Phase 3

Sama Amr -- 900211296 & Farida Madkour -- 900211360

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
###

%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving bankloan.csv to bankloan.csv
```

In [ ]:
```
df = pd.read_csv("bankloan.csv")
df.head()
```

Out[ ]:

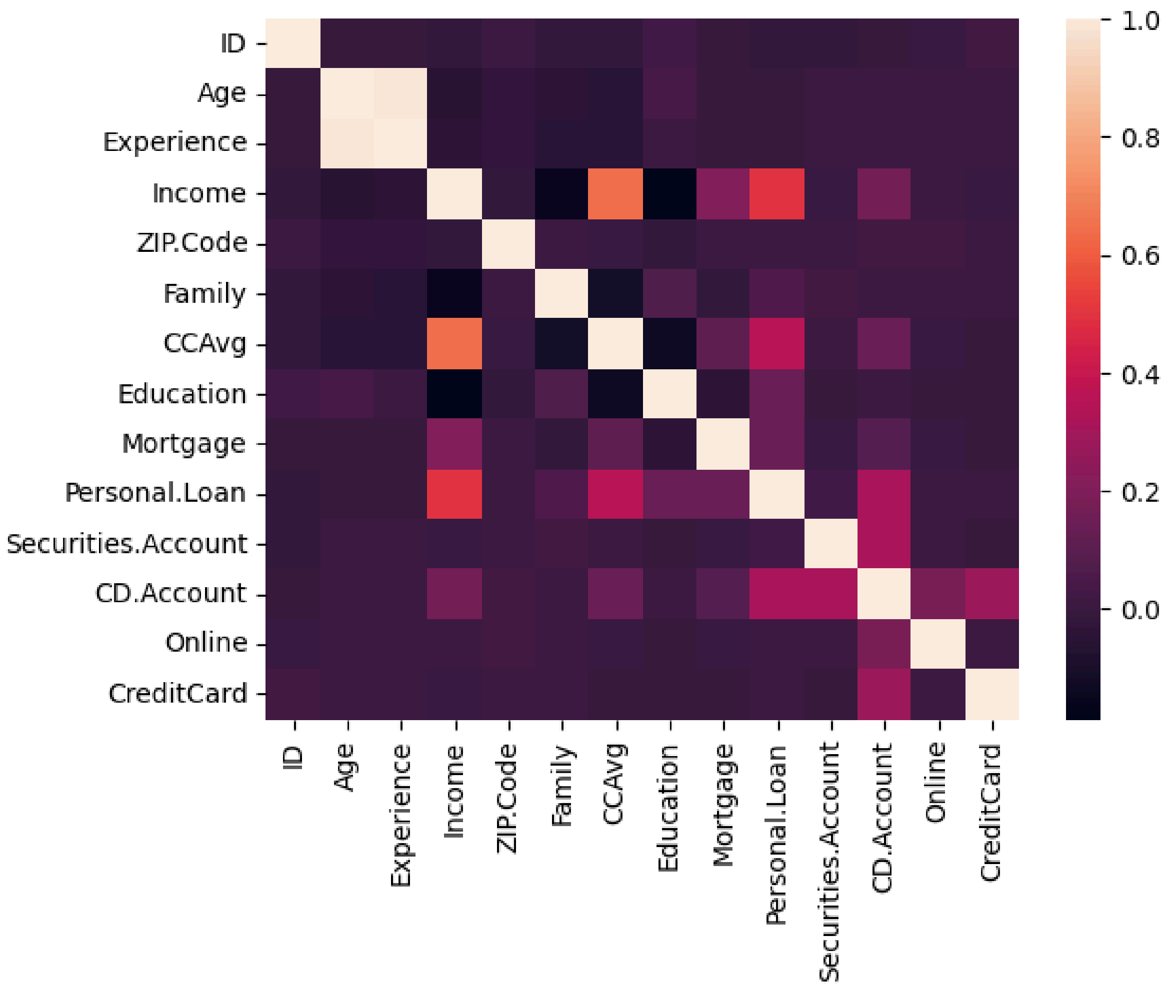
	ID	Age	Experience	Income	ZIP.Code	Family	CCAvg	Education	Mortgage	Personal.Loan
0	1	25	1	49	91107	4	1.6	1	0	0
1	2	45	19	34	90089	3	1.5	1	0	0
2	3	39	15	11	94720	1	1.0	1	0	0
3	4	35	9	100	94112	1	2.7	2	0	0
4	5	35	8	45	91330	4	1.0	2	0	0

In [ ]:
```
df.info()
print("--------------------------------")
print("List of Columns:", df.columns)
print("Shape:", df.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIP.Code            5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64  
 8   Mortgage            5000 non-null   int64  
 9   Personal.Loan       5000 non-null   int64  
 10  Securities.Account  5000 non-null   int64  
 11  CD.Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
--------------------------------
List of Columns: Index(['ID', 'Age', 'Experience', 'Income', 'ZIP.Code', 'Family',
'CCAvg',
       'Education', 'Mortgage', 'Personal.Loan', 'Securities.Account',
       'CD.Account', 'Online', 'CreditCard'],
      dtype='object')
Shape: (5000, 14)
```

In [ ]:
```
sns.heatmap(df.corr(), annot=False)
```

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



Drop ID, experience, and Zip Code columns since they're irrelevant

In [ ]:
```
df = df.drop(columns=['ID','Experience','ZIP.Code'])
df.head()
```

Out[ ]:

	Age	Income	Family	CCAvg	Education	Mortgage	Personal.Loan	Securities.Account	CD.Acco
0	25	49	4	1.6	1	0	0	1	
1	45	34	3	1.5	1	0	0	1	
2	39	11	1	1.0	1	0	0	0	
3	35	100	1	2.7	2	0	0	0	
4	35	45	4	1.0	2	0	0	0	

Check for missing values

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

Out[ ]:
```
Age                   0
Income                0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal.Loan         0
Securities.Account    0
CD.Account            0
Online                0
CreditCard            0
dtype: int64
```

Therefore, there is no missing values as specified by the non-null count and the sum calculated

Check for duplicate values and drop them

In [ ]:
```
df.duplicated().sum()
```

Out[ ]:
```
13
```

In [ ]:
```
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

Out[ ]:
```
0
```

Encodings

Change numeric/continous variables to type float and categorical/discrete variable to type category

In [ ]:
```
df['Income']=df['Income'].astype('float')
df['Family']=df['Family'].astype('category')
df['Education']=df['Education'].astype('category')
df['CCAvg']=df['CCAvg'].astype('float')
df['Mortgage']=df['Mortgage'].astype('float')
```

```
df['Personal.Loan']=df['Personal.Loan'].astype('category')
df['Securities.Account']=df['Securities.Account'].astype('category')
df['CD.Account']=df['CD.Account'].astype('category')
df['Online']=df['Online'].astype('category')
df['CreditCard']=df['CreditCard'].astype('category')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4987 entries, 0 to 4999
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 4987 non-null   int64
 1   Income              4987 non-null   float64
 2   Family              4987 non-null   category
 3   CCAvg               4987 non-null   float64
 4   Education           4987 non-null   category
 5   Mortgage            4987 non-null   float64
 6   Personal.Loan       4987 non-null   category
 7   Securities.Account  4987 non-null   category
 8   CD.Account          4987 non-null   category
 9   Online              4987 non-null   category
 10  CreditCard          4987 non-null   category
dtypes: category(7), float64(3), int64(1)
memory usage: 229.8 KB
```

Cut the Age and income into Ranges for better interpretations

In [ ]:

```
#minimum age = 23
#maximum age = 67
bins = [22,30,40,50,60,70]
df['Age_r'] = pd.cut(df['Age'], bins=bins, labels=['23-30', '30-40', '40-50', '50-6

#minimum age = 8
#maximum age = 224
bins = [7,20,100,150,200,250]
df['Income_r'] = pd.cut(df['Income'], bins=bins, labels=['Poor', 'Middle_Class', 'U
df.head()
```

Out[ ]:

	Age	Income	Family	CCAvg	Education	Mortgage	Personal.Loan	Securities.Account	CD.Acco
0	25	49.0	4	1.6	1	0.0	0	1	
1	45	34.0	3	1.5	1	0.0	0	1	
2	39	11.0	1	1.0	1	0.0	0	0	
3	35	100.0	1	2.7	2	0.0	0	0	
4	35	45.0	4	1.0	2	0.0	0	0	

Unique values of each of the variables

In [ ]:

```
print("Unique Family",pd.unique(df['Family']))
print("------------------------------------------")
print("Unique Education",pd.unique(df['Education']))
print("------------------------------------------")
print("Unique Personal.Loan",pd.unique(df['Personal.Loan']))
print("------------------------------------------")
print("Unique Securities.Account",pd.unique(df['Securities.Account']))
print("------------------------------------------")
print("Unique CD.Account",pd.unique(df['CD.Account']))
```

```
print("------------------------------------------")
print("Unique Online",pd.unique(df['Online']))
print("------------------------------------------")
print("Unique CreditCard",pd.unique(df['CreditCard']))
```

```
Unique Family [4, 3, 1, 2]
Categories (4, int64): [1, 2, 3, 4]
------------------------------------------
Unique Education [1, 2, 3]
Categories (3, int64): [1, 2, 3]
------------------------------------------
Unique Personal.Loan [0, 1]
Categories (2, int64): [0, 1]
------------------------------------------
Unique Securities.Account [1, 0]
Categories (2, int64): [0, 1]
------------------------------------------
Unique CD.Account [0, 1]
Categories (2, int64): [0, 1]
------------------------------------------
Unique Online [0, 1]
Categories (2, int64): [0, 1]
------------------------------------------
Unique CreditCard [0, 1]
Categories (2, int64): [0, 1]
```

Correlation Matrix along with a heatmap for our Numerical/Continous Variables

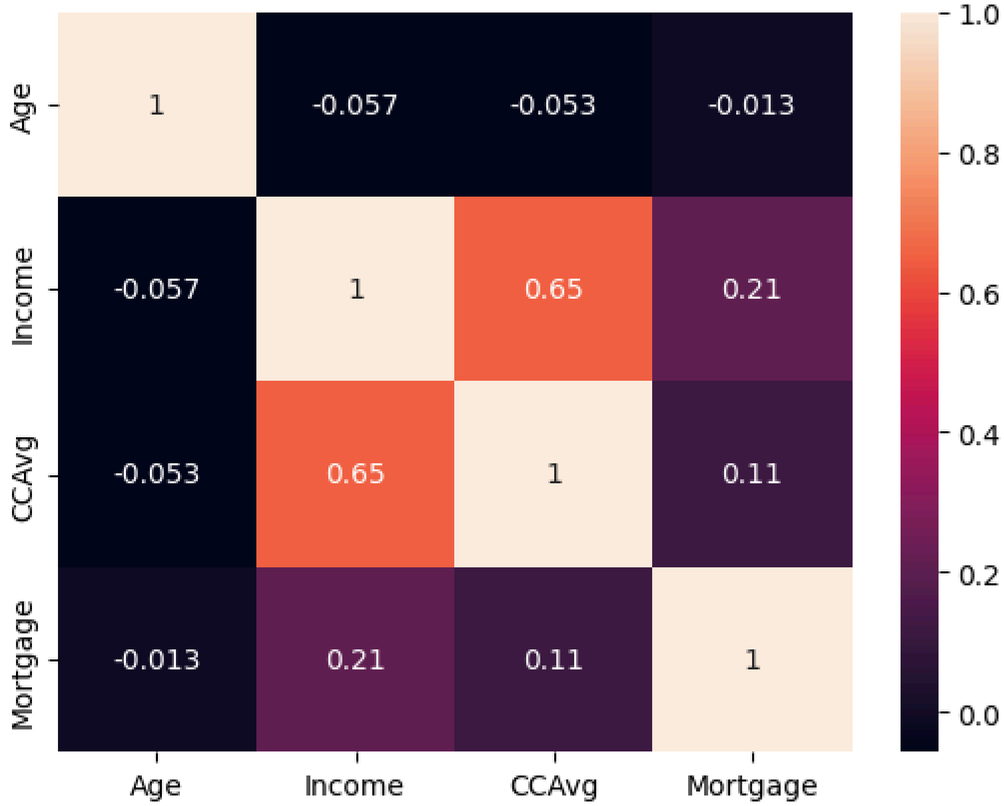
In [ ]:
```
df.corr()
```

Out[ ]:

	Age	Income	CCAvg	Mortgage
Age	1.000000	-0.056897	-0.052522	-0.013014
Income	-0.056897	1.000000	0.646065	0.206420
CCAvg	-0.052522	0.646065	1.000000	0.109162
Mortgage	-0.013014	0.206420	0.109162	1.000000

In [ ]:
```
sns.heatmap(df.corr(), annot=True)
```

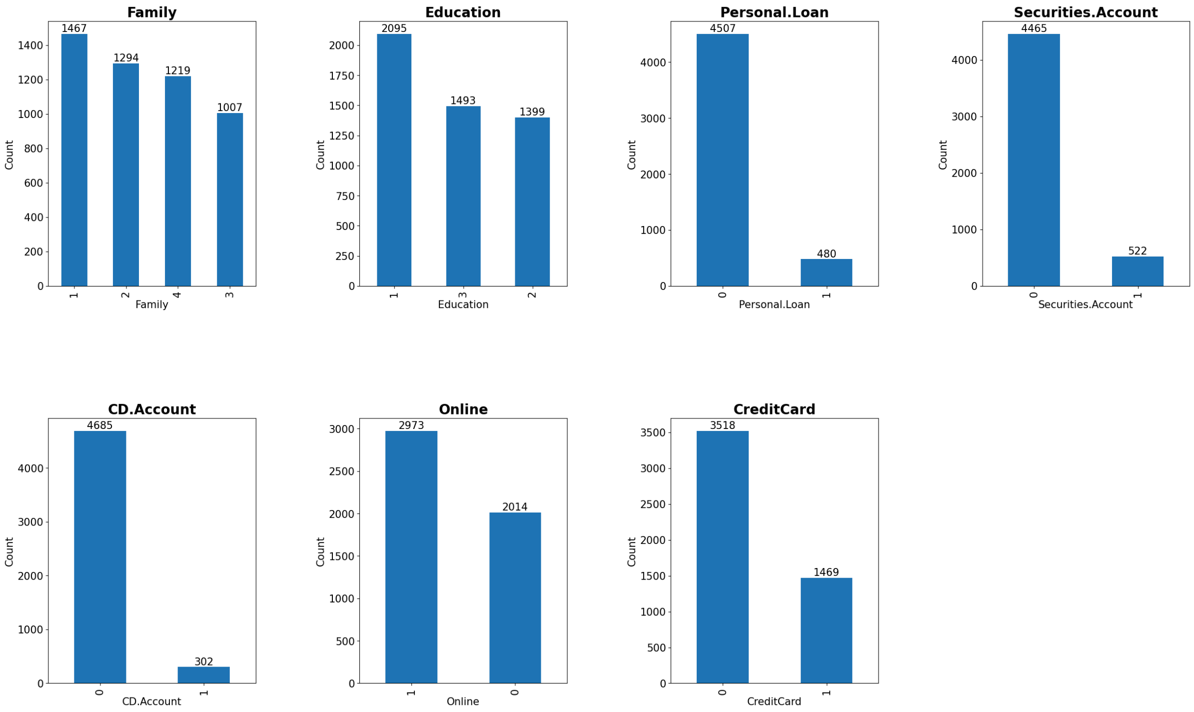
Out[ ]:
```
<Axes: >
```



Barplots for our Discrete variables to show their distributions

In [ ]:
```
plt.figure(figsize=(30,28))
plt.subplot(3,4,1)
df['Family'].value_counts().plot(kind='bar')
plt.title("Family", fontsize=20, fontweight="bold")
plt.xlabel('Family',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
counts=df['Family'].value_counts()
for i, count in enumerate (counts):
    plt.text(i,count+1,str(count), ha='center',va='bottom',fontsize=15)

plt.subplot(3,4,2)
df['Education'].value_counts().plot(kind='bar')
plt.title("Education", fontsize=20, fontweight="bold")
plt.xlabel('Education',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
counts=df['Education'].value_counts()
for i, count in enumerate (counts):
    plt.text(i,count+1,str(count), ha='center',va='bottom',fontsize=15)

plt.subplot(3,4,3)
df['Personal.Loan'].value_counts().plot(kind='bar')
plt.title("Personal.Loan", fontsize=20, fontweight="bold")
plt.xlabel('Personal.Loan',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
counts=df['Personal.Loan'].value_counts()
for i, count in enumerate (counts):
    plt.text(i,count+1,str(count), ha='center',va='bottom',fontsize=15)

plt.subplot(3,4,4)
```

```python
df['Securities.Account'].value_counts().plot(kind='bar')
plt.title("Securities.Account", fontsize=20, fontweight="bold")
plt.xlabel('Securities.Account',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
counts=df['Securities.Account'].value_counts()
for i, count in enumerate (counts):
    plt.text(i,count+1,str(count), ha='center',va='bottom',fontsize=15)

plt.subplot(3,4,5)
df['CD.Account'].value_counts().plot(kind='bar')
plt.title("CD.Account", fontsize=20, fontweight="bold")
plt.xlabel('CD.Account',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
counts=df['CD.Account'].value_counts()
for i, count in enumerate (counts):
    plt.text(i,count+1,str(count), ha='center',va='bottom',fontsize=15)

plt.subplot(3,4,6)
df['Online'].value_counts().plot(kind='bar')
plt.title("Online", fontsize=20, fontweight="bold")
plt.xlabel('Online',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
counts=df['Online'].value_counts()
for i, count in enumerate (counts):
    plt.text(i,count+1,str(count), ha='center',va='bottom',fontsize=15)

plt.subplot(3,4,7)
df['CreditCard'].value_counts().plot(kind='bar')
plt.title("CreditCard", fontsize=20, fontweight="bold")
plt.xlabel('CreditCard',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
counts=df['CreditCard'].value_counts()
for i, count in enumerate (counts):
    plt.text(i,count+1,str(count), ha='center',va='bottom',fontsize=15)

plt.subplots_adjust(wspace=0.5,hspace=0.5)
plt.show()
```



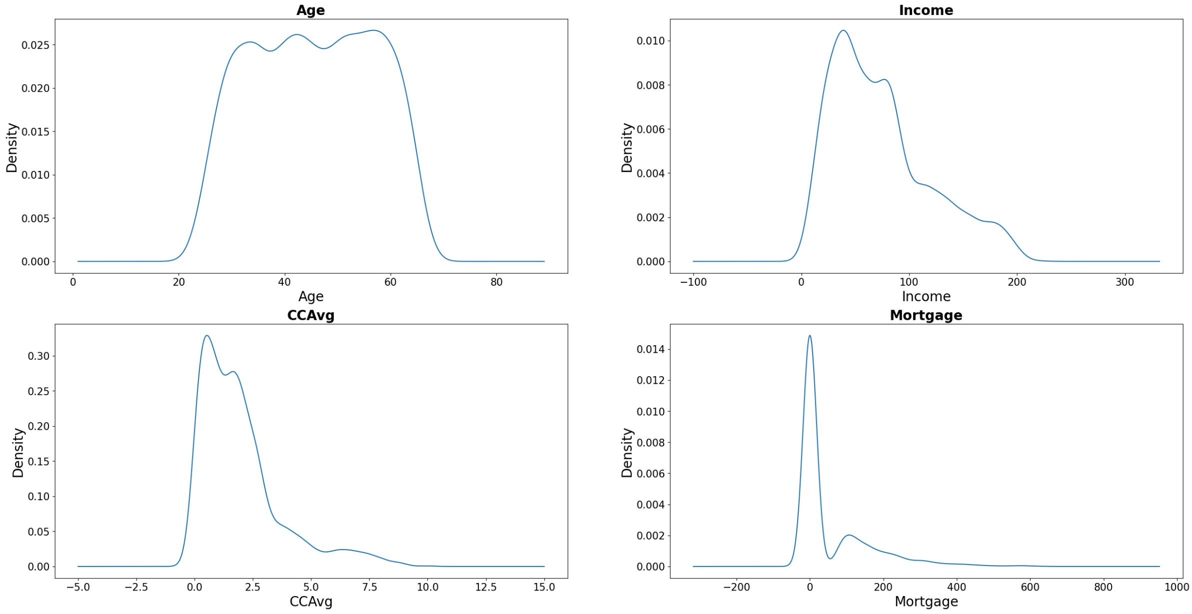
Linegraphs for our Continous variables to show their distributions

In [ ]:

```
plt.figure(figsize=(30,15))
plt.subplot(2,2,1)
df['Age'].plot(kind='density')
plt.title("Age", fontsize=20, fontweight="bold")
plt.xlabel('Age',fontsize=20)
plt.ylabel('Density',fontsize=20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(2,2,2)
df['Income'].plot(kind='density')
plt.xlabel('Income',fontsize=20)
plt.ylabel('Density',fontsize=20)
plt.title("Income", fontsize=20, fontweight="bold")
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(2,2,3)
df['CCAvg'].plot(kind='density')
plt.title("CCAvg", fontsize=20, fontweight="bold")
plt.xlabel('CCAvg',fontsize=20)
plt.ylabel('Density',fontsize=20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(2,2,4)
df['Mortgage'].plot(kind='density')
plt.title("Mortgage", fontsize=20, fontweight="bold")
plt.xlabel('Mortgage',fontsize=20)
plt.ylabel('Density',fontsize=20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
```

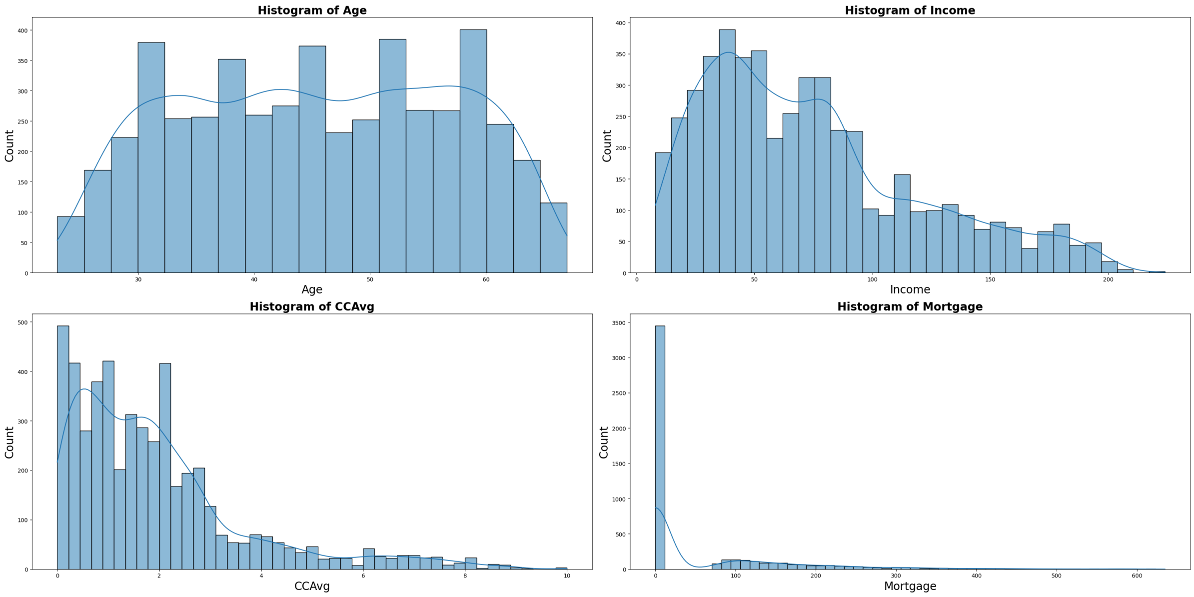
Out[ ]:

```
(array([-0.002,  0.   ,  0.002,  0.004,  0.006,  0.008,  0.01 ,  0.012,
         0.014,  0.016]),
 [Text(0, -0.002, '-0.002'),
  Text(0, 0.0, '0.000'),
  Text(0, 0.002, '0.002'),
  Text(0, 0.004, '0.004'),
  Text(0, 0.006, '0.006'),
  Text(0, 0.008, '0.008'),
  Text(0, 0.01, '0.010'),
  Text(0, 0.012, '0.012'),
  Text(0, 0.014, '0.014'),
  Text(0, 0.016, '0.016')])
```



In [ ]:

```python
dist_columns = ['Age', 'Income', 'CCAvg','Mortgage']

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(30, 15))

axes = axes.flatten()

for i, col in enumerate(dist_columns):
    sns.histplot(df[col], kde=True, ax=axes[i])
    axes[i].set_title(f'Histogram of {col}',fontsize=20,fontweight='bold')
    axes[i].set_xlabel(col,fontsize=20)
    axes[i].set_ylabel('Count',fontsize=20)


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

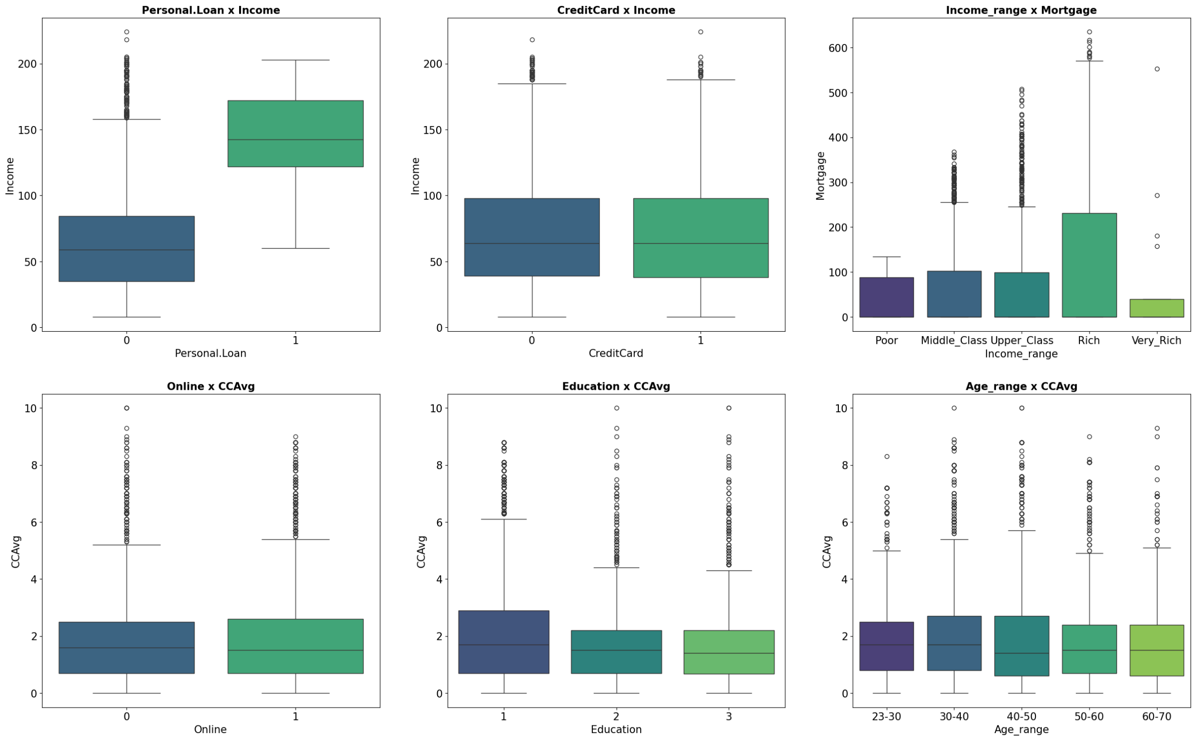


Box plots to show the relations and errors between each pair of variables

In [ ]:

```
plt.figure(figsize=(30,28))
plt.subplot(3,3,1)
sns.boxplot(x='Personal.Loan', y='Income', data=df, palette='viridis')
plt.xlabel('Personal.Loan',fontsize=15)
plt.ylabel('Income',fontsize=15)
plt.title('Personal.Loan x Income',fontsize=15, fontweight='bold')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(3,3,2)
sns.boxplot(x='CreditCard', y='Income', data=df, palette='viridis')
plt.xlabel('CreditCard',fontsize=15)
plt.ylabel('Income',fontsize=15)
plt.title('CreditCard x Income',fontsize=15, fontweight='bold')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(3,3,3)
sns.boxplot(x='Income_r', y='Mortgage', data=df, palette='viridis')
plt.xlabel('Income_range',fontsize=15)
plt.ylabel('Mortgage',fontsize=15)
plt.title('Income_range x Mortgage',fontsize=15, fontweight='bold')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(3,3,4)
sns.boxplot(x='Online', y='CCAvg', data=df, palette='viridis')
plt.xlabel('Online',fontsize=15)
plt.ylabel('CCAvg',fontsize=15)
plt.title('Online x CCAvg',fontsize=15, fontweight='bold')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(3,3,5)
sns.boxplot(x='Education', y='CCAvg', data=df, palette='viridis')
plt.xlabel('Education',fontsize=15)
plt.ylabel('CCAvg',fontsize=15)
plt.title('Education x CCAvg',fontsize=15, fontweight='bold')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.subplot(3,3,6)
```

```python
sns.boxplot(x='Age_r', y='CCAvg', data=df, palette='viridis')
plt.xlabel('Age_range',fontsize=15)
plt.ylabel('CCAvg',fontsize=15)
plt.title('Age_range x CCAvg',fontsize=15, fontweight='bold')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.show()
```



We can identify that income, CCAvg and Mortgage are 3 features containing outliers

Function to identify outliers

In [ ]:
```python
def outlier(df):

  Q1=df.quantile(0.25)

  Q3=df.quantile(0.75)

  IQR=Q3-Q1

  out = df[((df<(Q1-1.5*IQR)) | (df>(Q3+1.5*IQR)))]

  return out
```

In [ ]:
```python
skewed =['Income','CCAvg','Mortgage']
for col in skewed:
    outliers=outlier(df[col])
    print("Number of outliers in",col,":", str(len(outliers)),",It's Percentage is
    print("\n")
```

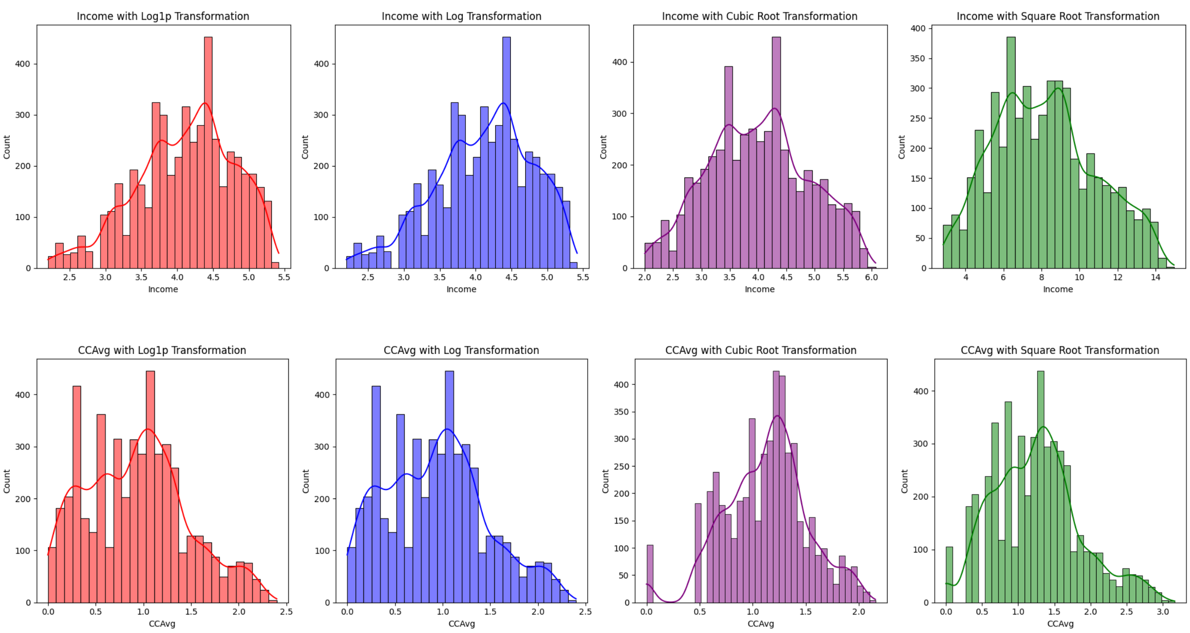
```
Number of outliers in Income : 96 ,It's Percentage is :  1.925005013033888 %


Number of outliers in CCAvg : 301 ,It's Percentage is :  6.035692801283337 %


Number of outliers in Mortgage : 291 ,It's Percentage is :  5.835171445758974 %
```

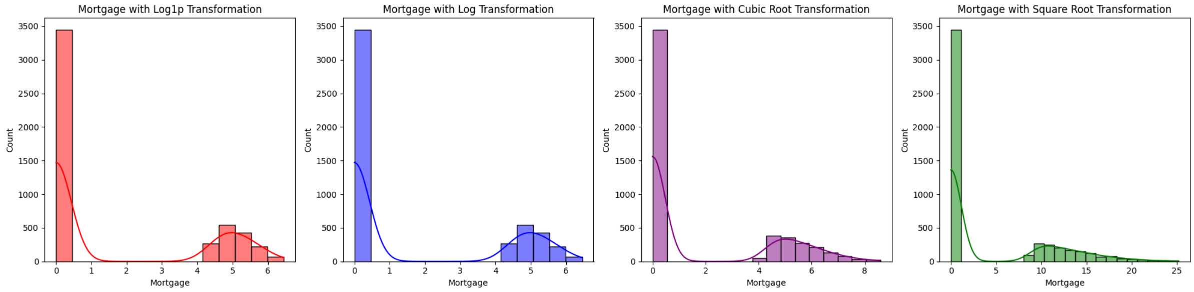
Outlier numbers are relatively low, yet they could better. In addition their line graphs and histograms are skewed. We found a solution to the problems by:

Find a suitable transformation for the skewed features

In [ ]:

```python
skewed =['Income','CCAvg','Mortgage']

for col in skewed:
    fig, axes = plt.subplots(1, 4, figsize=(20, 5))

    # Log1p Transformation
    sns.histplot(np.log1p(df[col]), color='red', ax=axes[0],kde=True)
    axes[0].set_title(f'{col} with Log1p Transformation')

    # Log Transformation
    sns.histplot(np.log(df[col] + 1), color='blue', ax=axes[1],kde=True)
    axes[1].set_title(f'{col} with Log Transformation')

    # Cubic Root Transformation
    sns.histplot(np.cbrt(df[col]), color='purple', ax=axes[2],kde=True)
    axes[2].set_title(f'{col} with Cubic Root Transformation')

    # Square Root Transformation
    sns.histplot(np.sqrt(df[col]), color='green', ax=axes[3],kde=True)
    axes[3].set_title(f'{col} with Square Root Transformation')

    plt.tight_layout()
    plt.show()
    print("\n")
```





Therefore, the best suitable transformation for:

- Income: Cubic root
- CCAvg: Cubic root
- Mortgage: Square root

In [ ]:
```
df['Income'] = np.cbrt(df['Income'])
df['CCAvg'] = np.cbrt(df['CCAvg'])
df['Mortgage'] = np.sqrt(df['Mortgage'])
```

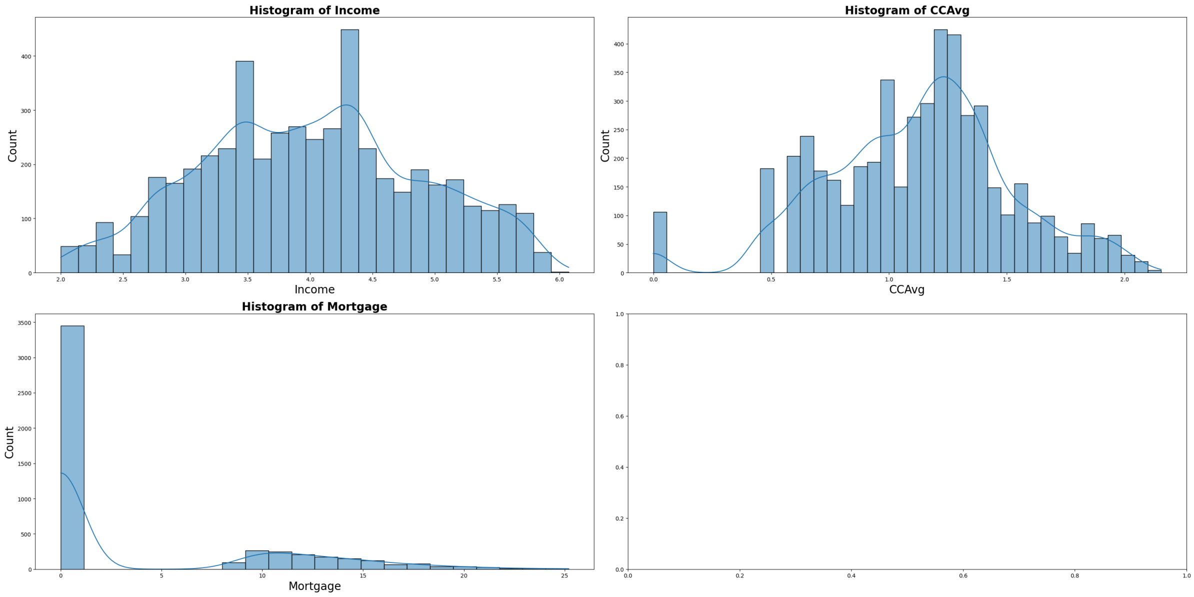
Test for outliers after the transformation and plot the histograms

In [ ]:
```
skewed =['Income','CCAvg','Mortgage']
for col in skewed:
    outliers=outlier(df[col])
    print("Number of outliers in",col,":", str(len(outliers)),",It's Percentage is
    print("\n")
```

```
Number of outliers in Income : 0 ,It's Percentage is :  0.0 %


Number of outliers in CCAvg : 109 ,It's Percentage is :  2.1856827752155605 %


Number of outliers in Mortgage : 1 ,It's Percentage is :  0.020052135552436335 %
```

Outliers are significantly reduced after the transformation

In [ ]:
```
dist_columns = ['Income', 'CCAvg','Mortgage']

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(30, 15))

axes = axes.flatten()

for i, col in enumerate(dist_columns):
    sns.histplot(df[col], kde=True, ax=axes[i])
    axes[i].set_title(f'Histogram of {col}',fontsize=20,fontweight='bold')
    axes[i].set_xlabel(col,fontsize=20)
    axes[i].set_ylabel('Count',fontsize=20)


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



Finally, here's a summary of our continous features

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

Out[ ]:

	Age	Income	CCAvg	Mortgage
count	4987.000000	4987.000000	4987.000000	4987.000000
mean	45.347704	4.006779	1.127270	4.046775
std	11.460838	0.887801	0.392599	6.346477
min	23.000000	2.000000	0.000000	0.000000
25%	35.000000	3.391211	0.887904	0.000000
50%	45.000000	4.000000	1.144714	0.000000
75%	55.000000	4.610436	1.375069	10.049876
max	67.000000	6.073178	2.154435	25.199206

Pilot Study (Phase 3)

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

df1['Income_r'] = pd.factorize(df1['Income_r'])[0] + 1
df1['Age_r'] = pd.factorize(df1['Age_r'])[0] + 1

df1['Income_r'] = df1['Income_r'].astype(int)
df1['Age_r'] = df1['Age_r'].astype(int)


print(df1.dtypes)

df1
```

```
Age                      int64
Income                 float64
Family                category
CCAvg                  float64
Education             category
Mortgage               float64
Personal.Loan         category
Securities.Account    category
CD.Account            category
Online                category
CreditCard            category
Age_r                    int64
Income_r                 int64
dtype: object
```

Out[ ]:

	Age	Income	Family	CCAvg	Education	Mortgage	Personal.Loan	Securities.Account	C
0	25	3.659306	4	1.169607	1	0.000000	0	1	
1	45	3.239612	3	1.144714	1	0.000000	0	1	
2	39	2.223980	1	1.000000	1	0.000000	0	0	
3	35	4.641589	1	1.392477	2	0.000000	0	0	
4	35	3.556893	4	1.000000	2	0.000000	0	0	
...	...	...	...	...	...	...	...	...	
4995	29	3.419952	1	1.238562	3	0.000000	0	0	
4996	30	2.466212	4	0.736806	1	9.219544	0	0	
4997	63	2.884499	2	0.669433	3	0.000000	0	0	
4998	65	3.659306	3	0.793701	2	0.000000	0	0	
4999	28	4.362071	3	0.928318	1	0.000000	0	0	

4987 rows × 13 columns

Standardize our variable

In [ ]:

```
from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler()

df_scaled=df1.copy()
columns = ['Age',       'Income',       'Family',       'CCAvg',        'Education'
for col in columns:
    df_scaled[col] = standard_scaler.fit_transform(np.array(df_scaled[col]).reshape

df_scaled.head()
```

Out[ ]:

	Age	Income	Family	CCAvg	Education	Mortgage	Personal.Loan	Securities.Acco
0	-1.775590	-0.391426	1.397399	0.107848	-1.047290	-0.637705	0	2.924
1	-0.030341	-0.864208	0.525860	0.044436	-1.047290	-0.637705	0	2.924
2	-0.553916	-2.008309	-1.217219	-0.324207	-1.047290	-0.637705	0	-0.341
3	-0.902966	0.715108	-1.217219	0.675583	0.143778	-0.637705	0	-0.341
4	-0.902966	-0.506793	1.397399	-0.324207	0.143778	-0.637705	0	-0.341

Checking if there are any missing values

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

Out[ ]:
```
Age                   0
Income                0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal.Loan         0
Securities.Account    0
CD.Account            0
Online                0
CreditCard            0
Age_r                 0
Income_r              0
dtype: int64
```

Assigning our target and decision variables

In [ ]:
```
Y = df_scaled['Personal.Loan']
X = df_scaled.drop(['Personal.Loan'],axis=1)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y,test_size=0.4, random_stat
```

In [ ]:
```
print (" Number of columns in our Features : ", X.shape[1])
```

```
 Number of columns in our Features :  12
```

Solving the Class imbalance problem

In [ ]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_upsampled, y_train_upsampled = smote.fit_resample(X_train, y_train)
```

In [ ]:
```
print("Before UpSampling, counts of Personal loan = '0': {}".format(sum(y_train==0)
print("Before UpSampling, counts of Personal loan = '1': {} \n".format(sum(y_train=


print("After UpSampling, counts of Personal loan = '0': {}".format(sum(y_train_upsa
print("After UpSampling, counts of Personal loan = '1': {} \n".format(sum(y_train_u
```

```
Before UpSampling, counts of Personal loan = '0': 2699
Before UpSampling, counts of Personal loan = '1': 293

After UpSampling, counts of Personal loan = '0': 2699
After UpSampling, counts of Personal loan = '1': 2699
```

Initialize a Data Frame to store the Accuracy, Precision, Recall,and F1 score for all our upcoming model

In [ ]:
```
EVAL_SCORE = pd.DataFrame(columns=['Model','Accuracy','Precision','Recall' ,'F1 Sco

EVAL_SCORE
```

Out[ ]:

Model	Accuracy	Precision	Recall	F1 Score

Decision Tree Model

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
import plotly.graph_objects as go
```

In [ ]:
```
max_depth_values = range(1, 50)

train_scores = []
test_scores = []

for depth in max_depth_values:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X_train_upsampled, y_train_upsampled)

    y_train_pred = clf.predict(X_train_upsampled)
    train_scores.append(accuracy_score(y_train_upsampled, y_train_pred))

    y_test_pred = clf.predict(X_test)
    test_scores.append(accuracy_score(y_test, y_test_pred))

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(max_depth_values), y=train_scores, mode='lines', na

fig.add_trace(go.Scatter(x=list(max_depth_values), y=test_scores, mode='lines', nam

fig.update_layout(
    title='Max Depth vs. Accuracy',
    xaxis=dict(title='Max Depth'),
    yaxis=dict(title='Accuracy'),
    legend=dict(x=0.7, y=0.9),
)

fig.show()
```

From this graph we could deduce that after a depth=3 the graph starts to flatten out; therfore, we'd build our decision tree model using a maximum depth=3. That would allow us to overcome overfitting problems

In [ ]:
```
min_samples_split_values = range(2, 30)

train_scores = []
test_scores = []

for split in min_samples_split_values:
    clf = DecisionTreeClassifier(min_samples_split=split, random_state=42)
    clf.fit(X_train_upsampled, y_train_upsampled)

    y_train_pred = clf.predict(X_train_upsampled)
    train_scores.append(accuracy_score(y_train_upsampled, y_train_pred))

    y_test_pred = clf.predict(X_test)
    test_scores.append(accuracy_score(y_test, y_test_pred))

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(min_samples_split_values), y=train_scores, mode='li

fig.add_trace(go.Scatter(x=list(min_samples_split_values), y=test_scores, mode='lin

fig.update_layout(
    title='Min Samples Split vs. Accuracy',
    xaxis=dict(title='Min Samples Split'),
    yaxis=dict(title='Accuracy'),
```

```
    legend=dict(x=0.7, y=0.9),
)

fig.show()
```

From the graph above we could conclude that the optimal minimum sample split is =3 where it provides the maximum accuracy

In [ ]:
```
Decision_Tree = DecisionTreeClassifier(max_depth=3,criterion='entropy',random_state
```

In [ ]:
```
Decision_Tree.fit(X_train_upsampled, y_train_upsampled)
```

Out[ ]:
```
DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
```

In [ ]:
```
y_pred_train = Decision_Tree.predict(X_train_upsampled)
y_pred_test = Decision_Tree.predict(X_test)
```

In [ ]:
```
train_accuracy = accuracy_score(y_train_upsampled, y_pred_train)
print(" Decision Tree Training Accuracy :" ,round(train_accuracy,2)*100)

test_accuracy = accuracy_score(y_test, y_pred_test)
print(" Decision Tree Testing Accuracy :" ,round(test_accuracy,2)*100)
```

```
 Decision Tree Training Accuracy : 98.0
 Decision Tree Testing Accuracy : 96.0
```

Using cross validation on our descision tree and testing the accuracy

In [ ]:
```
cv_scores_train = cross_val_score(Decision_Tree, X_train_upsampled, y_train_upsampl
print("Cross-Validation Scores on Training Data:  ", cv_scores_train)
print(" Mean Accuracy from Cross-Validation : ", cv_scores_train.mean())
```

```
Cross-Validation Scores on Training Data:   [0.96666667 0.96944444 0.97685185 0.97
034291 0.97126969]
 Mean Accuracy from Cross-Validation :  0.970915113445234
```

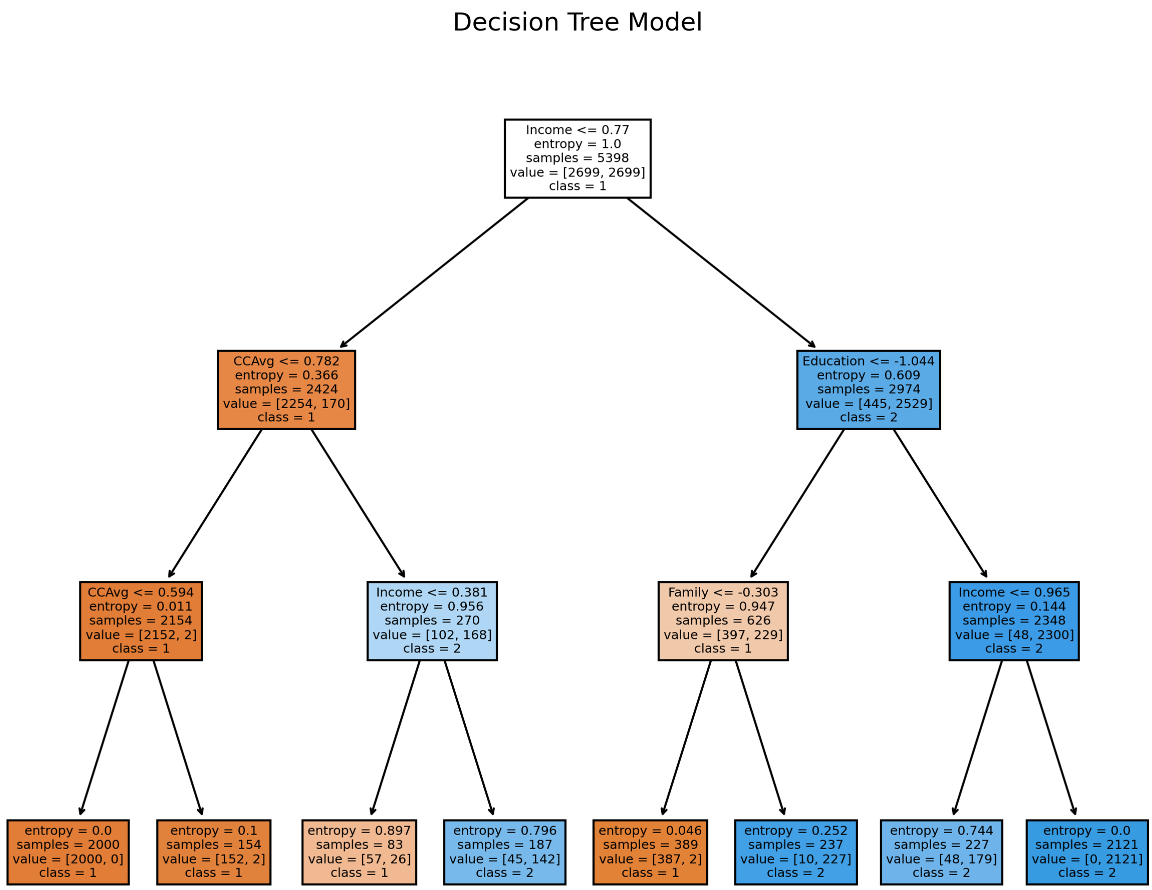
In [ ]:
```
conf_matrix = confusion_matrix(y_test, y_pred_test)

# Add labels for better understanding
tn, fp, fn, tp = conf_matrix.ravel()
display( pd.DataFrame(conf_matrix, columns=['Predicted Negative', 'Predicted Positi
```

	Predicted Negative	Predicted Positive
Actual Negative	1741	67
Actual Positive	3	184

In [ ]:
```
print("Classification Report : \n" ,classification_report(y_test, y_pred_test))
```

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

           0       1.00      0.96      0.98      1808
           1       0.73      0.98      0.84       187

    accuracy                           0.96      1995
   macro avg       0.87      0.97      0.91      1995
weighted avg       0.97      0.96      0.97      1995
```

In [ ]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

cn=["1","2"]
fn=['Age',      'Income',       'Family',       'CCAvg',        'Education',     'Mc

fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (10,8), dpi=300)
plot_tree(Decision_Tree, filled=True,feature_names = fn,class_names=cn)
plt.title("Decision Tree Model")
plt.show()
```



In [ ]:
```
accuracy = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test, y_pred_test, average='macro')
recall = recall_score(y_test, y_pred_test, average='macro')
f1_score= metrics.f1_score(y_test, y_pred_test, average='macro')

EVAL_SCORE = EVAL_SCORE.append({'Model': 'Decision Tree','Accuracy':accuracy,'Preci
EVAL_SCORE
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.964912	0.865674	0.97345	0.910238

Random Forest Model

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

clf = RandomForestClassifier(max_depth=2, random_state=0)

clf.fit(X_train, y_train)
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=2, random_state=0)
```

In [ ]:
```
y_pred_train_rf = clf.predict(X_train)
y_pred_test_rf = clf.predict(X_test)
```

In [ ]:
```
train_accuracy = accuracy_score(y_train, y_pred_train_rf)
print("Random Forest Training Accuracy :" ,round(train_accuracy,3)*100)
```

```
test_accuracy = accuracy_score(y_test, y_pred_test_rf)
print("Random Forest Testing Accuracy :" ,round(test_accuracy,3)*100)
```

```
Random Forest Training Accuracy : 91.7
Random Forest Testing Accuracy : 91.9
```

In [ ]:
```
conf_matrix = confusion_matrix(y_test, y_pred_test_rf)

# Add labels for better understanding
tn, fp, fn, tp = conf_matrix.ravel()
display( pd.DataFrame(conf_matrix, columns=['Predicted Negative', 'Predicted Positi
```

	Predicted Negative	Predicted Positive
Actual Negative	1808	0
Actual Positive	162	25

In [ ]:
```
print("Classification Report : \n" ,classification_report(y_test, y_pred_test_rf))
```

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

           0       0.92      1.00      0.96      1808
           1       1.00      0.13      0.24       187

    accuracy                           0.92      1995
   macro avg       0.96      0.57      0.60      1995
weighted avg       0.93      0.92      0.89      1995
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred_test_rf)
precision = precision_score(y_test, y_pred_test_rf, average='macro')
recall = recall_score(y_test, y_pred_test_rf, average='macro')
f1_score= metrics.f1_score(y_test, y_pred_test_rf, average='macro')

EVAL_SCORE = EVAL_SCORE.append({'Model': 'Random Forest','Accuracy':accuracy,'Preci
EVAL_SCORE
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.964912	0.865674	0.973450	0.910238
1	Random Forest	0.918797	0.958883	0.566845	0.596485

Logistic Regression Model

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logistic_model=LogisticRegression()
logistic_model.fit(X_train,y_train)
```

Out[ ]:
```
▾ LogisticRegression

LogisticRegression()
```

In [ ]:
```
y_pred_log=logistic_model.predict(X_test)

y_pred_train_log=logistic_model.predict(X_train)
```

In [ ]:
```
conf_matrix = confusion_matrix(y_test, y_pred_log)
```

```
# Add labels for better understanding
tn, fp, fn, tp = conf_matrix.ravel()
display( pd.DataFrame(conf_matrix, columns=['Predicted Negative', 'Predicted Positi
```

	Predicted Negative	Predicted Positive
Actual Negative	1783	25
Actual Positive	56	131

In [ ]:
```
print("Classification Report : \n" ,classification_report(y_test, y_pred_log))
```

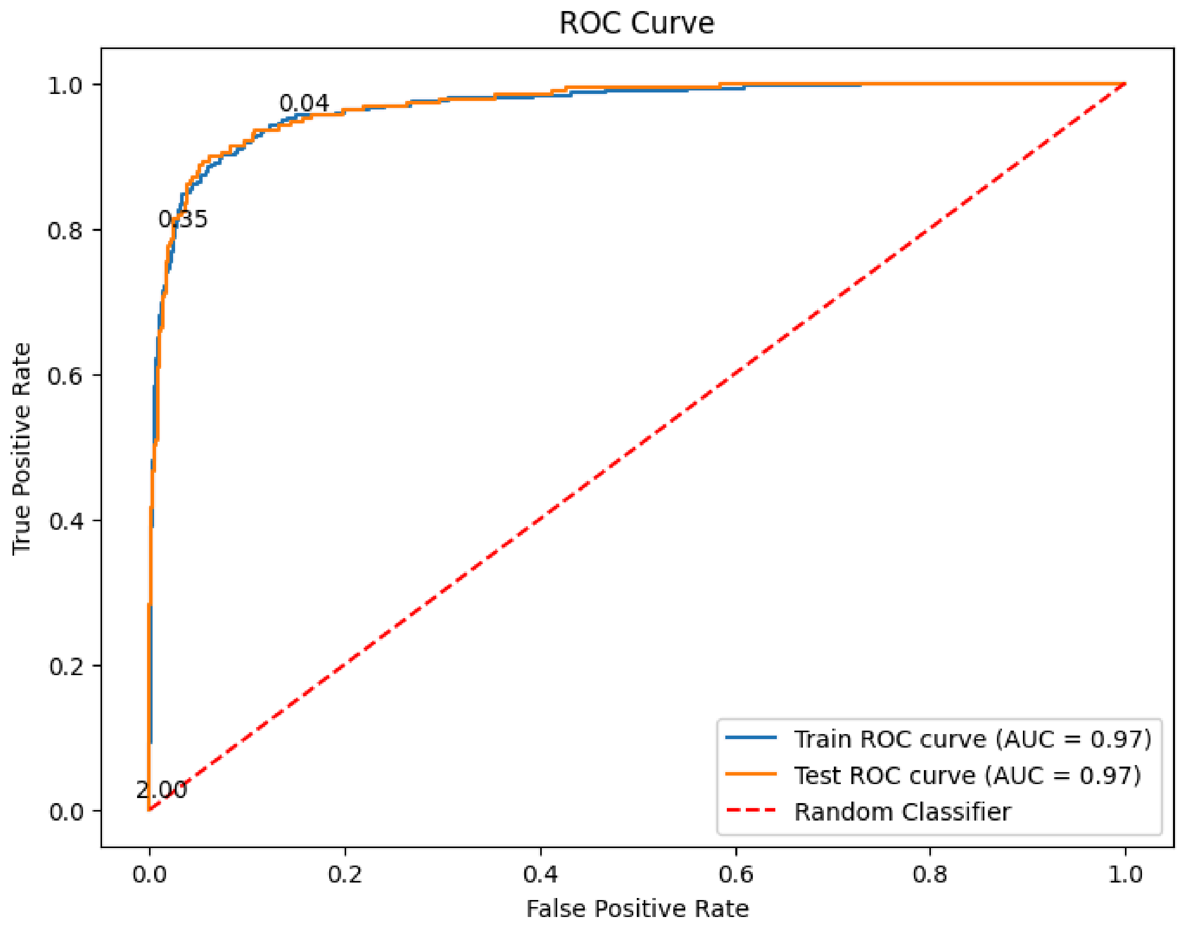
```
Classification Report :
               precision    recall  f1-score   support

           0       0.97      0.99      0.98      1808
           1       0.84      0.70      0.76       187

    accuracy                           0.96      1995
   macro avg       0.90      0.84      0.87      1995
weighted avg       0.96      0.96      0.96      1995
```

In [ ]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print(f"Logistic Regression training Accuracy score: {accuracy_score(y_train,y_pred

print(f"Logistic Regression testing Accuracy score: {accuracy_score(y_test,y_pred_l
```

```
Logistic Regression training Accuracy score: 95.9
Logistic Regression testing Accuracy score: 95.9
```

Plot the ROC curve to better identify the threshold by which we would split our data

In [ ]:
```
from sklearn.metrics import roc_curve, auc

train_probs = logistic_model.predict_proba(X_train)[:, 1]
test_probs = logistic_model.predict_proba(X_test)[:, 1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, train_probs)
roc_auc_train = auc(fpr_train, tpr_train)

fpr_test, tpr_test,thresholds_test = roc_curve(y_test, test_probs)
roc_auc_test = auc(fpr_test, tpr_test)

plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, label=f'Train ROC curve (AUC = {roc_auc_train:.2f})'
# Plot ROC curve for test set
plt.plot(fpr_test, tpr_test, label=f'Test ROC curve (AUC = {roc_auc_test:.2f})')

for i, threshold in enumerate(thresholds_test):
    if i % 50 == 0:
        plt.annotate(f'{threshold:.2f}', (fpr_test[i], tpr_test[i]), textcoords="of

plt.plot([0, 1], [0, 1], 'r--', label='Random Classifier')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



Therefore, the best threshold would be at 0.04

In [ ]:
```
threshold=0.04
test_prob=logistic_model.predict_proba(X_test)
y_pred_test_log = (test_prob[:, 1] >= threshold).astype(int)
```

In [ ]:
```
print(f"Logistic Regression testing Accuracy score: {accuracy_score(y_test,y_pred_t
```

```
Logistic Regression testing Accuracy score: 85.8
```

In [ ]:
```
conf_matrix = confusion_matrix(y_test, y_pred_test_log)

# Add labels for better understanding
tn, fp, fn, tp = conf_matrix.ravel()
display( pd.DataFrame(conf_matrix, columns=['Predicted Negative', 'Predicted Positi
```

	Predicted Negative	Predicted Positive
Actual Negative	1535	273
Actual Positive	10	177

In [ ]:
```
print("Classification Report : \n" ,classification_report(y_test, y_pred_test_log))
```

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

           0       0.99      0.85      0.92      1808
           1       0.39      0.95      0.56       187

    accuracy                           0.86      1995
   macro avg       0.69      0.90      0.74      1995
weighted avg       0.94      0.86      0.88      1995
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred_test_log)
precision = precision_score(y_test, y_pred_test_log, average='macro')
recall = recall_score(y_test, y_pred_test_log, average='macro')
f1_score= metrics.f1_score(y_test, y_pred_test_log, average='macro')

EVAL_SCORE = EVAL_SCORE.append({'Model': 'Logistic Regression','Accuracy':accuracy,
EVAL_SCORE
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.964912	0.865674	0.973450	0.910238
1	Random Forest	0.918797	0.958883	0.566845	0.596485
2	Logistic Regression	0.858145	0.693430	0.897764	0.735664

K-NN Classifier Model

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_predict, KFold

knn = KNeighborsClassifier(n_neighbors = 5)
```

In [ ]:
```
knn.fit(X_train, y_train)
```

Out[ ]:
```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

In [ ]:
```
kf = KFold(n_splits=10, random_state=5, shuffle=True)
y_pred_knn = cross_val_predict(knn, X, Y, cv=kf)
```

In [ ]:
```
knn.score(X_test, y_test)
```

Out[ ]:
```
0.9588972431077695
```

In [ ]:
```
k_values = range(1, 50)

#intialize 2 variables
best_k = 0
best_accuracy = 0

#iterate over the values of k and calculate the cross-validation for each, then cal
#accuracy and compare

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    kf = KFold(n_splits=10, random_state=k, shuffle=True)
    y_pred = cross_val_predict(knn, X_test, y_test , cv=kf)
```

```
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
#use cross-validation for the best k provided from the loop
final_knn = KNeighborsClassifier(n_neighbors=best_k)
kf = KFold(n_splits=10, random_state=best_k, shuffle=True)
y_pred_KNN = cross_val_predict(final_knn, X_test, y_test, cv=kf)
```

In [ ]:

```
accuracy_final = accuracy_score(y_test, y_pred_KNN)
precision_final = precision_score(y_test, y_pred_KNN, average='macro')
recall_final = recall_score(y_test, y_pred_KNN, average='macro')
f1_score_final = metrics.f1_score(y_test, y_pred_KNN, average='macro')

print(f"KNN Classification Results (Best k={best_k}):")
print(f"Accuracy: {accuracy_final:.4f}")
print(f"Precision: {precision_final:.4f}")
print(f"Recall: {recall_final:.4f}")
print(f"F1 Score: {f1_score_final:.4f}")
```

```
KNN Classification Results (Best k=1):
Accuracy: 0.9564
Precision: 0.8843
Recall: 0.8489
F1 Score: 0.8655
```

In [ ]:

```
conf_matrix = confusion_matrix(y_test, y_pred_KNN)

# Add labels for better understanding
tn, fp, fn, tp = conf_matrix.ravel()
display( pd.DataFrame(conf_matrix, columns=['Predicted Negative', 'Predicted Positi
```

	Predicted Negative	Predicted Positive
Actual Negative	1774	34
Actual Positive	53	134

In [ ]:

```
EVAL_SCORE = EVAL_SCORE.append({'Model': 'K-NN','Accuracy':accuracy_final,'Precisio
EVAL_SCORE
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.964912	0.865674	0.973450	0.910238
1	Random Forest	0.918797	0.958883	0.566845	0.596485
2	Logistic Regression	0.858145	0.693430	0.897764	0.735664
3	K-NN	0.956391	0.884305	0.848886	0.865498

Linear Regression Model

In [ ]:

```
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

Out[ ]:

```
▾ LinearRegression
LinearRegression()
```

In [ ]:
```
from sklearn.metrics import mean_squared_error

print('linear model coeff (w): {}'
     .format(linreg.coef_))
print('linear model intercept (b): {:.3f}'
     .format(linreg.intercept_))
print('R-squared score (training): {:.3f}'
     .format(linreg.score(X_train, y_train)))
print('R-squared score (test): {:.3f}'
     .format(linreg.score(X_test, y_test)))

y_pred_LR=linreg.predict(X_test)
print('MSE (test): {:.3f}'
     .format(mean_squared_error(y_test, y_pred_LR)))
```

```
linear model coeff (w): [ 0.00985661  0.07236611  0.0357375   0.01712037  0.072341
87  0.00831916
 -0.02275375  0.08017948 -0.01209817 -0.02220952 -0.00439635  0.07719119]
linear model intercept (b): -0.024
R-squared score (training): 0.385
R-squared score (test): 0.399
MSE (test): 0.051
```

We won't be able to add it to our evaluation table since linear regression is assessed by MSE and R^2 which are provided above rather than accuracy, precision, recall, and F1 score

Naive Bayes

In [ ]:
```
from sklearn.naive_bayes import GaussianNB

nbclf = GaussianNB()
nbclf.fit(X_train, y_train)
```

Out[ ]:
```
▾ GaussianNB

GaussianNB()
```

In [ ]:
```
print('Accuracy of GaussianNB classifier on training set: {:.2f}'
     .format(nbclf.score(X_train, y_train)))
print('Accuracy of GaussianNB classifier on test set: {:.2f}'
     .format(nbclf.score(X_test, y_test)))
```

```
Accuracy of GaussianNB classifier on training set: 0.88
Accuracy of GaussianNB classifier on test set: 0.90
```

In [ ]:
```
y_pred_NB=nbclf.predict(X_test)
```

In [ ]:
```
conf_matrix = confusion_matrix(y_test, y_pred_NB)

# Add labels for better understanding
tn, fp, fn, tp = conf_matrix.ravel()
display( pd.DataFrame(conf_matrix, columns=['Predicted Negative', 'Predicted Positi
```

	Predicted Negative	Predicted Positive
Actual Negative	1633	175
Actual Positive	34	153

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred_NB)
precision = precision_score(y_test, y_pred_NB, average='macro')
```

```
recall = recall_score(y_test, y_pred_NB, average='macro')
f1_score= metrics.f1_score(y_test, y_pred_NB, average='macro')

EVAL_SCORE = EVAL_SCORE.append({'Model': 'Naive Bayes','Accuracy':accuracy,'Precisi
EVAL_SCORE
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.964912	0.865674	0.973450	0.910238
1	Random Forest	0.918797	0.958883	0.566845	0.596485
2	Logistic Regression	0.858145	0.693430	0.897764	0.735664
3	K-NN	0.956391	0.884305	0.848886	0.865498
4	Naive Bayes	0.895238	0.723034	0.860695	0.767015

Assessing our Models

In [ ]:

```
EVAL_SCORE
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.964912	0.865674	0.973450	0.910238
1	Random Forest	0.918797	0.958883	0.566845	0.596485
2	Logistic Regression	0.858145	0.693430	0.897764	0.735664
3	K-NN	0.956391	0.884305	0.848886	0.865498
4	Naive Bayes	0.895238	0.723034	0.860695	0.767015

Sorting our evaluation table descendingly according to the F1 score

In [ ]:

```
EVAL_SCORE.sort_values(by='F1 Score', inplace = True, ascending= False)
EVAL_SCORE
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision Tree	0.964912	0.865674	0.973450	0.910238
3	K-NN	0.956391	0.884305	0.848886	0.865498
4	Naive Bayes	0.895238	0.723034	0.860695	0.767015
2	Logistic Regression	0.858145	0.693430	0.897764	0.735664
1	Random Forest	0.918797	0.958883	0.566845	0.596485

In [ ]:

```
plt.figure(figsize=(30,28))
plt.subplot(2,2,1)

acc = EVAL_SCORE.groupby('Model')['Accuracy'].mean()
acc.plot(kind='bar',color=sns.palettes.mpl_palette('Dark2'))
plt.xticks(fontsize=15)
plt.ylabel('Accuracy',fontsize=15)
plt.title('Model Accuracy',fontsize=20, fontweight="bold")

plt.subplot(2,2,2)
```

```
pr = EVAL_SCORE.groupby('Model')['Precision'].mean()
pr.plot(kind='bar',color=sns.palettes.mpl_palette('Dark2'))
plt.xticks(fontsize=15)
plt.ylabel('Precision',fontsize=15)
plt.title('Model Precision',fontsize=20, fontweight="bold")

plt.subplot(2,2,3)

rc = EVAL_SCORE.groupby('Model')['Recall'].mean()
rc.plot(kind='bar',color=sns.palettes.mpl_palette('Dark2'))
plt.xticks(fontsize=15)
plt.ylabel('Recall',fontsize=15)
plt.title('Model Recall',fontsize=20, fontweight="bold")

plt.subplot(2,2,4)

f1 = EVAL_SCORE.groupby('Model')['F1 Score'].mean()
f1.plot(kind='bar',color=sns.palettes.mpl_palette('Dark2'))
plt.xticks(fontsize=15)
plt.ylabel('F1 Score',fontsize=15)
plt.title('Model F1 Score',fontsize=20, fontweight="bold")

plt.subplots_adjust(wspace=0.5,hspace=0.5)
plt.show()
```