DSC 680

Inman, Gracie

Project 1

03/31/24

In [1]:
```
# load data
import pandas as pd
credit_risk_df = pd.read_csv("credit_risk_dataset.csv")
credit_risk_df.head()
```

Out[1]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rat
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.0
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.1
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.8
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.2
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.2

In [2]:
```
# check columns
credit_risk_df.columns
```

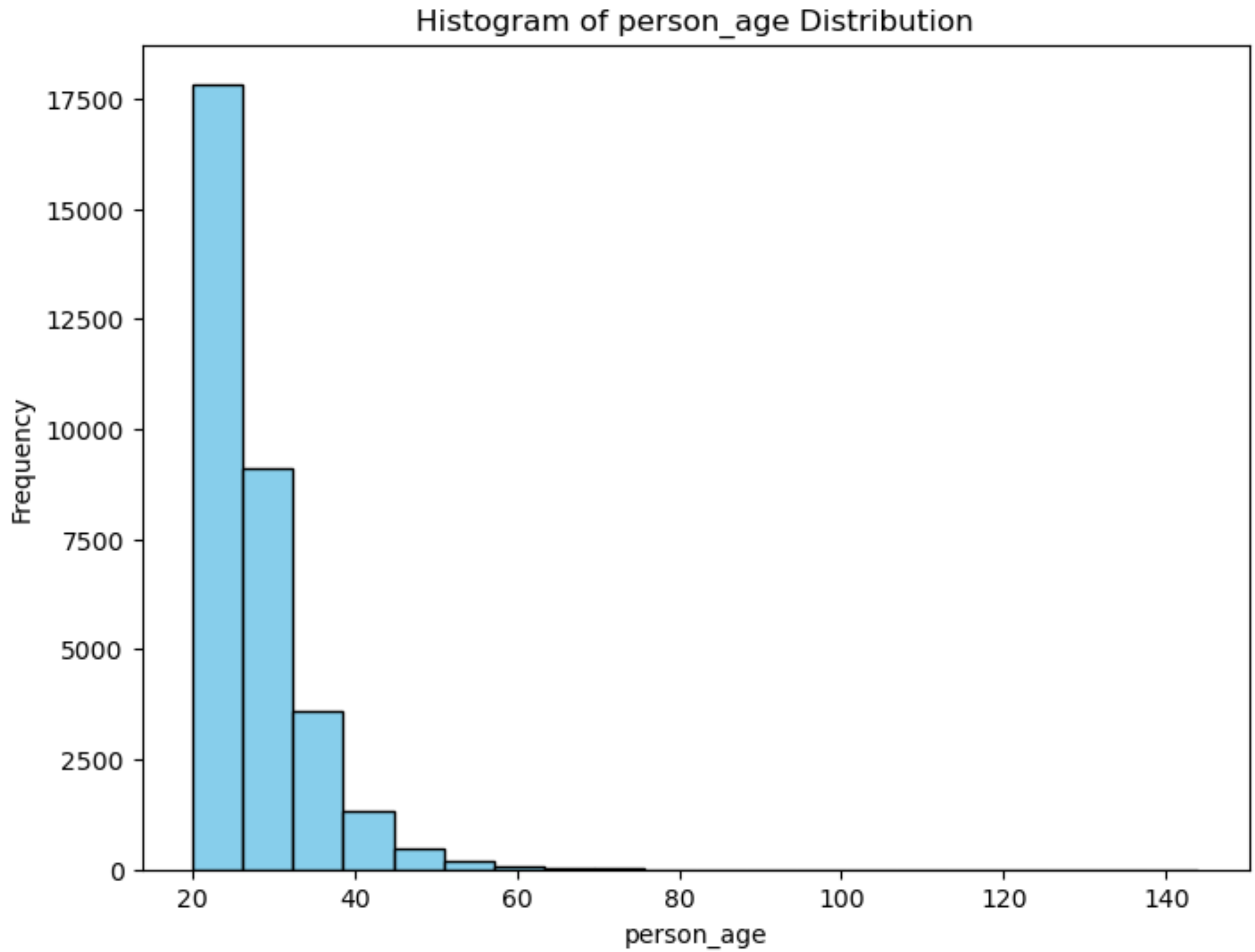
Out[2]:
```
Index(['person_age', 'person_income', 'person_home_ownership',
       'person_emp_length', 'loan_intent', 'loan_grade', 'loan_amnt',
       'loan_int_rate', 'loan_status', 'loan_percent_income',
       'cb_person_default_on_file', 'cb_person_cred_hist_length'],
      dtype='object')
```

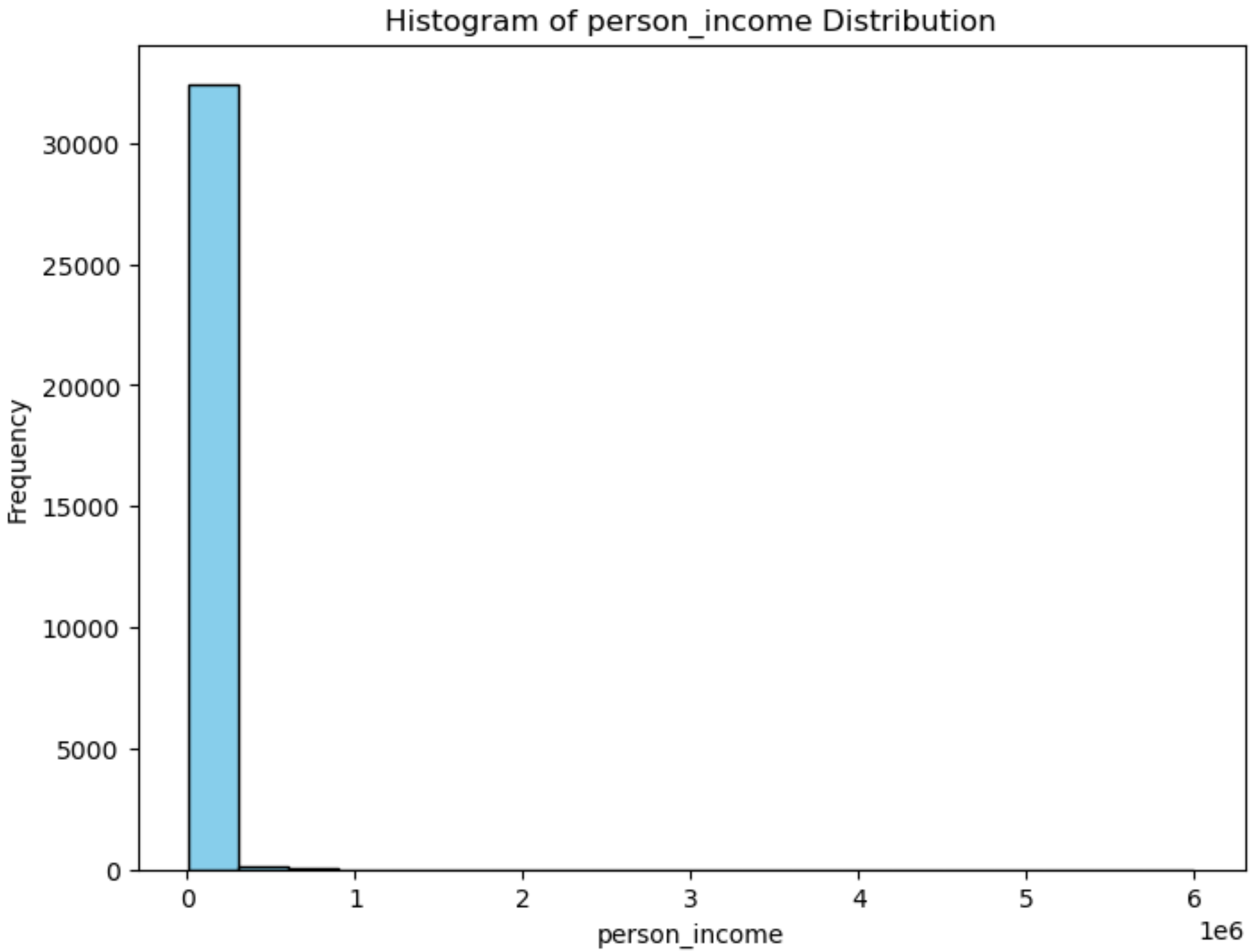
In [3]:
```
# drop unneed columns
credit_risk = credit_risk_df.drop(["loan_grade", "loan_int_rate"], axis = 1)
credit_risk.head()
```

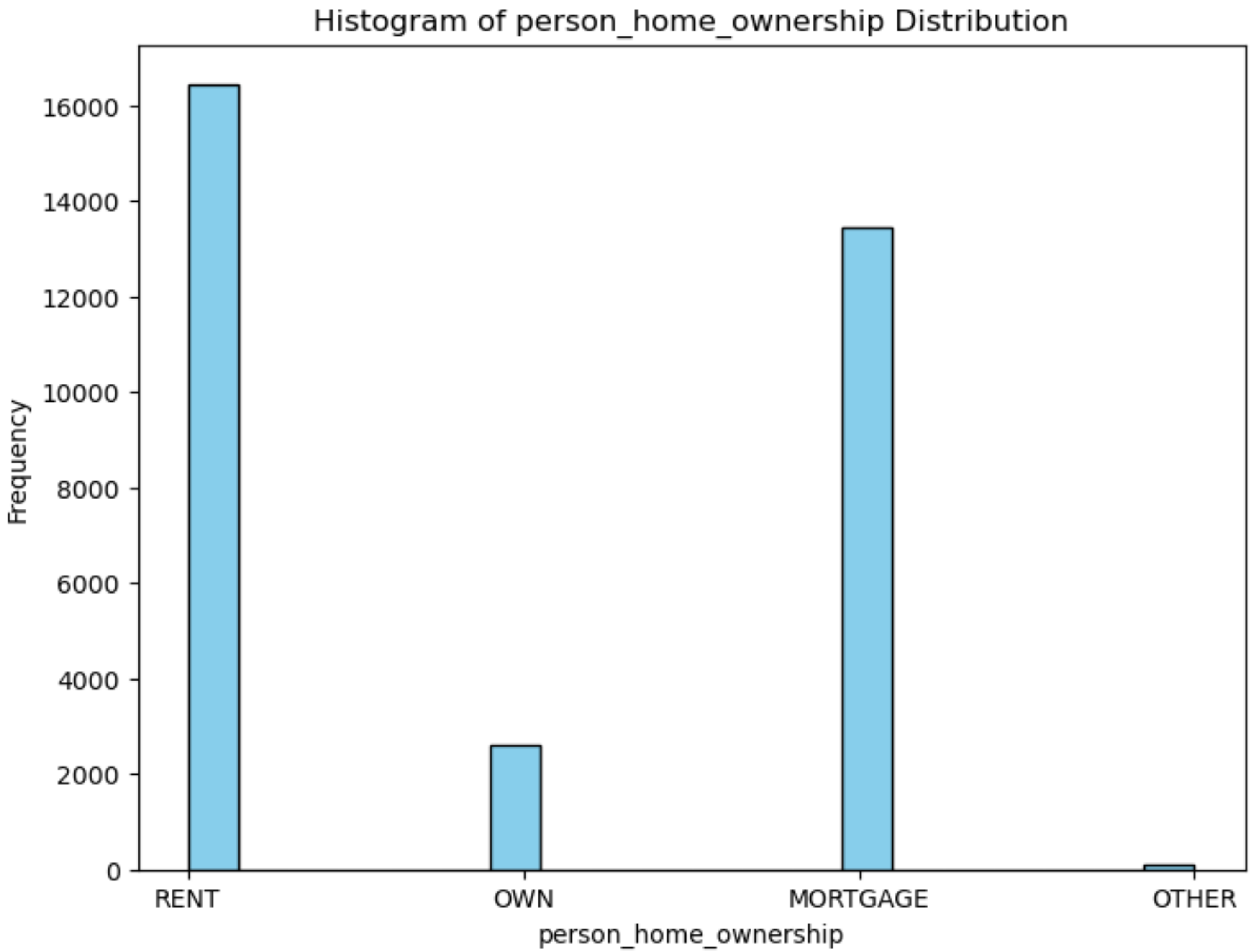
Out[3]:

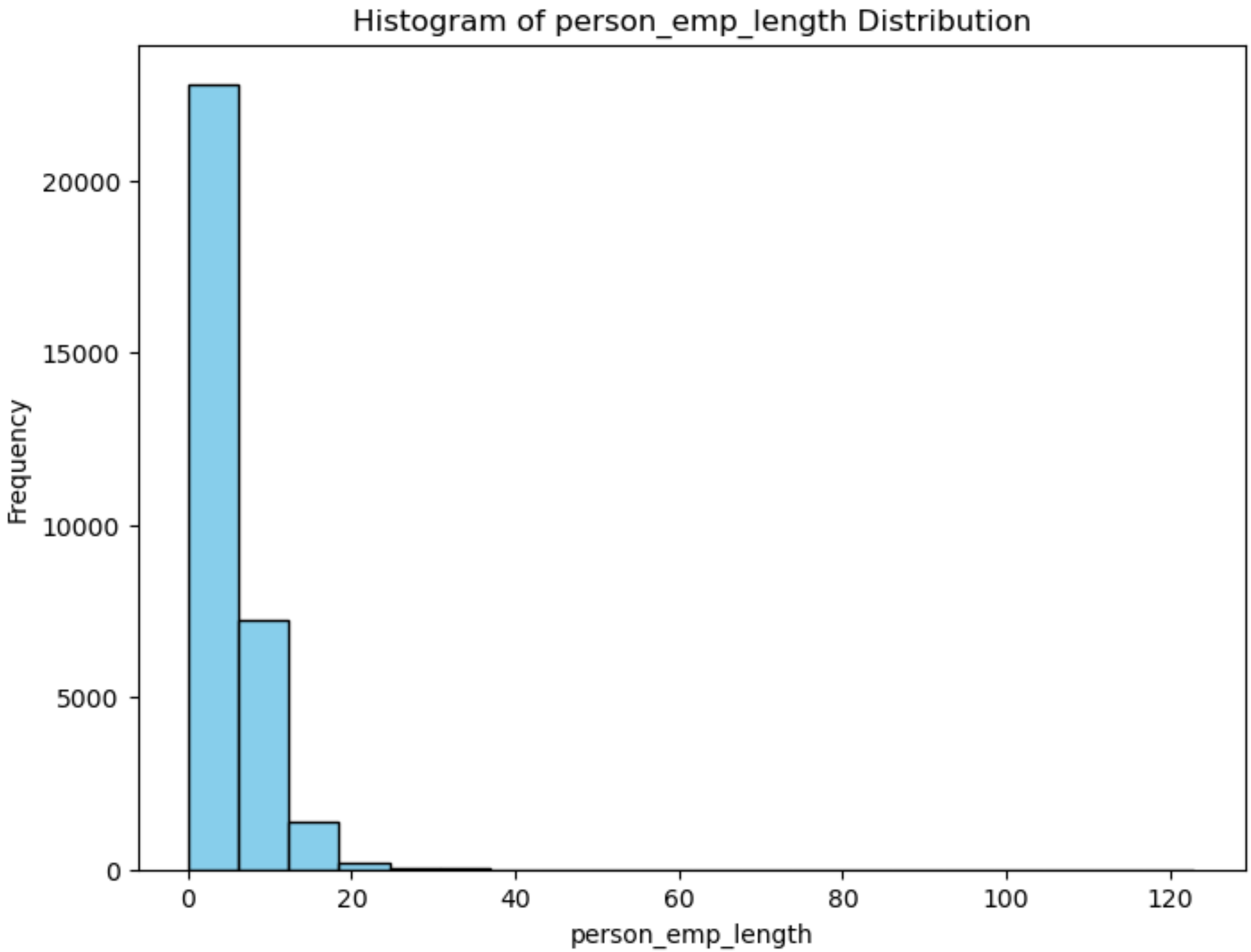
	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_amnt	loan_status	loan_percer
0	22	59000	RENT	123.0	PERSONAL	35000	1	
1	21	9600	OWN	5.0	EDUCATION	1000	0	
2	25	9600	MORTGAGE	1.0	MEDICAL	5500	1	
3	23	65500	RENT	4.0	MEDICAL	35000	1	
4	24	54400	RENT	8.0	MEDICAL	35000	1	

