```
import pandas as pd
```

Add text cell

```
train = pd.read_csv("train.csv")
```

```
train.head()
```

	Loan_ID	Gender	Married	Dependents	Education	S
0	LP001002	Male	No	0	Graduate	
1	LP001003	Male	Yes	1	Graduate	
2	LP001005	Male	Yes	0	Graduate	
3	LP001006	Male	Yes	0	Not Graduate	
4	LP001008	Male	No	0	Graduate	

Next steps: Generate code with `train` View recommended plots

```
train.shape
```

```
(614, 13)
```

```
train.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
```

```
# checking for missing values
```

```
train.isna().sum()
```

ValueError NameError

Please explain the error:

```
NameError: name 'rf_pred' is not d
```

Hypothesis

The error message "NameError: name 'rf_pred' is not defined" indicates that the variable `rf_pred` has not been assigned a value before it is used in the `confusion_matrix` function. This likely occurred because you haven't trained and predicted using a model (presumably a RandomForestClassifier). The variable `rf_pred` is commonly used to store the predicted values from a model.

Suggested Changes

```
from sklearn.metrics import conf
from sklearn.ensemble import Ran
from sklearn.model_selection imp

# Assuming X, y are already defi

# Splitting the data into train
X_train, X_test, y_train, y_test

# Create and train the RandomFor
rf_model = RandomForestClassifie
rf_model.fit(X_train, y_train) #

# Make predictions on the test s
rf_pred = rf_model.predict(X_tes

# Now you can use rf_pred in you
print(confusion_matrix(y_test, r
print(classification_report(y_te
print("Accuracy:", accuracy_scor
```

Use code with caution

Explanation of changes:

	0
Loan_ID	Add text cell
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0

dtype: int64

```
train.describe() #descriptive statistics
```

	ApplicantIncome	CoapplicantIncome	LoanAmount
count	614.000000	614.000000	592.000000
mean	5403.459283	1621.245798	146.412162
std	6109.041673	2926.248369	85.587325
min	150.000000	0.000000	9.000000
25%	2877.500000	0.000000	100.000000
50%	3812.500000	1188.500000	128.000000
75%	5795.000000	2297.250000	168.000000
max	81000.000000	41667.000000	700.000000

```
train.describe(include=[object])
```

	Loan_ID	Gender	Married	Dependents	Educati
count	614	601	611	599	6
unique	614	2	2	4	
top	LP002990	Male	Yes	0	Gradua
freq	1	489	398	345	4

1. **Import `RandomForestClassifier`:** The code now imports the necessary `RandomForestClassifier` from `sklearn.ensemble`.
2. **Initialize and train the model:** A `RandomForestClassifier` object named `rf_model` is created and trained using the `fit` method with the training data (`X_train`, `y_train`).
3. **Make predictions:** The `predict` method of the trained model is used to predict the target variable for the test data (`X_test`). These predictions are stored in the `rf_pred` variable.
4. **Evaluation:** Now that `rf_pred` is defined, it can be used within the `confusion_matrix`, `classification_report`, and `accuracy_score` functions to evaluate the model's performance.

▶ Sources

Rate this answer

```
train['Loan_Status'].value_counts()
```

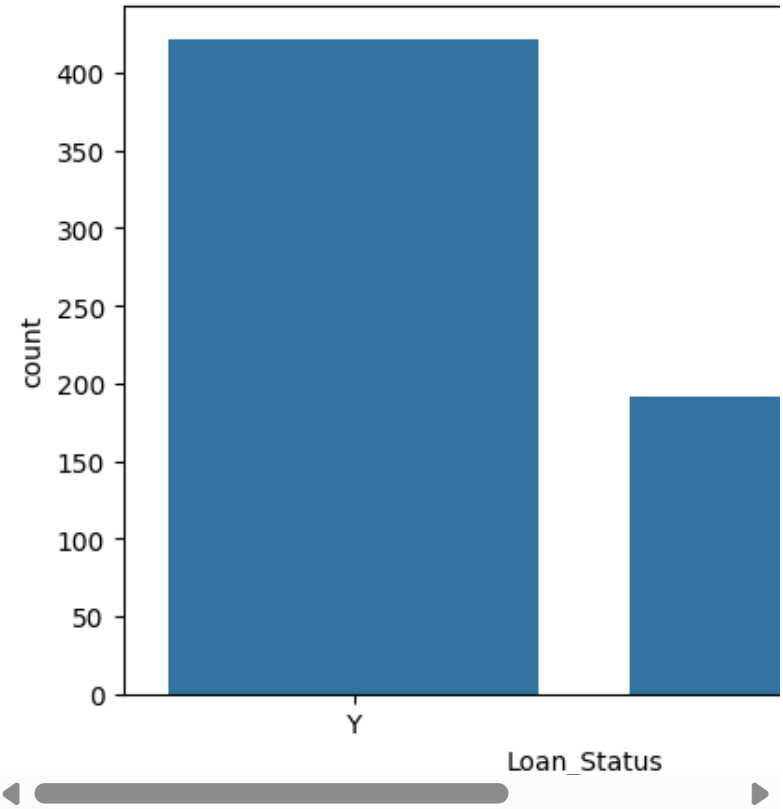
coun Add text cell

Loan_Status	
Y	422
N	192

dtype: int64

```
import seaborn as sns

sns.countplot(x=train['Loan_Status'])
```

<Axes: xlabel='Loan_Status', ylabel='count'>



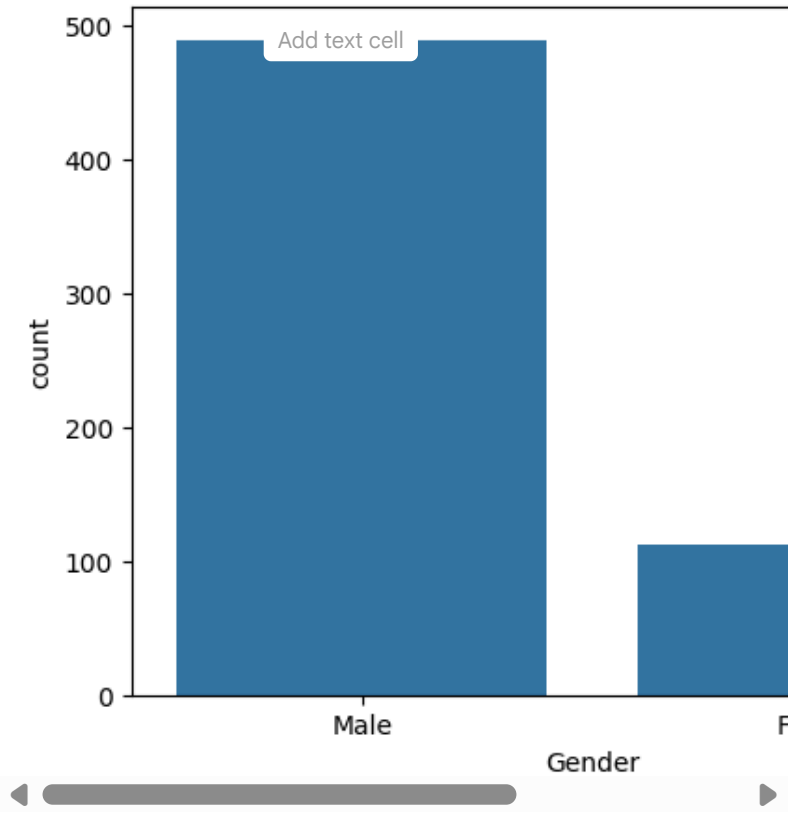

```
train['Gender'].value_counts()
```

Gender	count
Male	489
Female	112

dtype: int64

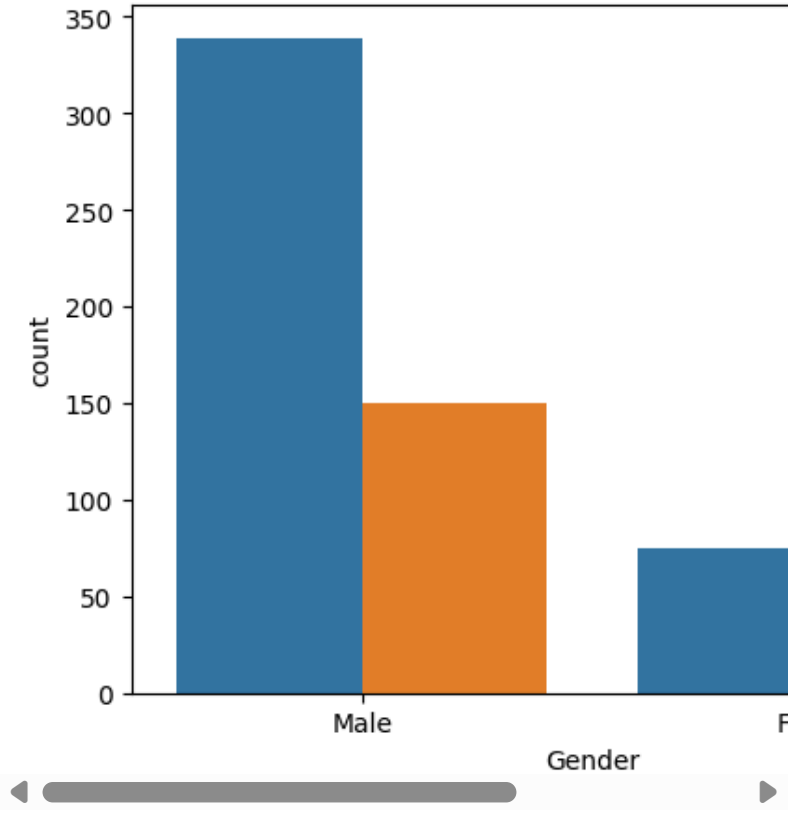
```
sns.countplot(x=train['Gender'])
```

```
<Axes: xlabel='Gender', ylabel='count'>
```



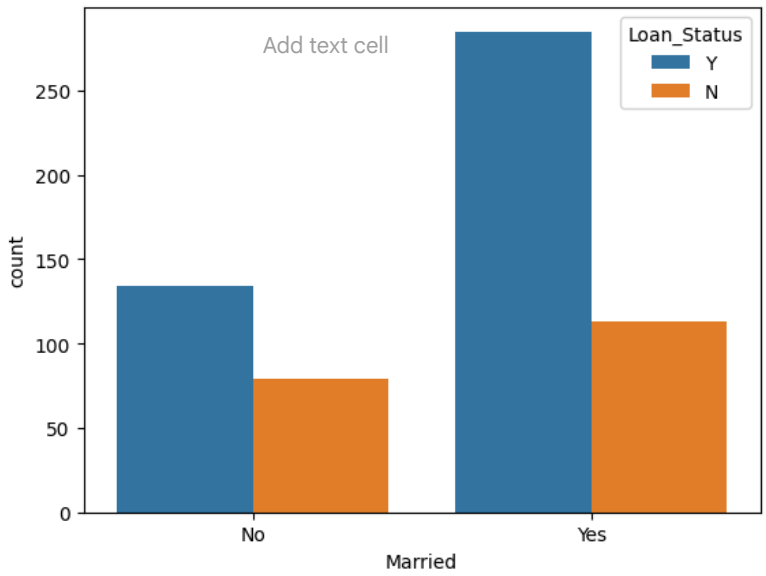

```
sns.countplot(x=train['Gender'], hue=train['Loan_Status'])
```

```
<Axes: xlabel='Gender', ylabel='count'>
```



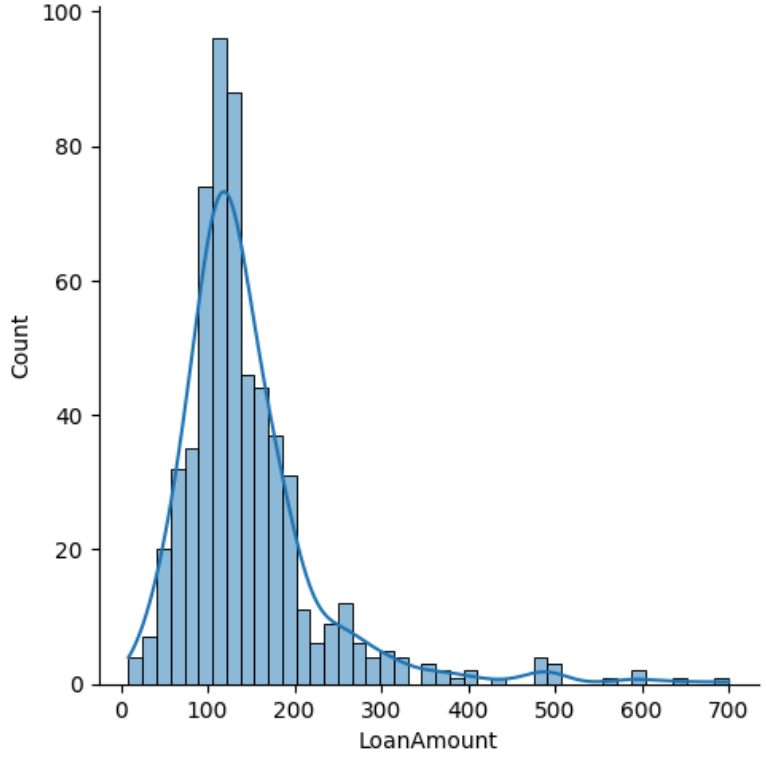

```
sns.countplot(x='Married', data=train, hue='Loan_Status')
```

```
<Axes: xlabel='Married', ylabel='count'>
```




```
sns.displot(train['LoanAmount'], kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7d6e9ffc24d0>
```



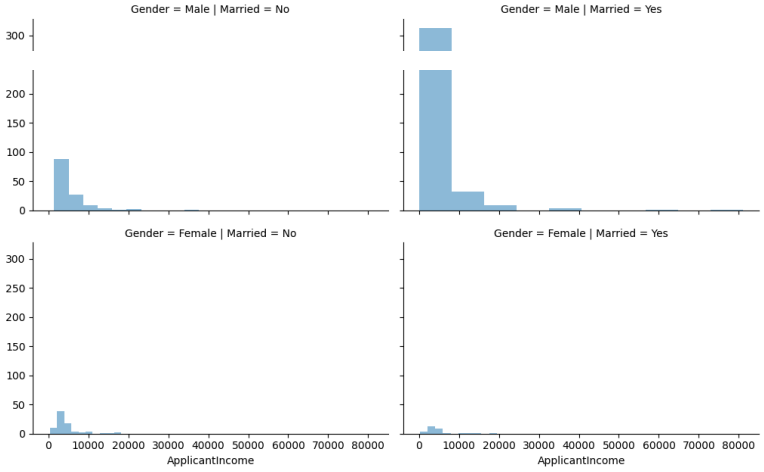

```
import matplotlib.pyplot as plt

grid = sns.FacetGrid(train, row='Gender', col='Married', he
```

```
grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)
```

Add text cell

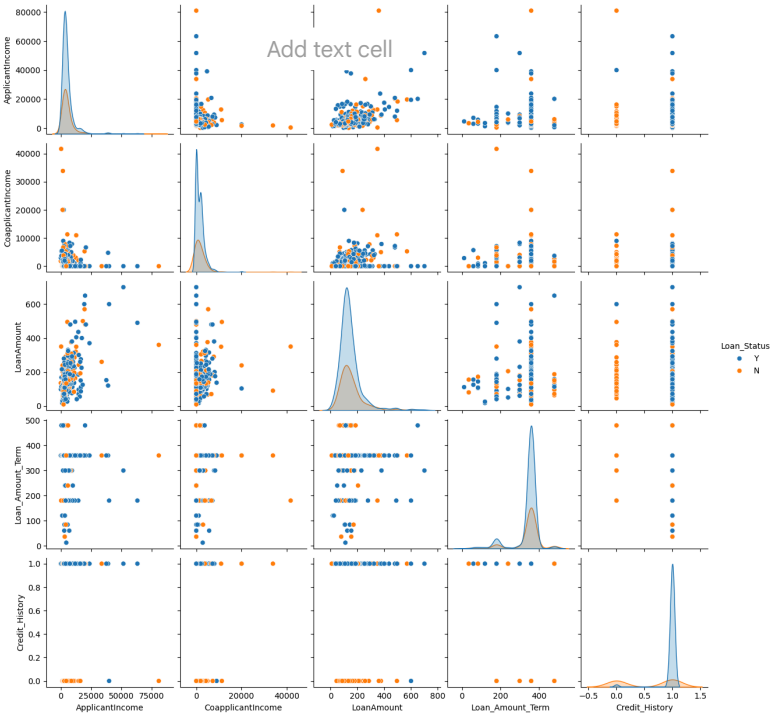
```
grid.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x7d6e65798550>
```




```
sns.pairplot(train, hue='Loan_Status', height=2.5)
```

```
<seaborn.axisgrid.PairGrid at 0x7d6e63692050>
```




```
train.isna().sum()
```

	0
Loan_ID	Add text cell
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0

dtype: int64

```
train['Married'] = train['Married'].fillna(train['Married']
```

```
train['Dependents'].fillna(train['Dependents'].mode()[0], i
```

```
<ipython-input-18-54017d2d16d3>:1: FutureWarning: A val
The behavior will change in pandas 3.0. This inplace me

For example, when doing 'df[col].method(value, inplace=

  train['Dependents'].fillna(train['Dependents'].mode()
```

```
train['Self_Employed'].fillna(train['Self_Employed'].mode()
```

```
<ipython-input-19-afd59976b687>:1: FutureWarning: A val
The behavior will change in pandas 3.0. This inplace me

For example, when doing 'df[col].method(value, inplace=

  train['Self_Employed'].fillna(train['Self_Employed'].
```

```
train['LoanAmount'].fillna(train['LoanAmount'].median(), ir

train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].

train['Credit_History'].skew()
```

```
<ipython-input-20-c03efc4edc79>:1: FutureWarning: A val
The behavior will change in pandas 3.0. This inplace me
```

Add text cell

```
For example, when doing 'df[col].method(value, inplace=

  train['LoanAmount'].fillna(train['LoanAmount'].mediar
<ipython-input-20-c03efc4edc79>:3: FutureWarning: A val
The behavior will change in pandas 3.0. This inplace me

For example, when doing 'df[col].method(value, inplace=

  train['Loan_Amount_Term'].fillna(train['Loan_Amount_T
np.float64(-1.8823610612186696)
```

```
train.isna().sum()
```

	0
Loan_ID	0
Gender	13
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	50
Property_Area	0
Loan_Status	0

dtype: int64

```
from sklearn.preprocessing import LabelEncoder

feature_col = ['Gender','Married','Dependents','Education',

le = LabelEncoder()

for col in feature_col:

    train[col] = le.fit_transform(train[col])


train.Loan_Status = train.Loan_Status.replace({"Y": 1, "N"
```

```
<ipython-input-23-92b0e0b1b9aa>:1: FutureWarning: Downc
  train.Loan_Status = train.Loan_Status.replace({"Y": 1
```

Add text cell

```
train['total_income'] = train['ApplicantIncome'] + train['C
```

```
train.drop(columns=['ApplicantIncome', 'CoapplicantIncome']
```

```
train.head(3)
```

	Loan_ID	Gender	Married	Dependents	Education	S
0	LP001002	1	0	0	0	
1	LP001003	1	1	1	0	
2	LP001005	1	1	0	0	

Next steps: Generate code with `train` View recommended plots

```
rel_feat=['Gender', 'Married', 'Dependents', 'Education', '
       'LoanAmount', 'Loan_Amount_Term', 'Credit_History',
       'Loan_Status', 'train_income']
```

```
rel_feat=['Gender', 'Married', 'Dependents', 'Education', '
       'LoanAmount', 'Loan_Amount_Term', 'Credit_History',
       'Loan_Status', 'total_income']
```

```
# Selecting only the relevant numerical features for correl
numerical_features = train[rel_feat].select_dtypes(include=
```

```
rel_feat_corr = numerical_features.corr()['Loan_Status']
```

```
print(rel_feat_corr) # To display correlation values.
```

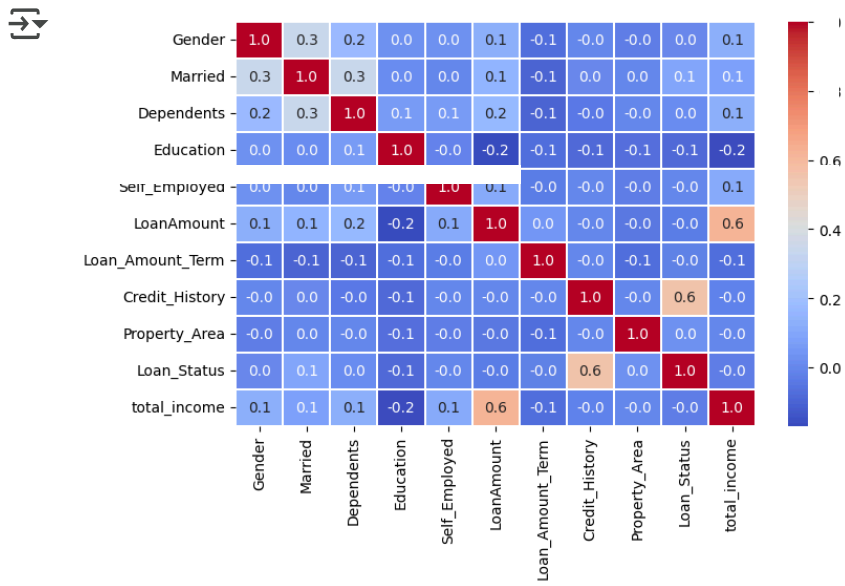
```
Gender              0.008690
Married             0.091478
Dependents          0.010118
Education          -0.085884
Self_Employed      -0.003700
LoanAmount         -0.033214
Loan_Amount_Term   -0.022549
Credit_History      0.561678
Property_Area       0.032112
Loan_Status         1.000000
total_income       -0.031271
Name: Loan_Status, dtype: float64
```

```
plt.figure(figsize=(8,5))
```

```
sns.heatmap(train[rel_feat].corr(), cmap='coolwarm', annot
```

Add text cell

```
plt.show()
```




```
#Separating target variable and other variables

X=train.drop(columns='Loan_Status')

y=train['Loan_Status']

#Splitting the data into train and test sets

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X, y, test_s
```

Enter a prompt here

0 / 2000

Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more