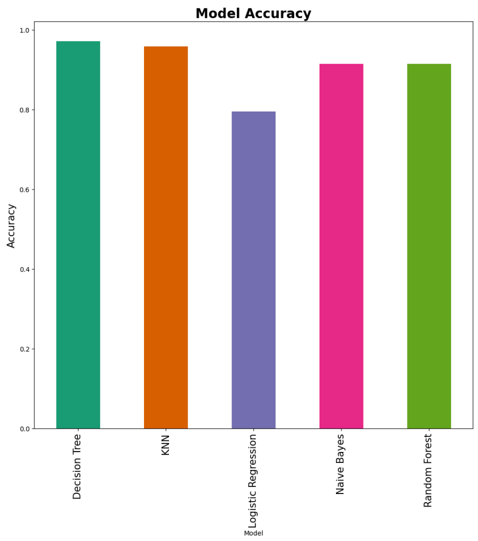
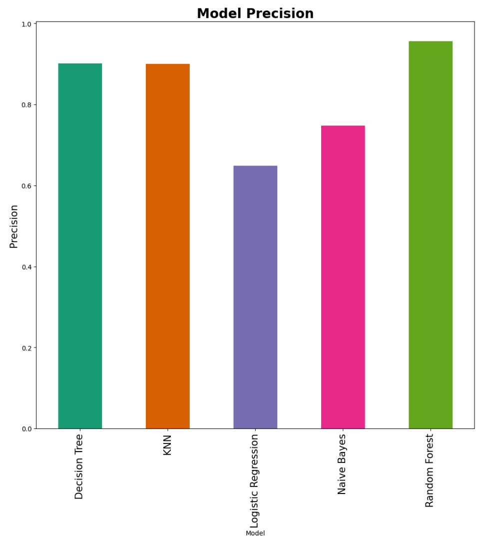
# Phase 4

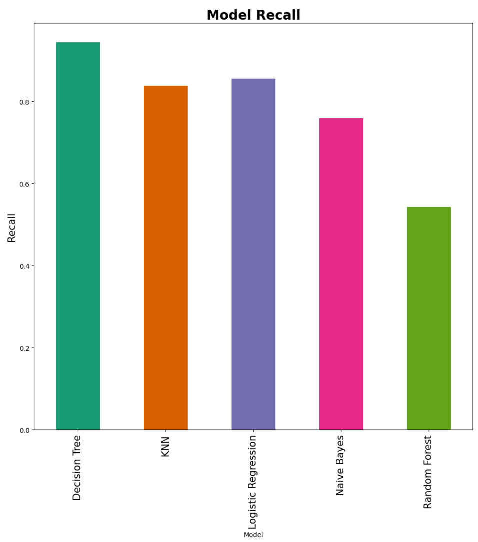
**Sama Amr -- 900211296 & Farida Madkour -- 900211360**

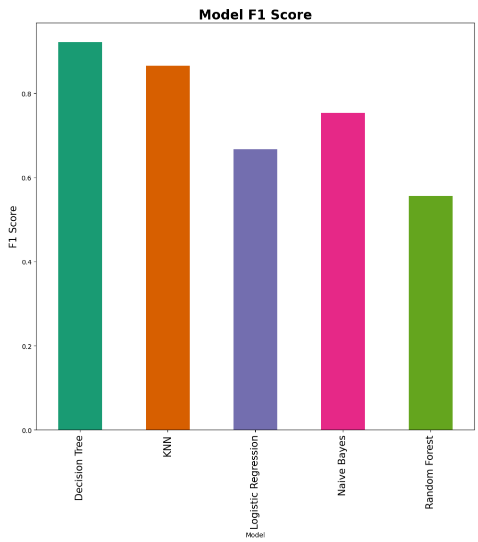
## As previously suggested by the graphs and the metric measures in the previous phases the Decision Tree shows the best performance accross all; therefore we decided to go further on with the Decision Tree as our Classification Model

| Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Decision Tree | 0.97193 | 0.901939 | 0.94376 | 0.921502 |
| KNN | 0.957895 | 0.900368 | 0.83773 | 0.86568 |
| Naive Bayes | 0.914286 | 0.747896 | 0.758533 | 0.753058 |
| Logistic Regression | 0.795489 | 0.648342 | 0.856004 | 0.667079 |
| Random Forest | 0.914286 | 0.956796 | 0.542781 | 0.55624 |




In [1]:
```
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, GridSearchCV,
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,
```

In [2]:
```
df = pd.read_csv("bankloan.csv")
df.head()
```

Out[2]:

| | ID | Age | Experience | Income | ZIP.Code | Family | Education | Securities.Account | CD.Accou |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 25 | 1 | 49 | 91107 | 4 | 1 | 1 | |
| **1** | 2 | 45 | 19 | 34 | 90089 | 3 | 1 | 1 | |
| **2** | 3 | 39 | 15 | 11 | 94720 | 1 | 1 | 0 | |
| **3** | 4 | 35 | 9 | 100 | 94112 | 1 | 2 | 0 | |
| **4** | 5 | 35 | 8 | 45 | 91330 | 4 | 2 | 0 | |

***Data pre-processing***

In [3]:
```
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   Education           5000 non-null   int64  
 7   Securities.Account  5000 non-null   int64  
 8   CD.Account          5000 non-null   int64  
 9   Online              5000 non-null   int64  
 10  CreditCard          5000 non-null   int64  
 11  CCAvg               5000 non-null   float64
 12  Mortgage            5000 non-null   int64  
 13  Personal.Loan       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', 'Education',
       'Securities.Account', 'CD.Account', 'Online', 'CreditCard', 'CCAv
g',
       'Mortgage', 'Personal.Loan'],
      dtype='object')
Shape: (5000, 14)
```

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

In [4]:
```python
df = df.drop(columns=['ID','Experience','ZIP.Code'])
df.head()
```

Out[4]:

| | Age | Income | Family | Education | Securities.Account | CD.Account | Online | CreditCard | CC |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | 49 | 4 | 1 | 1 | 0 | 0 | 0 | |
| **1** | 45 | 34 | 3 | 1 | 1 | 0 | 0 | 0 | |
| **2** | 39 | 11 | 1 | 1 | 0 | 0 | 0 | 0 | |
| **3** | 35 | 100 | 1 | 2 | 0 | 0 | 0 | 0 | |
| **4** | 35 | 45 | 4 | 2 | 0 | 0 | 0 | 1 | |

Check for missing values

In [5]:
```python
df.isnull().sum()
```

Out[5]:
```
Age                   0
Income                0
Family                0
Education             0
Securities.Account    0
CD.Account            0
Online                0
CreditCard            0
CCAvg                 0
Mortgage              0
Personal.Loan         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 [6]:
```python
df.duplicated().sum()
```

Out[6]: 13

In [7]:
```python
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

Out[7]: 0

Encodings

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

In [8]:
```
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   Education           4987 non-null   category
 4   Securities.Account  4987 non-null   category
 5   CD.Account          4987 non-null   category
 6   Online              4987 non-null   category
 7   CreditCard          4987 non-null   category
 8   CCAvg               4987 non-null   float64
 9   Mortgage            4987 non-null   float64
 10  Personal.Loan       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 [9]:
```
#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', '40-5(

#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_Cl
df.head()
```

Out[9]:

| | Age | Income | Family | Education | Securities.Account | CD.Account | Online | CreditCard | CC |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 25 | 49.0 | 4 | 1 | 1 | 0 | 0 | 0 | |
| **1** | 45 | 34.0 | 3 | 1 | 1 | 0 | 0 | 0 | |
| **2** | 39 | 11.0 | 1 | 1 | 0 | 0 | 0 | 0 | |
| **3** | 35 | 100.0 | 1 | 2 | 0 | 0 | 0 | 0 | |
| **4** | 35 | 45.0 | 4 | 2 | 0 | 0 | 0 | 1 | |

In [10]:
```
df['Age']=df['Age_r']
df['Income']=df['Income_r']
df.drop(columns=['Age_r', 'Income_r'], inplace=True)
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   category
 1   Income              4987 non-null   category
 2   Family              4987 non-null   category
 3   Education           4987 non-null   category
 4   Securities.Account  4987 non-null   category
 5   CD.Account          4987 non-null   category
 6   Online              4987 non-null   category
 7   CreditCard          4987 non-null   category
 8   CCAvg               4987 non-null   float64
 9   Mortgage            4987 non-null   float64
 10  Personal.Loan       4987 non-null   category
dtypes: category(9), float64(2)
memory usage: 162.1 KB
```

Unique values of each of the variables

In [11]:
```python
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]
```

Function to identify outliers

In [12]:
```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 [13]:

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

```
Number of outliers in CCAvg : 301 ,It's Percentage is :  6.03569280128333
7 %


Number of outliers in Mortgage : 291 ,It's Percentage is :  5.83517144575
8974 %
```

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:

The best suitable transformation for:

- CCAvg: Cubic root
- Mortgage: Square root

In [14]:

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

Test for outliers after the transformation and plot the histograms

In [15]:

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

```
Number of outliers in CCAvg : 109 ,It's Percentage is :  2.18568277521556
05 %


Number of outliers in Mortgage : 1 ,It's Percentage is :  0.0200521355524
36335 %
```

Outliers are significantly reduced after the transformation

Finally, here's a summary of our continous features

In [16]:
```
df.describe()
```

Out[16]:

| | CCAvg | Mortgage |
|---|---|---|
| **count** | 4987.000000 | 4987.000000 |
| **mean** | 1.127270 | 4.046775 |
| **std** | 0.392599 | 6.346477 |
| **min** | 0.000000 | 0.000000 |
| **25%** | 0.887904 | 0.000000 |
| **50%** | 1.144714 | 0.000000 |
| **75%** | 1.375069 | 10.049876 |
| **max** | 2.154435 | 25.199206 |

# Decision Tree Model

In [17]:
```
df1 = df.copy()

df1['Income'] = pd.factorize(df1['Income'])[0] + 1
df1['Age'] = pd.factorize(df1['Age'])[0] + 1

df1['Income'] = df1['Income'].astype(int)
df1['Age'] = df1['Age'].astype(int)


print(df1.dtypes)

df1
```

```
Age                      int32
Income                   int32
Family                category
Education             category
Securities.Account    category
CD.Account            category
Online                category
CreditCard            category
CCAvg                  float64
Mortgage               float64
Personal.Loan         category
dtype: object
```

Out[17]:

| | Age | Income | Family | Education | Securities.Account | CD.Account | Online | CreditCard |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 4 | 1 | 1 | 0 | 0 | 0 |
| **1** | 2 | 1 | 3 | 1 | 1 | 0 | 0 | 0 |
| **2** | 3 | 2 | 1 | 1 | 0 | 0 | 0 | 0 |
| **3** | 3 | 1 | 1 | 2 | 0 | 0 | 0 | 0 |
| **4** | 3 | 1 | 4 | 2 | 0 | 0 | 0 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **4995** | 1 | 1 | 1 | 3 | 0 | 0 | 1 | 0 |
| **4996** | 1 | 2 | 4 | 1 | 0 | 0 | 1 | 0 |
| **4997** | 5 | 1 | 2 | 3 | 0 | 0 | 0 | 0 |
| **4998** | 5 | 1 | 3 | 2 | 0 | 0 | 1 | 0 |
| **4999** | 1 | 1 | 3 | 1 | 0 | 0 | 1 | 1 |

4987 rows × 11 columns

Standardize our variable

In [18]:
```
from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler()

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

df_scaled.head()
```

Out[18]:

| | Age | Income | Family | Education | Securities.Account | CD.Account | Online | Cr |
|---|---|---|---|---|---|---|---|---|
| 0 | -1.642038 | -0.624504 | 1.397399 | -1.047290 | 2.924661 | -0.253892 | -1.214976 | -( |
| 1 | -0.812956 | -0.624504 | 0.525860 | -1.047290 | 2.924661 | -0.253892 | -1.214976 | -( |
| 2 | 0.016126 | 0.240848 | -1.217219 | -1.047290 | -0.341920 | -0.253892 | -1.214976 | -( |
| 3 | 0.016126 | -0.624504 | -1.217219 | 0.143778 | -0.341920 | -0.253892 | -1.214976 | -( |
| 4 | 0.016126 | -0.624504 | 1.397399 | 0.143778 | -0.341920 | -0.253892 | -1.214976 | |

Checking if there are any missing values

In [19]:
```
df_scaled.isnull().sum()
```

Out[19]:
```
Age                   0
Income                0
Family                0
Education             0
Securities.Account    0
CD.Account            0
Online                0
CreditCard            0
CCAvg                 0
Mortgage              0
Personal.Loan         0
dtype: int64
```

Assigning our target and decision variables

In [20]:
```
Y = df_scaled['Personal.Loan']
X = df_scaled.drop(['Personal.Loan'],axis=1)
```

In [21]:
```
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, ran
```

In [22]:
```
print (" Number of columns in our Features : ", X.shape[1])
```

```
 Number of columns in our Features :  10
```

***Solving the Class imbalance problem***

In [23]:
```
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 [24]:
```
print("Before UpSampling, counts of Personal loan = '0': {}".format(sum(y_t
print("Before UpSampling, counts of Personal loan = '1': {} \n".format(sum(


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

```
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 [25]:
```
EVAL_SCORE = pd.DataFrame(columns=['Model','Accuracy','Precision','Recall'

EVAL_SCORE
```

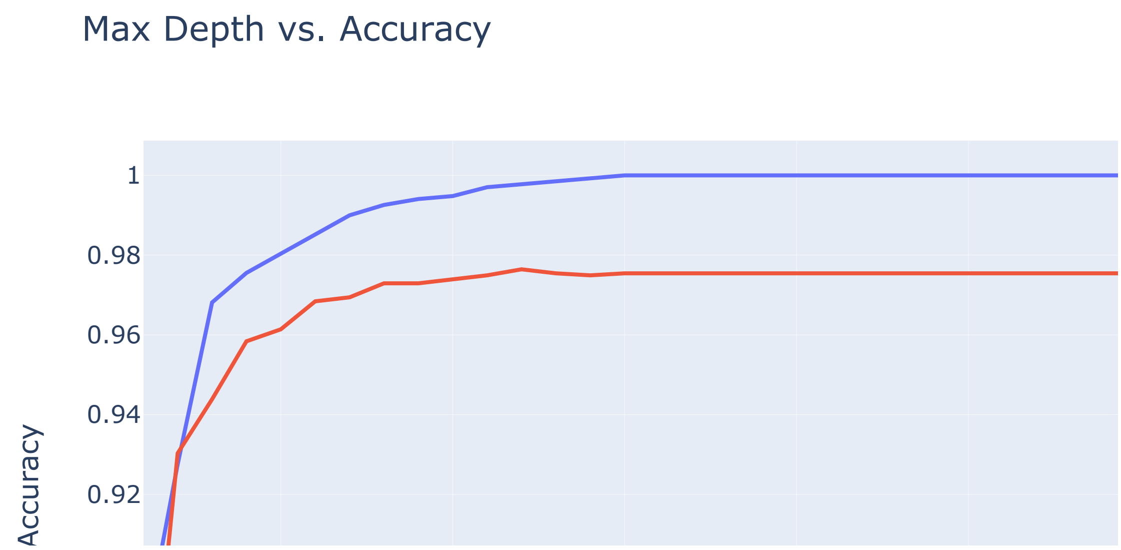
Out[25]:

| Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|

***Decision Tree Model***

In [26]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusio
from sklearn.model_selection import cross_val_score
import plotly.graph_objects as go
```

