In [44]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

LOAN DEFAULT PREDICTION USING MACHINE LEARNING

A small brief overview of the dataset

In [71]:
```
df = pd.read_csv('loan prediction data.csv')#dataset accessed from kaggle website
df.head()
```

Out[71]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	

First started by filling in any rows with missing information in the dataset with the mean of the responses in the columns

In [72]:
```
for column in ['Gender','Dependents', 'Self_Employed']:
    df[column].fillna(df[column].mode()[0],inplace = True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(),inplace = True)
df['LoanAmount'].fillna(df['Loan_Amount_Term'].mean(),inplace = True)
df['Credit_History'].fillna(df['Credit_History'].mean(),inplace = True)
```

Now I change categorical answers into numerical values so they can be computed

In [73]:
```
le = LabelEncoder()
for column in ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area']:
    df[column] = le.fit_transform(df[column])
```

In [74]:
```
df.head()
```

Out[74]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001015	1	1	0	0	0	5720	
1	LP001022	1	1	1	0	0	3076	
2	LP001031	1	1	2	0	0	5000	
3	LP001035	1	1	2	0	0	2340	
4	LP001051	1	0	0	1	0	3276	

In [75]:
```
mean_value1 = df['ApplicantIncome'].mean()
print(mean_value1)
mean_value2= df['CoapplicantIncome'].mean()
print(mean_value2)
```

```
4805.599455040872
1569.5776566757493
```

Getting the mean of the two income columns

In [76]:
```
df.dtypes
```

Out[76]:
```
Loan_ID               object
Gender                 int32
Married                int32
Dependents            object
Education              int32
Self_Employed          int32
ApplicantIncome        int64
CoapplicantIncome      int64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area          int32
dtype: object
```

Checking to see if all columns are recognized as the classes we want. Dependents is recognized as an object despite having numerical vales because it has specail characters ie;3+ meaning it still needs to be converted either to an int or float to be computed

In [77]:
```
df.head(21)
```

Out[77]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	LP001015	1	1	0	0	0	5720	
1	LP001022	1	1	1	0	0	3076	
2	LP001031	1	1	2	0	0	5000	
3	LP001035	1	1	2	0	0	2340	
4	LP001051	1	0	0	1	0	3276	
5	LP001054	1	1	0	1	1	2165	
6	LP001055	0	0	1	1	0	2226	
7	LP001056	1	1	2	1	0	3881	
8	LP001059	1	1	2	0	0	13633	
9	LP001067	1	0	0	1	0	2400	
10	LP001078	1	0	0	1	0	3091	
11	LP001082	1	1	1	0	0	2185	
12	LP001083	1	0	3+	0	0	4166	
13	LP001094	1	1	2	0	0	12173	
14	LP001096	0	0	0	0	0	4666	
15	LP001099	1	0	1	0	0	5667	
16	LP001105	1	1	2	0	0	4583	
17	LP001107	1	1	3+	0	0	3786	
18	LP001108	1	1	0	0	0	9226	
19	LP001115	1	0	0	0	0	1300	
20	LP001121	1	1	1	1	0	1888	

Replacing any Unique characters/values in the Dependents Column to be easily computed. 3+ is replaced with a standard 4 for anyone with more than 3 dependents

In [78]:
```
print('Unique values before replacement:',df['Dependents'].unique())
df.loc[df['Dependents']=='3+', 'Dependents']='4'
df['Dependents']=pd.to_numeric(df['Dependents'])
```

```
Unique values before replacement: ['0' '1' '2' '3+']
```

Confirming to see whether the values were replaced

In [79]:
```
df.head(21)
```

Out[79]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	LP001015	1	1	0	0	0	5720	
1	LP001022	1	1	1	0	0	3076	
2	LP001031	1	1	2	0	0	5000	
3	LP001035	1	1	2	0	0	2340	
4	LP001051	1	0	0	1	0	3276	
5	LP001054	1	1	0	1	1	2165	
6	LP001055	0	0	1	1	0	2226	
7	LP001056	1	1	2	1	0	3881	
8	LP001059	1	1	2	0	0	13633	
9	LP001067	1	0	0	1	0	2400	
10	LP001078	1	0	0	1	0	3091	
11	LP001082	1	1	1	0	0	2185	
12	LP001083	1	0	4	0	0	4166	
13	LP001094	1	1	2	0	0	12173	
14	LP001096	0	0	0	0	0	4666	
15	LP001099	1	0	1	0	0	5667	
16	LP001105	1	1	2	0	0	4583	
17	LP001107	1	1	4	0	0	3786	
18	LP001108	1	1	0	0	0	9226	
19	LP001115	1	0	0	0	0	1300	
20	LP001121	1	1	1	1	0	1888	

checking to see if the columns are of the intended class

In [80]:
```
df.dtypes
```

Out[80]:
```
Loan_ID               object
Gender                 int32
Married                int32
Dependents             int64
Education              int32
Self_Employed          int32
ApplicantIncome        int64
CoapplicantIncome      int64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area          int32
dtype: object
```

Creating a Default column by assigning values from the dataset. 1 to represent default and 0 representing no default

In [81]:
```
conditions = [(df['Self_Employed']==1)|(df['Property_Area']<1)|(df['Dependents']>3)|(d
              (df['Dependents']<=3)&(df['Married']==1)&(df['Education']==0)&(df['Appli
choices = [1,0]
df['Loan_Default'] = np.select(conditions, choices, default = 0)
```

checking for the default column

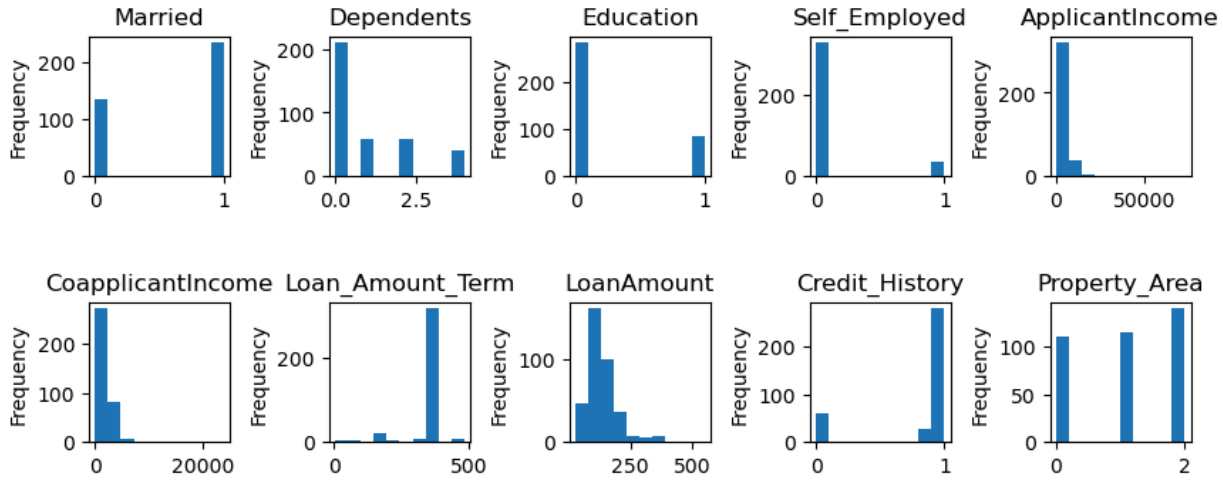
In [82]:
```
df.head(21)
```

Out[82]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	LP001015	1	1	0	0	0	5720	
1	LP001022	1	1	1	0	0	3076	
2	LP001031	1	1	2	0	0	5000	
3	LP001035	1	1	2	0	0	2340	
4	LP001051	1	0	0	1	0	3276	
5	LP001054	1	1	0	1	1	2165	
6	LP001055	0	0	1	1	0	2226	
7	LP001056	1	1	2	1	0	3881	
8	LP001059	1	1	2	0	0	13633	
9	LP001067	1	0	0	1	0	2400	
10	LP001078	1	0	0	1	0	3091	
11	LP001082	1	1	1	0	0	2185	
12	LP001083	1	0	4	0	0	4166	
13	LP001094	1	1	2	0	0	12173	
14	LP001096	0	0	0	0	0	4666	
15	LP001099	1	0	1	0	0	5667	
16	LP001105	1	1	2	0	0	4583	
17	LP001107	1	1	4	0	0	3786	
18	LP001108	1	1	0	0	0	9226	
19	LP001115	1	0	0	0	0	1300	
20	LP001121	1	1	1	1	0	1888	

plotting the distribution of column data

In [104…
```
num_rows, num_cols= 5,5
fig, axes= plt.subplots(nrows=num_rows,ncols=num_cols, figsize=(10,11),subplot_kw={'as
axes= axes.flatten()
columns_to_plot=['Married','Dependents','Education','Self_Employed','ApplicantIncome',
for i,column in enumerate(columns_to_plot):
    df[column].plot(kind='hist', bins=10, ax=axes[i], title=f'{column}')
for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])
plt.show()
```



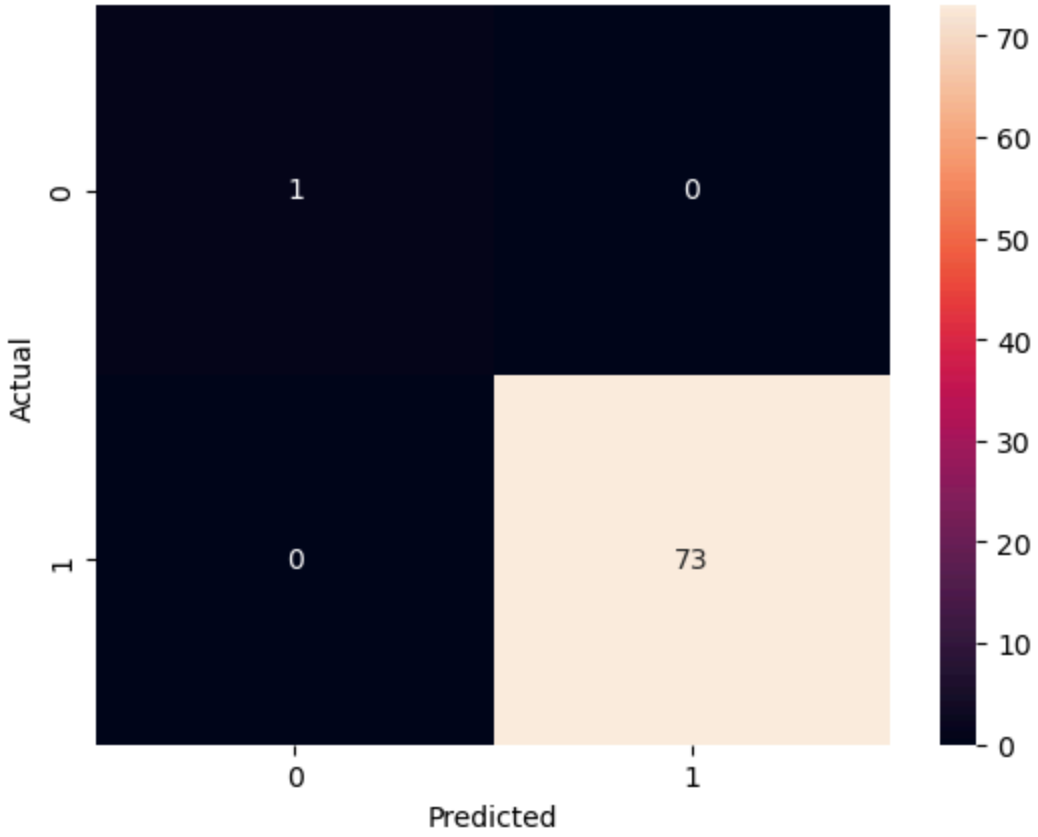
In [83]:
```
features= ['Married','Dependents','Education','Self_Employed','ApplicantIncome','Coapp
target= 'Loan_Default'
x = df[features]
y = df['Loan_Default']

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2, random_state=42)
model= RandomForestClassifier(random_state = 42)
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00        73

    accuracy                           1.00        74
   macro avg       1.00      1.00      1.00        74
weighted avg       1.00      1.00      1.00        74

[[ 1  0]
 [ 0 73]]
```

After, we go ahead to train the model(in this case i opted for a random forest classifier) to predict the number to likely default from the conditions we gave the model. The confusion matrix indicates that the model predicted 1 true negative and no false positives as well as no false negatives and 73 true positives from the dataset

A visual representation(heatmap) of the model's prediction

In [84]:
```
cm = confusion_matrix(y_test,predictions)
sns.heatmap(cm, annot=True, fmt='d')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [ ]:
```

```