and rejected application
approved_income = dataset[dataset['label'] == 0]['Annual_income']
rejected_income = dataset[dataset['label'] == 1]['Annual_income']

# Perform independent t-test
t_stat, p_value = stats.ttest_ind(approved_income, rejected_income, equal_var=False

print("p_value:",p_value)
print("t-stat:",t_stat)


# Check if the p-value is less than the significance level (0.05)
if p_value < 0.05:
    print("Reject the null hypothesis")
else:
    print("Fail to reject the null hypothesis")
    print("Null hypotheses is correct")
```

```
p_value: 0.3488009493612162
t-stat: -0.9389401977532872
Fail to reject the null hypothesis
Null hypotheses is correct
```

The p-value obtained from the test is 0.3488099493162162, which is greater than 0.05. Therefore, we fail to reject the null hypothesis. This means that based on the data and the T-test, there is not enough evidence to conclude that there is a significant difference in mean annual income between approved and rejected credit card applications.

ANOVA

Null Hypothesis (H0): There is no significant difference in mean annual income among different education levels.

Alternate Hypothesis (H1): There is a significant difference in mean annual income among different education levels.

In [35]:
```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Fit ANOVA model
model = ols('Annual_income ~ EDUCATION', data=dataset).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Check the p-value in the ANOVA table
p_value = anova_table['PR(>F)'][0]

print("p_value",p_value)

# Check if the p-value is less than the significance level (0.05)
if p_value < 0.05:
    print("\nReject the null hypothesis")
else:
    print("Fail to reject the null hypothesis")
```

```
p_value 3.328700819043824e-21

Reject the null hypothesis
```

The p-value obtained from the test is extremely low (3.33e-21), which is much less than 0.05. Therefore, we reject the null hypothesis. This means that based on the data and the ANOVA test, there is strong evidence to conclude that there is a significant difference in mean annual income among different education levels.

In [36]:
```
dataset
```

Out[36]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EI
0	5008827	M	Y	Y	0	180000.0	Pensioner	
1	5009744	F	Y	N	0	315000.0	Commercial associate	
2	5009746	F	Y	N	0	315000.0	Commercial associate	
3	5009749	F	Y	N	0	166500.0	Commercial associate	
4	5009752	F	Y	N	0	315000.0	Commercial associate	
...	...	...	...	...	...	...	...	
1543	5028645	F	N	Y	0	166500.0	Commercial associate	
1544	5023655	F	N	N	0	225000.0	Commercial associate	
1545	5115992	M	Y	Y	2	180000.0	Working	
1546	5118219	M	Y	N	0	270000.0	Working	S
1547	5053790	F	Y	Y	0	225000.0	Working	

1548 rows × 19 columns

In [37]:

```
# Saving this cleaned data set for solving SQL queries
dataset.to_csv('cleaned_data.csv', index=False)
```

Feature Engineering

In [38]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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   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   1548 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  1548 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: 241.9+ KB
```

Feature Creation

In [39]:
```
# Create "Age_years" column from Birthday_count
dataset['Age_years']=round(-dataset['Birthday_count']/365.2425)
```

In [40]:
```
dataset.columns
```

Out[40]:
```
Index(['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_Members',
       'label', 'Age_years'],
      dtype='object')
```

In [41]:
```
dataset.head()
```

Out[41]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUC
0	5008827	M	Y	Y	0	180000.0	Pensioner	ed
1	5009744	F	Y	N	0	315000.0	Commercial associate	ed
2	5009746	F	Y	N	0	315000.0	Commercial associate	ed
3	5009749	F	Y	N	0	166500.0	Commercial associate	ed
4	5009752	F	Y	N	0	315000.0	Commercial associate	ed

In [42]:
```
# Checking unique values in Age_years
dataset.Age_years.unique()
```

Out[42]:
```
array([51., 37., 44., 61., 50., 52., 24., 46., 35., 33., 49., 43., 60.,
       30., 55., 29., 36., 56., 58., 65., 64., 42., 32., 26., 28., 45.,
       67., 25., 57., 59., 38., 53., 31., 40., 34., 47., 41., 63., 54.,
       48., 62., 27., 39., 66., 23., 68., 22., 21.])
```

