In [18]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
# The dataset is in CSV format
data = pd.read_csv(r"C:\Users\MANAMI DAS\OneDrive\Desktop\cmi\Project\ML\loanapprov
```

In [20]:

```python
# Check which columns in the DataFrame 'data' have the data type 'object' (usually
obj = (data.dtypes == 'object')

# Print the number of categorical columns in the DataFrame
print("Categorical variables:", len(list(obj[obj].index)))
```

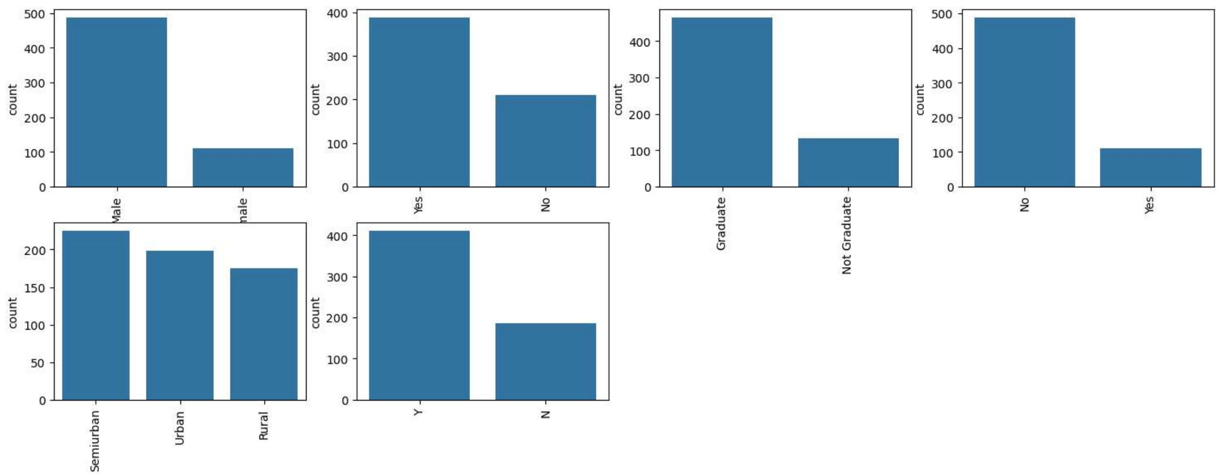
```
Categorical variables: 7
```

In [22]:

```python
# Preprocessing
# Drop columns that are not necessary for the prediction
# For example, 'Loan_ID' is just an identifier, and we don't need it for training t
data.drop(['Loan_ID'],axis=1,inplace=True)
```

In [24]:

```python
# Identify categorical columns (those with data type 'object')
obj = (data.dtypes == 'object')

# Get the list of columns that have categorical data type 'object'
object_cols = list(obj[obj].index)

# Set up the figure size for the plots
plt.figure(figsize=(18,36))

# Initialize the subplot index to place the plots
index = 1

# Loop through each categorical column and generate bar plots for their value count
for col in object_cols:
    # Get the value counts (frequency of each category) for the current categorical
    y = data[col].value_counts()

    # Create a subplot for each categorical column and set its position
    plt.subplot(11,4,index)

    # Rotate the x-axis labels by 90 degrees to avoid overlapping text
    plt.xticks(rotation=90)

    # Create a barplot with the categories on the x-axis and their frequency on the
    sns.barplot(x=list(y.index), y=y)

    # Increment the index to place the next plot in the next subplot
    index += 1
```



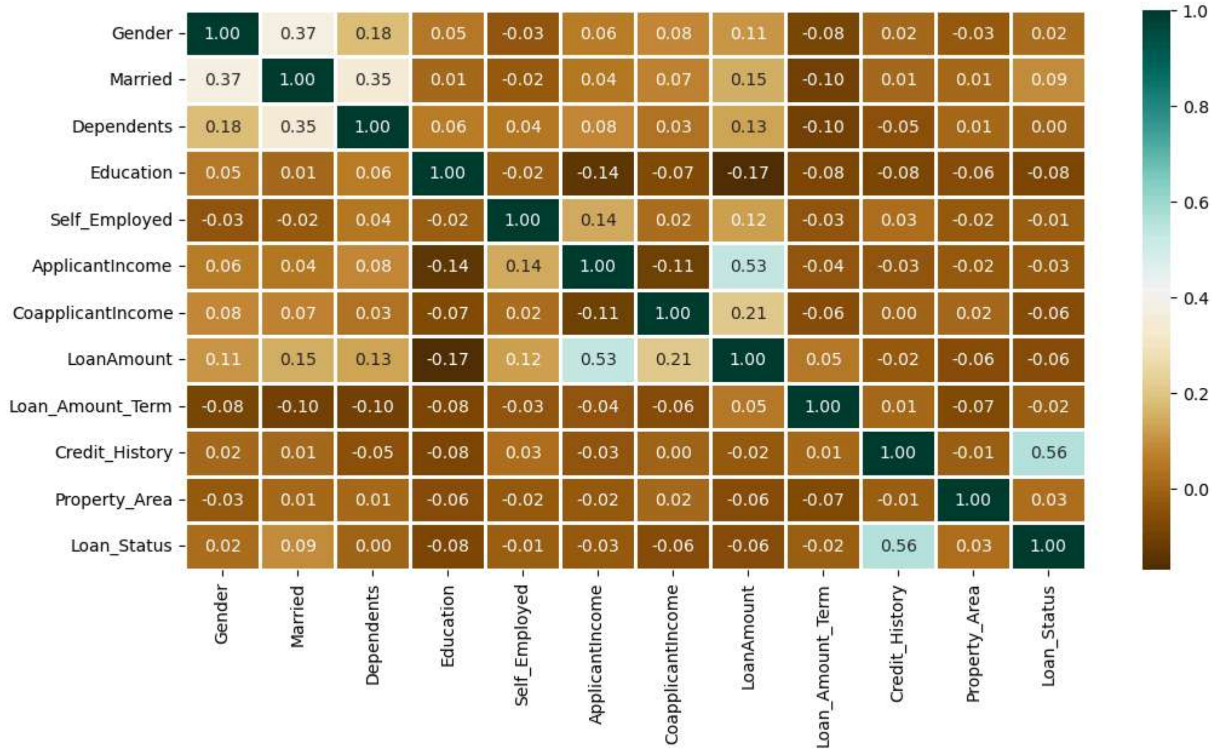

In [25]:
```
# Import label encoder from sklearn
from sklearn import preprocessing

# Create a label_encoder object which can convert categorical data to numeric label
label_encoder = preprocessing.LabelEncoder()

# Create a Boolean mask to identify categorical columns
obj = (data.dtypes == 'object')

# Loop through each column that is identified as a categorical column (i.e., those
for col in list(obj[obj].index):
    # Apply label encoding to the categorical column
    data[col] = label_encoder.fit_transform(data[col])
```

In [28]:
```
# Again check the object datatype columns. Let’s find out if there is still any lef
# Create a Boolean mask to identify columns with datatype 'object'
obj = (data.dtypes == 'object')

# Count the number of columns with datatype 'object' (categorical columns)
print("Categorical variables:", len(list(obj[obj].index)))
```

```
Categorical variables: 0
```

In [30]:
```
# Set up the figure size for the heatmap (12 inches wide, 6 inches tall)
plt.figure(figsize=(12,6))

# Create a heatmap to visualize the correlation matrix of the dataset
sns.heatmap(data.corr(), cmap='BrBG', fmt='.2f',
            linewidths=2, annot=True)
```

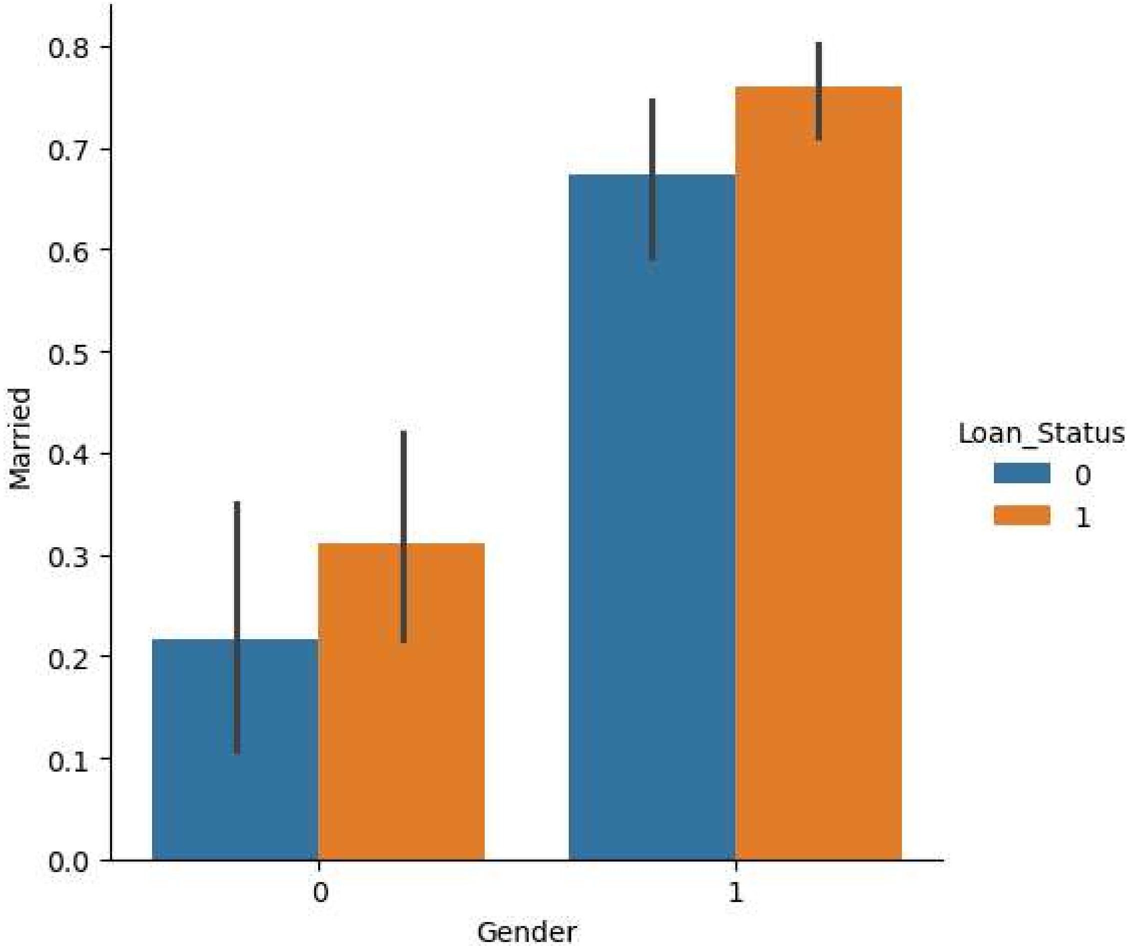
Out[30]: <Axes: >




In [32]:

```python
# Importing Seaborn for data visualization
import seaborn as sns

# Create a categorical plot (bar plot) to visualize the relationship between gender
sns.catplot(
    # 'Gender' will be plotted along the x-axis
    x="Gender",
    # 'Married' will be plotted along the y-axis
    y="Married",
    # Different hues (colors) represent the 'Loan_Status' variable (e.g., Approved,
    hue="Loan_Status",
    # We want a bar plot, as it's suited for comparing categorical data
    kind="bar",
    # The data to be used in the plot (here, 'data' is the DataFrame containing the
    data=data
)
```

Out[32]: <seaborn.axisgrid.FacetGrid at 0x2b4433c7aa0>




In [34]:
```
# Loop through all columns in the dataset
for col in data.columns:
    # Replace missing values (NaN) in each column with the mean of the respective c
    data[col] = data[col].fillna(data[col].mean())

# Check how many missing values (NaN) exist in each column after filling the missin
data.isna().sum()
```

Out[34]:
```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [36]:
```
# Importing train_test_split from sklearn to split the data into training and testi
from sklearn.model_selection import train_test_split

# Separating the features (X) and the target variable (Y)
```

```python
# Drop the 'Loan_Status' column from the dataset to get the features (X)
X = data.drop(['Loan_Status'], axis=1)

# 'Loan_Status' column is our target variable (Y)
Y = data['Loan_Status']

# Print the shape of X (features) and Y (target) to check the dimensions
# X.shape: Number of samples and features (n_samples, n_features)
# Y.shape: Number of samples (n_samples,)
X.shape, Y.shape

# Split the data into training and testing sets
# 60% of the data will be used for training and 40% for testing
# test_size=0.4: 40% for testing, random_state=1 ensures the split is reproducible
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.4,
                                                    random_state=1)

# Print the shape of training and testing sets
# X_train and Y_train will be used for training the model
# X_test and Y_test will be used for testing the model
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[36]: ((358, 11), (240, 11), (358,), (240,))

In [38]:
```python
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

# Initialize models
knn = KNeighborsClassifier(n_neighbors=3)
rfc = RandomForestClassifier(n_estimators=7, criterion='entropy', random_state=7)
svc = SVC()
lc = LogisticRegression(max_iter=500)  # Increased max_iter to avoid convergence wa

# Initialize scaler
scaler = StandardScaler()

# Scale the data: fit the scaler on the training data, and transform both training
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Making predictions on the training set
for clf in (rfc, knn, svc, lc):
    clf.fit(X_train_scaled, Y_train)  # Train on the scaled data
    Y_pred = clf.predict(X_train_scaled)  # Predict on the scaled training data
    print(f"Accuracy score of {clf.__class__.__name__} = {100*metrics.accuracy_scor
```

```
Accuracy score of RandomForestClassifier = 98.044693%
Accuracy score of KNeighborsClassifier = 81.284916%
Accuracy score of SVC = 81.005587%
Accuracy score of LogisticRegression = 80.446927%
```

In [40]:
```python
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

# Initialize models
knn = KNeighborsClassifier(n_neighbors=3)
rfc = RandomForestClassifier(n_estimators=7, criterion='entropy', random_state=7)
svc = SVC()
lc = LogisticRegression(max_iter=500)  # Increased max_iter to avoid convergence wa

# Initialize scaler
scaler = StandardScaler()

# Scale the data: fit the scaler on the training data, and transform both training
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Making predictions on the testing set
for clf in (rfc, knn, svc, lc):
    clf.fit(X_train_scaled, Y_train)  # Train the classifier on the scaled training
    Y_pred = clf.predict(X_test_scaled)  # Make predictions on the scaled test data
    print(f"Accuracy score of {clf.__class__.__name__} = {100 * metrics.accuracy_sc
```

```
Accuracy score of RandomForestClassifier = 82.500000%
Accuracy score of KNeighborsClassifier = 76.666667%
Accuracy score of SVC = 81.250000%
Accuracy score of LogisticRegression = 82.083333%
```

In [ ]:

In [ ]: