1) Step-by-step Data Preprocessing & EDA

1.1) Dataset

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
import json
```

```
import pandas as pd
```

```
df = pd.read_json('modcloth_final_data.json', lines=True)
```

```
df
```

	item_id	waist	size	quality	cup size	hips	bra size	category	bust	height	user_name	length	fit	user_id	shoe size	shoe width	r
0	123373	29.0	7	5.0	d	38.0	34.0	new	36	5ft 6in	Emily	just right	small	991571	NaN	NaN	
1	123373	31.0	13	3.0	b	30.0	36.0	new	NaN	5ft 2in	sydneybraden2001	just right	small	587883	NaN	NaN	
2	123373	30.0	7	2.0	b	NaN	32.0	new	NaN	5ft 7in	Ugggh	slightly long	small	395665	9.0	NaN	
3	123373	NaN	21	5.0	dd/e	NaN	NaN	new	NaN	NaN	alexmeyer626	just right	fit	875643	NaN	NaN	
4	123373	NaN	18	5.0	b	NaN	36.0	new	NaN	5ft 2in	dberrones1	slightly long	small	944840	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
82785	807722	NaN	8	4.0	b	NaN	36.0	outerwear	NaN	5ft 8in	Jennifer	just right	fit	727820	8.5	average	
82786	807722	NaN	12	5.0	ddd/f	NaN	34.0	outerwear	NaN	5ft 5in	Kelli	slightly long	small	197040	NaN	NaN	
82787	807722	NaN	12	5.0	dddd/g	36.0	32.0	outerwear	NaN	5ft 4in	elacount	just	fit	102493	NaN	NaN	

```
df.shape
```

```
(82790, 18)
```

```
df.head()
```

	item_id	waist	size	quality	cup size	hips	bra size	category	bust	height	user_name	length	fit	user_id	shoe size	shoe width	review_s
0	123373	29.0	7	5.0	d	38.0	34.0	new	36	5ft 6in	Emily	just right	small	991571	NaN	NaN	
1	123373	31.0	13	3.0	b	30.0	36.0	new	NaN	5ft 2in	sydneybraden2001	just right	small	587883	NaN	NaN	
2	123373	30.0	7	2.0	b	NaN	32.0	new	NaN	5ft 7in	Ugggh	slightly	small	395665	9.0	NaN	

1.2) EDA - Exploratory Data Analysis¶

```
df.columns
```

```
Index(['item_id', 'waist', 'size', 'quality', 'cup size', 'hips', 'bra size',
       'category', 'bust', 'height', 'user_name', 'length', 'fit', 'user_id',
       'shoe size', 'shoe width', 'review_summary', 'review_text'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82790 entries, 0 to 82789
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   item_id         82790 non-null  int64  
 1   waist           2882 non-null   float64
 2   size            82790 non-null  int64  
 3   quality         82722 non-null  float64
 4   cup size        76535 non-null  object 
 5   hips            56064 non-null  float64
 6   bra size        76772 non-null  float64
 7   category        82790 non-null  object 
 8   bust            11854 non-null  object 
 9   height          81683 non-null  object 
 10  user_name       82790 non-null  object 
 11  length          82755 non-null  object 
 12  fit             82790 non-null  object 
 13  user_id         82790 non-null  int64  
 14  shoe size       27915 non-null  float64
 15  shoe width      18607 non-null  object 
 16  review_summary  76065 non-null  object 
 17  review_text     76065 non-null  object 
dtypes: float64(5), int64(3), object(10)
memory usage: 11.4+ MB
```

Looking at the percentage of missing values per column

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

```
item_id             0
waist           79908
size                0
quality            68
cup size         6255
hips            26726
bra size         6018
category            0
bust            70936
height           1107
user_name           0
length             35
fit                 0
user_id             0
shoe size       54875
shoe width      64183
review_summary   6725
review_text      6725
dtype: int64
```

```
df.shape
```

```
(82790, 18)
```

```
(df.isnull().sum()/82790)*100
```

```
item_id          0.000000
waist           96.518903
size             0.000000
quality          0.082136
cup size         7.555260
hips            32.281677
bra size         7.268994
category         0.000000
```

```
bust              85.681846
height             1.337118
user_name          0.000000
length             0.042276
fit                0.000000
user_id            0.000000
shoe size         66.282160
shoe width        77.525063
review_summary     8.122962
review_text        8.122962
dtype: float64
```

```
missing_data = pd.DataFrame({'Total_missing': df.isnull().sum(),
                             'Percentage_of_missing': (df.isnull().sum()/82790)*100})
```

```
missing_data
```

	Total_missing	Percentage_of_missing
item_id	0	0.000000
waist	79908	96.518903
size	0	0.000000
quality	68	0.082136
cup size	6255	7.555260
hips	26726	32.281677
bra size	6018	7.268994
category	0	0.000000
bust	70936	85.681846
height	1107	1.337118
user_name	0	0.000000
length	35	0.042276
fit	0	0.000000
user_id	0	0.000000
shoe size	54875	66.282160
shoe width	64183	77.525063
review_summary	6725	8.122962
review_text	6725	8.122962

Statistical description of numerical variables

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
item_id	82790.0	469325.229170	213999.803314	123373.0	314980.00	454030.0	658440.00	807722.0
waist	2882.0	31.319223	5.302849	20.0	28.00	30.0	34.00	50.0
size	82790.0	12.661602	8.271952	0.0	8.00	12.0	15.00	38.0
quality	82722.0	3.949058	0.992783	1.0	3.00	4.0	5.00	5.0
hips	56064.0	40.358501	5.827166	30.0	36.00	39.0	43.00	60.0
bra size	76772.0	35.972125	3.224907	28.0	34.00	36.0	38.00	48.0
user_id	82790.0	498849.564718	286356.969459	6.0	252897.75	497913.5	744745.25	999972.0
shoe size	27915.0	8.145818	1.336109	5.0	7.00	8.0	9.00	38.0

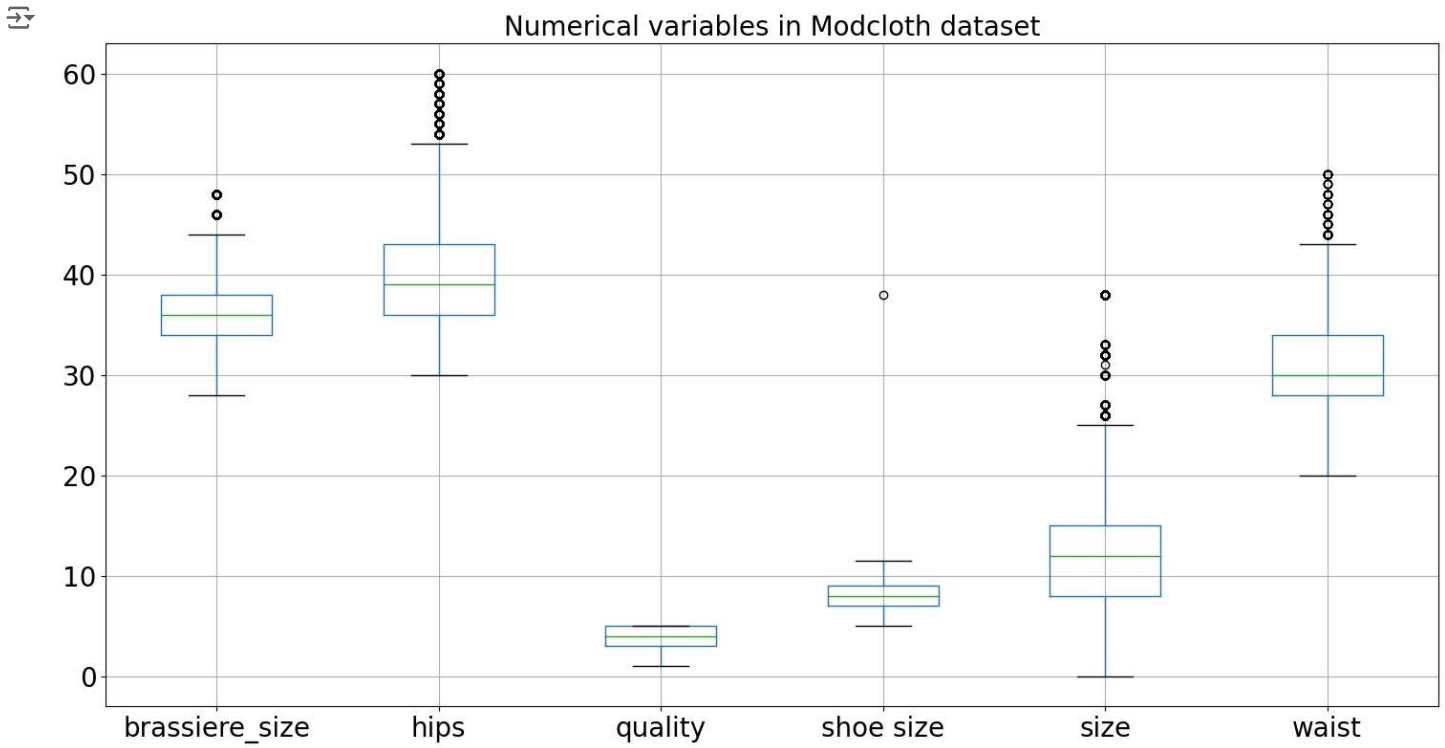
Boxplot of numerical variables

```
import matplotlib.pyplot as plt
```

```
df.rename(columns={'bra size': 'brassiere_size'}, inplace=True)
```

```
num_cols = ['brassiere_size','hips','quality','shoe size','size','waist']
```

```
plt.figure(figsize=(18,9))
df[num_cols].boxplot()
plt.title("Numerical variables in Modcloth dataset", fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```



Handling Outliers

shoe_size: We can clearly see that the single maximum value of shoe size (38) is an outlier and we should ideally remove that row or handle that outlier value. Let's take a look at that entry in our data.

brassiere_size: We can take a look at the top 8 bra-sizes (we can see that boxplot shows 2 values as outliers, as per the IQR- Inter-Quartile Range).

```
df.sort_values(by=['brassiere_size'], ascending=False).head(8)
```

	item_id	waist	size	quality	cup size	hips	brassiere_size	category	bust	height	user_name	length	fit	user_id	shoe size	s w
25059	391519	NaN	12	4.0	dd/e	NaN	48.0	dresses	NaN	5ft 7in	Jenna	slightly short	fit	756889	NaN	
77484	773564	NaN	38	5.0	dd/e	NaN	48.0	new	NaN	5ft 8in	el from oz	just right	fit	850421	NaN	
49818	538258	NaN	32	4.0	b	54.0	48.0	tops	NaN	5ft 5in	bri.m.baltes	slightly short	fit	18320	NaN	
61764	657242	NaN	26	3.0	dddd/g	51.0	48.0	new	NaN	5ft 9in	Kristen	just right	small	556885	NaN	
1281	126885	NaN	38	4.0	dd/e	NaN	48.0	new	NaN	5ft 9in	katrinafeil	just right	fit	586645	NaN	
41332	454030	NaN	12	5.0	k	60.0	48.0	tops	NaN	7ft 11in	bandnerd2522	just	fit	810539	11.5	ave

Joint Distribution of brassiere_size vs size¶

```
print(df.columns)
```

```
Index(['item_id', 'waist', 'size', 'quality', 'cup size', 'hips',
       'brassiere_size', 'category', 'bust', 'height', 'user_name', 'length',
       'fit', 'user_id', 'shoe size', 'shoe width', 'review_summary',
       'review_text'],
      dtype='object')
```

```
import seaborn as sns
```

```
data=df[['brassiere_size','size']]
```

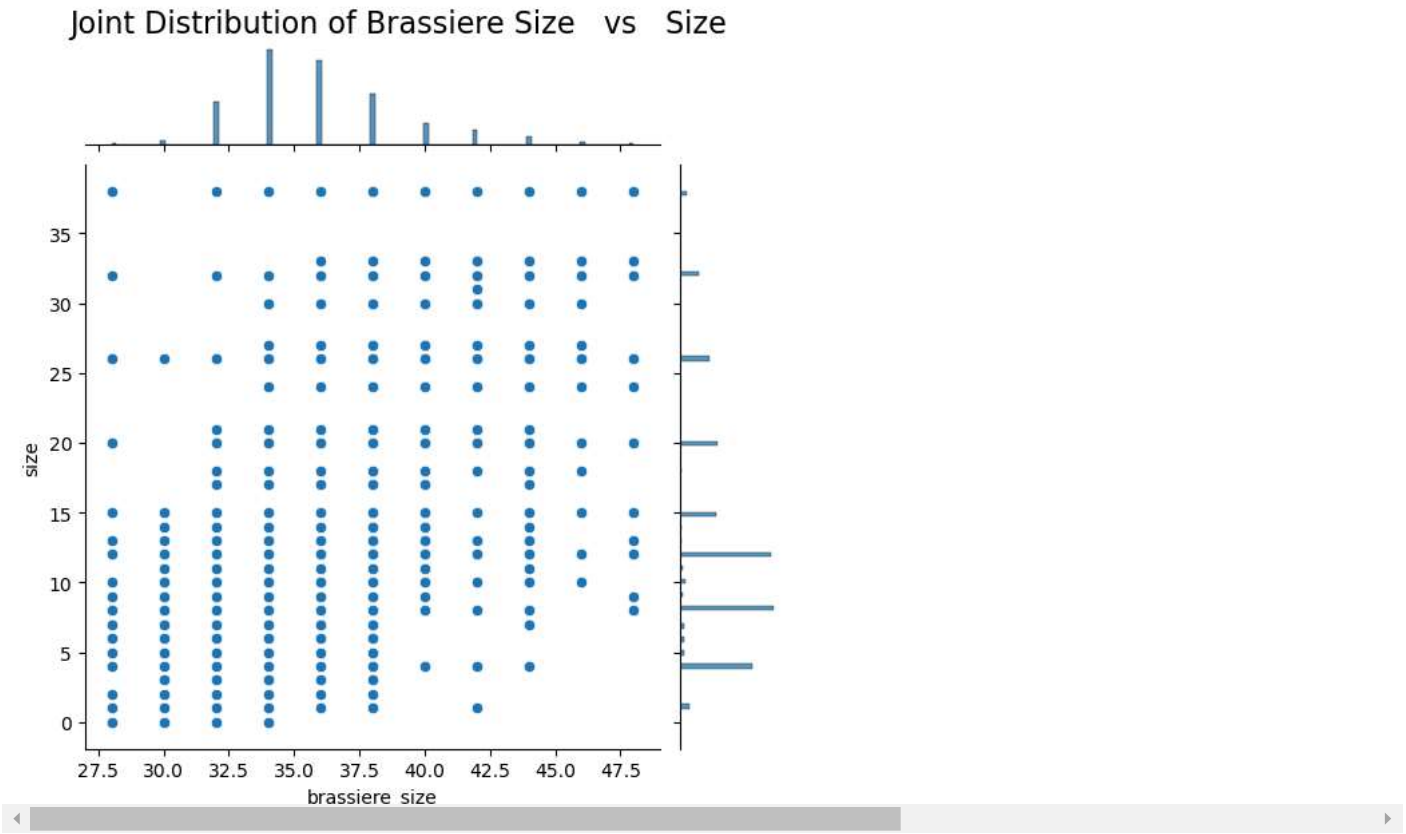
```
data
```

	brassiere_size	size
0	34.0	7
1	36.0	13
2	32.0	7
3	NaN	21
4	36.0	18
...	...	...
82785	36.0	8
82786	34.0	12
82787	32.0	12
82788	NaN	12
82789	32.0	4

82790 rows × 2 columns

```
plt.figure(figsize=(18,8))
sns.jointplot(x='brassiere_size', y='size', data=df, kind='scatter', height=6)
plt.suptitle("Joint Distribution of Brassiere Size   vs   Size", y=1.02, fontsize=16)
plt.show()
```

```
<Figure size 1800x800 with 0 Axes>
```



We can't see any significant deviation from usual behavior for bra-size, infact for all other numerical variables as well- we can expect the 'apparent' outliers, from the boxplot, to behave similarly. Now, we 'll head to preprocessing the dataset for suitable visualizations.

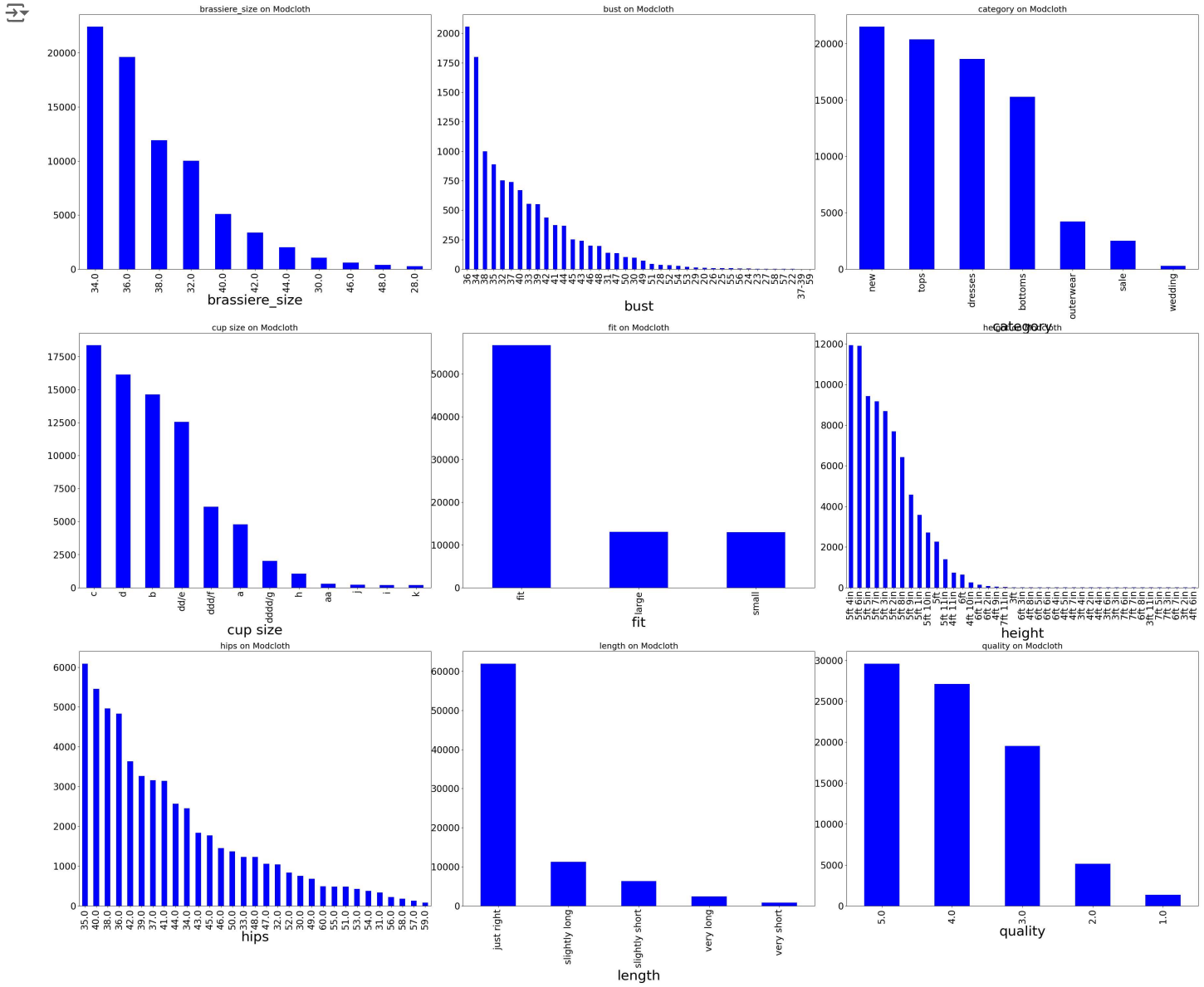
1.3) Data Cleaning & Pre-processing

Let's handle the variables and change the dtype to the appropriate type for each column. We define a function first for creating the distribution plot of different variables. Here, is the initial distribution of features.

Initial Distribution of features¶

```
def plot_distribution(col, ax):
    df[col][df[col].notnull()].value_counts().plot(kind='bar', facecolor='b', ax=ax)
    ax.set_xlabel('{}'.format(col), fontsize=30)
    ax.set_title("{} on Modcloth".format(col), fontsize= 18)
    ax.tick_params(axis='x', labelsize=20)  # Increase font size for x-axis ticks
    ax.tick_params(axis='y', labelsize=20)
    return ax
```

```
f, ax = plt.subplots(3,3, figsize = (35,15))
f.tight_layout(h_pad=12, w_pad=4, rect=[0, 0.07, 1, 1.93])
cols = ['brassiere_size','bust', 'category', 'cup size', 'fit', 'height', 'hips', 'length', 'quality']
k = 0
for i in range(3):
    for j in range(3):
        plot_distribution(cols[k], ax[i][j])
        k += 1
```



Step-by-step features processing:¶

brassiere_size:-

Although it looks numerical, it only ranges from 28 to 48, with most of the sizing lying around 34-38. It makes sense to convert this to categorical dtype. We'll fill the NA values into an 'Unknown' category. We can see above that most of the buyers have a bra-sizing of 34 or 36

bust:-

We can see by looking at the values which are not null, that bust should be an integer dtype. We also need to handle a special case where bust is given as - '37-39'. We'll replace the entry of '37-39' with the mean, i.e.- 38, for analysis purposes. Now we can safely convert the dtype to int. However, considering that roughly 86% of the bust data is missing, eventually it was decided to remove this feature.

category:-

none missing; change to dtype category

cup size:-

Change the dtype to category for this column. This col has around 7% missing values. Taking a look at the rows where this value is missing might hint us towards how to handle these missing values.

```
df.brassiere_size = df.brassiere_size.fillna('Unknown')
```

```
df.brassiere_size = df.brassiere_size.astype('category').cat.as_ordered()
```

```
df.at[37313,'bust'] = '38'
```

```
df.bust = df.bust.fillna(0).astype(int)
```

```
df.category = df.category.astype('category')
```

```
df[df['cup size'].isnull()].sample(20)
```

	item_id	waist	size	quality	cup size	hips	brassiere_size	category	bust	height	user_name	length	fit	user_id	shoe size	w
67510	699784	NaN	8	5.0	NaN	NaN	Unknown	bottoms	0	5ft 1in	Midori	slightly long	small	26026	NaN	
7203	152702	NaN	8	4.0	NaN	NaN	Unknown	new	0	5ft 8in	georgia.i.heath	just right	fit	910901	NaN	
61518	657081	NaN	8	5.0	NaN	NaN	Unknown	new	0	5ft 8in	renee	just right	large	333351	NaN	
55830	633619	NaN	4	5.0	NaN	NaN	Unknown	tops	0	5ft 5in	ali.kroeger	slightly long	large	233673	NaN	
8134	155530	NaN	15	3.0	NaN	NaN	Unknown	wedding	0	5ft 5in	katierawlings1	just right	fit	901241	NaN	
56534	643503	NaN	15	3.0	NaN	NaN	Unknown	tops	0	NaN	Bronie	slightly long	small	561658	6.5	ave
75194	757242	NaN	1	4.0	NaN	35.0	Unknown	bottoms	0	5ft 4in	kambersue	just right	large	923166	NaN	
75425	757242	NaN	8	3.0	NaN	NaN	Unknown	bottoms	0	5ft 5in	Detra	just right	fit	874288	8.5	ave
1596	126885	NaN	12	3.0	NaN	NaN	Unknown	new	0	5ft 10in	mjj	just right	small	979320	NaN	
52182	539980	NaN	8	2.0	NaN	NaN	Unknown	tops	0	5ft 5in	jessiepech	just right	fit	299854	NaN	
65547	693560	NaN	20	4.0	NaN	NaN	Unknown	bottoms	0	5ft 11in	lauraleighannd	slightly long	large	94851	11.0	ave
76711	760407	NaN	8	1.0	NaN	NaN	Unknown	bottoms	0	5ft 11in	jillygagnon	slightly long	large	681745	NaN	

We can't see anything glaring from the rows where this data is missing, however, as per the curator of the dataset- "Note that these datasets are highly sparse, with most products and customers having only a single transaction." It does point to that maybe these customers have not bought lingerie from modcloth yet and so modcloth does not have that data. So, it makes sense to fill these null values as 'Unknown'. From the prevalence of the values like dd/e, ddd/f, and dddd/g, we can assume these to be legit cup_sizes

```
df['cup size'].fillna('Unknown', inplace=True)
df['cup size'] = df['cup size'].astype('category').cat.as_ordered()

df.fit = df.fit.astype('category')
```

height- We need to parse the height column as currently it is a string object, of the form - Xft. Yin. It will make sense to convert height to cms. We also take a look at the rows where the height data is missing.

```
def get_cms(x):
    if type(x) == type(1.0):
        return
    try:
        return (int(x[0])*30.48) + (int(x[4:-2])*2.54)
    except:
        return (int(x[0])*30.48)
df.height = df.height.apply(get_cms)
```

```
df[df.height.isnull()].tail(20)
```

	item_id	waist	size	quality	cup size	hips	brassiere_size	category	bust	height	user_name	length	fit	user_id	shoe size
80685	796383	NaN	8	3.0	dd/e	NaN	36.0	outerwear	0	NaN	peewee8840518	just right	fit	457203	NaN
80745	797211	NaN	12	4.0	dd/e	NaN	34.0	sale	0	NaN	Ally	just right	small	224617	7.0
80889	800975	NaN	15	4.0	dddd/g	NaN	36.0	sale	0	NaN	Caitie	just right	small	234109	NaN
80907	800975	NaN	8	4.0	a	35.0	34.0	sale	0	NaN	blond4life05	just right	fit	463461	NaN
81126	803768	NaN	32	3.0	a	NaN	34.0	outerwear	0	NaN	carebearqt00	slightly short	fit	3991	10.5
81128	803768	NaN	20	5.0	Unknown	NaN	Unknown	outerwear	0	NaN	ethomann	just right	large	544966	9.0
81428	806479	NaN	12	5.0	c	NaN	42.0	outerwear	0	NaN	Amy	just right	fit	236299	NaN
81653	806479	NaN	20	5.0	Unknown	46.0	Unknown	outerwear	0	NaN	kjbozner	very long	large	32805	NaN
81655	806479	NaN	15	4.0	d	60.0	36.0	outerwear	0	NaN	Maria	just right	fit	952056	NaN
81658	806479	NaN	12	5.0	Unknown	NaN	Unknown	outerwear	0	NaN	michelle_reyes_m	just right	fit	606531	7.5
81699	806479	NaN	20	4.0	dd/e	NaN	38.0	outerwear	0	NaN	Jessie	just right	fit	323615	NaN
81711	806479	NaN	4	5.0	Unknown	NaN	Unknown	outerwear	0	NaN	cindobindo	just right	fit	750985	7.0
81861	806479	NaN	15	4.0	Unknown	NaN	Unknown	outerwear	0	NaN	lrn_sctt	slightly long	small	281250	8.0
81921	806479	NaN	12	4.0	Unknown	NaN	Unknown	outerwear	0	NaN	jessicamayblack	slightly long	large	894670	8.0
81964	806479	NaN	8	5.0	a	NaN	34.0	outerwear	0	NaN	Monica	just right	fit	213235	NaN

This filtering gives us interesting observations here:-

1) Some customers have given brassiere_size, cup_size data, whereas all other measurements are empty- possible first-time purchase at Modcloth for lingerie!

2) Some customers have given shoe_size and all other measurements are empty- possible first-time purchase at Modcloth for shoes! It leads us to saying that there are some first-time buyers in the dataset, also talked about by the authors of the data in [1]- about the sparsity of the data due to 1 transactions! Also, as we have no data about the height of these customers, it only makes sense to leave the missing values in the column as it is and possibly remove these rows for future statistical modeling. We have removed the corresponding rows.

1.4) Feature Engineering¶

Creating a new feature of first_time_user

Building on our observations above, it makes sense to identify the transactions which belong to first time users. We use the following logic to identify such transactions:-

a) If brassiere_size/cup_size have a value and height, hips, shoe_size, shoe_width and waist do not- it is a first time buyer of lingerie.

b) If shoe_size/shoe_width have a value and brassiere_size, cup_size, height, hips, and waist do not- it is a first time buyer of shoes.

c) If hips/waist have a value and brassiere_size, cup_size, height, shoe_size, and shoe_width do not- it is a first time buyer of a dress/tops

Below we will verify the above logic, with samples, before we create the new feature.

1. Looking at the few rows where either brassiere_size or cup_size exists, but no other measurements are available.

```
df[((df.brassiere_size != 'Unknown') | (df['cup size'] != 'Unknown')) & (df.height.isnull()) & (df.hips.isnull()) &
    (df['shoe size'].isnull()) & (df['shoe width'].isnull()) & (df.waist.isnull())].head(3)
```

	item_id	waist	size	quality	cup size	hips	brassiere_size	category	bust	height	user_name	length	fit	user_id	shoe size	shoe width
3	123373	NaN	21	5.0	dd/e	NaN	Unknown	new	0	NaN	alexmeyer626	just right	fit	875643	NaN	NaN

2. Looking at the few rows where either shoe_size or shoe_width exists, but no other measurements are available.

```
df[(df.brassiere_size== 'Unknown') & (df['cup size'] == 'Unknown') & (df.height.isnull()) & (df.hips.isnull()) &
    ((df['shoe size'].notnull()) | (df['shoe width'].notnull())) & (df.waist.isnull())].head(3)
```

	item_id	waist	size	quality	cup size	hips	brassiere_size	category	bust	height	user_name	length	fit	user_id	shoe size	w
553	125442	NaN	7	5.0	Unknown	NaN	Unknown	new	0	NaN	sharonpeporter	slightly long	fit	461540	7.0	ave

3. Looking at the few rows where either hips or waist exists, but no other measurements are available.

```
df[(df.brassiere_size == 'Unknown') & (df['cup size'] == 'Unknown') & (df.height.isnull()) & ((df.hips.notnull()) | (df.waist.notnull())) &
    (df['shoe size'].isnull()) & (df['shoe width'].isnull())].head(3)
```

	item_id	waist	size	quality	cup size	hips	brassiere_size	category	bust	height	user_name	length	fit	user_id	shoe size	shoe width
2364	131020	NaN	11	5.0	Unknown	39.0	Unknown	new	0	NaN	aislinnalysse	just right	fit	259712	NaN	NaN
3568	143287	NaN	12	5.0	Unknown	41.0	Unknown	new	38	NaN	a.stahle	just right	fit	300915	NaN	NaN

Adding a new column to the original data- first_time_user

Now we add a new column to the original data- first_time_user, which is a bool feature which indicates if a user, of a transaction, is a first-time user or not. This is based on the grounds that Modcloth has no previous information about the person, infact it is possible that the new user did multiple transactions in the first time!

```
lingerie = (((df.brassiere_size != 'Unknown') | (df['cup size'] != 'Unknown')) & (df.height.isnull()) & (df.hips.isnull()) &
    (df['shoe size'].isnull()) & (df['shoe width'].isnull()) & (df.waist.isnull()))
```

```
len(lingerie)
```

```
82790
```

```
shoe= ((df.brassiere_size == 'Unknown') & (df['cup size'] == 'Unknown') & (df.height.isnull()) & (df.hips.isnull()) &
    ((df['shoe size'].notnull()) | (df['shoe width'].notnull())) & (df.waist.isnull()))
```

```
len(shoe)
```

```
82790
```

```
dress= ((df.brassiere_size == 'Unknown') & (df['cup size'] == 'Unknown') & (df.height.isnull()) & ((df.hips.notnull()) | (df.waist.notnull()
    (df['shoe size'].isnull()) & (df['shoe width'].isnull()))
```

```
len(dress)
```

```
82790
```

```
df['first_time_user'] = (lingerie | shoe| dress)
```

```
df[df['first_time_user']!=False]
```

	item_id	waist	size	quality	cup size	hips	brassiere_size	category	bust	height	user_name	length	fit	user_id	shoe size
3	123373	NaN	21	5.0	dd/e	NaN	Unknown	new	0	NaN	alexmeyer626	just right	fit	875643	NaN
17	123373	NaN	15	4.0	dddd/g	NaN	36.0	new	0	NaN	Megan	just right	large	128353	NaN
43	123373	NaN	15	5.0	dd/e	NaN	36.0	new	0	NaN	sdhewey	just right	fit	670919	NaN
73	124124	NaN	20	4.0	c	NaN	38.0	new	0	NaN	Fiorella	just right	large	816504	NaN
131	124124	NaN	12	3.0	c	NaN	36.0	new	0	NaN	jmcrowder.1124	very short	fit	846091	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
82039	806856	NaN	12	4.0	Unknown	45.0	Unknown	outerwear	0	NaN	alm430	just right	fit	830508	NaN
82067	806856	NaN	4	5.0	Unknown	NaN	Unknown	outerwear	0	NaN	lindsayak	just right	fit	694263	7.5
82243	806856	NaN	12	5.0	dddd/g	NaN	36.0	outerwear	0	NaN	all_smilez17	just right	fit	95539	NaN
82583	806856	NaN	8	5.0	Unknown	NaN	Unknown	outerwear	0	NaN	tessa.a.forbes	just right	fit	882550	6.5
82609	806856	NaN	8	5.0	Unknown	NaN	Unknown	outerwear	0	NaN	prairygirl	just right	fit	270763	8.0

903 rows × 19 columns

```
df[df['first_time_user']!=False]['first_time_user']
```

```
3        True
17       True
43       True
73       True
131      True
         ...
82039    True
82067    True
82243    True
82583    True
82609    True
Name: first_time_user, Length: 903, dtype: bool
```

```
print("Total first time users: " + str(len(df[(lingerie | shoe | dress)].user_id.unique())))
```

```
Total first time users: 565
```

Handling hips column

hips- Hips column has a lot of missing values ~ 32.28%! We know this data would possibly be missing because Modcloth never got this data from the user most probably. We cannot remove such a significant chunk of the data, so we need another way of handling this feature. We will bin the data- on the basis of quartiles.

```
df.hips = df.hips.fillna(-1.0)
bins = [-5,0,31,37,40,44,75]
```

```
labels = ['Unknown','XS','S','M', 'L','XL']
df.hips = pd.cut(df.hips, bins, labels=labels)
```

```
df[['hips']].head(10)
```

	hips
0	M
1	XS
2	Unknown
3	Unknown
4	Unknown
5	L
6	Unknown
7	L
8	XL
9	L

Handling length column

length- There are only 35 missing rows in length, we'll take a look at these. We saw that most probably the customers did not leave behind the feedback or the data was corrupted in these rows. However, we should be able to impute these values using review related fields (if they are filled!). Or we could also simply choose to remove these rows. For the sake of this analysis, we will remove these rows.

```
missing_rows = df[df.length.isnull()].index
df.drop(missing_rows, axis = 0, inplace=True)
```

```
df[['length']].head(10)
```

	length
0	just right
1	just right
2	slightly long
3	just right
4	slightly long
5	just right
6	just right
7	just right
8	just right
9	just right

Handling quality

quality- There are only 68 missing rows in quality, we'll took a look at these. Similarly to length, the customers did not leave behind the feedback or the data was corrupted in these rows. We will remove these rows and convert the dtype to an ordinal variable (ordered categorical).

```
missing_rows = df[df.quality.isnull()].index
df.drop(missing_rows, axis = 0, inplace=True)
df.quality = df.quality.astype('category').cat.as_ordered()
```

```
df[['quality']].head(10)
```

	quality
0	5.0
1	3.0
2	2.0
3	5.0
4	5.0
5	5.0
6	1.0
7	5.0
8	4.0
9	5.0

Handling review_summary/ review_text

```
df.review_summary = df.review_summary.fillna('Unknown')
```

```
df[['review_summary']].head(10)
```

	review_summary
0	Unknown
1	Unknown
2	Unknown
3	Unknown
4	Unknown
5	Unknown
6	Unknown
7	Unknown
8	Unknown
9	Unknown

```
df.review_text = df.review_text.fillna('Unkown')
```

```
df[['review_text']].head(10)
```

	review_text
0	Unkown
1	Unkown
2	Unkown
3	Unkown
4	Unkown
5	Unkown
6	Unkown
7	Unkown
8	Unkown
9	Unkown

Handling shoe_size and shoe_width

shoe_size - Roughly 66.3% of the shoe_size data is missing. We will change the shoe_size into category dtype and fill the NA values as 'Unknown'.

shoe_width - Roughly 77.5% of the shoe_width data is missing. We will fill the NA values as 'Unknown'

```
from pandas.api.types import CategoricalDtype
shoe_widths_type = CategoricalDtype(categories=['Unknown','narrow','average','wide'], ordered=True)
```

```
df['shoe size'] = df['shoe size'].fillna('Unknown')
```

```
df[['shoe size']].head(10)
```

	shoe size
0	Unknown
1	Unknown
2	9.0
3	Unknown
4	Unknown
5	Unknown
6	Unknown
7	8.5
8	11.0
9	9.0

```
df['shoe size'] = df['shoe size'].astype('category').cat.as_ordered()
```

```
df['shoe width'] = df['shoe width'].fillna('Unknown')
```

```
df[['shoe width']].head(10)
```

	shoe width
0	Unknown
1	Unknown
2	Unknown
3	Unknown
4	Unknown
5	Unknown
6	Unknown
7	Unknown
8	wide
9	Unknown

```
df['shoe width'] = df['shoe width'].astype(shoe_widths_type)
```

```
df.drop(['waist', 'bust', 'user_name'], axis=1, inplace=True)
missing_rows = df[df.height.isnull()].index
df.drop(missing_rows, axis = 0, inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 81594 entries, 0 to 82789
Data columns (total 16 columns):
```

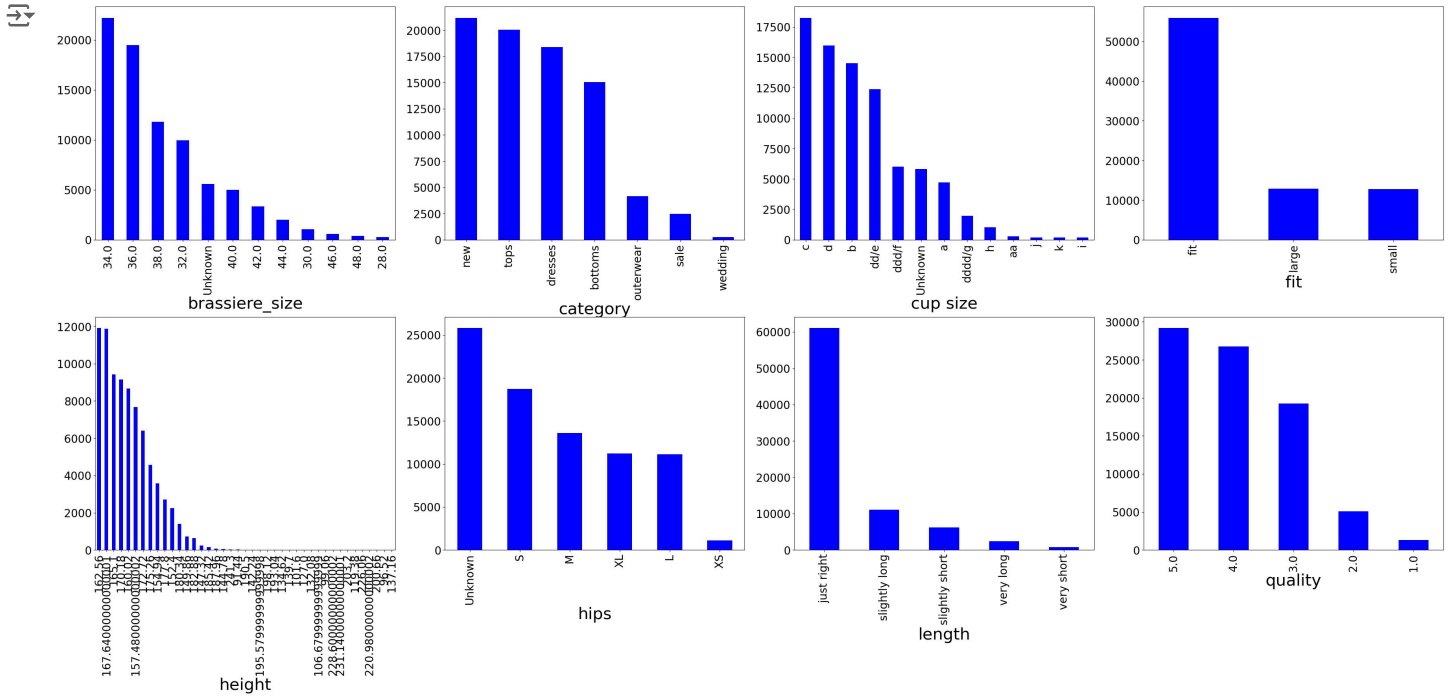
```
 #   Column           Non-Null Count  Dtype   
---  ------           --------------  -----   
 0   item_id          81594 non-null  int64   
 1   size             81594 non-null  int64   
 2   quality          81594 non-null  category
 3   cup size         81594 non-null  category
 4   hips             81594 non-null  category
 5   brassiere_size   81594 non-null  category
 6   category         81594 non-null  category
 7   height           81594 non-null  float64 
 8   length           81594 non-null  object  
 9   fit              81594 non-null  category
 10  user_id          81594 non-null  int64   
 11  shoe size        81594 non-null  category
 12  shoe width       81594 non-null  category
 13  review_summary   81594 non-null  object  
 14  review_text      81594 non-null  object  
 15  first_time_user  81594 non-null  bool    
dtypes: bool(1), category(8), float64(1), int64(3), object(3)
memory usage: 5.7+ MB
```

1.5) EDA via visualizations¶

1. Distribution of different features over Modcloth dataset¶

```
def plot_distribution(col, ax):
    df[col][df[col].notnull()].value_counts().plot(kind='bar', facecolor='b', ax=ax)
    ax.set_xlabel('{}'.format(col), fontsize=30)
    #ax.set_title("{} on Modcloth".format(col), fontsize= 18)
    ax.tick_params(axis='x', labelsize=20)  # Increase font size for x-axis ticks
    ax.tick_params(axis='y', labelsize=20)
    return ax
```

```
import matplotlib.pyplot as plt
f, ax = plt.subplots(2, 4, figsize=(35, 15))
f.tight_layout(h_pad=12, w_pad=6, rect=[0, 0.09, 1, 1.05])
cols = ['brassiere_size', 'category', 'cup size', 'fit', 'height', 'hips', 'length', 'quality']
k = 0

for i in range(2):
    for j in range(4):
        if k < len(cols):
            plot_distribution(cols[k], ax[i][j])
            k += 1
        else:
            ax[i][j].axis('off')
```



2. Categories vs. Fit/Length/Quality¶

Here, we will visualize how the items of different categories fared in terms of - fit, length, and quality. This will tell Modcloth which categories need more attention!

I have plotted 2 distributions in categories here:

1. Unnormalized- viewing the frequency counts directly- for comparison across categories. We also include the best fit, length, or quality measure in this plot.

2. Normalized - viewing the distribution for the category after normalizing the counts, amongst the category itself- it will help us compare what are major reason for return amongst the category itself. We exclude the best sizing & quality measures, so as to focus on the predominant reasons of return per category (if any).

```
def plot_barh(df,col, cmap = None, stacked=False, norm = None):
    df.plot(kind='barh', colormap=cmap, stacked=stacked)
    fig = plt.gcf()
    fig.set_size_inches(24,12)
    plt.title("Category vs {}-feedback -  Modcloth {}".format(col, '(Normalized)' if norm else ''), fontsize= 30)
    plt.ylabel('Category', fontsize = 28)
    plot = plt.xlabel('Frequency', fontsize=28)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
```

```
def norm_counts(t):
    norms = np.linalg.norm(t.fillna(0), axis=1)
    t_norm = t[0:0]
    for row, euc in zip(t.iterrows(), norms):
```

```
        t_norm.loc[row[0]] = list(map(lambda x: x/euc, list(row[1])))
    return t_norm
```

```
df.category.value_counts()
```

```
category
new          21177
tops         20048
dresses      18402
bottoms      15047
outerwear     4180
sale          2469
wedding        271
Name: count, dtype: int64
```

Category vs. Fit

```
import numpy as np
```

```
groupByCategory = df.groupby('category')
```

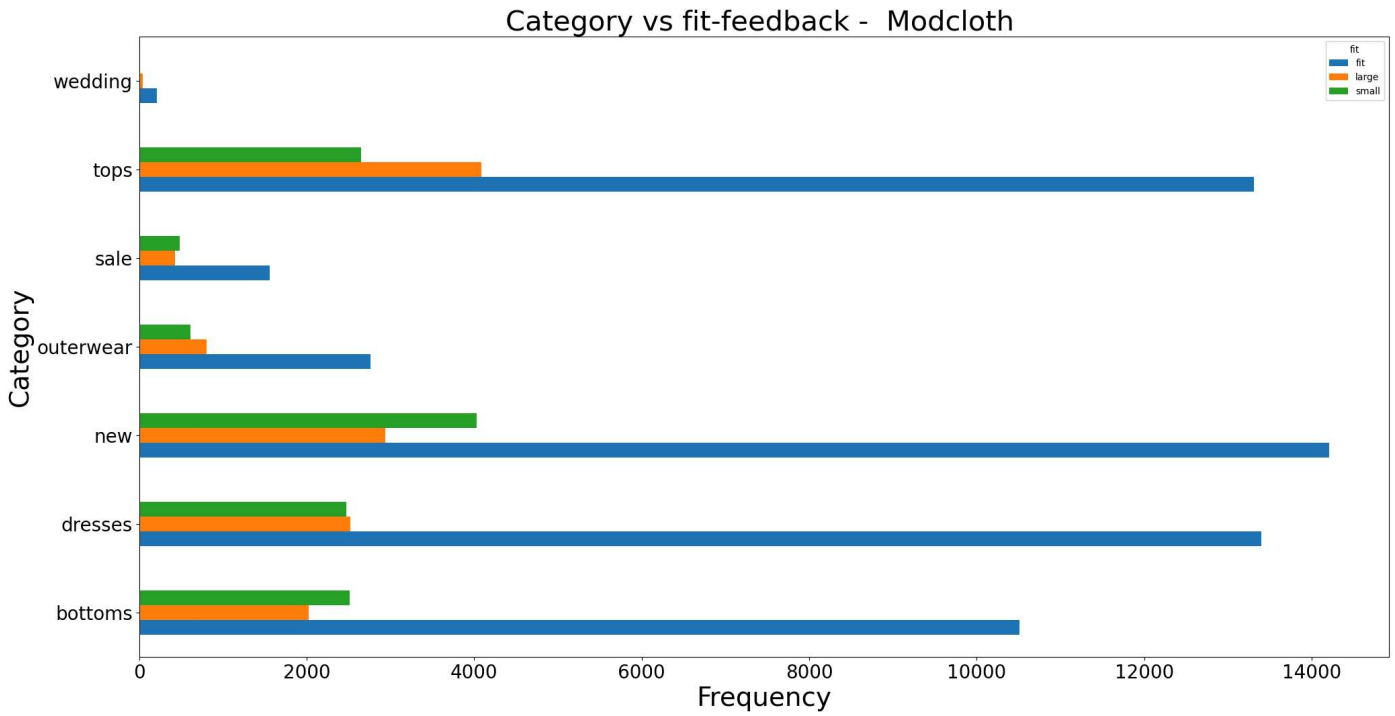
```
categoryFit = groupByCategory['fit'].value_counts()
```

```
categoryFit = categoryFit.unstack()
```

```
categoryFitNormalization = norm_counts(categoryFit)
```

```
categoryFitNormalization.drop(['fit'], axis=1, inplace=True)
```

```
plot_barh(categoryFit, 'fit')
```



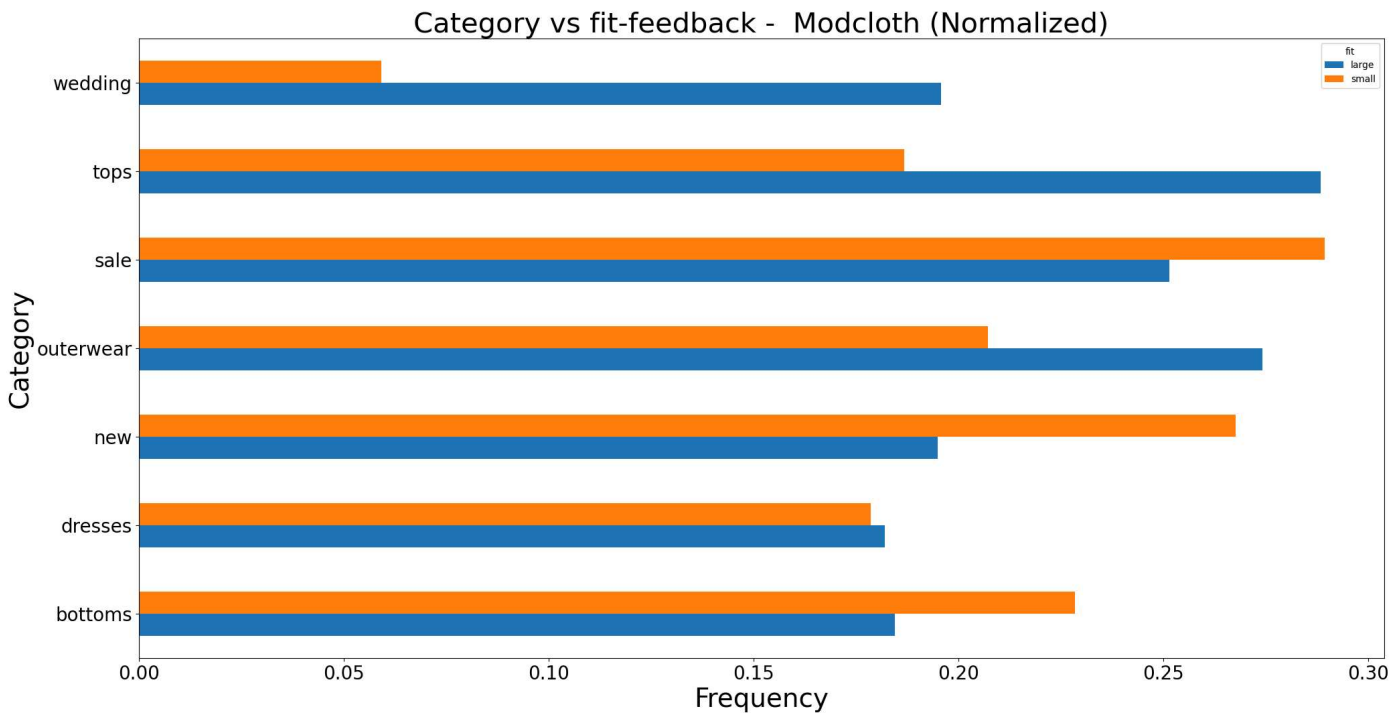
Observations:

Best-fit response (fit) has been highest for new, dresses, and tops categories.

Overall maximum bad fit-feedback has belonged mostly to 2 categories- new and tops! Dresses and bottoms categories follow.

Weddings, outerwear, and sale are not prominent in our visualization- mostly due to the lack of transactions in these categories.

```
plot_barh(categoryFitNormalization, 'fit', norm=1)
```



Here, we can see that amongst the categories themselves:

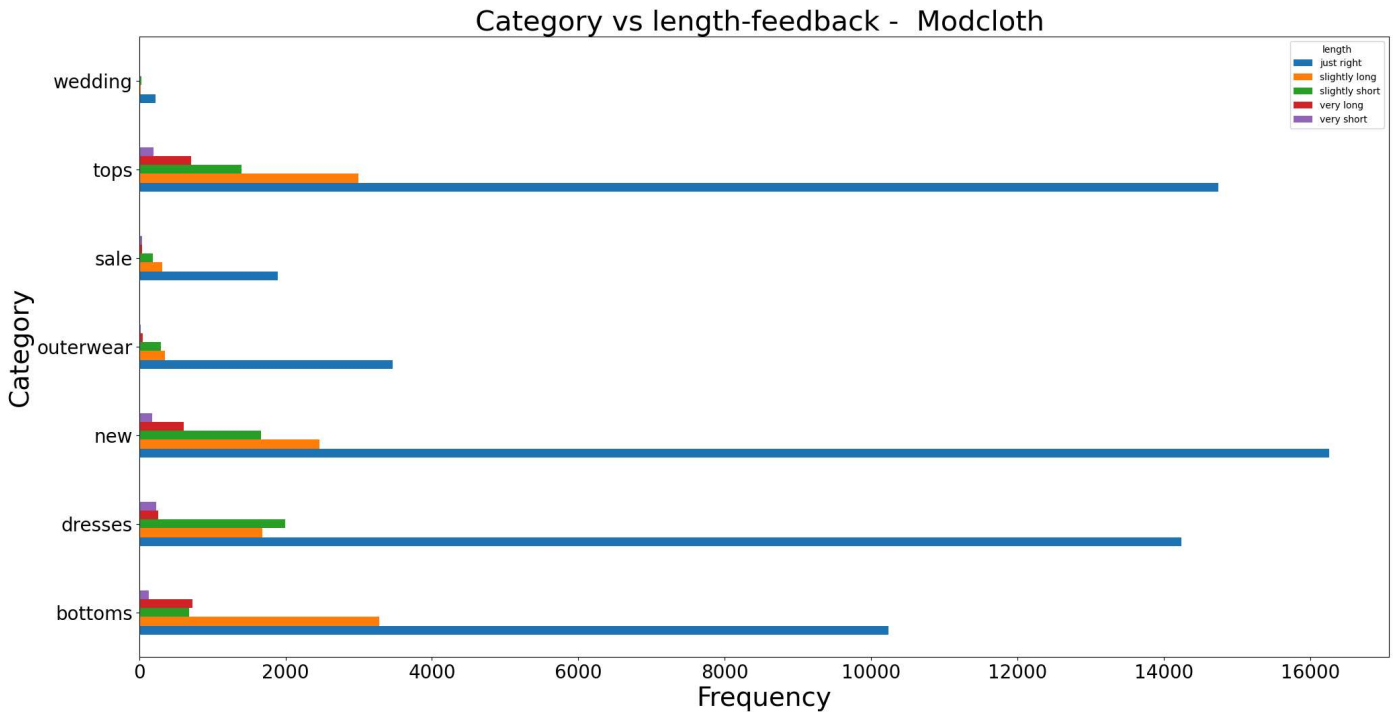
Wedding, tops, & outerwear categories usually have more returns due to large sizing.

New, sale, & bottoms usually have frequent returns due to small sized buys.

Dresses has similar return reasons, in terms of fit.

Category vs Length

```
categoryLength = groupByCategory['length'].value_counts()
categoryLength = categoryLength.unstack()
plot_barh(categoryLength, 'length')
```



Best length-fitting ('just right') belongs to tops, new, dresses and bottoms! (Also due to predominance of these categories in our total transactions- they make up almost 92% of our transactions!)

All transactions share a similar order of reasons for return (in the order of importance), which is kind of intuitive as well:

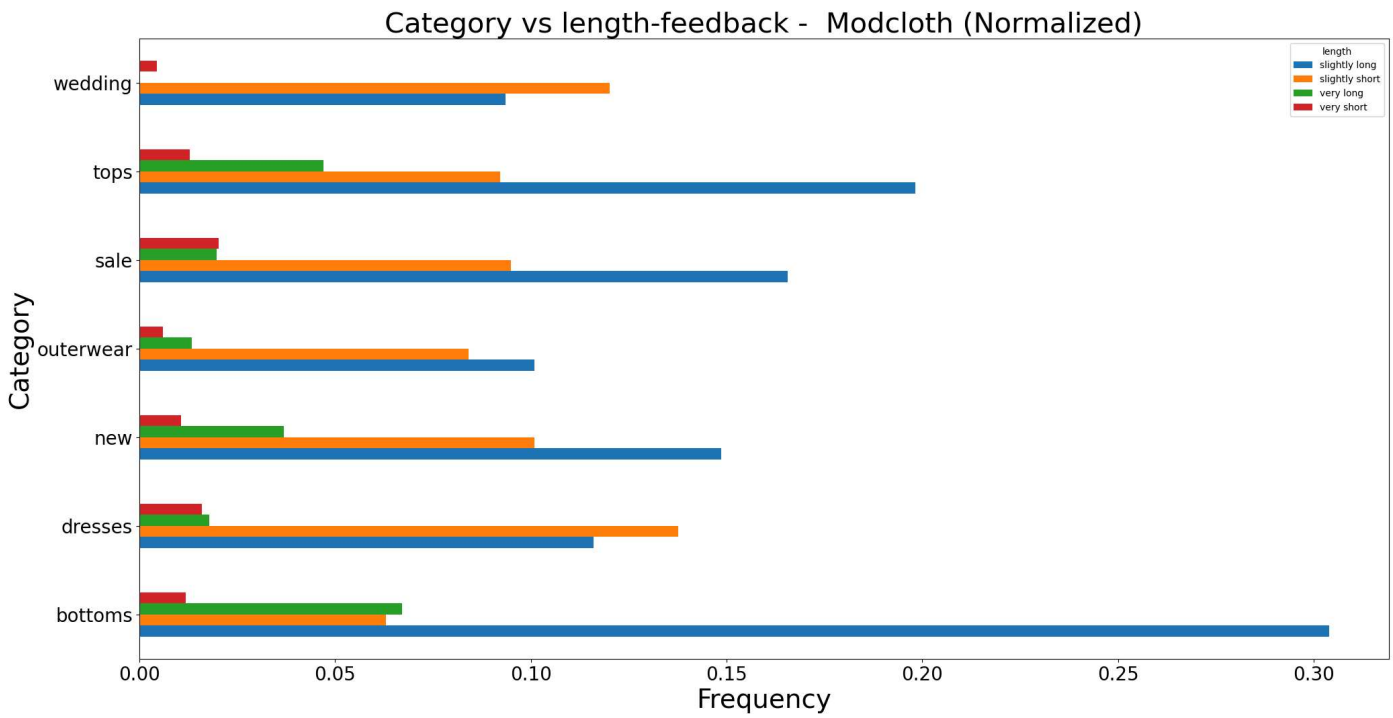
-slightly long

-slightly short

-very long

-very short

```
categoryLengthNorm = norm_counts(categoryLength)
categoryLengthNorm.drop(['just right'], axis = 1, inplace=True)
plot_barh(categoryLengthNorm, 'length', norm=1)
```



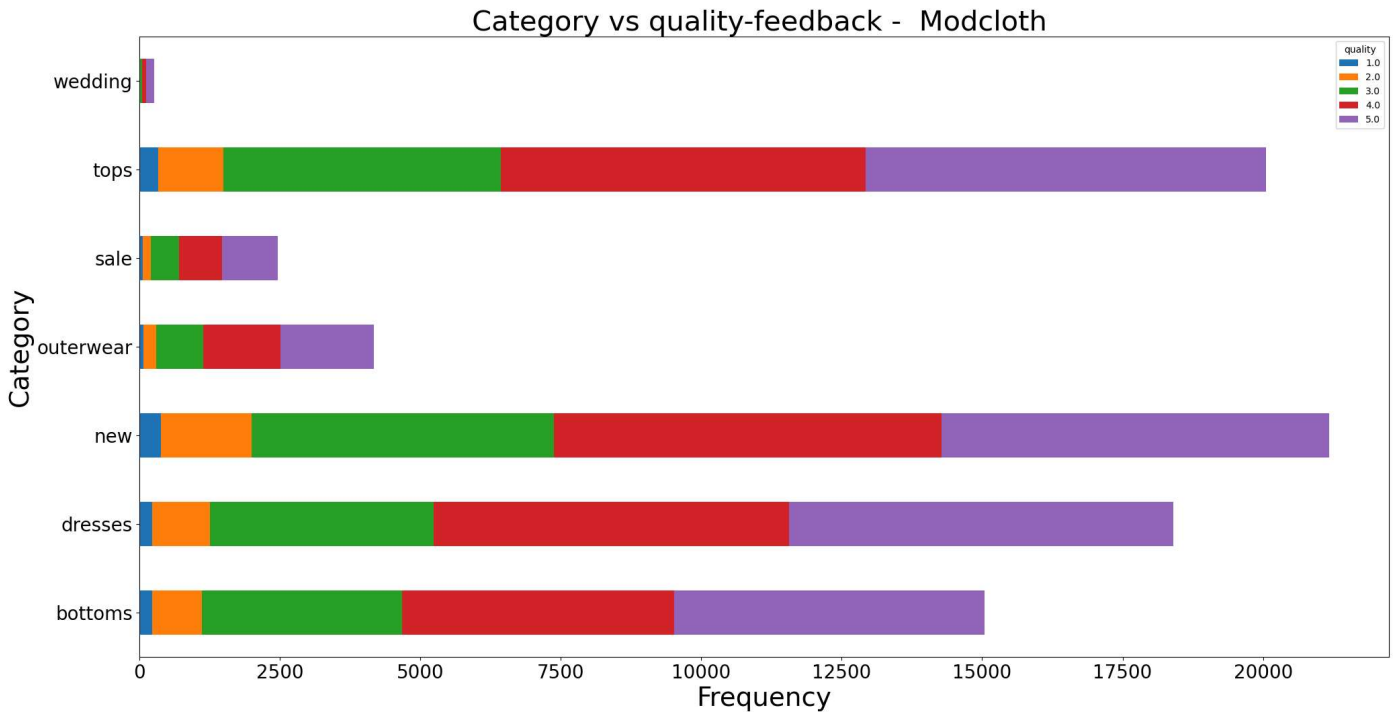
The normalized plot, focusing on the problems allows us to dig deeper into length-wise reasons of return per category:

-Customers tend to make 'slightly long' purchases in bottoms, new, sale, & tops categories.

-slightly short' returns take place mostly in dresses and wedding categories.

Category vs Quality

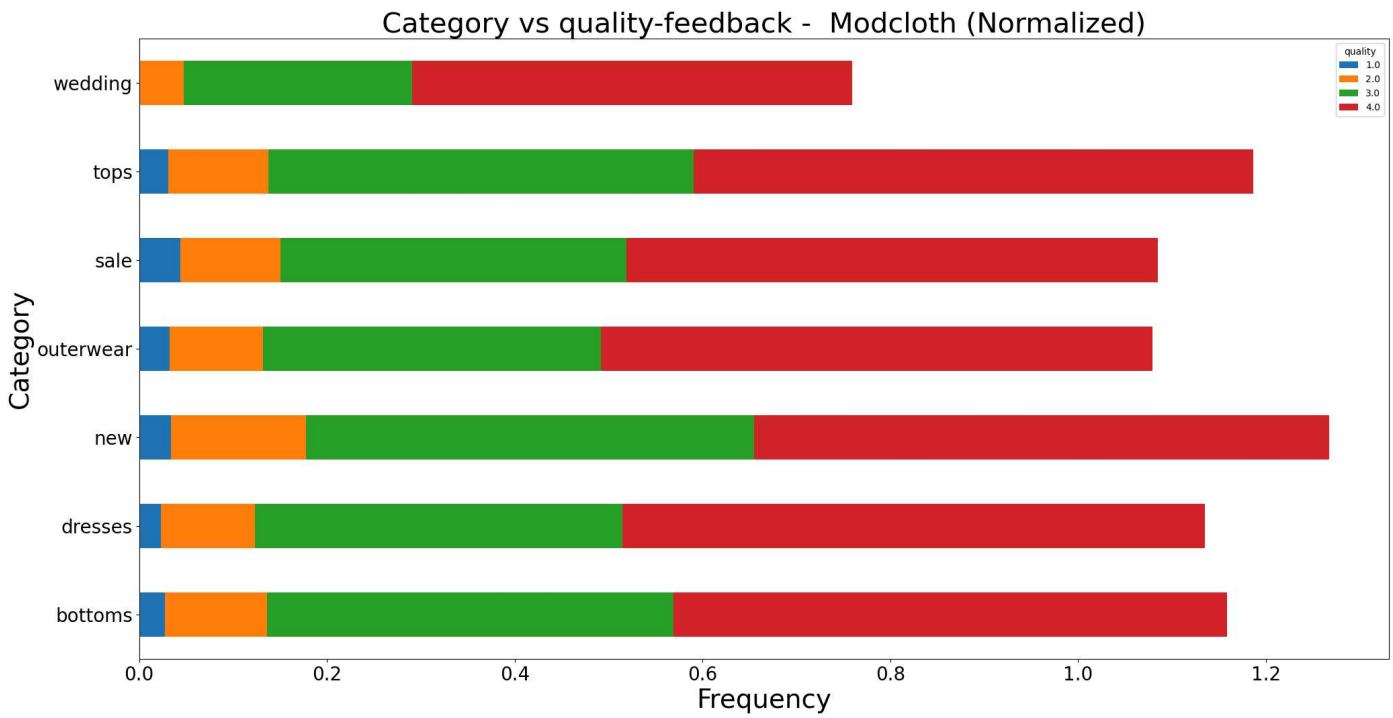
```
CategoryQuality = groupByCategory['quality'].value_counts()
CategoryQuality = CategoryQuality.unstack()
plot_barh(CategoryQuality, 'quality', stacked=3)
```



Almost the same share of people have rated the categories of tops, new, dresses, & bottoms as 5, 4, & 3.

All the trends in terms of share of ratings seems to be constant across categories.

```
categoryLengthNorm  = norm_counts(CategoryQuality)
categoryLengthNorm .drop([5.0], axis = 1, inplace=True)
plot_barh(categoryLengthNorm, 'quality',  stacked=3, norm=1)
```



Here also we can assert our previous observation that all the categories share similar share of ratings.

To nitpick- new , sale & tops seem to have a higher share than normal of bad ratings (1.0 & 2.0) in terms of quality.

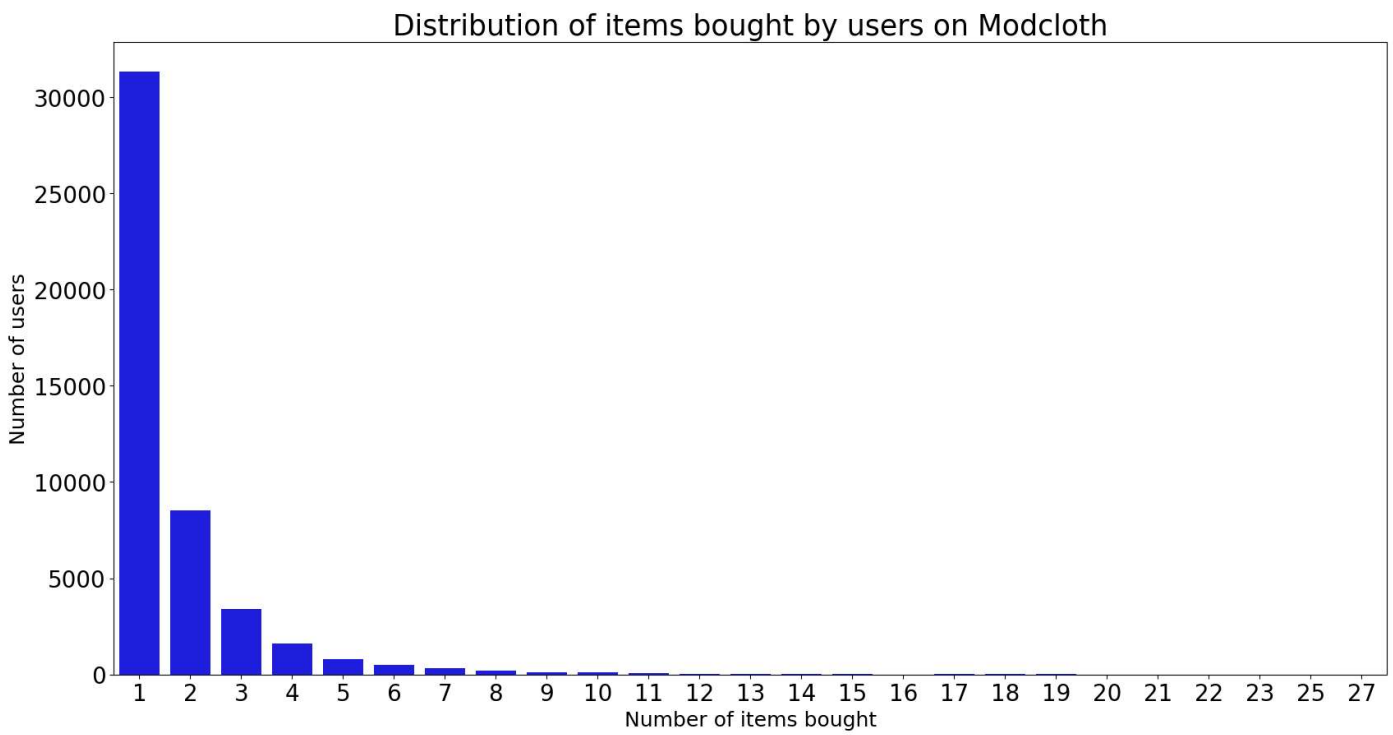
2. Total Number of Users vs Total Number of items bought¶

Visualizing the total number of users who bought x number of items, where we affirm the author's [1] statement that the data is very sparse with a major chunk (38.45%) of the users who bought only 1 item from the website during the time this data was collected.

```
items_bought = []
total_users = []
```

```
for i in range(min(df.user_id.value_counts()), max(df.user_id.value_counts())+1):
    all_users = sum(df.user_id.value_counts() == i)
    if all_users != 0:
        total_users.append(all_users)
        items_bought.append(i)
```

```
plt.xlabel("Number of items bought", fontsize = 18)
plt.ylabel("Number of users", fontsize = 18)
plt.title("Distribution of items bought by users on Modcloth",fontsize = 25)
__ = sns.barplot(x=items_bought, y=total_users, color='b')
fig = plt.gcf()
fig.set_size_inches(20,10)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```



A major chunk of the users (~40%) have only bought 1 item from Modcloth during the time this data was collected. Although we found only 903 out of those were first time users (no previous data existed of these customers). This explains and reaffirms the dataset curator's statement about sparsity of the data.

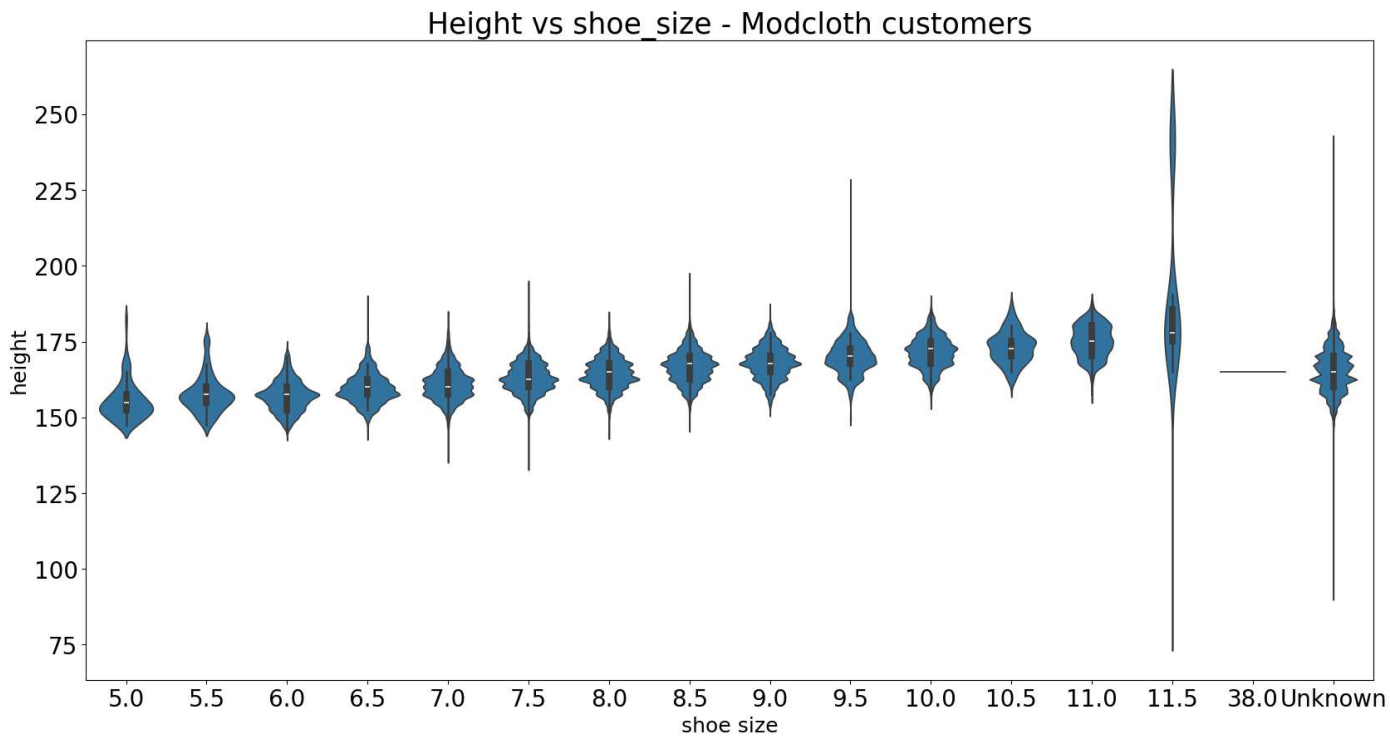
Most users bought 1, 2, or 3 products from Modcloth out of the ~80,000 transactions in this dataset.

3. Height vs shoe_size - Modcloth customers¶

It would be interesting to see if there exists a linear relation between the height of a person and their shoe-size, i.e.- it will mean shoe-size increases with increase in height!

```
import seaborn as sns
```

```
plt.figure(figsize=(20, 10))
plt.xlabel("shoe size", fontsize = 18)
plt.ylabel("height", fontsize = 18)
plt.title("Height vs shoe_size - Modcloth customers",fontsize = 25)
sns.violinplot(x='shoe size', y='height', data=df)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```



End of Step-by-step Data Preprocessing & EDA

```
Start coding or generate with AI.
```

2) Advance Data Preprocessing

2.1) Introduction

Data Preprocessing :

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Data Preprocessing involves below steps:

1) Getting the dataset

2) Importing libraries

3) Importing datasets

4) Finding Missing Data

5) Encoding Categorical Data

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
loan=pd.read_csv("Loan_Default.csv")
```

```
loan
```

	ID	year	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercia
0	24890	2019	cf	Sex Not Available	nopre	type1	p1	l1	nopc	nob/
1	24891	2019	cf	Male	nopre	type2	p1	l1	nopc	b/
2	24892	2019	cf	Male	pre	type1	p1	l1	nopc	nob/
3	24893	2019	cf	Male	nopre	type1	p4	l1	nopc	nob/
4	24894	2019	cf	Joint	pre	type1	p1	l1	nopc	nob/
...	...	...	...	...	...	...	...	...	...	.
148665	173555	2019	cf	Sex Not Available	nopre	type1	p3	l1	nopc	nob/
148666	173556	2019	cf	Male	nopre	type1	p1	l1	nopc	nob/
148667	173557	2019	cf	Male	nopre	type1	p4	l1	nopc	nob/
148668	173558	2019	cf	Female	nopre	type1	p4	l1	nopc	nob/
148669	173559	2019	cf	Female	nopre	type1	p3	l1	nopc	nob/

148670 rows × 34 columns

```
loan.head(10)
```

	ID	year	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	...
0	24890	2019	cf	Sex Not Available	nopre	type1	p1	l1	nopc	nob/c	...
1	24891	2019	cf	Male	nopre	type2	p1	l1	nopc	b/c	...
2	24892	2019	cf	Male	pre	type1	p1	l1	nopc	nob/c	...
3	24893	2019	cf	Male	nopre	type1	p4	l1	nopc	nob/c	...
4	24894	2019	cf	Joint	pre	type1	p1	l1	nopc	nob/c	...
5	24895	2019	cf	Joint	pre	type1	p1	l1	nopc	nob/c	...
6	24896	2019	cf	Joint	pre	type1	p3	l1	nopc	nob/c	...
7	24897	2019	NaN	Female	nopre	type1	p4	l1	nopc	nob/c	...
8	24898	2019	cf	Joint	nopre	type1	p3	l1	nopc	nob/c	...
9	24899	2019	cf	Sex Not Available	nopre	type3	p3	l1	nopc	nob/c	...

10 rows × 34 columns

```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148670 entries, 0 to 148669
Data columns (total 34 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   ID                         148670 non-null  int64  
 1   year                       148670 non-null  int64  
 2   loan_limit                 145326 non-null  object 
 3   Gender                     148670 non-null  object 
 4   approv_in_adv              147762 non-null  object 
 5   loan_type                  148670 non-null  object 
 6   loan_purpose               148536 non-null  object 
 7   Credit_Worthiness          148670 non-null  object 
 8   open_credit                148670 non-null  object 
 9   business_or_commercial     148670 non-null  object 
 10  loan_amount                148670 non-null  int64  
 11  rate_of_interest           112231 non-null  float64
 12  Interest_rate_spread       112031 non-null  float64
 13  Upfront_charges            109028 non-null  float64
 14  term                       148629 non-null  float64
 15  Neg_ammortization          148549 non-null  object 
 16  interest_only              148670 non-null  object 
 17  lump_sum_payment           148670 non-null  object 
 18  property_value             133572 non-null  float64
 19  construction_type          148670 non-null  object 
 20  occupancy_type             148670 non-null  object 
 21  Secured_by                 148670 non-null  object 
 22  total_units                148670 non-null  object 
 23  income                     139520 non-null  float64
 24  credit_type                148670 non-null  object 
 25  Credit_Score               148670 non-null  int64  
 26  co-applicant_credit_type   148670 non-null  object 
 27  age                        148470 non-null  object 
 28  submission_of_application  148470 non-null  object 
 29  LTV                        133572 non-null  float64
 30  Region                     148670 non-null  object 
 31  Security_Type              148670 non-null  object 
 32  Status                     148670 non-null  int64  
 33  dtir1                      124549 non-null  float64
dtypes: float64(8), int64(5), object(21)
memory usage: 38.6+ MB
```

2.2) Handling Missing Values

Missing data are not rare in real data sets. In fact, the chance that at least one data point is missing increases as the data set size increases. Missing data can occur any number of ways, some of which include the following.

a) Merging of source data sets

b) Random events

c) Failures of measurement

2.2.1) Visualizing Missing Data

1 Matrix

t is the nullity matrix that allows us to see the distribution of data across all columns in the whole dataset. It also shows a sparkline (or, in some cases, a striped line) that emphasizes rows in a dataset with the highest and lowest nullity.

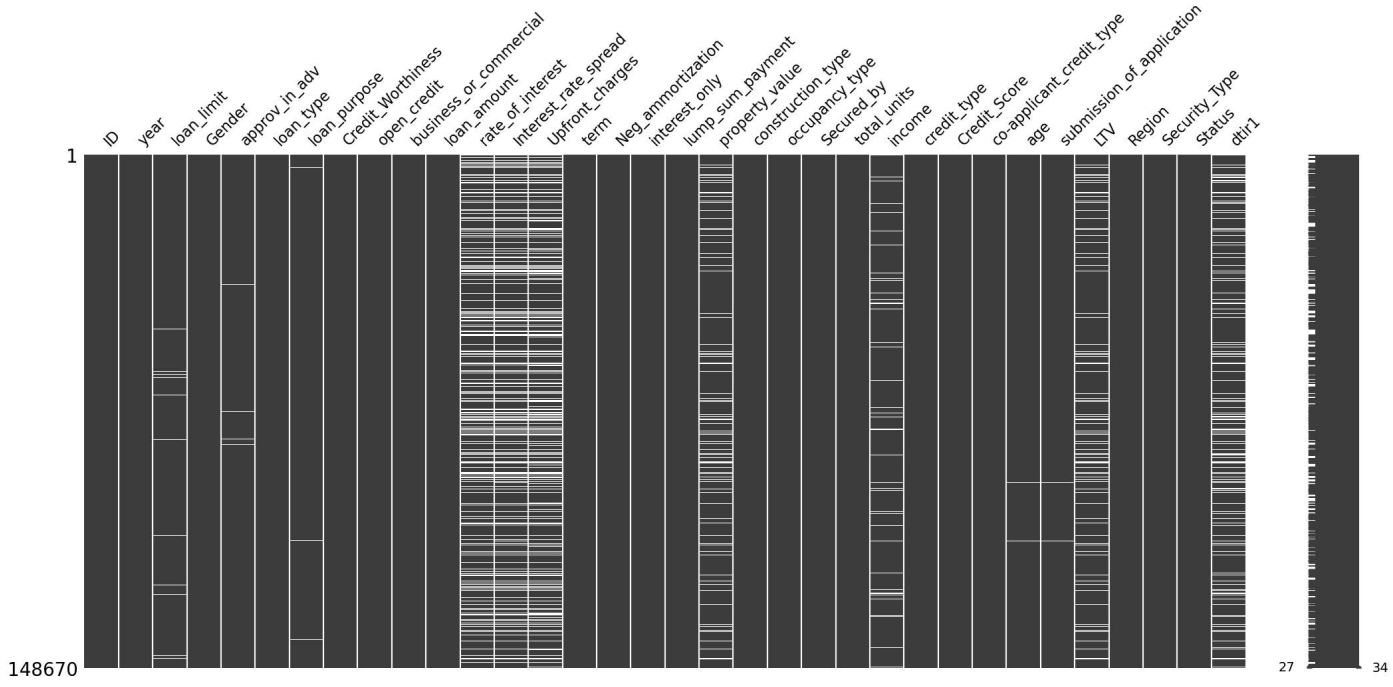
```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
import missingno as msno
```

```
plt.figure(figsize = (12,8))
msno.matrix(loan)
plt.show()
```

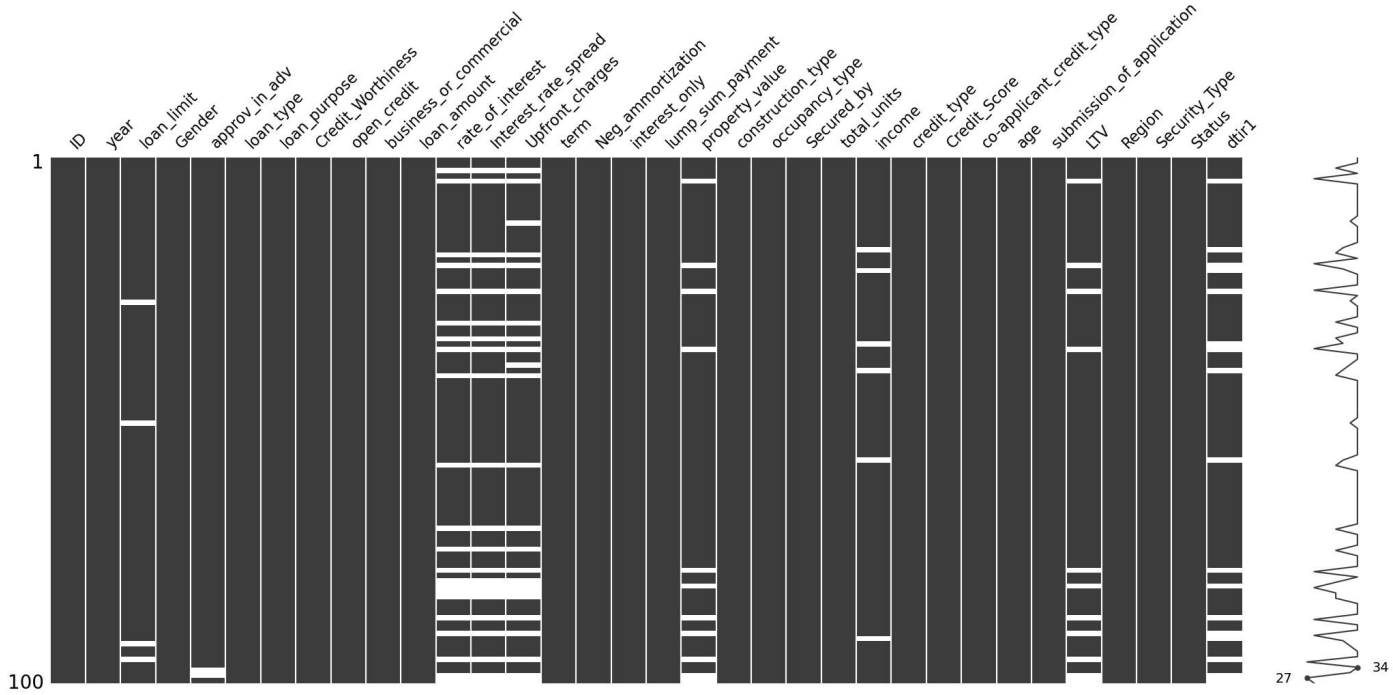
```
<Figure size 1200x800 with 0 Axes>
```



From the above plot we can interpret our dataset has lots of missing values in it

```
msno.matrix(loan.sample(100))
```

```
<AxesSubplot:>
```



2. Correlation Heatmap

Correlation heatmap measures nullity correlation between columns of the dataset. It shows how strongly the presence or absence of one feature affects the other.

Nullity correlation ranges from(-1 to 1):

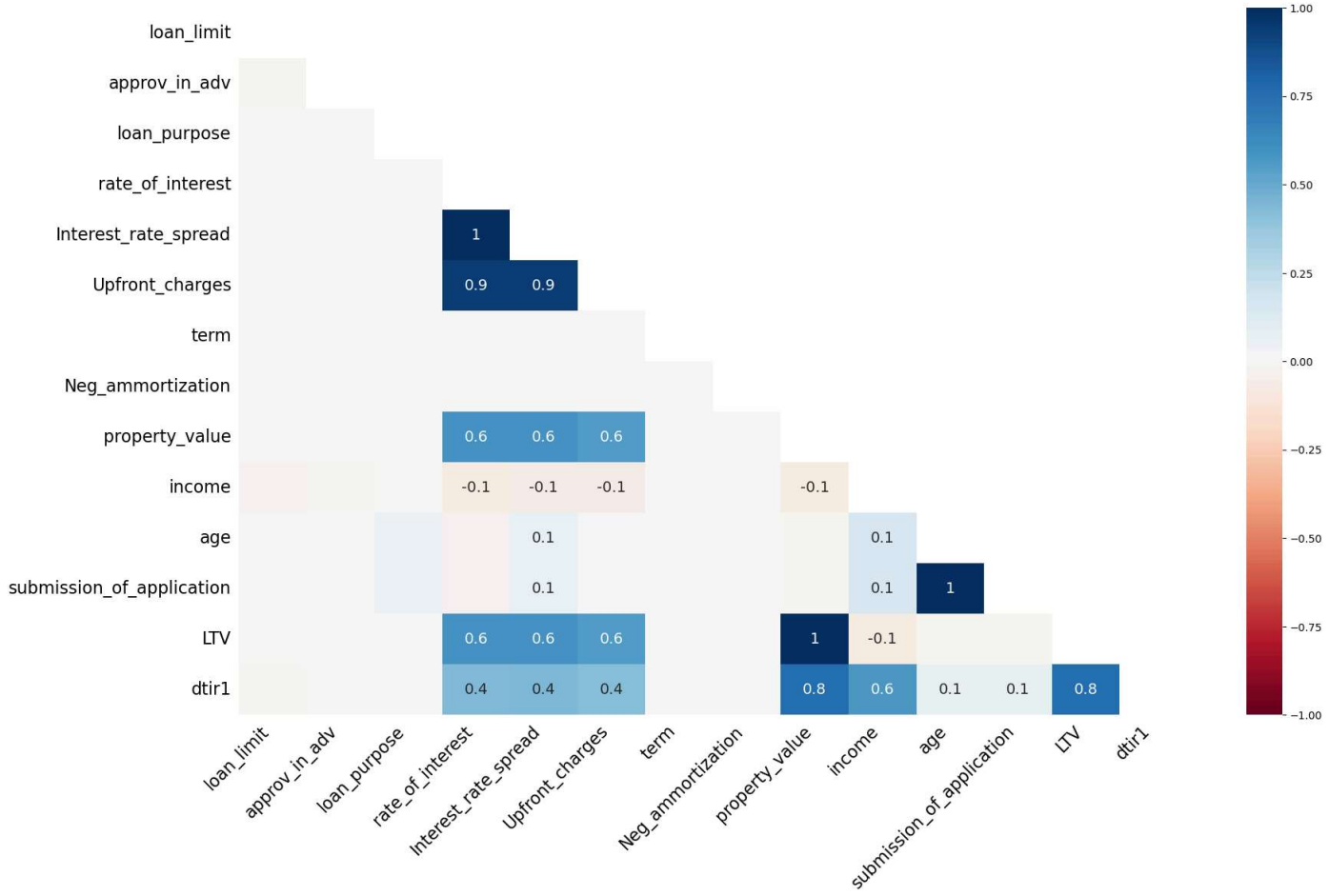
a) -1 means if one column(attribute) is present, the other is almost certainly absent.

b) 0 means there is no dependence between the columns(attributes).

c) 1 means if one column(attributes) is present, the other is also certainly present.

```
msno.heatmap(loan, labels = True)
```

<AxesSubplot:>



From above visualization we can easily interpret missingness of attribute rate_of_interest and upfront_charges is dependent on each other(correlation value = 1) means if one will be present another will be present.

3. Dendrogram

A dendogram is a tree diagram of missingness. It groups the highly correlated variables together.

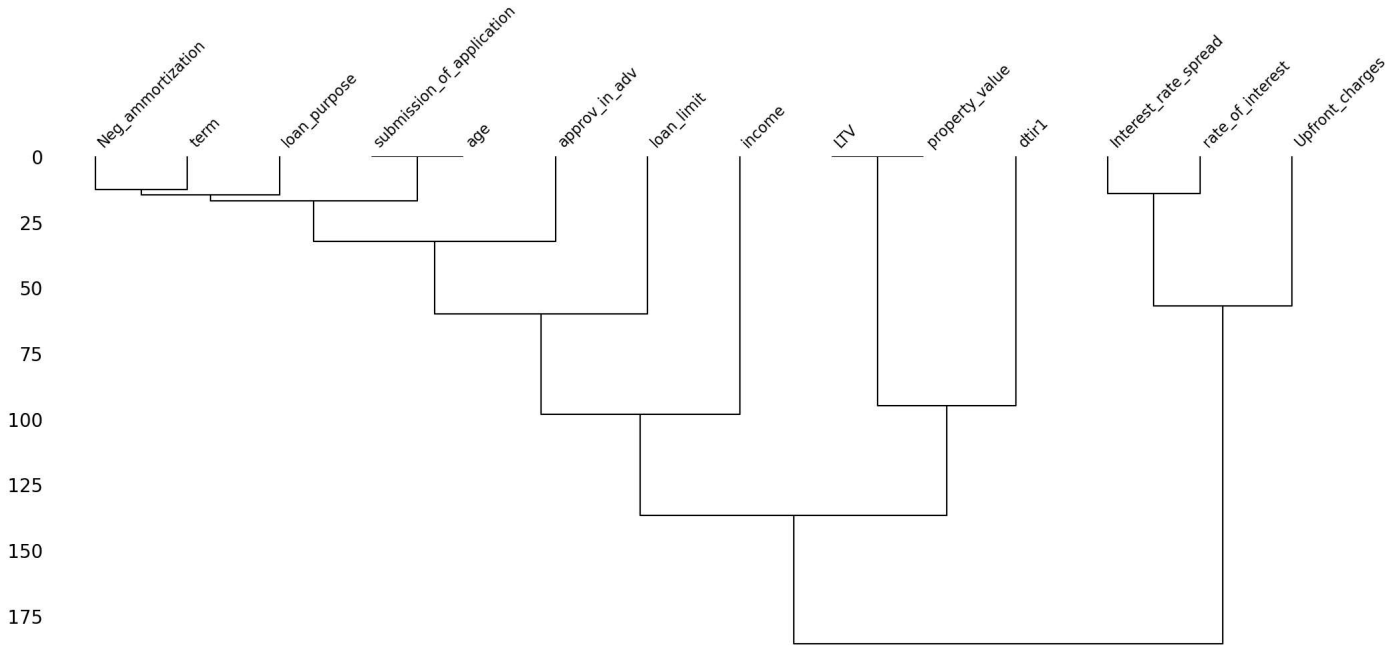
```
missing_columns = []


for col in loan.columns:
    if loan[col].isnull().sum() > 0:
        missing_columns.append(col)
```

```
missing_columns
```

```
['loan_limit',
 'approv_in_adv',
 'loan_purpose',
 'rate_of_interest',
 'Interest_rate_spread',
 'Upfront_charges',
 'term',
 'Neg_ammortization',
 'property_value',
 'income',
 'age',
 'submission_of_application',
 'LTV',
 'dtir1']
```

```
msno.dendrogram(loan[missing_columns])
```

```
<AxesSubplot:>
```



2.2.2 Simple Numerical Summaries

```
def get_numerical_summary(df):
    total_rows = df.shape[0]
    missing_columns = [col for col in df.columns if df[col].isnull().sum() > 0]
    missing_percent = {}

    for col in missing_columns:
        missing_count = df[col].isnull().sum()
        percentage = (missing_count / total_rows) * 100
        missing_percent[col] = percentage
        print(f"{col}: {missing_count} ({round(percentage, 3)}%)")
    return missing_percent
```

```
missing_percent = get_numerical_summary(loan)
```

```
loan_limit: 3344 (2.249%)
approv_in_adv: 908 (0.611%)
```

```
loan_purpose: 134 (0.09%)
rate_of_interest: 36439 (24.51%)
Interest_rate_spread: 36639 (24.645%)
Upfront_charges: 39642 (26.664%)
term: 41 (0.028%)
Neg_ammortization: 121 (0.081%)
property_value: 15098 (10.155%)
income: 9150 (6.155%)
age: 200 (0.135%)
submission_of_application: 200 (0.135%)
LTV: 15098 (10.155%)
dtir1: 24121 (16.225%)
```

2.3 Methods to Handle Missing Data

a) Deletion of Data

b) Encoding Missingness

c) Imputation Methods

2.3.1 Deletion of Data

According to Simple numerical Summaries the attribute Upfront_charges has largest missing values percentage of (26.664%) which is not ideal percentage to remove a feature but just for sake of implementation I will remove that feature.

```
loan_temp=loan.copy()
```

```
THRESHOLD = 20

for col, per in missing_percent.items():
    if per > THRESHOLD:
        loan_temp.drop(col, axis = 1, inplace = True)
```

```
above_threshold_missing=get_numerical_summary(loan_temp)
```

```
loan_limit: 3344 (2.249%)
approv_in_adv: 908 (0.611%)
loan_purpose: 134 (0.09%)
term: 41 (0.028%)
Neg_ammortization: 121 (0.081%)
property_value: 15098 (10.155%)
income: 9150 (6.155%)
age: 200 (0.135%)
submission_of_application: 200 (0.135%)
LTV: 15098 (10.155%)
dtir1: 24121 (16.225%)
```

Deletion of the Samples

We will try to delete those samples having missing values in more than 5 attributes

```
loan_temp=loan.copy()
```

```
# Getting Missing count of each sample

for i in range(loan_temp.shape[0]):
    loan_temp.loc[i, 'missing_count'] = loan_temp.iloc[i, :].isnull().sum()
```

```
print("Samples Before Removal : {}".format(loan_temp.shape[0]))
```

```
Samples Before Removal : 148670
```

```
THRESHOLD = 5
```

```
loan_temp.drop(loan_temp[loan_temp['missing_count'] > THRESHOLD].index, axis = 0, inplace = True)
```

```
print("Samples After Removal : {}".format(loan_temp.shape[0]))
```

```
Samples After Removal : 132360
```

2.3.2 Encoding Missingness

When an attribute is discrete in nature, missingness can be directly encoded into the attribute as if it were a naturally occurring category. For example in this dataset the attribute loan_limit has 3344 missing values so we can assign some new category to these missing values.

```
Categoricalmissing = [col for col in missing_columns if loan[col].dtype == 'object']
```

```
Categoricalmissing
```

```
['loan_limit',
 'approv_in_adv',
 'loan_purpose',
 'Neg_ammortization',
 'age',
 'submission_of_application']
```

```
loan['loan_limit'].value_counts()
```

```
loan_limit
cf     135348
ncf      9978
Name: count, dtype: int64
```

```
loan[Categoricalmissing] = loan[Categoricalmissing].fillna('Missing')
```

```
loan.loan_limit.value_counts()
```

```
loan_limit
cf         135348
ncf          9978
Missing      3344
Name: count, dtype: int64
```

```
loan[Categoricalmissing].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148670 entries, 0 to 148669
Data columns (total 6 columns):
 #   Column                     Non-Null Count   Dtype 
---  ------                     --------------   ----- 
 0   loan_limit                 148670 non-null  object
 1   approv_in_adv              148670 non-null  object
 2   loan_purpose               148670 non-null  object
 3   Neg_ammortization          148670 non-null  object
 4   age                        148670 non-null  object
 5   submission_of_application  148670 non-null  object
dtypes: object(6)
memory usage: 6.8+ MB
```

2.3.3 Imputation Methods

(a) K-Nearest Neighbors(KNN) for Imputation

When the training set is small or moderate in size, K-nearest neighbors can be a quick and effective method for imputing missing values. This procedure identifies a sample with one or more missing values. Then it identifies the K most similar samples in the training data that are complete (i.e., have no missing values in some columns). Similarity of samples for this method is defined by a distance metric. When all of the predictors are numeric, standard Euclidean distance is commonly used as the similarity metric.

```
from sklearn.impute import KNNImputer
```

```
loan_temp = loan.copy()
```

```
NumericCols = [col for col in loan_temp.columns if loan_temp[col].dtype != 'object']
```

```
NumericCols
```

```
['ID',
 'year',
 'loan_amount',
 'rate_of_interest',
 'Interest_rate_spread',
 'Upfront_charges',
 'term',
 'property_value',
 'income',
 'Credit_Score',
 'LTV',
 'Status',
 'dtir1']
```

```
loan_temp = loan_temp[NumericCols]
```

```
imputer= KNNImputer(n_neighbors = 5)
```

```
imputer.fit(loan_temp)
```

```
KNNImputer()
```

```
X = imputer.transform(loan_temp)
```

```
loan_temp = pd.DataFrame(X, columns = NumericCols)
```

```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148670 entries, 0 to 148669
Data columns (total 34 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   ID                         148670 non-null  int64  
 1   year                       148670 non-null  int64  
 2   loan_limit                 148670 non-null  object 
 3   Gender                     148670 non-null  object 
 4   approv_in_adv              148670 non-null  object 
 5   loan_type                  148670 non-null  object 
 6   loan_purpose               148670 non-null  object 
 7   Credit_Worthiness          148670 non-null  object 
 8   open_credit                148670 non-null  object 
 9   business_or_commercial     148670 non-null  object 
 10  loan_amount                148670 non-null  int64  
 11  rate_of_interest           112231 non-null  float64
 12  Interest_rate_spread       112031 non-null  float64
 13  Upfront_charges            109028 non-null  float64
 14  term                       148629 non-null  float64
 15  Neg_ammortization          148670 non-null  object 
 16  interest_only              148670 non-null  object 
 17  lump_sum_payment           148670 non-null  object 
 18  property_value             133572 non-null  float64
 19  construction_type          148670 non-null  object 
 20  occupancy_type             148670 non-null  object 
 21  Secured_by                 148670 non-null  object 
 22  total_units                148670 non-null  object 
 23  income                     139520 non-null  float64
 24  credit_type                148670 non-null  object 
 25  Credit_Score               148670 non-null  int64  
 26  co-applicant_credit_type   148670 non-null  object 
 27  age                        148670 non-null  object 
 28  submission_of_application  148670 non-null  object 
 29  LTV                        133572 non-null  float64
```

```
 30  Region                     148670 non-null  object
 31  Security_Type              148670 non-null  object
 32  Status                     148670 non-null  int64
 33  dtir1                      124549 non-null  float64
dtypes: float64(8), int64(5), object(21)
memory usage: 38.6+ MB
```

(b) Trees

```
del loan_temp
```

```
loan_temp=loan.copy()
```

```
missing_columns
```

```
['loan_limit',
 'approv_in_adv',
 'loan_purpose',
 'rate_of_interest',
 'Interest_rate_spread',
 'Upfront_charges',
 'term',
 'Neg_ammortization',
 'property_value',
 'income',
 'age',
 'submission_of_application',
 'LTV',
 'dtir1']
```

```
from sklearn.tree import DecisionTreeRegressor
```

```
DRegressor = DecisionTreeRegressor()
```

```
income = loan['income']
```

```
from sklearn.preprocessing import LabelEncoder
```

```
encoder=LabelEncoder()
```

```
CategoricalCols = [col for col in loan.columns if loan[col].dtype == 'object']
```

```
CategoricalCols
```

```
['loan_limit',
 'Gender',
 'approv_in_adv',
 'loan_type',
 'loan_purpose',
 'Credit_Worthiness',
 'open_credit',
 'business_or_commercial',
 'Neg_ammortization',
 'interest_only',
 'lump_sum_payment',
 'construction_type',
 'occupancy_type',
 'Secured_by',
 'total_units',
 'credit_type',
 'co-applicant_credit_type',
 'age',
 'submission_of_application',
 'Region',
 'Security_Type']
```

```
for col in CategoricalCols:
    loan_temp[col] = encoder.fit_transform(loan_temp[col])
```

```
loan.loan_limit.dtype
```

```
dtype('O')
```

```
from sklearn.ensemble import BaggingRegressor
```

```
from sklearn.tree import DecisionTreeRegressor
```

```
missing_cols = [col for col in loan.columns if loan[col].isnull().sum() > 0]
```

```
missing_cols
```

```
['rate_of_interest',
 'Interest_rate_spread',
 'Upfront_charges',
 'term',
 'property_value',
 'income',
 'LTV',
 'dtir1']
```

```
non_missing_cols = [col for col in loan.columns if loan[col].isnull().sum() == 0]
```

```
non_missing_cols
```

```
['ID',
 'year',
 'loan_limit',
 'Gender',
 'approv_in_adv',
 'loan_type',
 'loan_purpose',
 'Credit_Worthiness',
 'open_credit',
 'business_or_commercial',
 'loan_amount',
 'Neg_ammortization',
 'interest_only',
 'lump_sum_payment',
 'construction_type',
 'occupancy_type',
 'Secured_by',
 'total_units',
 'credit_type',
 'Credit_Score',
 'co-applicant_credit_type',
 'age',
 'submission_of_application',
 'Region',
 'Security_Type',
 'Status']
```

```
loan_temp
```

	ID	year	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial
0	24890	2019	1	3	1	0	1	0	0	1
1	24891	2019	1	2	1	1	1	0	0	0
2	24892	2019	1	2	2	0	1	0	0	1
3	24893	2019	1	2	1	0	4	0	0	1
4	24894	2019	1	1	2	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...
148665	173555	2019	1	3	1	0	3	0	0	1
148666	173556	2019	1	2	1	0	1	0	0	1
148667	173557	2019	1	2	1	0	4	0	0	1
148668	173558	2019	1	0	1	0	4	0	0	1
148669	173559	2019	1	0	1	0	3	0	0	1

148670 rows × 34 columns

```
def tree_imputation(df):
    missing_cols = [col for col in df.columns if df[col].isnull().sum() > 0]
    non_missing_cols = [col for col in df.columns if df[col].isnull().sum() == 0]
    for col in missing_cols:
        model = BaggingRegressor(DecisionTreeRegressor(), n_estimators = 40, max_samples = 1.0, max_features = 1.0, bootstrap = False, n_job

        col_missing = df[df[col].isnull()]
        temp = df.drop(df[df[col].isnull()].index, axis = 0)
        X = temp.loc[:, non_missing_cols]
        y = temp[col]

        model.fit(X, y)

        y_pred = model.predict(col_missing[non_missing_cols])

        df.loc[col_missing.index, col] = y_pred

    return df
```

```
loan_new = tree_imputation(loan_temp)
```

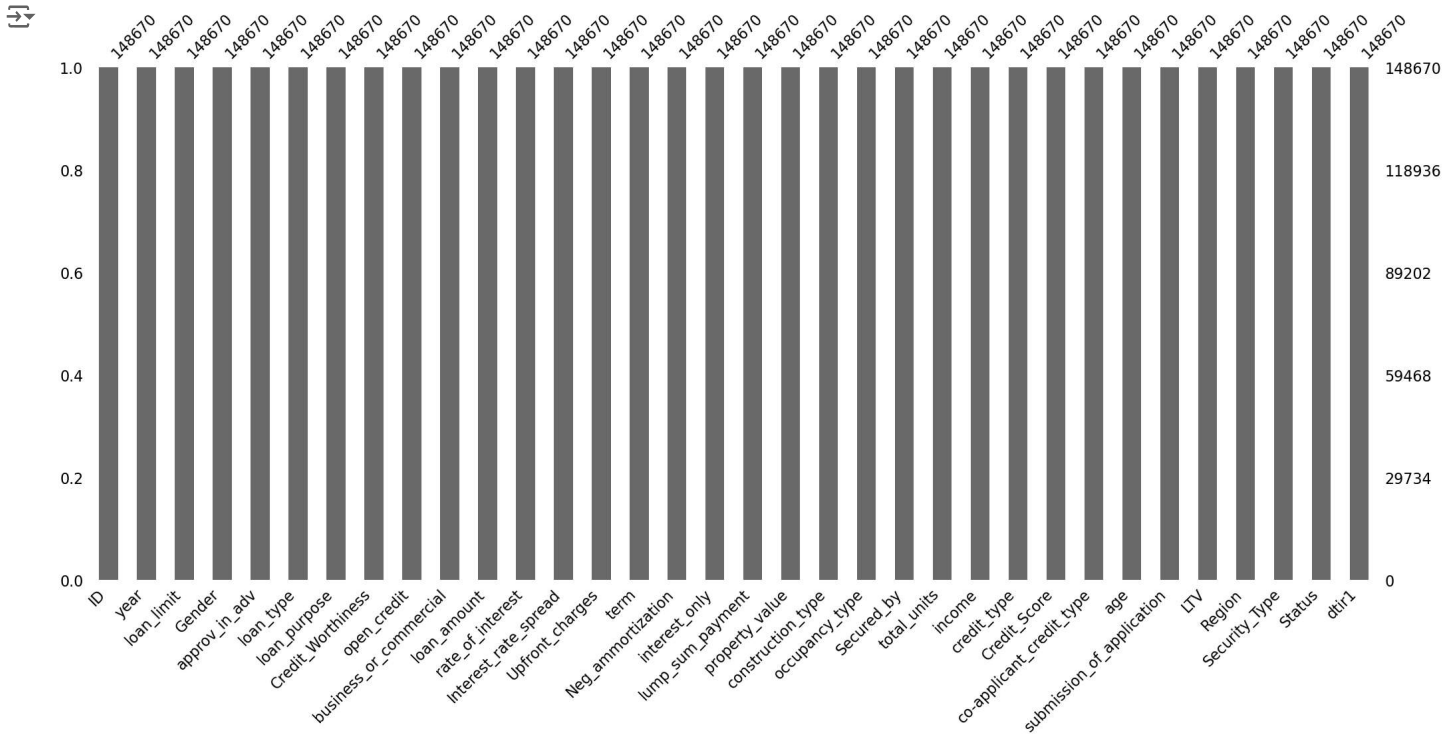
```
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version of
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
loan_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148670 entries, 0 to 148669
Data columns (total 34 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   ID                         148670 non-null  int64
 1   year                       148670 non-null  int64
 2   loan_limit                 148670 non-null  int64
```

```
 3   Gender                     148670 non-null  int64
 4   approv_in_adv              148670 non-null  int64
 5   loan_type                  148670 non-null  int64
 6   loan_purpose               148670 non-null  int64
 7   Credit_Worthiness          148670 non-null  int64
 8   open_credit                148670 non-null  int64
 9   business_or_commercial     148670 non-null  int64
 10  loan_amount                148670 non-null  int64
 11  rate_of_interest           148670 non-null  float64
 12  Interest_rate_spread       148670 non-null  float64
 13  Upfront_charges            148670 non-null  float64
 14  term                       148670 non-null  float64
 15  Neg_ammortization          148670 non-null  int64
 16  interest_only              148670 non-null  int64
 17  lump_sum_payment           148670 non-null  int64
 18  property_value             148670 non-null  float64
 19  construction_type          148670 non-null  int64
 20  occupancy_type             148670 non-null  int64
 21  Secured_by                 148670 non-null  int64
 22  total_units                148670 non-null  int64
 23  income                     148670 non-null  float64
 24  credit_type                148670 non-null  int64
 25  Credit_Score               148670 non-null  int64
 26  co-applicant_credit_type   148670 non-null  int64
 27  age                        148670 non-null  int64
 28  submission_of_application  148670 non-null  int64
 29  LTV                        148670 non-null  float64
 30  Region                     148670 non-null  int64
 31  Security_Type              148670 non-null  int64
 32  Status                     148670 non-null  int64
 33  dtir1                      148670 non-null  float64
dtypes: float64(8), int64(26)
memory usage: 38.6 MB
```

```
msno.bar(loan_new)
plt.show()
```




```
loan_new.isnull().sum()
```

```
ID                         0
year                       0
loan_limit                 0
Gender                     0
approv_in_adv              0
loan_type                  0
loan_purpose               0
Credit_Worthiness          0
open_credit                0
business_or_commercial     0
loan_amount                0
rate_of_interest           0
Interest_rate_spread       0
Upfront_charges            0
term                       0
Neg_ammortization          0
interest_only              0
lump_sum_payment           0
property_value             0
construction_type          0
occupancy_type             0
Secured_by                 0
total_units                0
income                     0
credit_type                0
Credit_Score               0
co-applicant_credit_type   0
age                        0
submission_of_application  0
LTV                        0
Region                     0
Security_Type              0
Status                     0
dtir1                      0
dtype: int64
```

```
loan_new = pd.concat([loan[CategoricalCols], loan_new.drop(CategoricalCols, axis = 1)], axis = 1)
```

```
loan_new.head(10)
```

	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	Neg_ammortizatio
0	cf	Sex Not Available	nopre	type1	p1	l1	nopc	nob/c	not_ne
1	cf	Male	nopre	type2	p1	l1	nopc	b/c	not_ne
2	cf	Male	pre	type1	p1	l1	nopc	nob/c	neg_amr
3	cf	Male	nopre	type1	p4	l1	nopc	nob/c	not_ne
4	cf	Joint	pre	type1	p1	l1	nopc	nob/c	not_ne
5	cf	Joint	pre	type1	p1	l1	nopc	nob/c	not_ne
6	cf	Joint	pre	type1	p3	l1	nopc	nob/c	not_ne
7	Missing	Female	nopre	type1	p4	l1	nopc	nob/c	not_ne
8	cf	Joint	nopre	type1	p3	l1	nopc	nob/c	not_ne
9	cf	Sex Not Available	nopre	type3	p3	l1	nopc	nob/c	not_ne

10 rows × 34 columns

```
loan_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148670 entries, 0 to 148669
Data columns (total 34 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   loan_limit                 148670 non-null  object
 1   Gender                     148670 non-null  object
 2   approv_in_adv              148670 non-null  object
 3   loan_type                  148670 non-null  object
 4   loan_purpose               148670 non-null  object
 5   Credit_Worthiness          148670 non-null  object
 6   open_credit                148670 non-null  object
 7   business_or_commercial     148670 non-null  object
```

```
 8   Neg_ammortization          148670 non-null  object
 9   interest_only              148670 non-null  object
 10  lump_sum_payment           148670 non-null  object
 11  construction_type          148670 non-null  object
 12  occupancy_type             148670 non-null  object
 13  Secured_by                 148670 non-null  object
 14  total_units                148670 non-null  object
 15  credit_type                148670 non-null  object
 16  co-applicant_credit_type   148670 non-null  object
 17  age                        148670 non-null  object
 18  submission_of_application  148670 non-null  object
 19  Region                     148670 non-null  object
 20  Security_Type              148670 non-null  object
 21  ID                         148670 non-null  int64
 22  year                       148670 non-null  int64
 23  loan_amount                148670 non-null  int64
 24  rate_of_interest           148670 non-null  float64
 25  Interest_rate_spread       148670 non-null  float64
 26  Upfront_charges            148670 non-null  float64
 27  term                       148670 non-null  float64
 28  property_value             148670 non-null  float64
 29  income                     148670 non-null  float64
 30  Credit_Score               148670 non-null  int64
 31  LTV                        148670 non-null  float64
 32  Status                     148670 non-null  int64
 33  dtir1                      148670 non-null  float64
dtypes: float64(8), int64(5), object(21)
memory usage: 38.6+ MB
```

We can see all missing values from the dataset are gone. Now as we temporarily encode categorical variables because we will encode them in later section so lets decode them.

2.4 Encoding Categorical Attributes

Categorical Features are those that contain qualitative data.This Section focuses primarily on methods that encode categorical data to numeric values.

Categorical variables/features are any feature type can be classified into three major types:

Nominal, Ordinal and Binary

Because of few potential issues with traditional approaches we now need to search for some unique approaches some of them are listed below:-

a) Supervised Encoding Methods

b) Approaching for Novel Categories

2.4.1 Supervised Encoding Methods

1) Effect or Likelihood Encoding

2) Target Encoding

3) Deep Learning Methods

2.4.2 Likelihood Encoding

```
del loan_temp
```

```
loan_demo = loan_new.copy()
```

```
from sklearn.preprocessing import LabelEncoder
```

```
encoder = LabelEncoder()
```

```
cat_cols = [col for col in loan.columns if loan_demo[col].dtype == 'object']
```

```
cat_cols
```

```
['loan_limit',
 'Gender',
 'approv_in_adv',
 'loan_type',
 'loan_purpose',
 'Credit_Worthiness',
 'open_credit',
 'business_or_commercial',
 'Neg_ammortization',
 'interest_only',
 'lump_sum_payment',
 'construction_type',
 'occupancy_type',
 'Secured_by',
 'total_units',
 'credit_type',
 'co-applicant_credit_type',
 'age',
 'submission_of_application',
 'Region',
 'Security_Type']
```

```
for col in cat_cols:
    loan_demo[col] = encoder.fit_transform(loan_demo[col])
```

```
loan_demo.head(10)
```

	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	Neg_ammortization
0	1	3	1	0	1	0	0	1	2
1	1	2	1	1	1	0	0	0	2
2	1	2	2	0	1	0	0	1	1
3	1	2	1	0	4	0	0	1	2
4	1	1	2	0	1	0	0	1	2
5	1	1	2	0	1	0	0	1	2
6	1	1	2	0	3	0	0	1	2
7	0	0	1	0	4	0	0	1	2
8	1	1	1	0	3	0	0	1	2
9	1	3	1	2	3	0	0	1	2

10 rows × 34 columns

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score
```

```
def likelihood_encoding(df, cat_cols, target_variable = "Status"):
    df_temp = df.copy()
    for col in cat_cols:
        effect = {}
        for category in df[col].unique():
            #print(category)

            try:
                temp = df[df[col] == category]
                lr = LogisticRegression()
```

```
            X = temp.drop(target_variable, axis = 1, inplace = False)
            y = temp[target_variable]
            lr.fit(X, y)

            effect[category] = accuracy_score(y, lr.predict(X))
        except Exception as E:
            print(E)

    for key, value in effect.items():
        effect[key] = np.log(effect[key] / (1 - effect[key] + 1e-6))

    df_temp.loc[:, col] = df_temp.loc[:, col].map(effect)
return df_temp
```

```
loan_demo = likelihood_encoding(loan_demo, cat_cols)
```

```
This solver needs samples of at least 2 classes in the data, but the data contains only one class: 1
This solver needs samples of at least 2 classes in the data, but the data contains only one class: 1
This solver needs samples of at least 2 classes in the data, but the data contains only one class: 1
This solver needs samples of at least 2 classes in the data, but the data contains only one class: 1
This solver needs samples of at least 2 classes in the data, but the data contains only one class: 1
```

```
loan_demo.head(10)
```

	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	Neg_ammortizatic
0	1.154236	0.915470	1.081446	1.221214	1.052378	1.134831	1.116422	1.206535	1.24384
1	1.154236	1.036897	1.081446	0.777966	1.052378	1.134831	1.116422	0.777966	1.24384
2	1.154236	1.036897	1.332389	1.221214	1.052378	1.134831	1.116422	1.206535	0.28602
3	1.154236	1.036897	1.081446	1.221214	1.210345	1.134831	1.116422	1.206535	1.24384
4	1.154236	1.439182	1.332389	1.221214	1.052378	1.134831	1.116422	1.206535	1.24384
5	1.154236	1.439182	1.332389	1.221214	1.052378	1.134831	1.116422	1.206535	1.24384
6	1.154236	1.439182	1.332389	1.221214	1.097893	1.134831	1.116422	1.206535	1.24384
7	1.028074	1.092456	1.081446	1.221214	1.210345	1.134831	1.116422	1.206535	1.24384
8	1.154236	1.439182	1.081446	1.221214	1.097893	1.134831	1.116422	1.206535	1.24384
9	1.154236	0.915470	1.081446	1.153569	1.097893	1.134831	1.116422	1.206535	1.24384

10 rows × 34 columns

```
loan_demo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148670 entries, 0 to 148669
Data columns (total 34 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   loan_limit                 148670 non-null  float64
 1   Gender                     148670 non-null  float64
 2   approv_in_adv              148670 non-null  float64
 3   loan_type                  148670 non-null  float64
 4   loan_purpose               148670 non-null  float64
 5   Credit_Worthiness          148670 non-null  float64
 6   open_credit                148670 non-null  float64
 7   business_or_commercial     148670 non-null  float64
 8   Neg_ammortization          148670 non-null  float64
 9   interest_only              148670 non-null  float64
 10  lump_sum_payment           148670 non-null  float64
 11  construction_type          148637 non-null  float64
 12  occupancy_type             148670 non-null  float64
 13  Secured_by                 148637 non-null  float64
 14  total_units                148670 non-null  float64
 15  credit_type                148670 non-null  float64
 16  co-applicant_credit_type   148670 non-null  float64
 17  age                        148470 non-null  float64
 18  submission_of_application  148470 non-null  float64
 19  Region                     148670 non-null  float64
 20  Security_Type              148637 non-null  float64
 21  ID                         148670 non-null  int64
 22  year                       148670 non-null  int64
 23  loan_amount                148670 non-null  int64
 24  rate_of_interest           148670 non-null  float64
```

```
 25  Interest_rate_spread       148670 non-null  float64
 26  Upfront_charges            148670 non-null  float64
 27  term                       148670 non-null  float64
 28  property_value             148670 non-null  float64
 29  income                     148670 non-null  float64
 30  Credit_Score               148670 non-null  int64
 31  LTV                        148670 non-null  float64
 32  Status                     148670 non-null  int64
 33  dtir1                      148670 non-null  float64
dtypes: float64(29), int64(5)
memory usage: 38.6 MB
```

```
del loan_demo
```

2.4.3 Target Encoding

```
loan_temp = loan_new.copy()
```

```
categoriacal = [col for col in loan_temp.columns if loan_temp[col].dtype == 'object']
```

```
categoriacal
```

```
['loan_limit',
 'Gender',
 'approv_in_adv',
 'loan_type',
 'loan_purpose',
 'Credit_Worthiness',
 'open_credit',
 'business_or_commercial',
 'Neg_ammortization',
 'interest_only',
 'lump_sum_payment',
 'construction_type',
 'occupancy_type',
 'Secured_by',
 'total_units',
 'credit_type',
 'co-applicant_credit_type',
 'age',
 'submission_of_application',
 'Region',
 'Security_Type']
```

```
for col in categoriacal:
    loan_temp[col] = encoder.fit_transform(loan_temp[col])
```

```
target_variable = "Status"
```

```
for col in cat_cols:
        weight = 7
        feat = loan_temp.groupby(col)[target_variable].agg(["mean", "count"])
        mean = feat['mean']
        count = feat['count']

        smooth = (count * mean + weight * mean) / (weight + count)

        loan_temp.loc[:, col] = loan_temp.loc[:, col].map(smooth)
```

```
loan_temp.head()
```

	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	Neg_ammortizatic
0	0.239708	0.285908	0.253272	0.227749	0.258768	0.243277	0.246709	0.230377	0.22380
1	0.239708	0.261914	0.253272	0.345439	0.258768	0.243277	0.246709	0.345439	0.22380
2	0.239708	0.261914	0.208937	0.227749	0.258768	0.243277	0.246709	0.230377	0.44596
3	0.239708	0.261914	0.253272	0.227749	0.229749	0.243277	0.246709	0.230377	0.22380
4	0.239708	0.191623	0.208937	0.227749	0.258768	0.243277	0.246709	0.230377	0.22380

5 rows × 34 columns

Target Encoding could be good choice for binary classification but for regression it is not, because it ignores intra-category variation of the target variable. This is addressed in Bayesian Target Encoding.

Target encoding has a tendency to overfit due to the target leakage.

Another problem is that some of the categories have few training examples, and the mean target value for these categories may assume extreme values, so encoding these values with mean may reduce the model performance.

```
loan['age'].value_counts()
```

```
age
45-54      34720
35-44      32818
55-64      32534
65-74      20744
25-34      19142
>74         7175
<25         1337
Missing      200
Name: count, dtype: int64
```

2.5 Deep Learning Methods

Another supervised approach comes from the deep learning literature on the analysis of textual data. In this case, large amounts of text can be cut up into individual words. Rather than making each of these words into its own indicator variable, word embedding or entity embedding approaches have been developed. Similar to the dimension reduction methods, the idea is to estimate a smaller set of numeric features that can be used to adequately represent the categorical predictors.

In addition to the dimension reduction, there is the possibility that these methods can estimate semantic relationships between words so that words with similar themes (e.g., “dog”, “pet”, etc.) have similar values in the new encodings. This technique is not limited to text data and can be used to encode any type of qualitative variable.

2.6 Approaches for Novel Categories

What if some new category introduce to some attribute in future how will we encode that variable then? If there is a possibility of encountering a new category in the future, one strategy would be to use the "other" category to capture new values.

While this approach may not be the most effective at extracting predictive information relative to the response for this specific category, it does enable the original model to be applied to new data without completely refitting and we do need to ensure that the "other" category is present in the training/testing data.

End of Advance Data Preprocessing

```
Start coding or generate with AI.
```

3) Data Preprocessing For Machine Learning

3.1 Dataset

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
import pandas as pd
```

```
loan=pd.read_csv('train_u6lujuX_CVtuZ9i (1).csv')
```

```
loan
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0
...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0

3.2 Understanding the Data

```
loan.shape
```

```
(614, 13)
```

```
loan.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
```

```
loan.head(10)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158.0	360.0	
8	LP001018	Male	Yes	2	Graduate	No	4006	1526.0	168.0	360.0	

```
loan.tail(10)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
604	LP002959	Female	Yes	1	Graduate	No	12000	0.0	496.0	360.0	
605	LP002960	Male	Yes	0	Not Graduate	No	2400	3800.0	NaN	180.0	
606	LP002961	Male	Yes	1	Graduate	No	3400	2500.0	173.0	360.0	
607	LP002964	Male	Yes	2	Not Graduate	No	3987	1411.0	157.0	360.0	
608	LP002974	Male	Yes	0	Graduate	No	3232	1950.0	108.0	360.0	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	

```
loan.columns.values
```

```
array(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome',
       'LoanAmount', 'Loan_Amount_Term', 'Credit_History',
       'Property_Area', 'Loan_Status'], dtype=object)
```

Basic Information

```
loan.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
```

Summary Statistics

```
loan.describe().T
```

	count	mean	std	min	25%	50%	75%	max
ApplicantIncome	614.0	5403.459283	6109.041673	150.0	2877.5	3812.5	5795.00	81000.0
CoapplicantIncome	614.0	1621.245798	2926.248369	0.0	0.0	1188.5	2297.25	41667.0
LoanAmount	592.0	146.412162	85.587325	9.0	100.0	128.0	168.00	700.0
Loan_Amount_Term	600.0	342.000000	65.120410	12.0	360.0	360.0	360.00	480.0
Credit_History	564.0	0.842199	0.364878	0.0	1.0	1.0	1.00	1.0

3.3 Cleaning Data

3.3.1 Handling Duplicated Values

```
loan.shape
```

```
(614, 13)
```

```
# Remove duplicates based on all columns
loan.drop_duplicates(inplace=True)
```

```
loan.shape
```

```
(614, 13)
```

No Duplicated Samples

Drop Unecessary Variables

```
# Drop Unecessary Variables
loan = loan.drop("Loan_ID", axis=1)
```

```
loan.shape
```

```
(614, 12)
```

Detect categorical and numerical features

Categorical Features

```
categorical_features = loan.select_dtypes(include=['object', 'category']).columns.tolist()
```

```
categorical_features
```

```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

Numerical Features

```
numerical_features = loan.select_dtypes(include=['int64', 'float64']).columns.tolist()
```

```
numerical_features
```

```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

Splitting Loan Dataset

Train Set: part of the original dataset on which the model learns.

Validation Set: A part of the main dataset that is not used in the training process and we use it only in the process of improving the performance of the model.

Test Set: A part of the main dataset on which the performance of the model is tested and reported as the final accuracy. These data should not be used when building the model because these data are real world examples.

```
X = loan.drop(columns='Loan_Status')
Y = loan['Loan_Status']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_val, y_train, y_val = train_test_split(X, Y, test_size=0.2,stratify=Y, shuffle=True, random_state = 40)
```

```
X_val.shape
```

```
(123, 11)
```

```
X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size=0.5, stratify=y_val, shuffle=True, random_state = 40)
```

```
 X_val.shape
```

```
(61, 11)
```

```
X_test.shape
```

```
(62, 11)
```

Handling Outlier Values

An Outlier is a sample that looks different from the majority of samples in the data set. Your dataset can contain outliers for two reasons:

1) Error in data collection and recording process : It is possible that in the process of data collection, values have been wrongly recorded by human (typing error) or computer system in the data collection file.

2. Actual data distribution : If the outliers are not due to "error in the data collection and recording process", it means that the actual distribution of the data is like this.

Distribution Of Numerical Features

```
import seaborn as sns
```

```
numerical_features
```

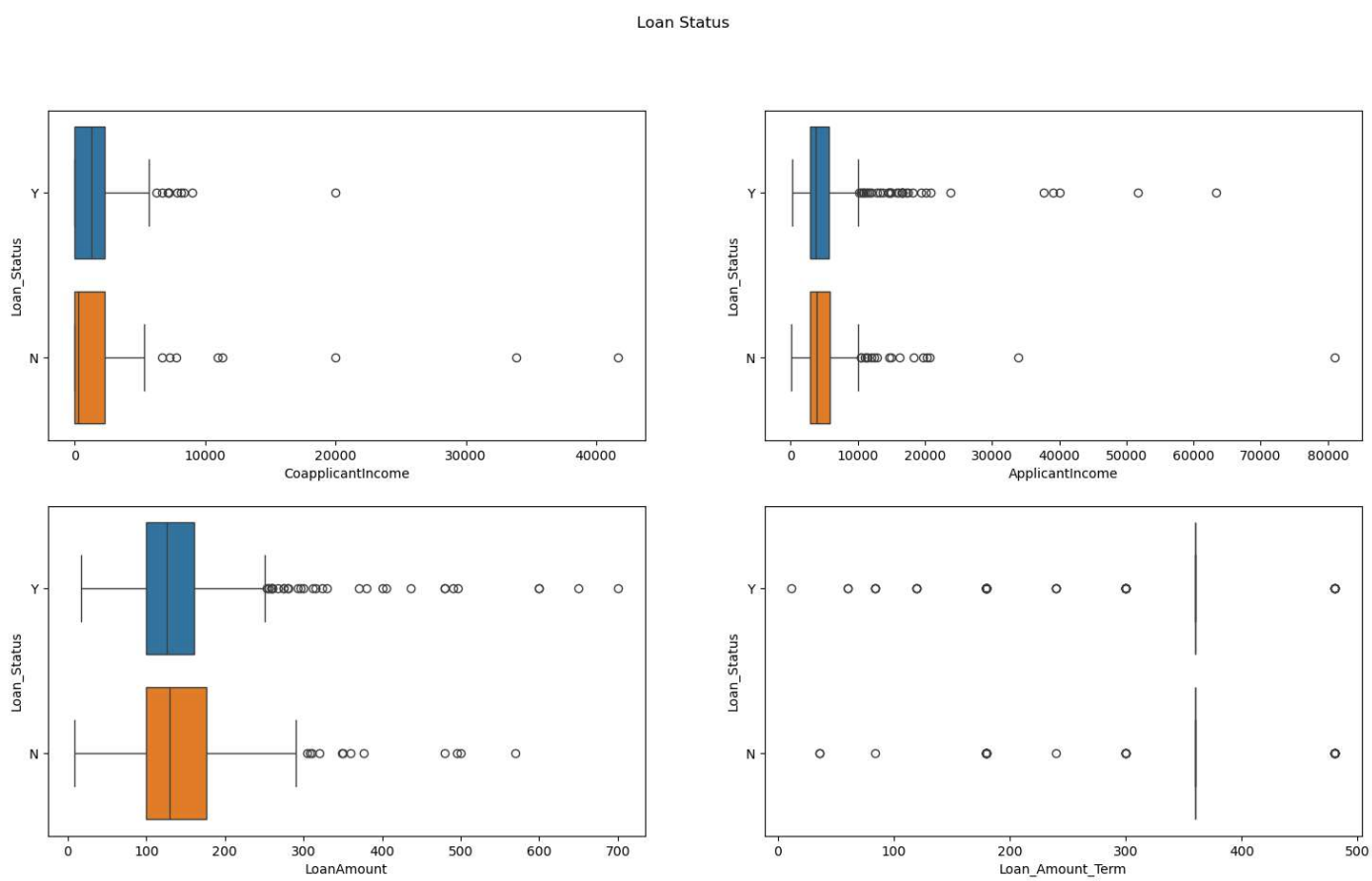
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

Here Continuous feature are : ApplicantIncome,CoapplicantIncome,LoanAmount,Loan_Amount_Term

```
import matplotlib.pyplot as plt
```

Plotting boxplot for continuous features

```
fig, axes = plt.subplots(2, 2, figsize=(18, 10))

fig.suptitle('Loan Status')

sns.boxplot(ax=axes[0, 0], x = loan['CoapplicantIncome'], y = loan['Loan_Status'], hue = loan['Loan_Status'])
sns.boxplot(ax=axes[0, 1], x = loan['ApplicantIncome'], y = loan['Loan_Status'], hue = loan['Loan_Status'])
sns.boxplot(ax=axes[1, 0], x = loan['LoanAmount'], y = loan['Loan_Status'], hue = loan['Loan_Status'])
sns.boxplot(ax=axes[1, 1], x = loan['Loan_Amount_Term'], y = loan['Loan_Status'], hue = loan['Loan_Status'])
plt.show()
```



Distribution Of Categorical Features

```
# Create an DataFrame for Analysis Categorical Features
categorical = pd.DataFrame(columns=['Features', 'Category', 'Count'])
```

```
for col in categorical_features:
    # Get the value counts for the column
    value_counts = loan[col].value_counts().reset_index()
    value_counts.columns = ['Category', 'Count']
    value_counts['Features'] = col
    categorical = pd.concat([categorical, value_counts], ignore_index=True)
```

```
# Display the result
grouped_df = categorical.groupby('Features').agg({'Category': list, 'Count': list})
grouped_df
```

Features	Category	Count
Dependents	[0, 1, 2, 3+]	[345, 102, 101, 51]
Education	[Graduate, Not Graduate]	[480, 134]
Gender	[Male, Female]	[489, 112]
Loan_Status	[Y, N]	[422, 192]
Married	[Yes, No]	[398, 213]
Property_Area	[Semiurban, Urban, Rural]	[233, 202, 179]
Self_Employed	[No, Yes]	[500, 82]

Handling Missing Values

many algorithms will fail if they see missing values. Also, the presence of missing values can cause errors in the exploratory analysis of the dataset.

To handle this problem, there are different ways, the most important of which are:

1) Remove rows/samples containing missing values

2) Remove columns/attributes containing missing values

3) Filling missing values with statistical characteristics or a constant value

4) Predicting missing values

5) Using an algorithm insensitive to missing values

Missing Value Analysis

```
Miss_Train = X_train.isna().sum()
Miss_Train
```

```
Gender               6
Married              2
Dependents          11
Education            0
Self_Employed       29
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          21
Loan_Amount_Term    12
Credit_History      42
Property_Area        0
dtype: int64
```

```
Miss_Val = X_val.isna().sum()
Miss_Val
```

```
Gender              4
Married             0
Dependents          2
Education           0
Self_Employed       2
ApplicantIncome     0
CoapplicantIncome   0
LoanAmount          1
Loan_Amount_Term    0
Credit_History      4
Property_Area       0
dtype: int64
```

```
Miss_test = X_test.isna().sum()
Miss_test
```

```
Gender              3
Married             1
Dependents          2
Education           0
Self_Employed       1
ApplicantIncome     0
CoapplicantIncome   0
LoanAmount          0
Loan_Amount_Term    2
Credit_History      4
Property_Area       0
dtype: int64
```

```
output_train = pd.DataFrame(Miss_Train, columns=['Missing Values X_train'])
output_train
```

	Missing Values X_train
Gender	6
Married	2
Dependents	11
Education	0
Self_Employed	29
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	21
Loan_Amount_Term	12
Credit_History	42
Property_Area	0

```
output_val = pd.DataFrame(Miss_Val, columns=['Missing Values X_val'])
output_val
```

	Missing Values X_val
Gender	4
Married	0
Dependents	2
Education	0
Self_Employed	2
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	1
Loan_Amount_Term	0
Credit_History	4
Property_Area	0

```
output_test = pd.DataFrame(Miss_test, columns=['Missing Values X_test'])
output_test
```

	Missing Values X_test
Gender	3
Married	1
Dependents	2
Education	0
Self_Employed	1
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	2
Credit_History	4
Property_Area	0

After Concatenation

```
output = pd.concat([output_train, output_val,output_test], axis=1, join='inner')
output
```

	Missing Values X_train	Missing Values X_val	Missing Values X_test
Gender	6	4	3
Married	2	0	1
Dependents	11	2	2
Education	0	0	0
Self_Employed	29	2	1
ApplicantIncome	0	0	0
CoapplicantIncome	0	0	0
LoanAmount	21	1	0
Loan_Amount_Term	12	0	2
Credit_History	42	4	4
Property_Area	0	0	0

Finding Rows with Null Values

```
null_loan = loan[loan.isnull().any(axis=1)]
null_loan
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hist
0	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	
11	Male	Yes	2	Graduate	NaN	2500	1840.0	109.0	360.0	
16	Male	No	1	Not Graduate	No	3596	0.0	100.0	240.0	
19	Male	Yes	0	Graduate	NaN	2600	3500.0	115.0	NaN	
23	NaN	Yes	2	Not Graduate	No	3365	1917.0	112.0	360.0	
...	...	...	...	...	...	...	...	...	...	...
592	NaN	No	3+	Graduate	Yes	9357	0.0	292.0	360.0	
597	Male	No	NaN	Graduate	No	2987	0.0	88.0	360.0	
600	Female	No	3+	Graduate	NaN	416	41667.0	350.0	180.0	
601	Male	Yes	0	Not Graduate	NaN	2894	2792.0	155.0	360.0	

```
len(null_loan)
```

```
134
```

Finding Rows with 50% or More Null Values

```
threshold = 0.5
```

```
null_threshold = int(threshold * len(loan.columns))
```

```
null_rows = loan[loan.apply(lambda x: x.isnull().sum(), axis=1) >= null_threshold]
```

```
null_rows.shape
```

```
(0, 12)
```

```
print(f"The number of rows consisting of more than 50% missing values is {len(null_rows)}")
```

```
The number of rows consisting of more than 50% missing values is 0
```

3.3.2 Handling Missing Values of Continuous Features

calculate mean columns

```
def calculate_mean(df, column):
    mean = df[column].mean().round()
    return mean
```

```
for col in ["LoanAmount","Loan_Amount_Term"]:
    print(f'Mean {col} in Trainset is: {calculate_mean(X_train, col)}')
    print(f'Mean {col} in Valset is: {calculate_mean(X_val, col)}')
    print(f'Mean {col} in Testset is: {calculate_mean(X_test, col)}')
```

```
Mean LoanAmount in Trainset is: 148.0
Mean LoanAmount in Valset is: 140.0
Mean LoanAmount in Testset is: 140.0
Mean Loan_Amount_Term in Trainset is: 341.0
Mean Loan_Amount_Term in Valset is: 346.0
Mean Loan_Amount_Term in Testset is: 346.0
```

filling missing values by mean

```
def fill_missing_values_by_mean(df, column):
    for col in cols:
        df[col].fillna(df[col].mean(), inplace=True)
    return df
```

```
cols = ["LoanAmount","Loan_Amount_Term"]
X_train = fill_missing_values_by_mean(X_train, cols)
X_train[["LoanAmount","Loan_Amount_Term"]].isnull().any()
```

```
LoanAmount          False
Loan_Amount_Term    False
dtype: bool
```

```
X_val = fill_missing_values_by_mean(X_val, cols)
X_val[["LoanAmount","Loan_Amount_Term"]].isnull().any()
```

```
LoanAmount          False
Loan_Amount_Term    False
dtype: bool
```

```
X_test = fill_missing_values_by_mean(X_test, cols)
X_test[["LoanAmount","Loan_Amount_Term"]].isnull().any()
```

```
LoanAmount          False
Loan_Amount_Term    False
dtype: bool
```

Handling Missing Values of Categorical Features

Common methods for handling missing values specific to categorical attributes include:

1) Filling with the most common category

2) Adding a weight variable

3) Creating a new category

4) Predicting the missing category: One of the more professional approaches mentioned in the previous chapter is to use a machine learning model for predicting the missing category.

```
categorialVariable = ['Gender', 'Married','Education', 'Dependents', 'Self_Employed', 'Property_Area', 'Credit_History']
```

Creating a function : Most frequent value for each categorical feature

```
def Frequent_Value(df, cols):
    list_of_most_frequent = {}
    for col in cols:
        f = df[col].mode().iloc[0]
        list_of_most_frequent[col] = f
    return pd.DataFrame(list_of_most_frequent, index=['Most Frequent'])
```

```
# Most frequent value for each categorical feature in X_train
Frequent_Value(X_train, categorialVariable)
```

	Gender	Married	Education	Dependents	Self_Employed	Property_Area	Credit_History
Most Frequent	Male	Yes	Graduate	0	No	Semiurban	1.0

```
# Most frequent value for each categorical feature in X_Val
Frequent_Value(X_val, categorialVariable)
```

	Gender	Married	Education	Dependents	Self_Employed	Property_Area	Credit_History
Most Frequent	Male	Yes	Graduate	0	No	Semiurban	1.0

```
# Most frequent value for each categorical feature in X_test
Frequent_Value(X_test, categorialVariable)
```

	Gender	Married	Education	Dependents	Self_Employed	Property_Area	Credit_History
Most Frequent	Male	Yes	Graduate	0	No	Semiurban	1.0

Handling Missing Values in Categorical Features

```
def Fill_Null(df,cols):

    for col in cols:
        df[col].fillna(df[col].mode()[0], inplace=True)

    return df
```

```
X_train = Fill_Null(X_train, categorialVariable)
X_train.isnull().any()
```

```
Gender              False
Married             False
Dependents          False
Education           False
Self_Employed       False
ApplicantIncome     False
CoapplicantIncome   False
LoanAmount          False
Loan_Amount_Term    False
Credit_History      False
Property_Area       False
dtype: bool
```

```
X_val = Fill_Null(X_val, categorialVariable)
X_val.isnull().any()
```

```
Gender              False
Married             False
Dependents          False
Education           False
Self_Employed       False
ApplicantIncome     False
CoapplicantIncome   False
LoanAmount          False
Loan_Amount_Term    False
Credit_History      False
Property_Area       False
dtype: bool
```

```
X_test = Fill_Null(X_test, categorialVariable)
X_test.isnull().any()
```

```
Gender              False
Married             False
Dependents          False
Education           False
Self_Employed       False
ApplicantIncome     False
CoapplicantIncome   False
LoanAmount          False
Loan_Amount_Term    False
Credit_History      False
Property_Area       False
dtype: bool
```

```
# Changing the data type of Credit_History to int using astype()
for df in [X_train, X_val, X_test]:
    df['Credit_History'] = df['Credit_History'].astype(int)
```

```
X_train.isna().sum().sum()
```

```
0
```

```
X_val.isna().sum().sum()
```

```
0
```

```
X_test.isna().sum().sum()
```

```
0
```

3.4 Feature Engineering

Feature engineering involves modifying existing features or creating new ones to enhance the model's performance

Let's review the characteristics of a good feature together:

1) High predictive power: The most important characteristic a feature should have is to provide the model with high predictive power.

2) Quick calculation: A good feature should be calculated and built quickly.

3) Trustworthiness: Additional information, not present in the dataset, can be used to train the models.

4) Irrelevance of features: When the value of a feature increases or decreases with the value of another attribute, we say that the two attributes are related.

Convert Categorical Features

Categorical features can be divided into the following four main groups:

1) Nominal: These consist of two or more categories that have no inherent order between their values. For example, "gender" is a nominal variable.

2) Ordinal: These have a specific order or hierarchy between the values of the groups. For example, age groups such as "young," "middle-aged," and "old" are ordinal variables.

In this dataset, we have seven categorical features, one of which is ordinal, and the remaining six are nominal.

```
Nominal_fetaures = ['Gender', 'Married','Education', 'Self_Employed', 'Property_Area']
```

Create a function to encode Categorical Features

```
def encode_categorical_features(df, cols):

    for col in cols:
        dummies = pd.get_dummies(df[col], dtype=int, prefix=col)
        df = pd.concat([df, dummies], axis=1)
        df = df.drop(labels=col, axis=1)
    return df
```

Encode the categorical features in the training and test sets

```
X_train = encode_categorical_features(X_train, Nominal_fetaures)
X_train
```

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Gender_Male	Married_N
101	0	4843	3806.0	151.0	360.0	1	0	1	
116	0	3167	2283.0	154.0	360.0	1	1	0	
191	0	12000	0.0	164.0	360.0	1	0	1	
237	0	3463	0.0	122.0	360.0	1	1	0	
222	0	2971	2791.0	144.0	360.0	1	0	1	
...	...	...	...	...	...	...	...	...	
579	0	3182	2917.0	161.0	360.0	1	0	1	
215	3+	3850	983.0	100.0	360.0	1	0	1	
92	2	3273	1820.0	81.0	360.0	1	0	1	
124	0	4300	2014.0	194.0	360.0	1	0	1	
570	1	3417	1750.0	186.0	360.0	1	0	1	

491 rows × 17 columns

```
X_val = encode_categorical_features(X_val, Nominal_fetaures )
X_val
```

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Gender_Male	Married_N
606	1	3400	2500.0	173.000000	360.0	1	0	1	
268	0	3418	0.0	135.000000	360.0	1	1	0	
422	0	1820	1719.0	100.000000	360.0	1	0	1	
220	0	2221	0.0	60.000000	360.0	0	0	1	
51	0	3086	0.0	120.000000	360.0	1	1	0	
...	...	...	...	...	...	...	...	...	
225	0	3250	0.0	170.000000	360.0	1	0	1	
348	0	6333	4583.0	259.000000	360.0	1	0	1	
103	0	4652	3583.0	140.116667	360.0	1	0	1	
167	0	2439	3333.0	129.000000	360.0	1	0	1	
56	0	2132	1591.0	96.000000	360.0	1	0	1	

61 rows × 17 columns

```
X_test = encode_categorical_features(X_test, Nominal_fetaures )
X_test
```

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Gender_Male	Married_N
340	3+	2647	1587.0	173.0	360.0	1	0	1	
478	1	16667	2250.0	86.0	360.0	1	0	1	
190	0	4885	0.0	48.0	360.0	1	0	1	
228	0	4758	0.0	158.0	480.0	1	0	1	
353	0	5500	0.0	105.0	360.0	0	1	0	
...	...	...	...	...	...	...	...	...	
370	0	15759	0.0	55.0	360.0	1	1	0	
303	1	1625	1803.0	96.0	360.0	1	0	1	
31	0	3167	0.0	74.0	360.0	1	0	1	
544	0	3017	663.0	102.0	360.0	1	1	0	
502	2	4865	5624.0	208.0	360.0	1	0	1	

62 rows × 17 columns

```
loan[["Dependents"]].head(15)
```

	Dependents
0	0
1	1
2	0
3	0
4	0
5	2
6	0
7	3+
8	2
9	1
10	2
11	2
12	2
13	0
14	2

```
encoder = {'0': 1/4, '1': 2/4, '2': 3/4, '3+': 4/4}
for df in [X_train, X_val, X_test]:
    df["Dependents"] = df['Dependents'].map(encoder)
```

After encoding

```
X_train[["Dependents"]].tail(15)
```

	Dependents
60	0.25
412	0.25
250	0.25
506	0.25
175	0.25
601	0.25
334	0.50
200	0.50
74	1.00
233	0.25
579	0.25
215	1.00
92	0.75
124	0.25
570	0.50

3.5 Balancing TrainSet

Convet target column to int

```
mapping = {'Y': 1,'N': 0}
y_train = pd.Series(y_train).map(mapping)
y_train
```

```
101    1
116    1
191    0
237    1
222    1
      ..
579    1
215    1
92     1
124    1
570    1
Name: Loan_Status, Length: 491, dtype: int64
```

```
y_val = pd.Series(y_val).map(mapping)
y_val
```

```
606    1
268    0
422    1
220    0
51     1
      ..
225    0
348    1
103    1
167    1
56     1
Name: Loan_Status, Length: 61, dtype: int64
```

```
y_test = pd.Series(y_test).map(mapping)
y_test
```

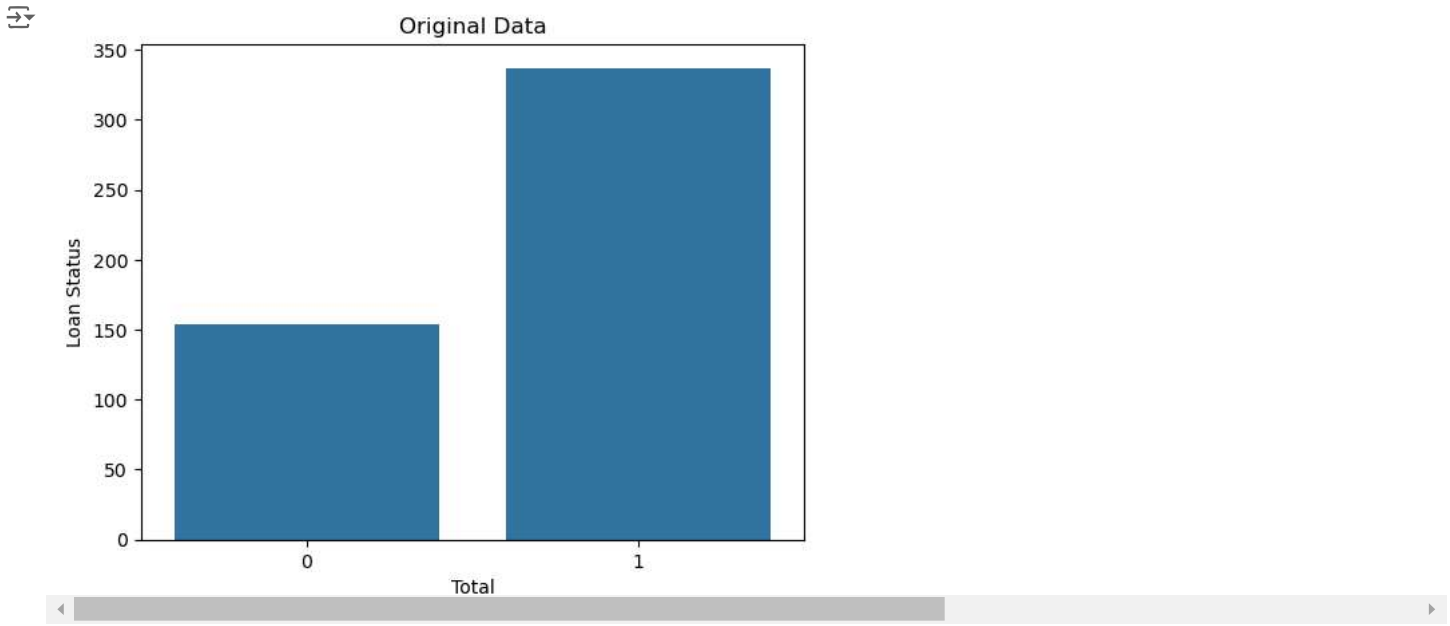
```
340    0
478    1
190    1
228    1
353    0
```

```
      ..
370    1
303    1
31     0
544    1
502    1
Name: Loan_Status, Length: 62, dtype: int64
```

Using SMOTE Technique to Balancing TrainSet

Orginal figure

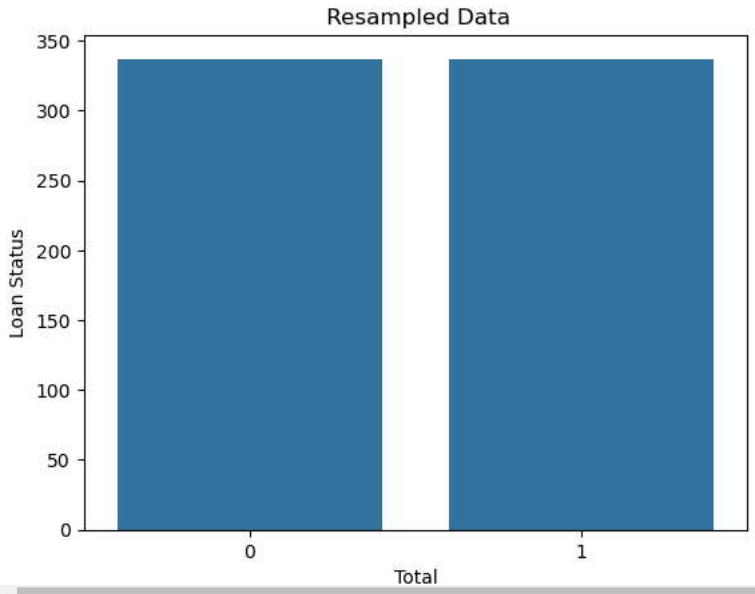
```
sns.countplot(x=y_train, data=X_train)
plt.ylabel('Loan Status')
plt.xlabel('Total')
plt.title('Original Data')
plt.show()
```



After Resampling

```
from imblearn.over_sampling import SMOTE
```

```
X_train, y_train = SMOTE().fit_resample(X_train, y_train)
sns.countplot(x=y_train, data=X_train)
plt.ylabel('Loan Status')
plt.xlabel('Total')
plt.title(' Resampled Data')
plt.show()
```



Dimensionality Reduction

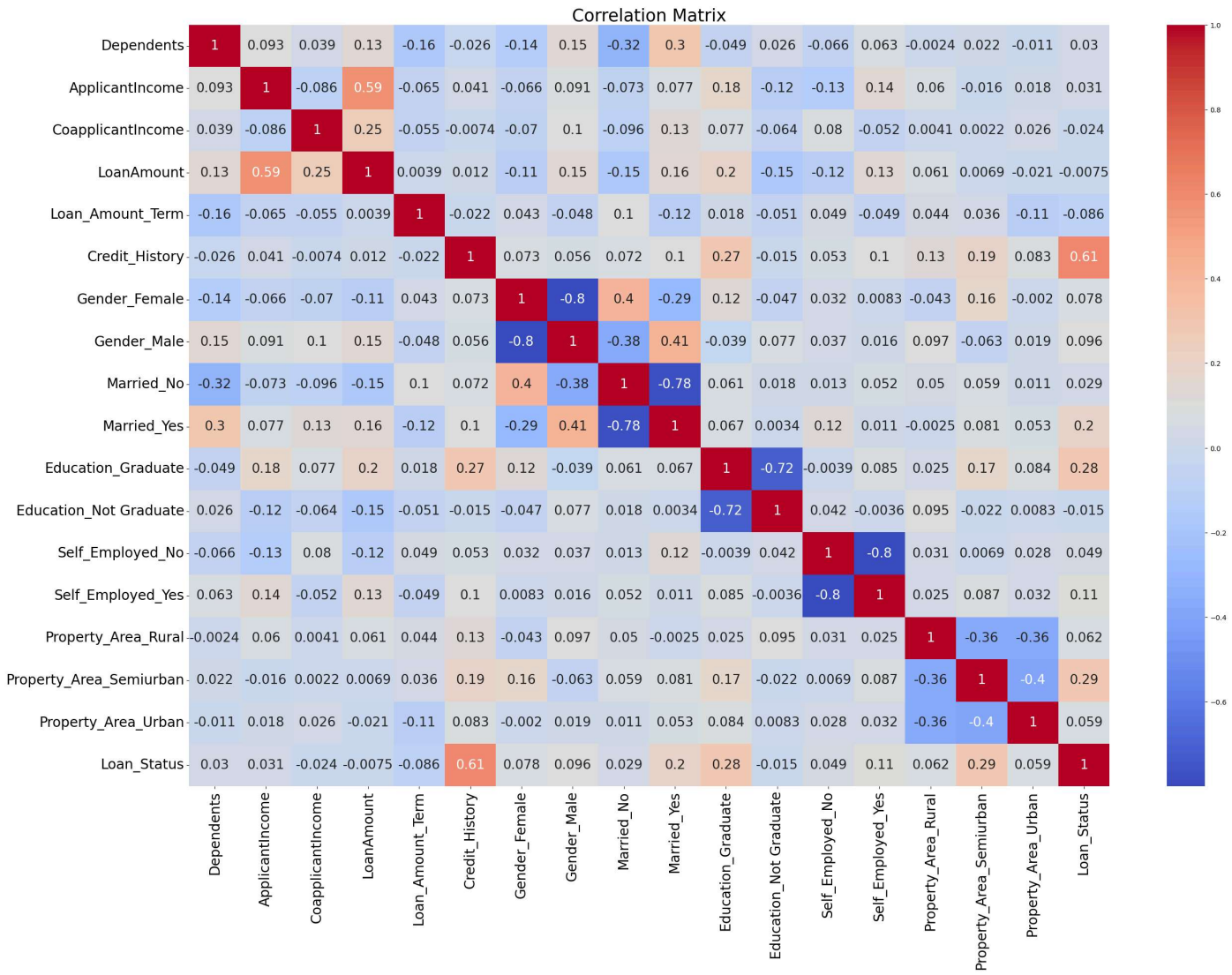
Dimensionality Reduction methods consist of two subsections:

1) Feature Selection and

2) Feature Extraction.

Correlation Analysis

```
train_copy = pd.concat([X_train, y_train], axis=1)
corr_matrix = train_copy.corr()

fig, ax = plt.subplots(figsize=(30, 20))


sns.heatmap(corr_matrix, cmap='coolwarm', annot=True, annot_kws={"size": 19}, ax=ax)


ax.set_title('Correlation Matrix', fontsize=25)
ax.tick_params(axis='x', labelsize=20)
ax.tick_params(axis='y', labelsize=20)

plt.show()
```



3.6 Mutual Information Calculation and Plotting

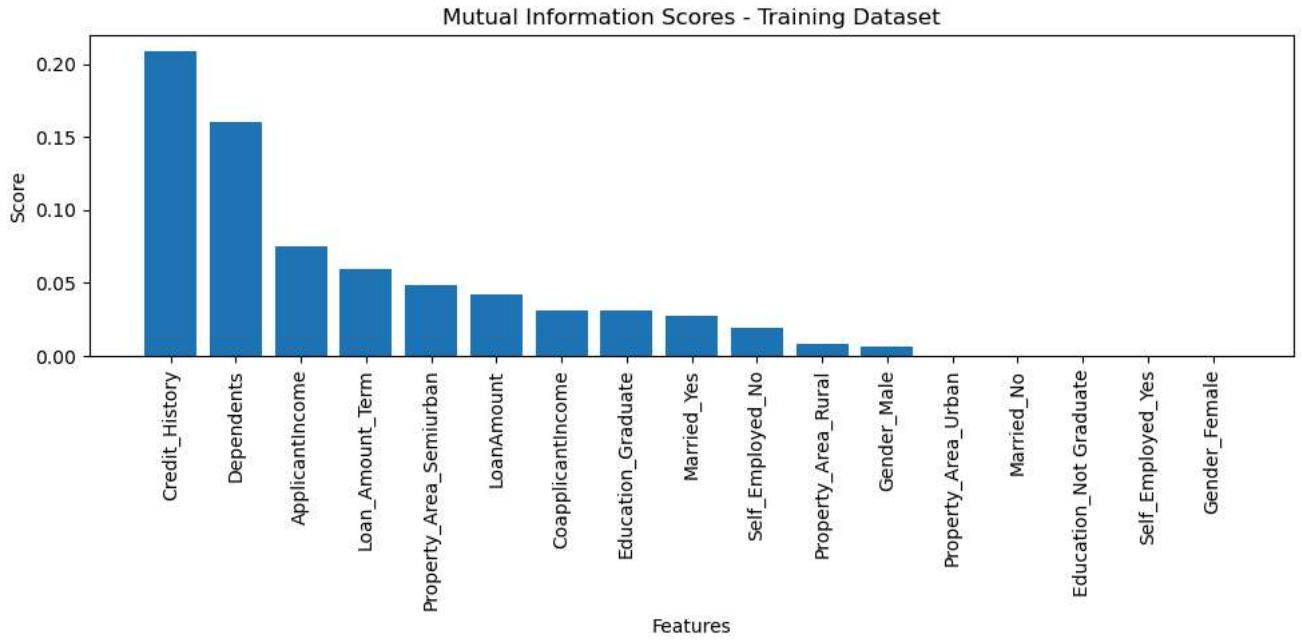
```
from sklearn.feature_selection import mutual_info_classif
```

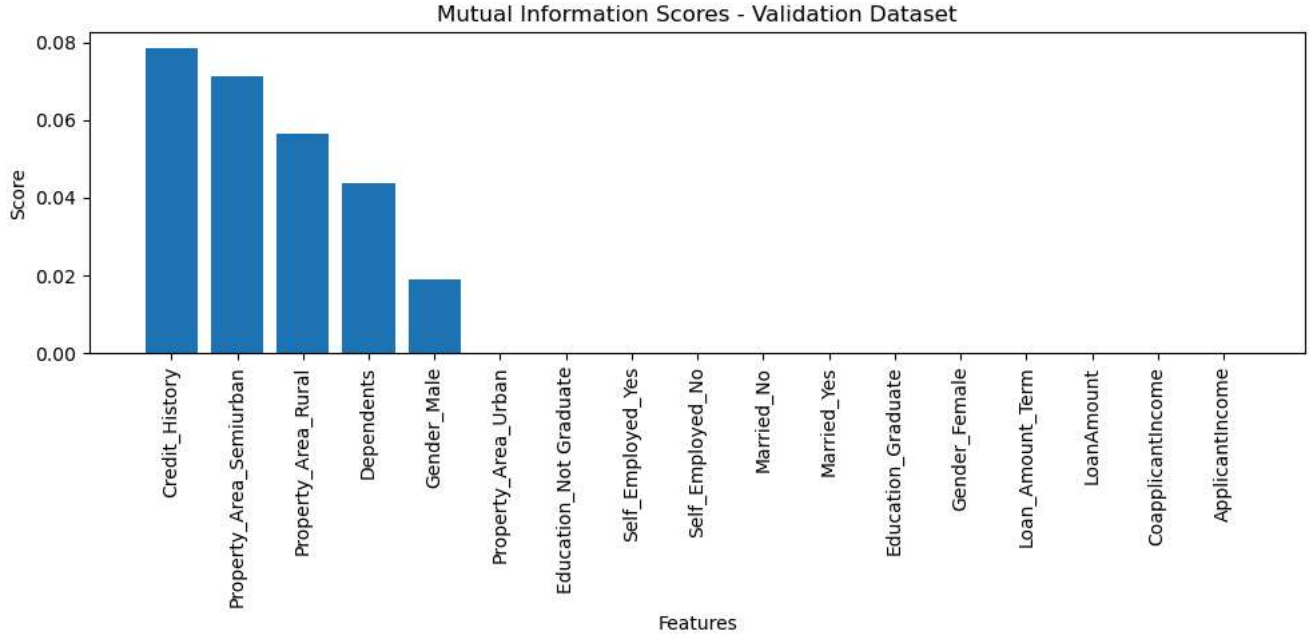
```
datasets = [("Training Dataset", X_train, y_train),
            ("Validation Dataset", X_val, y_val),
            ("Testing Dataset", X_test, y_test)]
```

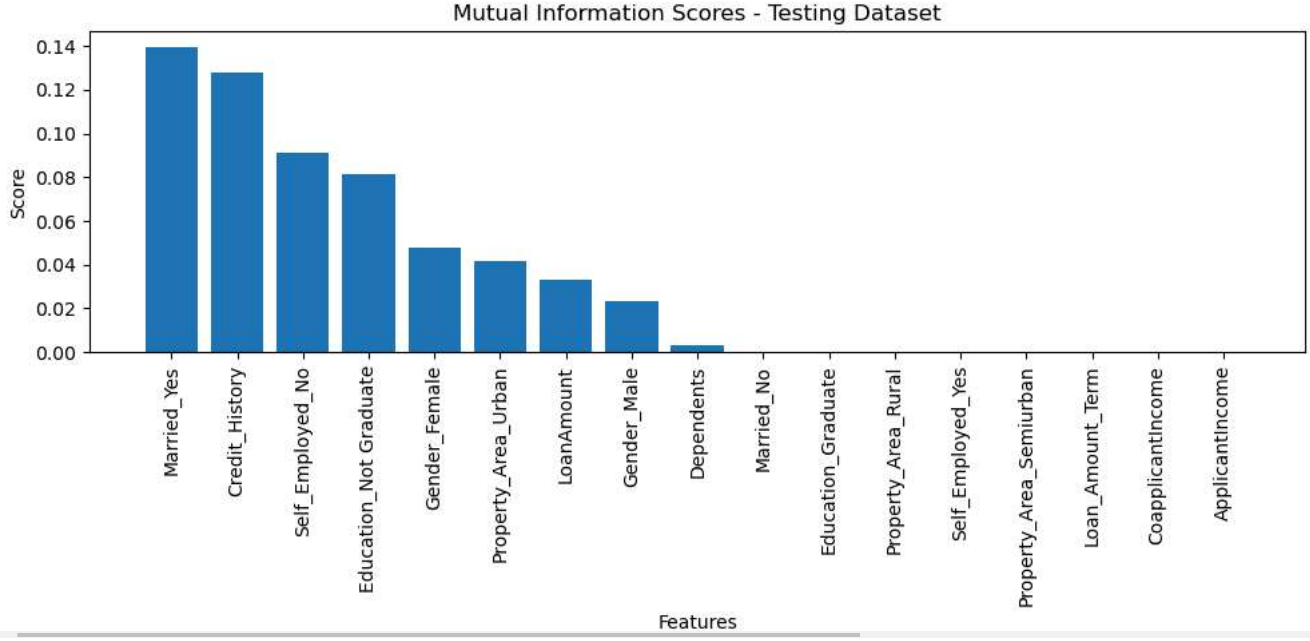
```
import numpy as np
```

```
for dataset_name, X, y in datasets:

    feature_names = X.columns
    scores = mutual_info_classif(X, y)


    sorted_indices = np.argsort(scores)[::-1]
    sorted_scores = scores[sorted_indices]
    sorted_features = feature_names[sorted_indices]


    plt.figure(figsize=(10, 5))
    plt.bar(range(len(sorted_scores)), sorted_scores)
    plt.xticks(range(len(sorted_scores)), sorted_features, rotation='vertical')
    plt.title(f"Mutual Information Scores - {dataset_name}")
    plt.xlabel("Features")
    plt.ylabel("Score")
    plt.tight_layout()
    plt.show()
```

[4]


Mutual Information Scores - Training Dataset
Score
0.20
0.15
0.10
0.05
0.00
Credit_History
Dependents
ApplicantIncome
Loan_Amount_Term
Property_Area_Semiurban
LoanAmount
CoapplicantIncome
Education_Graduate
Married_Yes
Self_Employed_No
Property_Area_Rural
Gender_Male
Property_Area_Urban
Married_No
Education_Not Graduate
Self_Employed_Yes
Gender_Female
Features


Mutual Information Scores - Validation Dataset
Score
0.08
0.06
0.04
0.02
0.00
Credit_History
Property_Area_Semiurban
Property_Area_Rural
Dependents
Gender_Male
Property_Area_Urban
Education_Not Graduate
Self_Employed_Yes
Self_Employed_No
Married_No
Married_Yes
Education_Graduate
Gender_Female
Loan_Amount_Term
LoanAmount
CoapplicantIncome
ApplicantIncome
Features


Mutual Information Scores - Testing Dataset
Score
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Married_Yes
Credit_History
Self_Employed_No
Education_Not Graduate
Gender_Female
Property_Area_Urban
LoanAmount
Gender_Male
Dependents
Married_No
Education_Graduate
Property_Area_Rural
Self_Employed_Yes
Property_Area_Semiurban
Loan_Amount_Term
CoapplicantIncome
ApplicantIncome
Features

Dropping Features with 0.0 scores in MI (Mutual Information)

```
zeroScore_fetaures = ['Dependents', 'CoapplicantIncome','Loan_Amount_Term', 'Gender_Female', 'Married_Yes',
                'Education_Graduate', 'Self_Employed_No', 'Self_Employed_Yes','Property_Area_Urban']
```

```
def Dropping(df, cols):

    for col in cols:
        df = df.drop(labels=col, axis=1)
    return df
```

Before Dropping:

```
X_train.shape,X_val.shape,X_test.shape
```

```
((674, 17), (61, 17), (62, 17))
```

After dropping :

```
X_train = Dropping(X_train, zeroScore_fetaures)
X_train.shape
```

```
(674, 8)
```

```
X_val = Dropping(X_val, zeroScore_fetaures)
X_val.shape
```

```
(61, 8)
```

```
X_test = Dropping(X_test, zeroScore_fetaures)
X_test.shape
```

```
(62, 8)
```

3.7 Feature Scaling

Generally, two methods can be used for feature scaling:

1) Scaling and

2) Normalization

Scalling the dataset using StandardScaler

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X_validation = X_val.values
```

```
X_test = X_test.values
```

```
scaled_x_train = scaler.fit_transform(X_train)
scaled_x_train
```

```
array([[-0.0680808 ,  0.0510223 ,  0.6495698 , ..., -0.49350019,
        -0.56707021,  1.58442945],
       [-0.36338475,  0.08728481,  0.6495698 , ..., -0.49350019,
        -0.56707021,  1.58442945],
       [ 1.19295161,  0.20815987,  0.6495698 , ..., -0.49350019,
        -0.56707021,  1.58442945],
       ...,
       [ 2.44587902,  4.72168357,  0.6495698 , ..., -0.49350019,
         1.76345006, -0.63114202],
       [-0.42047215, -0.06399136, -1.53948043, ..., -0.49350019,
        -0.56707021, -0.63114202],
       [-0.3896379 ,  0.05424353, -1.53948043, ..., -0.49350019,
        -0.56707021, -0.63114202]])
```

```
scaled_x_val = scaler.transform(X_validation)
scaled_x_val
```

```
array([[-0.32233116,  0.31694743,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.31915964, -0.1423778 ,  0.6495698 , -1.86106855,  1.5232193 ,
        -0.49350019,  1.76345006, -0.63114202],
       [-0.60072032, -0.56544051,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.53006585, -1.04894075, -1.53948043,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.3776566 , -0.32369039,  0.6495698 , -1.86106855,  1.5232193 ,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.01099341,  0.64331009,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.50451748, -0.13029029,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021,  1.58442945],
       [ 0.07710443,  0.40155997,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019,  1.76345006, -0.63114202],
       [-0.51949411, -0.40830293,  0.6495698 ,  0.53732572, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [-0.3774804 , -0.13029029, -1.53948043,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.27528691, -0.40830293,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.46328769, -0.37204041, -1.53948043,  0.53732572, -0.65650429,
         2.02634168, -0.56707021,  1.58442945],
       [-0.31070224, -0.20281533,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019,  1.76345006, -0.63114202],
       [ 0.10318139,  1.98502325,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019,  1.76345006, -0.63114202],
       [ 1.64806503,  3.49596149,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.27652028,  0.48617251,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.16710277, -0.56544051,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [ 5.72452806,  0.0631098 ,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.30083529, -0.79510312,  0.6495698 ,  0.53732572, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [-0.1579406 ,  0.15980985, -1.53948043,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.11900135,  0.13563484,  0.6495698 ,  0.53732572, -0.65650429,
         2.02634168, -0.56707021, -0.63114202],
       [-0.084467  , -0.63796554,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021,  1.58442945],
       [ 1.28104945,  2.0938108 ,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019,  1.76345006, -0.63114202],
       [ 0.69378928,  0.4619975 ,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019,  1.76345006, -0.63114202],
       [-0.60424423, -1.20607832,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.32849801, -0.42039044, -1.53948043,  0.53732572, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [-0.04922787, -0.26325286,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.57023847, -0.40830293,  0.6495698 ,  0.53732572, -0.65650429,
         2.02634168, -0.56707021,  1.58442945],
       [ 0.14987324,  0.42573498,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019,  1.76345006, -0.63114202],
```

```
scaled_x_test = scaler.transform(X_test)
scaled_x_test
```

```
array([[-0.4550065 ,  0.31694743,  0.6495698 ,  0.53732572, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [ 2.01525681, -0.73466559,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.06068059, -1.19399082,  0.6495698 ,  0.53732572,  1.5232193 ,
         2.02634168,  1.76345006, -0.63114202],
       [-0.08305744,  0.13563484,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [ 0.04767975, -0.50500298, -1.53948043, -1.86106855, -0.65650429,
        -0.49350019,  1.76345006, -0.63114202],
       [-0.01856982, -0.27534037, -1.53948043,  0.53732572,  1.5232193 ,
        -0.49350019,  1.76345006, -0.63114202],
       [-0.3936904 , -1.04894075,  0.6495698 , -1.86106855,  1.5232193 ,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.60072032, -0.62587804,  0.6495698 ,  0.53732572, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [-0.74520077, -0.44456545,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.32973138, -0.80719063,  0.6495698 ,  0.53732572,  1.5232193 ,
         2.02634168, -0.56707021,  1.58442945],
       [ 0.35602217, -0.44456545, -1.53948043,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.31704529, -0.22699034, -1.53948043,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.4222341 , -0.91597818,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021, -0.63114202],
       [ 0.86505147,  1.36856044,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.28356811, -1.47200345,  0.6495698 ,  0.53732572,  1.5232193 ,
         2.02634168, -0.56707021,  1.58442945],
       [-0.46170193,  0.22024738, -1.53948043,  0.53732572, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [-0.14613549, -0.23907785, -1.53948043,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.52442759, -0.44456545,  0.6495698 ,  0.53732572, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [-0.34805573, -1.16981581,  0.6495698 ,  0.53732572,  1.5232193 ,
        -0.49350019, -0.56707021, -0.63114202],
       [ 3.27258911,  2.6981861 ,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019,  1.76345006, -0.63114202],
       [ 0.33699304, -0.10611528,  0.6495698 , -1.86106855, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
       [ 0.10053845, -0.17864032,  0.6495698 ,  0.53732572,  1.5232193 ,
         2.02634168, -0.56707021,  1.58442945],
       [ 0.12432487, -0.16655281,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021,  1.58442945],
       [-0.58169118, -0.56544051,  0.6495698 , -1.86106855, -0.65650429,
         2.02634168, -0.56707021,  1.58442945],
       [-0.46628302, -0.32369039,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [ 0.07886638, -0.56544051,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019,  1.76345006, -0.63114202],
       [ 2.01508061,  1.54987303,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [ 0.97270702,  1.99711075,  0.6495698 ,  0.53732572, -0.65650429,
        -0.49350019, -0.56707021, -0.63114202],
       [-0.19899419, -0.27534037,  0.6495698 , -1.86106855, -0.65650429,
         2.02634168,  1.76345006, -0.63114202],
```

3.8 Feature Extraction

One of the most important and basic Feature Extraction algorithms is Principal Component Analysis (PCA). In general, the goal of this algorithm is to map data from higher-dimensional spaces to lower-dimensional spaces, preserving as much information and variance as possible.

Dimensionality Reduction with PCA

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = 5)
```

Before feature extraction :

```
scaled_x_train.shape,scaled_x_val.shape,scaled_x_test.shape
```

```
((674, 8), (61, 8), (62, 8))
```

```
train_reduced = pca.fit_transform(scaled_x_train)
```

```
val_reduced = pca.transform(scaled_x_val)
```

```
test_reduced = pca.transform(scaled_x_test)
```

After feature extraction :

```
train_reduced.shape,val_reduced.shape,test_reduced.shape
```

```
((674, 5), (61, 5), (62, 5))
```

3.9 Save Preprocessed DataSets ¶

Generate generic column names based on the number of features

```
num_features = train_reduced.shape[1]
column_names = [f'Feature_{i+1}' for i in range(num_features)]
```

Convert arrays to DataFrames

```
train_reduced_df = pd.DataFrame(train_reduced, columns=column_names)
```

```
val_reduced_df = pd.DataFrame(val_reduced, columns=column_names)
```

```
test_reduced_df = pd.DataFrame(test_reduced, columns=column_names)
```

```
train_reduced_df.to_csv('train_reduced.csv', index=False)
val_reduced_df.to_csv('val_reduced.csv', index=False)
test_reduced_df.to_csv('test_reduced.csv', index=False)
```

3.10 Load the preprocessed datasets

```
pd.read_csv('train_reduced.csv')
```

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5
0	-0.518782	1.619368	0.129448	1.022890	-0.028281
1	-0.778252	1.881956	-0.104719	0.256970	-0.300602
2	0.289745	1.938181	0.486858	0.973213	0.356591
3	-1.472702	0.844418	1.749624	-0.324075	-0.323681
4	-0.756507	1.526594	0.028754	1.038241	-0.137886
...	...	...	...	...	...
669	-0.086794	-0.581174	-1.076894	-1.614715	-0.399003
670	0.612425	-0.037156	-0.509303	0.124862	-0.771493
671	5.101733	0.243041	1.915262	0.376614	0.557275
672	0.161477	-0.493026	-1.011550	-1.637502	-0.316777
673	0.250245	-0.460558	-0.983857	-1.644915	-0.283871

674 rows × 5 columns

```
pd.read_csv('val_reduced.csv')
```

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5
0	0.253270	1.251024	-1.194546	0.870466	-0.467680
1	-1.022478	-0.372786	2.822655	0.249455	-0.566216
2	-0.171635	-0.336338	-0.810420	0.180750	-1.107895
3	-0.494650	-0.730221	-1.203544	-1.580556	-0.549715
4	-1.789779	2.158345	1.124680	0.426018	0.033925
...	...	...	...	...	...
56	0.841266	-1.346221	0.445062	0.708465	-1.136254
57	1.172784	1.591992	-0.886508	0.797272	-0.109679
58	0.138379	1.215139	-1.202435	0.884802	-0.487635
59	0.461461	-1.485395	0.325399	0.739982	-1.277996
60	-0.434582	0.999804	-1.407468	0.928190	-0.721310

61 rows × 5 columns

```
pd.read_csv('test_reduced.csv')
```

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5
0	0.404975	-2.241572	0.049869	1.432202	0.957229
1	0.942041	1.552375	-0.754516	0.851144	-0.030052
2	-1.038754	-2.110103	1.204388	1.647512	1.180087
3	0.279336	1.265981	-1.161711	0.872480	-0.438016
4	-0.349385	-1.041930	1.226147	-1.674382	-0.323319
...	...	...	...	...	...
57	-0.999846	2.493403	1.581268	0.395703	0.496905
58	-0.220245	-0.354529	-0.827460	0.184491	-1.127432
59	-0.982424	0.017756	0.501118	0.336345	-0.683156
60	-1.570365	0.861253	-0.589090	1.014124	1.678656
61	0.654550	1.400592	-1.056619	0.839118	-0.308614

62 rows × 5 columns

End of Data Preprocessing For Machine Learning

End of Assignment

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.