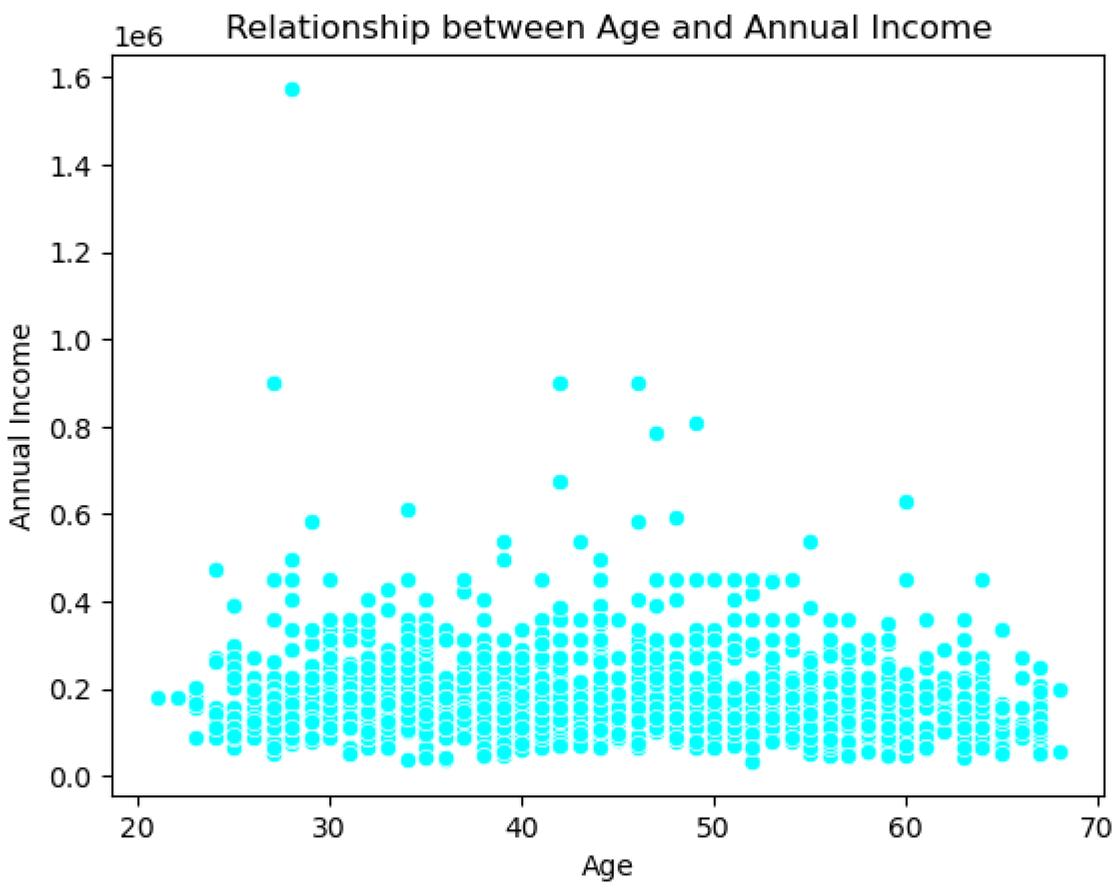
In [43]:
```
# Drop Birthday_count column
dataset.drop(columns='Birthday_count',inplace=True)
```

In [44]:
```
dataset.head()
```

Out[44]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUC
0	5008827	M	Y	Y	0	180000.0	Pensioner	ed
1	5009744	F	Y	N	0	315000.0	Commercial associate	ed
2	5009746	F	Y	N	0	315000.0	Commercial associate	ed
3	5009749	F	Y	N	0	166500.0	Commercial associate	ed
4	5009752	F	Y	N	0	315000.0	Commercial associate	ed

In [45]:
```
# Age vs Annual income
sns.scatterplot(data=dataset,x='Age_years',y='Annual_income',color='cyan')
plt.xlabel('Age')
plt.ylabel('Annual Income')
plt.title('Relationship between Age and Annual Income')
plt.show()
```



- This scatter plot shows that there is a weak positive correlation between age and annual income. The data points are scattered across the graph, indicating that there is no strong relationship between the two variables. However, the cluster of data points at the lower end of the income scale suggests that younger people tend to earn less than older people.

Scaling

In [46]:
```
from sklearn.preprocessing import StandardScaler

# Columns need to be scaled
scaled_columns= ['CHILDREN','Annual_income', 'Age_years','Employed_days', 'Mobile_p

# Initilization of StandardScaler
sc=StandardScaler()
dataset[scaled_columns]=sc.fit_transform(dataset[scaled_columns])
```

Encoding

In [47]:
```
# Columns need to be encoded
encoded_columns = ['GENDER','Car_Owner','Propert_Owner', 'Type_Income', 'EDUCATION'
dataset=pd.get_dummies(dataset,columns=encoded_columns,dtype='int')
```

In [48]:
```
dataset.head()
```

Out[48]:

	Ind_ID	CHILDREN	Annual_income	Employed_days	Mobile_phone	Work_Phone	Phone	E
0	5008827	-0.531645	-0.098116	2.220314	0.0	-0.512487	-0.669390	-
1	5009744	-0.531645	1.102824	-0.435171	0.0	1.951270	1.493899	-
2	5009746	-0.531645	1.102824	-0.435171	0.0	1.951270	1.493899	-
3	5009749	-0.531645	-0.218210	-0.435171	0.0	1.951270	1.493899	-
4	5009752	-0.531645	1.102824	-0.435171	0.0	1.951270	1.493899	-

5 rows × 55 columns

In [49]:
```
dataset.columns
```

Out[49]:
```
Index(['Ind_ID', 'CHILDREN', 'Annual_income', 'Employed_days', 'Mobile_phone',
       'Work_Phone', 'Phone', 'EMAIL_ID', 'Family_Members', 'label',
       'Age_years', 'GENDER_F', 'GENDER_M', 'Car_Owner_N', 'Car_Owner_Y',
       'Propert_Owner_N', 'Propert_Owner_Y',
       'Type_Income_Commercial associate', 'Type_Income_Pensioner',
       'Type_Income_State servant', 'Type_Income_Working',
       'EDUCATION_Academic degree', 'EDUCATION_Higher education',
       'EDUCATION_Incomplete higher', 'EDUCATION_Lower secondary',
       'EDUCATION_Secondary / secondary special',
       'Marital_status_Civil marriage', 'Marital_status_Married',
       'Marital_status_Separated', 'Marital_status_Single / not married',
       'Marital_status_Widow', 'Housing_type_Co-op apartment',
       'Housing_type_House / apartment', 'Housing_type_Municipal apartment',
       'Housing_type_Office apartment', 'Housing_type_Rented apartment',
       'Housing_type_With parents', 'Type_Occupation_Accountants',
       'Type_Occupation_Cleaning staff', 'Type_Occupation_Cooking staff',
       'Type_Occupation_Core staff', 'Type_Occupation_Drivers',
       'Type_Occupation_HR staff', 'Type_Occupation_High skill tech staff',
       'Type_Occupation_IT staff', 'Type_Occupation_Laborers',
       'Type_Occupation_Low-skill Laborers', 'Type_Occupation_Managers',
       'Type_Occupation_Medicine staff',
       'Type_Occupation_Private service staff',
       'Type_Occupation_Realty agents', 'Type_Occupation_Sales staff',
       'Type_Occupation_Secretaries', 'Type_Occupation_Security staff',
       'Type_Occupation_Waiters/barmen staff'],
      dtype='object')
```

Train Test Split

In [50]:
```
# Dropping Ind_ID column
dataset.drop(columns=['Ind_ID'],inplace=True)
```

In [51]:
```
# Independent Variables
X = dataset.drop(columns=['label'])

# Dependent/Target variable
y = dataset['label']
```

In [52]:
```
X
```

Out[52]:

	CHILDREN	Annual_income	Employed_days	Mobile_phone	Work_Phone	Phone	EMAIL_I
0	-0.531645	-0.098116	2.220314	0.0	-0.512487	-0.669390	-0.3190
1	-0.531645	1.102824	-0.435171	0.0	1.951270	1.493899	-0.3190
2	-0.531645	1.102824	-0.435171	0.0	1.951270	1.493899	-0.3190
3	-0.531645	-0.218210	-0.435171	0.0	1.951270	1.493899	-0.3190
4	-0.531645	1.102824	-0.435171	0.0	1.951270	1.493899	-0.3190
...	...	...	...	...	...	...	
1543	-0.531645	-0.218210	-0.446756	0.0	-0.512487	-0.669390	-0.3190
1544	-0.531645	0.302197	-0.439693	0.0	-0.512487	-0.669390	-0.3190
1545	2.044213	-0.098116	-0.448897	0.0	-0.512487	-0.669390	-0.3190
1546	-0.531645	0.702511	-0.435599	0.0	1.951270	1.493899	-0.3190
1547	-0.531645	0.302197	-0.451670	0.0	-0.512487	-0.669390	-0.3190

1548 rows × 53 columns

In [53]:
```
y
```

Out[53]:
```
0       1
1       1
2       1
3       1
4       1
       ..
1543    0
1544    0
1545    0
1546    0
1547    0
Name: label, Length: 1548, dtype: int64
```

In [55]:
```
# Train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=42
```

Section 4:

Machine Learning Models

Logistic Regression Model

In [56]:
```
# Import libraries for Logistic Regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score
```

In [57]:
```
# Initilization of LogisticRegression
lr_model=LogisticRegression()
lr_model.fit(X_train,y_train)
```

Out[57]:
```
▾ LogisticRegression
LogisticRegression()
```

In [58]:
```
# Score of Train dataset
lr_model.score(X_train,y_train)
```

Out[58]:
```
0.8818097876269622
```

In [59]:
```
# score of Test dataset
lr_model.score(X_test,y_test)
```

Out[59]:
```
0.9096774193548387
```

In [60]:
```
# Prediction
y_pred = lr_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}\n")

# Classification report
lr_report=classification_report(y_test,y_pred)
print("Classification Report:\n", lr_report)

# Confusion Matrix
conf_matrix=confusion_matrix(y_test,y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.91

Classification Report:
               precision    recall  f1-score   support

           0       0.91      1.00      0.95       422
           1       1.00      0.02      0.05        43

    accuracy                           0.91       465
   macro avg       0.95      0.51      0.50       465
weighted avg       0.92      0.91      0.87       465

Confusion Matrix:
 [[422   0]
 [ 42   1]]
```

Decision Tree Classification Model

In [61]:
```
# Import libraries for Decision Tree Classification model
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.model_selection import GridSearchCV
```

In [62]:
```
# Create decision tree classifier
DTC_model=DecisionTreeClassifier()

# Train the classifier
DTC_model.fit(X_train, y_train)

# Predict on the test set
y_pred = DTC_model.predict(X_test)

# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}\n")

# Classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.88

Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.92      0.93       422
           1       0.38      0.47      0.42        43

    accuracy                           0.88       465
   macro avg       0.66      0.69      0.68       465
weighted avg       0.89      0.88      0.89       465

Confusion Matrix:
 [[390  32]
 [ 23  20]]
```

In [63]:
```
# Hyperparameter tuning for selecting best parmeters for Decision tree classifier
param_grid = {
    'max_depth': [4,5, 10, 15, 20],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion': ['gini', 'entropy','log_loss']
}

# Perform GridSearchCV
grid_search = GridSearchCV(DTC_model, param_grid, cv=5,verbose=1)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 360 candidates, totalling 1800 fits
```

Out[63]:
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [64]:
```
# Best parameter
grid_search.best_params_
```

Out[64]:
```
{'criterion': 'log_loss',
 'max_depth': 5,
 'max_features': 'auto',
 'min_samples_leaf': 2,
 'min_samples_split': 2}
```

In [65]:
```
# Best score
grid_search.best_score_
```

Out[65]:
```
0.8808841099163679
```

In [66]:
```
# Best Estimator
best_DTC_model=grid_search.best_estimator_
```

```
best_DTC_model
```

Out[66]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='log_loss', max_depth=5, max_features='a
uto',
                       min_samples_leaf=2)
```

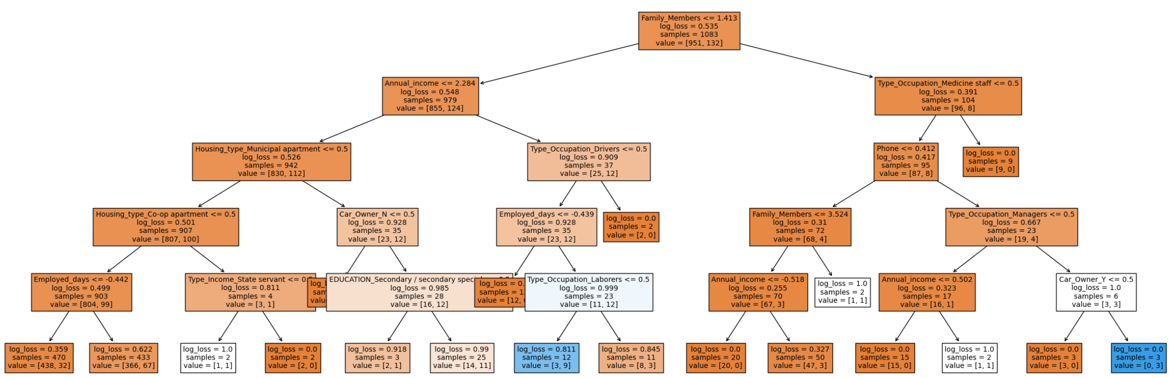
In [67]:

```
# Predict on the test set based on best parameters
y_pred_best = best_DTC_model.predict(X_test)

# Calculate accuracy
accuracy_best_parm = accuracy_score(y_test, y_pred_best)
print(f"Accuracy: {accuracy_best_parm:.2f}")

# Classification Report
print("\nClassification report:\n",classification_report(y_test,y_pred_best))

# Confusion matrix
print("Confusion matrix:\n",confusion_matrix(y_test,y_pred_best))
```

```
Accuracy: 0.91

Classification report:
               precision    recall  f1-score   support

           0       0.91      1.00      0.95       422
           1       0.50      0.02      0.04        43

    accuracy                           0.91       465
   macro avg       0.70      0.51      0.50       465
weighted avg       0.87      0.91      0.87       465

Confusion matrix:
 [[421   1]
 [ 42   1]]
```

In [68]:

```
# Plot the decision tree
plt.figure(figsize=(30,10))
plot_tree(best_DTC_model, feature_names=X.columns, filled=True,fontsize=10)
plt.show()
```



Random Forest Classification Model

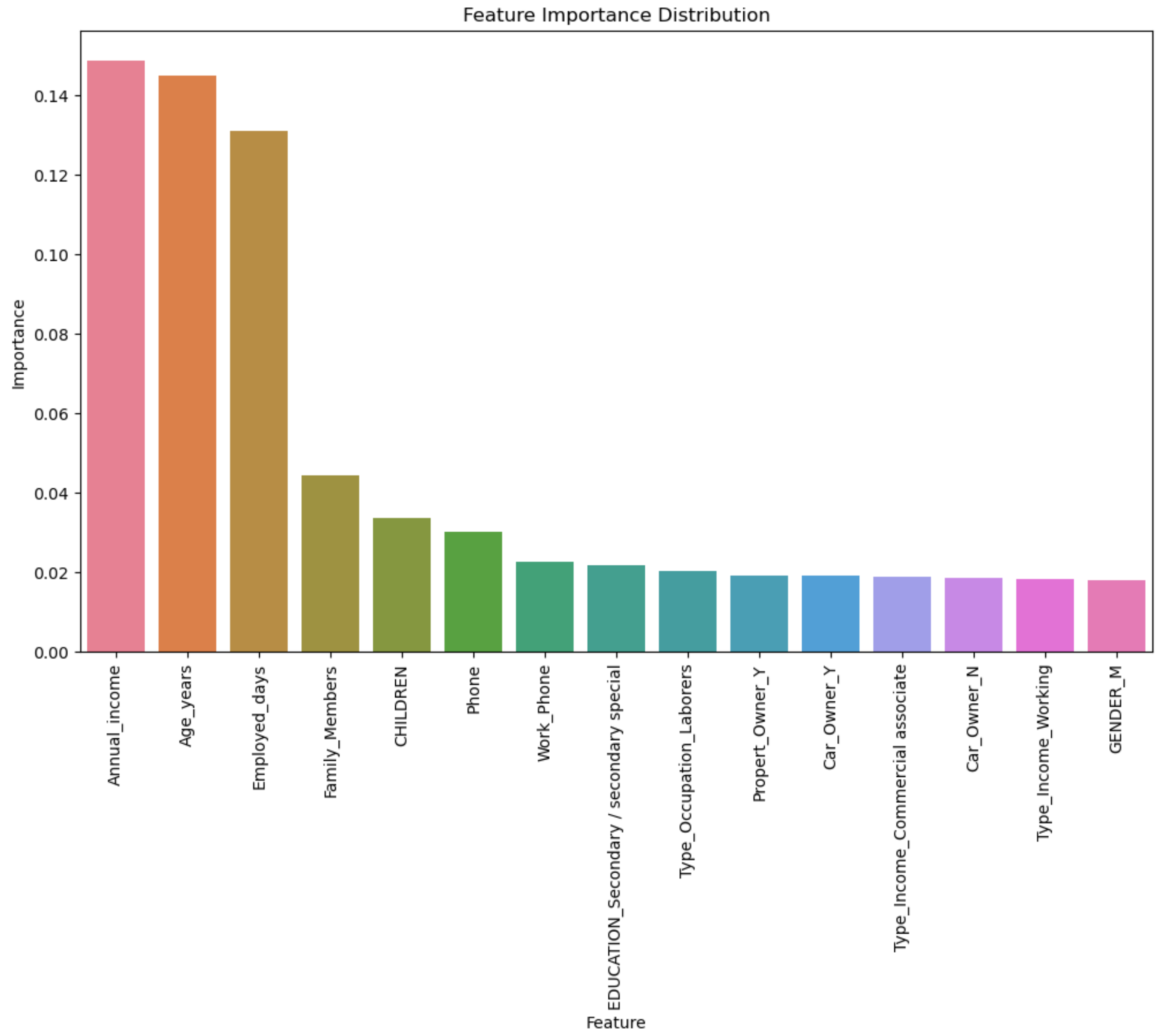
In [69]:

```
# Import libraries for Random Forest Classification model
from sklearn.ensemble import RandomForestClassifier
```

In [70]:
```
# Initialized the random forest classifier
RFC_model=RandomForestClassifier()

# Train the classifier
RFC_model.fit(X_train, y_train)

# Predict on the test set
y_pred = RFC_model.predict(X_test)

# Evaluate the model accuracy
accuracy_RFC = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy_RFC:.2f}")

# classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)

# confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.94
Classification Report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97       422
           1       0.89      0.40      0.55        43

    accuracy                           0.94       465
   macro avg       0.92      0.70      0.76       465
weighted avg       0.94      0.94      0.93       465

Confusion Matrix:
 [[420   2]
 [ 26  17]]
```

In [71]:
```
# Feature Importance
feature_importances = RFC_model.feature_importances_

sorted_indexes = feature_importances.argsort()[-15:][::-1] # sorting top 15 feature

cols = X_train.columns[sorted_indexes]

plt.figure(figsize=(12, 7))
sns.barplot(x=cols, y=feature_importances[sorted_indexes],palette='husl')
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importance Distribution')
plt.xticks(rotation=90)
plt.show()
```



- The features “Annual Income”, “Age_years”, and “Employed_days” have higher importance values, indicating they are significant predictors or contributors in the model building.
- Family_Members and CHILDREN do have some significant importance.

Gradient Boosting Classification Model

In [72]:
```
# Import libraries for Gradient Boosting Classification Model
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV
```

In [73]:
```
# Initilize the Gradient boosting classifier
GBC_model=GradientBoostingClassifier()
GBC_model.fit(X_train,y_train)
```

Out[73]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [74]:
```
# Prediction on test set
y_pred=GBC_model.predict(X_test)

# Evaluate the model accuracy
accuracy_GBC = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
# classification report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", class_report)

# confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.88

Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.98      0.95       422
           1       0.47      0.16      0.24        43

    accuracy                           0.91       465
   macro avg       0.69      0.57      0.60       465
weighted avg       0.88      0.91      0.88       465

Confusion Matrix:
 [[414   8]
 [ 36   7]]
```

In [75]:
```
# Hyperparameter tuning for selecting best parmeters for Gradient Boosting classifi
param={
    'criterion': ['friedman_mse', 'squared_error'],
    'n_estimators' : [50,100,150],
    'learning_rate': [0.01,0.1,0.2,0.5],
    'max_depth' : [2,3,4]
}

# Perform RandomizedSearchCV
Rand_GBC_search=RandomizedSearchCV(GBC_model,param,cv=5,scoring='accuracy')
Rand_GBC_search.fit(X_train,y_train)
```

Out[75]:
```
▸ RandomizedSearchCV
▸ estimator: GradientBoostingClassifier
  ▸ GradientBoostingClassifier
```

In [76]:
```
# Best Parameter
Rand_GBC_search.best_params_
```

Out[76]:
```
{'n_estimators': 150,
 'max_depth': 3,
 'learning_rate': 0.2,
 'criterion': 'squared_error'}
```

In [77]:
```
# Best Score
Rand_GBC_search.best_score_
```

Out[77]:
```
0.8882616487455198
```

In [78]:
```
# Best Estimator
best_GBC_model=Rand_GBC_search.best_estimator_
best_GBC_model
```

Out[78]:

```
GradientBoostingClassifier
GradientBoostingClassifier(criterion='squared_error', learning_rate=0.2,
                           n_estimators=150)
```

In [79]:

```
# Predict on the test set based on best parameters
y_pred_best = best_GBC_model.predict(X_test)

# Calculate accuracy
accuracy_best_parm = accuracy_score(y_test, y_pred_best)
print(f"Accuracy: {accuracy_best_parm:.2f}")

# Classification Report
print("\nClassification report:\n",classification_report(y_test,y_pred_best))

# Confusion matrix
print("Confusion matrix:\n",confusion_matrix(y_test,y_pred_best))
```

```
Accuracy: 0.91

Classification report:
               precision    recall  f1-score   support

           0       0.94      0.97      0.95       422
           1       0.56      0.35      0.43        43

    accuracy                           0.91       465
   macro avg       0.75      0.66      0.69       465
weighted avg       0.90      0.91      0.90       465

Confusion matrix:
 [[410  12]
 [ 28  15]]
```

XGBoost Model

In [80]:

```
# Import libraries for XGBoost Model
from xgboost import XGBClassifier
```

In [81]:

```
# Create an XGBoost Classifier
XGB_classifier=XGBClassifier(n_estimators=2, max_depth=2, learning_rate=1, objectiv
XGB_classifier.fit(X_train,y_train)
```

Out[81]:

```
XGBClassifier
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree
=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthw
ise',
              importance_type=None, interaction_constraints='', learnin
g_rate=1,
              max_bin=256, max_cat_to_onehot=4, max_delta_step=0, max_d
epth=2,
              max_leaves=0, min_child_weight=1, missing=nan,
```

In [82]:

```
# Prediction on the test set
y_pred = XGB_classifier.predict(X_test)
```

```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Classification report
report = classification_report(y_test, y_pred)
print("\nClassification report:\n",report)

# Confusion matrix
print("Confusion matrix:\n",confusion_matrix(y_test,y_pred))
```

```
Accuracy: 0.91

Classification report:
               precision    recall  f1-score   support

           0       0.91      1.00      0.95       422
           1       1.00      0.02      0.05        43

    accuracy                           0.91       465
   macro avg       0.95      0.51      0.50       465
weighted avg       0.92      0.91      0.87       465

Confusion matrix:
 [[422   0]
 [ 42   1]]
```

ML Model Comparison

In [97]:

```
data={
    'Model':['Logistic Regression Model','Decision Tree Model','Decision Tree Model
    'Accuracy':[0.91,0.87,0.91,0.94,0.87,0.92,0.91],
    'Precision (Class 0)':[0.91,0.94,0.91,0.94,0.92,0.94,0.91],
    'Precision (Class 1)':[1.00,0.36,1.00,0.89,0.47,0.58,1.00],
    'Recall (Class 0)':[1.00,0.91,1.00,1.00,0.98,0.97,1.00],
    'Recall (Class 1)':[0.02,0.47,0.05,0.40,0.16,0.35,0.02],
    'F1-Score (Class 0)':[0.95,0.93,0.95,0.97,0.95,0.95,0.95],
    'F1-Score (Class 1)':[0.05,0.40,0.09,0.55,0.24,0.43,0.05]
}
```

In [98]:

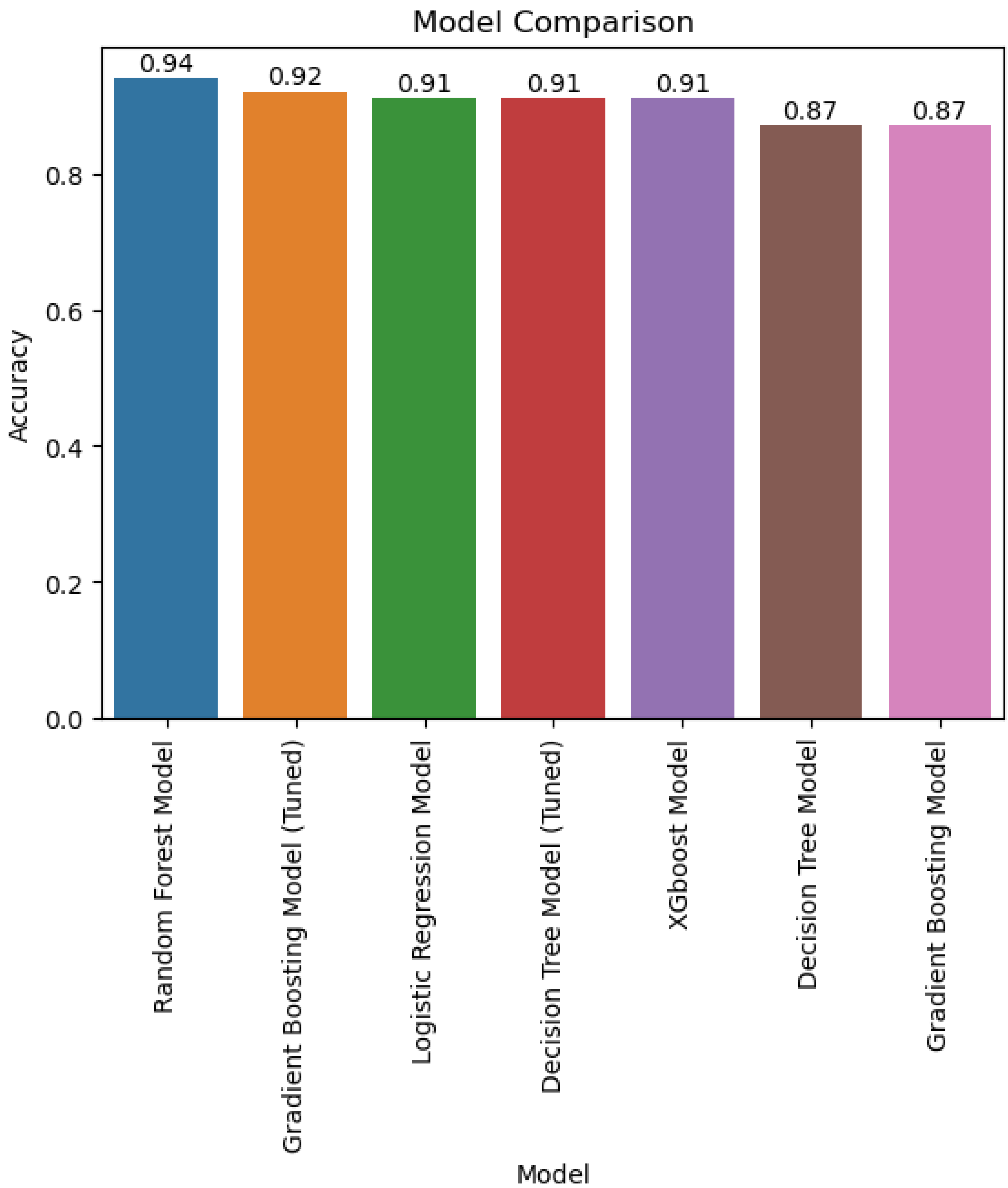
```
model=pd.DataFrame(data)
model
```

Out[98]:

	Model	Accuracy	Precision (Class 0)	Precision (Class 1)	Recall (Class 0)	Recall (Class 1)	F1-Score (Class 0)	F1-Score (Class 1)
0	Logistic Regression Model	0.91	0.91	1.00	1.00	0.02	0.95	0.05
1	Decision Tree Model	0.87	0.94	0.36	0.91	0.47	0.93	0.40
2	Decision Tree Model (Tuned)	0.91	0.91	1.00	1.00	0.05	0.95	0.09
3	Random Forest Model	0.94	0.94	0.89	1.00	0.40	0.97	0.55
4	Gradient Boosting Model	0.87	0.92	0.47	0.98	0.16	0.95	0.24
5	Gradient Boosting Model (Tuned)	0.92	0.94	0.58	0.97	0.35	0.95	0.43
6	XGboost Model	0.91	0.91	1.00	1.00	0.02	0.95	0.05

In [102…

```
# Sort the DataFrame by the 'Accuracy' column
sorted_model = model.sort_values(by='Accuracy', ascending=False)

# Create a bar plot
ax=sns.barplot(sorted_model,x='Model',y='Accuracy')

for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.get_hei
                ha='center', va='center', xytext=(0, 6), textcoords='offset points'

plt.xticks(rotation=90)
plt.title("Model Comparison")
plt.show()
```



- Logistics Regression Model shows high accuracy (0.91) with high precision (1.00) for Class 0. Low recall (0.02) and F1-score (0.05) for Class 1.This Shows that model has difficulty correctly identifying instances for class 1.
- Decision Tree Model shows lower accuracy (0.87) compared to Logistic Regression.Good precision (0.94) and recall (0.91) for Class 0, but lower precision (0.36) and recall (0.47) for Class 1.The F1-Score for Class 1 is moderate.
- The tuned Decision Tree model has Slightly improved accuracy (0.91) after hyperparameter tuning.However, there is still chances for improvement, as the recall for Class 1 remains relatively low.
- The Random Forest model performs well with High accuracy (0.94) and with good precision (0.94) and recall (1.00) for Class 0.
- The Gradient Boosting model shows high recall for Class 0 but struggles with both precision and recall for Class 1, resulting in a lower F1-Score for Class 1.
- The tuned Gradient Boosting model improves Improved accuracy (0.92) after tuning. Better performance for Class 1 compared to the untuned model.

- The XGBoost model shows similar performance to Logistic Regression. Challenges in predicting Class 1, with recall, and F1-score.
- Overall, the Random Forest model seems to perform well, including accuracy, precision, recall, and F1-Score. It looks to be the most balanced model.

SQL (Structured Query Language)

Note: Use only the cleaned data for SQL part of the project

- Group the customers based on their income type and find the average of their annual income.
- Find the female owners of cars and property.
- Find the male customers who are staying with their families.
- Please list the top five people having the highest income.
- How many married people are having bad credit?
- What is the highest education level and what is the total count?
- Between married males and females, who is having more bad credit?

In [86]:
```
# Import library for sql
import duckdb
conn=duckdb.connect()
```

In [87]:
```
# Import cleaned_data.csv for performing sql queries
sql_df=pd.read_csv("cleaned_data.csv")
sql_df
```

Out[87]:

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	El
0	5008827	M	Y	Y	0	180000.0	Pensioner	
1	5009744	F	Y	N	0	315000.0	Commercial associate	
2	5009746	F	Y	N	0	315000.0	Commercial associate	
3	5009749	F	Y	N	0	166500.0	Commercial associate	
4	5009752	F	Y	N	0	315000.0	Commercial associate	
...	...	...	...	...	...	...	...	
1543	5028645	F	N	Y	0	166500.0	Commercial associate	
1544	5023655	F	N	N	0	225000.0	Commercial associate	
1545	5115992	M	Y	Y	2	180000.0	Working	
1546	5118219	M	Y	N	0	270000.0	Working	S
1547	5053790	F	Y	Y	0	225000.0	Working	

1548 rows × 19 columns

In [88]:
```
# Connect with sql_df
conn.register('df',sql_df)
```

Out[88]: <duckdb.duckdb.DuckDBPyConnection at 0x22864086730>

Q1. Group the customers based on their income type and find the average of their annual income.

In [89]:
```
conn.execute("SELECT Type_income, AVG(Annual_income) AS AVG_annual_income FROM df G
```

Out[89]:

	Type_Income	AVG_annual_income
0	Commercial associate	233107.397260
1	State servant	211422.413793
2	Working	180848.210526
3	Pensioner	155343.496283

Q2. Find the female owners of cars and property.

In [90]:
```
conn.execute("SELECT Ind_ID,GENDER, Car_Owner, Propert_Owner FROM df WHERE GENDER='
```

Out[90]:

	Ind_ID	GENDER	Car_Owner	Propert_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

Q3. Find the male customers who are staying with their families.

In [91]:
```
conn.execute("SELECT Ind_ID,GENDER,Marital_Status,Family_Members FROM df WHERE GEND
```

Out[91]:

	Ind_ID	GENDER	Marital_status	Family_Members
0	5008827	M	Married	2
1	5010864	M	Married	3
2	5010868	M	Married	3
3	5021303	M	Married	3
4	5021310	M	Married	2
...	...	...	...	...
465	5096856	M	Married	2
466	5090942	M	Married	2
467	5118268	M	Married	3
468	5115992	M	Married	4
469	5118219	M	Civil marriage	2

470 rows × 4 columns

Q4. Please list the top five people having the highest income.

In [92]:
```
conn.execute("SELECT Ind_ID,GENDER,Annual_income FROM df ORDER BY Annual_income DES
```

Out[92]:

	Ind_ID	GENDER	Annual_income
0	5143231	F	1575000.0
1	5143235	F	1575000.0
2	5090470	M	900000.0
3	5079016	M	900000.0
4	5079017	M	900000.0

Q5. How many married people are having bad credit?

In [93]:
```
conn.execute("SELECT COUNT(*) AS count FROM df WHERE Marital_status='Married' AND l
```

Out[93]:

	count
0	114

Q6. What is the highest education level and what is the total count?

In [94]:
```
conn.execute("SELECT EDUCATION AS highest_education_level,COUNT(*) AS total_count F
```

Out[94]:

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

Q7. Between married males and females, who is having more bad credit?

In [95]:
```
conn.execute("SELECT Marital_status, GENDER,COUNT(*) AS bad_credit_count FROM df WH
```

Out[95]:

	Marital_status	GENDER	bad_credit_count
0	Married	F	567
1	Married	M	368

In [ ]: