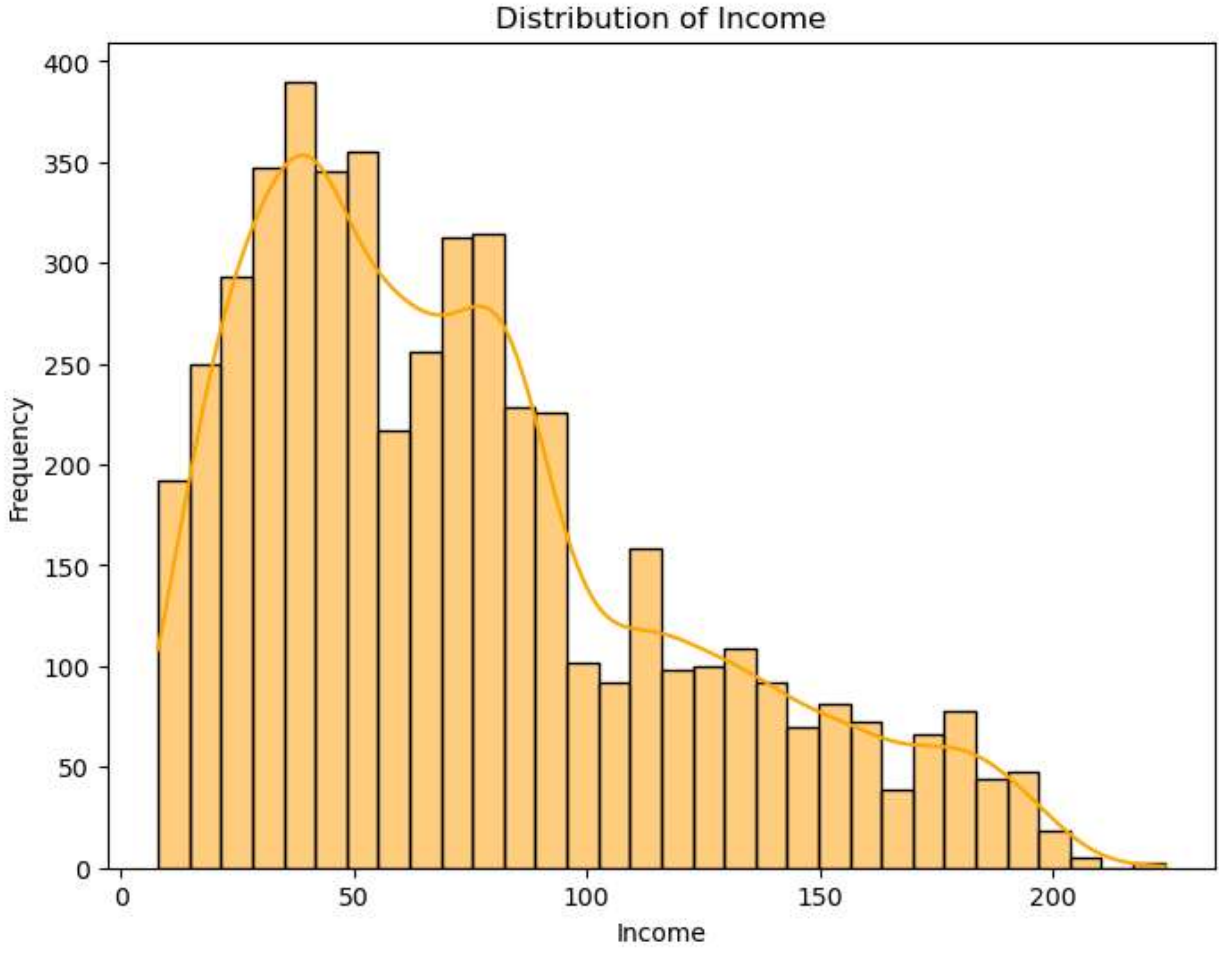
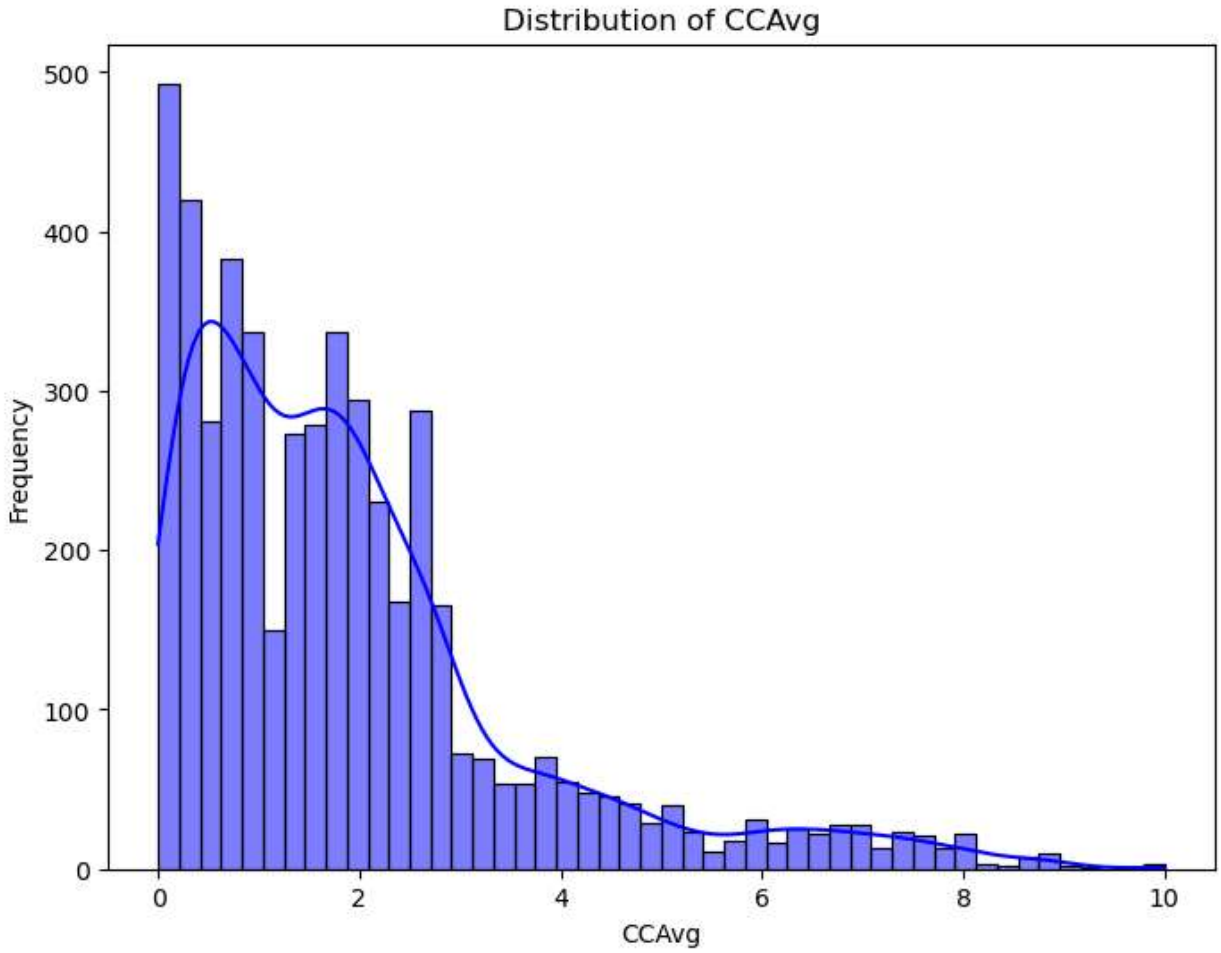
In [3]:
```
# Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv("Bank_Personal_Loan_Modelling.csv")

# Display the first few rows of the dataset
print(data.head())
```

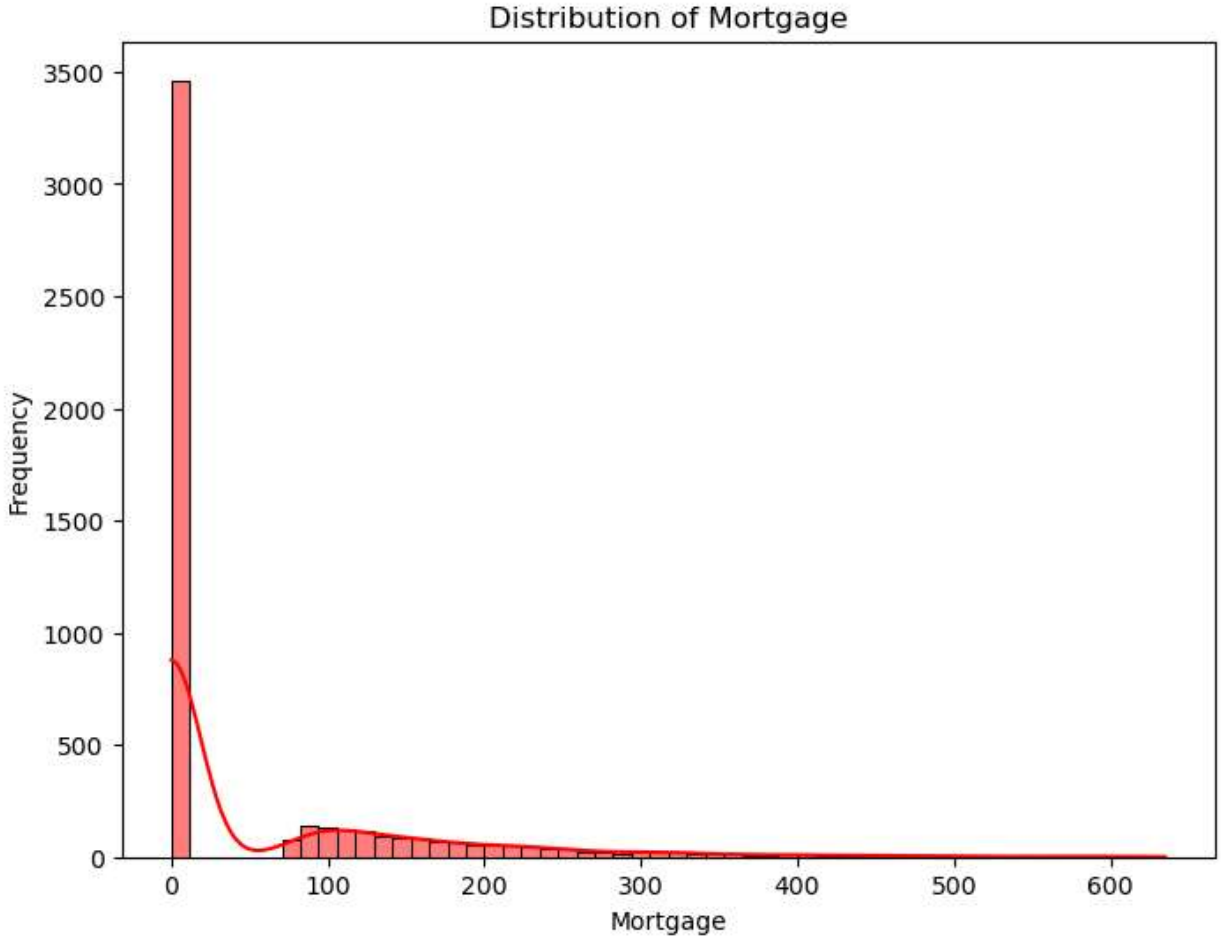
```
   ID  Age  Experience  Income  ZIP Code  Family  CCAvg  Education  Mortgage  \
0   1   25           1      49     91107       4    1.6          1         0
1   2   45          19      34     90089       3    1.5          1         0
2   3   39          15      11     94720       1    1.0       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

   Personal Loan  Securities Account  CD Account  Online  CreditCard
0              0                   1           0       0           0
1              0                   1           0       0           0
2              0                   0           0       0           0
3              0                   0           0       0           0
4              0                   0           0       0           1
```

In [4]:
```
# Visualize the distribution of Age
plt.figure(figsize=(8, 6))
sns.histplot(data['Age'], kde=True, color='green')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

# Visualize the distribution of Income
plt.figure(figsize=(8, 6))
sns.histplot(data['Income'], kde=True, color='orange')
plt.title('Distribution of Income')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.show()

# Visualize the distribution of CCAvg
plt.figure(figsize=(8, 6))
sns.histplot(data['CCAvg'], kde=True, color='blue')
plt.title('Distribution of CCAvg')
plt.xlabel('CCAvg')
plt.ylabel('Frequency')
plt.show()

# Visualize the distribution of Mortgage
plt.figure(figsize=(8, 6))
sns.histplot(data['Mortgage'], kde=True, color='red')
plt.title('Distribution of Mortgage')
plt.xlabel('Mortgage')
plt.ylabel('Frequency')
plt.show()
```

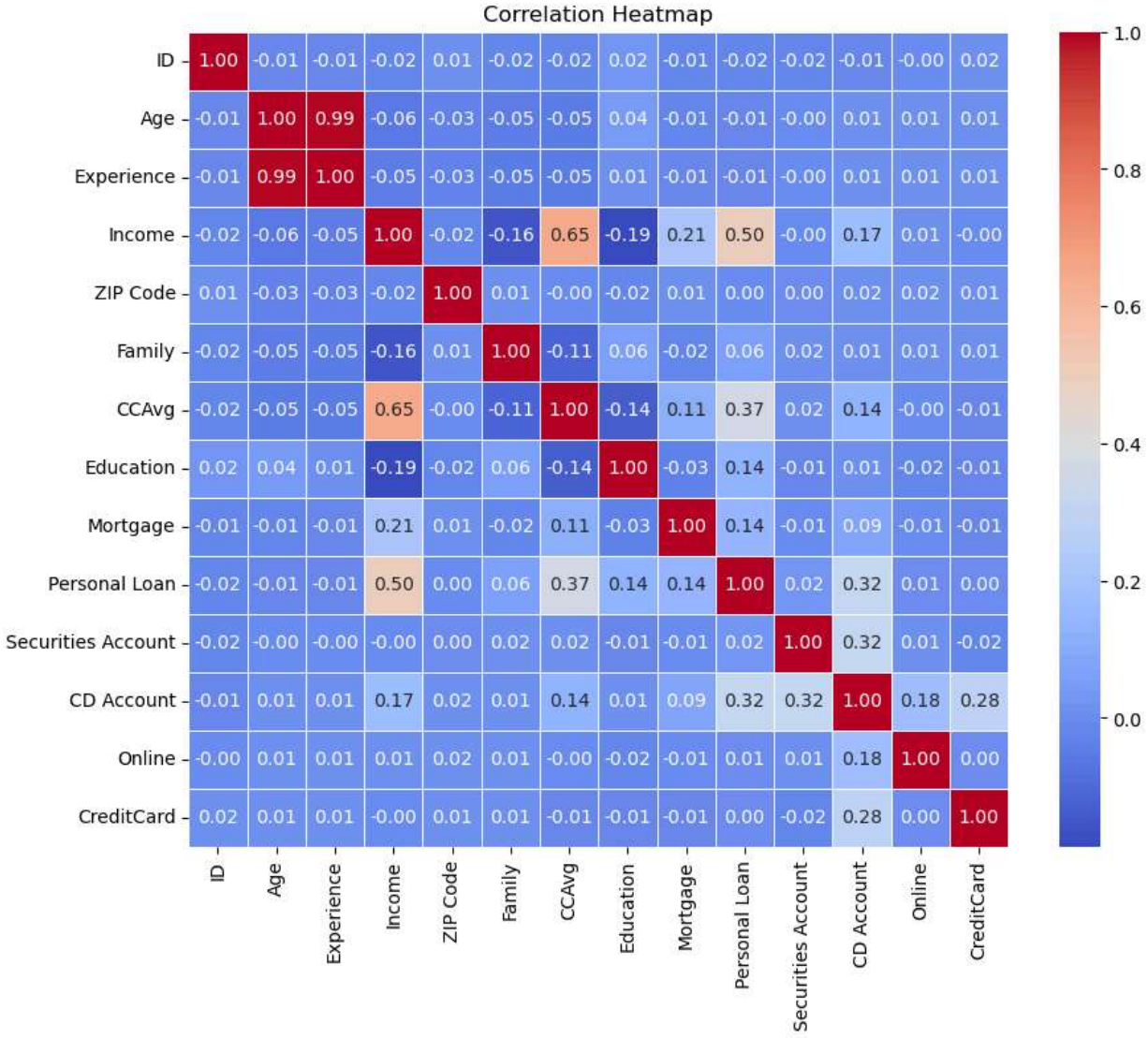

Distribution of Age
Frequency
Age
0
50
100
150
200
250
300
350
400
30
40
50
60


Distribution of Income
Frequency
Income
0
50
100
150
200
250
300
350
400
0
50
100
150
200


Distribution of CCAvg
Frequency
CCAvg
0
100
200
300
400
500
0
2
4
6
8
10


Distribution of Mortgage
Frequency
Mortgage
0
500
1000
1500
2000
2500
3000
3500
0
100
200
300
400
500
600

In [5]:
```
# Calculate the correlation matrix
corr_matrix = data.corr()

# Plot the correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



In [6]:
```
from sklearn.model_selection import train_test_split

# Drop irrelevant columns and split features (X) and target variable (y)
X = data.drop(columns=['Personal Loan'])
y = data['Personal Loan']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Print the shapes of the training and testing sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (4000, 13) (4000,)
Testing set shape: (1000, 13) (1000,)
```

In [7]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize models
logistic_model = LogisticRegression(random_state=42)
decision_tree_model = DecisionTreeClassifier(random_state=42)
random_forest_model = RandomForestClassifier(random_state=42)

# Train models
logistic_model.fit(X_train, y_train)
decision_tree_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)

# Predictions
y_pred_logistic = logistic_model.predict(X_test)
y_pred_dt = decision_tree_model.predict(X_test)
y_pred_rf = random_forest_model.predict(X_test)

# Evaluation
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

print("Logistic Regression Accuracy:", accuracy_logistic)
print("Decision Tree Accuracy:", accuracy_dt)
print("Random Forest Accuracy:", accuracy_rf)

# Print classification report for logistic regression
print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, y_pred_logistic))
```

```
Logistic Regression Accuracy: 0.908
Decision Tree Accuracy: 0.985
Random Forest Accuracy: 0.99

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       895
           1       0.63      0.30      0.41       105

    accuracy                           0.91      1000
   macro avg       0.78      0.64      0.68      1000
weighted avg       0.89      0.91      0.89      1000
```

In [9]:
```
import warnings
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Suppress convergence warnings
warnings.filterwarnings("ignore")

# Initialize logistic regression model
logistic_model = LogisticRegression(random_state=42)

# Initialize RFE
```

```
rfe = RFE(estimator=logistic_model, n_features_to_select=5)

# Fit RFE on training data
rfe.fit(X_train, y_train)

# Get selected features
selected_features = X_train.columns[rfe.support_]

print("Selected Features:", selected_features)
```

```
Selected Features: Index(['Income', 'Family', 'CCAvg', 'Education', 'CD Account'], dt
ype='object')
```

In [10]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize models
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42)
}

# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy:", accuracy)
    print(f"{name} Classification Report:")
    print(classification_report(y_test, y_pred))
```

```
Logistic Regression Accuracy: 0.908
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       895
           1       0.63      0.30      0.41       105

    accuracy                           0.91      1000
   macro avg       0.78      0.64      0.68      1000
weighted avg       0.89      0.91      0.89      1000

Decision Tree Accuracy: 0.985
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       895
           1       0.94      0.91      0.93       105

    accuracy                           0.98      1000
   macro avg       0.97      0.95      0.96      1000
weighted avg       0.98      0.98      0.98      1000

Random Forest Accuracy: 0.99
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       895
           1       0.98      0.92      0.95       105

    accuracy                           0.99      1000
   macro avg       0.99      0.96      0.97      1000
weighted avg       0.99      0.99      0.99      1000
```

In [12]:

```python
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization parameter for logistic regr
    'solver': ['liblinear', 'lbfgs'],  # Solver for logistic regression
}

# Initialize the model for hyperparameter tuning
logistic_model = LogisticRegression(random_state=42)

# Perform grid search cross-validation to find the best hyperparameters
grid_search = GridSearchCV(logistic_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Use the best hyperparameters to train the final model
final_model = LogisticRegression(**best_params, random_state=42)
final_model.fit(X_train, y_train)

# Evaluate the final model
y_pred_final = final_model.predict(X_test)
accuracy_final = accuracy_score(y_test, y_pred_final)
print("Final Model Accuracy:", accuracy_final)
```

```
print("Final Model Classification Report:")
print(classification_report(y_test, y_pred_final))
```

```
Best Hyperparameters: {'C': 1, 'solver': 'lbfgs'}
Final Model Accuracy: 0.908
Final Model Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       895
           1       0.63      0.30      0.41       105

    accuracy                           0.91      1000
   macro avg       0.78      0.64      0.68      1000
weighted avg       0.89      0.91      0.89      1000
```

In [13]:
```
from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix for the final model
cm_final = confusion_matrix(y_test, y_pred_final)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_final, annot=True, fmt="d", cmap="Oranges", cbar=False)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix (Final Model)")
plt.show()
```