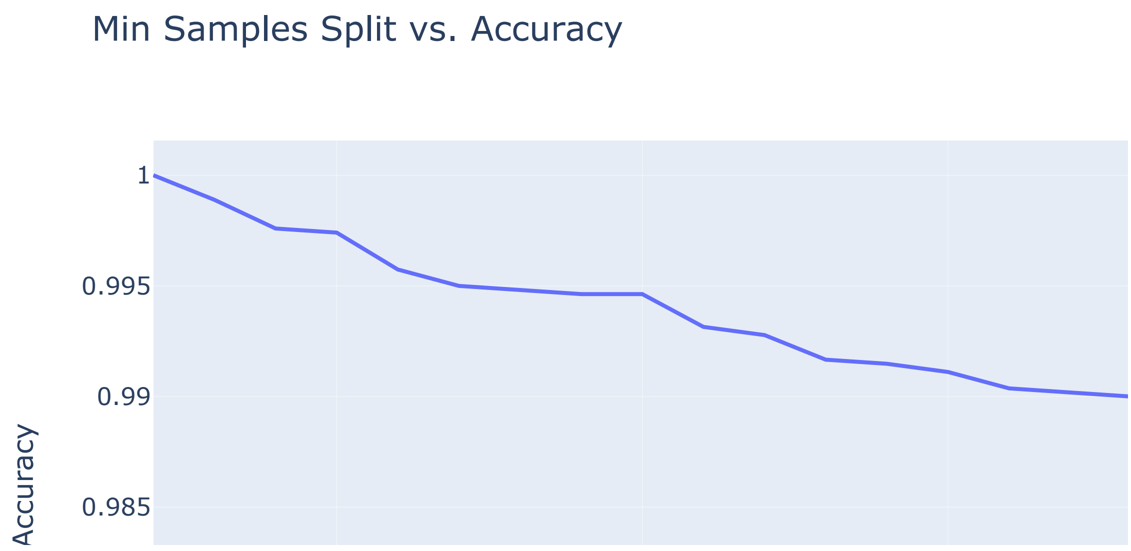
In [27]:
```
max_depth_values = range(1, 50)

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='li

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

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 [28]:

```
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,

fig.add_trace(go.Scatter(x=list(min_samples_split_values), y=test_scores, n

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 [29]:
```
Decision_Tree = DecisionTreeClassifier(max_depth=3,criterion='entropy',ranc
```

In [30]:
```
Decision_Tree.fit(X_train_upsampled, y_train_upsampled)
```

Out[30]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=4
2)
```

In [31]:
```
y_pred_train = Decision_Tree.predict(X_train_upsampled)
y_pred_test = Decision_Tree.predict(X_test)
```

In [32]:

```
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 : 97.0
 Decision Tree Testing Accuracy : 97.0
```

Using cross validation on our descision tree and testing the accuracy

In [33]:

```
cv_scores_train = cross_val_score(Decision_Tree, X_train_upsampled, y_trair
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:   [0.95648148 0.95833333 0.9759
2593 0.97034291 0.97405005]
 Mean Accuracy from Cross-Validation :  0.967026739436378
```

In [34]:

```
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', 'Predicte
```

| | Predicted Negative | Predicted Positive |
|---|---|---|
| **Actual Negative** | 1769 | 39 |
| **Actual Positive** | 17 | 170 |

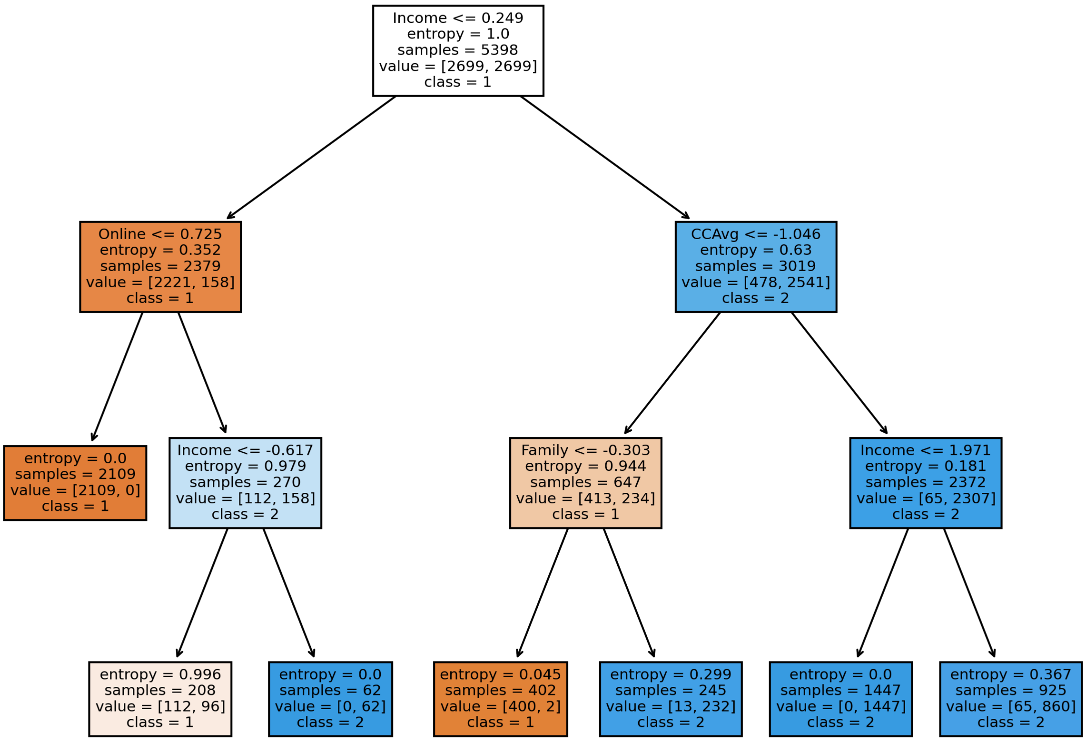
In [35]:

```
print("Classification Report : \n" ,classification_report(y_test, y_pred_te
```

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

           0       0.99      0.98      0.98      1808
           1       0.81      0.91      0.86       187

    accuracy                           0.97      1995
   macro avg       0.90      0.94      0.92      1995
weighted avg       0.97      0.97      0.97      1995
```

In [36]:
```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

cn=["1","2"]
fn=['Age','Income','Family','CCAvg','Education','Mortgage','Securities.Acc

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()
```

Decision Tree Model




In [37]:
```python
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')

new=pd.Series({'Model': 'Decision Tree','Accuracy':accuracy,'Precision':pre
EVAL_SCORE=pd.concat([EVAL_SCORE,new.to_frame().T], ignore_index=True)
EVAL_SCORE
```

Out[37]:

| | Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|
| **0** | Decision Tree | 0.97193 | 0.901939 | 0.94376 | 0.921502 |

# Model Design (Phase 4)

***Optimizing the parameters***

Testing using the f1 score since it's the most robust

In [38]:
```
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

dt_classifier = DecisionTreeClassifier()

grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring='f1')

grid_search.fit(X_train_upsampled, y_train_upsampled)

print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

best_dt_model = grid_search.best_estimator_
test_accuracy = best_dt_model.score(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
f': 1, 'min_samples_split': 2}
Best Score: 0.9862465018197307
Test Accuracy: 0.9759398496240601
```

In [39]:
```
Decision_Tree_opt = DecisionTreeClassifier(criterion= 'gini', max_depth= 1(
Decision_Tree_opt.fit(X_train_upsampled, y_train_upsampled)
```

Out[39]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10)
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.tree.Dec

In [40]:
```
y_pred_train_opt = Decision_Tree_opt.predict(X_train_upsampled)
y_pred_test_opt = Decision_Tree_opt.predict(X_test)
train_accuracy = accuracy_score(y_train_upsampled, y_pred_train_opt)
print(" Optimized Decision Tree Training Accuracy :" ,round(train_accuracy,

test_accuracy = accuracy_score(y_test, y_pred_test_opt)
print(" Optimized Decision Tree Testing Accuracy :" ,round(test_accuracy,2)
```

```
 Optimized Decision Tree Training Accuracy : 99.0
 Optimized Decision Tree Testing Accuracy : 97.0
```

In [41]:
```
cv_scores_train = cross_val_score(Decision_Tree_opt, X_train_upsampled, y_t
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:   [0.97159091 0.9862259  0.9926
3352 0.98897059 0.98895028]
 Mean Accuracy from Cross-Validation :  0.9856742372762997
```

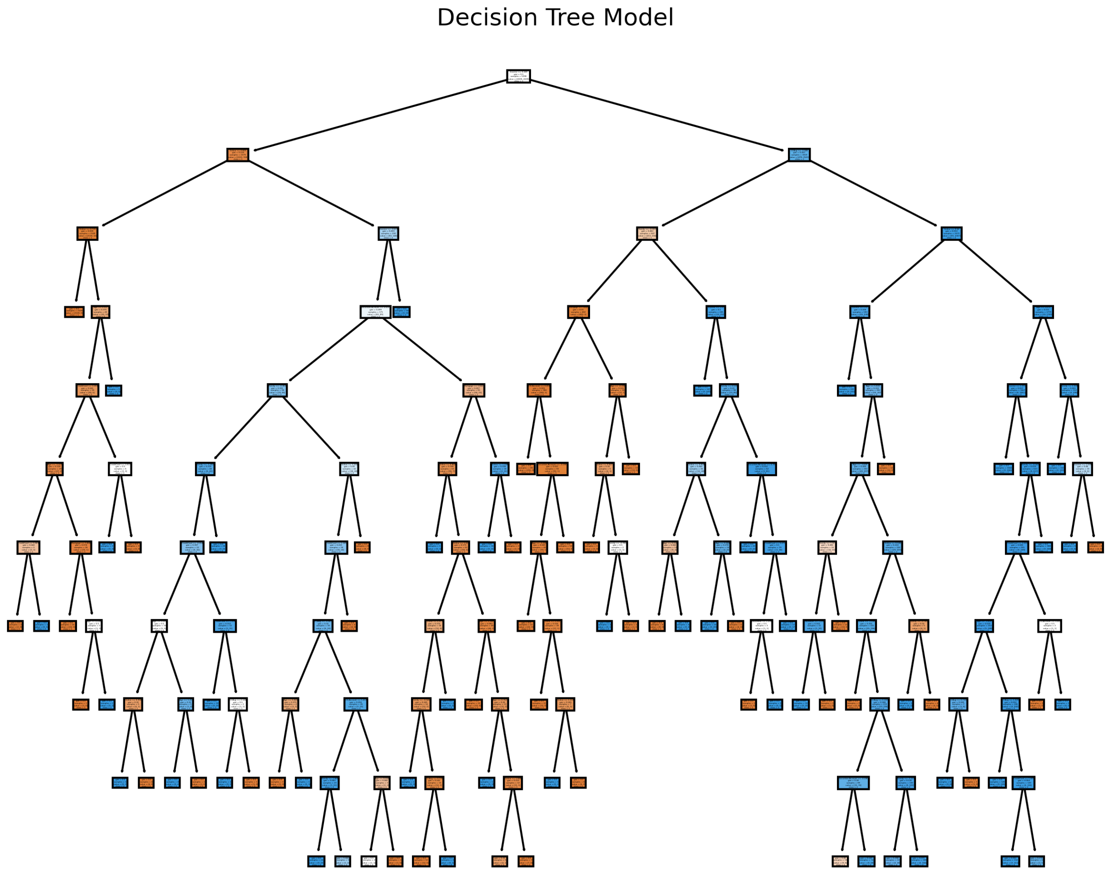
In [42]:
```
conf_matrix = confusion_matrix(y_test, y_pred_test_opt)

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

| | Predicted Negative | Predicted Positive |
|---|---|---|
| **Actual Negative** | 1780 | 28 |
| **Actual Positive** | 22 | 165 |

In [43]:
```
print("Classification Report : \n" ,classification_report(y_test, y_pred_te
```

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

           0       0.99      0.98      0.99      1808
           1       0.85      0.88      0.87       187

    accuracy                           0.97      1995
   macro avg       0.92      0.93      0.93      1995
weighted avg       0.98      0.97      0.98      1995
```

In [44]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

cn=["1","2"]
fn=['Age','Income','Family','CCAvg','Education','Mortgage','Securities.Acc

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




In [45]:
```
accuracy_opt = accuracy_score(y_test, y_pred_test_opt)
precision_opt = precision_score(y_test, y_pred_test_opt, average='macro')
recall_opt = recall_score(y_test, y_pred_test_opt, average='macro')
f1_score_opt= metrics.f1_score(y_test, y_pred_test_opt, average='macro')

new=pd.Series({'Model': 'Optimized Decision Tree','Accuracy':accuracy_opt,'
EVAL_SCORE=pd.concat([EVAL_SCORE,new.to_frame().T], ignore_index=True)
EVAL_SCORE
```

Out[45]:

| | Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|
| **0** | Decision Tree | 0.97193 | 0.901939 | 0.94376 | 0.921502 |
| **1** | Optimized Decision Tree | 0.974937 | 0.921357 | 0.933433 | 0.927285 |

# Implementing the Decision Tree Model from scratch (Not using python libraries)

In [46]:
```
class Node:
    def __init__(self,split_feature=None,split_value=None):
        self.split_feature= split_feature
        self.split_value = split_value
        self.label=None
        self.children =[]

class DTree:
    def calc_entropy(self,y):
        m = len(y)
        unique_labels, counts= np.unique(y,return_counts=True)
        entropy = [-1*counts[i]/m* np.log2(counts[i]/m) for i in range(len(
        total_entropy= np.sum(entropy)

        return total_entropy

    def get_best_feature(self,X,y,features):
        # calc total entropy
        total_entropy= self.calc_entropy(y)

        d= dict()
        # calc gain for each feature
        for feature in features:
            values= np.unique(X[feature])
            mi = 0
            for value in values:
                split_y = y[X[feature] == value]
                mi+= (len(split_y)/len(y)) * self.calc_entropy(split_y)

            d[feature]= total_entropy-mi

        s = dict(sorted(d.items(), key= lambda x: x[1], reverse= True))


        return list(s.keys())[0]




    def train_tree(self,X,y,node,features):
        if len(np.unique(y))==1:
            node.label= np.unique(y)
            return node

        elif X.empty:
            node.label = np.unique(y)[np.argmax(np.unique(y,return_counts=T
            return node


        best_feature= self.get_best_feature(X,y,features)
        node.split_feature= best_feature

        values= np.unique(X[best_feature])
        for value in values: #loop every branch
            split_x = X[X[best_feature] == value].drop(best_feature,axis=1)
            split_y = y[X[best_feature] == value]

            new_node= Node()
            new_node.split_value= value
            node.children.append(self.train_tree(split_x, split_y,new_node,
```

```python
        return node

    def fit(self,X,y):
        node= Node()
        self.tree= self.train_tree(X,y,node,X.columns)

        return self

    def predict(self,x_test):
        node= self.tree
        while True:
            if len(node.children) == 0:
                return node.label

            for child in node.children:
                if x_test[node.split_feature]== child.split_value:
                    node = child
                    break
```

In [47]:
```python
clf= DTree()
clf.fit(X_train_upsampled, y_train_upsampled)
```

Out[47]:
```
<__main__.DTree at 0x22d77d053d0>
```

# Implementation

In [48]:
```python
import pickle

Model = pickle.dumps(Decision_Tree_opt)
with open('model.pkl', 'wb') as file:
    file.write(Model)
```

In [54]:

```
from tkinter import *

window = Tk()
window.rowconfigure(0, weight=1)
window.columnconfigure(0, weight=1)
window.state('zoomed')

# Creating 3 frames that will be shuffled around in our application
page1 = Frame(window)
page2 = Frame(window)
page3 = Frame(window)

for frame in (page1, page2, page3):
    frame.grid(row=0, column=0, sticky='nsew')

def show_frame(frame):
    frame.tkraise()

show_frame(page1)
y_pred_pkl = None

# ============= Page 1 =========

# Adjusting the general shape of our gui
canvas = Canvas(
    page1,
    bg="#FFFFFF",
    height=982,
    width=1512,
    bd=0,
    highlightthickness=0,
    relief="ridge"
)
canvas.place(x=0, y=0)
canvas.create_rectangle(
    0.0,
    0.0,
    1512.0,
    123.0,
    fill="#8005CC",
    outline=""
)

canvas.create_text(
    32.0,
    42.0,
    anchor="nw",
    text="Loan Approvals ",
    fill="#FFFFFF",
    font=("Junge Regular", 50 * -1)
)

# declaring string variable for storing name and password
name_var=StringVar()
passw_var=StringVar()

error = Label(page1, text="", fg="red", bg="#FFFFFF")
error.place(x=500.0, y=600.0)

# Function to be used at the Login Button where it shifts to the next frame
# otherwise it turns entryboxes red if either is missing, and prints login
```

```
def submit():
    name = name_var.get()
    password = passw_var.get()

    name_entry.config(bg="white")
    passw_entry.config(bg="white")

    if not (name and password) or not (isinstance(name, str) and isinstance
        error.config(text="Invalid input, try again", fg="red", font=("Mont

        if not name:
            name_entry.config(bg="red")
        if not password:
            passw_entry.config(bg="red")

        return

    print("The name is: " + name)
    print("The password is: " + password)

    name_var.set("")
    passw_var.set("")

    show_frame(page2)

    error.config(text="")

name_label = Label(page1, text = 'Username', font=('calibre',20),fg="black'
name_label.place(x=200,y=160)

name_entry = Entry(page1,textvariable = name_var, font=('calibre',20,'norma
name_entry.place(x=200,y=200)

passw_label = Label(page1, text = 'Password', font = ('calibre',20),fg="bla
passw_label.place(x=200,y=310)

passw_entry=Entry(page1, textvariable = passw_var, font = ('calibre',20,'no
passw_entry.place(x=200,y=350)

Login = Button(
    page1,
    text='Login',
    font=("Montserrat Medium", int(20.0)),
    borderwidth=0,
    highlightthickness=0,
    command=submit,
    relief="flat",
    fg='white',
    bg='#8105CC',
    activebackground='#DABCFF'
)
Login.place(
    x=450.0,
    y=550.0,
    width=327.0,
    height=44.0
)

# ======== Page 2 ===========
canvas = Canvas(
    page2,
```

```
    bg="#FFFFFF",
    height=982,
    width=1512,
    bd=0,
    highlightthickness=0,
    relief="ridge"
)
canvas.place(x=0, y=0)
canvas.create_rectangle(
    0.0,
    0.0,
    1512.0,
    123.0,
    fill="#8005CC",
    outline="")

canvas.create_text(
    32.0,
    42.0,
    anchor="nw",
    text="Loan Approvals ",
    fill="#FFFFFF",
    font=("Junge Regular", 50 * -1)
)

canvas.create_text(
    400.0,
    89.0,
    anchor="nw",
    text="1 Entry",
    fill="#FFFFFF",
    font=("Junge Regular", 25 * -1)
)

# Dropdown boxes options
options = {
    "Age": ["Select Age", "23-30", "30-40", "40-50", "50-60", "60-70"],
    "Income": ["Select Income", "Poor", "Middle_Class", "Upper_Class", "Ric
    "Family": ["Select Family", "1", "2", "3", "4"],
    "Education": ["Select Education", "1", "2", "3"],
    "Securities Account": ["Securities Account", "1", "0"],
    "CD Account": ["CD Account", "1", "0"],
    "Online": ["Online", "1", "0"],
    "Credit Card": ["Credit Card", "1", "0"]
}

selected_options = {key: StringVar(page2) for key in options.keys()}

# Creating the different dropdowns and assigning each to their correspondin
dropdown_menus = []
for index, (label_text, option_values) in enumerate(options.items(), start=
    selected_option = selected_options[label_text]
    selected_option.set(option_values[0])

    dropdown_menu = OptionMenu(page2, selected_option, *option_values)
    dropdown_menu.config(width=19)
    dropdown_menu.config(height=2)
    dropdown_menu.config(font=("Montserrat Medium", int(14.0)))

    dropdown_menus.append(dropdown_menu)
```

```
    if index==0:
        dropdown_menu.place(x=20, y=200)
    if index==1:
        dropdown_menu.place(x=340, y=200)
    if index==2:
        dropdown_menu.place(x=680, y=200)
    if index==3:
        dropdown_menu.place(x=1020, y=200)

    if index==4:
        dropdown_menu.place(x=20, y=340)
    if index==5:
        dropdown_menu.place(x=340, y=340)
    if index==6:
        dropdown_menu.place(x=680, y=340)
    if index==7:
        dropdown_menu.place(x=1020, y=340)


error_label = Label(page2, text="", fg="red", bg="#FFFFFF")
error_label.place(x=500.0, y=600.0)

# Creating entry boxes for the numeric variables (CCAvg and Mortgage)
CCAVG_var=IntVar()
MORTGAGE_var=IntVar()

CCAVG_var.set("")
MORTGAGE_var.set("")

CCAVG_label = Label(page2, text = 'CCAvg', font=('calibre',20),fg="black",
CCAVG_label.place(x=150,y=450)

CCAVG_entry=Entry(page2, textvariable = CCAVG_var, font = ('calibre',20,'nc
CCAVG_entry.place(x=300,y=450)

MORTGAGE_label = Label(page2, text = 'Mortgage', font=('calibre',20),fg="bl
MORTGAGE_label.place(x=600,y=450)

MORTGAGE_entry=Entry(page2, textvariable = MORTGAGE_var, font = ('calibre',
MORTGAGE_entry.place(x=750,y=450)

def add_to_answer():
    return [selected_option.get() for selected_option in selected_options.v

# Function to validate that the entry boxes aren't empty and that they are
def validate_entry(entry):
    entry_value = entry.get()
    if not entry_value:
        entry.config(bg="red")
        return False
    elif not entry_value.replace('.', '').isdigit():
        entry.config(bg="red")
        return False
    else:
        entry.config(bg="white")
        return True

# Function used at the decision button where the dropdowns and entry boxes
# next frame; in addition to saving all the answers inputed by the used and
def check():
    global y_pred_pkl
```

```
    answer = add_to_answer()
    all_filled = all(value != options[list(options.keys())[index]][0] for i
    ccavg_valid = validate_entry(CCAVG_entry)
    mortgage_valid = validate_entry(MORTGAGE_entry)
    if all_filled:
        for dropdown_menu in dropdown_menus:
            dropdown_menu.config(bg="white")
        error_label.config(text="")

        for i in range(1, 6):
            if answer[0] == options['Age'][i]:
                answer[0] = i
            if answer[1] == options['Income'][i]:
                answer[1] = i
        if not (ccavg_valid and mortgage_valid):
                error_label.config(text="Missing input, try again", fg="red
                return
        show_frame(page3)

        entry_values = [CCAVG_var.get(), MORTGAGE_var.get()]
        answer += entry_values

        pickled_model = pickle.load(open('model.pkl', 'rb'))
        answer1=standard_scaler.fit_transform(np.array(answer).reshape(-1,1
        y_pred_pkl = pickled_model.predict(answer1)
        show_text(y_pred_pkl)
        print("Input:",answer)
        print("Scaled Input:",answer1)

        #Retraining the model
        initial_shape = X_train.shape
        X_train_updated = pd.concat([X_train, pd.DataFrame(answer1, columns
        if X_train_updated.shape != initial_shape:
            y_pred_pkl1=pd.Series(y_pred_pkl)
            y_train1=pd.concat([y_train, y_pred_pkl1], ignore_index=True)
            Decision_Tree_opt.fit(X_train_updated, y_train1)
            Model = pickle.dumps(Decision_Tree_opt)
            with open('model.pkl', 'wb') as file:
                file.write(Model)

    else:
        for index, value in enumerate(answer):
            if value == options[list(options.keys())[index]][0]:
                dropdown_menus[index].config(bg="red")
            else:
                dropdown_menus[index].config(bg="white")

        if not (ccavg_valid and mortgage_valid):
            error_label.config(text="Missing input, try again", fg="red", f
            return
        error_label.config(text="Missing input, try again", fg="red", font=



Decision = Button(
    page2,
    text = 'Decision',
    font=("Montserrat Medium", int(20.0)),
    borderwidth=0,
    highlightthickness=0,
    command=check,
```

```
    relief="flat",
    fg='white',
    bg='#8105CC',
    activebackground='#DABCFF'
)
Decision.place(
    x=450.0,
    y=550.0,
    width=327.0,
    height=44.0
)

# ======== Page 3 ===========
canvas = Canvas(
    page3,
    bg="#FFFFFF",
    height=982,
    width=1512,
    bd=0,
    highlightthickness=0,
    relief="ridge"
)
canvas.place(x=0, y=0)
canvas.create_rectangle(
    0.0,
    0.0,
    1512.0,
    123.0,
    fill="#8005CC",
    outline=""
)

canvas.create_text(
    32.0,
    42.0,
    anchor="nw",
    text="Loan Approvals ",
    fill="#FFFFFF",
    font=("Junge Regular", 50 * -1)
)

text_label = Label(page3, text="")
text_label.place(x=370.0, y=300.0)

# Function that displayes the decision according to the output of the Model
def show_text(z):
    if z == 0:
        text_label.config(text="Disapprove Loan",bg='white', fg="red", font
        print("Disapprove Loan")
    elif z == 1:
        text_label.config(text="Approve Loan", bg='white', fg="green", font
        print("Approve Loan")

Back = Button(
    page3,
    text = 'Back',
    font=("Montserrat Medium", int(20.0)),
    borderwidth=0,
    highlightthickness=0,
    command=lambda:show_frame(page2),
    relief="flat",
```

```python
    fg='white',
    bg='#8105CC',
    activebackground='#DABCFF'
)
Back.place(
    x=450.0,
    y=550.0,
    width=327.0,
    height=44.0
)
window.mainloop()
```

```
The name is: Sama Amr
The password is: sama
Disapprove Loan
Input: [3, 2, '2', '1', '0', '1', '1', '0', 1, 8]
Scaled Input: [[ 0.49743719  0.04522156  0.04522156 -0.40699407 -0.859209
7  -0.40699407
  -0.40699407 -0.8592097  -0.40699407  2.75851535]]
Approve Loan
Input: [2, 3, '2', '1', '0', '1', '1', '0', 1, 2]
Scaled Input: [[ 0.77777778  1.88888889  0.77777778 -0.33333333 -1.444444
44 -0.33333333
  -0.33333333 -1.44444444 -0.33333333  0.77777778]]
```