Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

from scipy.stats import skew

import warnings
warnings.filterwarnings('ignore')
```

Load Data

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

Out[2]:

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

In [3]:
```
df.shape
```

Out[3]:
```
(5000, 14)
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

In [5]:
```
df.info()
```

```
<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
```

WE will drop two columns ID and Zip Code as it is not important for prediction

In [6]:
```
df.drop(['ID','ZIP Code'],axis=1,inplace=True)
```

In [7]:
```
df.describe()
```

Out[7]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000
mean	45.338400	20.104600	73.774200	2.396400	1.937938	1.881000	56.498800	0.096000	0.104400	0.06040
std	11.463166	11.467954	46.033729	1.147663	1.747659	0.839869	101.713802	0.294621	0.305809	0.23825
min	23.000000	-3.000000	8.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.00000
25%	35.000000	10.000000	39.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	0.00000
50%	45.000000	20.000000	64.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	0.00000
75%	55.000000	30.000000	98.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	0.00000
max	67.000000	43.000000	224.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	1.00000

Imbalance Data

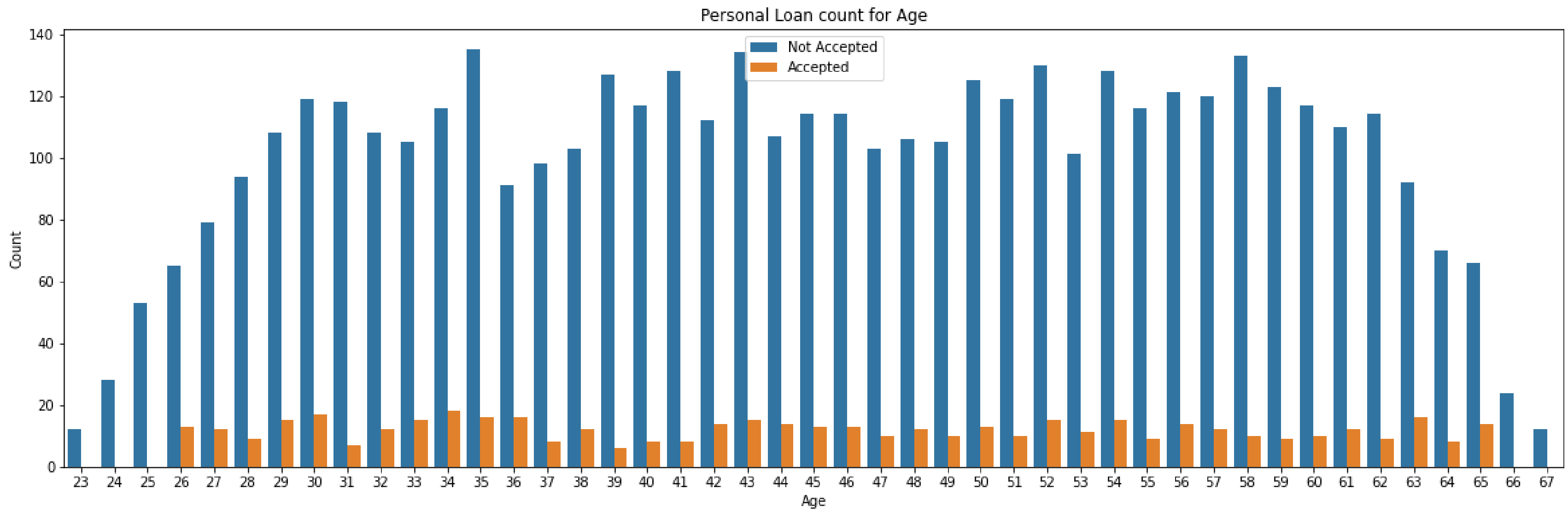
In [8]:
```
print("Accepted")
print(df[df["Personal Loan"] == 1]["Personal Loan"].count())
print("Not Accepted")
print(df[df["Personal Loan"] == 0]["Personal Loan"].count())
```

```
Accepted
480
Not Accepted
4520
```

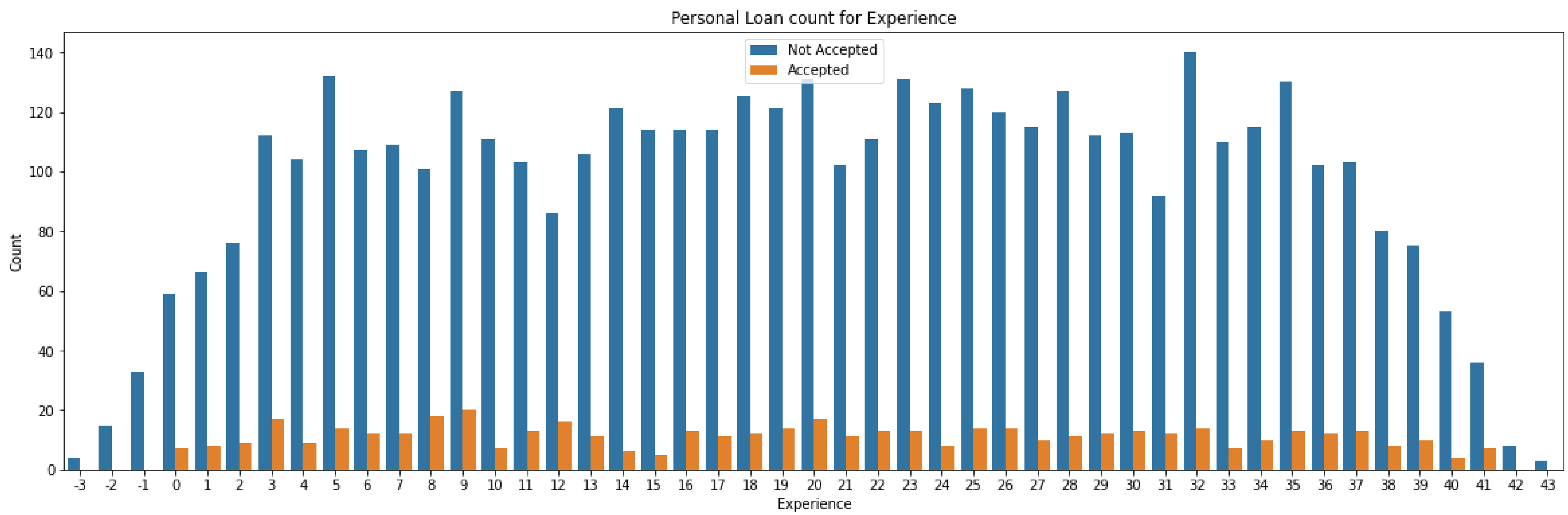
Imbalanced dataset not appears here cy llokin at sshape of dataset,so we perform data visualization is executed first to know potential relationship.

Data Visualization

In [9]:
```
plt.figure(figsize=(20,6))
sns.countplot(data=df,x='Age',hue='Personal Loan')
plt.title('Personal Loan count for Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```

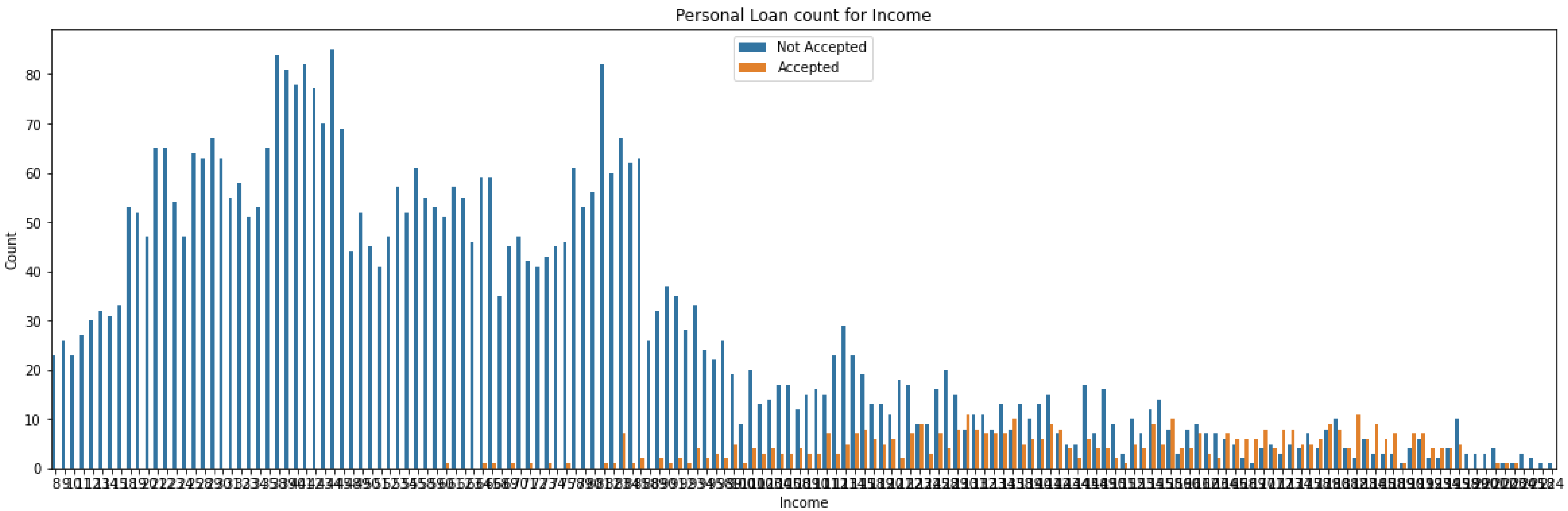


In [10]:
```
plt.figure(figsize=(20,6))
sns.countplot(data=df,x='Experience',hue='Personal Loan')
plt.title('Personal Loan count for Experience')
plt.xlabel('Experience')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



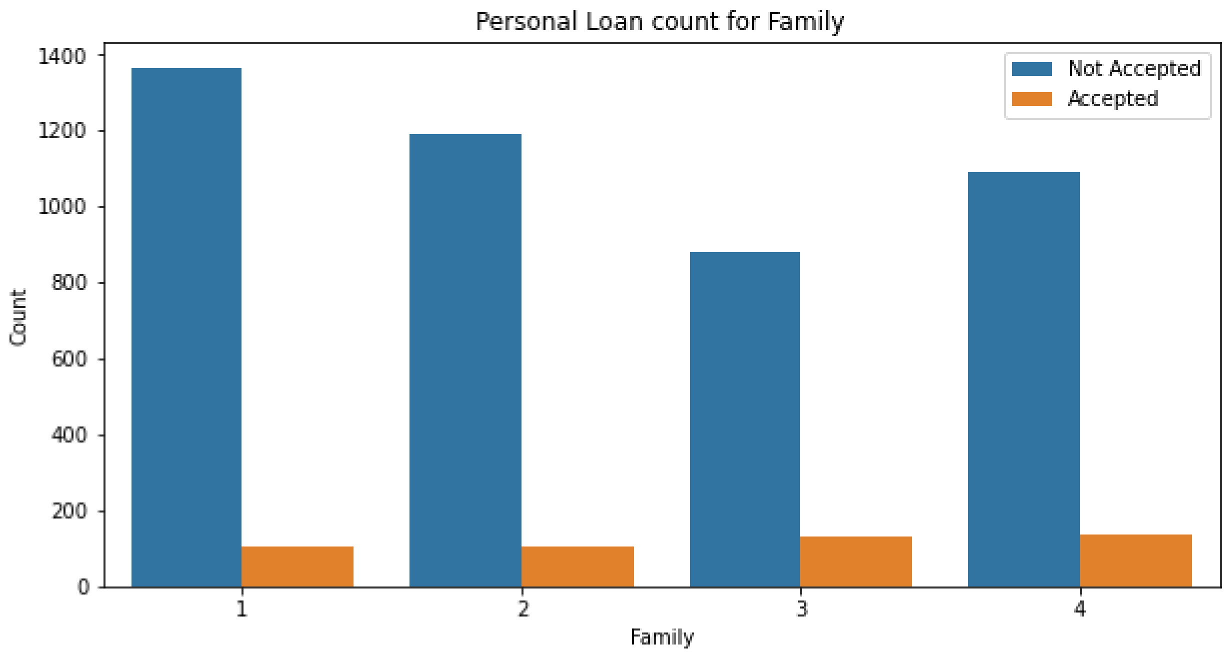
It is quite weird about experience can be negative. we will deal while doing data processing

In [11]:
```
plt.figure(figsize=(20,6))
sns.countplot(data=df,x='Income',hue='Personal Loan')
plt.title('Personal Loan count for Income')
plt.xlabel('Income')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```

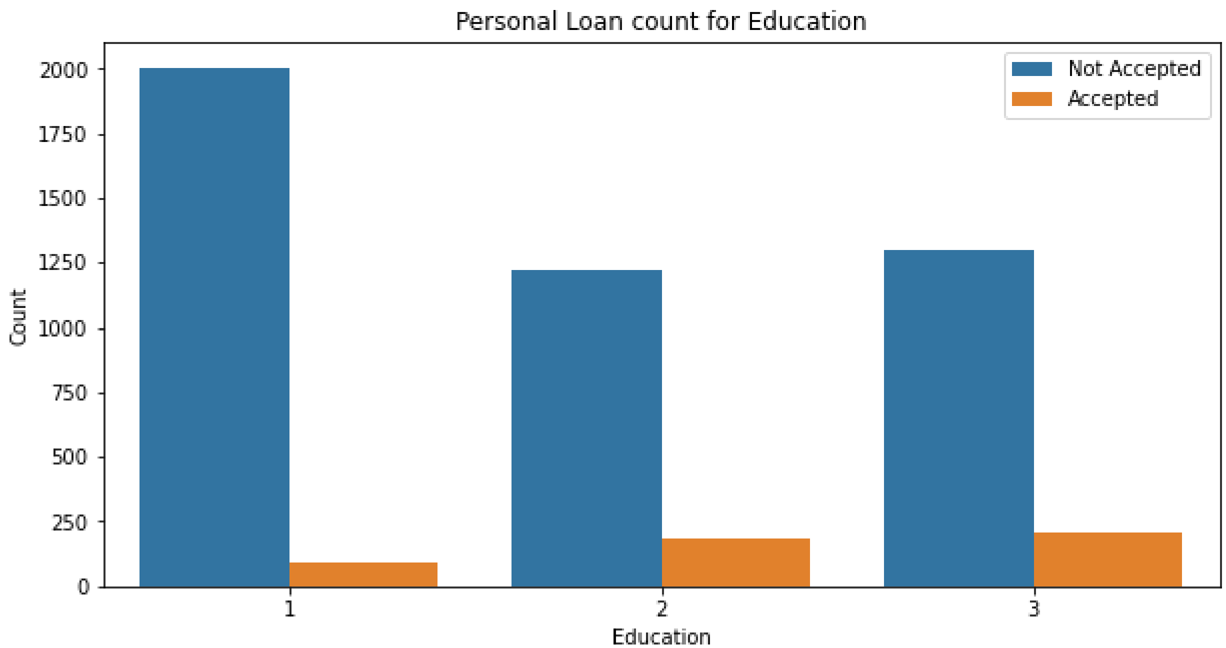


Too much clumsy.

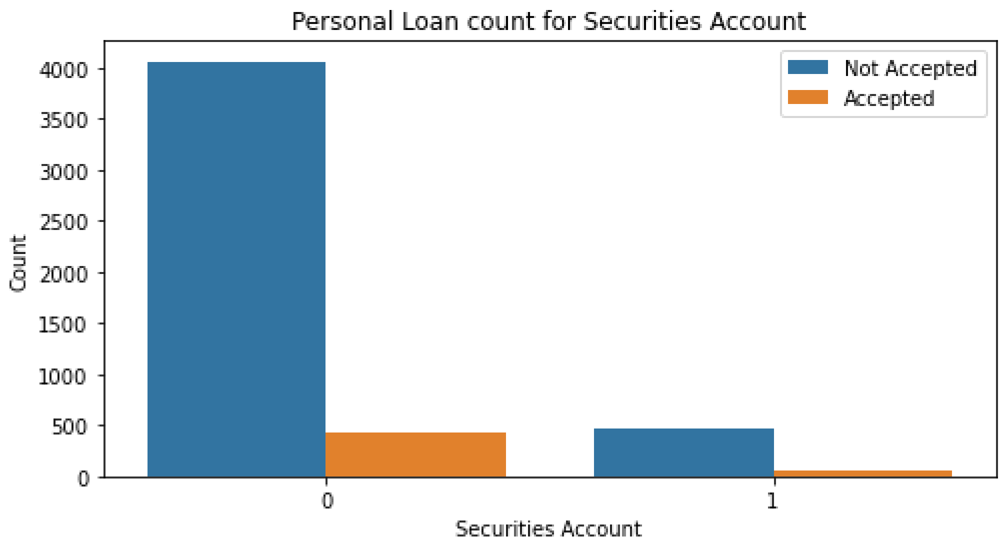
In [12]:
```
plt.figure(figsize=(10,5))
sns.countplot(data=df,x='Family',hue='Personal Loan')
plt.title('Personal Loan count for Family')
plt.xlabel('Family')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



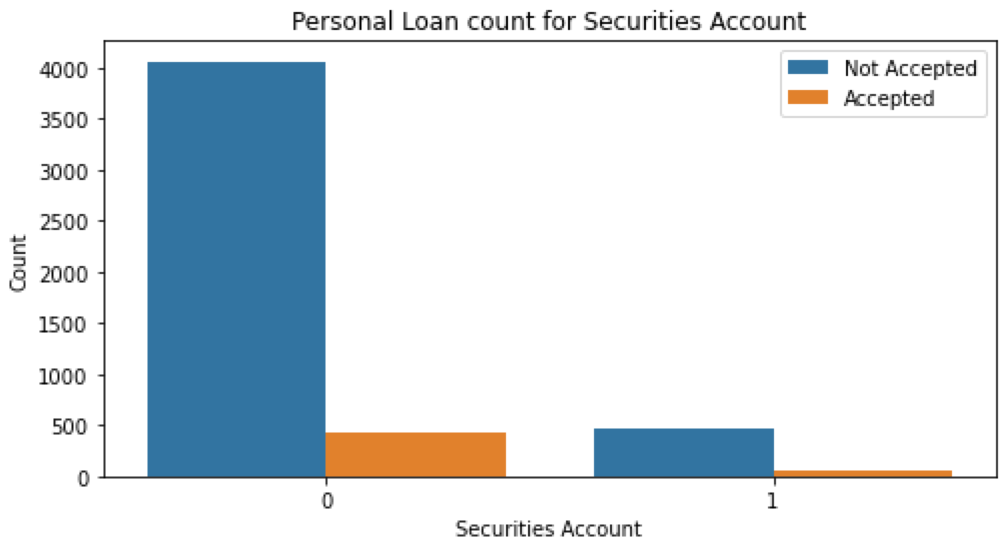
In [13]:
```
plt.figure(figsize=(10,5))
sns.countplot(data=df,x='Education',hue='Personal Loan')
plt.title('Personal Loan count for Education')
plt.xlabel('Education')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



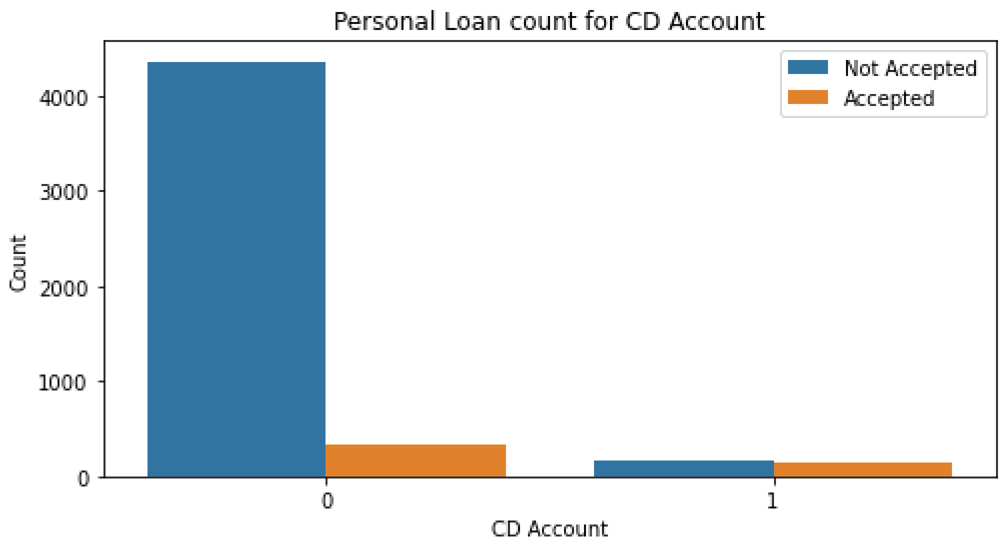
In [14]:
```
plt.figure(figsize=(8,4))
sns.countplot(data=df,x='Securities Account',hue='Personal Loan')
plt.title('Personal Loan count for Securities Account')
plt.xlabel('Securities Account')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



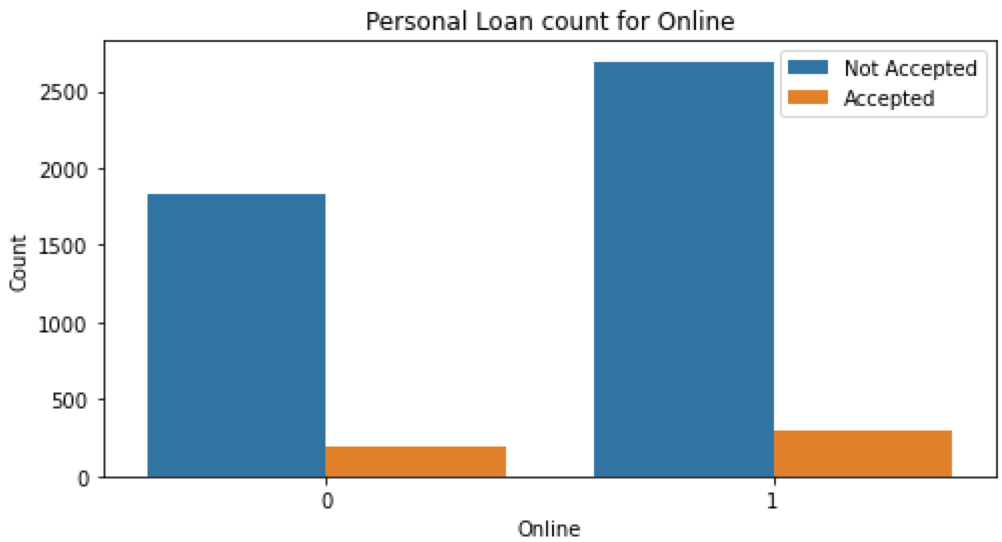
In [15]:
```
plt.figure(figsize=(8,4))
sns.countplot(data=df,x='Securities Account',hue='Personal Loan')
plt.title('Personal Loan count for Securities Account')
plt.xlabel('Securities Account')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



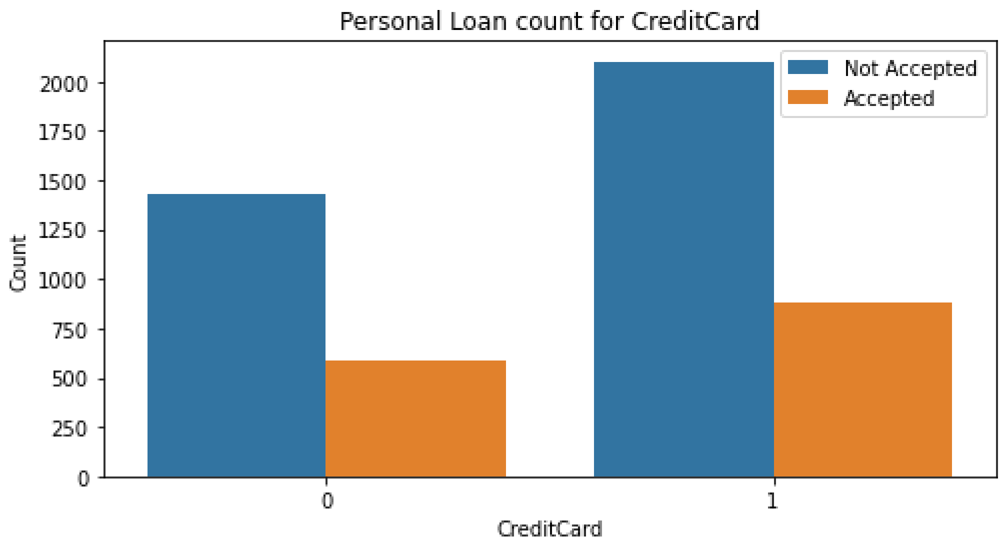
In [16]:
```
plt.figure(figsize=(8,4))
sns.countplot(data=df,x='CD Account',hue='Personal Loan')
plt.title('Personal Loan count for CD Account')
plt.xlabel('CD Account')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



In [17]:
```
plt.figure(figsize=(8,4))
sns.countplot(data=df,x='Online',hue='Personal Loan')
plt.title('Personal Loan count for Online')
plt.xlabel('Online')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



In [18]:
```
plt.figure(figsize=(8,4))
sns.countplot(data=df,x='Online',hue='CreditCard')
plt.title('Personal Loan count for CreditCard')
plt.xlabel('CreditCard')
plt.ylabel('Count')
plt.legend([ "Not Accepted","Accepted"])
plt.show()
```



Dividing the columns in the dataset in to numeric and categorical attributes.

In [19]:
```
cols = set(df.columns)
cols_numeric = set(['Age', 'Experience', 'Income', 'CCAvg', 'Mortgage','Personal Loan'])
cols_categorical = list(cols - cols_numeric)
cols_categorical
```

Out[19]:
```
['CD Account',
 'Family',
 'Securities Account',
 'Online',
 'CreditCard',
 'Education']
```

In [20]:
```
for col in cols_categorical:
    df[col] = df[col].astype('object')
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Age                 5000 non-null   int64  
 1   Experience          5000 non-null   int64  
 2   Income              5000 non-null   int64  
 3   Family              5000 non-null   object 
 4   CCAvg               5000 non-null   float64
 5   Education           5000 non-null   object 
 6   Mortgage            5000 non-null   int64  
 7   Personal Loan       5000 non-null   int64  
 8   Securities Account  5000 non-null   object 
 9   CD Account          5000 non-null   object 
 10  Online              5000 non-null   object 
 11  CreditCard          5000 non-null   object 
dtypes: float64(1), int64(5), object(6)
memory usage: 468.9+ KB
```

Separate Data

In [22]:
```
df_num = df.select_dtypes(include=['int64','float64']).drop("Personal Loan",axis=1)
df_cat = df.select_dtypes(include='object')
```

In [23]:
```
df_num.head(3)
```

Out[23]:

	Age	Experience	Income	CCAvg	Mortgage
0	25	1	49	1.6	0
1	45	19	34	1.5	0
2	39	15	11	1.0	0

In [24]:
```
df_cat.head(3)
```

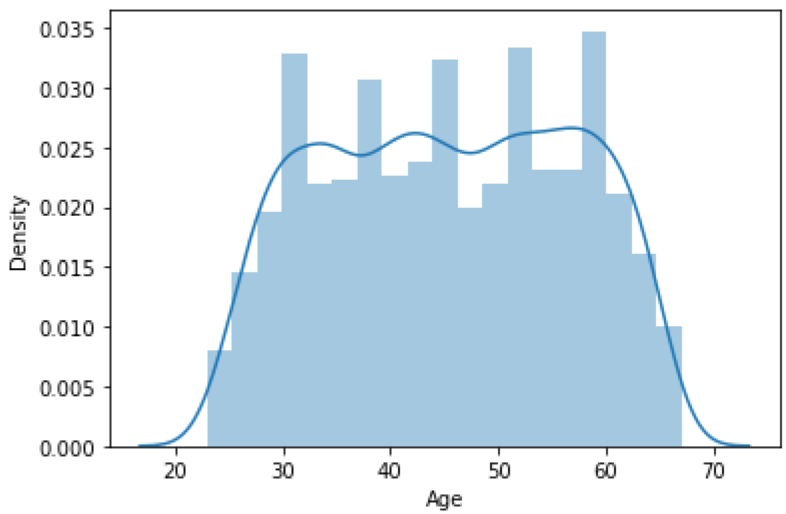
Out[24]:

	Family	Education	Securities Account	CD Account	Online	CreditCard
0	4	1	1	0	0	0
1	3	1	1	0	0	0
2	1	1	0	0	0	0

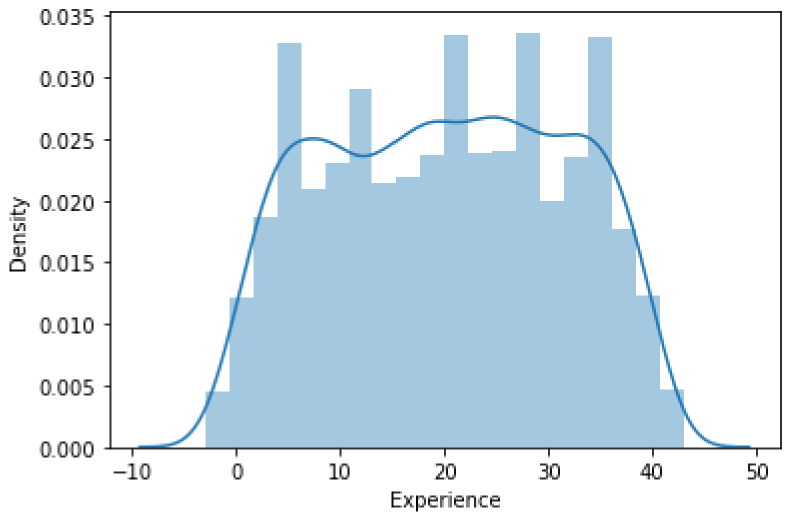
Skewness

In [25]:
```
for col in df_num:
    print(col, skew(df_num[col]))
    plt.figure()
    sns.distplot(df_num[col])
    plt.show()
```

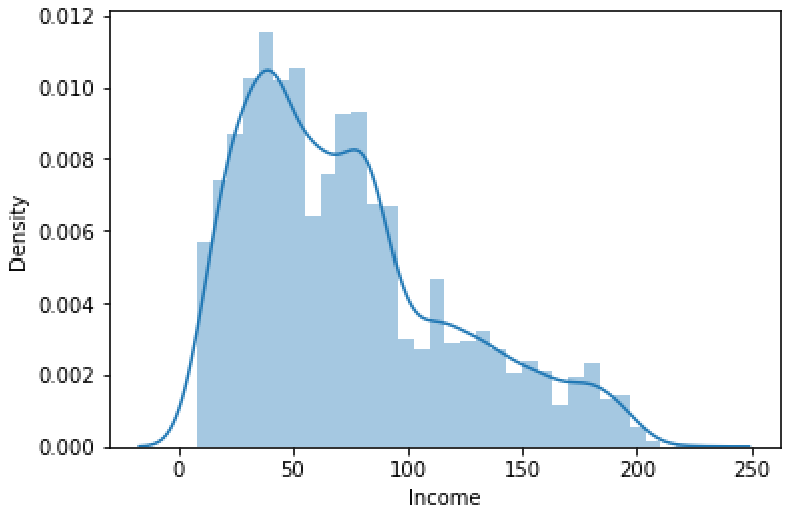
```
Age -0.02933187857476698
```



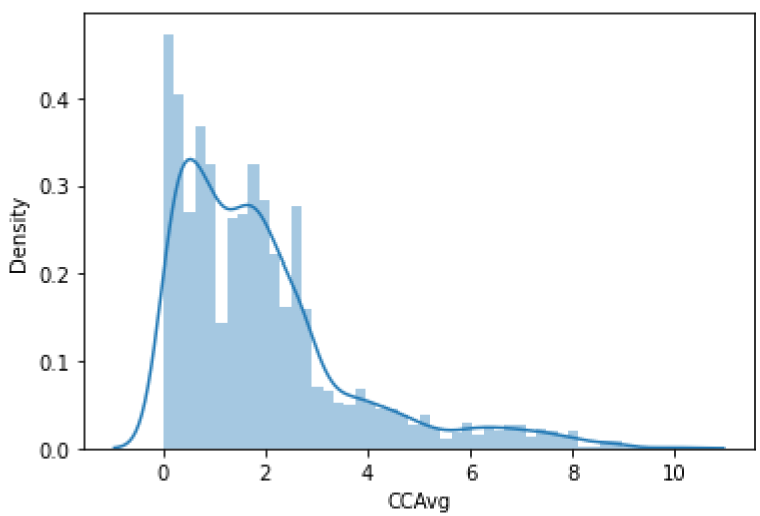

```
Experience -0.026316790337654442
```



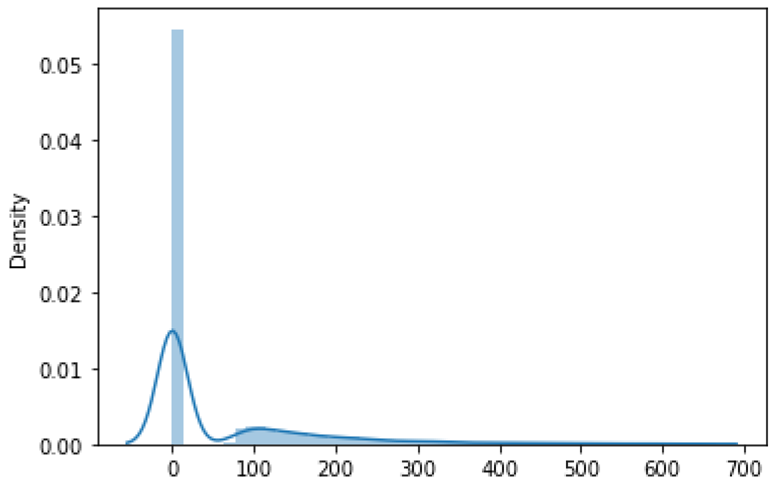

```
Income 0.8410861846424931
```



CCAvg 1.5979637637001873



Mortgage 2.103371065804789



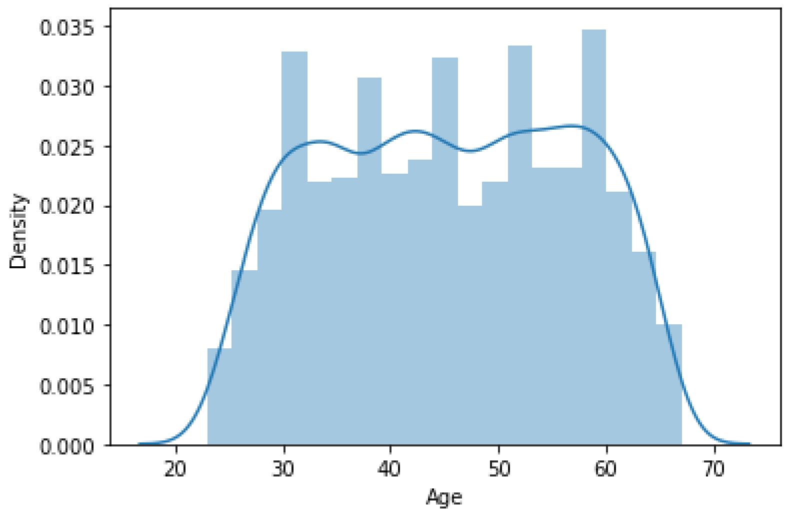
Reducing Skewness

Columns Income,CCAvg and Morgage having positive skewness or right skewness. so try to reduce the skewness by applying square root.

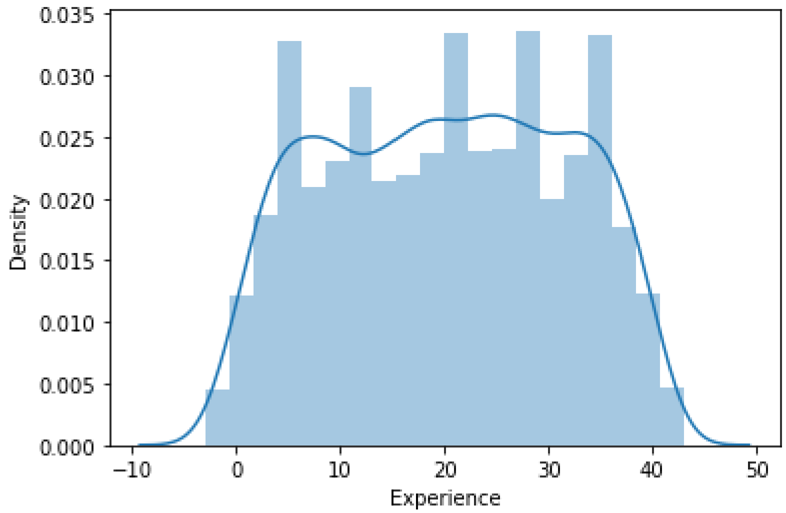
In [26]:
```
skew_col = ['Income','CCAvg','Morgage']

for col in df_num:
    if col in skew_col:
        df_num[col] = np.sqrt(df_num[col])
        print(col, skew(df_num[col]))
    else:
        print(col, skew(df_num[col]))
    plt.figure()
    sns.distplot(df_num[col])
    plt.show()
```

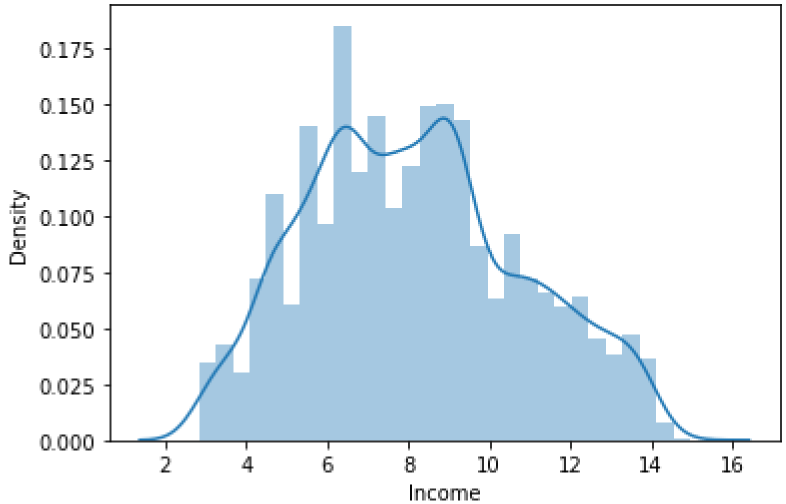
Age -0.029331878574766698



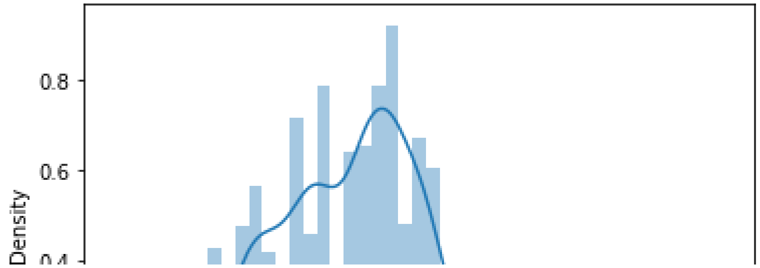
Experience -0.026316790337654442



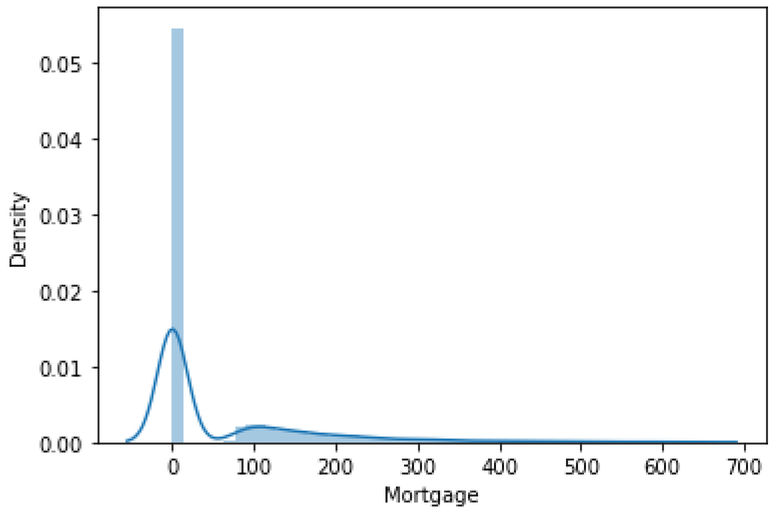

```
Income 0.26035759523724794
```




```
CCAvg 0.4238991859957578
```

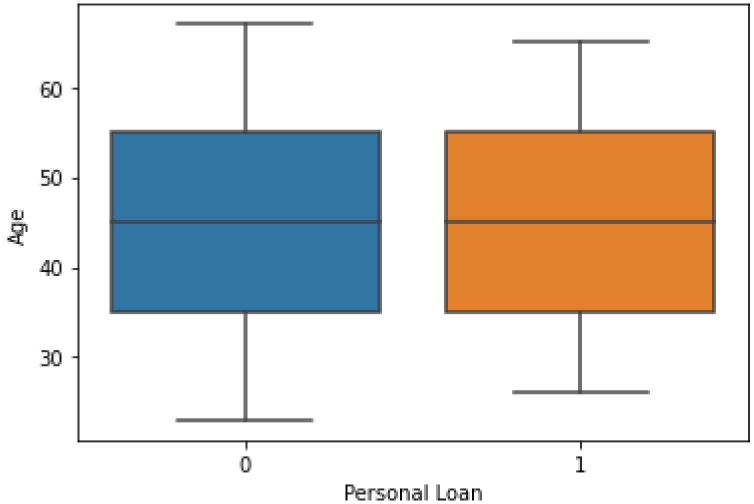


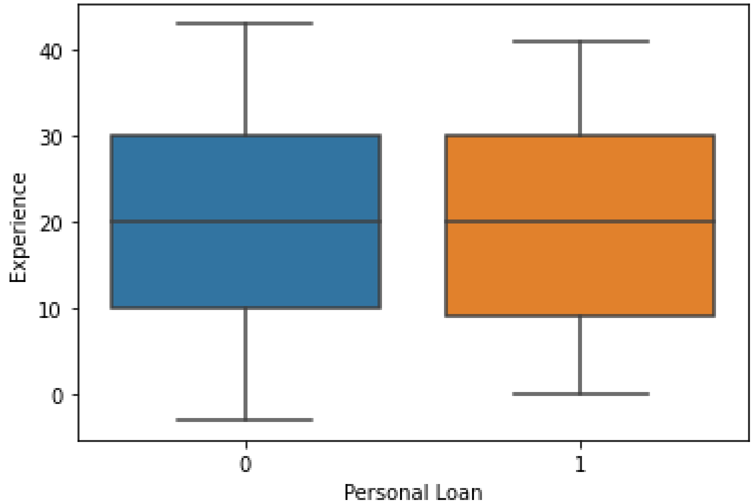

```
Mortgage 2.103371065804789
```

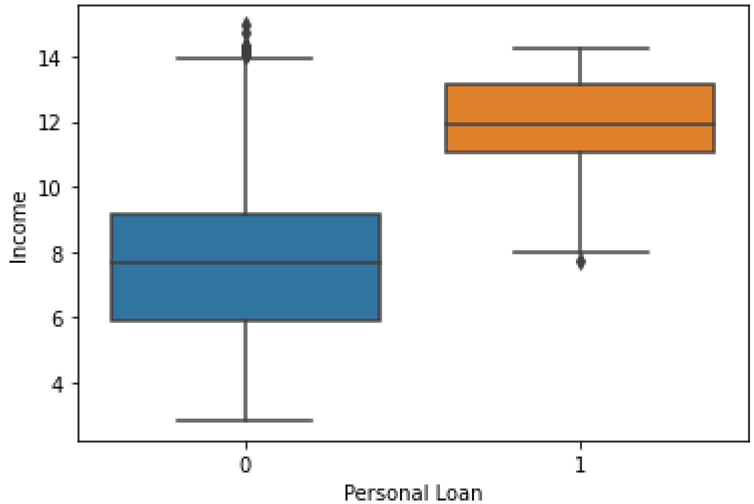


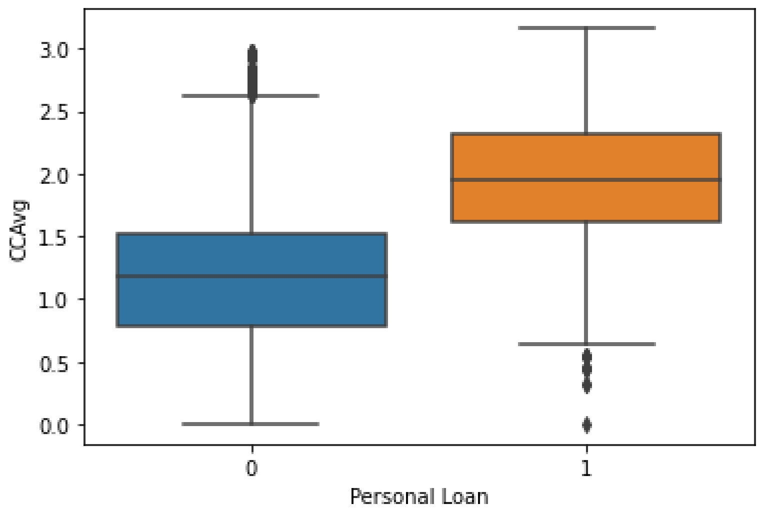
Outliers

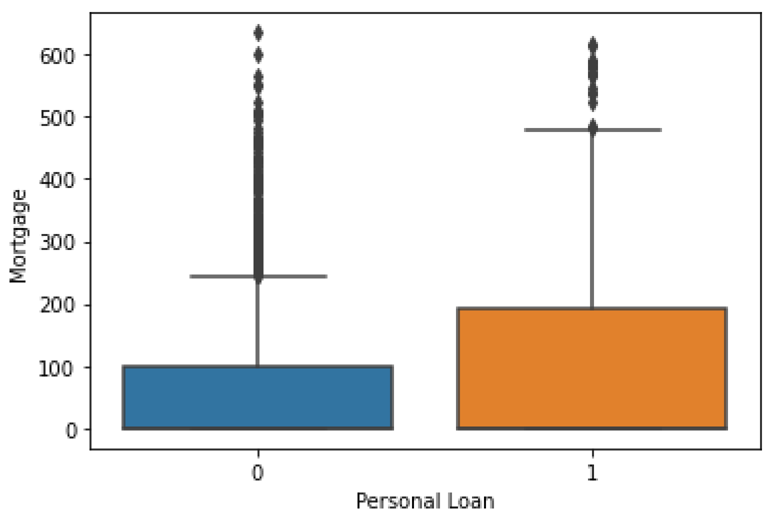
In [27]:
```
for col in df_num:
    plt.figure()
    sns.boxplot(df["Personal Loan"],df_num[col])
    plt.show()
```




Experience
40
30
20
10
0
0
1
Personal Loan


Income
14
12
10
8
6
4
0
1
Personal Loan


CCAvg
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
Personal Loan


Mortgage
600
500
400
300
200
100
0
0
1
Personal Loan

Data Cleaning

In [28]:
```
df_num.describe()
```

Out[28]:

	Age	Experience	Income	CCAvg	Mortgage
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	45.338400	20.104600	8.166221	1.253347	56.498800
std	11.463166	11.467954	2.662416	0.605915	101.713802
min	23.000000	-3.000000	2.828427	0.000000	0.000000
25%	35.000000	10.000000	6.244998	0.836660	0.000000
50%	45.000000	20.000000	8.000000	1.224745	0.000000
75%	55.000000	30.000000	9.899495	1.581139	101.000000
max	67.000000	43.000000	14.966630	3.162278	635.000000

In [29]:
```
df_num[df_num['Mortgage'] == 0]['Mortgage'].count()
```

Out[29]:
```
3462
```

Mortgage : Value of house mortgage if any. it could be possible that a person have no house in his morgage. so i live as it is

Training Dataset and Testing Dataset

Training Dataset and Testing Dataset

In [35]:
```
df_clean = pd.concat([df_cat,df_num],axis=1)
```

In [36]:
```
X = df_clean
y = df["Personal Loan"]
```

In [37]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

In [38]:
```
X_train.columns
```

Out[38]:
```
Index(['Family', 'Education', 'Securities Account', 'CD Account', 'Online',
       'CreditCard', 'Age', 'Experience', 'Income', 'CCAvg', 'Mortgage'],
      dtype='object')
```

In [40]:
```
def create_model(model,X_train=X_train,y_train=y_train):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print(classification_report(y_test,y_pred))
    return model
```

Baseline Model - Logistic Regression

In [43]:
```
lr = LogisticRegression()
create_model(lr)
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      1351
           1       0.81      0.57      0.67       149

    accuracy                           0.94      1500
   macro avg       0.88      0.78      0.82      1500
weighted avg       0.94      0.94      0.94      1500
```

Out[43]:
```
LogisticRegression()
```

Decision Tree Classifier

In [44]:
```
dt = DecisionTreeClassifier()
create_model(dt)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1351
           1       0.90      0.88      0.89       149

    accuracy                           0.98      1500
   macro avg       0.94      0.93      0.94      1500
weighted avg       0.98      0.98      0.98      1500
```

Out[44]:
```
DecisionTreeClassifier()
```

Random Forest Classifier

In [52]:
```
rt =RandomForestClassifier(n_estimators=200,max_depth=10)
create_model(rt)
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1351
           1       0.97      0.84      0.90       149

    accuracy                           0.98      1500
   macro avg       0.98      0.92      0.94      1500
weighted avg       0.98      0.98      0.98      1500
```

Out[52]:
```
RandomForestClassifier(max_depth=10, n_estimators=200)
```

Gradient Boosting

In [56]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [64]:
```
gb = GradientBoostingClassifier(n_estimators=100,max_depth=25)
create_model(gb)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1351
           1       0.91      0.88      0.89       149

    accuracy                           0.98      1500
   macro avg       0.95      0.93      0.94      1500
weighted avg       0.98      0.98      0.98      1500
```

Out[64]:
```
GradientBoostingClassifier(max_depth=25)
```

SVM

In [66]:
```
from sklearn.svm import SVC
```

1. Polynomial

In [70]:
```
poly_svc = SVC(random_state=1,kernel="poly",C=0.5)
create_model(poly_svc)
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95      1351
           1       0.88      0.05      0.09       149

    accuracy                           0.90      1500
   macro avg       0.89      0.52      0.52      1500
weighted avg       0.90      0.90      0.86      1500
```

Out[70]:
```
SVC(C=0.5, kernel='poly', random_state=1)
```

2. Radial Bias

In [71]:
```
radial_svc = SVC(random_state=1,kernel="rbf")
create_model(radial_svc)
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95      1351
           1       0.62      0.03      0.06       149

    accuracy                           0.90      1500
   macro avg       0.76      0.52      0.51      1500
weighted avg       0.88      0.90      0.86      1500
```

Out[71]:
```
SVC(random_state=1)
```