STEP 1: PROBLEM STATEMENT

Loan approval is vital in the financial sector, determining applicants' eligibility for credit based on financial profiles. This process impacts individuals' access to funds for essential needs and affects lenders' profitability and risk. It involves assessing factors like credit scores, income, and employment to grant loans responsibly, minimizing default risk. However, the process can be time-consuming and inconsistent, impacting efficiency and fairness in lending.A comprehensive evaluation process ensures that loans are granted responsibly to applicants who are most likely to repay, reducing the risk of default.

This is a classification problem where we have to predict whether a loan will be approved or not. Specifically, it is a binary classification problem where we have to predict either one of the two classes given i.e. approved or not approved

STEP 1.1: EXPECTED OUTCOME

The expected outcome of this loan approval prediction project is a robust, data-driven model capable of accurately predicting loan approval status based on applicants' financial and personal information, such as credit scores, income levels, employment status, and asset values. This model will help streamline the approval process, improve consistency in decision-making, and enhance fairness by reducing potential biases. Additionally, the analysis will offer insights into key factors influencing loan approval, enabling lending institutions to make informed, risk-aware decisions that support both applicant needs and organizational goals.

STEP 1.2: OBJECTIVE

The objective of this loan approval prediction project is to develop a reliable predictive model that identifies the likelihood of loan approval based on applicants' financial and demographic characteristics.

1. Improve Decision-Making Efficiency: Reduce the time and resources required for manual loan assessment.
2. Enhance Consistency and Fairness: Provide a standardized approach to loan approval, minimizing human bias.
3. Identify Key Predictive Factors: Offer insights into the most influential attributes in loan approval, helping to guide lending policies and strategies.
4. Mitigate Default Risk: Enable responsible lending by approving loans for applicants with a high probability of repayment, reducing the institution's risk exposure.

1.3: LOADING NECESSARY LIBRARIES

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

1.4: LOADING THE DATA

The loan approval dataset is a collection of financial records and associated information used to determine the eligibility of individuals or organizations for obtaining loans from a lending institution. It includes various factors such as cibil score, income, employment status, loan term, loan amount, assets value, and loan status.

The dependent variable or target variable is the Loan_Status, while the rest are independent variable or features.

In [5]:

```
loan_df = pd.read_csv('loan_approval_dataset.csv')
```

In [6]:

```
loan_df.head()
```

Out[6]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value	loan_status
0	1	2	Graduate	No	9600000	29900000	12	778	2400000	17600000	22700000	8000000	Approved
1	2	0	Not Graduate	Yes	4100000	12200000	8	417	2700000	2200000	8800000	3300000	Rejected
2	3	3	Graduate	No	9100000	29700000	20	506	7100000	4500000	33300000	12800000	Rejected
3	4	3	Graduate	No	8200000	30700000	8	467	18200000	3300000	23300000	7900000	Rejected
4	5	5	Not Graduate	Yes	9800000	24200000	20	382	12400000	8200000	29400000	5000000	Rejected

2. DESCRIPTIVE STATISTICS

To understand its structure, including the number of records, columns, and data types.

In [7]:

```
loan_df.shape
#(rows, columns)
```

Out[7]:

```
(4269, 13)
```

In [8]:

```
loan_df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   loan_id                    4269 non-null   int64 
 1    no_of_dependents          4269 non-null   int64 
 2    education                 4269 non-null   object
 3    self_employed             4269 non-null   object
 4    income_annum              4269 non-null   int64 
 5    loan_amount               4269 non-null   int64 
 6    loan_term                 4269 non-null   int64 
 7    cibil_score               4269 non-null   int64 
 8    residential_assets_value  4269 non-null   int64 
 9    commercial_assets_value   4269 non-null   int64 
 10   luxury_assets_value       4269 non-null   int64 
 11   bank_asset_value          4269 non-null   int64 
 12   loan_status               4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
```

There are 2 data types in the data

1. object: Object format means variables are categorical. Categorical variables in our dataset are:Education, Self_Employed,Loan_Status
2. nt64: It represents the integer variables.The rest of the columns are in this format.

Datatypes are an important concept because statistical methods can only be used with certain data types.

We will have to convert the catergorical variables into numerical format for statistiscal analysis.

In [9]:

```
loan_df.describe()
```

Out[9]:

	loan_id	no_of_dependents	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value
count	4269.000000	4269.000000	4.269000e+03	4.269000e+03	4269.000000	4269.000000	4.269000e+03	4.269000e+03	4.269000e+03	4.269000e+03
mean	2135.000000	2.498712	5.059124e+06	1.513345e+07	10.900445	599.936051	7.472617e+06	4.973155e+06	1.512631e+07	4.976692e+06
std	1232.498479	1.695910	2.806840e+06	9.043363e+06	5.709187	172.430401	6.503637e+06	4.388966e+06	9.103754e+06	3.250185e+06
min	1.000000	0.000000	2.000000e+05	3.000000e+05	2.000000	300.000000	-1.000000e+05	0.000000e+00	3.000000e+05	0.000000e+00
25%	1068.000000	1.000000	2.700000e+06	7.700000e+06	6.000000	453.000000	2.200000e+06	1.300000e+06	7.500000e+06	2.300000e+06
50%	2135.000000	3.000000	5.100000e+06	1.450000e+07	10.000000	600.000000	5.600000e+06	3.700000e+06	1.460000e+07	4.600000e+06
75%	3202.000000	4.000000	7.500000e+06	2.150000e+07	16.000000	748.000000	1.130000e+07	7.600000e+06	2.170000e+07	7.100000e+06
max	4269.000000	5.000000	9.900000e+06	3.950000e+07	20.000000	900.000000	2.910000e+07	1.940000e+07	3.920000e+07	1.470000e+07

2.1: EXPLORATORY DATA ANALYSIS(EDA)

understanding the dataset's underlying patterns, relationships, and potential anomalies before diving into modeling or making predictions. EDA typically involves visualizing data, identifying patterns, and uncovering important insights.

In [10]:

```
loan_df.isnull().any()
#checking for null or missing values
```

Out[10]:

	0
loan_id	False
no_of_dependents	False
education	False
self_employed	False
income_annum	False

	0
loan_amount	False
loan_term	False
cibil_score	False
residential_assets_value	False
commercial_assets_value	False
luxury_assets_value	False
bank_asset_value	False
loan_status	False

dtype: bool

In [11]:

```
loan_df.isnull().sum()
```

Out[11]:

	0
loan_id	0
no_of_dependents	0
education	0
self_employed	0
income_annum	0
loan_amount	0
loan_term	0
cibil_score	0
residential_assets_value	0
commercial_assets_value	0
luxury_assets_value	0
bank_asset_value	0
loan_status	0

dtype: int64

In [12]:

```
loan_df.duplicated().sum()
#checks for duplicate rows in the DataFrame loan_df
```

Out[12]:

```
0
```

In [13]:

```
print(loan_df.columns)
Index(['loan_id', ' no_of_dependents', ' education', ' self_employed',
       ' income_annum', ' loan_amount', ' loan_term', ' cibil_score',
       ' residential_assets_value', ' commercial_assets_value',
       ' luxury_assets_value', ' bank_asset_value', ' loan_status'],
      dtype='object')
```

In [14]:

```
loan_df[' loan_status'].unique()
# checking for unique values in the loan_status column
```

Out[14]:

```
array([' Approved', ' Rejected'], dtype=object)
```

In [15]:

```
## Group by loan_status and review the output.
loan_gr = loan_df.groupby(' loan_status' , axis = 0)
pd.DataFrame(loan_gr.size(), columns=['# of observations'])
```

Out[15]:

	# of observations
loan_status	
Approved	2656
Rejected	1613

Those approved are 2656 and those rejected are 1613. Lets visualize this to confirm.

2.2: VISUALIZATION

** Univariate analysis**

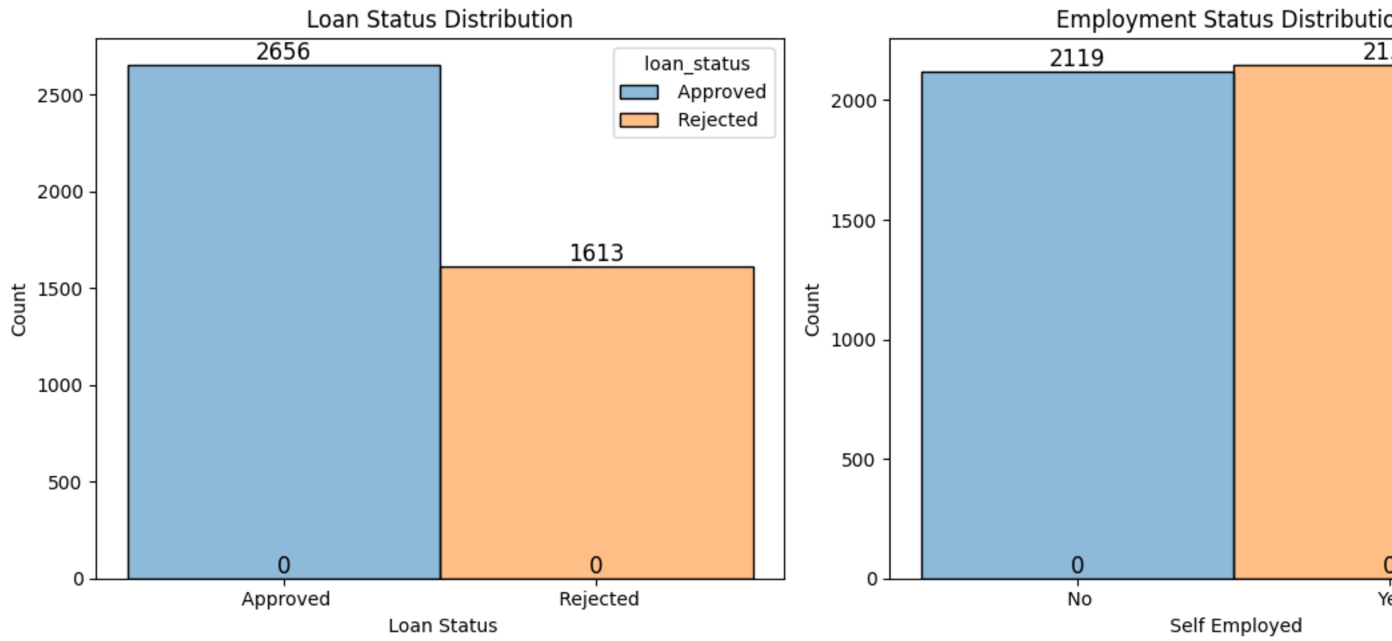
Univariate analysis is when we analyze each variable individually. For categorical features we can use frequency table which will calculate the number of each category in a particular variable. For numerical features, a histogram is used.

1. VISUALIZING CATEGORICAL FEATURES

In [16]:

```
fig, axs = plt.subplots(1, 3, figsize=(18, 5))  # Create 1 row and 3 
columns of subplots

# List of features to plot
features = [' loan_status', ' self_employed' , ' education' ]
titles = ['Loan Status Distribution', 'Employment Status Distribution', 
'Education Status Distribution']

# Loop through the features to create plots and add data labels
for ax, feature, title in zip(axs, features, titles):
    sns.histplot(data=loan_df, x=feature, hue=feature, ax=ax, kde=False)
    ax.set_title(title)
    ax.set_xlabel(feature.replace('_', ' ').title())
    ax.set_ylabel('Count')

    # Add data labels
    for container in ax.containers:
```

```
        ax.bar_label(container, fontsize=12)

# Adjust layout for better spacing
plt.tight_layout()
plt.show()

## hue: This will color the bars in the histogram based on different
categories
#kde=False: No density curve will be plotted alongside the histogram bars
```



INTERPRETATION:

2150 people are selfemployed while 2119 are not self_employed. There's a roughly equal distribution between self-employed and non-self-employed applicants, meaning employment type is fairly balanced in the dataset.

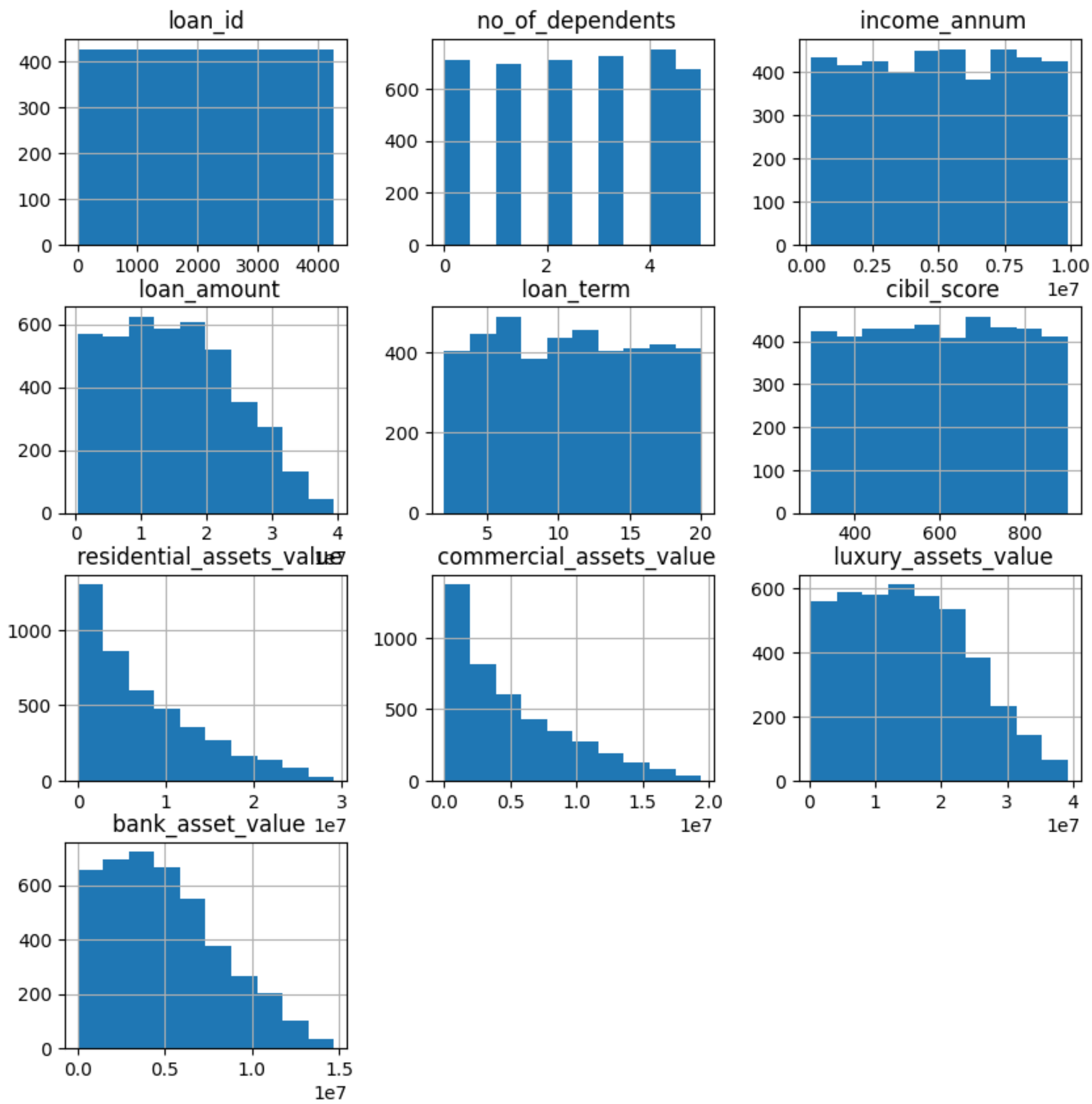
2144 people are graduates and 2125 people are not graduates. the education status is also nearly equally distributed, with a slight majority of applicants having graduated.

Those approved are 2656 and those rejected are 1613. There are significantly more loans approved than rejected in this dataset.

2. VISUALIZING NUMERICAL FEATURES

In [17]:

```
#Histogram to visualize the numerical features- to show the distribution of
the data
loan_df.hist(figsize=(10,10))
plt.show()
```



BIVARIATE ANALYSIS-

Refers to analysis of two variables to understand the relationship or association between them

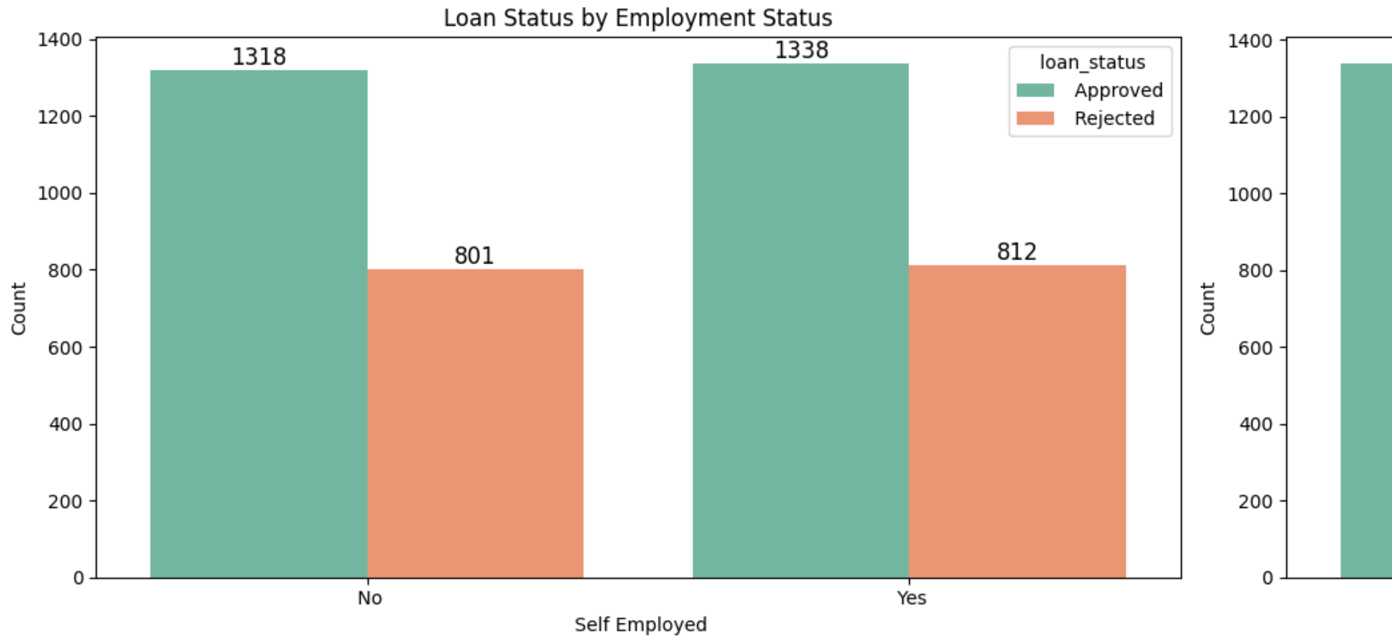
1. Categorical Independent Variable vs Target Variable

BAR PLOTS TO VISUALIZE LOAN STATUS BY EMPLOYMENT AND EDUCATION STATUS

In [18]:

```
fig, axs = plt.subplots(1, 2, figsize=(18, 5))  # Create 1 row and 3
columns of subplots

# List of features to plot
features = [' self_employed', ' education']
titles = ['Loan Status by Employment Status', 'Loan Status by Education
Status' ]
```

```
# Loop through the features to create plots and add data labels
for ax, feature, title in zip(axs, features, titles):
    sns.countplot(data=loan_df, x=feature, hue=' loan_status', ax=ax,
palette='Set2' )
    ax.set_title(title)
    ax.set_xlabel(feature.replace('_', ' ').title())
    ax.set_ylabel('Count')
#The palette parameter in seaborn controls the colors used for the
different elements (bars,

    # Add data labels
    for container in ax.containers:
        ax.bar_label(container, fontsize=12)

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```



INTERPRETATION:

1338 self_employed people were approved for a loan.
812 self_employed people were rejected for a loan.

1318 non self_employed people were approved for a loan.

801 non self_employed people were rejected for a loan

The number of loan approvals is consistently higher than the rejections for both self-employed and non-self-employed individuals. Being self-employed does not appear to significantly impact the likelihood of loan approval compared to not being self-employed.

==

1339 Graduates were approved for a loan. 805 Graduates were rejected for a loan.

1317 Non-Graduates were approved for a loan.

808 Non_Graduates were rejected for a loan

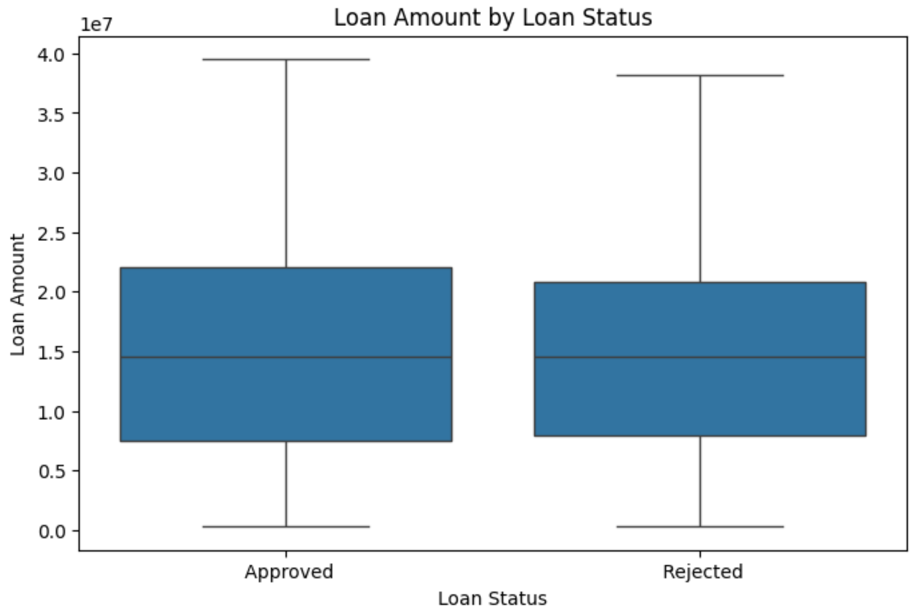
The pattern of loan approval is similar for both graduates and non-graduates, with more approvals than rejections in both categories. Education status (being a graduate or not) seems to have a minimal impact on the loan approval rate.

In both charts, the counts of loan approvals are consistently higher than rejections across all categories (employment and education status), suggesting that these factors may not be the primary determinants of loan approval decisions in this dataset.

2. **Numerical Independent Variable VS target variable**

In [19]:

```
# Box plot to analyze the relationship between loan_status and loan_amount
plt.figure(figsize=(8, 5))
sns.boxplot(x=' loan_status', y= ' loan_amount', data=loan_df)

# Add title and labels
plt.title('Loan Amount by Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Loan Amount')

plt.show()
```



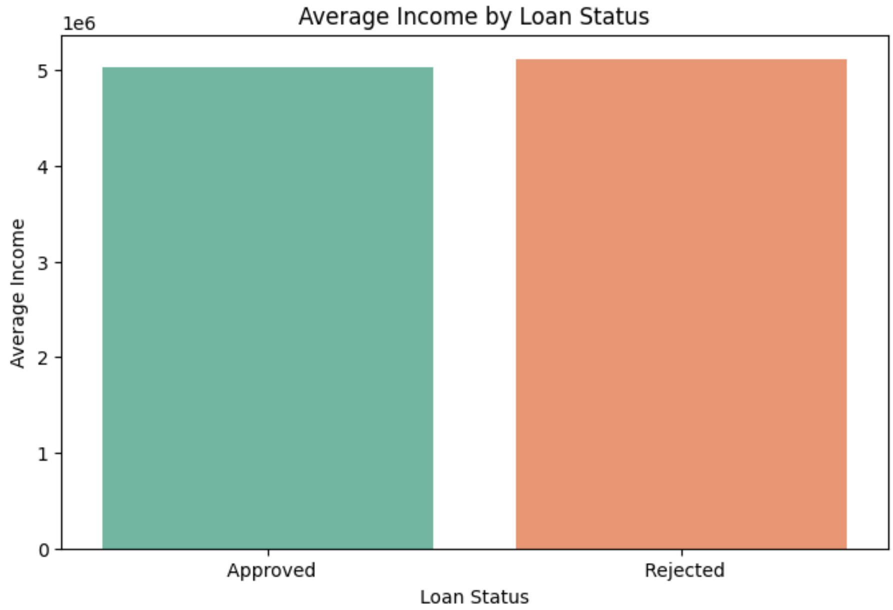
INTERPRETATION

1. The median for both Approved and Rejected loans appears similar, each 1.5Million
2. The box represents the middle 50% of the loan amounts (the range from the 25th to the 75th percentile).

Both boxes have similar sizes, indicating that the variability of loan amounts is quite similar for both Approved and Rejected loans. 3. Both categories have whiskers extending upwards, indicating that there are some higher loan amounts beyond the IQR. 4. The lower whisker is almost touching 0 for both categories, meaning there are very small loan amounts present as well. 5. There do not appear to be any extreme outliers as no individual points are plotted outside the whiskers. This indicates that most of the loan amounts are within a reasonable range. 6. Since the median and spread of loan amounts for both categories are similar, it suggests that loan amount alone may not be a strong differentiating factor in determining loan approval status in this dataset.

In [20]:

```
#bar chart to analyze the relationship between loan_status and average
income
# Group by loan_status and calculate the mean of income
avg_income_by_status = loan_df.groupby(' loan_status')['
income_annum'].mean().reset_index()

# Plot a bar chart
plt.figure(figsize=(8, 5))
sns.barplot(x=' loan_status', y= ' income_annum', hue=' loan_status',
data=avg_income_by_status, palette='Set2')

# Add title and labels
plt.title('Average Income by Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Average Income')

# Show the plot
plt.show()
```

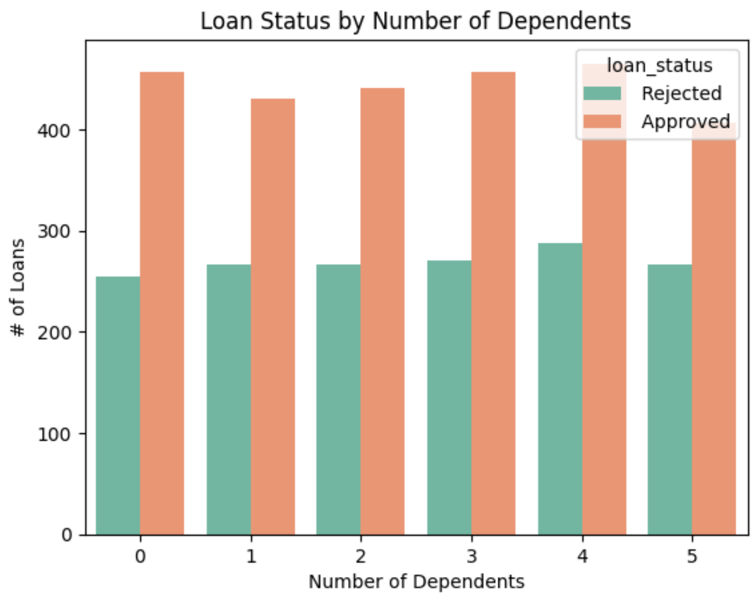


INTERPRETATION

1. Both bars are of nearly the same height, suggesting that there is little to no significant difference in the average income between those who had their loans approved and those who had them rejected. The average income is around 5 million units for both groups.
2. This implies that income may not be the primary factor influencing the approval or rejection of loans in this dataset.

In [21]:

```
#bar chart to analyze the relationshhip between no of dependents and
loan_status
# Create a count plot with 'dependents' on the x-axis and 'loan_status' as
hue
sns.countplot(x=' no_of_dependents', hue=' loan_status', data=loan_df,
palette='Set2')

# Add title and labels
plt.title('Loan Status by Number of Dependents')
plt.xlabel('Number of Dependents')
plt.ylabel('# of Loans')

# Show the plot
plt.show()
```



INTERPRETATION

1. For every category of dependents (0 to 5), the number of approved loans (in orange) is consistently higher than the number of rejected loans (in green). This suggests that, regardless of the number of dependents, the likelihood of loan approval is generally higher than rejection.
2. No Dependents (0): The number of approved loans is much higher compared to rejected loans, indicating a higher approval rate for individuals with no dependents.
3. 1, 2, 3, 4, and 5 Dependents: A similar pattern can be observed, where approved loans outnumber rejected loans, though the gap between approved and rejected loans narrows slightly as the number of dependents increases.
4. The number of loans in each category (whether approved or rejected) appears to be fairly consistent across different dependent categories, suggesting that the number of dependents may not be a very strong factor influencing loan approval in this dataset.
5. From this chart, it can be inferred that the number of dependents does not have a dramatic effect on loan approval, but there is a slight decrease in approval rates as the number of dependents increases.

3. **correlation between all the numerical variables** -

The variables with darker color means their correlation is more.

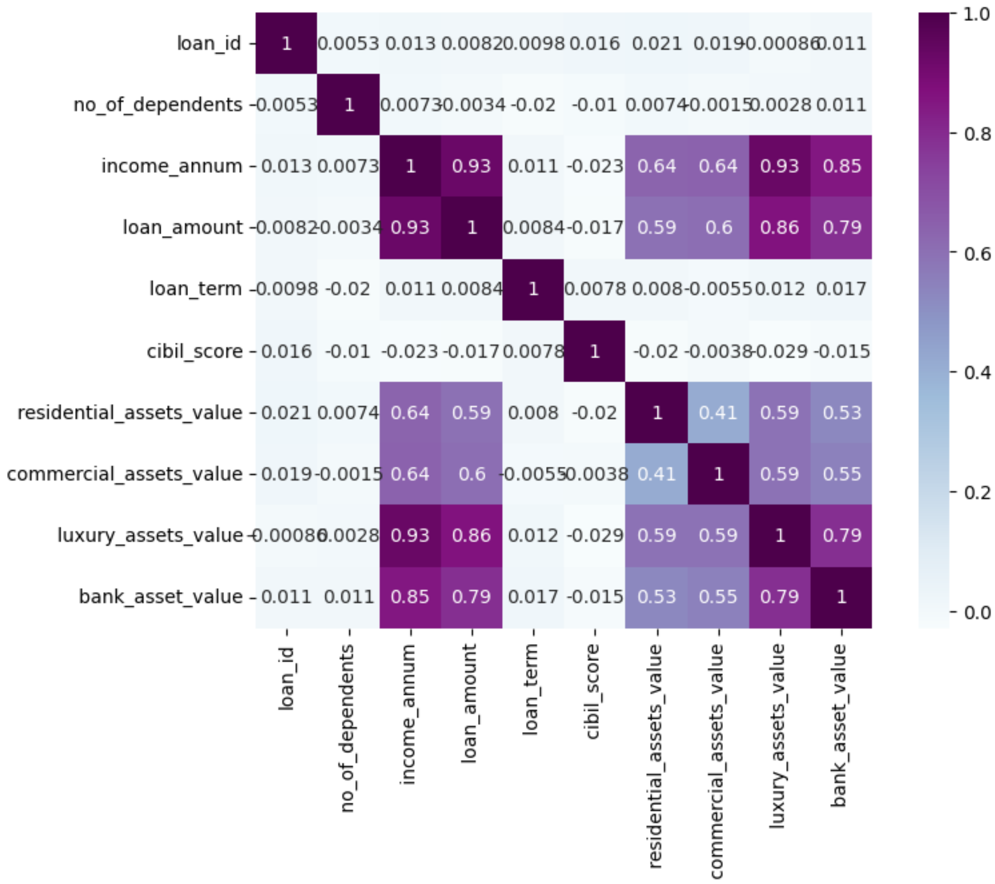
In [22]:

```
#correlation between all the numerical variables
# calculate and visualize correlation matrix
correlation_matrix = loan_df.drop(columns=[' education', ' self_employed',' loan_status']).corr()
f, ax = plt.subplots(figsize=(9, 6))
sns.heatmap(correlation_matrix, vmax=1, square=True, cmap="BuPu", annot=True)
```

```
correlation_matrix
```

Out[22]:

	loan_id	no_of_dependents	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value
loan_id	1.000000	0.005326	0.012592	0.008170	0.009809	0.016323	0.020936	0.018595	-0.000862	0.010765
no_of_dependents	0.005326	1.000000	0.007266	-0.003366	-0.020111	-0.009998	0.007376	-0.001531	0.002817	0.011163
income_annum	0.012592	0.007266	1.000000	0.927470	0.011488	-0.023034	0.636841	0.640328	0.929145	0.851093
loan_amount	0.008170	-0.003366	0.927470	1.000000	0.008437	-0.017035	0.594596	0.603188	0.860914	0.788122
loan_term	0.009809	-0.020111	0.011488	0.008437	1.000000	0.007810	0.008016	-0.005478	0.012490	0.017177
cibil_score	0.016323	-0.009998	-0.023034	-0.017035	0.007810	1.000000	-0.019947	-0.003769	-0.028618	-0.015478
residential_assets_value	0.020936	0.007376	0.636841	0.594596	0.008016	-0.019947	1.000000	0.414786	0.590932	0.527418
commercial_assets_value	0.018595	-0.001531	0.640328	0.603188	-0.005478	-0.003769	0.414786	1.000000	0.591128	0.548576
luxury_assets_value	-0.000862	0.002817	0.929145	0.860914	0.012490	-0.028618	0.590932	0.591128	1.000000	0.788517
bank_asset_value	0.010765	0.011163	0.851093	0.788122	0.017177	-0.015478	0.527418	0.548576	0.788517	1.000000



INTERPRETATION:

Strong Positive Correlations:

1. income_annum and loan_amount: The correlation is 0.93, indicating a strong positive relationship. This suggests that as the annual income increases, the loan amount tends to increase as well.
2. income_annum and luxury_assets_value: A high correlation of 0.93 indicates that individuals with higher income are more likely to have higher values of luxury assets.
3. luxury_assets_value and loan_amount: The correlation is 0.86, showing a strong relationship between the value of luxury assets and the loan amount.
4. income_annum and bank_asset_value: With a correlation of 0.85, this suggests that higher income is associated with higher bank asset values.

Weak Correlations:

1. Most variables like no_of_dependents, loan_term, and cibil_score have very low correlations (close to zero) with other variables, suggesting that they do not have a strong direct relationship with the other factors in this dataset.

The matrix shows that financial factors (income, asset values, and loan amount) have relatively strong interconnections, suggesting that individuals with higher incomes are more likely to have higher asset values and take larger loan amounts.

3. DATA PREPROCESSING

In [23]:

```
print(loan_df.columns)
Index(['loan_id', ' no_of_dependents', ' education', ' self_employed',
       ' income_annum', ' loan_amount', ' loan_term', ' cibil_score',
       ' residential_assets_value', ' commercial_assets_value',
       ' luxury_assets_value', ' bank_asset_value', ' loan_status'],
      dtype='object')
```

In [24]:

```
#Droping the loan_id since its redundant and not useful
loan_df.drop('loan_id', axis=1, inplace=True)
#inplace=True: Directly applies the change to the DataFrame without needing to reassign it.
```

In [25]:

```
loan_df.head()
```

Out[25]:

	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value	loan_status
0	2	Graduate	No	9600000	29900000	12	778	2400000	17600000	22700000	8000000	Approved
1	0	Not Graduate	Yes	4100000	12200000	8	417	2700000	2200000	8800000	3300000	Rejected
2	3	Graduate	No	9100000	29700000	20	506	7100000	4500000	33300000	12800000	Rejected
3	3	Graduate	No	8200000	30700000	8	467	18200000	3300000	23300000	7900000	Rejected
4	5	Not Graduate	Yes	9800000	24200000	20	382	12400000	8200000	29400000	5000000	Rejected

In [26]:

```
loan_df.shape
```

Out[26]:

```
(4269, 12)
```

In [27]:

```
print(loan_df.columns)
```

```
Index([' no_of_dependents', ' education', ' self_employed', '
income_annum',
       ' loan_amount', ' loan_term', ' cibil_score',
       ' residential_assets_value', ' commercial_assets_value',
       ' luxury_assets_value', ' bank_asset_value', ' loan_status'],
      dtype='object')
```

In [28]:

```
X = loan_df.drop(' loan_status', axis=1) #feature variables
y = loan_df[' loan_status'] #target variables
```

In [29]:

```
#converting categorical variables to numerical format
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

categorical_columns = [' education', ' self_employed' ]

for column in categorical_columns:
    X[column] = le.fit_transform(X[column])

y = le.fit_transform(y)
```

In [30]:

```
y
```

Out[30]:

```
array([0, 1, 1, ..., 1, 0, 0])
```

In [31]:

```
X[' education' ]
```

Out[31]:

	education
0	0
1	1
2	0
3	0
4	1
...	...
4264	0
4265	1
4266	1
4267	1
4268	0

4269 rows × 1 columns

dtype: int64

FEATURE STANDADIZATION

In [32]:

```
from sklearn.preprocessing import StandardScaler

scaler =StandardScaler()
X = scaler.fit_transform(X)
```

SPLITTING THE DATA INTO TRAINING AND TESTING DATASETS

In [33]:

```
from sklearn.model_selection import train_test_split

##Split data set in train 80% and test 20%
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, 
random_state=7)
#Using the same random_state value will produce the same split each time 
you run the code.
```

In [34]:

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
#The shapes are represented as tuples (number of samples, number ofn 
features) for X and (number of samples,) for y.
```

Out[34]:

```
((3415, 11), (3415,), (854, 11), (854,))
```

4. MODEL TRAINING

In [56]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[56]:

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

LogisticRegression?Documentation for LogisticRegressioniFitted

```
LogisticRegression()
```

5. MODEL EVALUATION

In [57]:

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

# Evaluate the model
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the metrics
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-Score: {f1:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
Accuracy: 0.92
Precision: 0.91
Recall: 0.88
F1-Score: 0.89
Confusion Matrix:
[[491  30]
 [ 39 294]]
```

So our predictions are over 92% accurate, i.e. we have identified 92% of the loan status correctly.

The performance of our model seems encouraging, with accuracy of 92%, precision of 91% and recall of 88%.

In [37]:

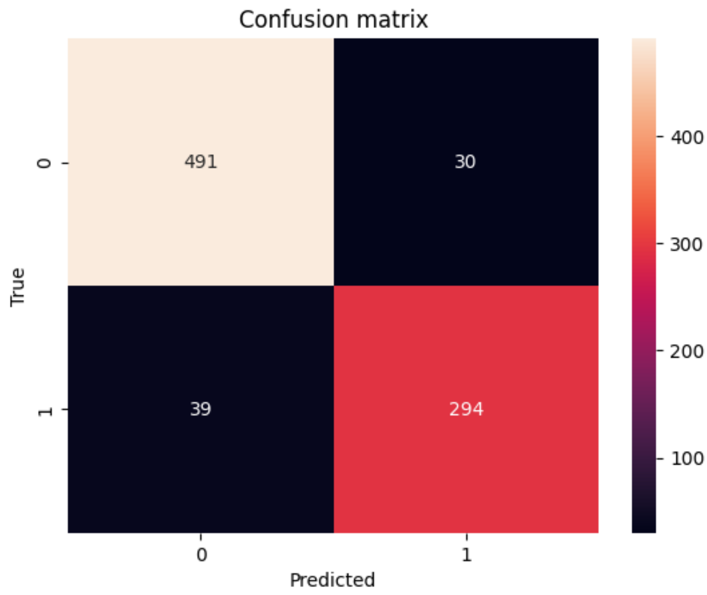
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       521
           1       0.91      0.88      0.89       333

    accuracy                           0.92       854
   macro avg       0.92      0.91      0.91       854
weighted avg       0.92      0.92      0.92       854
```

In [40]:

```
#heatmap to visualize the confusion matrix
import seaborn as sns
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='g')
plt.title('Confusion matrix')
plt.xlabel('Predicted')
plt.ylabel('True') #actual
```

Out[40]:

```
Text(50.722222222222214, 0.5, 'True')
```



Interpretation:

1. Rows represent the true values (actual labels).
2. Columns represent the predicted values from the model.
3. True Positives (TP): The model correctly predicted 294 instances of the positive class (1).
4. True Negatives (TN): The model correctly predicted 491 instances of the negative class (0).
5. False Positives (FP): The model incorrectly predicted 30 instances as positive when they were actually negative.
6. False Negatives (FN): The model incorrectly predicted 39 instances as negative when they were actually positive.

6. Grid search

We will try to improve the accuracy by tuning the hyperparameters for this model. We will use grid search to get the optimized values of hyper parameters. GridSearch is a way to select the best of a family of hyper parameters, parametrized by a grid of parameters.

Grid Search - This is an exhaustive search method that helps you automatically find the best combination of hyperparameters for your machine learning model. Instead of manually trying out different values for hyperparameters, GridSearchCV does it for you by testing all possible combinations of the specified hyperparameter values and selecting the best one based on the scoring metric.

In [1]:

```
from sklearn.model_selection import GridSearchCV
```

In [44]:

```
model = LogisticRegression(max_iter=1000)

# Define the hyperparameters and their values to search
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],      # Inverse of regularization strength
    'solver': ['liblinear', 'saga'],  # Optimization algorithm
    'penalty': ['l1', 'l2'],           # Regularization type
}

#l1 and l2 correspond to Lasso and Ridge regularization, respectively.
#The grid search will try all possible combinations of these
hyperparameters (5 values of C, 2 solvers, and 2 penalties), meaning a
total of 20 combinations will be tested.
```

In [45]:

```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           scoring='accuracy', cv=5, n_jobs=-1)

#cv=5: This specifies 5-fold cross-validation, meaning the data will be
split into 5 subsets.
#The model will be trained on 4 of the subsets and tested on the remaining
one.
#This will repeat 5 times, rotating the test set each time, and the results
will be averaged.
```

In [46]:

```
grid_search.fit(X_train, y_train)
```

Out[46]:

```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1',
'l2'],
                         'solver': ['liblinear', 'saga']},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

GridSearchCV[?Documentation for GridSearchCV]iFitted

```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1',
'l2'],
                         'solver': ['liblinear', 'saga']},
             scoring='accuracy')
```

best_estimator_: LogisticRegression

```
LogisticRegression(C=0.01, max_iter=1000, penalty='l1', solver='liblinear')
```

LogisticRegression[?Documentation for LogisticRegression]

```
LogisticRegression(C=0.01, max_iter=1000, penalty='l1', solver='liblinear')
```

In [47]:

```
print(f'Best Parameters: {grid_search.best_params_}') #This returns the
best hyperparameter combination found during the grid search
```

```
print(f'Best Cross-Validation Score: {grid_search.best_score_:.2f}')  #This gives the highest accuracy score obtained during cross-validation with the best combination of hyperparameters.
Best Parameters: {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
Best Cross-Validation Score: 0.94
```

The initial accuracy of 92% and the current best cross-validation score of 94% indicate that the hyperparameter tuning has improved the model's performance.

EVALUATE THE MODEL ON THE TEST SET

In [54]:

```
best_model = grid_search.best_estimator_

# Predict on the test data
y_pred = best_model.predict(X_test)

# Evaluate the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the metrics
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-Score: {f1:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
Accuracy: 0.94
Precision: 0.90
Recall: 0.96
F1-Score: 0.93
Confusion Matrix:
[[485  36]
 [ 12 321]]
```

This are the Metrics before:

Accuracy: 0.92

Precision: 0.91

Recall: 0.88

F1-Score: 0.89

Confusion Matrix: [[491 30] [39 294]]

==

And these are the metrics after: Accuracy: 0.94

Precision: 0.90

Recall: 0.96

F1-Score: 0.93

Confusion Matrix: [[485 36] [12 321]]

INTERPRETATION:

There's a slight improvement in accuracy from 92% to 94%. This means the tuned model is making fewer incorrect predictions overall.

Significant improvement in recall shows that the tuned model is much better at correctly identifying loans that should be approved.

F1-score has improved, meaning the model's balance between precision and recall is stronger after tuning.

In conclusion, tuning has made the model much better at identifying approved loans (true positives) with minimal trade-offs in precision. This suggests the tuned model is more effective overall.

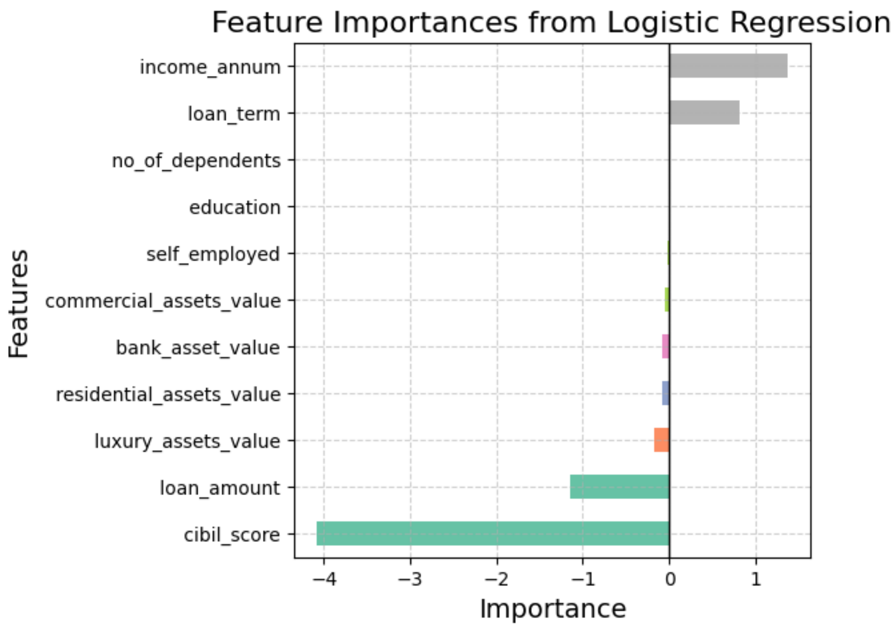
7. FEATURE IMPORTANCE

Let us find the feature importance now, i.e. which features are most important for this problem. We will use feature_importances_ attribute of sklearn to do so. It will return the feature importances (the higher, the more important the feature).

In [68]:

```
# Assuming 'importances' is a pandas Series with feature importances
importances = pd.Series(model.coef_[0], index=loan_df.drop(' loan_status',
axis=1).columns)

# Set a color map or define specific colors
colors = plt.cm.get_cmap('Set2', len(importances))

# Create a horizontal bar chart with different colors
plt.figure(figsize=(5, 5))
```

```
importances.sort_values().plot(kind='barh', color=[colors(i) for i in
np.arange(len(importances))])

# Add a vertical line at x=0 for reference (to differentiate
positive/negative impacts)
plt.axvline(x=0, color='black', linewidth=1)

# Set the titles and labels
plt.title('Feature Importances from Logistic Regression', fontsize=16)
plt.xlabel('Importance', fontsize=14)
plt.ylabel('Features', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)

# Show the plot
plt.show()
```



INTERPRETATION

CIBIL score : has the largest negative coefficient, indicating a strong negative impact on the likelihood of loan approval.

loan_amount: This feature also has a negative coefficient, suggesting that higher loan amounts decrease the probability of loan approval.

income_annum and loan_term: Both features have positive coefficients, implying that higher annual income and longer loan terms increase the likelihood of loan approval.

The other features contribute little to the model's predictions, suggesting that they do not significantly affect the outcome of loan approval in this context.

8. SAVING THE MODEL

In [69]:

```
import joblib

# Save the model
joblib.dump(model, 'loan_approval_model.pkl') #This is useful for
preserving the model after training so it can be reused without retraining
it.

# Load the model
loaded_model = joblib.load('loan_approval_model.pkl')
```

In [70]:

```
from google.colab import files

# Download the model file to your local computer
files.download('loan_approval_model.pkl')
```

CONCLUSION

The loan approval prediction model was built using logistic regression to classify whether a loan application would be approved or rejected based on key applicant features such as education status, income, employment status, and loan amount. After hyperparameter tuning, the model's performance improved, with the accuracy increasing from 92% to 94%.

BUSINESS IMPLICATION

1. The model can serve as a valuable tool for financial institutions to streamline the loan approval process. With high recall, the model ensures that most eligible applicants are not rejected, while precision remains high enough to minimize the approval of unqualified applicants.
2. By understanding the factors that influence approval decisions, institutions can adjust their criteria or provide tailored advice to applicants, improving the overall quality of loan portfolios.

SUGGESTIONS FOR IMPROVEMEMT

1. Further feature engineering could be explored, adding more nuanced features (e.g., applicant's debt-to-income ratio) that might improve prediction performance.
2. The model could be integrated into a live system for real-time loan approval decisions or used to enhance existing approval processes.
3. Regular retraining and validation with new data should be conducted to ensure the model stays accurate as economic conditions and lending criteria evolve.