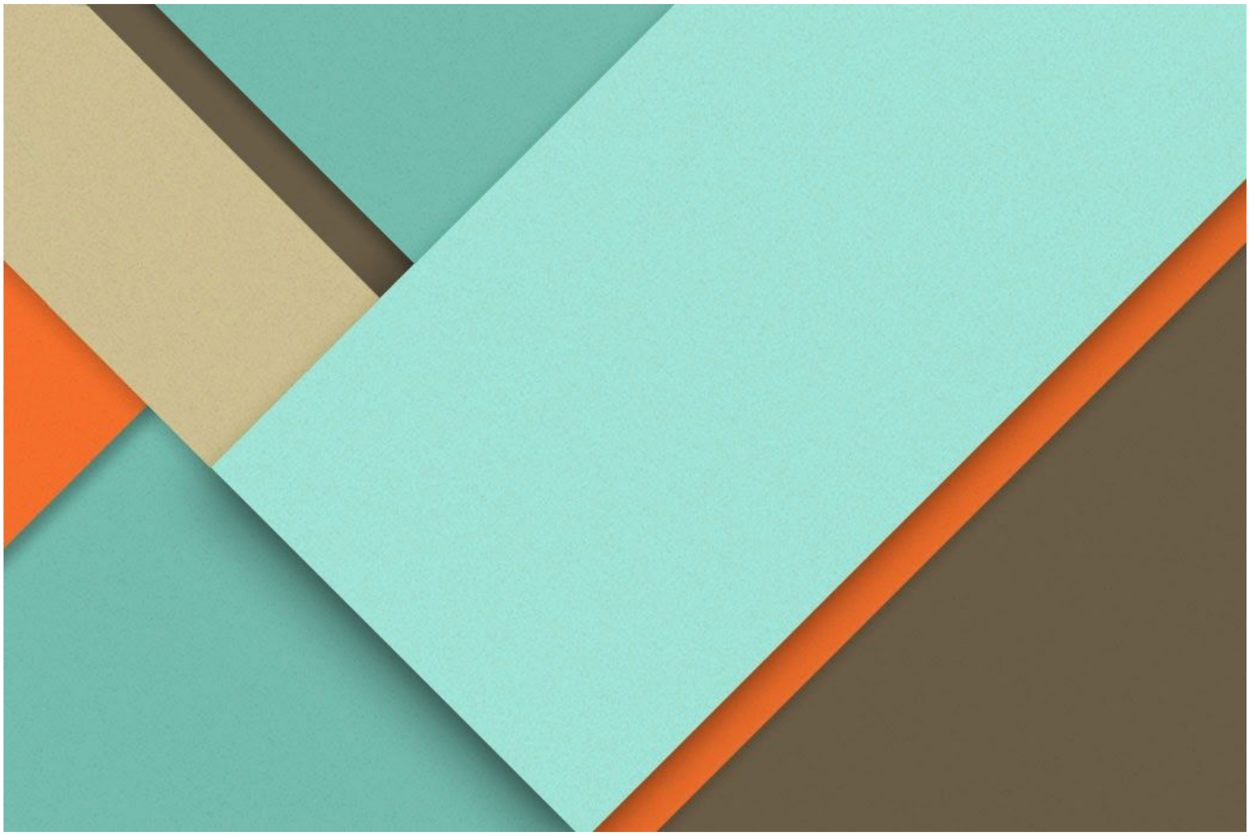

Machine Learning Project(SkillSanta)

Loan Prediction

Nikhil S A
ambigarnikhil@gmail.com
IIIT Dharwad
Kalaburagi, 6364077543

Table of contents:

EDA(Exploratory data analysis):

EDA is used for analyzing and investigating data sets and summarizing their main characteristics, often employing data visualization methods. It is basically understanding the dataset given to train the model.This can be achieved by using pandas module. Let us see how it works,

```
train=pd.read_csv("F:\MACHINE LEARNING\Project/finance_train_data.csv")
```

```
train.head(6)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	1.0

First we are reading the given training csv file , and then using head() method we are printing first five elements of the csv file and we can notice that dataset is not completely filled but it consists of some NaN elements(we can see first element in the LoanAmount column is NaN ie, Not a Number).We should either remove the entire row or fill that element with either mean, median, mode of that column.

```
train.dtypes
```

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

Using dtypes we will get to know about the type of data of each column.

```
train.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
```

Using info() method pandas we will get column name , number of non-null numbers , datatype of columns and memory usage.

```
train.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

describe() method gives the statistical analysis of the dataset.

```
train.corr()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
ApplicantIncome	1.000000	-0.116605	0.570909	-0.045306	-0.014715
CoapplicantIncome	-0.116605	1.000000	0.188619	-0.059878	-0.002056
LoanAmount	0.570909	0.188619	1.000000	0.039447	-0.008433
Loan_Amount_Term	-0.045306	-0.059878	0.039447	1.000000	0.001470
Credit_History	-0.014715	-0.002056	-0.008433	0.001470	1.000000

corr() method will return a number between 0 to 1,i.e. if it returns a number closer to 1 then we can conclude that both the variables are closely related to each other and vice-versa.

```
category=["Gender","Married",'Dependents','Loan_Amount_Term','Credit_History','Self_Employed','LoanAmount']
for i in category:
    print(train[i].isnull().value_counts())
```

```
False    601
True      13
Name: Gender, dtype: int64
False    611
True       3
Name: Married, dtype: int64
False    599
True      15
Name: Dependents, dtype: int64
False    600
True      14
Name: Loan_Amount_Term, dtype: int64
False    564
True      50
Name: Credit_History, dtype: int64
False    582
True      32
Name: Self_Employed, dtype: int64
False    592
True      22
Name: LoanAmount, dtype: int64
```

Using this piece of code we are printing how many elements are missing in each column.(True stands for missing elements i.e. NaN)

```
train['Gender'].fillna(train['Gender'].mode()[0], inplace=True)
train['Married'].fillna(train['Married'].mode()[0], inplace=True)
train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=True)
train['Credit_History'].fillna(train['Credit_History'].mode()[0], inplace=True)
train['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
#numerical
train['LoanAmount'].fillna(train['LoanAmount'].mean(), inplace=True)
```

Using fillna() method we are replacing the NaN with mode , mean of that column.

```
from sklearn.preprocessing import LabelEncoder
category= ['Gender','Married','Dependents','Education','Self_Employed','Property_Area','Loan_Status']
encoder= LabelEncoder()
for i in category:
    train[i] = encoder.fit_transform(train[i])
train.dtypes
```

```
Loan_ID               object
Gender                 int32
Married                int32
Dependents             int32
Education              int32
Self_Employed          int32
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area          int32
Loan_Status            int32
dtype: object
```

Using the above piece of code we are changing the object datatype of the required column to integer datatype, as the machine understands only 0 and 1's.

```
X=train[['Gender', 'Married', 'Dependents', 'Education','Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount'
X[0:5]
```

```
array([[1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 5.84900000e+03, 0.00000000e+00, 1.46412162e+02,
        3.60000000e+02, 1.00000000e+00, 2.00000000e+00],
       [1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 4.58300000e+03, 1.50800000e+03, 1.28000000e+02,
        3.60000000e+02, 1.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 3.00000000e+03, 0.00000000e+00, 6.60000000e+01,
        3.60000000e+02, 1.00000000e+00, 2.00000000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 1.00000000e+00,
        0.00000000e+00, 2.58300000e+03, 2.35800000e+03, 1.20000000e+02,
        3.60000000e+02, 1.00000000e+00, 2.00000000e+00],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 6.00000000e+03, 0.00000000e+00, 1.41000000e+02,
        3.60000000e+02, 1.00000000e+00, 2.00000000e+00]])
```

```
Y=train["Loan_Status"].values
Y[0:5]
```

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

To use scikit-learn library, we have to convert the Pandas data frame to a Numpy array.

```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

```
array([[ 0.47234264, -1.37208932, -0.73780632, -0.52836225, -0.39260074,
         0.07299082, -0.55448733,  0.        ,  0.2732313 ,  0.41173269,
         1.22329839],
       [ 0.47234264,  0.72881553,  0.25346957, -0.52836225, -0.39260074,
        -0.13441195, -0.03873155, -0.21927331,  0.2732313 ,  0.41173269,
        -1.31851281],
       [ 0.47234264,  0.72881553, -0.73780632, -0.52836225,  2.54711697,
        -0.39374734, -0.55448733, -0.957641  ,  0.2732313 ,  0.41173269,
         1.22329839],
       [ 0.47234264,  0.72881553, -0.73780632,  1.89264089, -0.39260074,
        -0.46206247,  0.2519796 , -0.31454656,  0.2732313 ,  0.41173269,
         1.22329839],
       [ 0.47234264, -1.37208932, -0.73780632, -0.52836225, -0.39260074,
         0.09772844, -0.55448733, -0.06445428,  0.2732313 ,  0.41173269,
         1.22329839]])
```

Data Standardization gives data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases.

Data Wrangling:

Data Wrangling is not used, as we are not merging any columns, not even grouping the columns and not even concatenating the other column to the dataframe.

Algorithm:

I have used two algorithms for predicting loan and they are:

1) KNN(K-th Nearest Neighbour)
2) Logistic Regression

KNN(K-th Nearest Neighbour):

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of K number of neighbors
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Logistic Regression:

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Model Training:

Logistic Regression model:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

```
LogisticRegression(C=0.01, solver='liblinear')
```

```
yhat = LR.predict(X_test)
yhat
```

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

```
import sklearn.metrics as metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
accuracy = metrics.accuracy_score(yhat,y_test)
print("Accuracy : %s" % "{0:.3%}".format(accuracy))
print("r2_score:",r2_score(y_test, yhat))
print("mse:",mean_squared_error(y_test, yhat))
```

```
Accuracy : 79.675%
r2_score: 0.0737951807228916
mse: 0.2032520325203252
```

Logistic Regression model has an accuracy of around 80%, the r2 value for the model is 0.0737 just because R2 is small doesn't mean that your model is bad or worthless of being interpreted. Even small R2 can have unique contributions in relation to your field of study. I think a model with small R2 that has a unique contribution may be more relevant than the one with large R2 without a unique contribution. And the model has an MSE of 0.203 which is pretty good.

KNN(K-th Nearest Neighbour) model:

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import sklearn.metrics as metrics
```

```
k = 7
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
KNeighborsClassifier(n_neighbors=7)
```

Why did we take K-value = 7.?

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

```
array([0.6504065 , 0.6097561 , 0.72357724, 0.72357724, 0.77235772,
       0.76422764, 0.78861789, 0.7804878 , 0.76422764])
```

```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.7886178861788617 with k= 7
```

```python
yhat = neigh.predict(X_test)
```

```python
import sklearn.metrics as metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
accuracy = metrics.accuracy_score(yhat,y_test)
print("Accuracy : %s" % "{0:.3%}".format(accuracy))
print("r2_score:",r2_score(y_test, yhat))
print("mse:",mean_squared_error(y_test, yhat))
```

```
Accuracy : 78.862%
r2_score: 0.0367469879518072
mse: 0.21138211382113822
```

KNN model has an accuracy of around 79%, the r2 value for the model is 0.0367 just because R2 is small doesn't mean that your model is bad or worthless of being interpreted. Even small R2 can have unique contributions in relation to your field of study. I think a model with small R2 that has a unique contribution may be more relevant than the one with large R2 without a unique contribution. And the model has an MSE of 0.211 which is pretty good.

INNOVATION:

I got a very good sort of knowledge from this project of loan prediction. I learnt many things in data preprocessing i.e. how to deal with missing value elements/NaN elements, Converting non-integer values into integer values.Till now I only saw model creation in classes but this was my first time creating my own model for predicting the loan_stuff, And I learnt many things throughout the ML sessions conducted by SkillSanta Team.