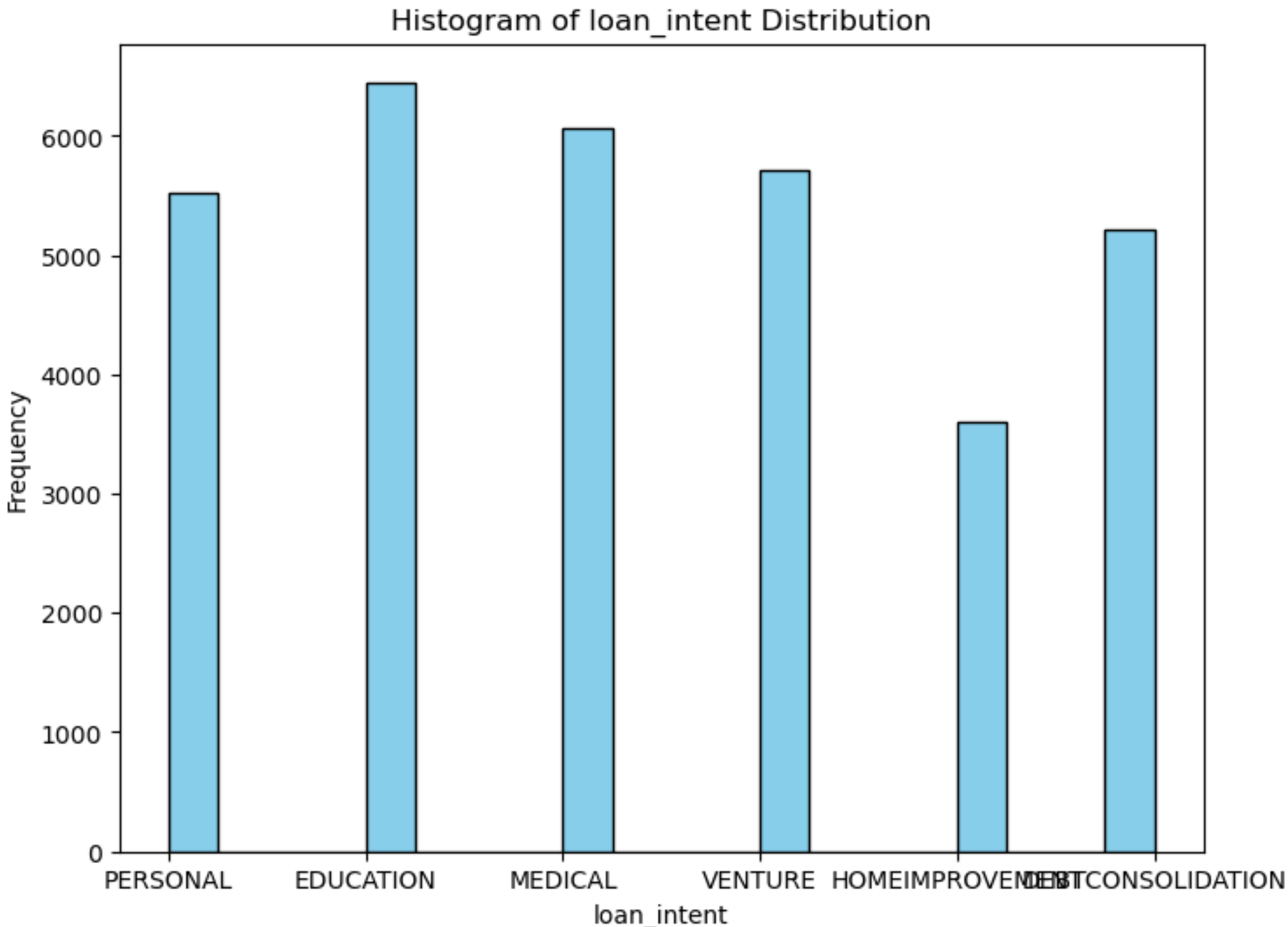
In [4]:
```
# visualize columns
import matplotlib.pyplot as plt
import pandas as pd
for column in credit_risk.columns:
    plt.figure(figsize=(8, 6))
    plt.hist(credit_risk[column], bins=20, color='skyblue', edgecolor='black')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.title(f'Histogram of {column} Distribution')
    plt.show()
```

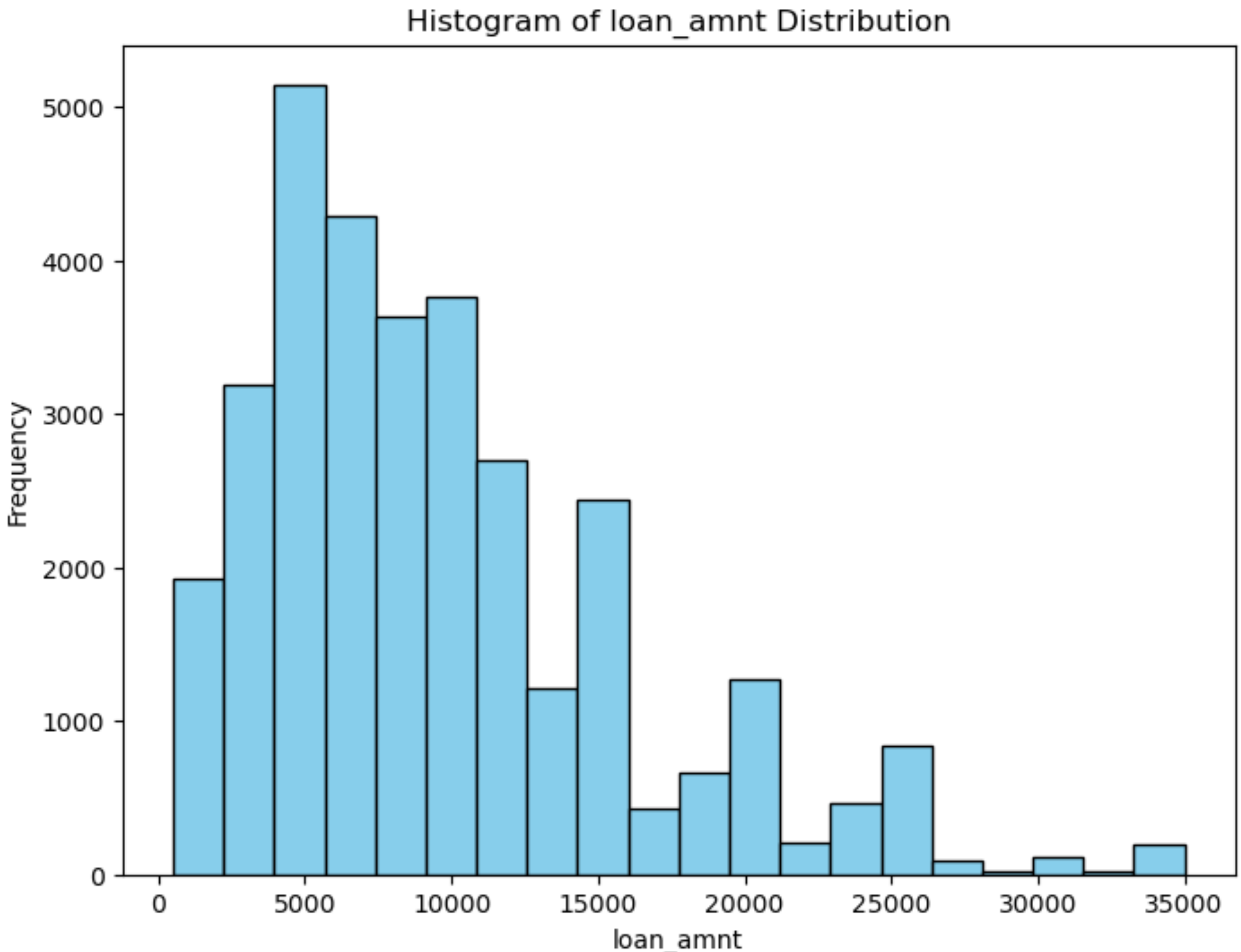

Histogram of person_age Distribution
Frequency
17500
15000
12500
10000
7500
5000
2500
0
20
40
60
80
100
120
140
person_age

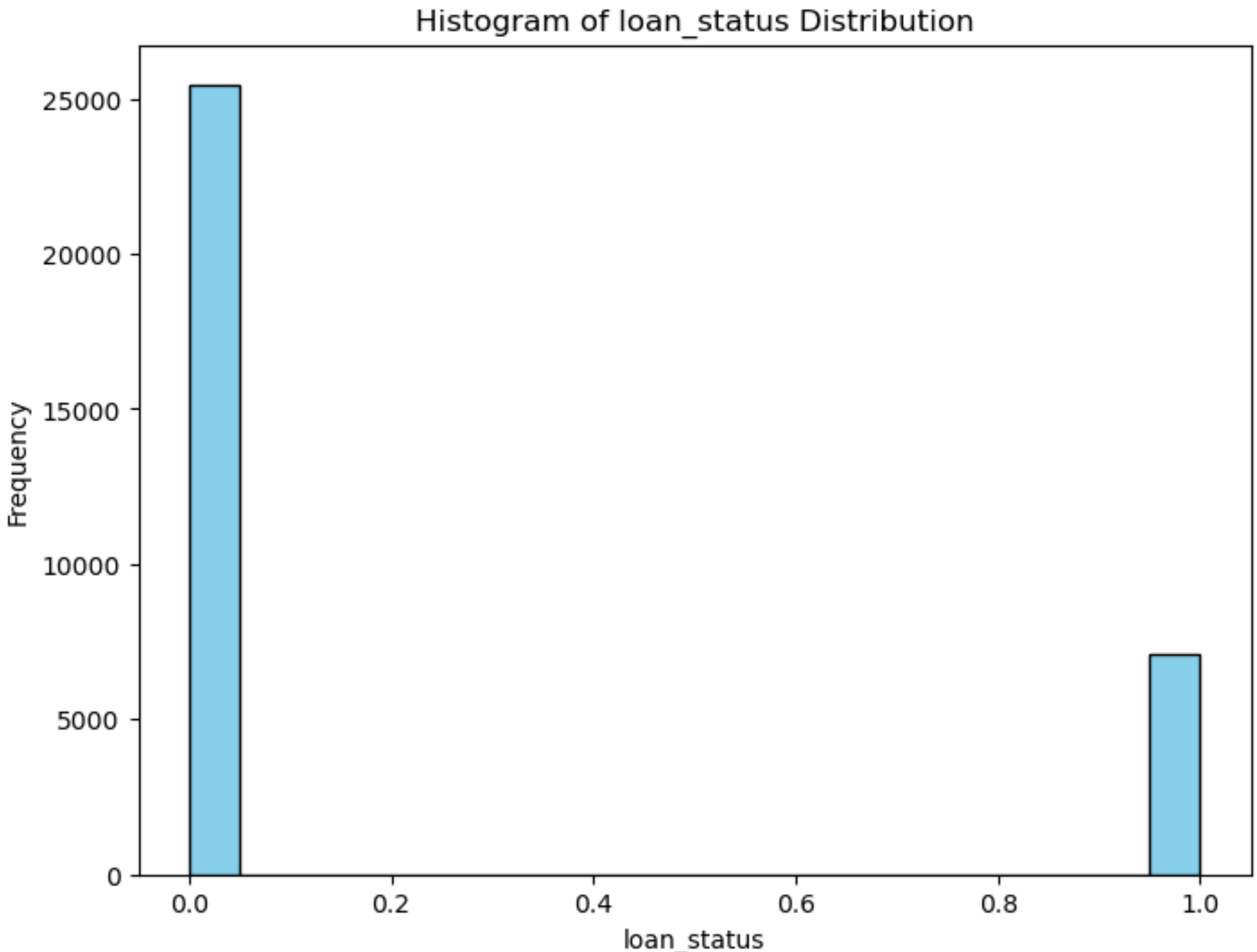

Histogram of person_income Distribution
Frequency
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
1e6
person_income

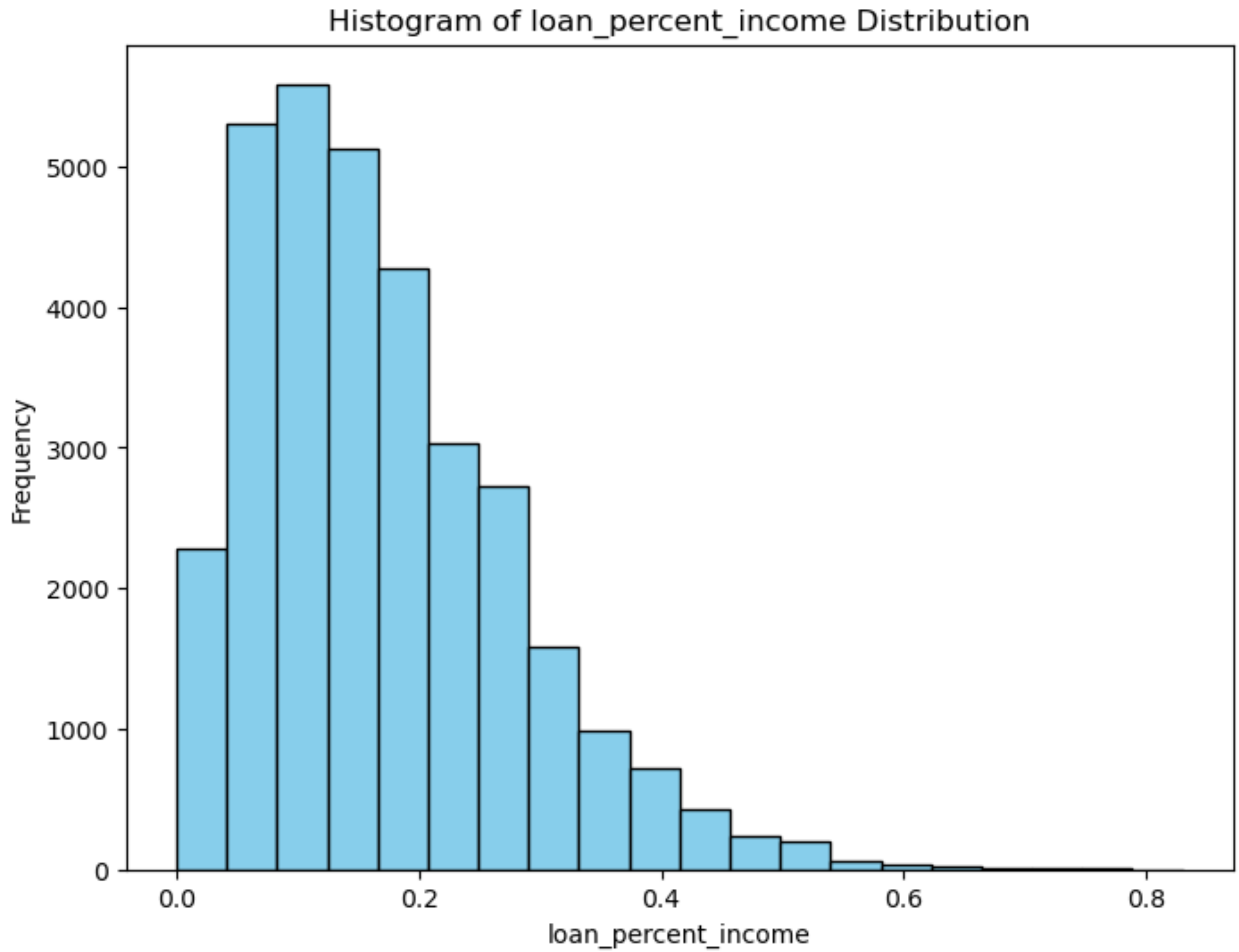

Histogram of person_home_ownership Distribution
Frequency
16000
14000
12000
10000
8000
6000
4000
2000
0
RENT
OWN
MORTGAGE
OTHER
person_home_ownership

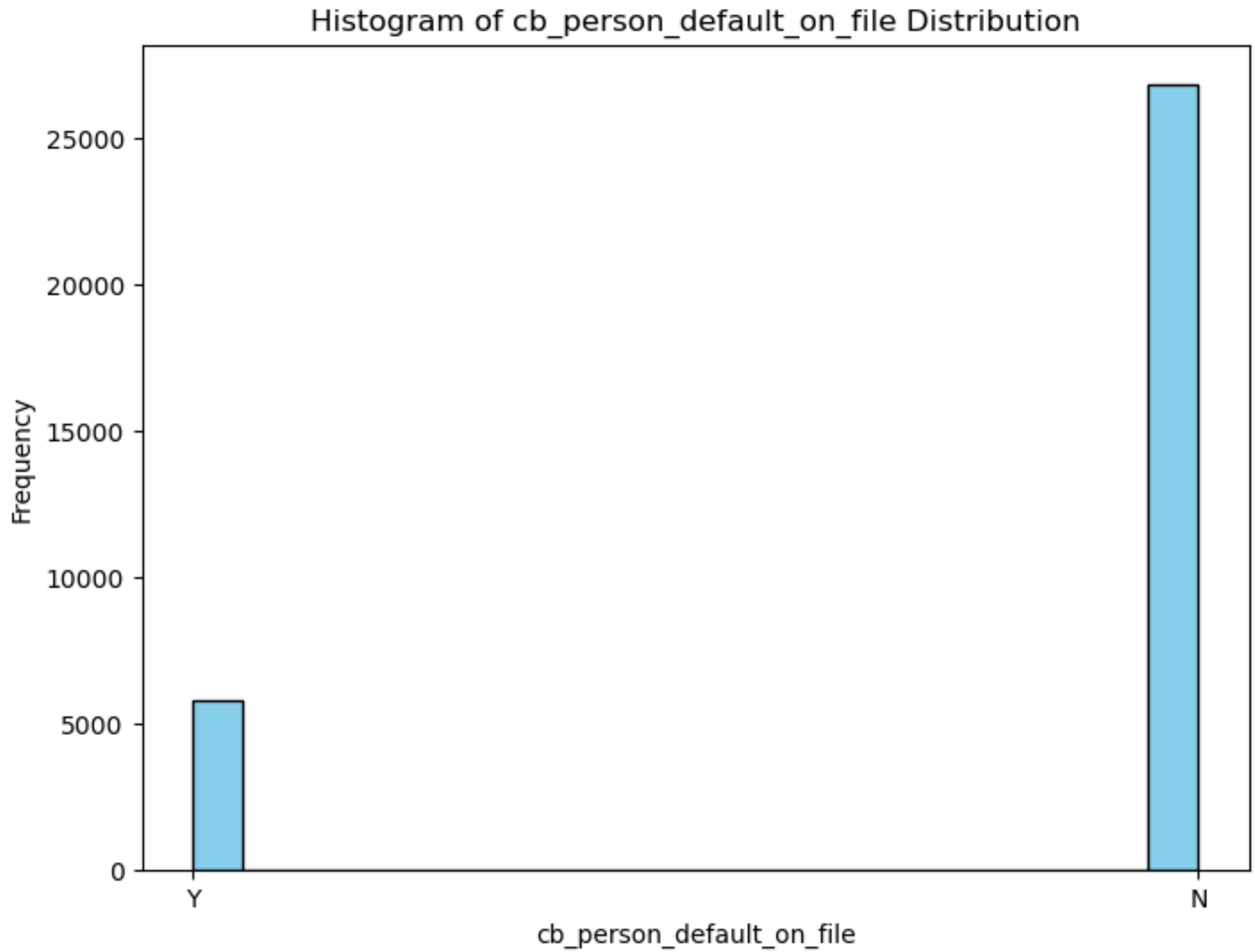

Histogram of person_emp_length Distribution
Frequency
20000
15000
10000
5000
0
0
20
40
60
80
100
120
person_emp_length

