Sandeshkumar Mukesh Yadav **D15C**
Roll No. 61 **DS-1**

Aim: Introduction to Data science and Data preparation using Pandas steps.

- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren't useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- Standardization and normalization of columns

Steps:

1) Loading data in Pandas and extracting information about the dataset.

To load a file onto python for analysis, we need to make use of the pandas library. It gives us functionalities to read a CSV (Comma Separated Values) file and perform various functions on it.

Commands: import pandas as pd (Importing the pandas library onto Google Colab Notebook) df = pd.read_csv() (Mounts and reads the file in Python and assigns it to variable df for ease of use further)
(Note: Replace with the actual path of the file in "")

dataset.info(): This command gives all the information about the features (columns) of the dataset and the data type of each of these columns. It also gives a summary of all the values in the dataset.

Sandeshkumar Mukesh Yadav **D15C**
Roll No. 61 **DS-1**

```
import pandas as pd
import numpy as np
```

```
# loading the dataset to pandas df
dataset = pd.read_csv("/content/financial_risk_assessment.csv")
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    15000 non-null  int64
 1   Gender                 15000 non-null  object
 2   Education Level        15000 non-null  object
 3   Marital Status         15000 non-null  object
 4   Income                 12750 non-null  float64
 5   Credit Score           12750 non-null  float64
 6   Loan Amount            12750 non-null  float64
 7   Loan Purpose           15000 non-null  object
 8   Employment Status      15000 non-null  object
 9   Years at Current Job   15000 non-null  int64
 10  Payment History        15000 non-null  object
 11  Debt-to-Income Ratio   15000 non-null  float64
 12  Assets Value           12750 non-null  float64
 13  Number of Dependents   12750 non-null  float64
 14  City                   15000 non-null  object
 15  State                  15000 non-null  object
 16  Country                15000 non-null  object
 17  Previous Defaults      12750 non-null  float64
 18  Marital Status Change  15000 non-null  int64
 19  Risk Rating            15000 non-null  object
dtypes: float64(7), int64(3), object(10)
memory usage: 2.3+ MB
```

2) df.head(): As mentioned before, head function give us the first 5 rows of the dataset. This allows for the user to get an overview on what values are being listed in the dataset.

```
[ ] dataset.head()
```

	Age	Gender	Education Level	Marital Status	Income	Credit Score	Loan Amount	Loan Purpose	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	City	State	Country	Previous Defaults	Marital Status Change
0	49	Male	PhD	Divorced	72799.0	688.0	45713.0	Business	Unemployed	19	Poor	0.154313	120228.0	0.0	Port Elizabeth	AS	Cyprus	2.0	2
1	57	Female	Bachelor's	Widowed	NaN	690.0	33835.0	Auto	Employed	6	Fair	0.148920	55849.0	0.0	North Catherine	OH	Turkmenistan	3.0	2
2	21	Non-binary	Master's	Single	55687.0	600.0	36623.0	Home	Employed	8	Fair	0.362398	180700.0	3.0	South Scott	OK	Luxembourg	3.0	2
3	59	Male	Bachelor's	Single	26508.0	622.0	26541.0	Personal	Unemployed	2	Excellent	0.454964	157319.0	3.0	Robinhaven	PR	Uganda	4.0	2
4	25	Non-binary	Bachelor's	Widowed	49427.0	766.0	36528.0	Personal	Unemployed	10	Fair	0.143242	287140.0	NaN	New Heather	IL	Namibia	3.0	1

3) dataset.shape(): returns the dimensions of the dataset as a tuple (rows, columns), helping to understand its size.

```
[ ] dataset.shape
```

```
(15000, 20)
```

4) Describe the dataset

dataset.describe(): provides statistical summaries of numerical columns, including count, mean, standard deviation, min, max, and quartiles (25%, 50%, 75%).

```
dataset.describe()
```

	Age	Income	Credit Score	Loan Amount	Years at Current Job	Debt-to-Income Ratio	Assets Value	Number of Dependents	Previous Defaults	Marital Status Change
count	15000.000000	12750.000000	12750.000000	12750.000000	15000.000000	15000.000000	12750.000000	12750.00000	12750.000000	15000.000000
mean	43.452667	69933.398510	699.109098	27450.010902	9.476267	0.350438	159741.497176	2.02651	1.992471	0.998467
std	14.910732	29163.626207	57.229465	12949.940135	5.769707	0.143919	80298.115832	1.41130	1.416909	0.813782
min	18.000000	20005.000000	600.000000	5000.000000	0.000000	0.100004	20055.000000	0.00000	0.000000	0.000000
25%	31.000000	44281.500000	650.000000	16352.500000	4.000000	0.227386	90635.250000	1.00000	1.000000	0.000000
50%	43.000000	69773.000000	699.000000	27544.000000	9.000000	0.350754	159362.000000	2.00000	2.000000	1.000000
75%	56.000000	95922.750000	748.000000	38547.500000	15.000000	0.476095	228707.000000	3.00000	3.000000	2.000000
max	69.000000	119997.000000	799.000000	49998.000000	19.000000	0.599970	299999.000000	4.00000	4.000000	2.000000

If the parameter of include=”all” is included { df.describe(include=“all”)}, this includes even the non numeric values and gives some more information on fields such as count of unique values, top value, etc.

```
dataset.describe(include="all")
```

	Age	Gender	Education Level	Marital Status	Income	Credit Score	Loan Amount	Loan Purpose	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	City
count	15000.000000	15000	15000	15000	12750.000000	12750.000000	12750.000000	15000	15000	15000.000000	15000	15000.000000	12750.000000	12750.00000	15000
unique	NaN	3	4	4	NaN	NaN	NaN	4	3	NaN	4	NaN	NaN	NaN	10614
top	NaN	Non-binary	Bachelor's	Widowed	NaN	NaN	NaN	Personal	Employed	NaN	Good	NaN	NaN	NaN	East Michael
freq	NaN	5059	3829	3893	NaN	NaN	NaN	3771	5026	NaN	3822	NaN	NaN	NaN	19
mean	43.452667	NaN	NaN	NaN	69933.398510	699.109098	27450.010902	NaN	NaN	9.476267	NaN	0.350438	159741.497176	2.02651	NaN
std	14.910732	NaN	NaN	NaN	29163.626207	57.229465	12949.940135	NaN	NaN	5.769707	NaN	0.143919	80298.115832	1.41130	NaN
min	18.000000	NaN	NaN	NaN	20005.000000	600.000000	5000.000000	NaN	NaN	0.000000	NaN	0.100004	20055.000000	0.00000	NaN
25%	31.000000	NaN	NaN	NaN	44281.500000	650.000000	16352.500000	NaN	NaN	4.000000	NaN	0.227386	90635.250000	1.00000	NaN
50%	43.000000	NaN	NaN	NaN	69773.000000	699.000000	27544.000000	NaN	NaN	9.000000	NaN	0.350754	159362.000000	2.00000	NaN
75%	56.000000	NaN	NaN	NaN	95922.750000	748.000000	38547.500000	NaN	NaN	15.000000	NaN	0.476095	228707.000000	3.00000	NaN
max	69.000000	NaN	NaN	NaN	119997.000000	799.000000	49998.000000	NaN	NaN	19.000000	NaN	0.599970	299999.000000	4.00000	NaN

5) Dropping the columns

dataset.drop() is used to remove specified rows or columns from the dataset.

- dataset.drop(columns=['column_name']) → Drops a specific column.
- dataset.drop(index=[row_index]) → Drops a specific row.

```
# dropping the columns that aren't useful
cols = ['Marital Status', 'Marital Status Change', 'Loan Purpose','City','State']
df = dataset.drop(cols, axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    15000 non-null  int64
 1   Gender                 15000 non-null  object
 2   Education Level        15000 non-null  object
 3   Income                 12750 non-null  float64
 4   Credit Score           12750 non-null  float64
 5   Loan Amount            12750 non-null  float64
 6   Employment Status      15000 non-null  object
 7   Years at Current Job   15000 non-null  int64
 8   Payment History        15000 non-null  object
 9   Debt-to-Income Ratio   15000 non-null  float64
 10  Assets Value           12750 non-null  float64
 11  Number of Dependents   12750 non-null  float64
 12  Country                15000 non-null  object
 13  Previous Defaults      12750 non-null  float64
 14  Risk Rating            15000 non-null  object
dtypes: float64(7), int64(2), object(6)
memory usage: 1.7+ MB
```

Before Dropping:

Sandeshkumar Mukesh Yadav **D15C**
Roll No. 61 **DS-1**

```
[ ] dataset.shape
```

```
(15000, 20)
```

After Dropping:

```
df.shape
```

```
(15000, 15)
```

As observed here, the columns of *'Marital Status', 'Marital Status Change', 'Loan Purpose','City','State'* have been dropped.

6) Drop rows with maximum missing rows

df["missing_count"] = df.isnull().sum(axis=1)
max_missing = df["missing_count"].max()
Here the maximum missing count is 6. So to clean up some of the data, we will remove the rows with 4 or more missing values. df = df[df["missing_count"] < 4]
The above set of commands do the following function:
i) Create a column called missing_count where the sum of all the cells having null values is stored.
ii) The maximum value from this missing_count column is considered for deletion
iii) Finally, we update the dataset by keeping the rows which have missing values less than a particular value

```
df["missing_count"] = df.isnull().sum(axis=1)
max_missing = df["missing_count"].max()
print(df.head())

df = df[df["missing_count"] < 4]
df.shape
```

```
   Age      Gender Education Level   Income  Credit Score  Loan Amount  \
0   49        Male             PhD  72799.0         688.0      45713.0
1   57      Female      Bachelor's      NaN         690.0      33835.0
2   21  Non-binary        Master's  55687.0         600.0      36623.0
3   59        Male      Bachelor's  26508.0         622.0      26541.0
4   25  Non-binary      Bachelor's  49427.0         766.0      36528.0

  Employment Status  Years at Current Job Payment History  \
0        Unemployed                    19            Poor
1          Employed                     6            Fair
2          Employed                     8            Fair
3        Unemployed                     2       Excellent
4        Unemployed                    10            Fair

   Debt-to-Income Ratio  Assets Value  Number of Dependents       Country  \
0              0.154313      120228.0                   0.0        Cyprus
1              0.148920       55849.0                   0.0  Turkmenistan
2              0.362398      180700.0                   3.0    Luxembourg
3              0.454964      157319.0                   3.0        Uganda
4              0.143242      287140.0                   NaN       Namibia

   Previous Defaults Risk Rating  missing_count
0                2.0         Low              0
1                3.0      Medium              1
2                3.0      Medium              0
3                4.0      Medium              0
4                3.0         Low              1
(14909, 16)
```

Sandeshkumar Mukesh Yadav **D15C**
Roll No. 61 **DS-1**

To check the total missing values in each columns.
df.isnull().sum() is used to check for missing values (NaN) in a dataset. Here's how it works:

df.isnull() creates a DataFrame of the same shape as df, where each value is True if it's missing (NaN) and False otherwise.
.sum() then counts the number of True values (missing values) in each column.

7) Take care of the missing values

```
df.isnull().sum()
```

	0
Age	0
Gender	0
Education Level	0
Income	2189
Credit Score	2193
Loan Amount	2187
Employment Status	0
Years at Current Job	0
Payment History	0
Debt-to-Income Ratio	0
Assets Value	2196
Number of Dependents	2185
Country	0
Previous Defaults	2182
Risk Rating	0
missing_count	0

So, there are many missing values, hence performing the next step.

- To take care of the missing data that has not been removed, one of the 2 methods can be used: If the feature is of a numeric data type, we can use either mean, median or mode of the feature. If the data is normally distributed, use mean, if it is skewed, use median, and if many values are repeated, use mode.
- If the feature contains different categories, there are 2 ways. Either fill it with the mode of the column, or add a custom value such as “Data Unavailable”.

```
# handling the missing data
df.fillna({'Income':df['Income'].median()},inplace=True)
```

```
df.fillna({'Credit Score':df['Credit Score'].median()},inplace=True)
```

To check the columns with missing values, using the following command
df[df.isnull().any(axis=1)] filters and returns all rows that contain at least one missing (NaN) value.

```
missing_rows = df[df.isnull().any(axis=1)]
print(missing_rows)
```

```
Empty DataFrame
Columns: [Age, Gender, Education Level, Income, Credit Score, Loan Amount, Employment Status, Years at Current Job, Payment History, Debt-to-Income Ratio, Assets Value, Number of De
Index: []
```

8) Creating dummy variables

pd.get_dummies(df, columns=categorical_columns, prefix=categorical_columns, drop_first=False) is used to convert categorical variables into one-hot encoded format. This transformation helps machine learning models process categorical data.

Breaking Down the Code:
pd.get_dummies(df, columns=categorical_columns, prefix=categorical_columns, drop_first=False)

- Converts each categorical column into multiple binary (0/1) columns, representing unique categories.
- prefix=categorical_columns ensures that the new columns have meaningful names.
- drop_first=False keeps all categories (if True, it drops the first category to avoid multicollinearity).
- for col in categorical_columns: df_dummies[col] = df[col]

This restores the original categorical columns back into df_dummies, so the dataset now contains both original and encoded versions.

```python
categorical_columns = ['Risk Rating', 'Gender', 'Employment Status', 'Payment History']

df_dummies = pd.get_dummies(df, columns=categorical_columns, prefix=categorical_columns, drop_first=False)

for col in categorical_columns:
    df_dummies[col] = df[col]

print(df_dummies.head())
```

```
   Age Education Level   Income  Credit Score  Loan Amount  \
0   49             PhD  72799.0         688.0      45713.0
1   57      Bachelor's  69773.0         690.0      33835.0
2   21        Master's  55687.0         600.0      36623.0
3   59      Bachelor's  26508.0         622.0      26541.0
5   30             PhD  69773.0         717.0      15613.0

   Years at Current Job  Debt-to-Income Ratio   Assets Value  \
0                    19              0.154313  120228.000000
1                     6              0.148920   55849.000000
2                     8              0.362398  180700.000000
3                     2              0.454964  157319.000000
5                     5              0.295984  159741.497176

   Number of Dependents       Country  ...  Employment Status_Self-employed  \
0                   0.0        Cyprus  ...                            False
1                   0.0  Turkmenistan  ...                            False
2                   3.0    Luxembourg  ...                            False
3                   3.0        Uganda  ...                            False
5                   4.0       Iceland  ...                            False

   Employment Status_Unemployed  Payment History_Excellent  \
0                          True                      False
1                         False                      False
2                         False                      False
3                          True                       True
5                          True                      False

   Payment History_Fair  Payment History_Good  Payment History_Poor  \
0                 False                 False                  True
1                  True                 False                 False
2                  True                 False                 False
3                 False                 False                 False
5                  True                 False                 False

   Risk Rating      Gender  Employment Status  Payment History
0          Low        Male         Unemployed             Poor
1       Medium      Female           Employed             Fair
2       Medium  Non-binary           Employed             Fair
3       Medium        Male         Unemployed        Excellent
5       Medium  Non-binary         Unemployed             Fair
```

9) Detecting Outlier data

Using IQR Value:

In this method, we find the IQR value for the column; which is the difference between Q1 - 1.5 * IQR and Q3 + 1.5 * IQR. This is a standard that is followed, the factor 1.5 can be modified between 1 to 3 based on the requirement.

Command:

```python
Q1 = df['Data_Value'].quantile(0.25)
Q3 = df['Data_Value'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Data_Value'] < lower_bound) | (df['Data_Value'] > upper_bound)]
```

This method gives the outliers and hence can be removed.

Using Manual method:

Checking for Outlier data in Excel using different value ranges.

And then using the preprocessed data.

Sandeshkumar Mukesh Yadav **D15C**
Roll No. 61 **DS-1**

```
df.to_csv('financial_risk_preprocessed.csv', index=False)
```

```
[ ] cleaned_df = pd.read_csv('/content/financial_risk_preprocessed WITH DEL.csv')
```

10) Standardization and Normalization of columns

- StandardScaler: Standardizes features by removing the mean and scaling to unit variance.
- MinMaxScaler: Normalizes features to a fixed range (0 to 1 by default).

Standardize Column:
Using formula:
mean_value = df["Data_Value"].mean()
std_value = df["Data_Value"].std()
df["Standardized_Data_Value"] = (df["Data_Value"] - mean_value) / std_value
Using Library:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Standardized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])

Normalize column:
Method 1:
Formula min_val = df['Data_Value'].min()
max_val = df['Data_Value'].max()
df['Data_Value_Normalized'] = (df['Data_Value'] - min_val) / (max_val - min_val)
Method 2:
Scaler library from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Normalized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])

Here, the columns, Income, Credit Score, and Loan Amount are standardized and normalized

Sandeshkumar Mukesh Yadav **D15C**
Roll No. 61 **DS-1**

```
[ ] from sklearn.preprocessing import StandardScaler, MinMaxScaler


    standard_scaler = StandardScaler()
    min_max_scaler = MinMaxScaler()

    cleaned_df['Income'] = standard_scaler.fit_transform(cleaned_df[['Income']])
    cleaned_df['Credit Score'] = standard_scaler.fit_transform(cleaned_df[['Credit Score']])

    cleaned_df['Loan Amount'] = min_max_scaler.fit_transform(cleaned_df[['Loan Amount']])

    print(cleaned_df[['Income', 'Credit Score', 'Loan Amount']].head())
```

```
     Income  Credit Score  Loan Amount
0 -0.015390     -0.209478     0.000914
1 -0.017087     -0.172080     0.000677
2 -0.024984     -1.854955     0.000732
3 -0.041344     -1.443586     0.000531
4 -0.017087      0.332782     0.000312
```

Sandeshkumar Mukesh Yadav **D15C**
Roll No. 61 **DS-1**

Conclusion:

In this experiment, we utilized **Pandas** and **Scikit-Learn** for data preprocessing, focusing on **normalization** and **standardization** to enhance dataset quality and efficiency. The dataset, provided in CSV format, was first imported into **Google Colab**, where we examined its structure using df.info() to retrieve details about its features and data types. The initial few rows were displayed using df.head() to get an overview of the data.

To ensure completeness, missing values were identified using df.isnull(). Various techniques were applied to handle them, such as **removing incomplete rows** or replacing missing values with statistical measures like the **mean, median, or mode**.

Next, categorical variables were encoded using **one-hot encoding (dummy variables)**, making them suitable for machine learning models. Outliers were detected and managed using the **Interquartile Range (IQR) method**, where values deviating significantly from the **first (Q1) and third quartile (Q3) thresholds** were removed. Additionally, **Excel** was used for manual data refinement to minimize the impact of extreme values on model performance.

For feature scaling, **StandardScaler** was applied to numerical columns like **Income** and **Credit Score**, ensuring they were standardized with a **mean of zero and unit variance**, making them suitable for models that assume normally distributed data. Meanwhile, **MinMaxScaler** was used to scale **Loan Amount** between **0 and 1**.

By implementing these preprocessing steps, the dataset was effectively cleaned and transformed, ensuring it was well-structured and optimized for further analysis.