# Data analysis and visualization

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from math import pi

import seaborn as sns

# classifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

# metrics

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier

from xgboost import XGBClassifier

import warnings

warnings.filterwarnings('ignore')

loan\_train = pd.read\_csv("D:/loan\_train.csv")

loan\_test = pd.read\_csv("D:/loan\_test.csv")

# read in csv file as a DataFrame

lp\_df = pd.read\_csv("D:/loan\_train.csv")

loan\_train.head()

# explore the first 20 rows

lp\_df.head(20)

loan\_train\_cc = loan\_train.copy()

loan\_train.columns

loan\_test.columns

lp\_df.shape

lp\_df.info()

loan\_train.dtypes

#summary statistics

lp\_df.describe()

len(loan\_train)

len(loan\_test)

loan\_train.isna().values.any()

loan\_test.isna().values.any()

#the Id column is not needed, let's drop it

lp\_df.drop('Loan\_ID',axis=1,inplace=True)

# explore the first 5 rows

lp\_df.head()

#missing values in decsending order

lp\_df.isnull().sum().sort\_values(ascending=False)

#filling the missing data

null\_cols = ['Credit\_History', 'Self\_Employed', 'LoanAmount','Dependents', 'Loan\_Amount\_Term', 'Gender', 'Married']

for col in null\_cols:

print(f"{col}:\n{lp\_df[col].value\_counts()}\n","-"\*50)

lp\_df[col] = lp\_df[col].fillna(

lp\_df[col].dropna().mode().values[0] )

lp\_df.isnull().sum().sort\_values(ascending=False)

# Let's visualize the missing data in the TEST data

import seaborn as sns

plt.figure(figsize=(10,6))

sns.displot(

data=loan\_train.isna().melt(value\_name="missing"),

y="variable",

hue="missing",

multiple="fill",

aspect=1.25)

plt.show()

loan\_train.isna().sum()

loan\_train['Credit\_History'].fillna(method='ffill', inplace=True)

loan\_train['Credit\_History'].isna().values.any()

median\_loan = loan\_train['Loan\_Amount\_Term'].median()

loan\_train['Loan\_Amount\_Term'].fillna((median\_loan), inplace=True)

loan\_train['Loan\_Amount\_Term'].isna().values.any()

# We'll fill this column using the median of the values

median\_loan\_amount = loan\_train['LoanAmount'].median()

loan\_train['LoanAmount'].fillna((median\_loan\_amount), inplace=True)

loan\_train['LoanAmount'].isna().values.any()

# Count the values to know which occurs most frequently

loan\_train['Self\_Employed'].value\_counts()

#Fill with mode

loan\_train['Self\_Employed'].fillna('No', inplace=True)

loan\_train['Self\_Employed'].isna().values.any()

# fill with mode

loan\_train['Dependents'].fillna(0, inplace=True)

loan\_train['Dependents'].isna().values.any()

loan\_train['Married'].mode()

# fill with mode

loan\_train['Married'].fillna('Yes', inplace=True)

loan\_train['Married'].isna().values.any()

loan\_train['Gender'].mode()

# fill with mode

loan\_train['Gender'].fillna('Male', inplace=True)

loan\_train['Gender'].isna().values.any()

# Let's run a quick check

loan\_train.isna().sum()

# A preview of missing data in the testing set

loan\_test.isna().sum()

# fill in credit history

loan\_test['Credit\_History'].fillna(method='ffill', inplace=True)

# fill in loan amount term

median\_loan\_test = loan\_test['Loan\_Amount\_Term'].median()

loan\_test['Loan\_Amount\_Term'].fillna((median\_loan\_test), inplace=True)

# fill in loan amount

median\_loan\_amount\_test = loan\_test['LoanAmount'].median()

loan\_test['LoanAmount'].fillna((median\_loan\_amount\_test), inplace=True)

# fill in self employed

loan\_test['Self\_Employed'].fillna('No', inplace=True)

# fill in dependents

loan\_test['Dependents'].fillna(0, inplace=True)

# fill in gender

loan\_test['Gender'].fillna('Male', inplace=True)

loan\_test.isna().values.any()

loan\_test.isna().sum()

loan\_train.duplicated().values.any()

loan\_test.duplicated().values.any()

loan\_train.head()

#list of all the numeric columns

num = lp\_df.select\_dtypes('number').columns.to\_list()

#list of all the categoric columns

cat = lp\_df.select\_dtypes('object').columns.to\_list()

#numeric df

loan\_num = lp\_df[num]

#categoric df

loan\_cat = lp\_df[cat]

# Print value counts

print(lp\_df[cat[-1]].value\_counts())

# Plot countplot with 'Loan\_Status' as hue

plt.figure(figsize=(8, 10))

sns.set(style="whitegrid")

sns.countplot(x=cat[-1], data=lp\_df)

plt.show()

for i in loan\_num:

plt.hist(loan\_num[i])

plt.title(i)

plt.show()

for i in loan\_cat:

plt.figure(figsize=(15,10))

plt.subplot(2,3,1)

sns.countplot(x=i ,hue='Loan\_Status', data=lp\_df ,palette='plasma')

plt.xlabel(i, fontsize=14)

# Bar charts to get a high level view of categorical data

fig, ax = plt.subplots(3, 2, figsize=(16, 18))

loan\_train.groupby(['Gender'])[['Gender']].count().plot.bar(

color=plt.cm.Paired(np.arange(len(loan\_train))), ax=ax[0,0])

loan\_train.groupby(['Married'])[['Married']].count().plot.bar(

color=plt.cm.Paired(np.arange(len(loan\_train))), ax=ax[0,1])

loan\_train.groupby(['Education'])[['Education']].count().plot.bar(

color=plt.cm.Paired(np.arange(len(loan\_train))), ax=ax[1,0])

loan\_train.groupby(['Self\_Employed'])[['Self\_Employed']].count().plot.bar(

color=plt.cm.Paired(np.arange(len(loan\_train))), ax=ax[1,1])

loan\_train.groupby(['Loan\_Status'])[['Loan\_Status']].count().plot.bar(

color=plt.cm.Paired(np.arange(len(loan\_train))),ax=ax[2,0])

loan\_train.groupby(['Property\_Area'])[['Loan\_Status']].count().plot.bar(

color=plt.cm.Paired(np.arange(len(loan\_train))),ax=ax[2,1])

plt.show()

# Here, I pass all categorical columns into a list

categorical\_columns = loan\_train\_cc.select\_dtypes('object').columns.to\_list()

# Then, I filter he list to remove Loan\_ID column which is not relevant to the analysis

categorical\_columns[1:]

# This code loops through the list, and creates a chart for each

for i in categorical\_columns[1:]:

plt.figure(figsize=(15,10))

plt.subplot(3,2,1)

sns.countplot(x=i ,hue='Loan\_Status', data=loan\_train\_cc, palette='ocean')

plt.xlabel(i, fontsize=14)

#Plot4- Scatterplot

fig, ax = plt.subplots(2,2, figsize=(14,12))

sns.scatterplot(data=loan\_train,x="ApplicantIncome", y="LoanAmount",s=70, hue="Loan\_Status", palette='ocean',ax=ax[0,0])

sns.histplot(loan\_train, x=loan\_train['LoanAmount'], bins=10, ax=ax[0,1])

sns.scatterplot(data=loan\_train,x='CoapplicantIncome', y='LoanAmount',s=70, hue='Loan\_Status',palette='ocean', ax=ax[1,0])

sns.scatterplot(data=loan\_train,x='Loan\_Amount\_Term', y='LoanAmount', s=70, hue='Loan\_Status',palette='ocean', ax=ax[1,1])

plt.show()

loan\_train.corr()

# Let's plot correlation overview of the variables.

fig, ax = plt.subplots(figsize=(9, 7))

correlations = loan\_train.corr()

# plotting correlation heatmap

dataplot = sns.heatmap(correlations, cmap="YlGnBu", annot=True)

# displaying heatmap

plt.show()

# Let's take another preview of the data

loan\_train.head()

#first identify all categorical columns & pass into a variable

objectlist\_train = loan\_train.select\_dtypes(include = "object").columns

# Then Label Encoding for object to numeric conversion

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for feature in objectlist\_train:

loan\_train[feature] = le.fit\_transform(loan\_train[feature].astype(str))

print (loan\_train.info())

# Now, repeat the same process to encode the test data

objectlist\_test = loan\_test.select\_dtypes(include='object').columns

for feature in objectlist\_test:

loan\_test[feature] = le.fit\_transform(loan\_test[feature].astype(str))

print (loan\_test.info())

sns.heatmap(lp\_df.corr() ,cmap='cubehelix\_r')

x = loan\_train.iloc[:,1:].drop('Loan\_Status', axis=1) # drop loan\_status column because that is what we are predicting

y = loan\_train['Loan\_Status']

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x, y, test\_size=0.30, random\_state=0)

X = lp\_df.drop('Loan\_Status' , axis = 1 )

y = lp\_df['Loan\_Status']

X\_train ,X\_test , y\_train , y\_test = train\_test\_split(X , y , test\_size = 0.3 , random\_state =102)

adamodel = AdaBoostClassifier()

adamodel.fit(X\_train , y\_train)

pred\_l = adamodel.predict(X\_test)

acc\_l = accuracy\_score(y\_test , pred\_l)\*100

acc\_l

gbk = GradientBoostingClassifier()

gbk.fit(X\_train, y\_train)

pred\_gbc = gbk.predict(X\_test)

acc\_gbc = accuracy\_score(y\_test , pred\_gbc)\*100

acc\_gbc

XGB = XGBClassifier()

XGB.fit(X\_train, y\_train)

pred\_x = XGB.predict(X\_test)

acc\_x = accuracy\_score(y\_test , pred\_x)\*100

acc\_x

models = pd.DataFrame({

'Model': ['Ada Boosting Classifier','Gradient Boosting Classifier', 'Extreme Gradient Boosting Classifier'],

'Score': [acc\_l,acc\_gbc,acc\_x ]})

models.sort\_values(by='Score', ascending=False)

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

from xgboost import XGBClassifier

loan\_data = pd.read\_csv('D:/loan\_train.csv')

print(loan\_data.columns)

# Use the actual target column name

target\_column = 'Loan\_Status'

X = loan\_data.drop(target\_column, axis=1)

y = LabelEncoder().fit\_transform(loan\_data[target\_column])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = XGBClassifier()

# Drop 'Loan\_ID' column before one-hot encoding

loan\_data\_encoded = pd.get\_dummies(loan\_data.drop('Loan\_ID', axis=1), columns=categorical\_columns)

# Update target\_column based on your actual target variable

target\_column = 'Loan\_Status'

# Separate features (X) and target variable (y)

X = loan\_data\_encoded.drop(target\_column, axis=1)

y = LabelEncoder().fit\_transform(loan\_data\_encoded[target\_column])

# Continue with the rest of your code

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = XGBClassifier()

model.fit(X\_train, y\_train)

y\_pred\_prob = model.predict\_proba(X\_test)[:, 1]

import matplotlib.pyplot as plt

from xgboost import plot\_importance

# Assuming 'model' is your trained XGBoost model

# If needed, train the model before running this code

# Plot feature importance

plot\_importance(model)

plt.show()

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

from xgboost import XGBClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

# Assuming you have your data loaded in X and y

# If needed, preprocess your data before running this code

# 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\_state=42)

# Initialize and train the XGBoost model

model = XGBClassifier()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Create a confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Display the confusion matrix using seaborn

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.title("Confusion Matrix")

plt.show()

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from xgboost import XGBClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import make\_classification

# Generate a synthetic dataset (replace with your actual data)

X, y = make\_classification(n\_samples=1000, n\_features=20, random\_state=42)

# 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\_state=42)

# Initialize and train the XGBoost model with a higher number of estimators

model = XGBClassifier(n\_estimators=500) # You can adjust this value

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Compute performance metrics

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)

# Display the performance metrics

print(f"Accuracy: {accuracy:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1-Score: {f1:.2f}")

import pandas as pd

# Assuming you have your dataset loaded into a DataFrame called 'loan\_data'

# Replace 'loan\_data' with the actual variable name of your DataFrame

# Display general information about the dataset

loan\_data.info()

# Display the first few rows of the dataset

print("\nFirst few rows of the dataset:")

print(loan\_data.head())

import matplotlib.pyplot as plt

models = ['XG Boost', 'DecisionTree', 'RandomForest']

accuracy\_values = [90.0, 63.01, 72.60]

plt.bar(models, accuracy\_values, color=['skyblue', 'pink', 'red'])

plt.ylabel('Accuracy')

plt.title('Model Comparison')

plt.ylim(0, 100)

# Add labels to each bar

for i, v in enumerate(accuracy\_values):

plt.text(i, v + 1, str(round(v, 2)), ha='center', va='bottom')

plt.show()