# loan-prediction-project-ml

May 5, 2023

#**Summary To Explain Project (Keypoints)**

- Import Required Library
- Display Top 5 , Last 5 Data and Display Dataset Information
- Check Shape and Null Value From Dataset
- Handle Missing & Categorical Column Data
- Store Target Column & Other Feature Column
- Feature Scaling
- Split Dataset For Testign
- Train & Check Different ML Model
- Save Model
- GUI (In Google Colab GUI is Not Working That's Why Code is Commented)

**Note :** *Here is the small dataset so I don't use any other library for cleaning or pre-processing dataset. Also I don't use One-Hot Encoding.*

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import joblib
from tkinter import *
import pandas as pd
```

[2]:
```
df = pd.read_csv("/content/drive/MyDrive/MyDataSet/Load_Prediction/train.csv")
```

# 1 1. Display Top 5 Rows && Last 5 Rows of The Dataset

[3]:
```
df.head(10)
```

[3]:

| | 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 |
| 5 | LP001011 | Male | Yes | 2 | Graduate | Yes |
| 6 | LP001013 | Male | Yes | 0 | Not Graduate | No |
| 7 | LP001014 | Male | Yes | 3+ | Graduate | No |
| 8 | LP001018 | Male | Yes | 2 | Graduate | No |
| 9 | LP001020 | Male | Yes | 1 | Graduate | No |

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term |
|---|---|---|---|---|
| 0 | 5849 | 0.0 | NaN | 360.0 |
| 1 | 4583 | 1508.0 | 128.0 | 360.0 |
| 2 | 3000 | 0.0 | 66.0 | 360.0 |
| 3 | 2583 | 2358.0 | 120.0 | 360.0 |
| 4 | 6000 | 0.0 | 141.0 | 360.0 |
| 5 | 5417 | 4196.0 | 267.0 | 360.0 |
| 6 | 2333 | 1516.0 | 95.0 | 360.0 |
| 7 | 3036 | 2504.0 | 158.0 | 360.0 |
| 8 | 4006 | 1526.0 | 168.0 | 360.0 |
| 9 | 12841 | 10968.0 | 349.0 | 360.0 |

| | Credit_History | Property_Area | Loan_Status |
|---|---|---|---|
| 0 | 1.0 | Urban | Y |
| 1 | 1.0 | Rural | N |
| 2 | 1.0 | Urban | Y |
| 3 | 1.0 | Urban | Y |
| 4 | 1.0 | Urban | Y |
| 5 | 1.0 | Urban | Y |
| 6 | 1.0 | Urban | Y |
| 7 | 0.0 | Semiurban | N |
| 8 | 1.0 | Urban | Y |
| 9 | 1.0 | Semiurban | N |

[4]:
```
df.tail()
```

[4]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed |
|---|---|---|---|---|---|---|
| 609 | LP002978 | Female | No | 0 | Graduate | No |
| 610 | LP002979 | Male | Yes | 3+ | Graduate | No |
| 611 | LP002983 | Male | Yes | 1 | Graduate | No |
| 612 | LP002984 | Male | Yes | 2 | Graduate | No |

```
613  LP002990   Female      No          0  Graduate          Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N
```

# 2 2. Find Shape of Our Dataset (Number of Rows And Number of Columns)

[5]:
```python
df.shape
```

[5]:
```
(614, 13)
```

# 3 3. Get Information About Data Set

[6]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
```

```
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

# 4 4. Check Null Values

[7]:
```
df.isnull().sum()
```

[7]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

# 5 5. Handle Missing Value

[8]:
```
df = df.drop('Loan_ID',axis=1)
```

[9]:
```
columns = ['Gender','Dependents','LoanAmount','Loan_Amount_Term']
```

[10]:
```
df = df.dropna(subset=columns)
```

[11]:
```
df.isnull().sum()
```

[11]:
```
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed        30
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      0
Credit_History       48
Property_Area         0
Loan_Status           0
dtype: int64
```

[12]:
```
df['Self_Employed'].mode()[0]
```

[12]: 'No'

[13]:
```
df['Self_Employed'] =df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
```

[14]:
```
df['Credit_History'].mode()[0]
```

[14]: 1.0

[15]:
```
df['Credit_History'] =df['Credit_History'].fillna(df['Credit_History'].
↪mode()[0])
```

# 6 6. Handling Categorical Columns

[16]:
```
df.head()
```

[16]:
```
  Gender Married Dependents     Education Self_Employed  ApplicantIncome  \
1   Male     Yes          1      Graduate            No             4583
2   Male     Yes          0      Graduate           Yes             3000
3   Male     Yes          0  Not Graduate            No             2583
4   Male      No          0      Graduate            No             6000
5   Male     Yes          2      Graduate           Yes             5417

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0
5             4196.0       267.0             360.0             1.0

  Property_Area Loan_Status
1         Rural           N
2         Urban           Y
3         Urban           Y
4         Urban           Y
5         Urban           Y
```

[17]:
```
df['Dependents'] =df['Dependents'].replace(to_replace="3+",value='4')
```

[18]:
```
df['Loan_Status'].unique()
```

[18]: array(['N', 'Y'], dtype=object)

[19]:
```
df['Gender'] = df['Gender'].map({'Male':1,'Female':0}).astype('int')
df['Married'] = df['Married'].map({'Yes':1,'No':0}).astype('int')
```

```
df['Education'] = df['Education'].map({'Graduate':1,'Not Graduate':0}).
↪astype('int')
df['Self_Employed'] = df['Self_Employed'].map({'Yes':1,'No':0}).astype('int')
df['Property_Area'] = df['Property_Area'].map({'Rural':0,'Semiurban':2,'Urban':
↪1}).astype('int')
df['Loan_Status'] = df['Loan_Status'].map({'Y':1,'N':0}).astype('int')
```

[20]:
```
df.head()
```

[20]:
```
   Gender  Married Dependents  Education  Self_Employed  ApplicantIncome  \
1       1        1          1          1              0             4583
2       1        1          0          1              1             3000
3       1        1          0          0              0             2583
4       1        0          0          1              0             6000
5       1        1          2          1              1             5417

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0
5             4196.0       267.0             360.0             1.0

   Property_Area  Loan_Status
1              0            0
2              1            1
3              1            1
4              1            1
5              1            1
```

# 7 7. Store Target Value In X and Other Features in y

[21]:
```
X = df.drop('Loan_Status',axis=1)
```

[22]:
```
y = df['Loan_Status']
```

# 8 8. Feature Scaling

[23]:
```
df.sample(5)
```

[23]:
```
     Gender  Married Dependents  Education  Self_Employed  ApplicantIncome  \
395       1        1          2          1              0             3276
570       1        1          1          1              0             3417
529       1        0          0          0              0             6783
230       1        1          1          1              0             2491
```

```
275        1         1           1          1              0            2750

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
395              484.0       135.0             360.0             1.0
570             1750.0       186.0             360.0             1.0
529                0.0       130.0             360.0             1.0
230             2054.0       104.0             360.0             1.0
275             1842.0       115.0             360.0             1.0

     Property_Area  Loan_Status
395              2            1
570              1            1
529              2            1
230              2            1
275              2            1
```

[24]:
```python
cols = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term']
```

[25]:
```python
st = StandardScaler()
X[cols]=st.fit_transform(X[cols])
```

[26]:
```python
X
```

[26]:
```
     Gender  Married Dependents  Education  Self_Employed  ApplicantIncome  \
1         1         1          1          1              0        -0.128694
2         1         1          0          1              1        -0.394296
3         1         1          0          0              0        -0.464262
4         1         0          0          1              0         0.109057
5         1         1          2          1              1         0.011239
..      …       …         …         …            …              …
609       0         0          0          1              0        -0.411075
610       1         1          4          1              0        -0.208727
611       1         1          1          1              0         0.456706
612       1         1          2          1              0         0.374659
613       0         0          0          1              1        -0.128694

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1            -0.049699   -0.214368          0.279961             1.0
2            -0.545638   -0.952675          0.279961             1.0
3             0.229842   -0.309634          0.279961             1.0
4            -0.545638   -0.059562          0.279961             1.0
5             0.834309    1.440866          0.279961             1.0
..                 …         …               …             …
609          -0.545638   -0.893134          0.279961             1.0
610          -0.545638   -1.262287         -2.468292             1.0
611          -0.466709    1.274152          0.279961             1.0
612          -0.545638    0.488213          0.279961             1.0
```

```
613          -0.545638   -0.154828          0.279961              0.0

     Property_Area
1                0
2                1
3                1
4                1
5                1
..             ...
609              0
610              0
611              1
612              1
613              2

[553 rows x 11 columns]
```

## 9 9. Split Dataset For Testing && Checking

[27]:
```
x_train, x_valid, y_train, y_valid = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state = 42)

print(x_train.shape)
print(x_valid.shape)
print(y_train.shape)
print(y_valid.shape)
```

```
(414, 11)
(139, 11)
(414,)
(139,)
```

## 10 10. Trained Different ML Model && Check

[28]:
```
model = RandomForestClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_valid, y_pred))
```

```
# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 1.0
Validation Accuracy : 0.762589928057554
F1 score : 0.8405797101449276
[[19 25]
 [ 8 87]]
```

[29]:

```
model = DecisionTreeClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 1.0
Validation Accuracy : 0.697841726618705
f1 score : 0.7789473684210526
[[23 21]
 [21 74]]
```

[30]:

```
model = SVC()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 0.8285024154589372
Validation Accuracy : 0.8057553956834532
```

```
f1 score : 0.8755760368663594
[[17 27]
 [ 0 95]]
```

[31]:
```
model = LogisticRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 0.8043478260869565
Validation Accuracy : 0.8129496402877698
f1 score : 0.8796296296296297
[[18 26]
 [ 0 95]]
```

# 11 11. Save The Model

[32]:
```
X = df.drop('Loan_Status',axis=1)
y = df['Loan_Status']
```

[33]:
```
rf = RandomForestClassifier(n_estimators=270,
 min_samples_split=5,
 min_samples_leaf=5,
 max_features='sqrt',
 max_depth=5)
```

[34]:
```
rf.fit(X,y)
```

[34]:
```
RandomForestClassifier(max_depth=5, min_samples_leaf=5, min_samples_split=5,
                       n_estimators=270)
```

[35]:
```
joblib.dump(rf,'loan_status_predict')
```

[35]:
```
['loan_status_predict']
```

[36]:
```
model = joblib.load('loan_status_predict')
```

[37]:
```
df.head()
```

[37]:
```
   Gender  Married Dependents  Education  Self_Employed  ApplicantIncome  \
1       1        1          1          1              0             4583
2       1        1          0          1              1             3000
3       1        1          0          0              0             2583
4       1        0          0          1              0             6000
5       1        1          2          1              1             5417

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0
5             4196.0       267.0             360.0             1.0

   Property_Area  Loan_Status
1              0            0
2              1            1
3              1            1
4              1            1
5              1            1
```

[38]:
```
# df[0:1]
p1 = np.array(df.values[8,:])[0:11]
print(p1)
```

```
[1 1 '1' 1 0 12841 10968.0 349.0 360.0 1.0 2]
```

[39]:
```
# import pandas as pd
# df = pd.DataFrame({
#     'Gender':1,
#     'Married':1,
#     'Dependents':2,
#     'Education':0,
#     'Self_Employed':0,
#     'ApplicantIncome':2889,
#     'CoapplicantIncome':0.0,
#     'LoanAmount':45,
#     'Loan_Amount_Term':180,
#     'Credit_History':0,
#     'Property_Area':1
# },index=[0])
```

[40]:
```
result = model.predict([p1])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but RandomForestClassifier was fitted with feature
```

names
  warnings.warn(

[41]:
```
if result==1:
    print("Loan Approved")
else:
    print("Loan Not Approved")
```

Loan Approved

# 12 12. GUI

[42]:
```
# def show_entry():

#     p1 = float(e1.get())
#     p2 = float(e2.get())
#     p3 = float(e3.get())
#     p4 = float(e4.get())
#     p5 = float(e5.get())
#     p6 = float(e6.get())
#     p7 = float(e7.get())
#     p8 = float(e8.get())
#     p9 = float(e9.get())
#     p10 = float(e10.get())
#     p11 = float(e11.get())

#     model = joblib.load('loan_status_predict')
#     df = pd.DataFrame({
#     'Gender':p1,
#     'Married':p2,
#     'Dependents':p3,
#     'Education':p4,
#     'Self_Employed':p5,
#     'ApplicantIncome':p6,
#     'CoapplicantIncome':p7,
#     'LoanAmount':p8,
#     'Loan_Amount_Term':p9,
#     'Credit_History':p10,
#     'Property_Area':p11
# },index=[0])
#     result = model.predict(df)

#     if result == 1:
#         Label(master, text="Loan approved").grid(row=31)
#     else:
#         Label(master, text="Loan Not Approved").grid(row=31)
```

```
# master =Tk()
# master.title("Loan Status Prediction Using Machine Learning")
# label = Label(master,text = "Loan Status Prediction",bg = "black",
#                fg = "white").grid(row=0,columnspan=2)

# Label(master,text = "Gender [1:Male ,0:Female]").grid(row=1)
# Label(master,text = "Married [1:Yes,0:No]").grid(row=2)
# Label(master,text = "Dependents [1,2,3,4]").grid(row=3)
# Label(master,text = "Education").grid(row=4)
# Label(master,text = "Self_Employed").grid(row=5)
# Label(master,text = "ApplicantIncome").grid(row=6)
# Label(master,text = "CoapplicantIncome").grid(row=7)
# Label(master,text = "LoanAmount").grid(row=8)
# Label(master,text = "Loan_Amount_Term").grid(row=9)
# Label(master,text = "Credit_History").grid(row=10)
# Label(master,text = "Property_Area").grid(row=11)


# e1 = Entry(master)
# e2 = Entry(master)
# e3 = Entry(master)
# e4 = Entry(master)
# e5 = Entry(master)
# e6 = Entry(master)
# e7 = Entry(master)
# e8 = Entry(master)
# e9 = Entry(master)
# e10 = Entry(master)
# e11 = Entry(master)


# e1.grid(row=1,column=1)
# e2.grid(row=2,column=1)
# e3.grid(row=3,column=1)
# e4.grid(row=4,column=1)
# e5.grid(row=5,column=1)
# e6.grid(row=6,column=1)
# e7.grid(row=7,column=1)
# e8.grid(row=8,column=1)
# e9.grid(row=9,column=1)
# e10.grid(row=10,column=1)
# e11.grid(row=11,column=1)

# Button(master,text="Predict",command=show_entry).grid()

# mainloop()
```