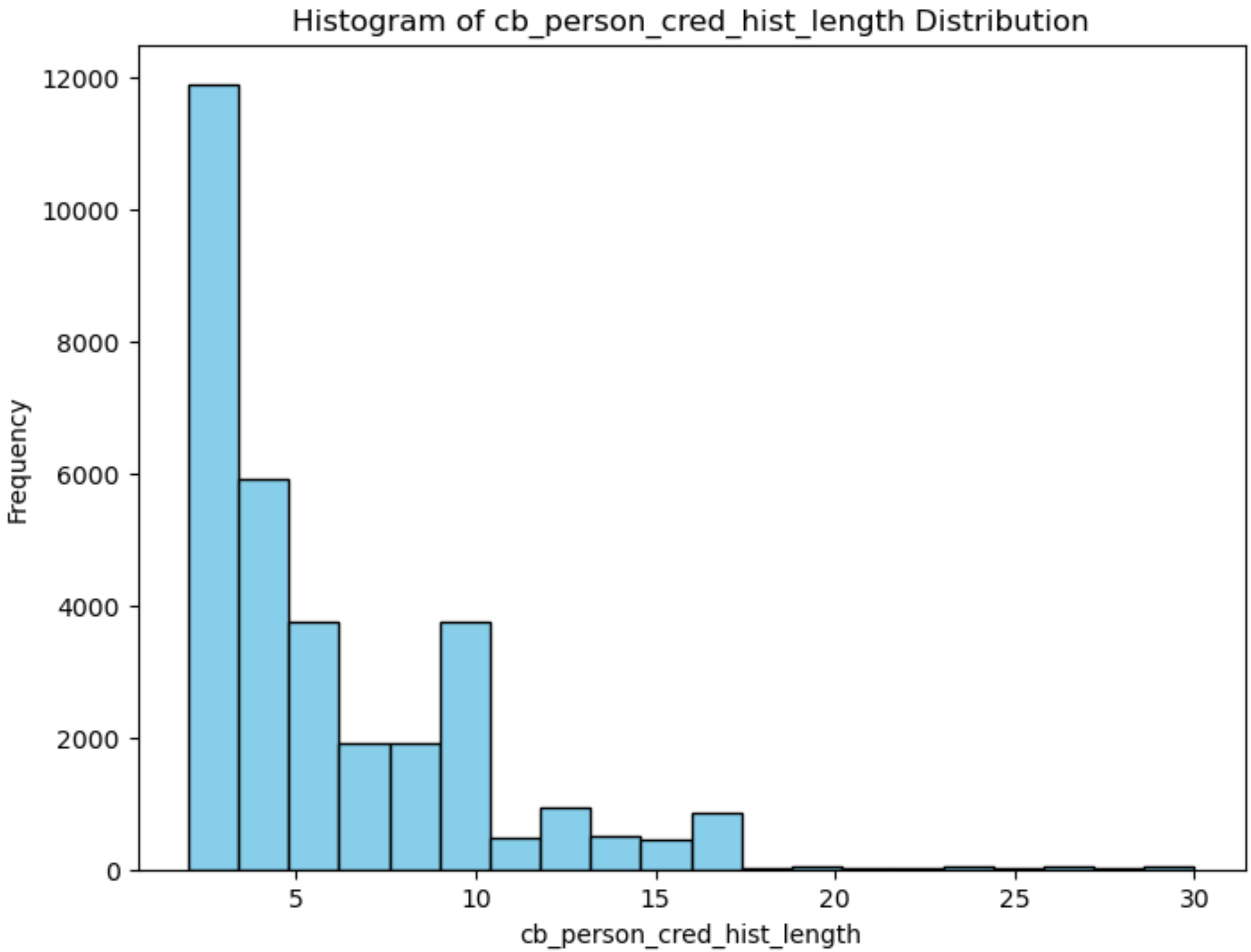

Histogram of loan_intent Distribution
Frequency
6000
5000
4000
3000
2000
1000
0
PERSONAL
EDUCATION
MEDICAL
VENTURE
HOMEIMPROVEMENT
DEBTCONSOLIDATION
loan_intent


Histogram of loan_amnt Distribution
Frequency
loan_amnt
0
1000
2000
3000
4000
5000
0
5000
10000
15000
20000
25000
30000
35000


Histogram of loan_status Distribution
Frequency
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
loan_status


Histogram of loan_percent_income Distribution
Frequency
0
1000
2000
3000
4000
5000
0.0
0.2
0.4
0.6
0.8
loan_percent_income


Histogram of cb_person_default_on_file Distribution
Frequency
25000
20000
15000
10000
5000
0
Y
N
cb_person_default_on_file


Histogram of cb_person_cred_hist_length Distribution
Frequency
0
2000
4000
6000
8000
10000
12000
5
10
15
20
25
30
cb_person_cred_hist_length

In [5]:
```
# Check columns values
for column in credit_risk.columns:
    unique_values = credit_risk[column].unique()
    print(f"Unique values for '{column}':")
    print(unique_values)
    print()
```

```
Unique values for 'person_age':
[ 22  21  25  23  24  26 144 123  20  32  34  29  33  28  35  31  27  30
  36  40  50  45  37  39  44  43  41  46  38  47  42  48  49  58  65  51
  53  66  61  54  57  59  62  60  55  52  64  70  78  69  56  73  63  94
  80  84  76  67]

Unique values for 'person_income':
[  59000    9600   65500 ...  720000 1900000    4888]

Unique values for 'person_home_ownership':
['RENT' 'OWN' 'MORTGAGE' 'OTHER']

Unique values for 'person_emp_length':
[123.   5.   1.   4.   8.   2.   6.   7.   0.   9.   3.  10.  nan  11.
  18.  12.  17.  14.  16.  13.  19.  15.  20.  22.  21.  24.  23.  26.
  25.  27.  28.  31.  41.  34.  29.  38.  30.]

Unique values for 'loan_intent':
['PERSONAL' 'EDUCATION' 'MEDICAL' 'VENTURE' 'HOMEIMPROVEMENT'
 'DEBTCONSOLIDATION']

Unique values for 'loan_amnt':
[35000  1000  5500  2500  1600  4500 30000  1750 34800 34000  1500 33950
 33000  4575  1400 32500  4000  2000 32000 31050 24250  7800 20000 10000
 25000 18000 12000 29100 28000  9600  3000  6100  4200  4750  4800  2700
 27600  3250 27500 27050 27000 26000 25600 25475 21600 11900 25300  3650
  6000  2400  3600  7500  4950 21000 16000 22000  7750 24000 15000 15500
  9000 23050  5375  6250  5000  2100 14000  6200  9950  4475  2600  8000
  4600  3500  7200  8800  3175  2800 13000  1800  3300  3200  2275  5600
  3625  4375 24750 24500  3900 13750 15250 24150  2250  4975  4900 23975
 23750 23600 23575  5400  3375 23400 23000  1200 22750 22500 22400 22250
  7400 21700  7000 21500 21450 21250  9250 20900 20500 20400 20375 20050
```

```
  6400  5650 16600  7125  3550  1275  3800  1625  8500  7575  5200  4025
  4400  3825  6500  5875  1550  7350  6700  8300 10625 19900 19800   500
   700   750 19000 18950 18800 18750 18725 18550  7100 18500 18400 18250
  1300  5800 18225 18200  1375 17950 17800 17750 17700  3975 17625 17600
 17500 17475 17400 17200 17000 16950 16875 16800 16750 16700 16525 16500
 11500 16425 16400 16300 16250 16075 16050 11100  1525 10800  7850 11325
 14500  5975  1075  1100  1150  3025  9475  1325  2750  1350  3725  1925
  5175  6300  8400  1450  9800  1475 14125 12300  9500  5225 12200 10750
  1675  1700  8875  5150  1775  6075  1825  1850  1875  1900  1950 11000
 14950 14900  7600 14850 14800 13250  5125  2050  2125  2150  3075  2200
  2225  2300  2350  6600  6950  2425  2450 13600 13500  9200 13475 13450
 13400 13350 13300 13275 13225 13200 13100 13050 12250  7550 11200  1050
 11225  8250 11050  2850  2875  2900  2925  2975 12500 10150  8325  1250
 12375  8125  6425  9750 14400  5100 10950  6800  9450 22550  6900  8575
  3050  3100  3125  3150  7775 13650  2950 12800   800  3325  3350  3400
  3450  5775  8700 11625 11300  5250  7275 14775  5300  6725  3525  3575
 15800 14600  6350 10900 10875 12400 10775 10700 10600 10500 10450 10400
 10375 10325 10300 10250 10200  3700  3750  3850  3950  5550  7675  5700
  5325  9875  4350  4450  4300 10850  8100  4550  4650  4700  4725 13025
  2525 15450  6625 17050  7975  9700  8200  4850 19200 13975  8675  9350
  9975  9100  9900 14750  7050  5750 15075 12600 15600 22800  6650 13800
  8475 18900 14300  8975  8950  8900  8850  8650 14550  4150  9050  4075
 14650  8450  9125  4325  5950  9925  7375 11700  9225 10075  5275 23500
  8600  5425  5450 12725 13850  5525  5575  5625  5675  5825  5850  5900
  5925  2550 15750 19500 10525 18650 13700  9825  9175  7075  7025 11400
  8375  6025  6150 15825  6225 15200 14100  2650  6975  6325  6375 19750
  2625  6550  6575  5025  6850  6750  6775  6475  6450  6825  6875  6925
  8525  3775 24200 11075  7150  7175  4225  7875 21825  7250  7300 19125
  7325  7475 17300  9575 12875 11425 19725   900 17450 14075 12275 31300
  7525 15700 11600 14825  7650  7700  7900  7925  7950 13375 25850 21200
 23275 10425 15850  6125  5075  5050 12900  9525 29800 21650  8050  8075
 23525  8150  8350 27250  2475  8550  8625  8725  8750  8775  7425  9150
  9300  9325  9375  9400  9425  9550 29000 12150 19600 26400 15900  4275
  4250 13950  7450  4125  4100  4050 11875 18300 31825 11125 16100 29700
  6675 15350 10675 10025 10100 10125  3425 14200 11250 17825 11525 11550
 11575 11650 11750 11775 11800 11850 11975 25975 14625  8825 27525 19075
 14700 18600  2825  4925 21400  1125 20675 16200 12475 18150 12100 12125
 13675 12450  2775  2725  2675  4175 12950 12700 12750 24175 10925 13625
 13900 25200 12975 14350  3275 14275 20600 23800 29175 21850  9850 14525
```

```
 14575 27300 12075 17325  9625 19950  4525 22600 19400 20800 15125 12225
 15400 18325 15550 15625 15650 15675 23450 10575 19425 19550 19650  2375
  2325 31000 30750 29550 28800 14725 22200 24625 23850 23475 22950 22650
 21725 20200  2075  5725 19975 19850 19775   725   950 18825 17975 17900
 17725 17250 16775 16450 14975 13575 13425 13150 13075 10225  3925  5350
 10725 10550 10275  3675 12775  9775  1425 14675  4625  4425  4675 12650
  4875  1225 17350  9275 10825 10175 15150  5475 17375 10650  8275 15025
  9725  6525 15875 10475 27575 10975 12625  8025  7225 10050  4775 20475
  7725  8225 23200 16725 21100 22100  7625 25500 20150 18050 23700 19700
 15275 11175 11350 11450 11475 19150 19450  9075 21125 24800 24400 12325
 12350  2575 12025 14150 17875 11025 14475 26375 13125 27400 14050 28250
 15975 33250  6275 22350 24100 15050 17525 15175 23100 11275 13175 19925
 32400 30600 31400 27175 24375  8175 23325 18125  3225 26800 17925 14250
 12925 13775 17850 20700 11375 15575 15775 19275 29850]

Unique values for 'loan_status':
[1 0]

Unique values for 'loan_percent_income':
[0.59 0.1  0.57 0.53 0.55 0.25 0.45 0.44 0.42 0.16 0.41 0.37 0.32 0.3
 0.06 0.29 0.31 0.22 0.52 0.14 0.49 0.13 0.5  0.35 0.17 0.27 0.33 0.08
 0.03 0.21 0.63 0.47 0.4  0.07 0.38 0.34 0.04 0.23 0.15 0.11 0.43 0.51
 0.02 0.28 0.26 0.19 0.39 0.09 0.05 0.61 0.18 0.6  0.01 0.48 0.12 0.54
 0.56 0.46 0.36 0.24 0.2  0.72 0.64 0.69 0.77 0.83 0.65 0.67 0.58 0.71
 0.68 0.7  0.66 0.   0.76 0.62 0.78]

Unique values for 'cb_person_default_on_file':
['Y' 'N']

Unique values for 'cb_person_cred_hist_length':
[ 3  2  4  8  7  6  9 10  5 11 16 15 12 13 17 14 25 28 27 22 19 29 23 26
 20 21 30 24 18]
```

In [6]:
```
credit_risk.shape
```

Out[6]:
```
(32581, 10)
```

In [7]:
```
# Check for missing values
missing_values = credit_risk.isnull().sum()
print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
person_age                      0
person_income                   0
person_home_ownership           0
person_emp_length             895
loan_intent                     0
loan_amnt                       0
loan_status                     0
loan_percent_income             0
cb_person_default_on_file       0
cb_person_cred_hist_length      0
dtype: int64
```

In [8]:
```
credit_risk = credit_risk.dropna()
print("Shape of cleaned dataset:", credit_risk.shape)
```

```
Shape of cleaned dataset: (31686, 10)
```

In [9]:
```
# Encode Categorical Values
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

credit_risk['person_home_ownership_encoded'] = label_encoder.fit_transform(credit_risk['person_home_owner

# Print mapping
print("Mapping of encoded labels:")
for label, encoded_label in zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)):
    print(f"{label}: {encoded_label}")

# Check data
print("\nFirst few rows of the dataset with encoded 'person_home_ownership':")
print(credit_risk[['person_home_ownership', 'person_home_ownership_encoded']].head())
```

```
Mapping of encoded labels:
MORTGAGE: 0
OTHER: 1
OWN: 2
RENT: 3

First few rows of the dataset with encoded 'person_home_ownership':
  person_home_ownership  person_home_ownership_encoded
0                  RENT                              3
1                   OWN                              2
2              MORTGAGE                              0
3                  RENT                              3
4                  RENT                              3
```

In [10]:

```
credit_risk['cb_person_default_on_file_encoded'] = label_encoder.fit_transform(credit_risk['cb_person_def

# Print mapping
print("Mapping of encoded labels:")
for label, encoded_label in zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)):
    print(f"{label}: {encoded_label}")

# Check
print("\nFirst few rows of the dataset with encoded 'cb_person_default_on_file':")
print(credit_risk[['cb_person_default_on_file', 'cb_person_default_on_file_encoded']].head())
```

```
Mapping of encoded labels:
N: 0
Y: 1

First few rows of the dataset with encoded 'cb_person_default_on_file':
  cb_person_default_on_file  cb_person_default_on_file_encoded
0                         Y                                  1
1                         N                                  0
2                         N                                  0
3                         N                                  0
4                         Y                                  1
```

In [11]:
```
credit_risk['loan_intent_encoded'] = label_encoder.fit_transform(credit_risk['loan_intent'])

# Print mapping
print("Mapping of encoded labels:")
for label, encoded_label in zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)):
    print(f"{label}: {encoded_label}")

# Check
print("\nFirst few rows of the dataset with encoded 'loan_intent':")
print(credit_risk[['loan_intent', 'loan_intent_encoded']].head())
```

```
Mapping of encoded labels:
DEBTCONSOLIDATION: 0
EDUCATION: 1
HOMEIMPROVEMENT: 2
MEDICAL: 3
PERSONAL: 4
VENTURE: 5

First few rows of the dataset with encoded 'loan_intent':
  loan_intent  loan_intent_encoded
0    PERSONAL                    4
1   EDUCATION                    1
2     MEDICAL                    3
3     MEDICAL                    3
4     MEDICAL                    3
```

In [12]:
```
credit_risk.head()
```

Out[12]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_amnt	loan_status	loan_percer
0	22	59000	RENT	123.0	PERSONAL	35000	1	
1	21	9600	OWN	5.0	EDUCATION	1000	0	
2	25	9600	MORTGAGE	1.0	MEDICAL	5500	1	
3	23	65500	RENT	4.0	MEDICAL	35000	1	
4	24	54400	RENT	8.0	MEDICAL	35000	1	

In [13]:

```
credit_risk_encoded = credit_risk.drop(['person_home_ownership', 'cb_person_default_on_file', 'loan_inter
credit_risk_encoded.head()
```

Out[13]:

	person_age	person_income	person_emp_length	loan_amnt	loan_status	loan_percent_income	cb_person_cred_hist_length
0	22	59000	123.0	35000	1	0.59	3
1	21	9600	5.0	1000	0	0.10	2
2	25	9600	1.0	5500	1	0.57	3
3	23	65500	4.0	35000	1	0.53	2
4	24	54400	8.0	35000	1	0.55	4

In [14]:

```
# split data
from sklearn.model_selection import train_test_split

X = credit_risk_encoded.drop('loan_status', axis=1)
y = credit_risk_encoded['loan_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Shapes of training and test
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (22180, 9)
Shape of X_test: (9506, 9)
Shape of y_train: (22180,)
Shape of y_test: (9506,)
```

In [15]:
```
# train models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

# Initialize
logistic_regression = LogisticRegression()
random_forest = RandomForestClassifier()
naive_bayes = GaussianNB()

# Train
logistic_regression.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
naive_bayes.fit(X_train, y_train)
```

Out[15]:
```
▾ GaussianNB
GaussianNB()
```

In [16]:
```
# evaluate
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix
# dictionary
results = {}

# Logistic regression
logistic_regression_predictions = logistic_regression.predict(X_test)
logistic_regression_accuracy = accuracy_score(y_test, logistic_regression_predictions)
logistic_regression_precision = precision_score(y_test, logistic_regression_predictions)
logistic_regression_confusion_matrix = confusion_matrix(y_test, logistic_regression_predictions)
results['Logistic Regression'] = {'Accuracy': logistic_regression_accuracy,
                                  'Precision': logistic_regression_precision,
                                  'Confusion Matrix': logistic_regression_confusion_matrix}
```

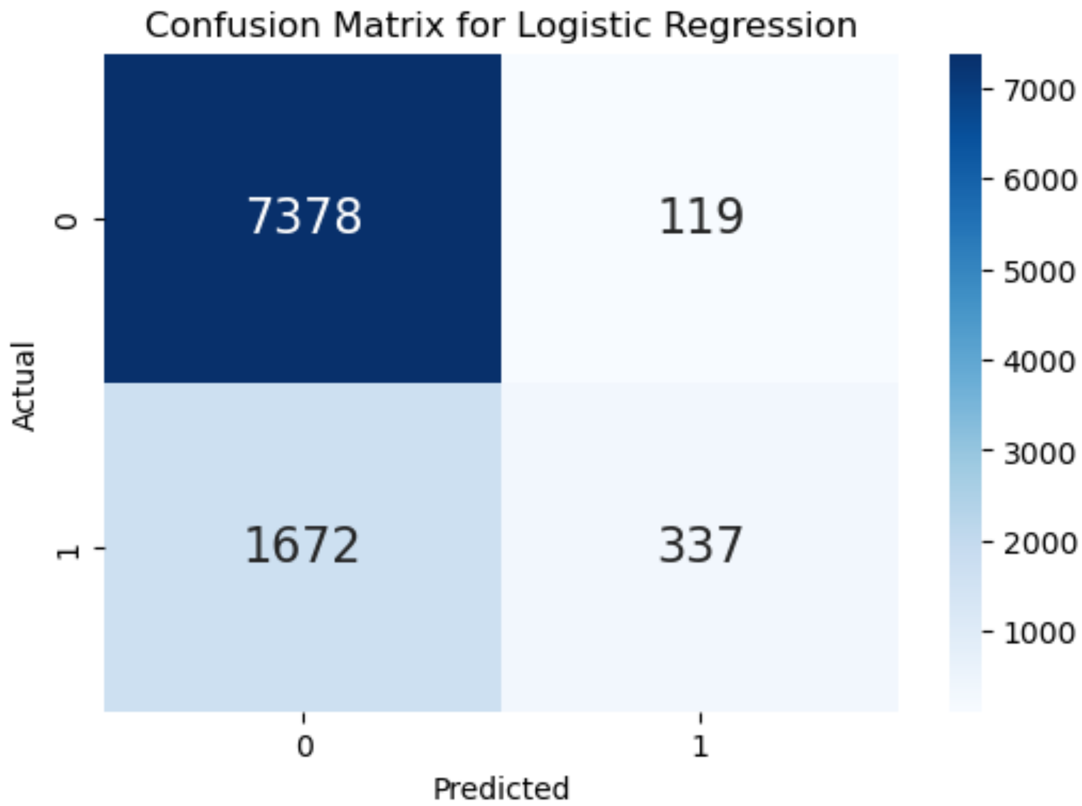
In [17]:
```
# random forest
random_forest_predictions = random_forest.predict(X_test)
random_forest_accuracy = accuracy_score(y_test, random_forest_predictions)
random_forest_precision = precision_score(y_test, random_forest_predictions)
random_forest_confusion_matrix = confusion_matrix(y_test, random_forest_predictions)
results['Random Forest'] = {'Accuracy': random_forest_accuracy,
                            'Precision': random_forest_precision,
                            'Confusion Matrix': random_forest_confusion_matrix}
```

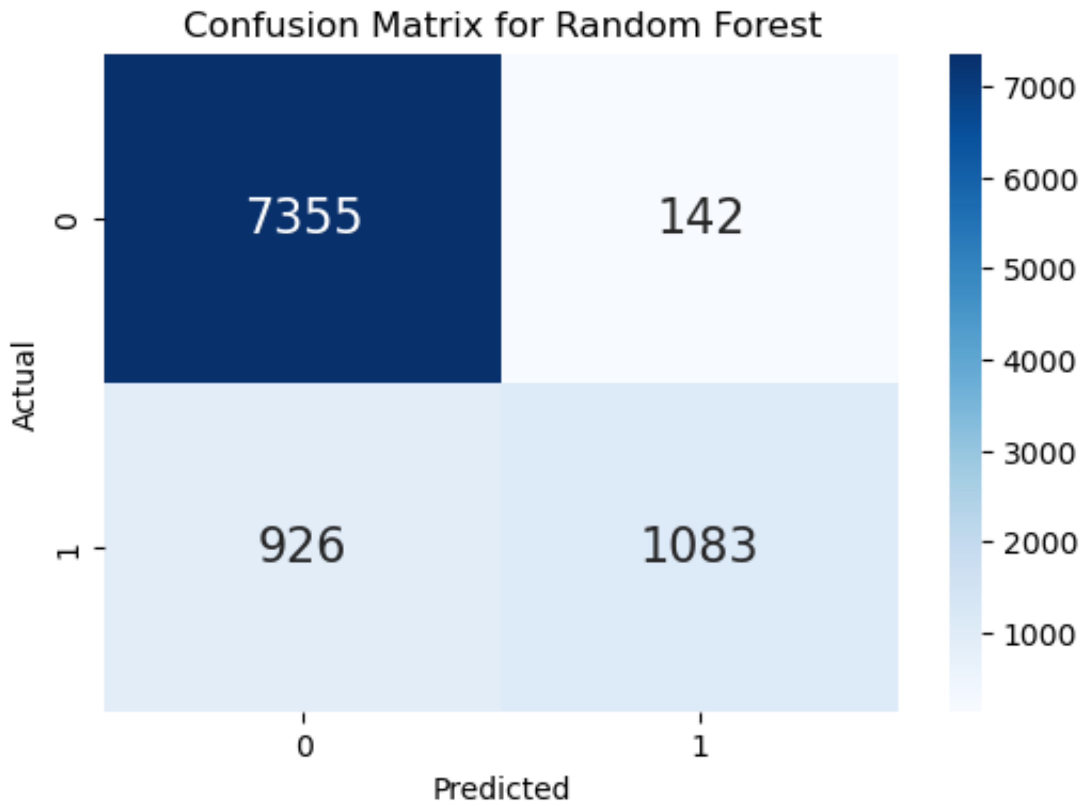
In [18]:
```
# Naive Bayes
naive_bayes_predictions = naive_bayes.predict(X_test)
naive_bayes_accuracy = accuracy_score(y_test, naive_bayes_predictions)
naive_bayes_precision = precision_score(y_test, naive_bayes_predictions)
naive_bayes_confusion_matrix = confusion_matrix(y_test, naive_bayes_predictions)
results['Naïve Bayes'] = {'Accuracy': naive_bayes_accuracy,
                          'Precision': naive_bayes_precision,
                          'Confusion Matrix': naive_bayes_confusion_matrix}
```

