STEP 1. UNDERSTAND DATASET

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

In [2]:
```
file = pd.read_csv("train_loan.csv")
file
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

614 rows × 13 columns

In [3]:
```
file.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [4]:
```
file.shape
```

Out[4]:
```
(614, 13)
```

In [5]:
```
file.columns
```

Out[5]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [6]:
```
type(file)
```

Out[6]:
```
pandas.core.frame.DataFrame
```

In [7]:
```
file.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [8]:
```
file.value_counts()
```

```
N                  1
LP001005  Male    Yes      0          Graduate      Yes            3000
0.0                66.0       360.0            1.0             Urban
Y                  1
LP002347  Male    Yes      0          Graduate      No             3246
1417.0             138.0      360.0            1.0             Semiurban
Y                  1
LP002345  Male    Yes      0          Graduate      No             1025
2773.0             112.0      360.0            1.0             Rural
Y                  1
LP002342  Male    Yes      2          Graduate      Yes            1600
20000.0            239.0      360.0            1.0             Urban
N                  1

..
LP001674  Male    Yes      1          Not Graduate  No             2600
2500.0             90.0       360.0            1.0             Semiurban
Y                  1
LP001673  Male    No       0          Graduate      Yes            11000
0.0                83.0       360.0            1.0             Urban
N                  1
```

In [9]:
```
file.describe()
```

Out[9]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

STEP 2. DATA CLEANING

In [10]:
```
file.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)
```

In [11]:
```
file.head()
```

Out[11]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [12]:
```
file['Dependents'].value_counts()
```

Out[12]:
```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [13]:
```
loan = file.replace(to_replace='3+',value=3)
```

In [14]:
```
loan.head()
```

Out[14]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [15]:
```
onvert categorical columns to numerical values
n.replace({'Married':{'No':0,'Yes':1},'Gender':{'Male':1,'Female':0},'Self_Employe
                   'Property_Area':{'Rural':0,'Semiurban':1,'Urban':2},'Education
```

In [16]:
```
loan.head()
```

Out[16]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	LP001002	1.0	0.0	0	1	0.0	5849	
1	LP001003	1.0	1.0	1	1	0.0	4583	
2	LP001005	1.0	1.0	0	1	1.0	3000	
3	LP001006	1.0	1.0	0	0	0.0	2583	
4	LP001008	1.0	0.0	0	1	0.0	6000	

In [62]:
```
e = loan.drop(columns=['Loan_ID'],axis=1)
```

In [63]:
```
e.head()
```

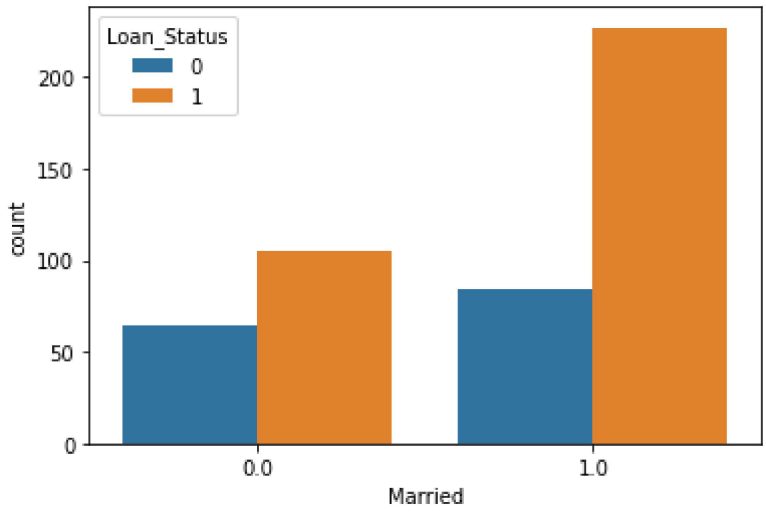
Out[63]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
1	1.0	1.0	1	1	0.0	4583	1508.0
2	1.0	1.0	0	1	1.0	3000	0.0
3	1.0	1.0	0	0	0.0	2583	2358.0
4	1.0	0.0	0	1	0.0	6000	0.0
5	1.0	1.0	2	1	1.0	5417	4196.0

STEP 3. EXPLORATORY DATA ANALYSIS

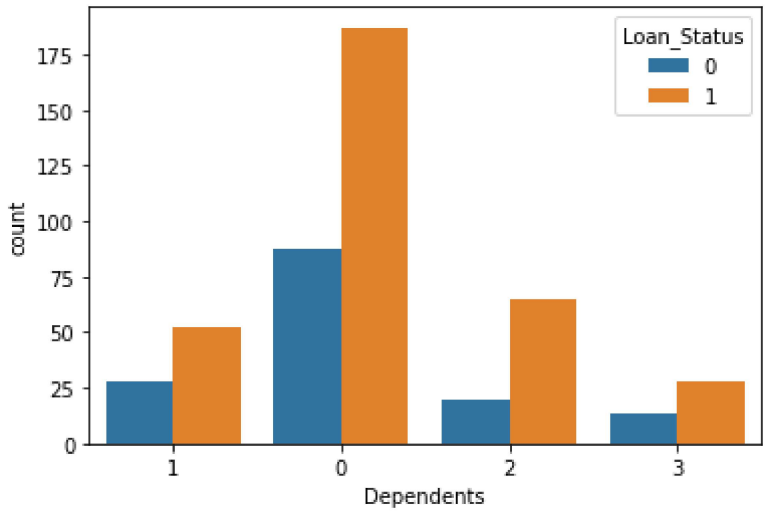
In [64]:
```
# Married & Loan Status
import seaborn as sns
sns.countplot(x='Married',hue='Loan_Status',data=loan)
```

Out[64]: <AxesSubplot:xlabel='Married', ylabel='count'>



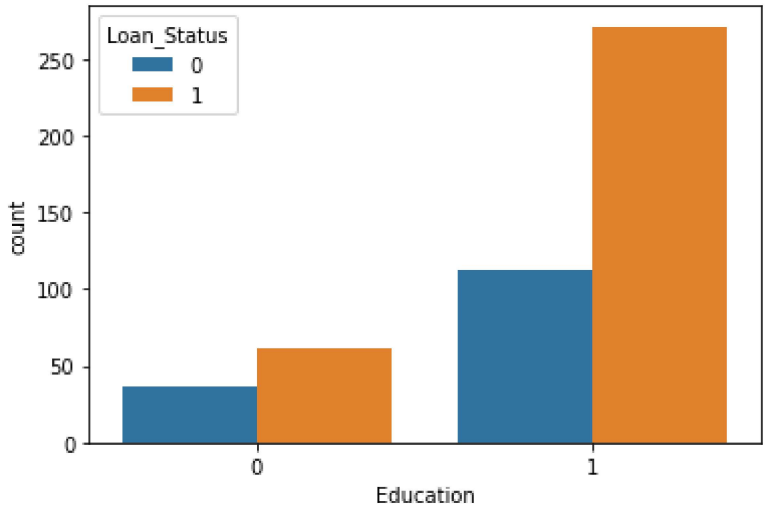
In [65]:
```
sns.countplot(x='Dependents',hue='Loan_Status',data=loan)
```

Out[65]: <AxesSubplot:xlabel='Dependents', ylabel='count'>



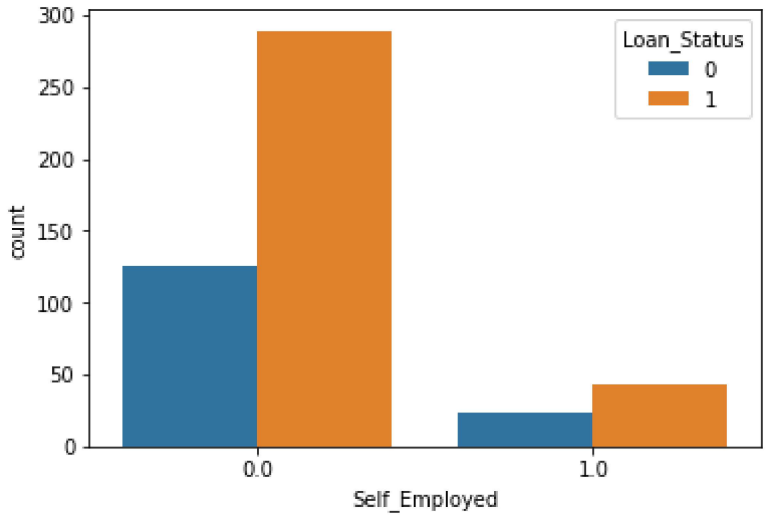
In [66]:
```
sns.countplot(x='Education',hue='Loan_Status',data=loan)
```

Out[66]:
```
<AxesSubplot:xlabel='Education', ylabel='count'>
```



In [67]:
```
sns.countplot(x='Self_Employed',hue='Loan_Status',data=loan)
```

Out[67]:
```
<AxesSubplot:xlabel='Self_Employed', ylabel='count'>
```



STEP 4. EXTRACT X AND y from the dataframe

In [78]:
```
X = loan.drop(columns=['Loan_ID'],axis=1)
y = loan[['Loan_Status']]
```

In [79]:
```
print(X)
print(y)
```

```
611               2            1
612               2            1
613               1            0

[480 rows x 12 columns]
     Loan_Status
1              0
2              1
3              1
4              1
5              1
..           ...
609            1
610            1
611            1
612            1
613            0

[480 rows x 1 columns]
```

STEP 5. ONE HOT ENCODING

In [80]:
```
encoding = pd.get_dummies(["Loan_Status"])
encoding
```

Out[80]:

	Loan_Status
0	1

STEP 6.MODEL BUILDING

In [86]:
```
from sklearn.model_selection import train_test_split as tts
```

In [87]:
```
X_train,X_test,Y_train,Y_test=tts(X,Y,test_size=0.3,stratify=Y,random_state=42)
```

In [93]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [96]:
```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
model = LinearSVC()

# train the model on the training data
model.fit(X_train, Y_train)

# make predictions on the test data
y_pred = model.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63: Data
ConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:985: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

In [97]:
```
accuracy = accuracy_score(Y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6944444444444444
```

In [98]:
```
from sklearn.metrics import confusion_matrix

# calculate the confusion matrix
cm = confusion_matrix(Y_test, y_pred)

# print the confusion matrix
print(cm)
```

```
[[  0  44]
 [  0 100]]
```

In [99]:
```
from sklearn.metrics import classification_report
# print the classification report
print(classification_report(Y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        44
           1       0.69      1.00      0.82       100

    accuracy                           0.69       144
   macro avg       0.35      0.50      0.41       144
weighted avg       0.48      0.69      0.57       144


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
248: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
248: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1
248: UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

STEP 7. PERFORMANCE COMPARISONS

In [101]:
```
from sklearn.linear_model import LogisticRegression
# create a logistic regression model with default hyperparameters
lr_model = LogisticRegression()

# train the logistic regression model on the training data
lr_model.fit(X_train, Y_train)

# make predictions on the test data using the logistic regression model
lr_y_pred = lr_model.predict(X_test)

# calculate the accuracy of the logistic regression model
lr_accuracy = accuracy_score(Y_test, lr_y_pred)
print('Logistic regression accuracy:', lr_accuracy)

# print the classification report for the logistic regression model
print('Logistic regression classification report:')
print(classification_report(Y_test, lr_y_pred))
```

```
Logistic regression accuracy: 1.0
Logistic regression classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        44
           1       1.00      1.00      1.00       100

    accuracy                           1.00       144
   macro avg       1.00      1.00      1.00       144
weighted avg       1.00      1.00      1.00       144


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63: Data
ConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
3: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

In [102]:
```
from sklearn.linear_model import SGDClassifier
# create a SGDClassifier model with default hyperparameters
sgd_model = SGDClassifier()

# train the SGDClassifier model on the training data
sgd_model.fit(X_train, Y_train)

# make predictions on the test data using the SGDClassifier model
sgd_y_pred = sgd_model.predict(X_test)

# calculate the accuracy of the SGDClassifier model
sgd_accuracy = accuracy_score(Y_test, sgd_y_pred)
print('SGDClassifier accuracy:', sgd_accuracy)

# print the classification report for the SGDClassifier model
print('SGDClassifier classification report:')
print(classification_report(Y_test, sgd_y_pred))
```

```
SGDClassifier accuracy: 0.4513888888888889
SGDClassifier classification report:
              precision    recall  f1-score   support

           0       0.34      0.84      0.48        44
           1       0.80      0.28      0.41       100

    accuracy                           0.45       144
   macro avg       0.57      0.56      0.45       144
weighted avg       0.66      0.45      0.44       144



C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63: Data
ConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

In [103]:
```
from sklearn.svm import LinearSVC, SVC
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate some synthetic data
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, randon

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# Train and evaluate a LinearSVC model
linear_svc = LinearSVC(random_state=42)
linear_svc.fit(X_train, y_train)
linear_svc_score = linear_svc.score(X_test, y_test)
print(f"LinearSVC score: {linear_svc_score:.3f}")

# Train and evaluate an SVC model with a linear kernel
svc_linear = SVC(kernel="linear", random_state=42)
svc_linear.fit(X_train, y_train)
svc_linear_score = svc_linear.score(X_test, y_test)
print(f"SVC with linear kernel score: {svc_linear_score:.3f}")

# Train and evaluate an SVC model with a polynomial kernel
svc_poly = SVC(kernel="poly", degree=3, random_state=42)
svc_poly.fit(X_train, y_train)
svc_poly_score = svc_poly.score(X_test, y_test)
print(f"SVC with polynomial kernel score: {svc_poly_score:.3f}")

# Train and evaluate an SVC model with an RBF kernel
svc_rbf = SVC(kernel="rbf", gamma=0.1, random_state=42)
svc_rbf.fit(X_train, y_train)
svc_rbf_score = svc_rbf.score(X_test, y_test)
print(f"SVC with RBF kernel score: {svc_rbf_score:.3f}")

# Train and evaluate an SVC model with a sigmoid kernel
svc_sigmoid = SVC(kernel="sigmoid", random_state=42)
svc_sigmoid.fit(X_train, y_train)
svc_sigmoid_score = svc_sigmoid.score(X_test, y_test)
print(f"SVC with sigmoid kernel score: {svc_sigmoid_score:.3f}")
```

```
LinearSVC score: 0.835
SVC with linear kernel score: 0.845
SVC with polynomial kernel score: 0.895
SVC with RBF kernel score: 0.935
SVC with sigmoid kernel score: 0.665

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:985: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

In [ ]: