```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
!pip install dmba
from dmba import classificationSummary
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score, f1_score
```

Show hidden output

Upload files from local system

```
from google.colab import files
up = files.upload()
```

Choose Files loan_approval.csv

loan_approval.csv(text/csv) - 115793 bytes, last modified: 11/7/2025 - 100% done

```
Saving loan_approval.csv to loan_approval (1).csv
```

Read the loan approval dataset

```
loan_df = pd.read_csv('loan_approval.csv')
print(loan_df.head(5))
```

```
              name              city  income  credit_score  loan_amount  \
0     Allison Hill         East Jill  113810           389        39698
1     Brandon Hall     New Jamesside   44592           729        15446
2     Rhonda Smith      Lake Roberto   33278           584        11189
3  Gabrielle Davis  West Melanieview  127196           344        48823
4     Valerie Gray         Mariastad   66048           496        47174

   years_employed  points  loan_approved
0              27    50.0          False
1              28    55.0          False
2              13    45.0          False
3              29    50.0          False
4               4    25.0          False
```

Check for missing values and information

```
print(loan_df.isnull().sum())
loan_df.info()
```

```
name              0
city              0
income            0
credit_score      0
loan_amount       0
years_employed    0
points            0
loan_approved     0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   name            2000 non-null   object
 1   city            2000 non-null   object
 2   income          2000 non-null   int64
 3   credit_score    2000 non-null   int64
 4   loan_amount     2000 non-null   int64
```

```
 5   years_employed  2000 non-null   int64
 6   points          2000 non-null   float64
 7   loan_approved   2000 non-null   bool
dtypes: bool(1), float64(1), int64(4), object(2)
memory usage: 111.5+ KB
```

Create copies and drop unnecessary columns

```
loan_df2 = loan_df
loan_df3 = loan_df
loan_df4 = loan_df
loan_df = loan_df.drop(['name', 'city', 'points'], axis=1)
print(loan_df.head())
```

```
   income  credit_score  loan_amount  years_employed  loan_approved
0  113810           389        39698              27          False
1   44592           729        15446              28          False
2   33278           584        11189              13          False
3  127196           344        48823              29          False
4   66048           496        47174               4          False
```

NN classification:

Define predictor and outcome for NN classification

```
predictors = ['income', 'credit_score', 'loan_amount', 'years_employed']
outcome = 'loan_approved'

predictors = [c for c in loan_df.columns if c != outcome]
print(loan_df.shape)
```

```
(2000, 5)
```

Split data into training data and validation data, and scale features

```
X = loan_df[predictors]
y = loan_df[outcome]
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

scaler = StandardScaler()
scaler.fit(train_X)

train_X = scaler.transform(train_X)
valid_X = scaler.transform(valid_X)
```

Train an NN classifier

```
clf = MLPClassifier(hidden_layer_sizes=(6), activation='logistic', solver='adam',
random_state=1, max_iter=10000)
clf.fit(train_X, train_y.values)
```

```
MLPClassifier(activation='logistic', hidden_layer_sizes=6, max_iter=10000,
              random_state=1)
```

classification summary

```
classificationSummary(train_y, clf.predict(train_X))
classificationSummary(valid_y, clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.9075)

       Prediction
```

```
Actual   0   1
     0 614  55
     1  56 475
Confusion Matrix (Accuracy 0.8975)

       Prediction
Actual   0   1
     0 412  40
     1  42 306
```

```
valid_pred = clf.predict(valid_X)
precision = precision_score(valid_y, valid_pred)
recall = recall_score(valid_y, valid_pred)
f1 = f1_score(valid_y, valid_pred)

print(f"NN Classifier Metrics:")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1-Score:  {f1:.4f}")
```

```
NN Classifier Metrics:
Precision: 0.8844
Recall:    0.8793
F1-Score:  0.8818
```

NN regression:

Drop unnecessary columns

```
loan_df2 = loan_df2.drop(['name', 'city', 'loan_approved'], axis=1)
print(loan_df2.head())
```

```
   income  credit_score  loan_amount  years_employed  points
0  113810           389        39698              27    50.0
1   44592           729        15446              28    55.0
2   33278           584        11189              13    45.0
3  127196           344        48823              29    50.0
4   66048           496        47174               4    25.0
```

Define predictors and outcome for NN regression

```
predictors2 = ['income', 'credit_score', 'loan_amount', 'years_employed']
outcome2 = 'points'
```

Split data into training data and validation data, and scale features

```
X2 = loan_df2[predictors2]
y2 = loan_df2[outcome2]
train_X2, valid_X2, train_y2, valid_y2 = train_test_split(X2, y2, test_size=0.4, random_state=1)

scaler2 = StandardScaler()
scaler2.fit(train_X2)

train_X2 = scaler2.transform(train_X2)
valid_X2 = scaler2.transform(valid_X2)
```

Train an NN regressor

```
clf2 = MLPRegressor(hidden_layer_sizes=(6), activation='relu', solver='adam',
random_state=1, max_iter=10000)
clf2.fit(train_X2, train_y2.values)
```

```
MLPRegressor
MLPRegressor(hidden_layer_sizes=6, max_iter=10000, random_state=1)
```

NN regression model performance

```
train_pred = clf2.predict(train_X2)
valid_pred = clf2.predict(valid_X2)

train_r2 = r2_score(train_y2, train_pred)
valid_r2 = r2_score(valid_y2, valid_pred)
train_rmse = np.sqrt(mean_squared_error(train_y2, train_pred))
valid_rmse = np.sqrt(mean_squared_error(valid_y2, valid_pred))

print("Train:")
print("R-squared:", train_r2)
print("RMSE:", train_rmse)

print("Valid:")
print("R-squared:", valid_r2)
print("RMSE:", valid_rmse)
```

```
Train:
R-squared: 0.9295515300418117
RMSE: 4.98450161674012
Valid:
R-squared: 0.923129384117913
RMSE: 5.10474437354557
```

NN regression Cross-Validation

```
scores_clf2 = cross_val_score(
    clf2, X2, y2, cv=5,
    scoring='neg_root_mean_squared_error')

scores_clf2 = -scores_clf2

print("NN regression Cross-Validation:")
print(f"RMSE Scores (5-fold): {np.round(scores_clf2, 3)}")
print(f"Mean RMSE: {scores_clf2.mean():.3f}")
print(f"RMSE Std Dev: {scores_clf2.std():.3f}")
```

```
NN regression Cross-Validation:
RMSE Scores (5-fold): [14.205 12.755 13.41  12.768 13.291]
Mean RMSE: 13.286
RMSE Std Dev: 0.531
```

CART regression:

Define predictor and outcome for CART

```
predictors3 = ['income', 'credit_score', 'loan_amount', 'years_employed']
outcome3 = 'points'

X3 = loan_df3[predictors3]
y3 = loan_df3[outcome3]
train_X3, valid_X3, train_y3, valid_y3 = train_test_split(X3, y3, test_size=0.4, random_state=1)
```

Train an CART regressor with max_depth=8

```
cart_r8 = DecisionTreeRegressor(max_depth=8, random_state=1)
cart_r8.fit(train_X3, train_y3.values)

train_pred_cart_r8 = cart_r8.predict(train_X3)
valid_pred_cart_r8 = cart_r8.predict(valid_X3)

cart_r8_train_r3 = r2_score(train_y3, train_pred_cart_r8)
cart_r8_valid_r3 = r2_score(valid_y3, valid_pred_cart_r8)
cart_r8_train_rmse3 = np.sqrt(mean_squared_error(train_y3, train_pred_cart_r8))
cart_r8_valid_rmse3 = np.sqrt(mean_squared_error(valid_y3, valid_pred_cart_r8))
```

CART8 regression model performance

```
print("CART_r8 Train:")
print("R-squared:", cart_r8_train_r3)
print("RMSE:", cart_r8_train_rmse3)

print("CART_r8 Valid:")
print("R-squared:", cart_r8_valid_r3)
print("RMSE:", cart_r8_valid_rmse3)
```

```
CART_r8 Train:
R-squared: 0.993666627854971
RMSE: 1.4945259573300949
CART_r8 Valid:
R-squared: 0.9653461079018368
RMSE: 3.427439742339933
```

Train an CART regressor with max_depth=7

```
cart_r7 = DecisionTreeRegressor(max_depth=7, random_state=1)
cart_r7.fit(train_X3, train_y3.values)

train_pred_cart_r7 = cart_r7.predict(train_X3)
valid_pred_cart_r7 = cart_r7.predict(valid_X3)

cart_r7_train_r3 = r2_score(train_y3, train_pred_cart_r7)
cart_r7_valid_r3 = r2_score(valid_y3, valid_pred_cart_r7)
cart_r7_train_rmse3 = np.sqrt(mean_squared_error(train_y3, train_pred_cart_r7))
cart_r7_valid_rmse3 = np.sqrt(mean_squared_error(valid_y3, valid_pred_cart_r7))
```

CART7 regression model performance

```
print("CART_r7 Train:")
print("R-squared:", cart_r7_train_r3)
print("RMSE:", cart_r7_train_rmse3)

print("CART_r7 Valid:")
print("R-squared:", cart_r7_valid_r3)
print("RMSE:", cart_r7_valid_rmse3)
```

```
CART_r7 Train:
R-squared: 0.9855979145954754
RMSE: 2.253713206104595
CART_r7 Valid:
R-squared: 0.9607408935212078
RMSE: 3.6480772344659123
```

Train an CART regressor with max_depth=9

```
cart_r9 = DecisionTreeRegressor(max_depth=9, random_state=1)
cart_r9.fit(train_X3, train_y3.values)

train_pred_cart_r9 = cart_r9.predict(train_X3)
valid_pred_cart_r9 = cart_r9.predict(valid_X3)

cart_r9_train_r3 = r2_score(train_y3, train_pred_cart_r9)
cart_r9_valid_r3 = r2_score(valid_y3, valid_pred_cart_r9)
cart_r9_train_rmse3 = np.sqrt(mean_squared_error(train_y3, train_pred_cart_r9))
cart_r9_valid_rmse3 = np.sqrt(mean_squared_error(valid_y3, valid_pred_cart_r9))
```

CART9 regression model performance

```
print("CART_r9 Train:")
print("R-squared:", cart_r9_train_r3)
print("RMSE:", cart_r9_train_rmse3)

print("CART_r9 Valid:")
print("R-squared:", cart_r9_valid_r3)
print("RMSE:", cart_r9_valid_rmse3)
```

```
CART_r9 Train:
R-squared: 0.9976059462347057
RMSE: 0.9188675366951881
CART_r9 Valid:
R-squared: 0.9674563081839435
RMSE: 3.3214462448298687
```

Train an CART regressor with max_depth=10

```
cart_r10 = DecisionTreeRegressor(max_depth=10, random_state=1)
cart_r10.fit(train_X3, train_y3.values)

train_pred_cart_r10 = cart_r10.predict(train_X3)
valid_pred_cart_r10 = cart_r10.predict(valid_X3)

cart_r10_train_r3 = r2_score(train_y3, train_pred_cart_r10)
cart_r10_valid_r3 = r2_score(valid_y3, valid_pred_cart_r10)
cart_r10_train_rmse3 = np.sqrt(mean_squared_error(train_y3, train_pred_cart_r10))
cart_r10_valid_rmse3 = np.sqrt(mean_squared_error(valid_y3, valid_pred_cart_r10))
```

CART10 regression model performance

```
print("CART_r10 Train:")
print("R-squared:", cart_r10_train_r3)
print("RMSE:", cart_r10_train_rmse3)

print("CART_r10 Valid:")
print("R-squared:", cart_r10_valid_r3)
print("RMSE:", cart_r10_valid_rmse3)
```

```
CART_r10 Train:
R-squared: 0.9992613598778436
RMSE: 0.5103902926539864
CART_r10 Valid:
R-squared: 0.9664697571568879
RMSE: 3.3714146389925452
```

Cart_r9 Cross-Validation

```
scores_cart_r9 = cross_val_score(
    cart_r9, X3, y3, cv=5,
    scoring='neg_root_mean_squared_error')

scores_cart_r9 = -scores_cart_r9

print("Cart_r9 Cross-Validation:")
print(f"RMSE Scores (5-fold): {np.round(scores_cart_r9, 3)}")
print(f"Mean RMSE: {scores_cart_r9.mean():.3f}")
print(f"RMSE Std Dev: {scores_cart_r9.std():.3f}")
```

```
Cart_r9 Cross-Validation:
RMSE Scores (5-fold): [2.811 3.077 3.308 3.527 3.06 ]
Mean RMSE: 3.156
RMSE Std Dev: 0.243
```

CART classification:

```
loan_df4 = loan_df4.drop(['name', 'city', 'points'], axis=1)

X4 = loan_df4[predictors]
```

```
y4 = loan_df4[outcome]
train_X4, valid_X4, train_y4, valid_y4 = train_test_split(X4, y4, test_size=0.4, random_state=1)
```

Train an CART classification with max_depth=9

```
cart_c9 = DecisionTreeClassifier(max_depth=9, random_state=1)
cart_c9.fit(train_X4, train_y4)

classificationSummary(train_y4, cart_c9.predict(train_X4))
classificationSummary(valid_y4, cart_c9.predict(valid_X4))
```

```
Confusion Matrix (Accuracy 0.9983)

       Prediction
Actual   0   1
     0 667   2
     1   0 531
Confusion Matrix (Accuracy 0.9725)

       Prediction
Actual   0   1
     0 446   6
     1  16 332
```

Train an CART classification with max_depth=8

```
cart_c8 = DecisionTreeClassifier(max_depth=8, random_state=1)
cart_c8.fit(train_X4, train_y4)

classificationSummary(train_y4, cart_c8.predict(train_X4))
classificationSummary(valid_y4, cart_c8.predict(valid_X4))
```

```
Confusion Matrix (Accuracy 0.9967)

       Prediction
Actual   0   1
     0 665   4
     1   0 531
Confusion Matrix (Accuracy 0.9712)

       Prediction
Actual   0   1
     0 445   7
     1  16 332
```

Train an CART classification with max_depth=10

```
cart_c10 = DecisionTreeClassifier(max_depth=10, random_state=1)
cart_c10.fit(train_X4, train_y4)

classificationSummary(train_y4, cart_c10.predict(train_X4))
classificationSummary(valid_y4, cart_c10.predict(valid_X4))
```

```
Confusion Matrix (Accuracy 0.9983)

       Prediction
Actual   0   1
     0 669   0
     1   2 529
Confusion Matrix (Accuracy 0.9725)

       Prediction
Actual   0   1
     0 446   6
     1  16 332
```

Cart_c9 Cross-Validation

```
scores_cart_c9 = cross_val_score(
    cart_c9, X4, y4, cv=5,
    scoring='neg_root_mean_squared_error')
```

```
scores_cart_c9 = -scores_cart_c9

print("CART classification Cross-Validation:")
print(f"RMSE Scores (5-fold): {np.round(scores_cart_c9, 3)}")
print(f"Mean RMSE: {scores_cart_c9.mean():.3f}")
print(f"RMSE Std Dev: {scores_cart_c9.std():.3f}")
```

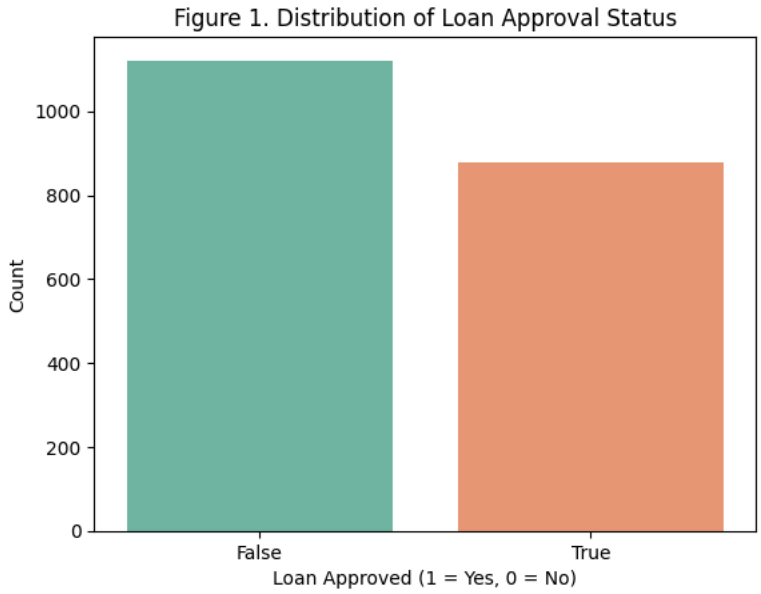
```
CART classification Cross-Validation:
RMSE Scores (5-fold): [0.158 0.173 0.158 0.18  0.166]
Mean RMSE: 0.167
RMSE Std Dev: 0.009
```

```
valid_pred4 = cart_c9.predict(valid_X4)
precision = precision_score(valid_y4, valid_pred4)
recall = recall_score(valid_y4, valid_pred4)
f1 = f1_score(valid_y4, valid_pred4)

print(f"CART Classifier Metrics:")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1-Score:  {f1:.4f}")
```

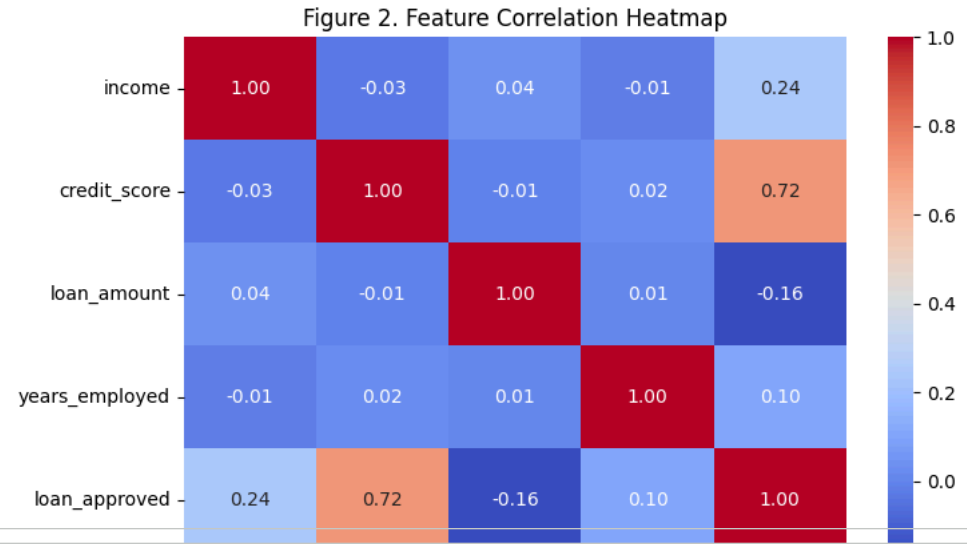
```
CART Classifier Metrics:
Precision: 0.9822
Recall:    0.9540
F1-Score:  0.9679
```

```
sns.countplot(x='loan_approved', data=loan_df, palette='Set2')
plt.title('Figure 1. Distribution of Loan Approval Status')
plt.xlabel('Loan Approved (1 = Yes, 0 = No)')
plt.ylabel('Count')
plt.show()
```

```
/tmp/ipython-input-876353508.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x='loan_approved', data=loan_df, palette='Set2')
```

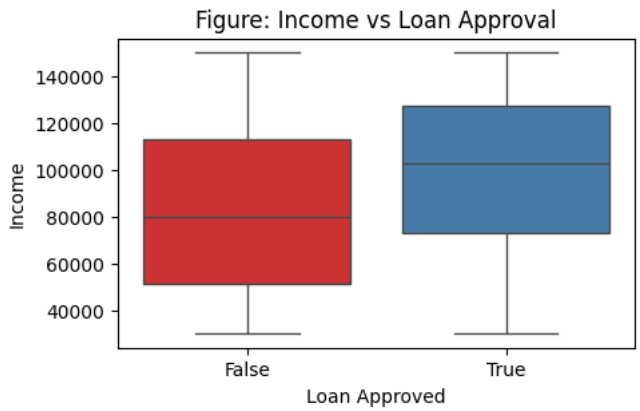



```
plt.figure(figsize=(8,5))
sns.heatmap(loan_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Figure 2. Feature Correlation Heatmap')
plt.show()
```

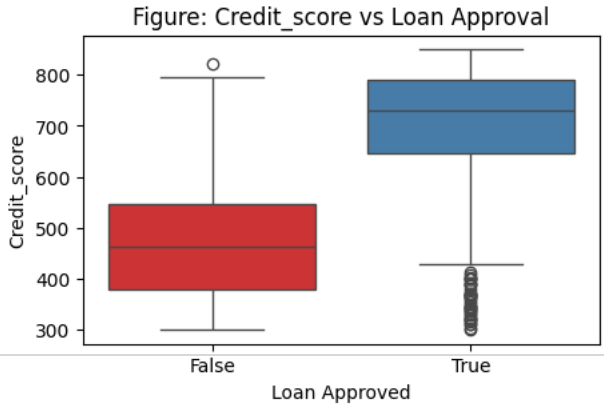



```
features = ['income', 'credit_score', 'loan_amount', 'years_employed']

for col in features:
    plt.figure(figsize=(5,3))
    sns.boxplot(x='loan_approved', y=col, data=loan_df, palette='Set1')
    plt.title(f'Figure: {col.capitalize()} vs Loan Approval')
    plt.xlabel('Loan Approved')
    plt.ylabel(col.capitalize())
    plt.show()
```

```
/tmp/ipython-input-1790707769.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.boxplot(x='loan_approved', y=col, data=loan_df, palette='Set1')
```




```
/tmp/ipython-input-1790707769.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.boxplot(x='loan_approved', y=col, data=loan_df, palette='Set1')
```




```
/tmp/ipython-input-1790707769.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.boxplot(x='loan_approved', y=col, data=loan_df, palette='Set1')
```