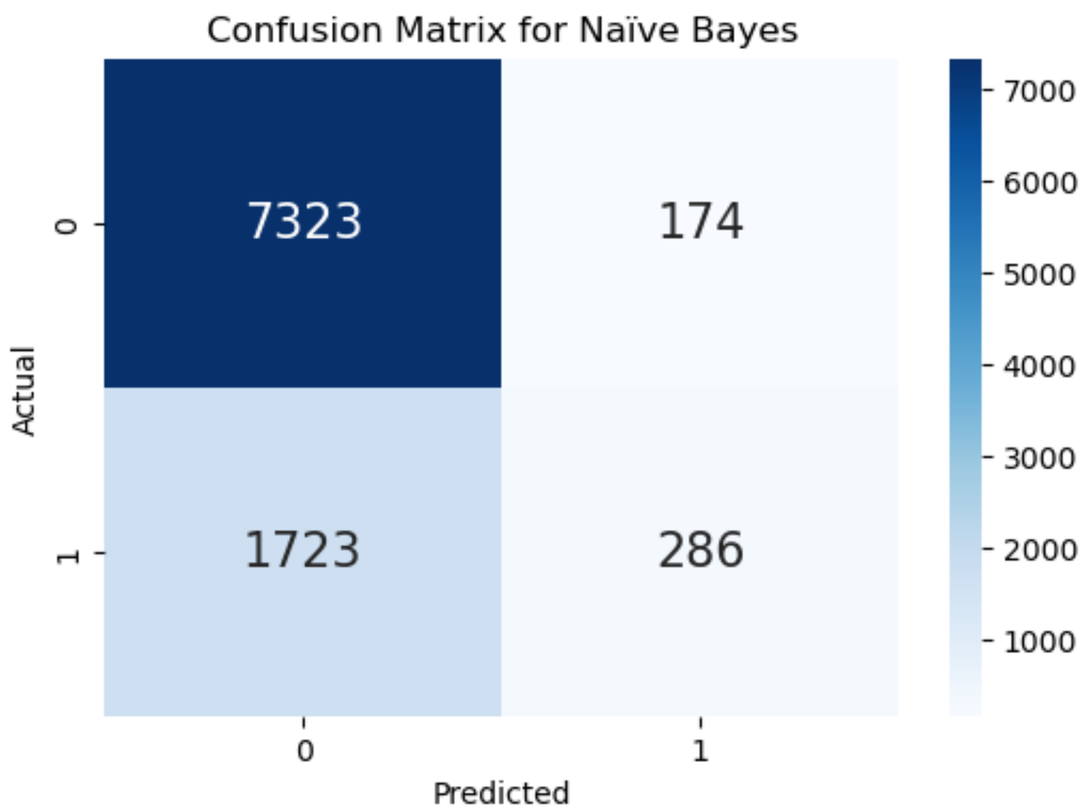
In [19]:
```
for model, metrics in results.items():
    print(f" {model}:")
    print(f"Accuracy: {metrics['Accuracy']:.4f}")
    print(f"Precision: {metrics['Precision']:.4f}")
    print("Confusion Matrix:")
    print(metrics['Confusion Matrix'])
    print()
```

```
 Logistic Regression:
Accuracy: 0.8116
Precision: 0.7390
Confusion Matrix:
[[7378  119]
 [1672  337]]

 Random Forest:
Accuracy: 0.8876
Precision: 0.8841
Confusion Matrix:
[[7355  142]
 [ 926 1083]]

 Naïve Bayes:
Accuracy: 0.8004
Precision: 0.6217
Confusion Matrix:
[[7323  174]
 [1723  286]]
```

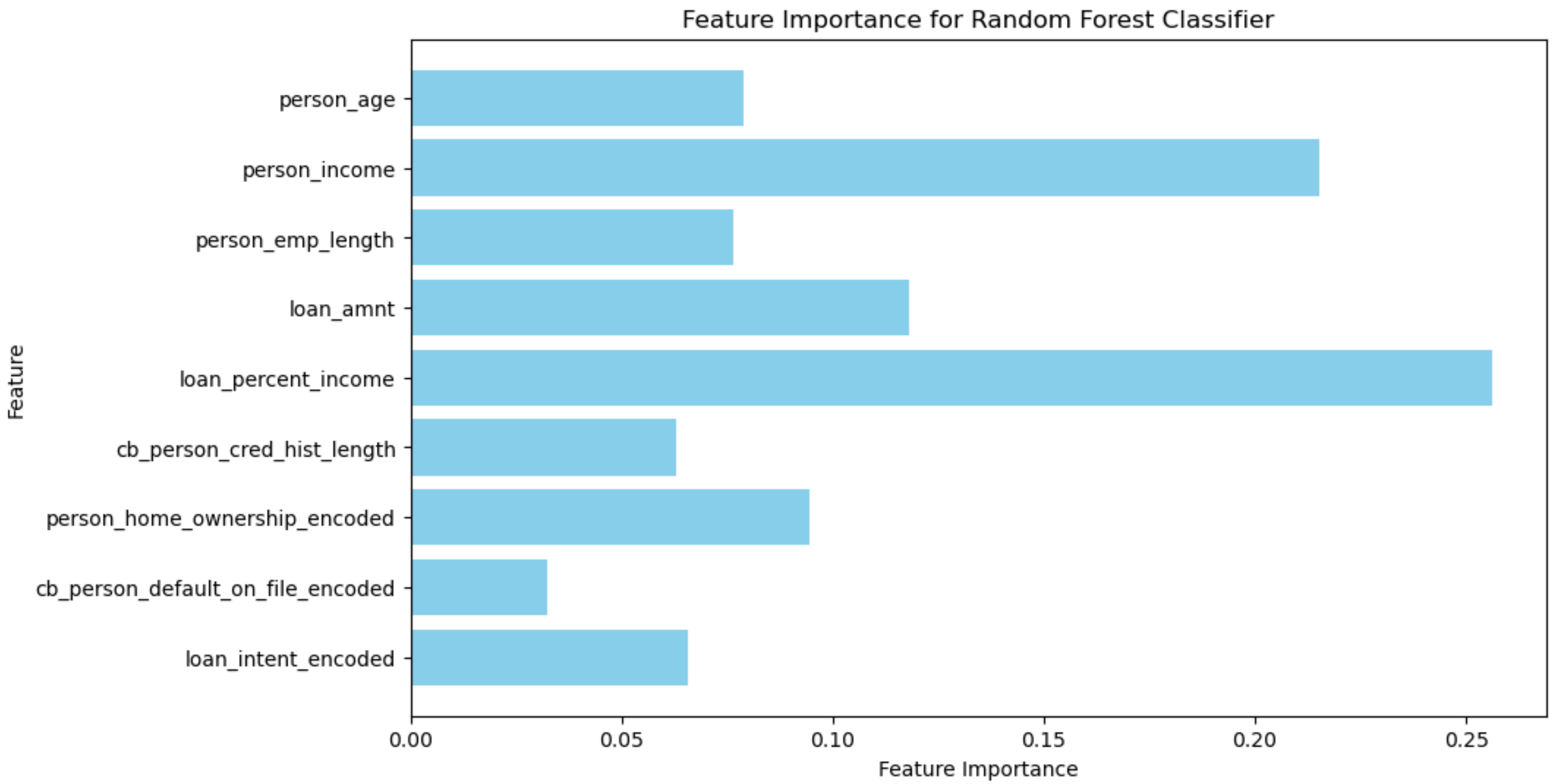
In [20]:
```
import seaborn as sns
for model, metrics in results.items():
    plt.figure(figsize=(6, 4))
    sns.heatmap(metrics['Confusion Matrix'], annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16})
    plt.title(f'Confusion Matrix for {model}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```


Confusion Matrix for Logistic Regression
7378
119
1672
337
Actual
Predicted
0
1
0
1
7000
6000
5000
4000
3000
2000
1000


Confusion Matrix for Random Forest
7355
142
926
1083
Actual
Predicted
0
1
0
1
7000
6000
5000
4000
3000
2000
1000



In [21]:
```
feature_importances = random_forest.feature_importances_

# Features
feature_names = X.columns

# Bar plot
plt.figure(figsize=(10, 6))
plt.barh(feature_names, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance for Random Forest Classifier')
plt.gca().invert_yaxis()
plt.show()
```


Feature Importance for Random Forest Classifier
person_age
person_income
person_emp_length
loan_amnt
loan_percent_income
cb_person_cred_hist_length
person_home_ownership_encoded
cb_person_default_on_file_encoded
loan_intent_encoded
Feature
0.00
0.05
0.10
0.15
0.20
0.25
Feature Importance

In [22]:
```
from sklearn.ensemble import AdaBoostClassifier

# AdaBoost
adaboost_random_forest = AdaBoostClassifier(base_estimator=random_forest)

# Train AdaBoost
adaboost_random_forest.fit(X_train, y_train)

# Evaluate
adaboost_random_forest_accuracy = adaboost_random_forest.score(X_test, y_test)
adaboost_random_forest_predictions = adaboost_random_forest.predict(X_test)
adaboost_random_forest_precision = precision_score(y_test, adaboost_random_forest_predictions)
adaboost_random_forest_confusion_matrix = confusion_matrix(y_test, adaboost_random_forest_predictions)

# Print
print("Metrics for AdaBoost with Random Forest:")
print(f"Accuracy: {adaboost_random_forest_accuracy:.4f}")
print(f"Precision: {adaboost_random_forest_precision:.4f}")
print("Confusion Matrix:")
print(adaboost_random_forest_confusion_matrix)
```

```
/Users/gracieinman/anaconda3/lib/python3.10/site-packages/sklearn/ensemble/_base.py:166: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
Metrics for AdaBoost with Random Forest:
Accuracy: 0.8772
Precision: 0.9033
Confusion Matrix:
[[7396  101]
 [1066  943]]
```

In [ ]: