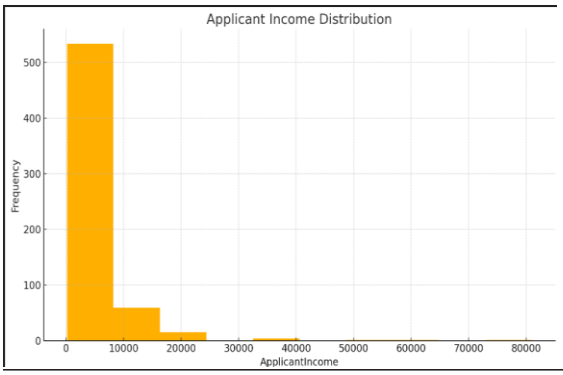
Data Collection and Preprocessing Phase

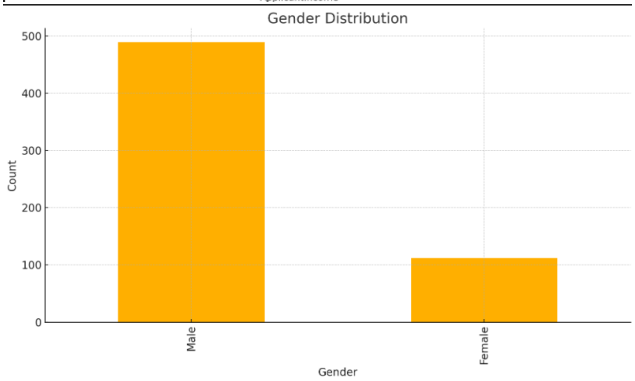
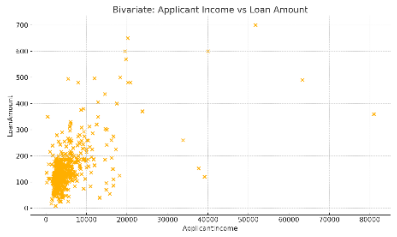
Date	05 July 2025
Team ID	SWTID1749835773
Project Title	Applicant Credibility Prediction For Loan Approval
Maximum Marks	6 Marks

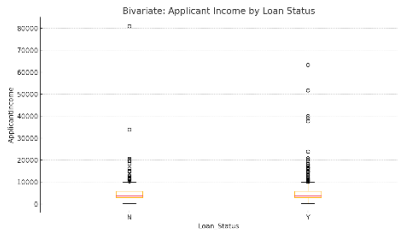
Data Exploration and Preprocessing Report

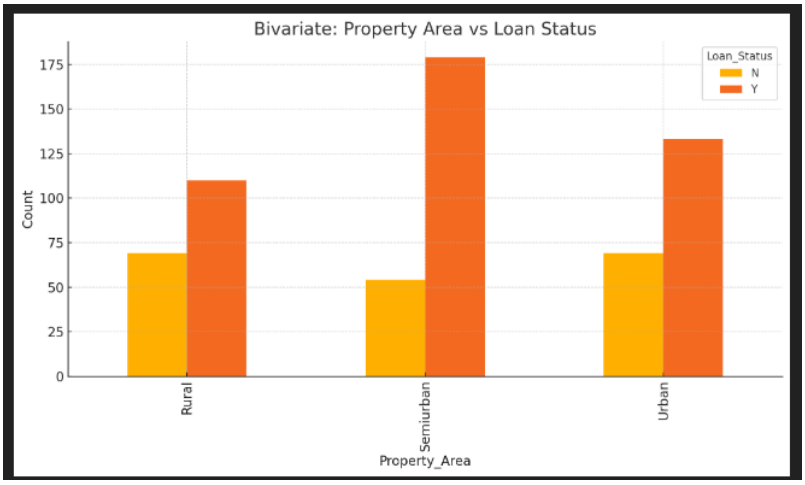
The dataset contains loan-application information (demographics, incomes, loan details, credit history, etc.). Below we explore structure, spot anomalies, and document every cleaning/feature-engineering step carried out in datapreprocessing.ipynb, providing a robust foundation for subsequent modelling.

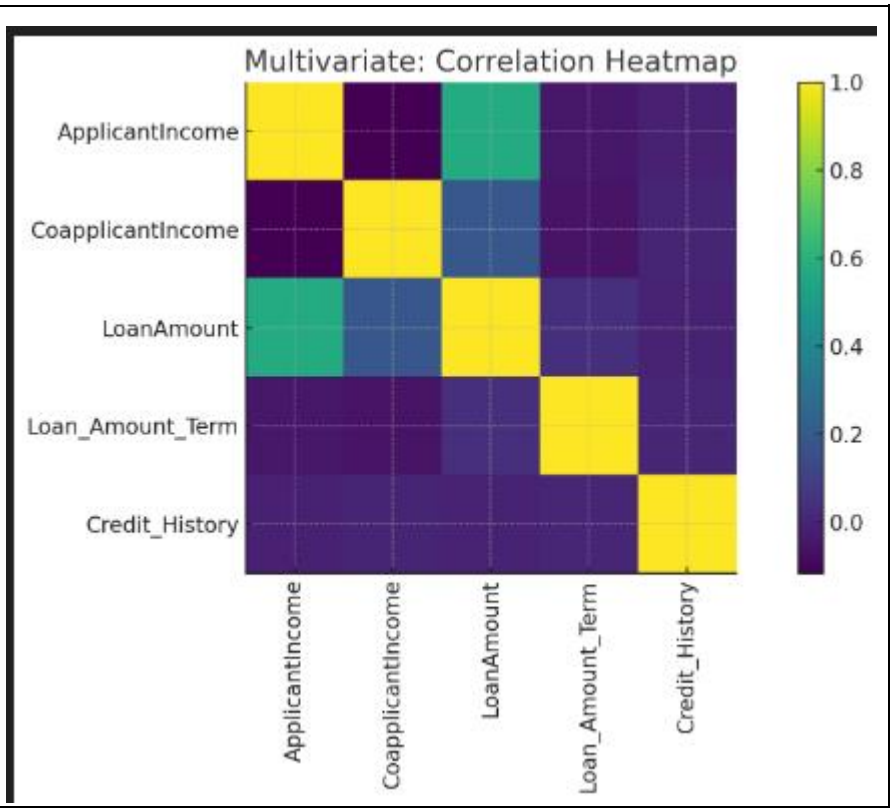
Section	Description
Data Overview	Dimension: 614 rows × 13 columns Descriptive statistics: Loan_ID Gender Married Dependents Education Self_Employed 0 LP001002 Male No 0 Graduate No 1 LP001003 Male Yes 1 Graduate No 2 LP001005 Male Yes 0 Graduate Yes 3 LP001006 Male Yes 0 Not Graduate No 4 LP001008 Male No 0 Graduate No
Univariate Analysis	

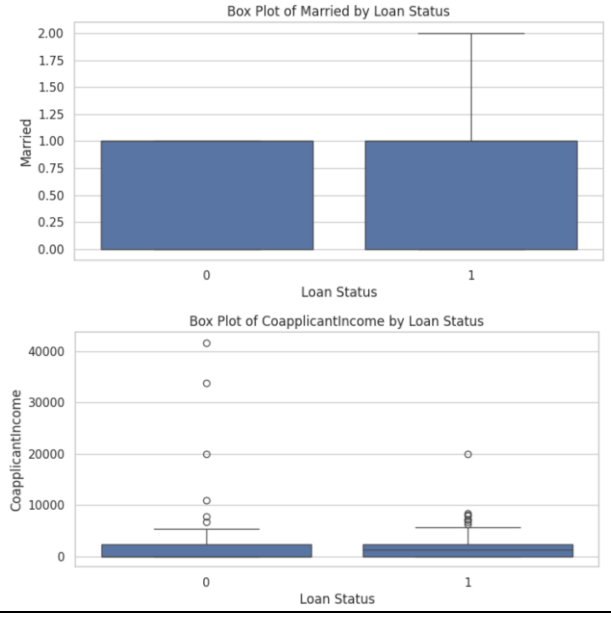
<table>
<tr>
<td></td>
<td> 

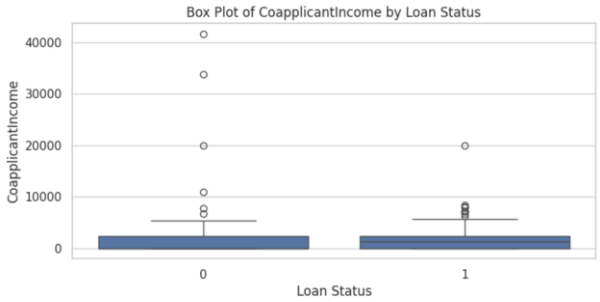

</td>
</tr>
<tr>
<td>Bivariate Analysis</td>
<td> 




</td>
</tr>
</table>

Multivariate Analysis	

Outliers and Anomalies	



Data Preprocessing Code Screenshots

Loading Data

```
# Step 1: Load Dataset
df = pd.read_csv("C:\\Users\\barsn\\Downloads\\loan_prediction.csv")
print("Dataset Shape:", df.shape)
df.head()
```

[2] ✓ 0.1s

```
Dataset Shape: (614, 13)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
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	

Handling Missing Data

```
# Step 3: Handling Null Values (Fill or Drop)

# Filling categorical nulls with mode
cat_cols = df.select_dtypes(include='object').columns
for col in cat_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)

# Filling numeric nulls with median
num_cols = df.select_dtypes(include=np.number).columns
for col in num_cols:
    df[col].fillna(df[col].median(), inplace=True)

# Re-checking nulls
df.isnull().sum()
```

```
Loan_ID              0
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
```

Data Transformation

```
# Identify categorical columns (object or category dtype)
categorical_cols = df.select_dtypes(include=['object', 'category']).columns

# Initialize label encoders dictionary
label_encoders = {}

# Apply label encoding only to categorical columns
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Drop the first column (loan id) as it is not useful for prediction
df.drop(df.columns[0], axis=1, inplace=True)

df.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
0	1	0	0	0	0	5849	0.0	NaN	360.0	1.0	
1	1	1	1	0	0	4583	1508.0	128.0	360.0	1.0	
2	1	1	0	0	1	3000	0.0	66.0	360.0	1.0	
3	1	1	0	1	0	2583	2358.0	120.0	360.0	1.0	
4	1	0	0	0	0	6000	0.0	141.0	360.0	1.0	

Feature Engineering

```
# Load Dataset
df = pd.read_csv('/content/loan_prediction.csv')
print("Dataset Shape:", df.shape)
df.head()
# Checking For Null Values
print("Missing values:\n", df.isnull().sum())
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
# Identify categorical columns (object or category dtype)
categorical_cols = df.select_dtypes(include=['object', 'category']).columns

# Initialize label encoders dictionary
label_encoders = {}

# Apply label encoding only to categorical columns
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Drop the first column (loan id) as it is not useful for prediction
df.drop(df.columns[0], axis=1, inplace=True)

df.head()
# Handling Null Values (Fill or Drop)
# You can choose strategies like filling with mode/mean or dropping

# Filling categorical nulls with mode
cat_cols = df.select_dtypes(include='object').columns

for col in cat_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

```
# Filling numeric nulls with median
num_cols = df.select_dtypes(include=np.number).columns

for col in num_cols:
    df[col].fillna(df[col].median(), inplace=True)

# Re-checking nulls
df.isnull().sum()
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']
#selecting 5 best feature for training
from sklearn.feature_selection import SelectKBest , f_regression

selector = SelectKBest(score_func=f_regression, k=5)
x_new = selector.fit_transform(X, y)

mask= selector.get_support()
selected_feature = X.columns[mask]
print(selected_feature)
df[selected_feature]
# Step 5: Balancing The Dataset (If Target is Imbalanced)
# Let's assume target column is 'Loan_Status'
sns.countplot(x='Loan_Status', data=df)

# Apply SMOTE only if binary classification and imbalance exists
X = df[selected_feature]
y = df['Loan_Status']

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
print("Before SMOTE:", y.value_counts())
print("After SMOTE:", y_resampled.value_counts())
df.isnull().sum()
df_new= pd.DataFrame(X_resampled)
df_new['Loan_Status']= y_resampled
df_new.isnull().sum()
x= df_new[selected_feature]
y= df_new['Loan_Status']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
#parameter selection
param_grid = {
    "criterion": ["gini", "entropy"],
    "max_depth":np.arange(3,20)
}
model = DecisionTreeClassifier()
grid_search_dctree = GridSearchCV( model, param_grid, cv=5, verbose=1, n_jobs=-1)
grid_search_dctree.fit(x_train, y_train)
print(f"Best hyperparameters found by Grid Search: {grid_search_dctree.best_params_}")
model_opt = grid_search_dctree.best_estimator_
y_predict = model_opt.predict(x_test)
accuracy = accuracy_score(y_test, y_predict)
print(f"Accuracy: {accuracy}")
```

```
[ ] from sklearn.ensemble import RandomForestClassifier
```

```
# random forest classifier
df1= pd.read_csv("/content/loan_prediction.csv")



# Step 4: Handling Categorical Values using Label Encoding
label_encoders = {}
for col in df1.columns:
    le = LabelEncoder()
    df1[col] = le.fit_transform(df1[col])
    label_encoders[col] = le

#filling null values
num_cols = df1.select_dtypes(include=np.number).columns

for col in num_cols:
    df1[col].fillna(df1[col].median(), inplace=True)

#dividing into x and y
x = df1.drop('Loan_Status',axis=1)
x = x.drop('Loan_ID', axis=1)
y= df1["Loan_Status"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

#model making
single_tree= DecisionTreeClassifier(max_depth=5 , random_state=42)
single_tree.fit(x_train,y_train)


single_tree_predict= single_tree.predict(x_test)
```

```
param_grid = {'criterion': ['gini', 'entropy'],
    'n_estimators': [50, 100, 150],
    'max_depth':np.arange(0,21)
    }

rf_model = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    cv=5,                # 5-fold cross-validation
    n_jobs=-1,           # Use all CPU cores
    scoring='accuracy',  # Or 'roc_auc' for ROC-AUC
    verbose=2
)

grid_search.fit(x_train, y_train)

print(f"Best hyperparameters found by Grid Search: {grid_search.best_params_}")
best_rf_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_rf_model.predict(x_test)
best_rf_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_rf_model.predict(x_test)

accuracy_randomforest = accuracy_score(y_test, y_pred)
```

```
best_rf_model = grid_search.best_estimator_


# Make predictions on the test set
y_pred = best_rf_model.predict(x_test)

accuracy_randomforest = accuracy_score(y_test, y_pred)
print(classification_report(y_test,y_pred))
print(f1_score)
```

```
#KNN MODEL
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

df2= pd.read_csv("/content/loan_prediction.csv")

# Step 4: Handling Categorical Values using Label Encoding
label_encoders = {}
for col in df2.columns:
    le = LabelEncoder()
    df2[col] = le.fit_transform(df2[col])
    label_encoders[col] = le

#filling null values
num_cols = df2.select_dtypes(include=np.number).columns

for col in num_cols:
    df2[col].fillna(df2[col].median(), inplace=True)


x = df2.drop('Loan_ID',axis=1)
x = x.drop('Loan_Status' , axis=1)
y = df2['Loan_Status']
x
```

```
[ ] #split data
x_train , x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=42)


#Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
```

```
#parameter selection

param_grid = {
    'n_neighbors': np.arange(1, 11),          # Number of neighbors to try
    'weights': ['uniform', 'distance'],       # Weight function used in prediction
    'p': [1, 2]                               # Power parameter for Minkowski metric (1=Manhattan, 2=Euclidean)
}

model_knn = KNeighborsClassifier()
grid_search = GridSearchCV(model_knn, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)
grid_search.fit(X_train_scaled, y_train)

print(f"Best hyperparameters found by Grid Search: {grid_search.best_params_}")

model_knn_best= grid_search.best_estimator_
model_knn_best.fit(X_train_scaled, y_train)
y_pred = model_knn_best.predict(X_test_scaled) # Use scaled data for prediction
```

```
[ ] #xgboost

import numpy as np
from xgboost import XGBClassifier

df2= pd.read_csv("/content/loan_prediction.csv")

# Step 4: Handling Categorical Values using Label Encoding
label_encoders = {}
for col in df2.columns:
    le = LabelEncoder()
    df2[col] = le.fit_transform(df2[col])
    label_encoders[col] = le

#filling null values
num_cols = df2.select_dtypes(include=np.number).columns

for col in num_cols:
    df2[col].fillna(df2[col].median(), inplace=True)


x = df2.drop('Loan_ID',axis=1)
x = x.drop('Loan_Status' , axis=1)
y = df2['Loan_Status']
```

```
#split data
x_train , x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=42)


param_grid = {
    'learning_rate': [0.5,0.1, 0.01, 0.05],
    'n_estimators': [50, 100, 200]}


model = XGBClassifier(random_state=42, eval_metric='logloss')

grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    scoring='roc_auc',   # Or 'accuracy', 'f1', etc.
    cv=5,                # 5-fold cross-validation
    n_jobs=-1,           # Use all CPU cores
    verbose=1
)

grid_search.fit(x_train, y_train)
print("Best score:", grid_search.best_score_)
print("Best parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

y_pred_xgb = best_model.predict(x_test)
```

Save Processed Data

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Stat
0	1	0	0	0	0	5849	0.0	128.0	360.0	1.0	2	
1	1	1	1	0	0	4583	1508.0	128.0	360.0	1.0	0	
2	1	1	0	0	1	3000	0.0	66.0	360.0	1.0	2	
3	1	1	0	1	0	2583	2358.0	120.0	360.0	1.0	2	
4	1	0	0	0	0	6000	0.0	141.0	360.0	1.0	2	
...	...	...	...	...	...	...	...	...	...	...	...	...
609	0	0	0	0	0	2900	0.0	71.0	360.0	1.0	0	
610	1	1	3	0	0	4106	0.0	40.0	180.0	1.0	0	
611	1	1	1	0	0	8072	240.0	253.0	360.0	1.0	2	
612	1	1	2	0	0	7583	0.0	187.0	360.0	1.0	2	
613	0	0	0	0	1	4583	0.0	133.0	360.0	0.0	1	

614 rows × 12 columns