EDA(Exploratory data analysis) - Preprocessing

Data science is often thought to consist of advanced statistical and machine learning techniques. However, there is another key component to any data science endeavor that is often undervalued or forgotten: exploratory data analysis (EDA). It is a classical and under-utilized approach that helps you quickly build a relationship with the new data.

It is always better to explore each data set using multiple exploratory techniques and compare the results.

The goal of this step is to understand the dataset, identify the missing values & outliers if any using visual and quantitative methods to get a sense of the story it tells.

Steps in Data Exploration and Preprocessing:

1. Identification of variables and data types
2. Analyzing the basic metrics
3. Non-Graphical Univariate Analysis
4. Graphical Univariate Analysis
5. Bivariate Analysis
   a. Correlation Analysis

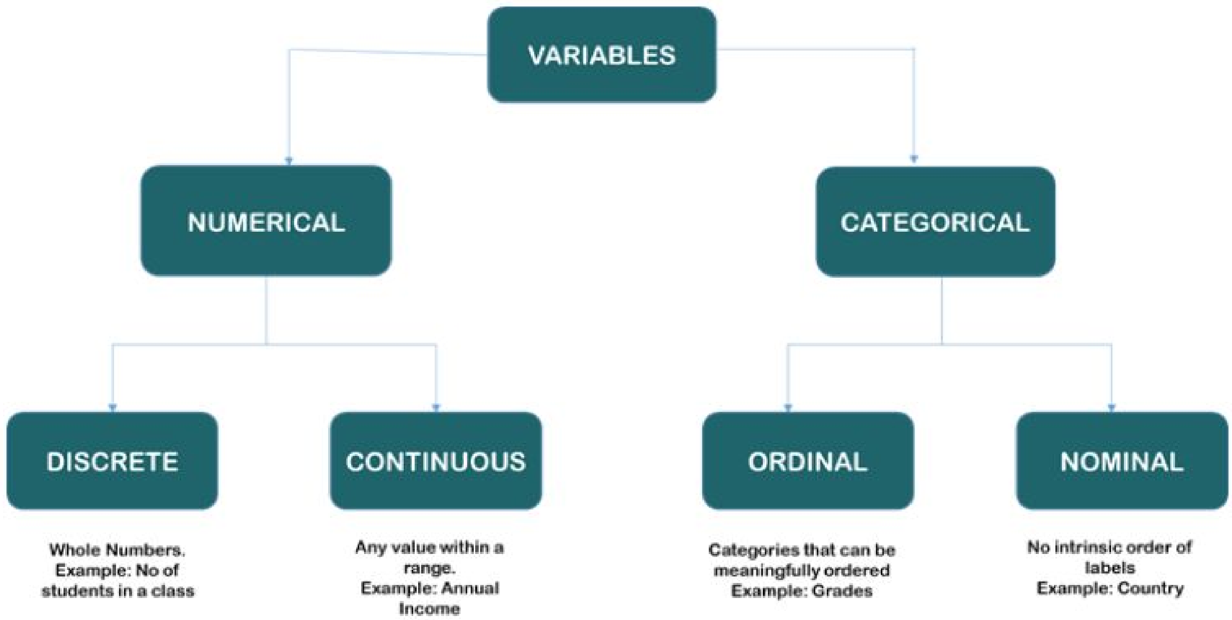
6. Variable transformations
7. Missing value treatment
8. Outlier treatment
9. Dimensionality Reduction(Feature selection)
    a. Principal Component Analysis (PCA)
    b. Linear Discriminant Analysis (LDA)
    c. Generalized Discriminant Analysis (GDA)

Here we are taking **train.csv** for EDA process

The sample dataset contains 29 columns and 233155 rows.

Variable identification:

The very first step in exploratory data analysis is to identify the type of variables in the dataset. Variables are of two types — Numerical and Categorical. They can be further classified as follows:



(Classification of Variable)

Once the type of variables is identified, the next step is to identify the Predictor (Inputs) and Target (output) variables.

In the above dataset, the numerical variables are,

Unique ID, disbursed_amount, asset_cost, ltv, Current_pincode_ID, PERFORM_CNS.SCORE, PERFORM_CNS.SCORE.DESCRIPTION, PRI.NO.OF.ACCTS, PRI.ACTIVE.ACCTS, PRI.OVERDUE.ACCTS, PRI.CURRENT.BALANCE, PRI.SANCTIONED.AMOUNT, PRI.DISBURSED.AMOUNT, NO.OF_INQUIRIES

And the categorical variables are,

branch_id, supplier_id, manufacturer_id, Date.of.Birth, Employment.Type, DisbursalDate, State_ID, Employee_code_ID, MobileNo_Avl_Flag, Aadhar_flag, PAN_flag, VoterID_flag, Driving_flag, Passport_flag, loan_default

The target value is ***loan_default*** and the rest 28 features can be assumed as the predictor variables.

Importing Libraries:

```
#importing libraries

import pandas as pd

import numpy as np

import matplotlib as plt

import seaborn as sns
```

Pandas library is a data analysis tool used for data manipulation, Numpy for scientific computing and Matplotlib & Seaborn for data visualization.

Importing Dataset:

```
train = pd.read_csv("train.csv")
```

Let’s import the dataset using read_csv method and assign it to the variable ‘train’.

Identification of data types:

The .dtypes method to identify the data type of the variables in the dataset.

```
train.dtypes
```

```
UniqueID                         int64
disbursed_amount                 int64
asset_cost                       int64
ltv                            float64
branch_id                        int64
supplier_id                      int64
manufacturer_id                  int64
Current_pincode_ID               int64
Date.of.Birth                   object
Employment.Type                 object
DisbursalDate                   object
State_ID                         int64
Employee_code_ID                 int64
```

A snippet of output for the above code

Both *Date.of.Birth* and *DisbursalDate* are of the object type. We have to convert it to DateTime type during data cleaning.

Size of the dataset:

We can get the size of the dataset using the .shape method

train.shape

Statistical Summary of Numeric Variables:

Pandas describe() is used to view some basic statistical details like count, percentiles, mean, std and maximum value of a data frame or a series of numeric values. As it gives the count of each variable, we can identify the missing values using this method.

```
train.describe()
```

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pincode_ID
count	233154.000000	233154.000000	2.331540e+05	233154.000000	233154.000000	233154.000000	233154.000000	233154.000000
mean	535917.573376	54356.993528	7.586507e+04	74.746530	72.936094	19638.635035	69.028054	3396.880247
std	68315.693711	12971.314171	1.894478e+04	11.456636	69.834995	3491.949566	22.141304	2238.147502
min	417428.000000	13320.000000	3.700000e+04	10.030000	1.000000	10524.000000	45.000000	1.000000
25%	476786.250000	47145.000000	6.571700e+04	68.880000	14.000000	16535.000000	48.000000	1511.000000
50%	535978.500000	53803.000000	7.094600e+04	76.800000	61.000000	20333.000000	86.000000	2970.000000
75%	595039.750000	60413.000000	7.920175e+04	83.670000	130.000000	23000.000000	86.000000	5677.000000
max	671084.000000	990572.000000	1.628992e+06	95.000000	261.000000	24803.000000	156.000000	7345.000000

8 rows × 37 columns

A snippet of output for the above code

Non-Graphical Univariate Analysis:

To get the count of unique values:

The value_counts() method in Pandas returns a series containing the counts of all the unique values in a column. The output will be in descending order so that the first element is the most frequently-occurring element.

Let’s apply value counts to *loan_default* column

train['loan_default'].value_counts()

```
0    182543
1     50611
Name: loan_default, dtype: int64
```

To get the list & number of unique values:

The nunique() function in Pandas returns a series with a number of distinct observations in a column.

train['branch_id'].nunique()

Similarly, unique() function of pandas returns the list of unique values in the dataset.

train['branch_id'].unique()

```
array([ 67,  78,  34, 130,  74,  11,   5,  20,  63,  48,  79,   3,  42,
       142,  36,  16, 146, 147,  65,   9,   1, 152,  29,  10,  70,  19,
         7,  85,  61,  17,   8, 153,  18, 162,  68,  72,  64,   2, 160,
       251, 103, 104, 120, 136,  77,  13, 138, 135,  73, 248,  15, 165,
        62,  76, 105, 249, 250, 255, 254,  82, 158, 159, 117, 202, 259,
       207,  35,  69,  97,  43, 257, 258, 260, 111,  66, 261, 101,  14,
       121, 217,  84, 100], dtype=int64)
```

Filtering based on Conditions:

Dataset can be filtered using different conditions, which can be implemented with the use of logical operators in python. For example, == (double equal to), ≤ (less than or equal to), ≥(greater than or equal to) etc

Let's apply the same to our dataset and filter out the column which has the *Employment.Type* as "Salaried"

train[(train['Employment.Type'] == "Salaried")]

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pincode_ID	Date.of.Birth	Employment.Type
0	420825	50578	58400	89.55	67	22807	45	1441	1/1/1984	Salaried
6	529269	46349	61500	76.42	67	22807	45	1502	1/6/1988	Salaried
7	510278	43894	61900	71.89	67	22807	45	1501	4/10/1989	Salaried
9	510980	52603	61300	86.95	67	22807	45	1492	1/6/1968	Salaried
11	486821	64769	74190	89.23	67	22807	45	1446	7/9/1984	Salaried
12	478647	53278	61330	89.68	67	22807	45	1497	1/6/1974	Salaried
13	479533	49478	57010	89.46	67	22807	45	1497	16-08-84	Salaried
15	600655	47549	61400	79.80	67	22807	45	1440	5/7/1994	Salaried
21	467015	31184	57110	56.91	67	22807	45	1498	29-02-84	Salaried
25	586411	55213	68600	83.09	67	22807	45	1494	1/1/1986	Salaried
31	525983	46549	69518	69.05	67	22744	86	1480	23-05-90	Salaried
32	501823	57259	70100	82.74	67	22807	45	1497	1/6/1966	Salaried

A snippet of output for the above code

Now let’s filter out the records based on two conditions using the AND (&) operator.

```
train[(train['Employment.Type'] == "Salaried") & (train['branch_id'] == 100)]
```

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pincode_ID	Date.of.Birth	Employment.Type
192434	620818	58259	77933	76.99	100	18731	86	644	15-02-68	Salaried
192436	433804	56259	65761	88.20	100	18731	86	631	1/1/1984	Salaried
192437	648534	59213	68817	88.64	100	20571	86	638	10/8/1984	Salaried
192439	627548	74079	103777	73.23	100	21335	51	656	5/6/1988	Salaried
192445	530872	61213	75321	83.64	100	24273	86	650	10/8/1969	Salaried
192446	587546	53303	69792	78.81	100	18731	86	636	1/1/1976	Salaried
192447	648979	24141	66637	39.02	100	18731	86	664	15-03-87	Salaried
192449	642808	52303	70187	76.94	100	18731	86	629	17-09-92	Salaried
192452	617778	48349	60820	82.21	100	18731	86	662	1/1/1993	Salaried
192453	625232	41210	70944	60.61	100	20571	86	630	9/7/1977	Salaried
192454	512984	23074	37816	63.47	100	21335	51	646	1/1/1956	Salaried
192455	637647	62213	72483	88.30	100	18731	86	629	1/1/1961	Salaried
192457	598604	41394	68817	62.48	100	18731	86	641	15-11-85	Salaried

A snippet of output for the above code

You can try ou the same example using the OR operator (|) as well.

Finding null values:

When we import our dataset from a CSV file, many blank columns are imported as null values into the Data Frame which can later create problems while operating that data frame. Pandas isnull() method is used to check and manage NULL values in a data frame.

```
train.apply(lambda x: sum(x.isnull()),axis=0)
```

```
UniqueID                    0
disbursed_amount            0
asset_cost                  0
ltv                         0
branch_id                   0
supplier_id                 0
manufacturer_id             0
Current_pincode_ID          0
Date.of.Birth               0
Employment.Type          7661
DisbursalDate               0
State_ID                    0
Employee_code_ID            0
MobileNo_Avl_Flag           0
Aadhar_flag                 0
PAN_flag                    0
VoterID_flag                0
Driving_flag                0
Passport_flag               0
```

A snippet of output for the above code

A snippet of output for the above code

We can see that there are 7661 missing records in the column '*Employment.Type*'. These missing records should be either deleted or imputed in the data preprocessing stage. I will talk about different ways to handle missing values in detail in my next article.

Data Type Conversion using to_datetime() and astype() methods:

Pandas astype() method is used to change the data type of a column. to_datetime() method is used to change particularly to DateTime type. When the data frame imported from a CSV file, the data type of the columns are set automatically, which many times is not what it actually should have. For example, in the above dataset, *Date.of.Birth* and *DisbursalDate* are both set as object type but they should be DateTime.

Example of to_datetime():

```
train['Date.of.Birth']= pd.to_datetime(train['Date.of.Birth'])
```

Example of astype():

```
train['ltv'] = train['ltv'].astype('int64')
```

Graphical Univariate Analysis:

Histogram:

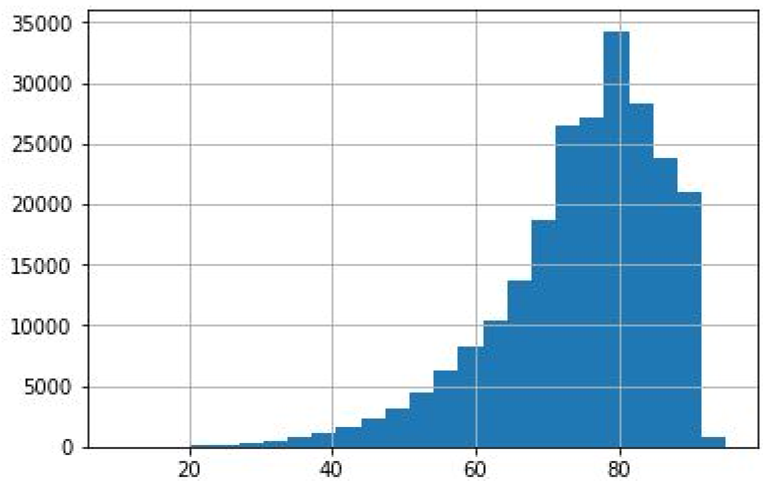
Histograms are one of the most common graphs used to display numeric data. Histograms two important things we can learn from a histogram:

1. distribution of the data — Whether the data is normally distributed or if it's skewed (to the left or right)
2. To identify outliers — Extremely low or high values that do not fall near any other data points.

Lets plot histogram for the *'ltv'* feature in our dataset

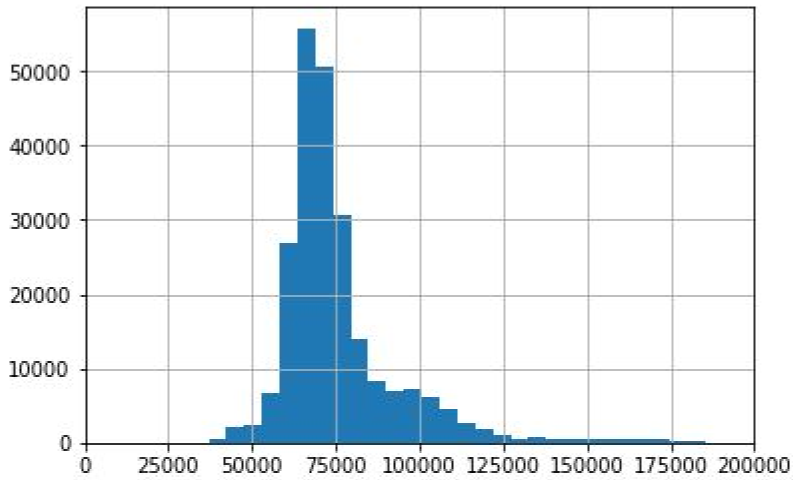
```
train['ltv'].hist(bins=25)
```

<matplotlib.axes._subplots.AxesSubplot at 0xf57dff0>



Here, the distribution is skewed to the left.

train['asset_cost'].hist(bins=200)



The above one is a normal distribution with a few outliers in the right end.

Box Plots:

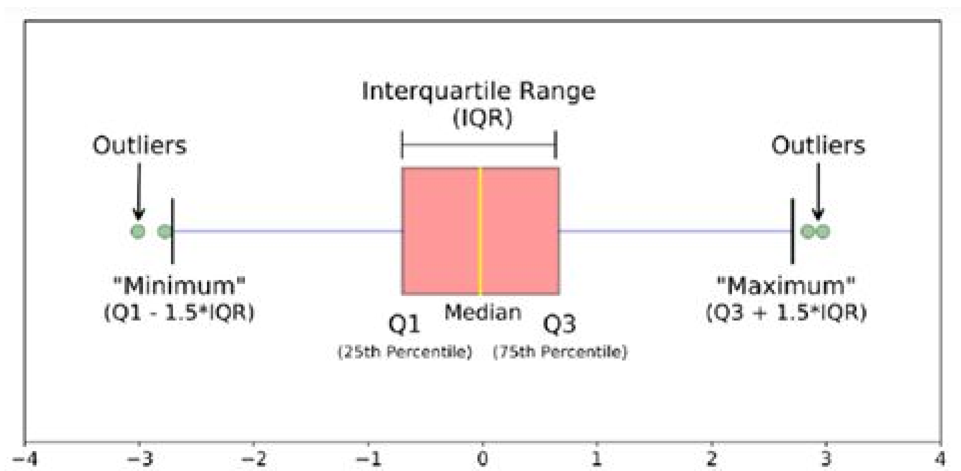
A Box Plot is the visual representation of the statistical summary of a given data set.

The Summary includes:

- Minimum
- First Quartile
- Median (Second Quartile)
- Third Quartile

- Maximum

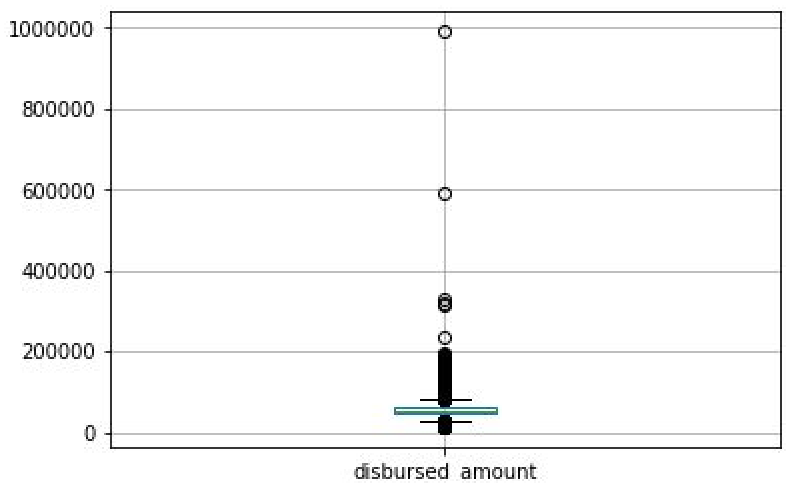
It is also used to identify the outliers in the dataset.



Example:

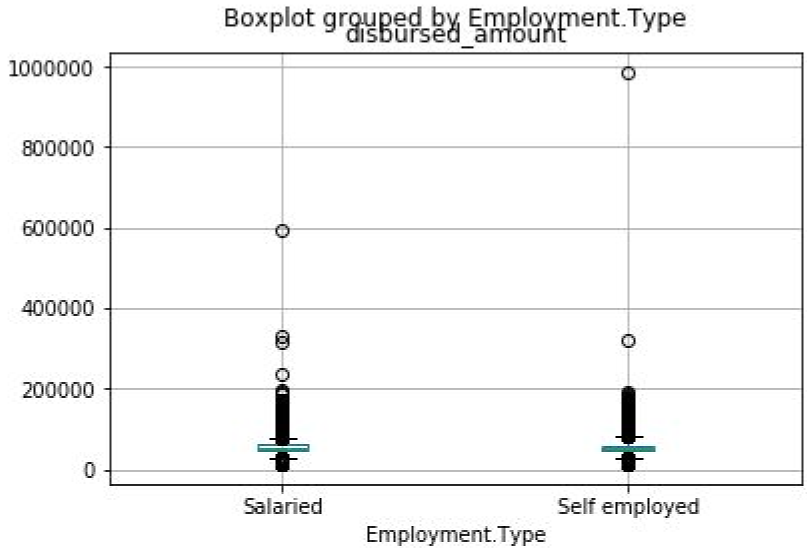
print(train.boxplot(column='disbursed_amount'))

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



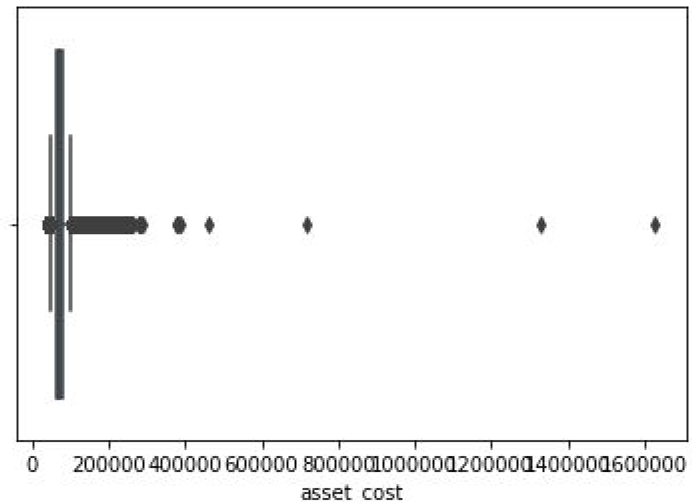
Here we can see that the mean is around 50000. There are also few outliers at 60000 and 1000000 which should be treated in the preprocessing stage.

train.boxplot(column='disbursed_amount', by = 'Employment.Type')




```
sns.boxplot(x=train['asset_cost'])
```

<matplotlib.axes._subplots.AxesSubplot at 0xf5a2bd0>

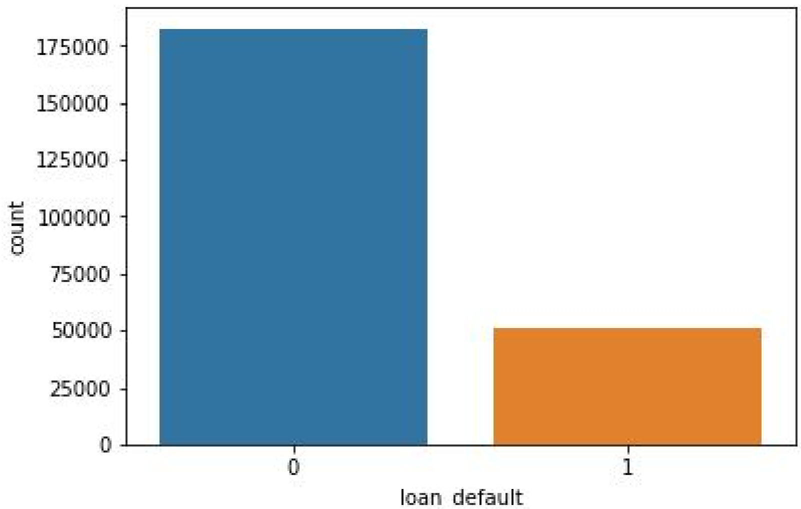


Count Plots:

A count plot can be thought of as a histogram across a categorical, instead of numeric, variable. It is used to find the frequency of each category.

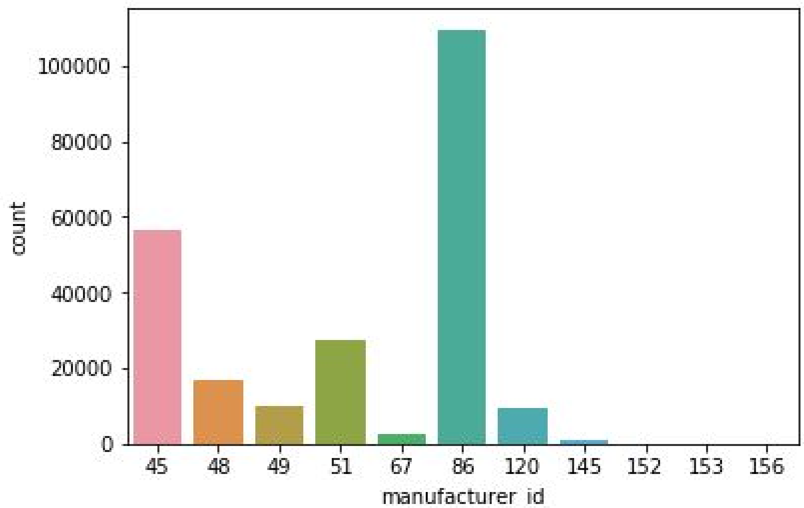
sns.countplot(train.loan_default)

```
<matplotlib.axes._subplots.AxesSubplot at 0x12626f10>
```



sns.countplot(train.manufacturer_id)

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b2fd070>
```



Here we can see that category “86” is dominating over the other categories.

These are the basic, initial steps in exploratory data analysis.

Bivariate analysis

Bivariate analysis is the simultaneous analysis of two variables (attributes). It explores the concept of relationship between two variables, whether there exists an association and the strength of this association, or whether there are differences between two variables and the significance of these differences. There are three types of bivariate analysis.

Linear Correlation

Linear correlation quantifies the strength of a linear relationship between two numerical variables. When there is no correlation between two variables, there is no tendency for the values of one quantity to increase or decrease with the values of the second quantity.

$$r = \frac{Covar(x,y)}{\sqrt{Var(x)Var(y)}}$$

$$Covar(x,y) = \frac{\sum(x-\bar{x})(y-\bar{y})}{n}$$

$$Var(x) = \frac{\sum(x-\bar{x})^2}{n}$$

$$Var(y) = \frac{\sum(y-\bar{y})^2}{n}$$

$r : Linear\ Correlation$

$Covar : Covariance$

$Var : Variance$

r only measures the strength of a linear relationship and is always between -1 and 1 where -1 means perfect negative linear correlation and +1 means perfect positive linear correlation and zero means no linear correlation.

Temperature	83	64	72	81	70	68	65	75	71	85	80	72	69	75
Humidity	86	65	90	75	96	80	70	80	91	85	90	95	70	70

	Variance	Covariance	Correlation
Temperature	40.10	19.78	**0.32**
Humidity	98.23		

There is a weak linear correlation between Temperature and Humidity.