Topics:

1. Handling missing values — many more → basic
   - categorical; binary; Numeric
2. Categorical to Numeric conversion } → Loan approval
3. Find most "useful" variables in Loan approval } → ...
4. Pre-processing Datasets → Normalization & Standardization }

last class

Ops:

→ [

① 1 vs 2 vs 3 class break ✓

11.4% 8.9% 79.7%

revise solve problems & case-study least freq ans-questions

② Problem-solving sessions]

↔ TBD

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:07

EDA_FE.ipynb - Colaboratory | markov

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=f5n7flegGQEe

Update

+ Code + Text Connect

↳ 11 cells hidden

Basic Data Exploration

[ ] ↳ 9 cells hidden

Basic Data visualization: Univariate

[ ] ↳ 11 cells hidden

Simple Feature Engineering

abi lily_to_pay_EM!

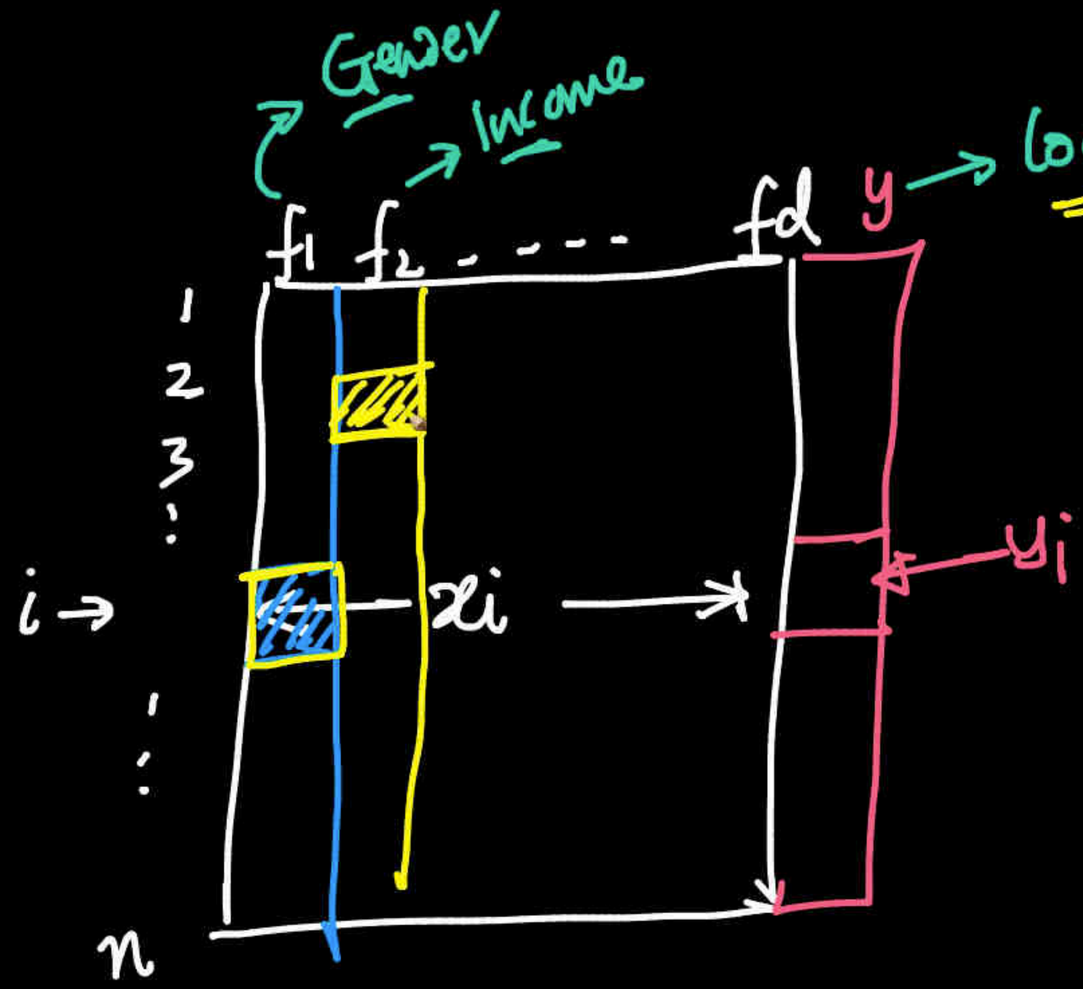
[ ] ↳ 37 cells hidden

Missing Values & Data Cleaning

[ ] ↳ 9 cells hidden

4 / 4

Missing Data



categorical

Q

fi: Gender → M or f (binary)

0 1

discrete

missing-value-imputation

↳ mode: most freq. occuring value

↳ random

↳ "other" ✓

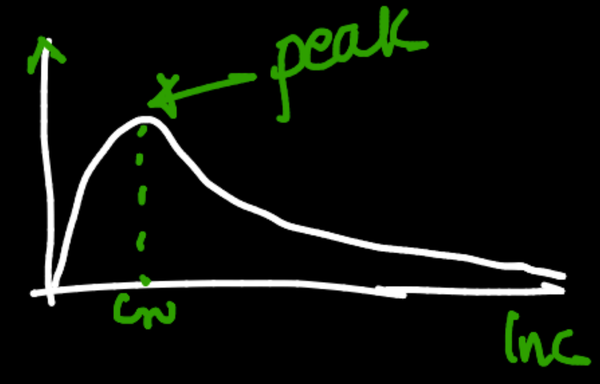
↳ there is some info in the fact that the data is missing

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:20

Q

Income: Numeric

mean or median



614 2015

[options] → median income using Gender, age, Pincode

→ mode

Model-based imputation (later)

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:20

EDA_FE.ipynb - Colaboratory | markov

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=SkB8xczxbRc2

+ Code + Text Connect

Missing Values & Data Cleaning

```
data.isna().sum()
```

```
Gender                13
Married                3
Dependents            15
Education              0
Self_Employed         32
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount            22
Loan_Amount_Term      14
Credit_History        50
Property_Area          0
Loan_Status            0
TotalIncome            0
Loan_Amount_per_year  36
EMI                   36
Able_to_pay_EMI        0
dtype: int64
```

```
[ ] # Function to create a data frame with number and percentage of missing data in a data frame

    def missing_to_df(df):
```

8 / 8

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:20

EDA_FE.ipynb - Colaboratory | markov

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=SkB8xczxbRc2

+ Code + Text Connect

Missing Values & Data Cleaning

```
data.isna().sum()
```

```
Gender                13
Married                3
Dependents            15
Education              0
Self_Employed         32
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount            22
Loan_Amount_Term      14
Credit_History        50
Property_Area          0
Loan_Status            0
TotalIncome            0
Loan_Amount_per_year  36
EMI                   36
Able_to_pay_EMI        0
dtype: int64
```

```
[ ] # Function to create a data frame with number and percentage of missing data in a data frame

    def missing_to_df(df):
```

9 / 9

```
Loan_Amount_per_year    36
EMI                     36
Able_to_pay_EMI          0
dtype: int64
```

[32]
```
# Function to create a data frame with number and percentage of missing data in a data frame

def missing_to_df(df):
    #Number and percentage of missing data in training data set for each column
    total_missing_df = df.isnull().sum().sort_values(ascending =False)
    percent_missing_df = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending=False)
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, keys=['Total', 'Percent'])
    return missing_data_df
```

[33]
```
missing_df = missing_to_df(data)
missing_df[missing_df['Total'] > 0]
```

	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192
Self_Employed	32	5.211726

```
[33] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192
Self_Employed	32	5.211726
LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599

```
# Credit History=2 for nan/missing values.
data['Credit_History'] = data['Credit_History'].fillna(2)
```

```
EMI                  36
Able_to_pay_EMI       0
dtype: int64
```

```
[32] # Function to create a data frame with number and percentage of missing data in a data frame

     def missing_to_df(df):
         #Number and percentage of missing data in training data set for each column
         total_missing_df = df.isnull().sum().sort_values(ascending =False)
         percent_missing_df = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending=False)
         missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, keys=['Total', 'Percent'])
         return missing_data_df
```

```
[33] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192
Self_Employed	32	5.211726

```
TotalIncome                0
Loan_Amount_per_year      36
EMI                       36
Able_to_pay_EMI            0
dtype: int64
```

```
[32] # Function to create a data frame with number and percentage of missing data in a data frame

def missing_to_df(df):
    #Number and percentage of missing data in training data set for each column
    total_missing_df = df.isnull().sum().sort_values(ascending =False)
    percent_missing_df = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending=False)
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, keys=['Total', 'Percent'])
    return missing_data_df
```

```
[33] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192

Credit History vs Loan Approval

is Cibil Score

```
[ ] data['Credit_History'].value_counts()
```

```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

```
[ ] sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
    #Observation:
    ## We can clearly see that the approval rate is 80% if your credit history is aligned with the guidlines.
    ## Hence this is the most important question that can be considered.
```

```
[ ]
```

Missing Values & Data Cleaning

Credit History vs Loan Approval

```
[ ]  data['Credit_History'].value_counts()
```

```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

```
[ ]  sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
     #Observation:
     ## We can clearly see that the approval rate is 80% if your credit history is aligned with the guidlines.
     ## Hence this is the most important question that can be considered.
```

```
[ ]
```

Missing Values & Data Cleaning

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:28

EDA_FE.ipynb - Colaboratory | markov

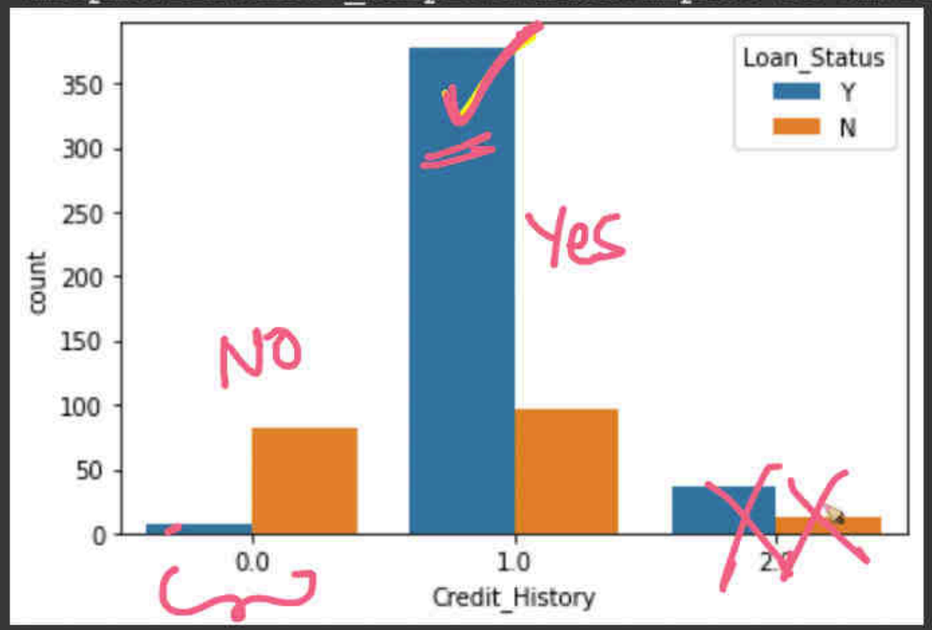
colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=mezoRK5waMTr

+ Code + Text RAM Disk

```
Name: Credit_History, dtype: int64
```

```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
#Observation:
## We can clearly see that the approval rate is 80% if your credit history is aligned with the guidlines.
## Hence this is the most important question that can be considered.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac71b2cc50>
```

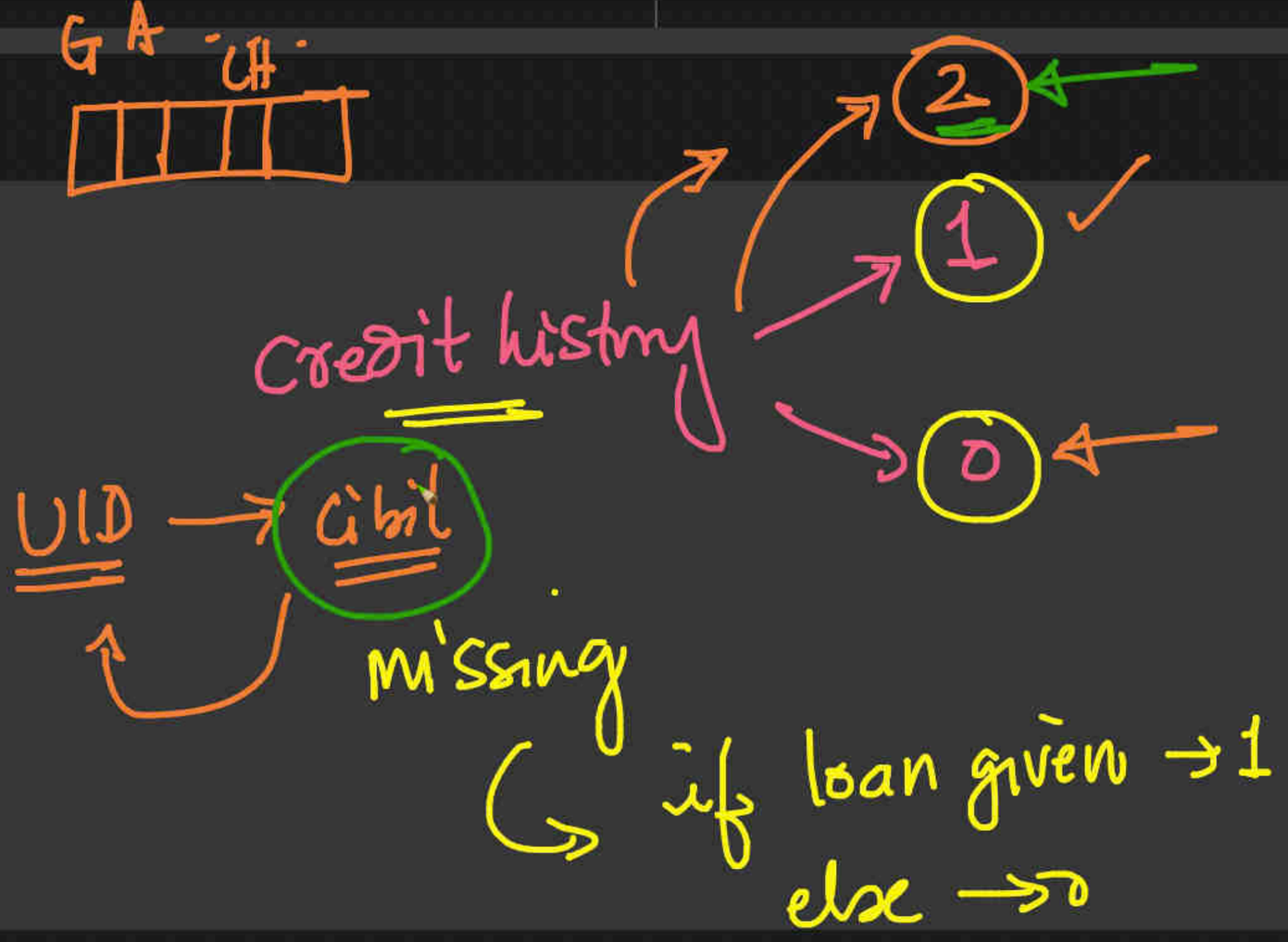



```
[ ]
```

16 / 16

```
[33] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192
Self_Employed	32	5.211726
LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599




```
[34] # Credit History=2 for nan/missing values.
     data['Credit_History'] = data['Credit_History'].fillna(2)
```

EMI	36	5.863192
Self_Employed	32	5.211726
LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599

[34] 0s
```
# Credit History=2 for nan/missing values.
data['Credit_History'] = data['Credit_History'].fillna(2)
```

[ ]
```
# Self_Employed = 'Other' for nan
data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

[ ]
```
data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

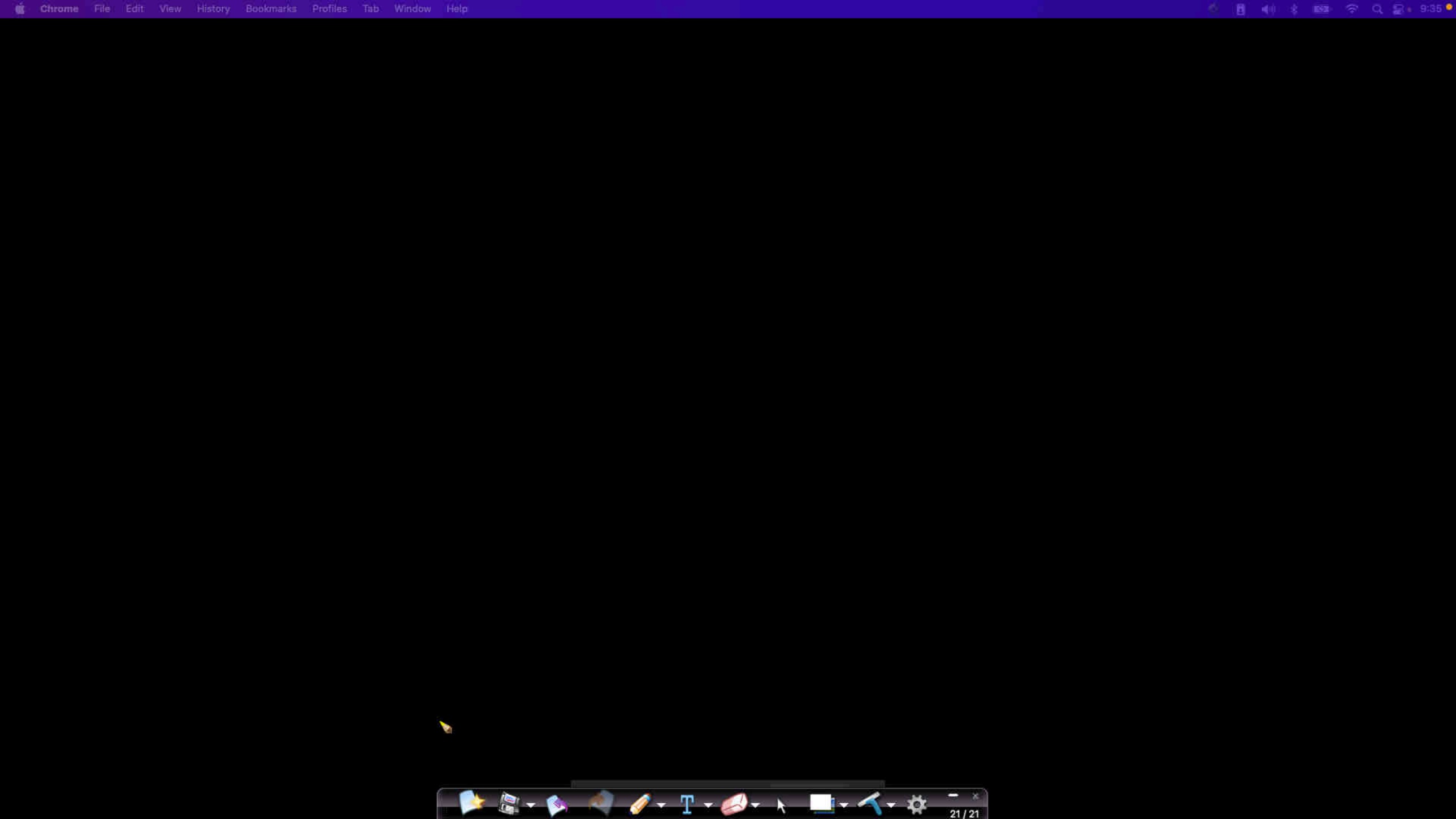
Q

imputation fi → Model-based imputation

$P(\widehat{fi} \mid f_1, f_2, \ldots f_{i-1}\ f_{i+1} \ldots, f_d)$

Simple Bayes (cumbersome)

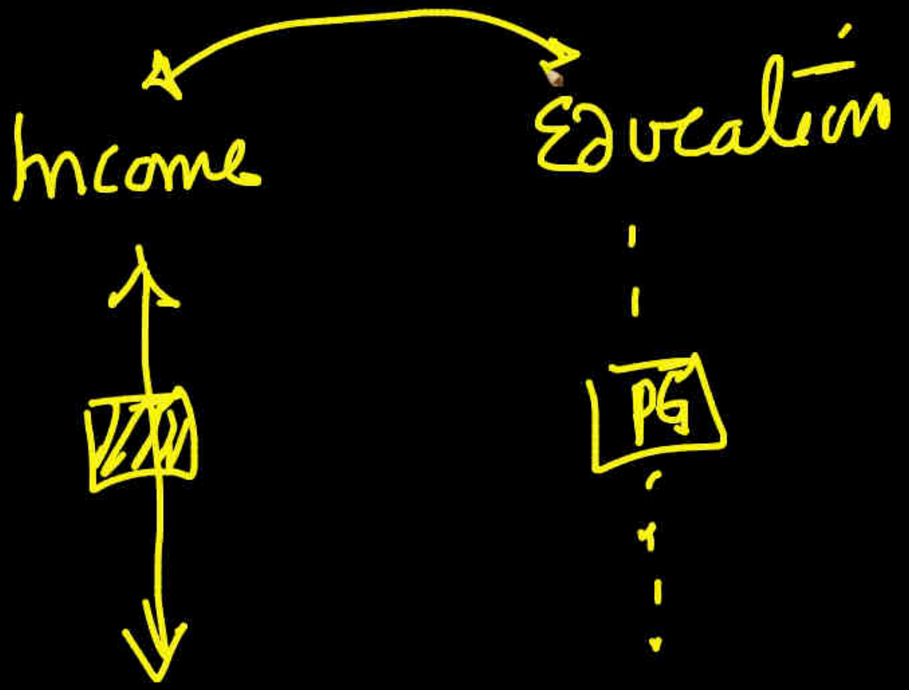
ML-models


Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:35
21 / 21

Q

$n = \#\text{rows} = \overline{1 \text{ to } M}$

so rows → missing → OK to skip



model-based
imp

	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192
Self_Employed	32	5.211726
LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599

```
[34] # Credit History=2 for nan/missing values.
     data['Credit_History'] = data['Credit_History'].fillna(2)
```

```
[ ] # Self_Employed = 'Other' for nan
    data.Self_Employed.unique()
```

Gender	13	2.117264
Married	3	0.488599

```
[34] # Credit History=2 for nan/missing values.
     data['Credit_History'] = data['Credit_History'].fillna(2)
```

```
[ ] # Self_Employed = 'Other' for nan
    data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

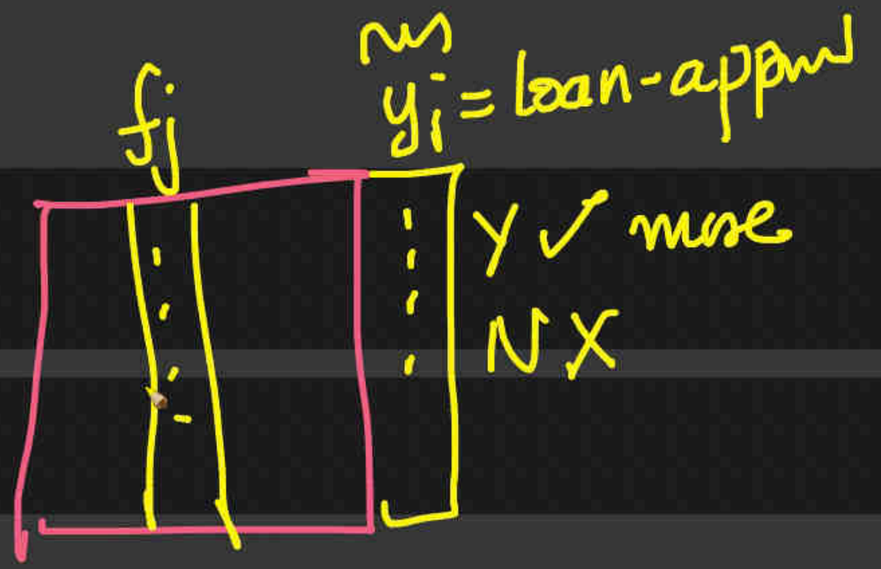
```
[ ] data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

```
[ ] # median imputation for numerical columns.
    from sklearn.impute import SimpleImputer

    num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

    median_imputer = SimpleImputer(strategy = 'median')
    for col in num_missing:
```

Gender	13	2.117264
Married	3	0.488599




```
[34] # Credit History=2 for nan/missing values.
     data['Credit_History'] = data['Credit_History'].fillna(2)
```

```
[ ] # Self_Employed = 'Other' for nan
    data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

```
[ ] data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

```
[ ] # median imputation for numerical columns.
    from sklearn.impute import SimpleImputer

    num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

    median_imputer = SimpleImputer(strategy = 'median')
    for col in num_missing:
```

Gender	13	2.117264
Married	3	0.488599

[34] 0s
```
# Credit History=2 for nan/missing values.
data['Credit_History'] = data['Credit_History'].fillna(2)
```

[ ]
```
# Self_Employed = 'Other' for nan
data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

[ ]
```
data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

[ ]
```
# median imputation for numerical columns.
from sklearn.impute import SimpleImputer

num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

median_imputer = SimpleImputer(strategy = 'median')
for col in num_missing:
```

```
data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

```
[ ] data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

ML-lib

scikit-learn

```
[ ] # median imputation for numerical columns
    from sklearn.impute import SimpleImputer

    num_missing = ['EMI', 'Loan_Amount_per_year', 'LoanAmount', 'Loan_Amount_Term']

    median_imputer = SimpleImputer(strategy = 'median')
    for col in num_missing:
        data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[ ] # Highest Freq imputation for some categorical columns.
    cat_missing = ['Gender', 'Married','Dependents']

    freq_imputer = SimpleImputer(strategy = 'most_frequent')
    for col in cat_missing:
        data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

```
[ ] data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

```
[ ] # median imputation for numerical columns.
    from sklearn.impute import SimpleImputer

    num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

    median_imputer = SimpleImputer(strategy = 'median')
    for col in num_missing:
        data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[ ] # Highest Freq imputation for some categorical columns.
    cat_missing = ['Gender', 'Married','Dependents']

    freq_imputer = SimpleImputer(strategy = 'most_frequent')
    for col in cat_missing:
        data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

sklearn.impute.SimpleImputer

class `sklearn.impute.`**`SimpleImputer`**(*, *missing_values=nan*, *strategy='mean'*, *fill_value=None*, *verbose='deprecated'*, *copy=True*, *add_indicator=False*) [source]

Imputation transformer for completing missing values.

Read more in the User Guide.

New in version 0.20: `SimpleImputer` replaces the previous `sklearn.preprocessing.Imputer` estimator which is now removed.

Parameters:

missing_values : *int, float, str, np.nan, None or pandas.NA, default=np.nan*
The placeholder for the missing values. All occurrences of `missing_values` will be imputed. For pandas' dataframes with nullable integer dtypes with missing values, `missing_values` can be set to either `np.nan` or `pd.NA`.

strategy : *str, default='mean'*
The imputation strategy.

- If "mean", then replace missing values using the mean along each column. Can only

scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
sklearn.impute.SimpleImputer
Examples using
sklearn.impute.SimpleImputer

Parameters:

missing_values : *int, float, str, np.nan, None or pandas.NA, default=np.nan*

The placeholder for the missing values. All occurrences of `missing_values` will be imputed. For pandas' dataframes with nullable integer dtypes with missing values, `missing_values` can be set to either `np.nan` or `pd.NA`.

strategy : *str, default='mean'*

The imputation strategy.

- If "mean", then replace missing values using the mean along each column. Can only be used with numeric data.
- If "median", then replace missing values using the median along each column. Can only be used with numeric data.
- If "most_frequent", then replace missing using the most frequent value along each column. Can be used with strings or numeric data. If there is more than one such value, only the smallest is returned.
- If "constant", then replace missing values with fill_value. Can be used with strings or numeric data.

New in version 0.20: strategy="constant" for fixed value imputation.

fill_value : *str or numerical value, default=None*

When strategy == "constant", fill_value is used to replace all occurrences of [illegible] fill_value will be 0 when imputing numerical data [illegible]ta types

```
data.Self_Employed.unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

```
data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

```
# median imputation for numerical columns.
from sklearn.impute import SimpleImputer

num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

median_imputer = SimpleImputer(strategy = 'median')
for col in num_missing:
    data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
# Highest Freq imputation for some categorical columns.
cat_missing = ['Gender', 'Married','Dependents']

freq_imputer = SimpleImputer(strategy = 'most_frequent')
for col in cat_missing:
    data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
array(['No', 'Yes', nan], dtype=object)
```

```
[ ] data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

```
# median imputation for numerical columns.
from sklearn.impute import SimpleImputer

num_missing = ['EMI', 'Loan_Amount_per_year', 'LoanAmount', 'Loan_Amount_Term']

median_imputer = SimpleImputer(strategy = 'median')
for col in num_missing:
    data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[ ] # Highest Freq imputation for some categorical columns.
    cat_missing = ['Gender', 'Married','Dependents']

    freq_imputer = SimpleImputer(strategy = 'most_frequent')
    for col in cat_missing:
        data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[ ] missing_df = missing_to_df(data)
    missing_df[missing_df['Total'] > 0]
```

```
num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

median_imputer = SimpleImputer(strategy = 'median')
for col in num_missing:
    data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
# Highest Freq imputation for some categorical columns.
cat_missing = ['Gender', 'Married','Dependents']

freq_imputer = SimpleImputer(strategy = 'most_frequent')
for col in cat_missing:
    data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
missing_df = missing_to_df(data)
missing_df[missing_df['Total'] > 0]
```

Categorical to Numerical encoding

Nominal vs Ordinal variables

1. One Hot Encoding

[39] 0s

```
# Highest Freq imputation for some categorical columns.
cat_missing = ['Gender', 'Married','Dependents']

freq_imputer = SimpleImputer(strategy = 'most_frequent')
for col in cat_missing:
    data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

[40] 0s

```
missing_df = missing_to_df(data)
missing_df[missing_df['Total'] > 0]
```

	Total	Percent

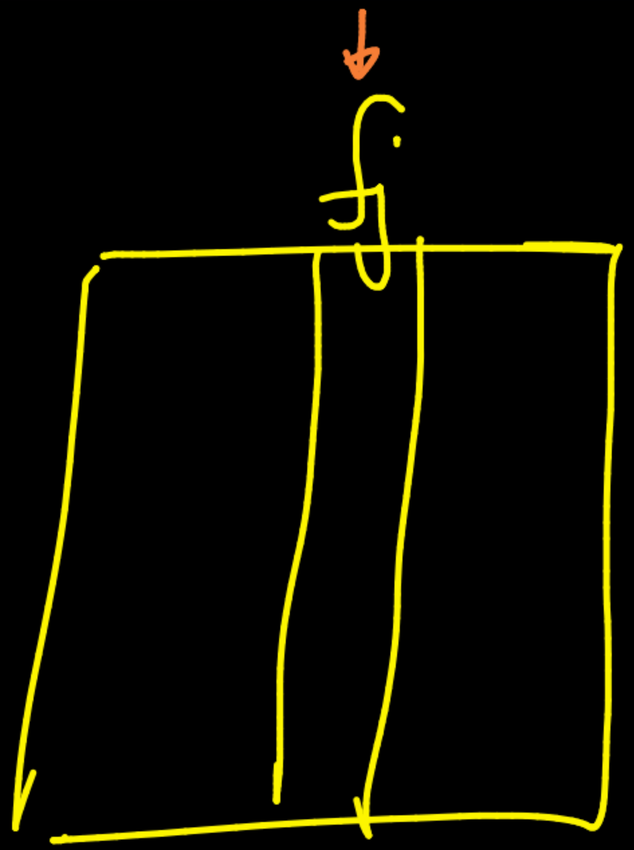
Categorical to Numerical encoding

Nominal vs Ordinal variables

1. One Hot Encoding
2. Label encoding
3. Target Encoding

Appropriate encoding depends on what our task is (and) what we do next?

Task: Compute Correlation (PCC and SPCC) between each feature and the Loan Status

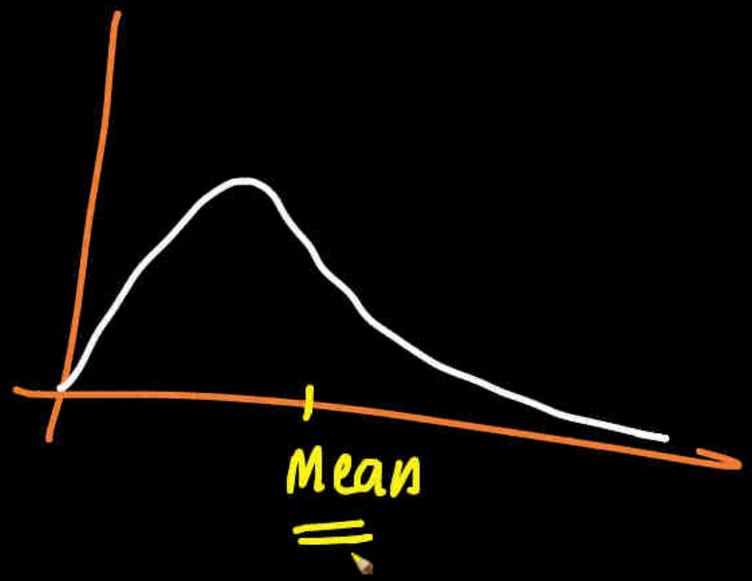


imputation

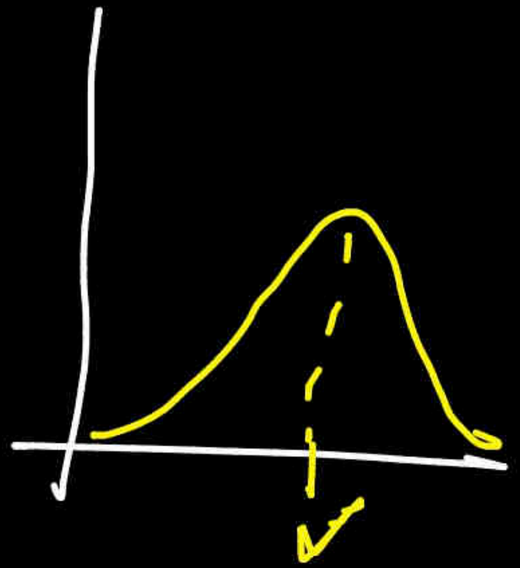
drop column

>50% of values are missing

Q



(log)
boxcox



```
[38] # median imputation for numerical columns.
     from sklearn.impute import SimpleImputer

     num_missing = ['EMI', 'Loan_Amount_per_year', 'LoanAmount', 'Loan_Amount_Term']

     median_imputer = SimpleImputer(strategy = 'median')
     for col in num_missing:
         data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[39] # Highest Freq imputation for some categorical columns.
     cat_missing = ['Gender', 'Married','Dependents']

     freq_imputer = SimpleImputer(strategy = 'most_frequent')
     for col in cat_missing:
         data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[40] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent

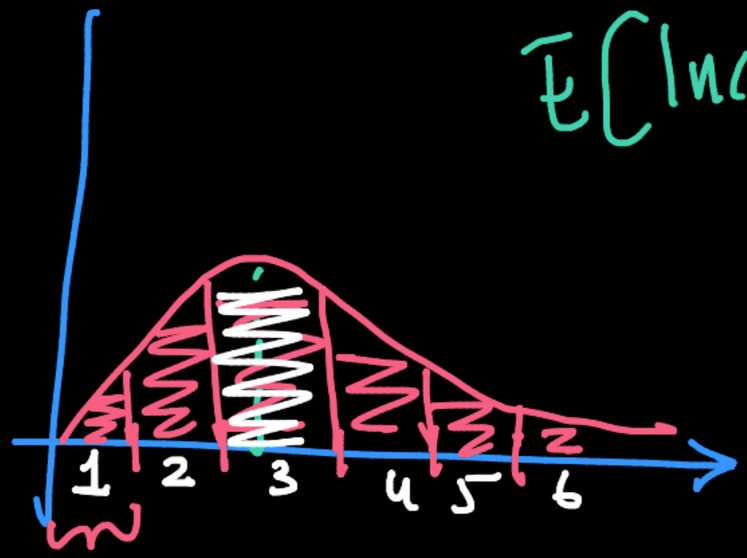
Categorical to Numerical encoding

1MM rows

100 rows ← Missing → drop

Q

$E[\text{Income}] = \text{Mean}$



pdf

Categorical
↳ Numeric

{ Income (Numeric) → L, M, H1, H2, H3, H4, VH (Categorical) binning

plots

[χ^2 Test: Income-bin vs loan-approval

monotonic Linear

SRCC & PCC → numeric

Gender

non-ordinal feature

M → H 1 2 1

F → -H 2 1 3

O → O 3 3 2

numeric

Loan status

0 → NO

1 → Yes

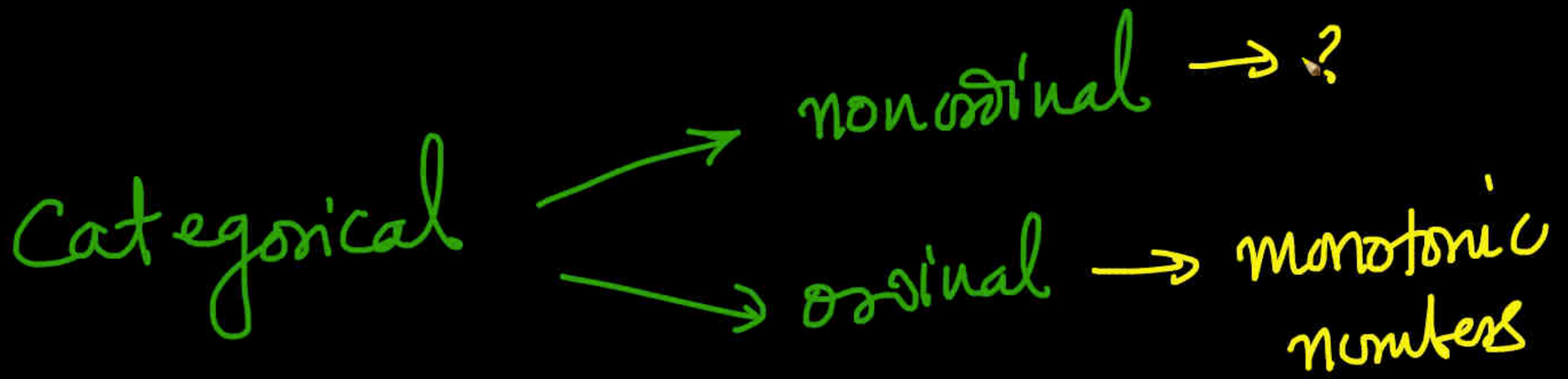
numeric

Education

ordinal features

natural ordering

HS	→ 0	4
UG	→ 1	5
PG	→ 2	10
Phd	→ 3	20



Gender

Data → 613

M → 0.8

F → 0.5

O → 0.2

response coding ↓

$P(\text{loanapp}=1 \mid G=M)$

$P(\text{loanApp}=1 \mid G=F)$

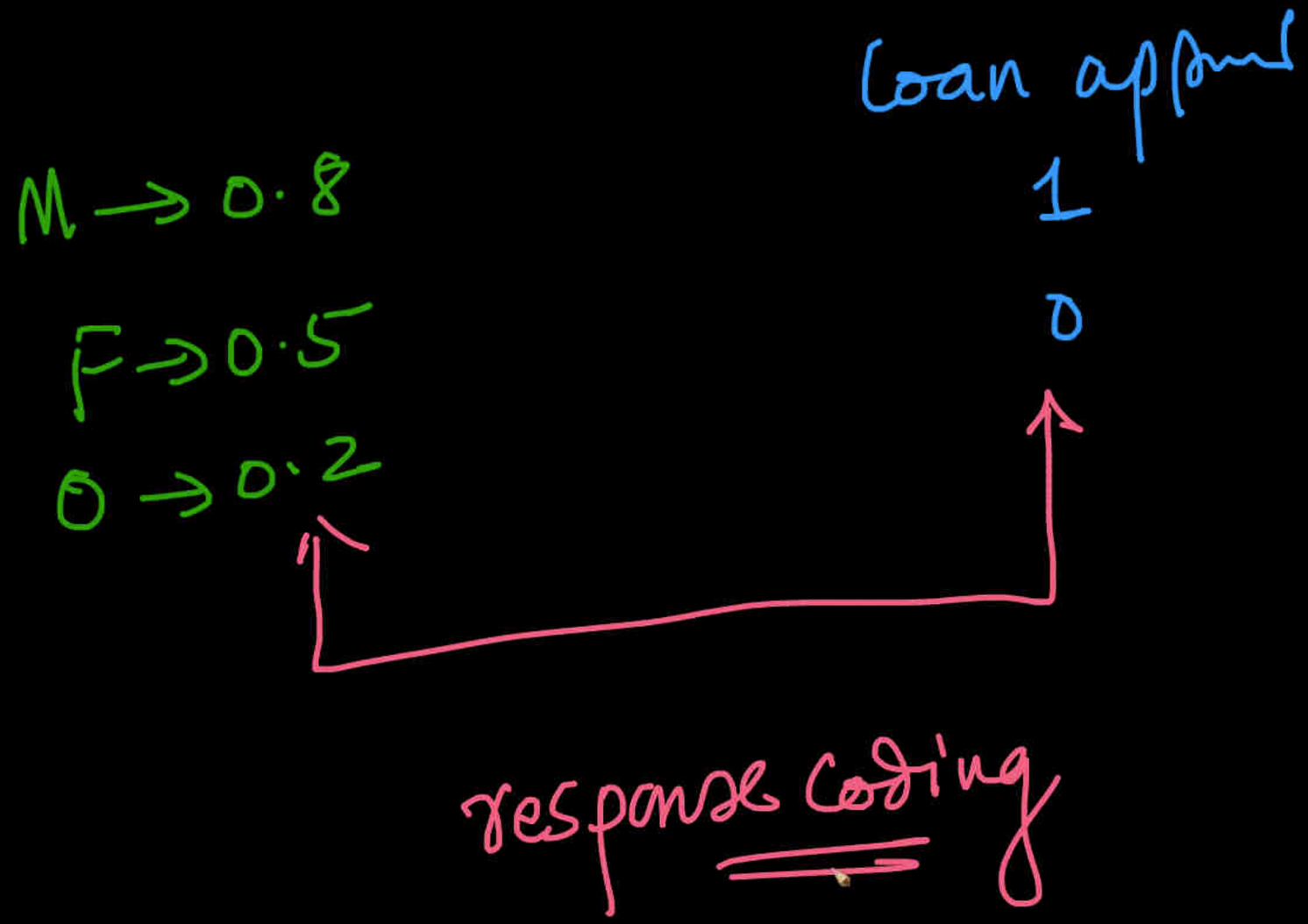
$P(IA=1 \mid G=O)$

613 obsen

Obj / Task: ?

CC between Gender & Loan Approved

new-cust M



If G has no. corr to loan approval

M → 0.6

F → 0.61

O → 0.59

```
    data['Credit_History'] = data['Credit_History'].fillna(2)
```

```
[36] # Self_Employed = 'Other' for nan
     data.Self_Employed.unique()
```

```
     array(['No', 'Yes', nan], dtype=object)
```

```
[37] data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

```
[38] # median imputation for numerical columns.
     from sklearn.impute import SimpleImputer

     num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

     median_imputer = SimpleImputer(strategy = 'median')
     for col in num_missing:
         data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[39] # Highest Freq imputation for some categorical columns.
     cat_missing = ['Gender', 'Married','Dependents']

     freq_imputer = SimpleImputer(strategy =
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:12

EDA_FE.ipynb - Colaboratory | markov | sklearn.impute.SimpleImputer

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=sp3SJkJmKaim

+ Code + Text RAM Disk

```
[38] # median imputation for numerical columns.
     from sklearn.impute import SimpleImputer

     num_missing = ['EMI', 'Loan_Amount_per_year', 'LoanAmount', 'Loan_Amount_Term']

     median_imputer = SimpleImputer(strategy = 'median')
     for col in num_missing:
         data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

→ list

```
[39] # Highest Freq imputation for some categorical columns.
     cat_missing = ['Gender', 'Married','Dependents']

     freq_imputer = SimpleImputer(strategy = 'most_frequent')
     for col in cat_missing:
         data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[40] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent

48 / 48

```
[38] # median imputation for numerical columns.
     from sklearn.impute import SimpleImputer

     num_missing = ['EMI', 'Loan_Amount_per_year', 'LoanAmount', 'Loan_Amount_Term']

     median_imputer = SimpleImputer(strategy = 'median')
     for col in num_missing:
         data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[39] # Highest Freq imputation for some categorical columns.
     cat_missing = ['Gender', 'Married','Dependents']

     freq_imputer = SimpleImputer(strategy = 'most_frequent')
     for col in cat_missing:
         data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[40] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:13

EDA_FE.ipynb - Colaboratory | markov | sklearn.impute.SimpleImputer

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=sp3SJkJmKaIm

+ Code + Text RAM Disk

```
[38] # median imputation for numerical columns.
     from sklearn.impute import SimpleImputer

     num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

     median_imputer = SimpleImputer(strategy = 'median')
     for col in num_missing:
         data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[39] # Highest Freq imputation for some categorical columns.
     cat_missing = ['Gender', 'Married','Dependents']

     freq_imputer = SimpleImputer(strategy = 'most_frequent')
     for col in cat_missing:
         data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[40] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent

50 / 50

```
[38] # median imputation for numerical columns.
     from sklearn.impute import SimpleImputer

     num_missing = ['EMI', 'Loan_Amount_per_year',  'LoanAmount',  'Loan_Amount_Term']

     median_imputer = SimpleImputer(strategy = 'median')
     for col in num_missing:
         data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[39] # Highest Freq imputation for some categorical columns.
     cat_missing = ['Gender', 'Married','Dependents']

     freq_imputer = SimpleImputer(strategy = 'most_frequent')
     for col in cat_missing:
         data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[40] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent

```
[38] # median imputation for numerical columns.
     from sklearn.impute import SimpleImputer

     num_missing = ['EMI', 'Loan_Amount_per_year', 'LoanAmount', 'Loan_Amount_Term']

     median_imputer = SimpleImputer(strategy = 'median')
     for col in num_missing:
         data[col] = pd.DataFrame(median_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[39] # Highest Freq imputation for some categorical columns.
     cat_missing = ['Gender', 'Married','Dependents']

     freq_imputer = SimpleImputer(strategy = 'most_frequent')
     for col in cat_missing:
         data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
```

```
[40] missing_df = missing_to_df(data)
     missing_df[missing_df['Total'] > 0]
```

	Total	Percent

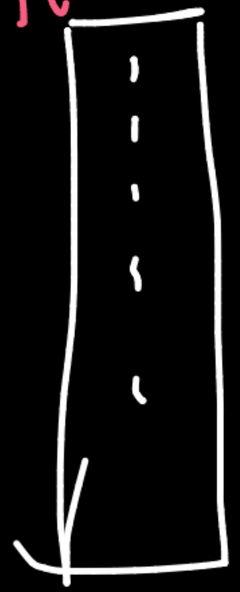
Q

Gender → M/F/O

loan app = 1 or 0

Response coding

$P(y_i = 1 | \ldots)$



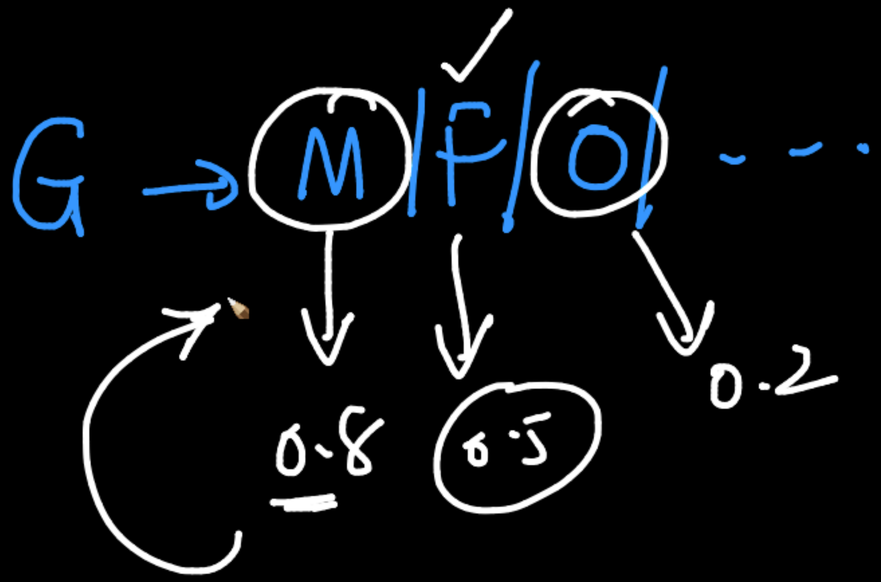
isM isF isO ← 1 hot encoding

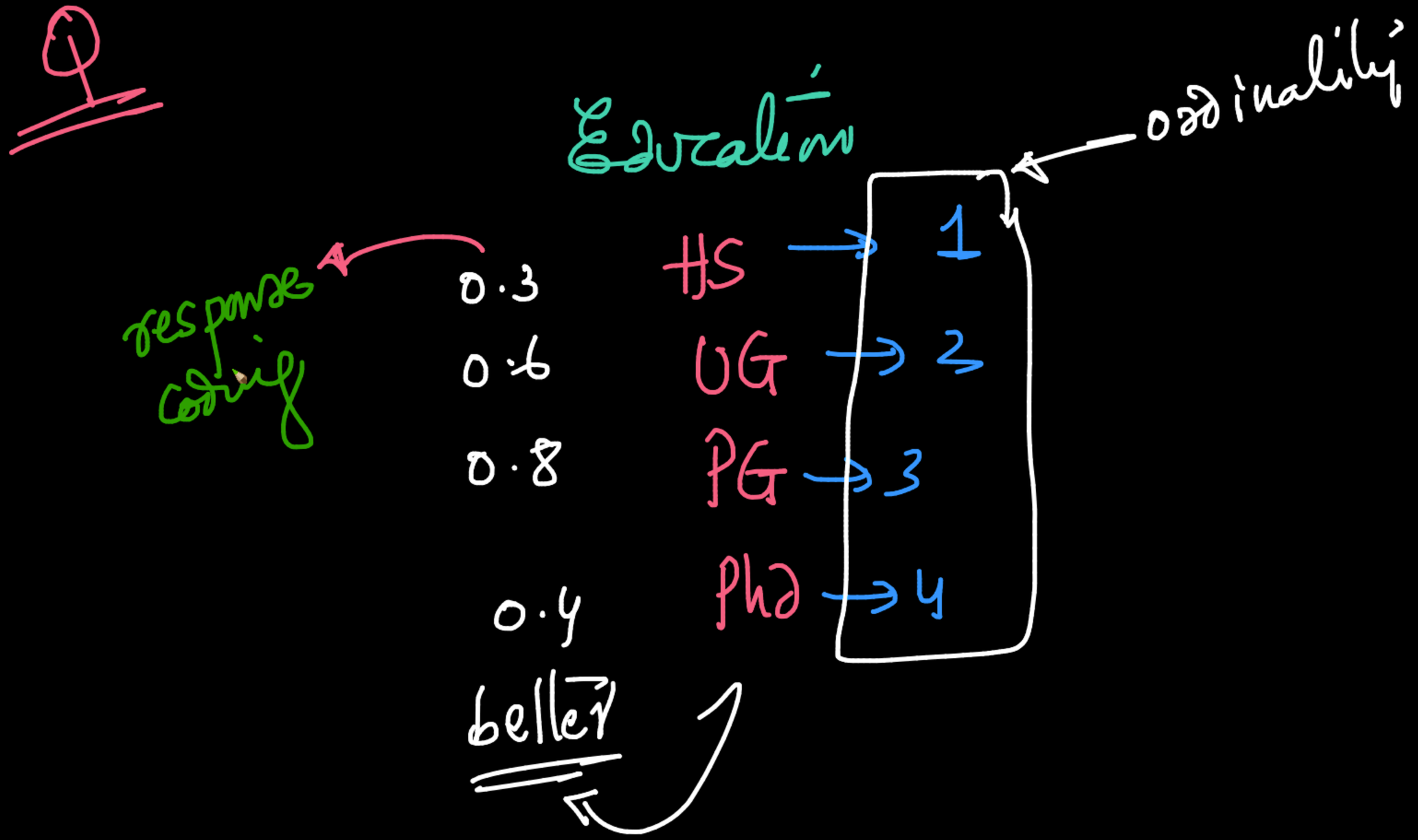
1	0	0
0	1	0
0	0	1

Task: CC before Gender & loan App

↑ Linear Models (ML) (Clarity)

is M Correlate with Loan Appr




Education
ordinality
response coding
0.3
0.6
0.8
0.4
HS
UG
PG
Phd
1
2
3
4
below

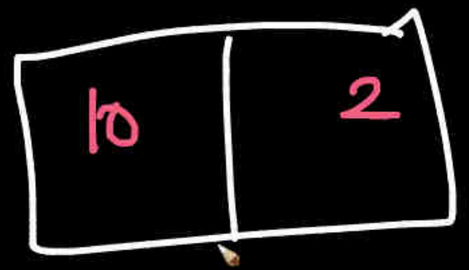
Education

UG

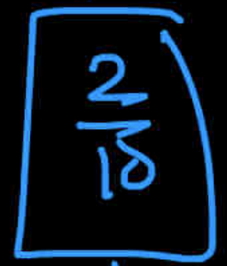
HS

PhD

PG



Data sparsity



not reliable

Categorical to Numerical encoding

Nominal vs Ordinal variables

1. One Hot Encoding
2. Label encoding
3. Target Encoding

Appropriate encoding depends on what our task is (and) what we do next?

Task: Compute Correlation (PCC and SRCC) between each feature and the Loan-Status

```
[ ]  s = (data.dtypes == 'object')
     object_cols = list(s[s].index)
     object_cols
```

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

Categorical to Numerical encoding

Nominal vs Ordinal variables

1. One Hot Encoding
2. Label encoding
3. Target Encoding

loan approved

0 , 1

2

Appropriate encoding depends on what our task is (and) what we do next?

Task: Compute Correlation (PCC and SRCC) between each feature and the Loan-Status

```
[ ]  s = (data.dtypes == 'object')
     object_cols = list(s[s].index)
     object_cols
```

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

Categorical to Numerical encoding

Nominal vs Ordinal variables

1. One Hot Encoding
2. Label encoding
3. Target Encoding

Appropriate encoding depends on what our task is (and) what we do next?

Task: Compute Correlation (PCC and SRCC) between each feature and the Loan-Status

```
[ ]  s = (data.dtypes == 'object')
     object_cols = list(s[s].index)
     object_cols
```

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

Loan Status

```
# Loan_Status
col='Loan_Status'
data[col].value_counts()
```

```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
col='Loan_Status'
data[col] = label_encoder.fit_transform(data[col])
```

```
data[col].value_counts()
```

```
1    422
0    192
Name: Loan_Status, dtype: int64
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:31

EDA_FE.ipynb - Colaboratory | markov | sklearn.impute.SimpleImputer

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=U-lTHyZPiAek

+ Code + Text

RAM Disk

-1 +1

SRCC

Gender

```
[ ] #Gender
    data['Gender'].value_counts()
```

1 , 0

```
    Male      502
    Female    112
    Name: Gender, dtype: int64
```

```
[ ] from sklearn.preprocessing import LabelEncoder

    label_encoder = LabelEncoder()
    col='Gender'
    data[col] = label_encoder.fit_transform(data[col])
```

```
[ ] data[col].value_counts()
```

```
    1    502
    0    112
    Name: Gender, dtype: int64
```

61 / 61

Married

↳ 2 cells hidden

Property Area

```
[ ] # col='Property_Area'
    col='Property_Area'
    data[col].value_counts()
```

```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

```
[ ] !pip install category_encoders
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: category_encoders in /usr/local/lib/python3.7/dist-packages (2.4.1)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-packages (from category_encoder
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (0.5.
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.4.
```

```
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.2
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1->category
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.1->category_encoders)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->ca
```

```
[ ] from category_encoders import TargetEncoder

    te = TargetEncoder()
    data[col] = te.fit_transform(data[col], data['Loan_Status'])
```

```
[ ] col='Property_Area'
    data[col].value_counts()
```

```
0.768240    233
0.658416    202
0.614525    179
Name: Property_Area, dtype: int64
```

Education

```
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.1->category_encoders)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->ca
```

```
[ ] from category_encoders import TargetEncoder

    te = TargetEncoder()
    data[col] = te.fit_transform(data[col], data['Loan_Status'])
```

1 vs 0

```
[ ] col='Property_Area'
    data[col].value_counts()
```

```
0.768240    233
0.658416    202
0.614525    179
Name: Property_Area, dtype: int64
```

Education

```
[ ] col='Education'
    data[col].value_counts()
```

```
[ ] from category_encoders import TargetEncoder

    te = TargetEncoder()
    data[col] = te.fit_transform(data[col], data['Loan_Status'])
```

```
[ ] col='Property_Area'
    data[col].value_counts()
```

```
0.768240    233
0.658416    202
0.614525    179
Name: Property_Area, dtype: int64
```

Education

```
[ ] col='Education'
    data[col].value_counts()
```

```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

Self Employed

```
[ ] col='Self_Employed'
    data[col].value_counts()
```

```
No       500
Yes       82
Other     32
Name: Self_Employed, dtype: int64
```

```
[ ] from category_encoders import TargetEncoder

    te = TargetEncoder()
    data[col] = te.fit_transform(data[col], data['Loan_Status'])
    data[col].value_counts()
```

```
0.686000    500
0.682927     82
0.718750     32
Name: Self_Employed, dtype: int64
```

```
[ ] s = (data.dtypes == 'object')
    object_cols = list(s[s].index)
    object_cols
```

Self Employed

```
[ ] col='Self_Employed'
    data[col].value_counts()
```

```
No       500
Yes       82
Other     32
Name: Self_Employed, dtype: int64
```

```
[ ] from category_encoders import TargetEncoder

    te = TargetEncoder()
    data[col] = te.fit_transform(data[col], data['Loan_Status'])
    data[col].value_counts()
```

```
0.686000    500
0.682927     82
0.718750     32
Name: Self_Employed, dtype: int64
```

```
[ ] s = (data.dtypes == 'object')
    object_cols = list(s[s].index)
    object_cols
```

Self Employed

```
[ ] col='Self_Employed'
    data[col].value_counts()
```

```
No       500
Yes       82
Other     32
Name: Self_Employed, dtype: int64
```

```
[ ] from category_encoders import TargetEncoder

    te = TargetEncoder()
    data[col] = te.fit_transform(data[col], data['Loan_Status'])
    data[col].value_counts()
```

```
0.686000    500
0.682927     82
0.718750     32
Name: Self_Employed, dtype: int64
```

```
[ ] s = (data.dtypes == 'object')
    object_cols = list(s[s].index)
    object_cols
```

```
# No more non numeric cols.
```

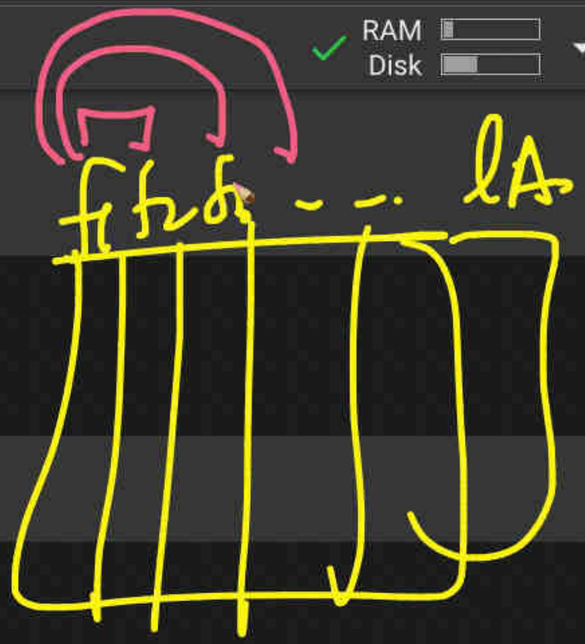
```
[]
```

```
[ ] data.dtypes
```

```
Gender                  int64
Married                 int64
Dependents            float64
Education               int64
Self_Employed         float64
ApplicantIncome         int64
CoapplicantIncome     float64
LoanAmount            float64
Loan_Amount_Term      float64
Credit_History        float64
Property_Area         float64
Loan_Status             int64
TotalIncome           float64
Loan_Amount_per_year  float64
EMI                   float64
Able_to_pay_EMI         int64
dtype: object
```

Correlation Coefficients

Correlation Coefficients



```
[ ]  #PCC
     plt.figure(figsize=(15, 15))
     sns.heatmap(data.corr(method='pearson'), square=True,annot=True)
```

```
[ ]  #SRCC
     plt.figure(figsize=(15, 15))
     sns.heatmap(data.corr(method='spearman'), square=True,annot=True)
```

```
[ ]  plt.scatter(data['Credit_History'], data['Loan_Status'])
     plt.show()
     #sometimes scatter plots can be misleading do to catgeorical nature of the data
```

```
[ ]  sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

Correlation Coefficients

```
[ ]  #PCC
     plt.figure(figsize=(15, 15))
     sns.heatmap(data.corr(method='pearson'), square=True,annot=True)
```

```
[ ]  #SRCC
     plt.figure(figsize=(15, 15))
     sns.heatmap(data.corr(method='spearman'), square=True,annot=True)
```

```
[ ]  plt.scatter(data['Credit_History'], data['Loan_Status'])
     plt.show()
     #sometimes scatter plots can be misleading do to catgeorical nature of the data
```

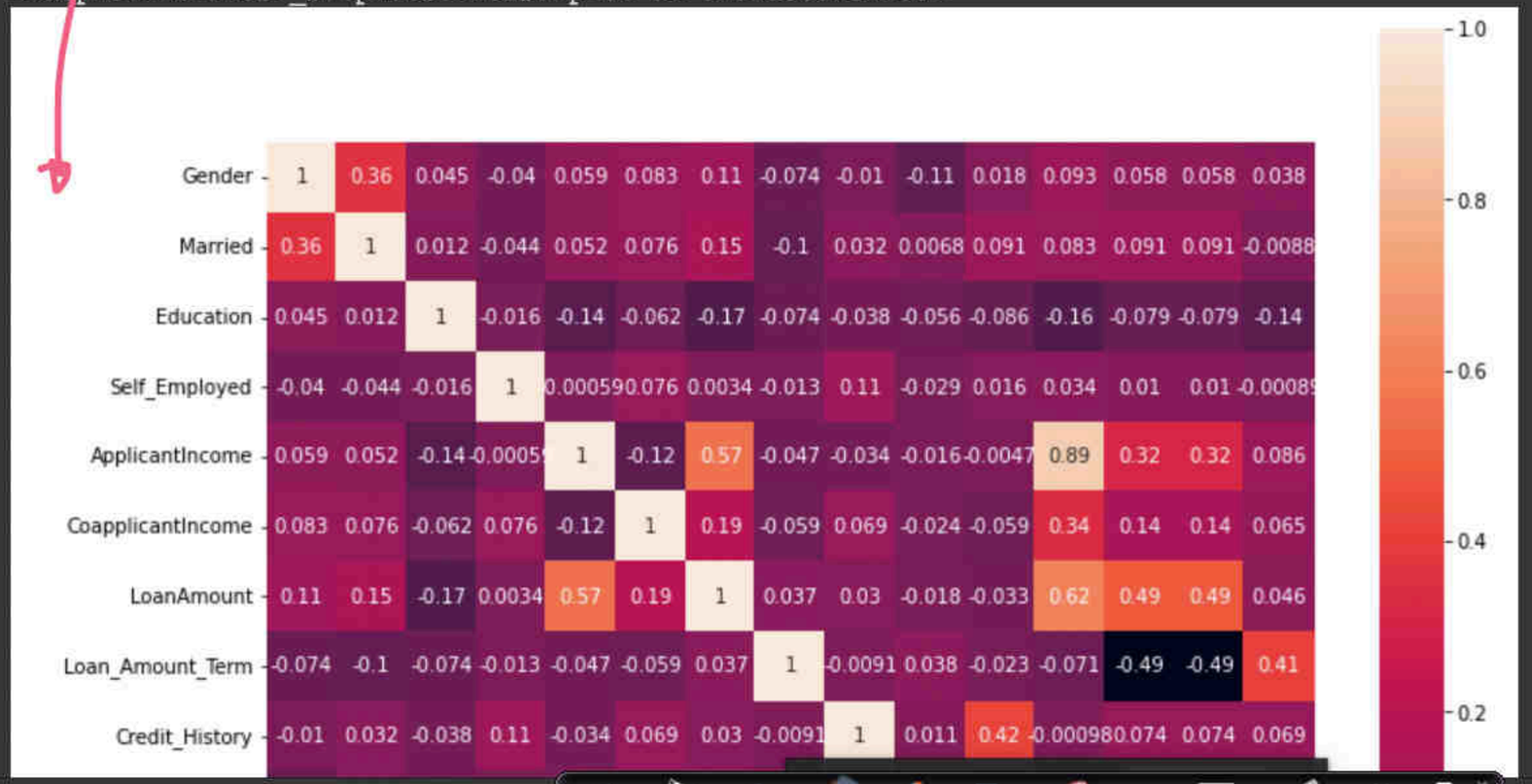
```
[ ]  sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

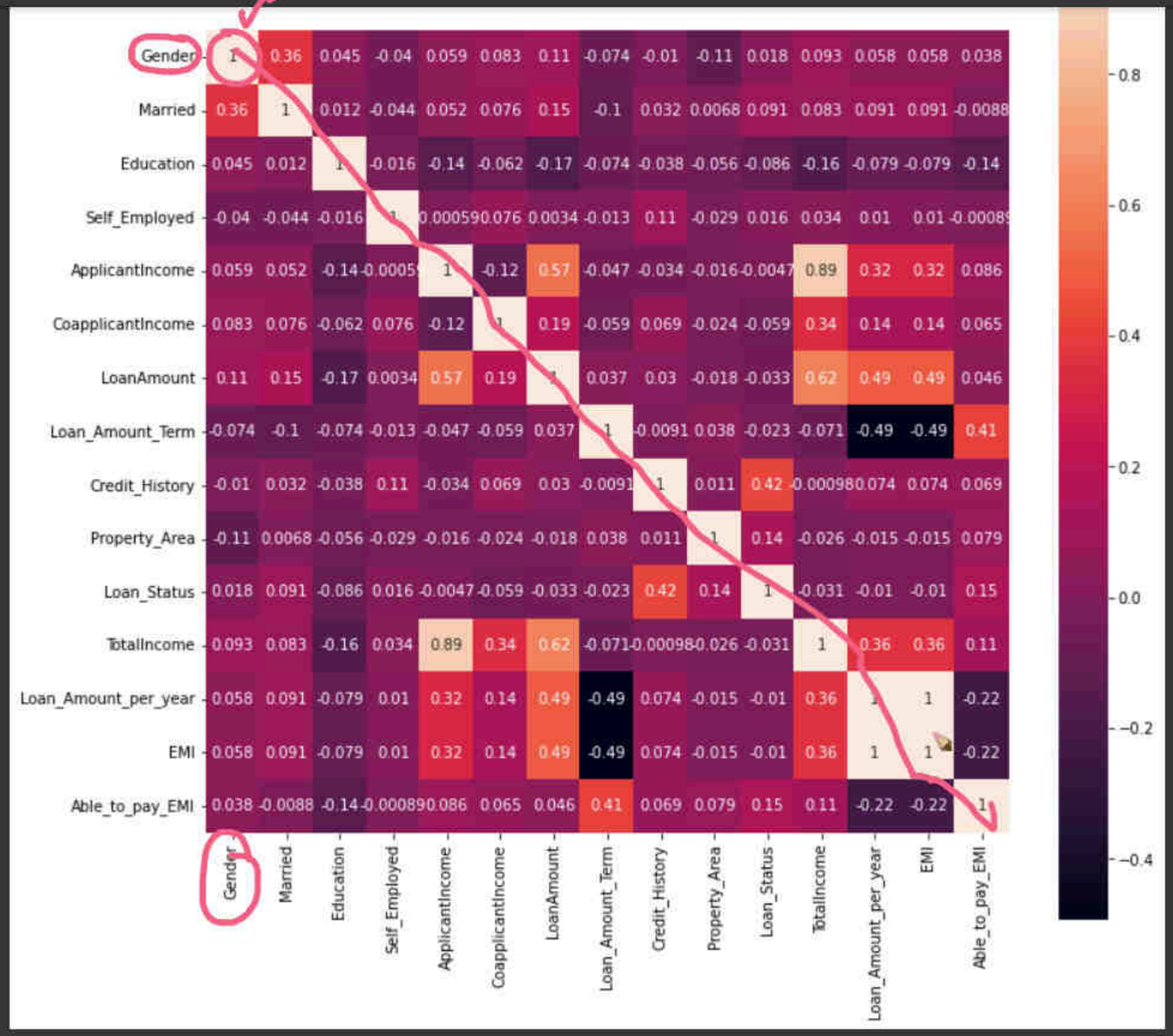

Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 -0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01 -0.00089
ApplicantIncome 0.059 0.052 -0.14 -0.00059 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.0047 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 -0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 0.086 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

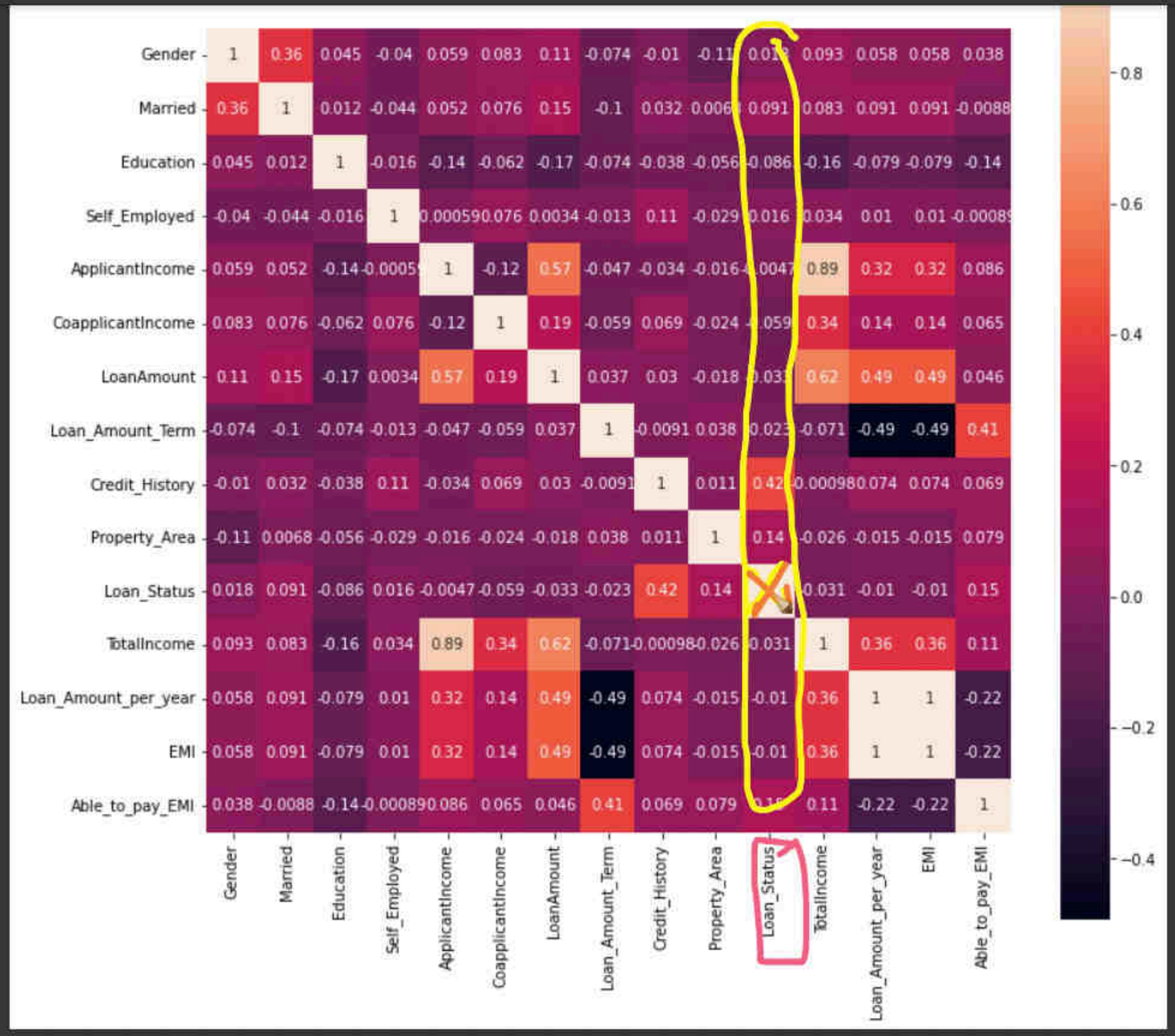
Correlation Coefficients

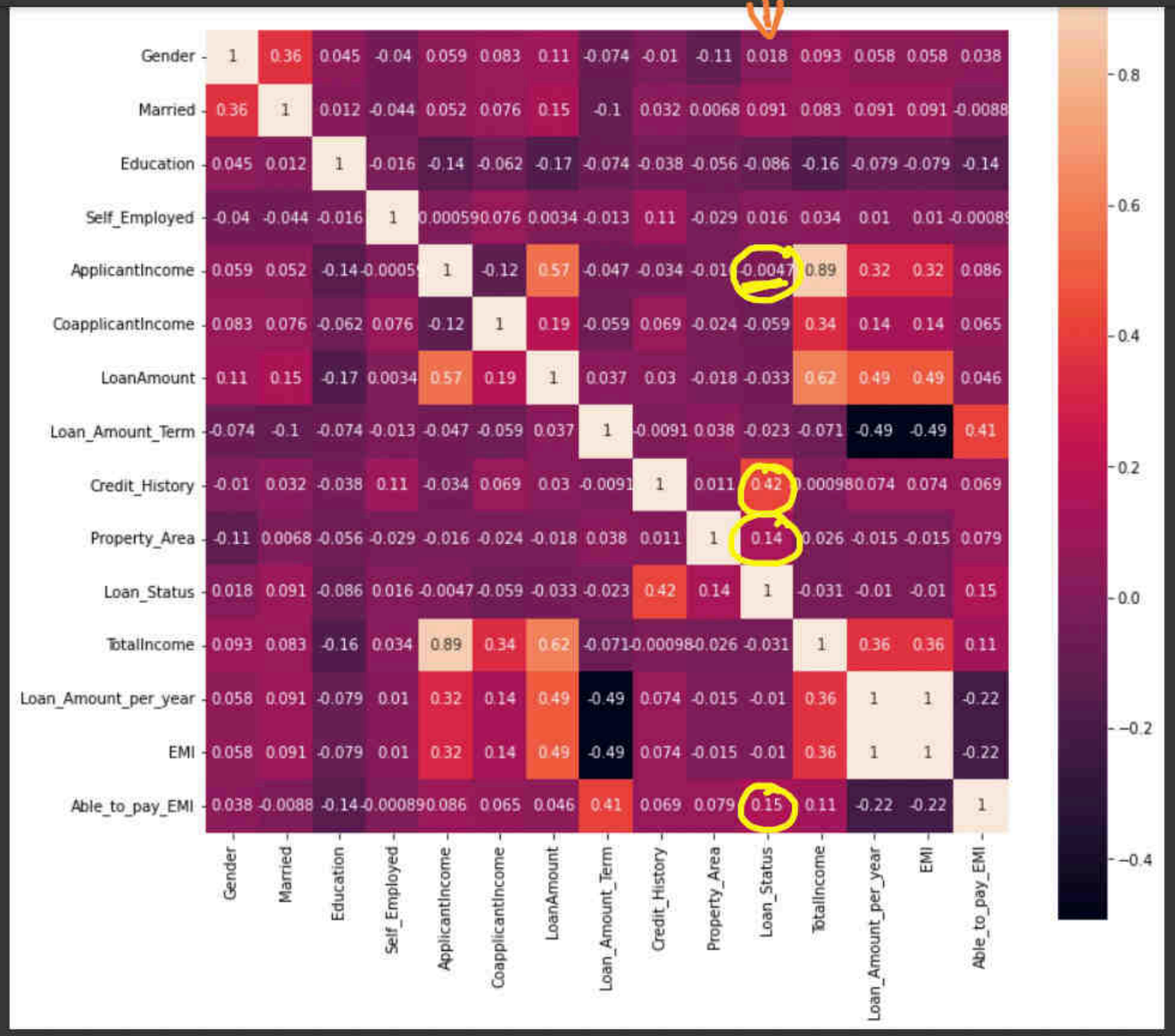
```
#PCC
plt.figure(figsize=(12, 12))
sns.heatmap(data.corr(method='pearson'), square=True,annot=True)
```

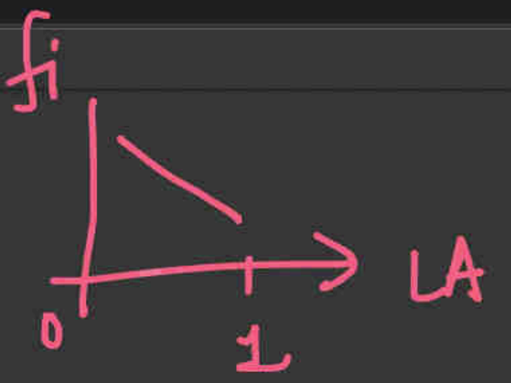
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac6315f450>
```




Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01 -0.00089
ApplicantIncome 0.059 0.052 -0.14 -0.0005 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.0047 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 -0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 0.086 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.006 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01 -0.00089
ApplicantIncome 0.059 0.052 -0.14 -0.0005 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.0047 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 -0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 0.086 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

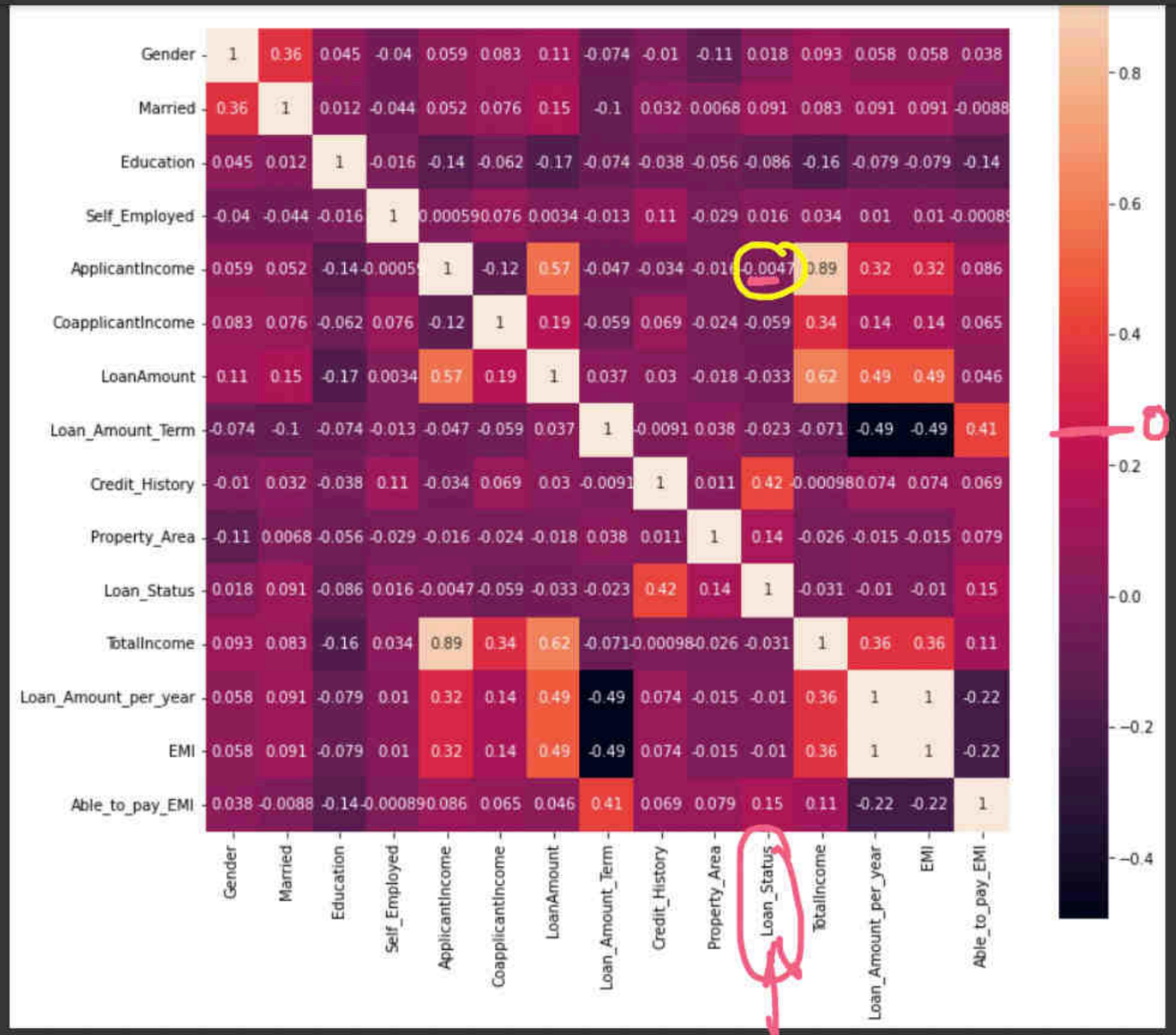


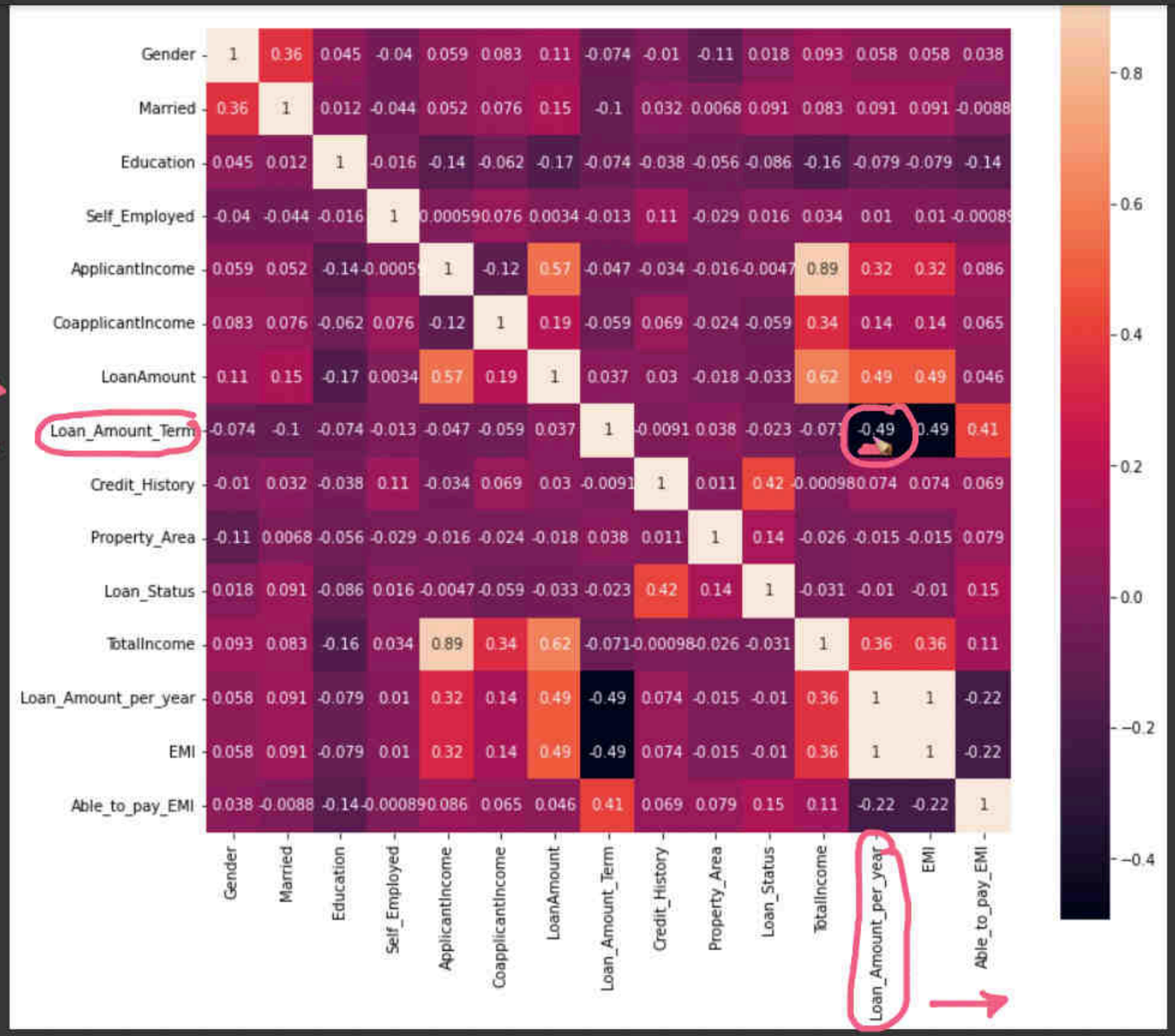


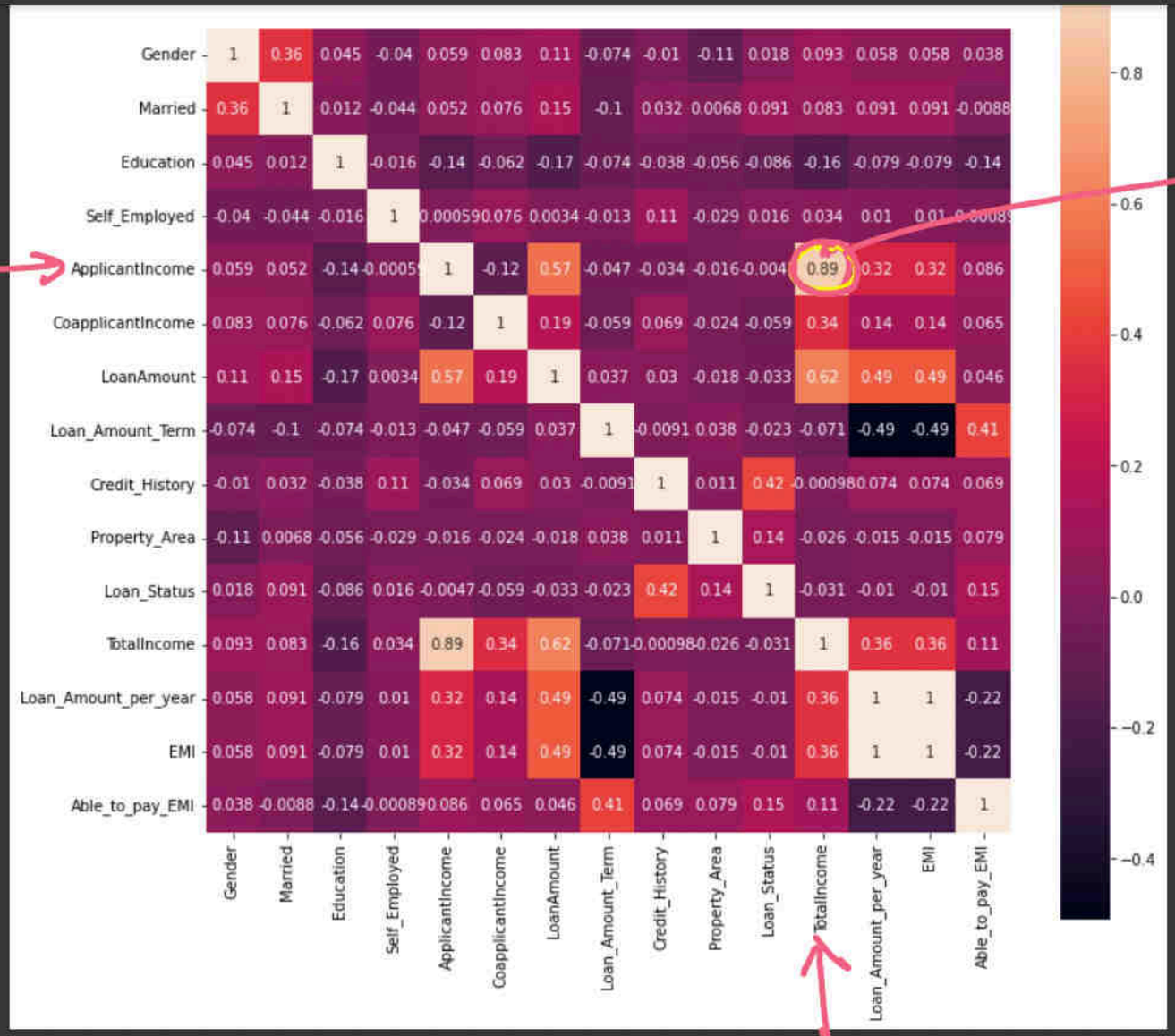
PCC

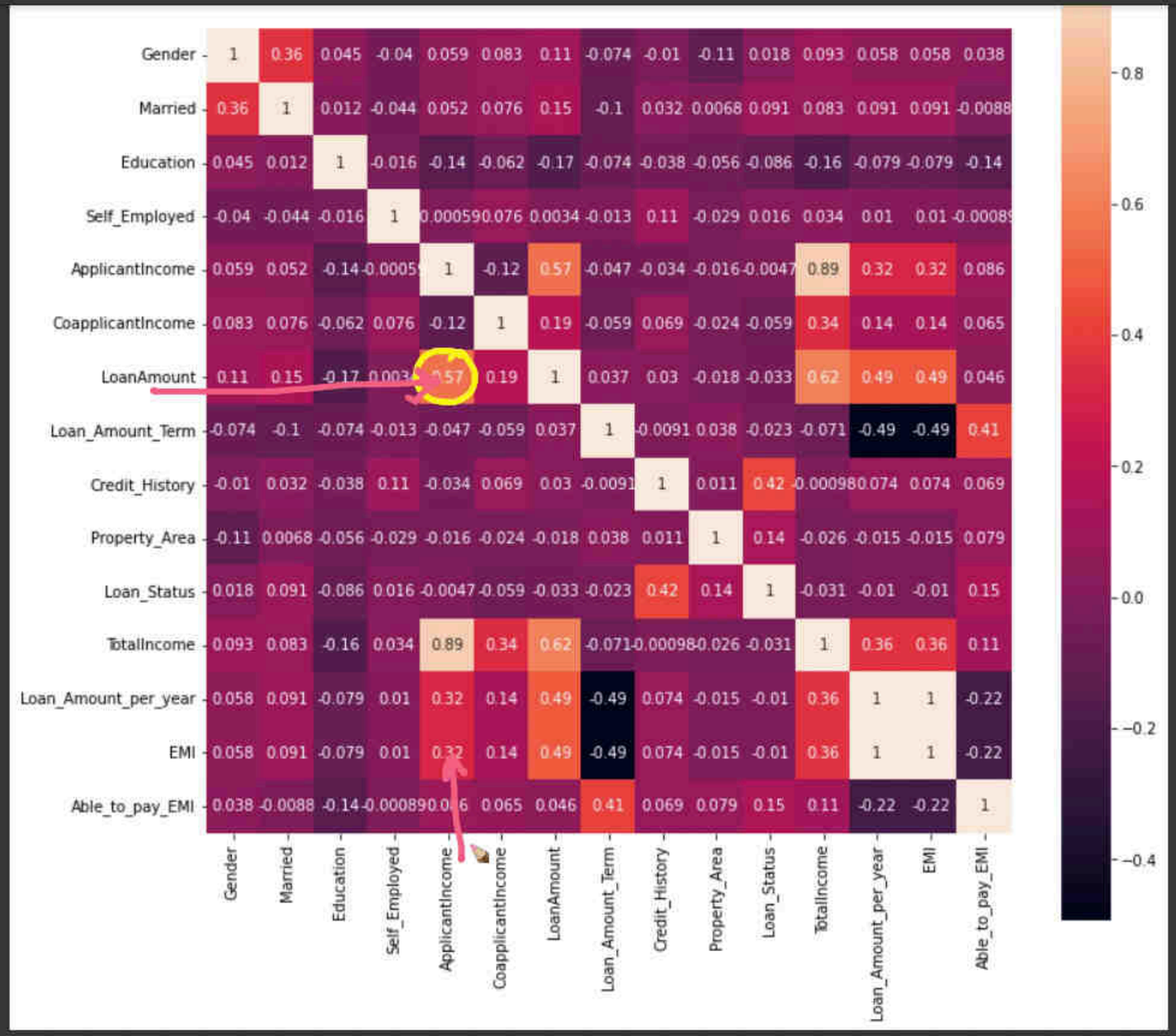
1. CH ✓
2. ABLC - EMI ✓
3. Proop-area ✓

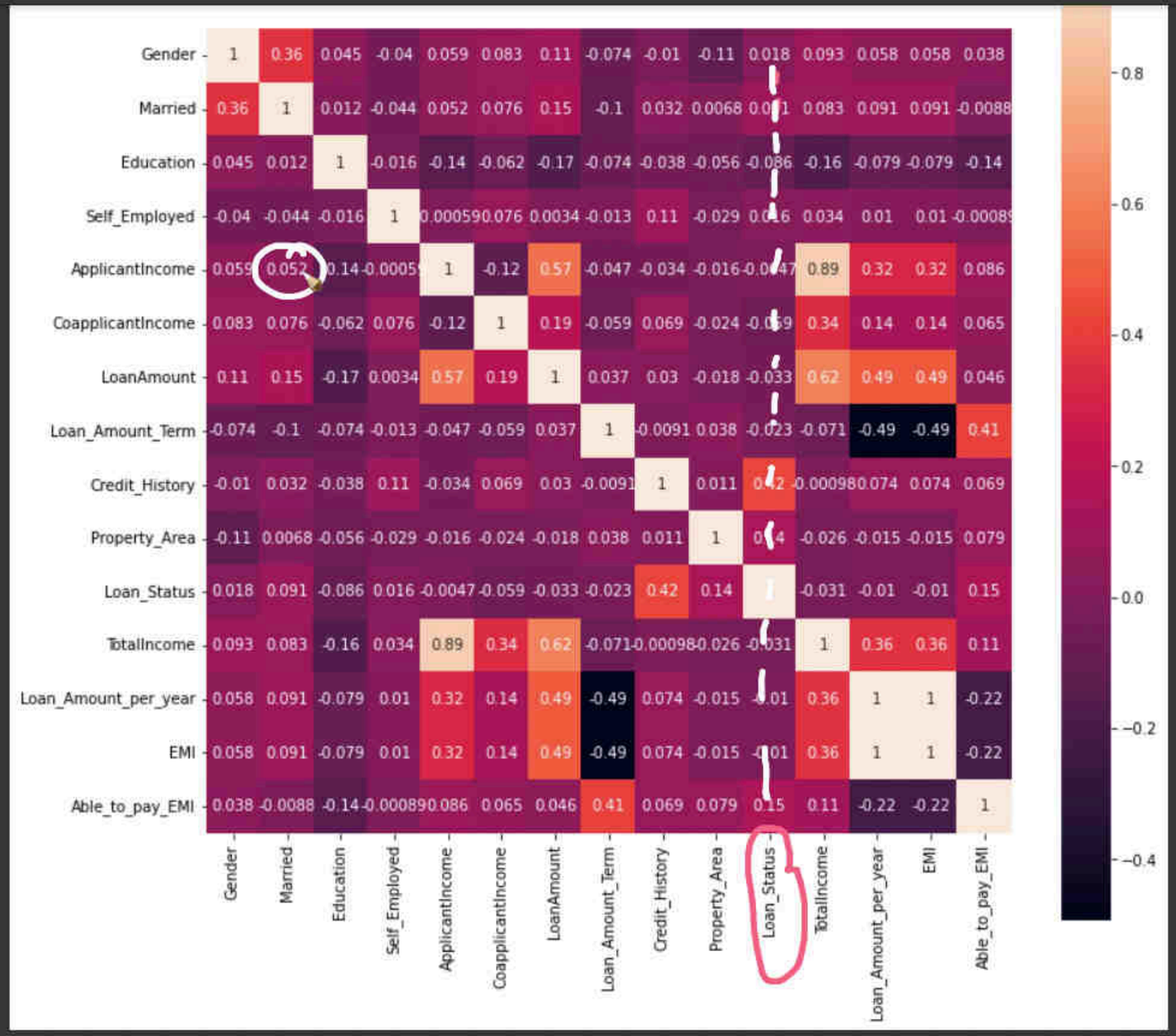
least useful: Applicome


Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01 -0.00089
ApplicantIncome 0.059 0.052 -0.14 -0.0005 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.0047 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 -0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 0.086 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

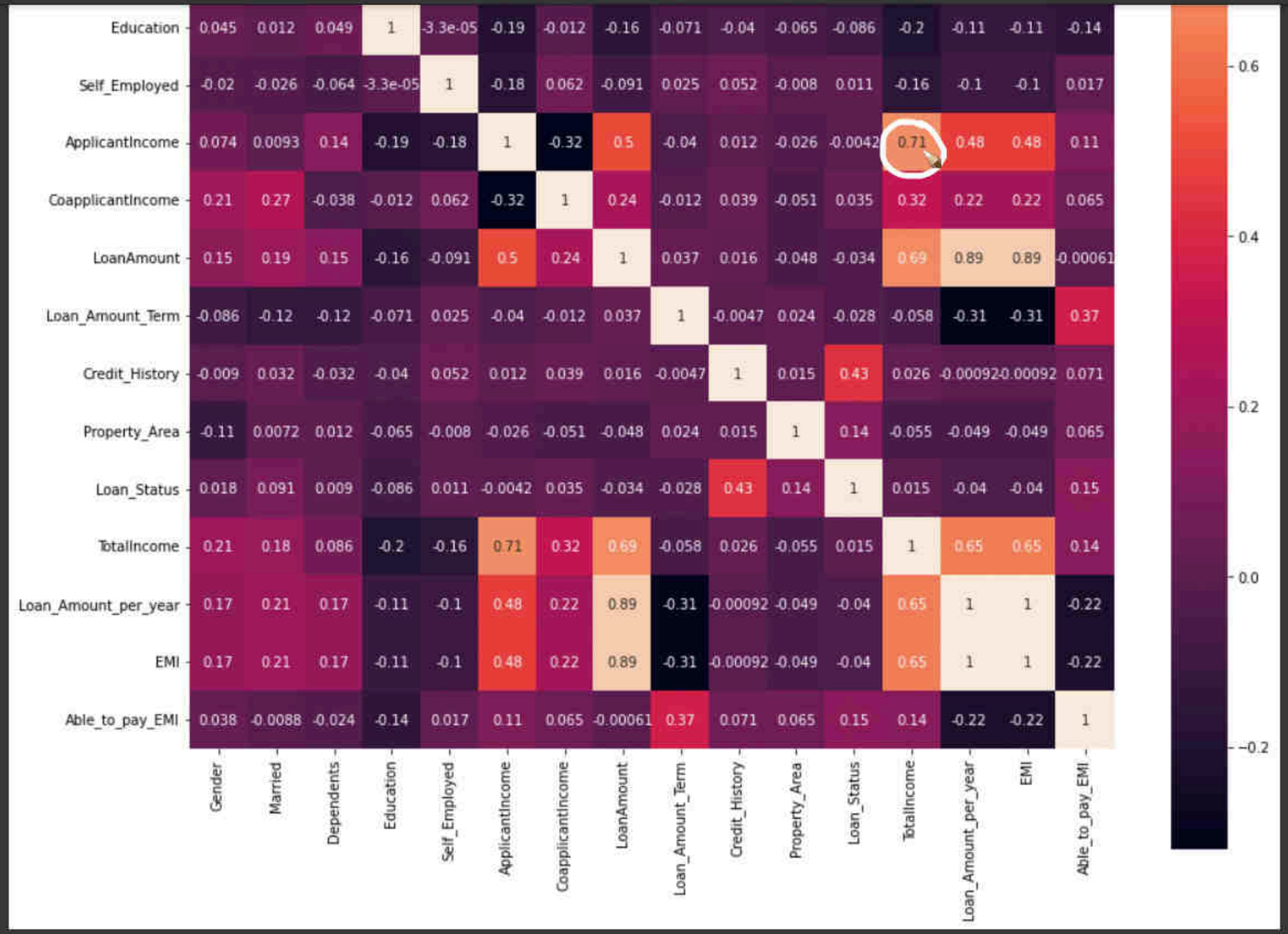

Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01 -0.00089
ApplicantIncome 0.059 0.052 -0.14 -0.0005 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.0047 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 0.086 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01
ApplicantIncome 0.059 0.052 -0.14 -0.0005 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.004 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 -0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 0.086 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

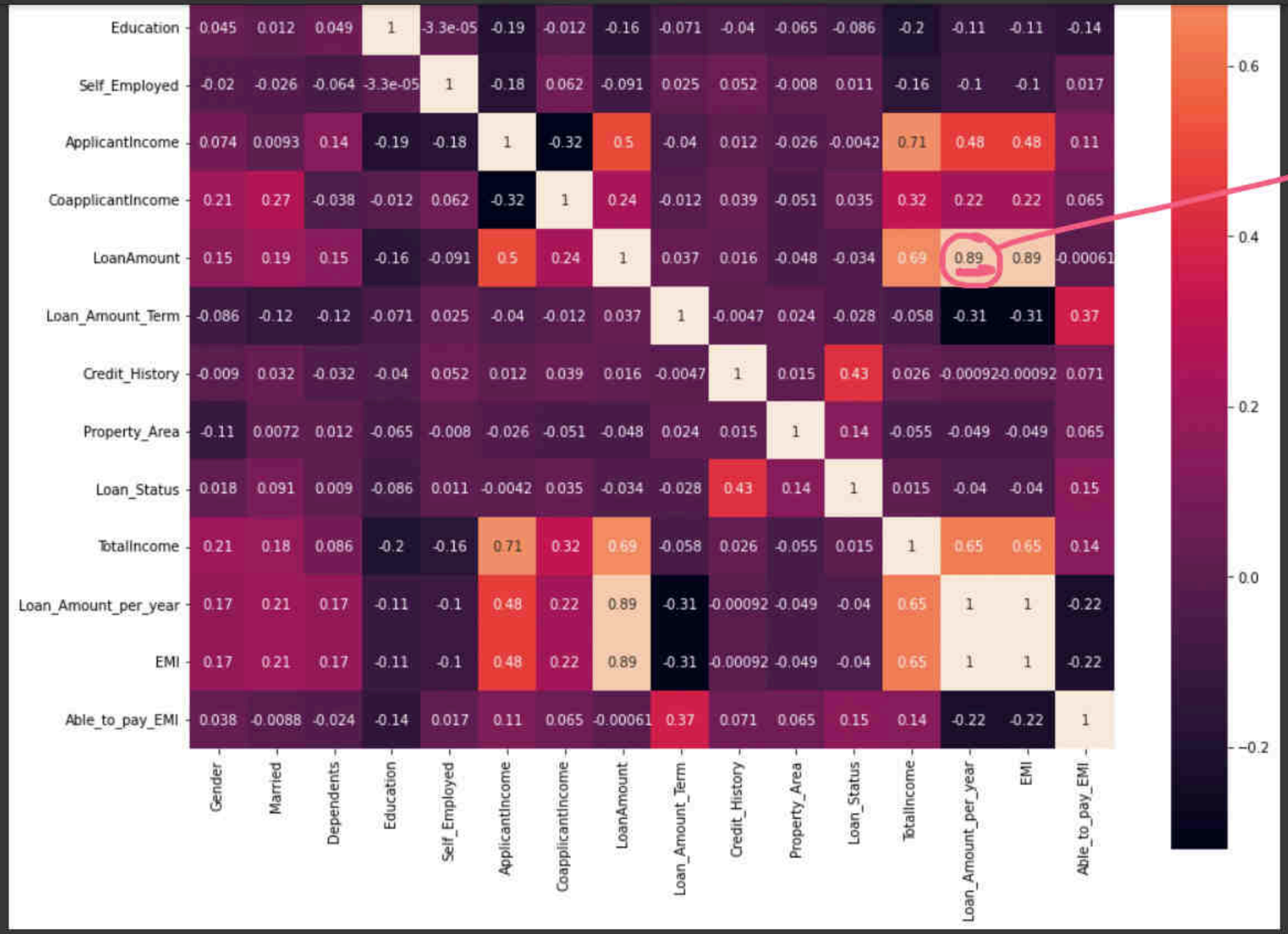

Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01 -0.00089
ApplicantIncome 0.059 0.052 -0.14 -0.0005 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.0047 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.003 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 -0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 -0.06 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Gender 1 0.36 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
Married 0.36 1 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
Education 0.045 0.012 1 -0.016 -0.14 -0.062 -0.17 -0.074 -0.038 -0.056 -0.086 -0.16 -0.079 -0.079 -0.14
Self_Employed -0.04 -0.044 -0.016 1 0.00059 0.076 0.0034 -0.013 0.11 -0.029 0.016 0.034 0.01 0.01 -0.00089
ApplicantIncome 0.059 0.052 0.14 -0.0005 1 -0.12 0.57 -0.047 -0.034 -0.016 -0.0047 0.89 0.32 0.32 0.086
CoapplicantIncome 0.083 0.076 -0.062 0.076 -0.12 1 0.19 -0.059 0.069 -0.024 -0.059 0.34 0.14 0.14 0.065
LoanAmount 0.11 0.15 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49 0.046
Loan_Amount_Term -0.074 -0.1 -0.074 -0.013 -0.047 -0.059 0.037 1 -0.0091 0.038 -0.023 -0.071 -0.49 -0.49 0.41
Credit_History -0.01 0.032 -0.038 0.11 -0.034 0.069 0.03 -0.0091 1 0.011 0.42 -0.00098 0.074 0.074 0.069
Property_Area -0.11 0.0068 -0.056 -0.029 -0.016 -0.024 -0.018 0.038 0.011 1 0.14 -0.026 -0.015 -0.015 0.079
Loan_Status 0.018 0.091 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 1 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 -0.16 0.034 0.89 0.34 0.62 -0.071 -0.00098 -0.026 -0.031 1 0.36 0.36 0.11
Loan_Amount_per_year 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
EMI 0.058 0.091 -0.079 0.01 0.32 0.14 0.49 -0.49 0.074 -0.015 -0.01 0.36 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.14 -0.00089 0.086 0.065 0.046 0.41 0.069 0.079 0.15 0.11 -0.22 -0.22 1
Gender
Married
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Corr(fi, fj)

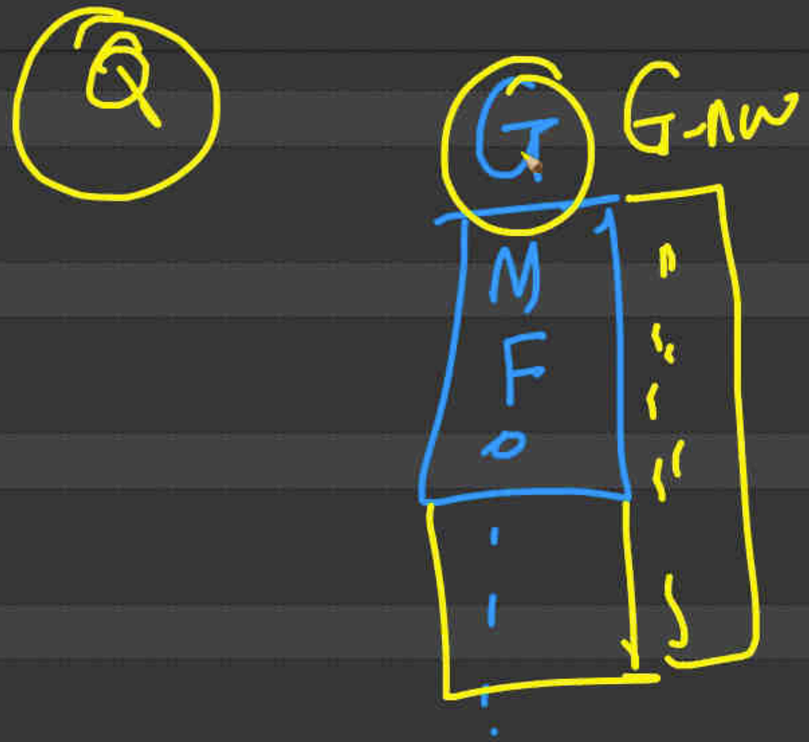



```
[ ]  plt.scatter(data['Credit_History'], data['Loan_Status'])
```



30yrs lo

```
[ ]  plt.scatter(data['Credit_History'], data['Loan_Status'])
```



[ ] ↳ 2 cells hidden

Property Area

[ ] ↳ 4 cells hidden

Education

[ ] ↳ 2 cells hidden

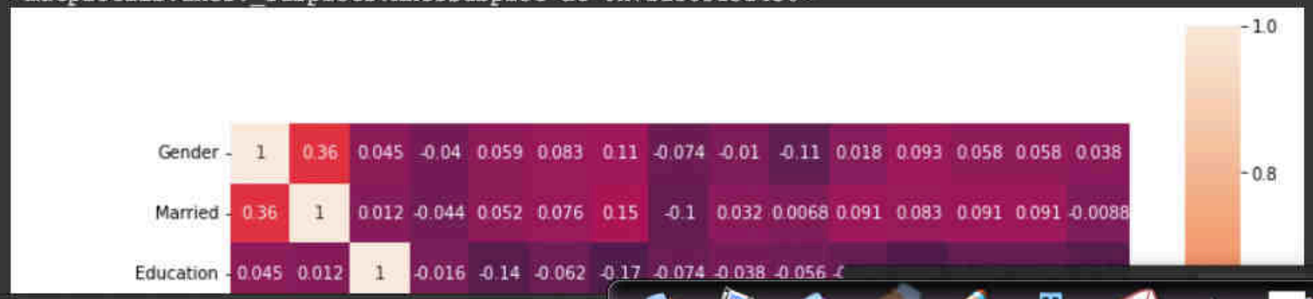
Self Employed

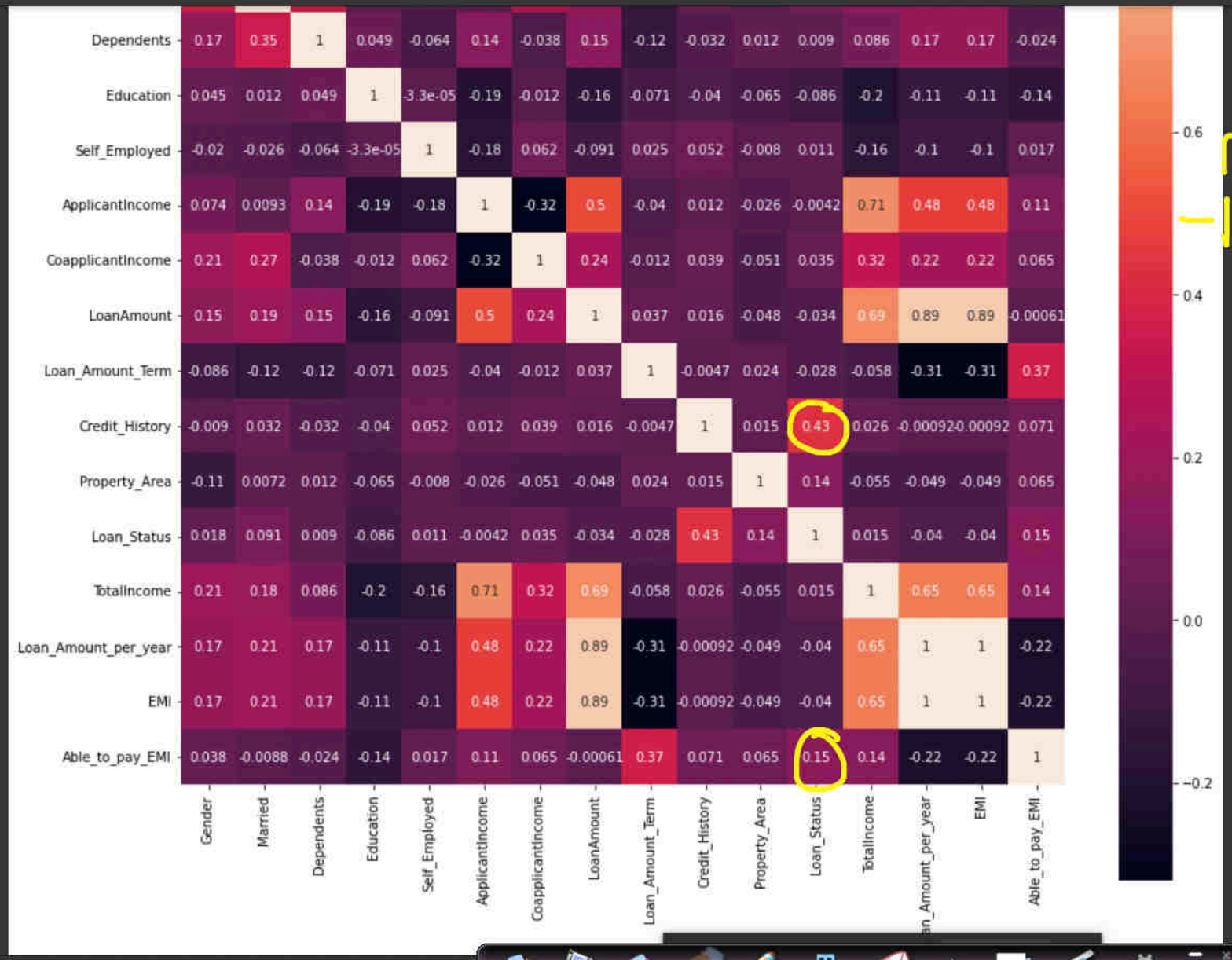
[ ] ↳ 4 cells hidden

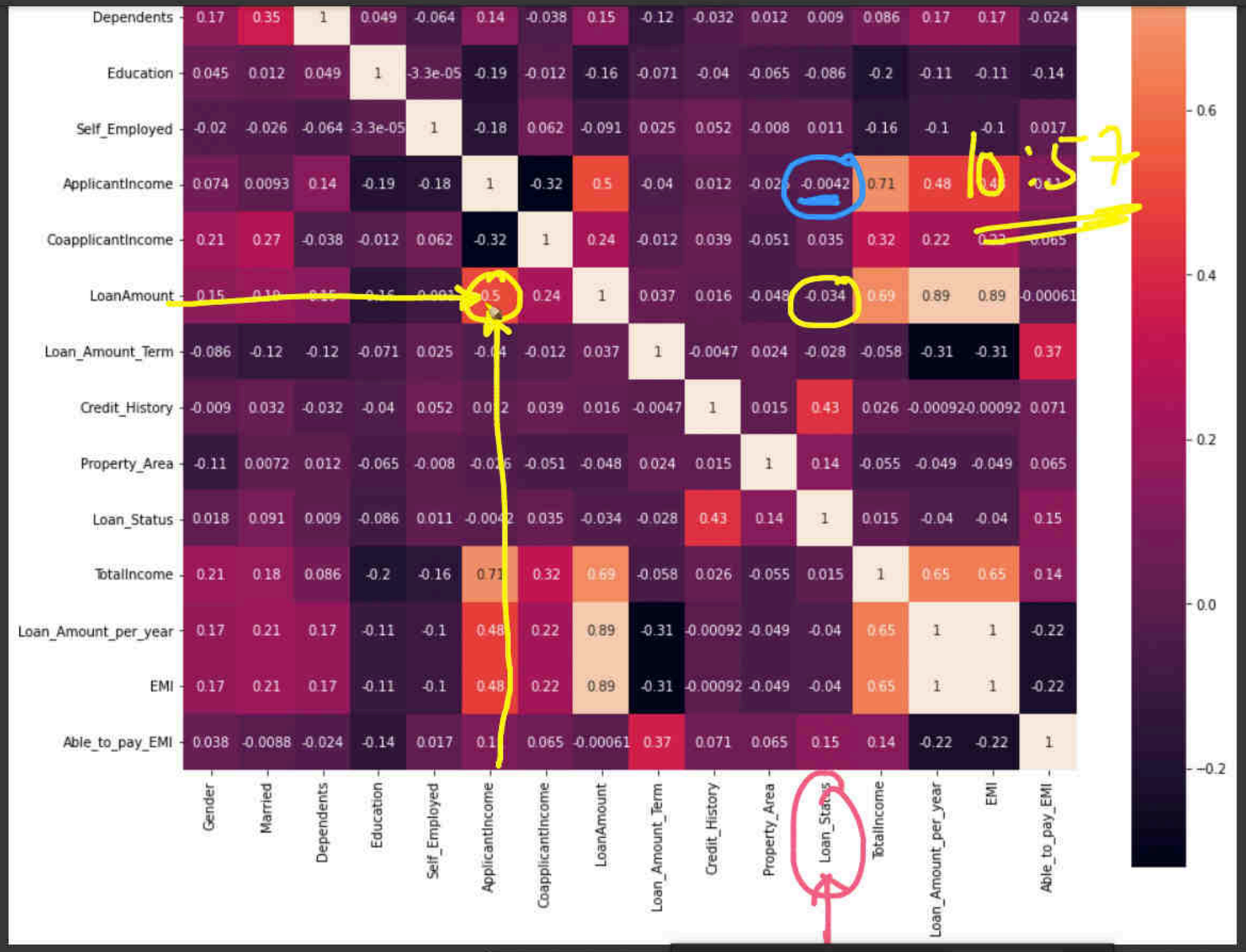
Correlation Coefficients

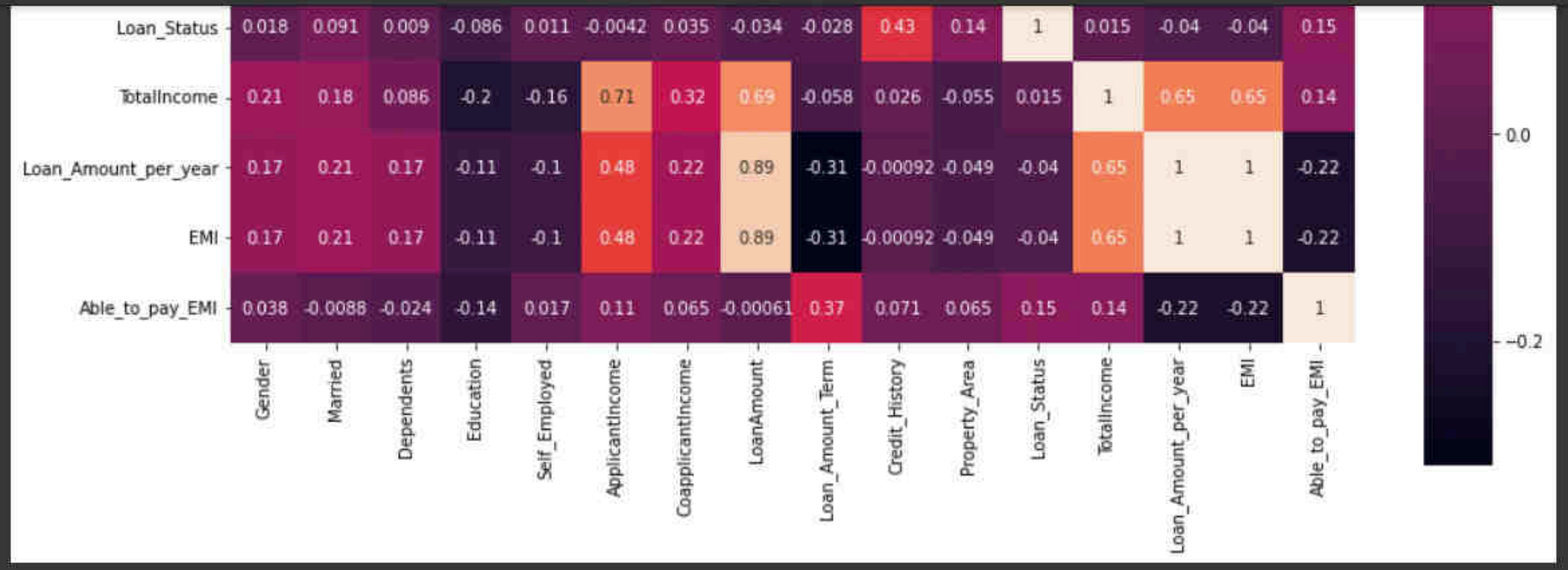
```
#PCC
plt.figure(figsize=(12, 12))
sns.heatmap(data.corr(method='pearson'), square=True,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac6315f450>
```



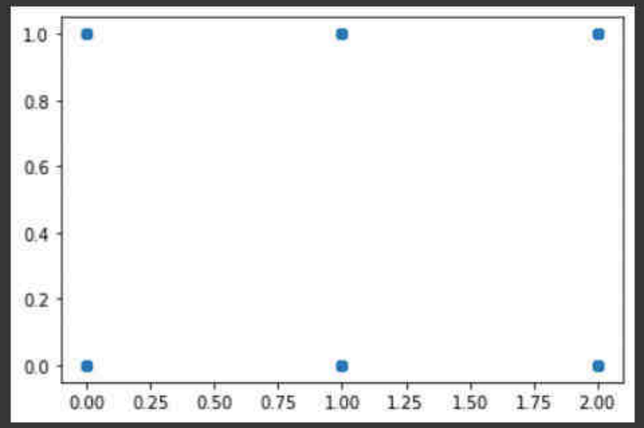

Dependents 0.17 0.35 1 0.049 -0.064 0.14 -0.038 0.15 -0.12 -0.032 0.012 0.009 0.086 0.17 0.17 -0.024
Education 0.045 0.012 0.049 1 -3.3e-05 -0.19 -0.012 -0.16 -0.071 -0.04 -0.065 -0.086 -0.2 -0.11 -0.11 -0.14
Self_Employed -0.02 -0.026 -0.064 -3.3e-05 1 -0.18 0.062 -0.091 0.025 0.052 -0.008 0.011 -0.16 -0.1 -0.1 0.017
ApplicantIncome 0.074 0.0093 0.14 -0.19 -0.18 1 -0.32 0.5 -0.04 0.012 -0.026 -0.0042 0.71 0.48 0.48 0.11
CoapplicantIncome 0.21 0.27 -0.038 -0.012 0.062 -0.32 1 0.24 -0.012 0.039 -0.051 0.035 0.32 0.22 0.22 0.065
LoanAmount 0.15 0.19 0.15 -0.16 -0.091 0.5 0.24 1 0.037 0.016 -0.048 -0.034 0.69 0.89 0.89 -0.00061
Loan_Amount_Term -0.086 -0.12 -0.12 -0.071 0.025 -0.04 -0.012 0.037 1 -0.0047 0.024 -0.028 -0.058 -0.31 -0.31 0.37
Credit_History -0.009 0.032 -0.032 -0.04 0.052 0.012 0.039 0.016 -0.0047 1 0.015 0.43 0.026 -0.00092 -0.00092 0.071
Property_Area -0.11 0.0072 0.012 -0.065 -0.008 -0.026 -0.051 -0.048 0.024 0.015 1 0.14 -0.055 -0.049 -0.049 0.065
Loan_Status 0.018 0.091 0.009 -0.086 0.011 -0.0042 0.035 -0.034 -0.028 0.43 0.14 1 0.015 -0.04 -0.04 0.15
TotalIncome 0.21 0.18 0.086 -0.2 -0.16 0.71 0.32 0.69 -0.058 0.026 -0.055 0.015 1 0.65 0.65 0.14
Loan_Amount_per_year 0.17 0.21 0.17 -0.11 -0.1 0.48 0.22 0.89 -0.31 -0.00092 -0.049 -0.04 0.65 1 1 -0.22
EMI 0.17 0.21 0.17 -0.11 -0.1 0.48 0.22 0.89 -0.31 -0.00092 -0.049 -0.04 0.65 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.024 -0.14 0.017 0.11 0.065 -0.00061 0.37 0.071 0.065 0.15 0.14 -0.22 -0.22 1
Gender
Married
Dependents
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.6
0.4
0.2
0.0
−0.2


Dependents 0.17 0.35 1 0.049 -0.064 0.14 -0.038 0.15 -0.12 -0.032 0.012 0.009 0.086 0.17 0.17 -0.024
Education 0.045 0.012 0.049 1 -3.3e-05 -0.19 -0.012 -0.16 -0.071 -0.04 -0.065 -0.086 -0.2 -0.11 -0.11 -0.14
Self_Employed -0.02 -0.026 -0.064 -3.3e-05 1 -0.18 0.062 -0.091 0.025 0.052 -0.008 0.011 -0.16 -0.1 -0.1 0.017
ApplicantIncome 0.074 0.0093 0.14 -0.19 -0.18 1 -0.32 0.5 -0.04 0.012 -0.0042 0.71 0.48
CoapplicantIncome 0.21 0.27 -0.038 -0.012 0.062 -0.32 1 0.24 -0.012 0.039 -0.051 0.035 0.32 0.22
LoanAmount 0.24 1 0.037 0.016 -0.034 0.69 0.89 0.89 -0.00061
Loan_Amount_Term -0.086 -0.12 -0.12 -0.071 0.025 -0.012 0.037 1 -0.0047 0.024 -0.028 -0.058 -0.31 -0.31 0.37
Credit_History -0.009 0.032 -0.032 -0.04 0.052 0.039 0.016 -0.0047 1 0.015 0.43 0.026 -0.00092 -0.00092 0.071
Property_Area -0.11 0.0072 0.012 -0.065 -0.008 -0.051 -0.048 0.024 0.015 1 0.14 -0.055 -0.049 -0.049 0.065
Loan_Status 0.018 0.091 0.009 -0.086 0.011 0.035 -0.034 -0.028 0.43 0.14 1 0.015 -0.04 -0.04 0.15
TotalIncome 0.21 0.18 0.086 -0.2 -0.16 0.71 0.32 0.69 -0.058 0.026 -0.055 0.015 1 0.65 0.65 0.14
Loan_Amount_per_year 0.17 0.21 0.17 -0.11 -0.1 0.48 0.22 0.89 -0.31 -0.00092 -0.049 -0.04 0.65 1 1 -0.22
EMI 0.17 0.21 0.17 -0.11 -0.1 0.48 0.22 0.89 -0.31 -0.00092 -0.049 -0.04 0.65 1 1 -0.22
Able_to_pay_EMI 0.038 -0.0088 -0.024 -0.14 0.017 0.1 0.065 -0.00061 0.37 0.071 0.065 0.15 0.14 -0.22 -0.22 1
Gender
Married
Dependents
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
TotalIncome
Loan_Amount_per_year
EMI
Able_to_pay_EMI
0.6
0.4
0.2
0.0
−0.2



10:57

```
[ ]  plt.scatter(data['Credit_History'], data['Loan_Status'])
     plt.show()
     #sometimes scatter plots can be misleading do to catgeorical nature of the data
```



Q

Multi-class

LS

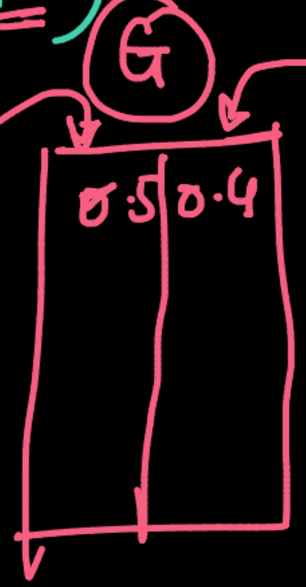
$y_i \rightarrow$ S M L (K)

{ G = M / F / O

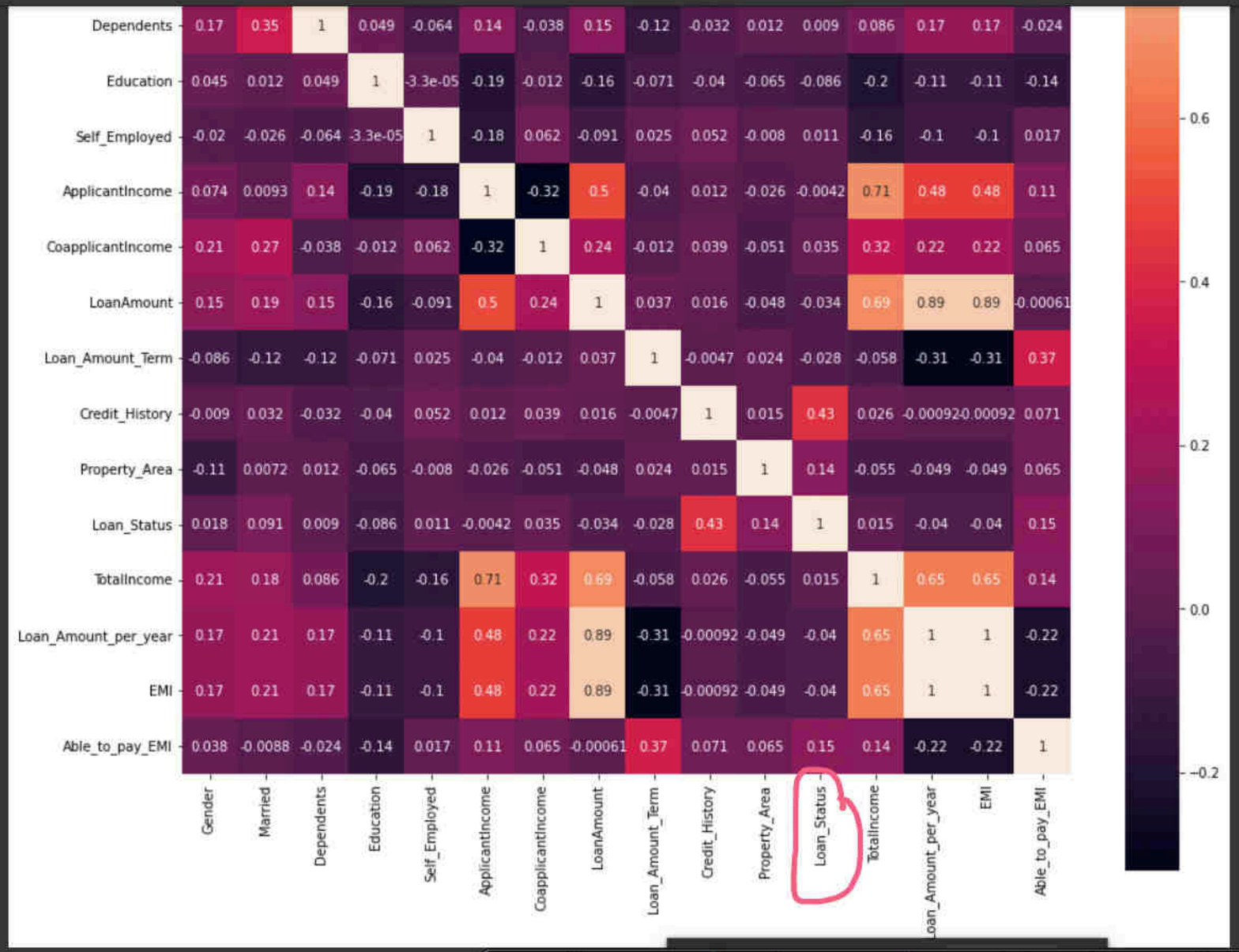
(later) feature imp (ML)

$P(y_i = S \mid G = M)$

$P(y_i = M \mid G = M)$



χ^2-test ✓

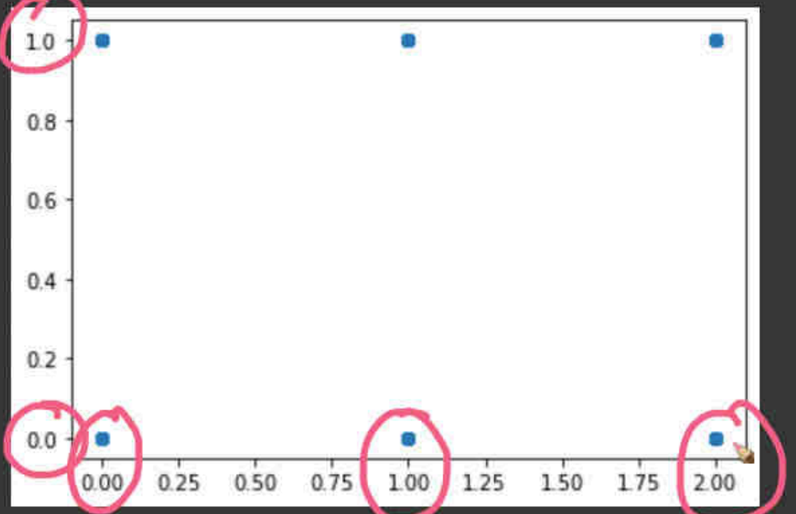


Univariate

↳ fi vs LA

[fi, fj] vs LA


```
plt.scatter(data['Credit_History'], data['Loan_Status'])
plt.show()
#sometimes scatter plots can be misleading do to catgeorical nature of the data
```

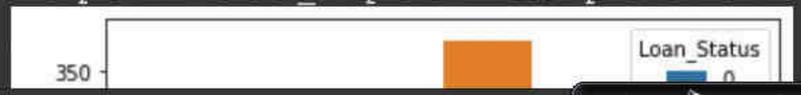


LA

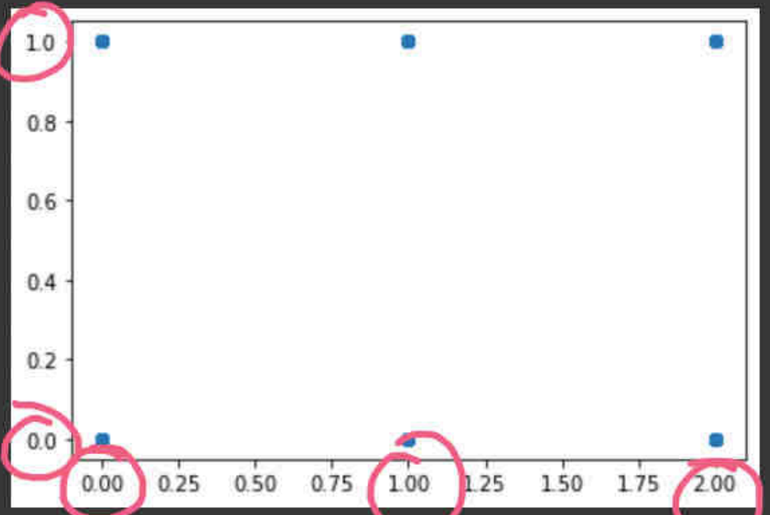
CH

```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>
```



```
plt.scatter(data['Credit_History'], data['Loan_Status'])
plt.show()
#sometimes scatter plots can be misleading do to catgeorical nature of the data
```



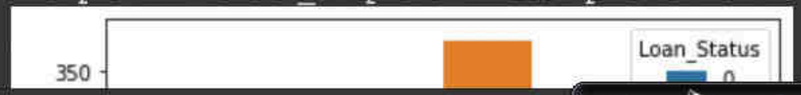
LA

CH

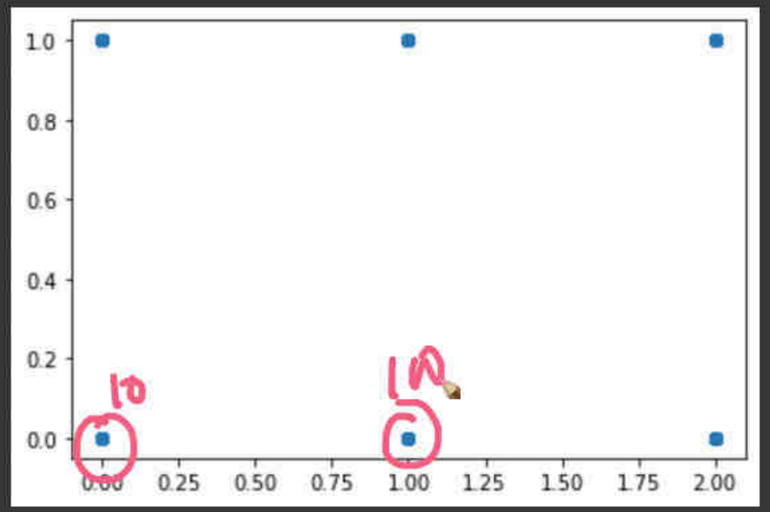
VS SRCC → 0.45

```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>
```

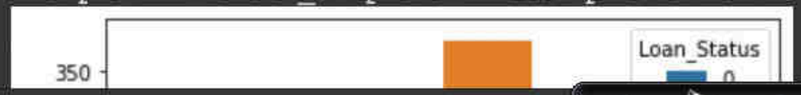



```
plt.scatter(data['Credit_History'], data['Loan_Status'])
plt.show()
#sometimes scatter plots can be misleading do to catgeorical nature of the data
```

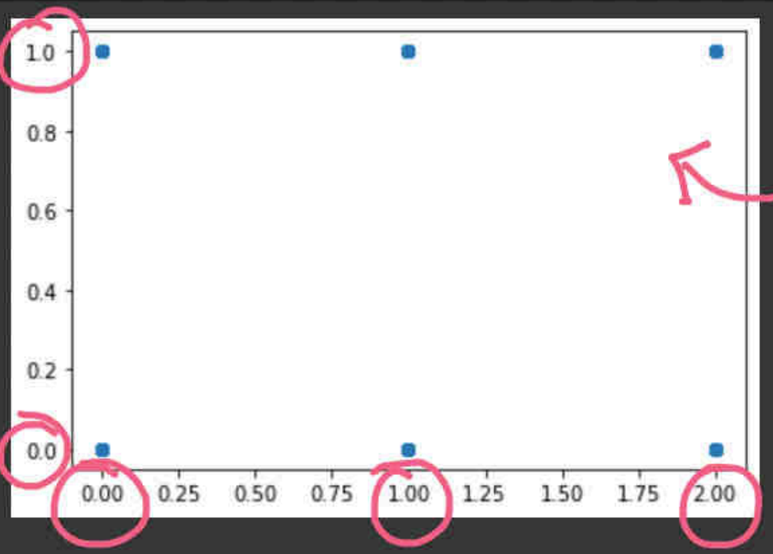



```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>
```



```
plt.scatter(data['Credit_History'], data['Loan_Status'])
plt.show()
#sometimes scatter plots can be misleading do to catgeorical nature of the data
```



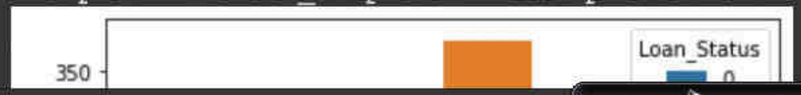
LA

CH

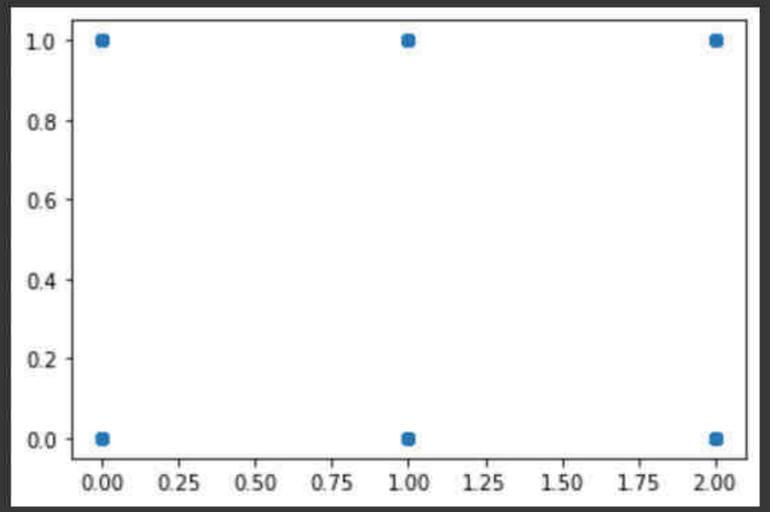
613

```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>
```

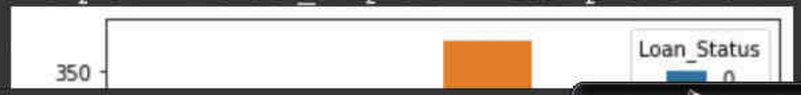


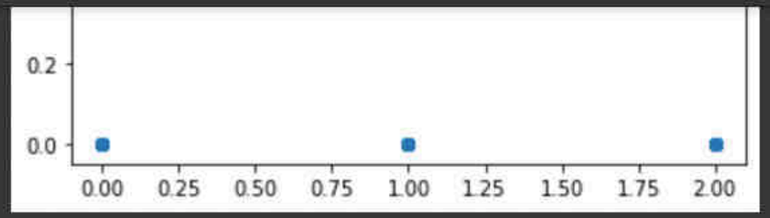

```
plt.scatter(data['Credit_History'], data['Loan_Status'])
plt.show()
#sometimes scatter plots can be misleading do to catgeorical nature of the data
```



```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

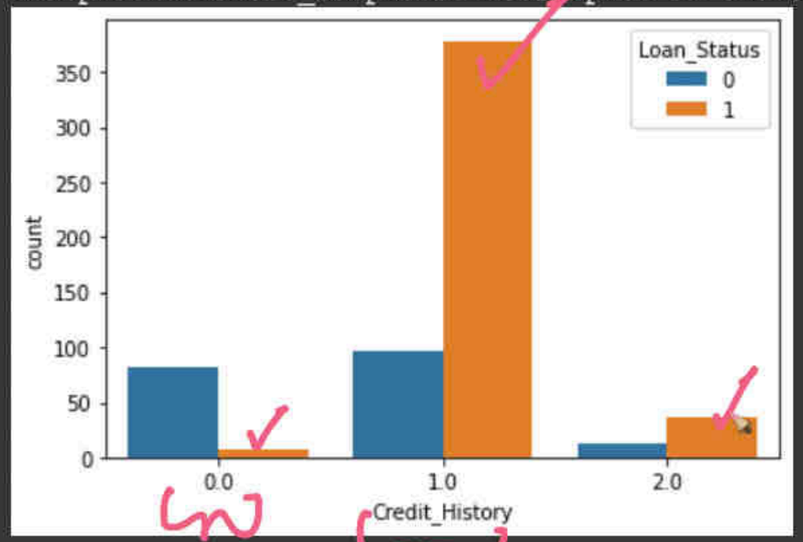
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>
```






```
[ ]  sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>
```



Column Standarization and Normalization

- Mean centering and Variance scaling (Standard Scaling)
- MinMax Scaling

Col. Std & normalization

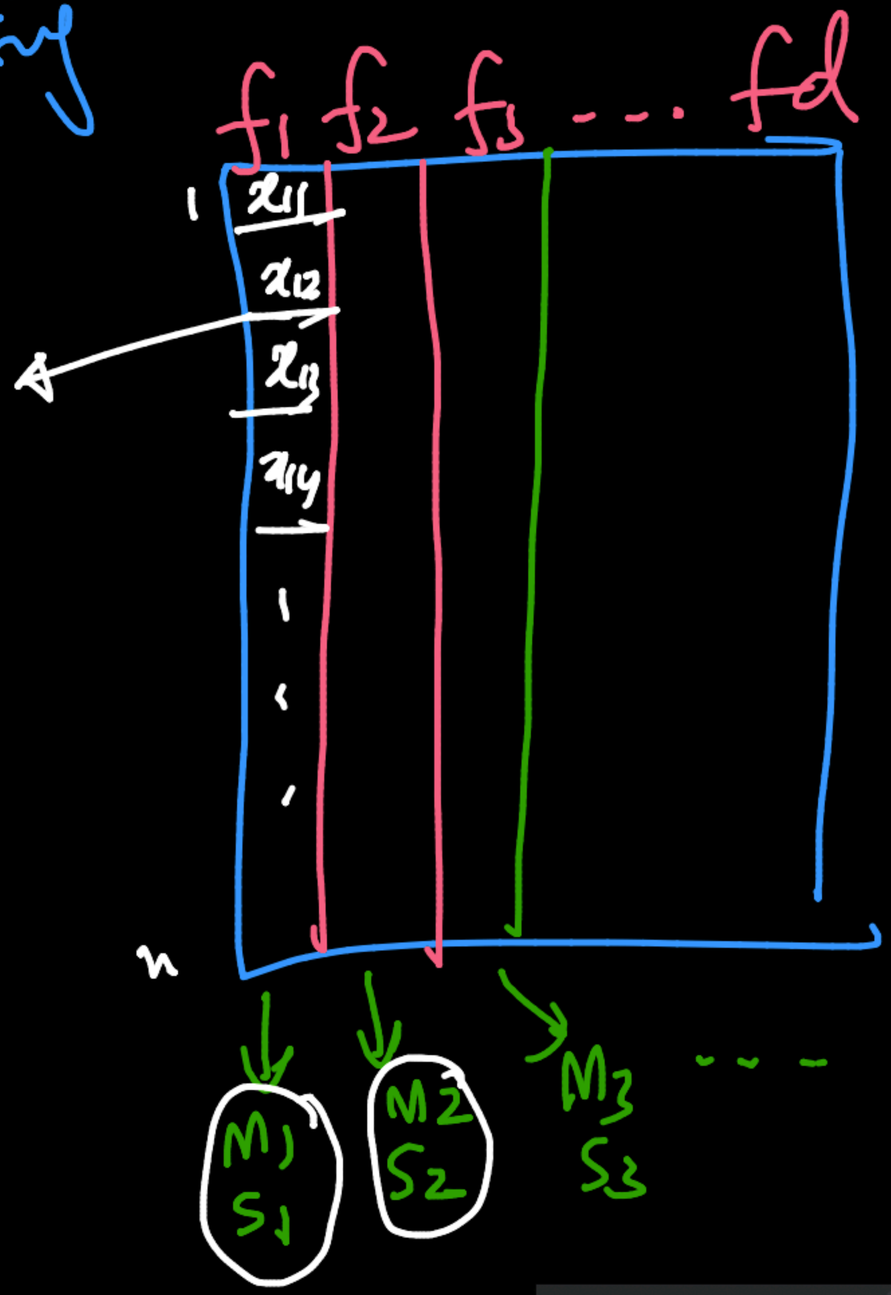
$X = x_1 \dots x_n$

$\frac{x_i - m}{s} \Rightarrow$ standardization

Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:09

Pre-processing

- Linear reg
- DL (later)



$$\frac{x_{1i} - m_1}{s_1}$$

colum. std

↓

each col. is std.. independently

everything has to be numbers

Categorical → Numeric → ML

why → aids in optimization problems (ML)

→ f1 are sensible

later ...

Cat → 0 or 1 → leave it as is

$X = x_1, x_2 \ldots x_i \ldots x_n$ = num...

NO-NO-NO

N(011)

Std → mean Centering & Var. scaling

$x_i'' = \frac{x_i - m}{\sigma}$

{ mean = 0, $\sigma = 1$

Normalization

→ Image-processing & CV

for each col

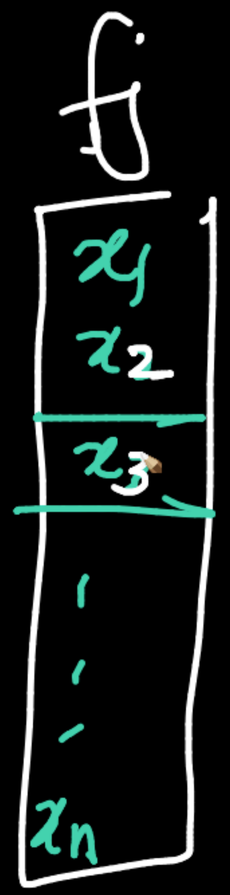
- Min-Max Scaling

min

max

$$\frac{x_i - \min}{\max - \min} = x_i'$$

↳ $[0, 1]$



+ Code + Text

RAM
Disk

Column Standarization and Normalization

- Mean centering and Variance scaling (Standard Scaling)
- MinMax Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = StandardScaler()
std_data = scaler.fit_transform(data)
std_data = pd.DataFrame(std_data, columns=data.columns)
std_data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount
0	0.472343	-1.372089	-0.737806	-0.528362	-0.174052	0.072991	-0.554487	-0.211241	0.2
1	0.472343	0.728816	0.253470	-0.528362	-0.174052	-0.134412	-0.038732	-0.211241	0.2
2	0.472343	0.728816	-0.737806	-0.528362	-0.586643	-0.393747	-0.554487	-0.948996	0.2
3	0.472343	0.728816	-0.737806	1.892641	-0.174052	-0.462062	0.251980	-0.306435	0.2
4	0.472343	-1.372089	-0.737806	-0.528362	-0.174052	0.097728	-0.554487	-0.056551	0.2

Column Standarization and Normalization

- Mean centering and Variance scaling (Standard Scaling)
- MinMax Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = StandardScaler()
std_data = scaler.fit_transform(data)
std_data = pd.DataFrame(std_data, columns=data.columns)
std_data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount
0	0.472343	-1.372089	-0.737806	-0.528362	-0.174052	0.072991	-0.554487	-0.211241	0.2
1	0.472343	0.728816	0.253470	-0.528362	-0.174052	-0.134412	-0.038732	-0.211241	0.2
2	0.472343	0.728816	-0.737806	-0.528362	-0.586643	-0.393747	-0.554487	-0.948996	0.2
3	0.472343	0.728816	-0.737806	1.892641	-0.174052	-0.462062	0.251980	-0.306435	0.2
4	0.472343	-1.372089	-0.737806	-0.528362	-0.174052	0.097728	-0.554487	-0.056551	0.2

Kurtosis

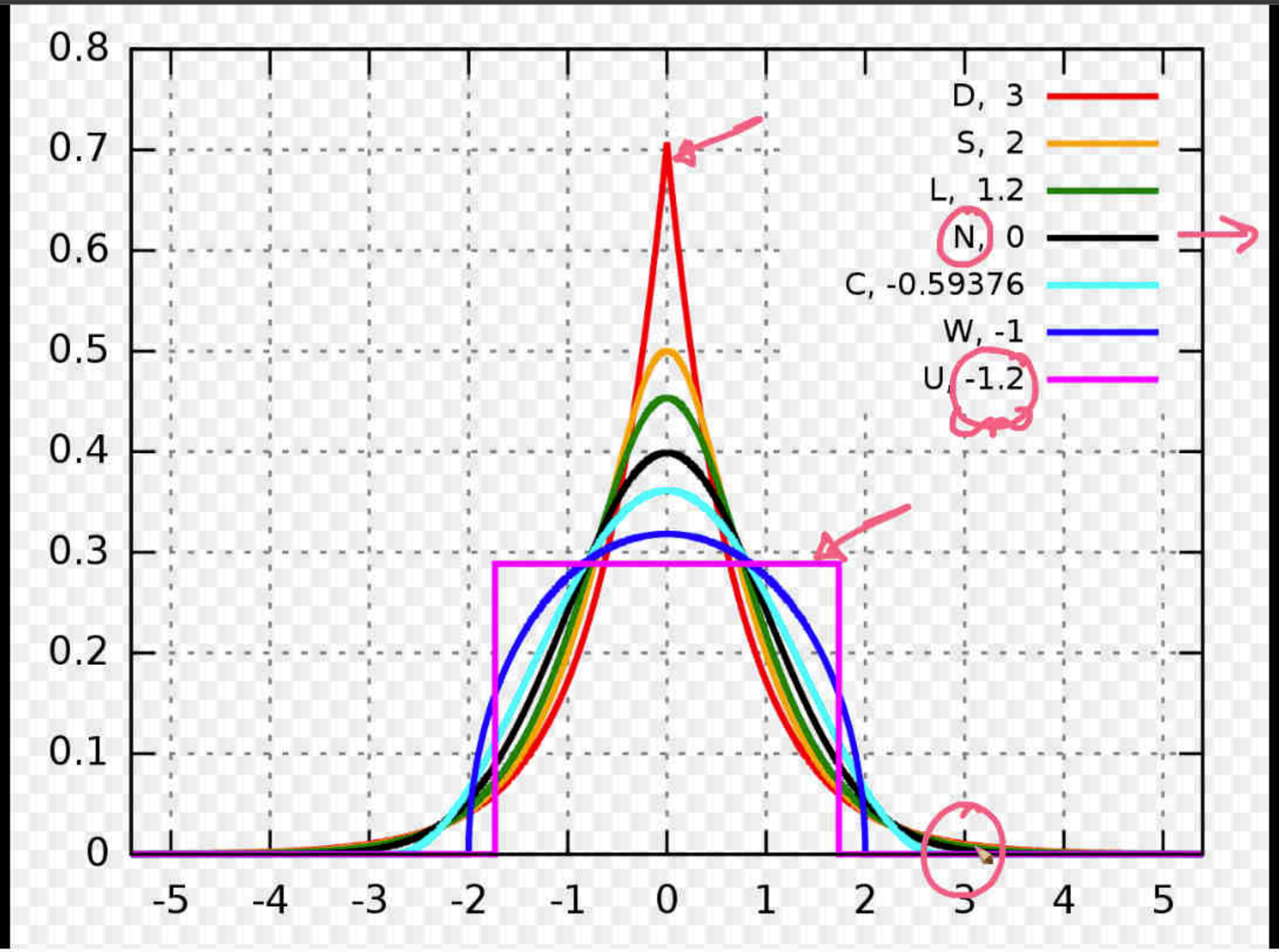
how tailed is your dist

Gaussian

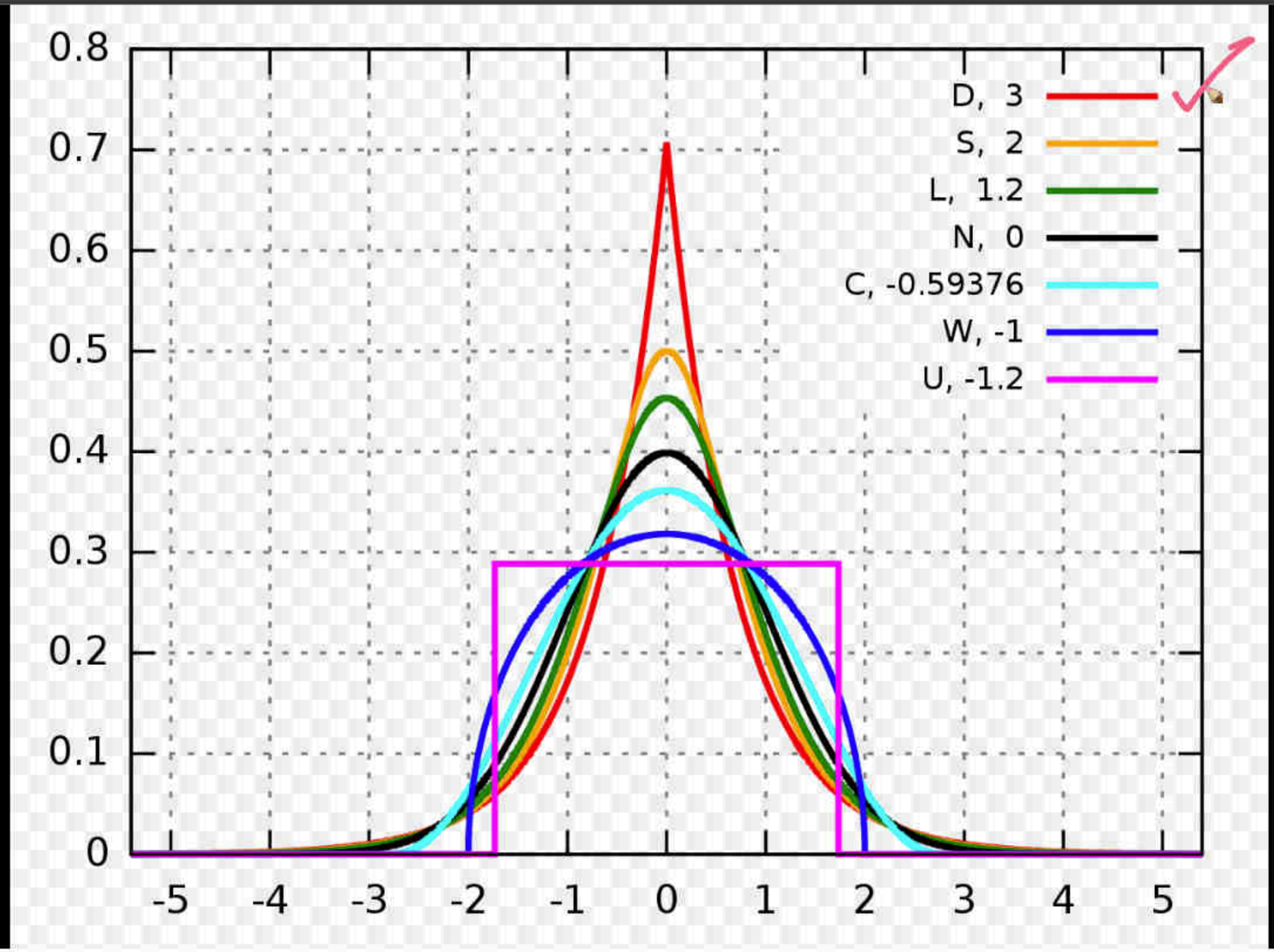
"3"

Excess Kurtosis

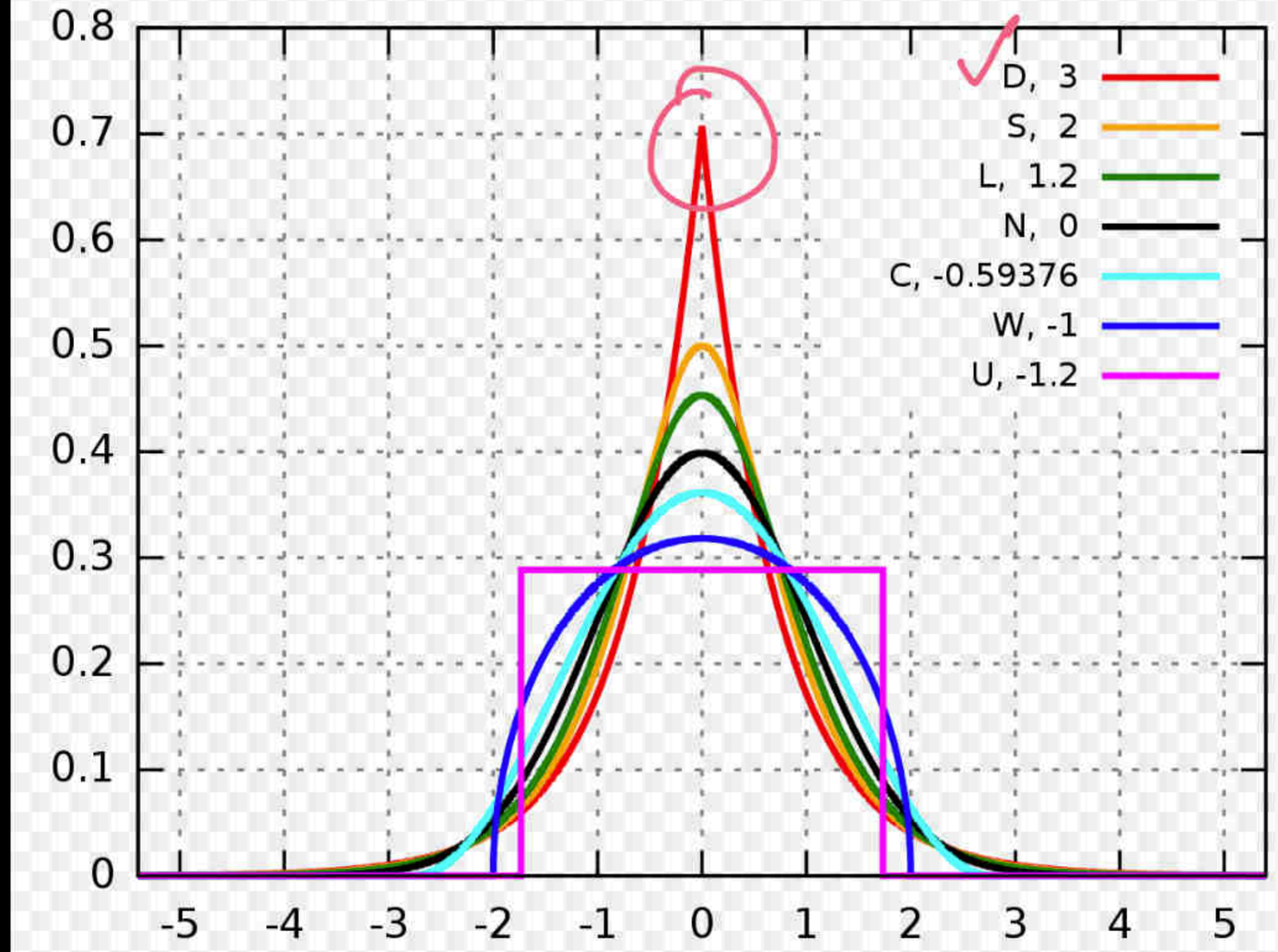
= K - 3



Probability density functions for selected distributions with mean 0, variance 1 and different excess kurtosis



Probability density functions for selected distributions with mean 0, variance 1 and different excess kurtosis



Probability density functions for selected distributions with mean 0, variance 1 and different excess kurtosis

$X: x_1\ x_2 \cdots x_n$

$E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$

Pearson moments [edit]

The kurtosis is the fourth standardized moment, defined as

$$\operatorname{Kurt}[X] = \operatorname{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\operatorname{E}[(X-\mu)^4]}{(\operatorname{E}[(X-\mu)^2])^2} = \frac{\mu_4}{\sigma^4},$$

where μ_4 is the fourth central moment and σ is the standard deviation. Several letters are used in the literature to denote the kurtosis. A very common choice is κ, which is fine as long as it is clear that it does not refer to a cumulant. Other choices include γ_2, to be similar to the notation for skewness, although sometimes this is instead reserved for the excess kurtosis.

The kurtosis is bounded below by the squared skewness plus 1:[4]:432

$$\frac{\mu_4}{\sigma^4} \geq \left(\frac{\mu_3}{\sigma^3}\right)^2 + 1,$$

where μ_3 is the third central moment. The lower bound is realized by the Bernoulli distribution. There is no upper limit to the kurtosis of a general probability distribution, and it may be infinite.

A reason why some authors favor the excess kurtosis is that cumulants are extensive. Formulas related to the extensive property are more naturally expressed in terms of the excess kurtosis. For example, let X_1, ..., X_n be independent random variables for which the fourth m[illegible]dom variable defined by the sum of the X_i. The excess

Pearson moments [edit]

The kurtosis is the fourth standardized moment, defined as

$$\operatorname{Kurt}[X] = \operatorname{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\operatorname{E}[(X-\mu)^4]}{(\operatorname{E}[(X-\mu)^2])^2} = \frac{\mu_4}{\sigma^4},$$

where μ_4 is the fourth central moment and σ is the standard deviation. Several letters are used in the literature to denote the kurtosis. A very common choice is κ, which is fine as long as it is clear that it does not refer to a cumulant. Other choices include γ_2, to be similar to the notation for skewness, although sometimes this is instead reserved for the excess kurtosis.

The kurtosis is bounded below by the squared skewness plus 1:[4]:432

$$\frac{\mu_4}{\sigma^4} \geq \left(\frac{\mu_3}{\sigma^3}\right)^2 + 1,$$

where μ_3 is the third central moment. The lower bound is realized by the Bernoulli distribution. There is no upper limit to the kurtosis of a general probability distribution, and it may be infinite.

A reason why some authors favor the excess kurtosis is that cumulants are extensive. Formulas related to the extensive property are more naturally expressed in terms of the excess kurtosis. For example, let X_1, ..., X_n be independent random variables for which the fourth m[illegible]dom variable defined by the sum of the X_i. The excess kurtosis of X_i is

$X: x_1\ x_2\ \cdots\ x_n$

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i-\mu}{s}\right)^4 = \text{Kurt}$$

$$Ek = \text{Kurt} - 3$$

Pearson moments [edit]

The kurtosis is the fourth standardized moment, defined as

$$\operatorname{Kurt}[X] = \operatorname{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\operatorname{E}[(X-\mu)^4]}{(\operatorname{E}[(X-\mu)^2])^2} = \frac{\mu_4}{\sigma^4},$$

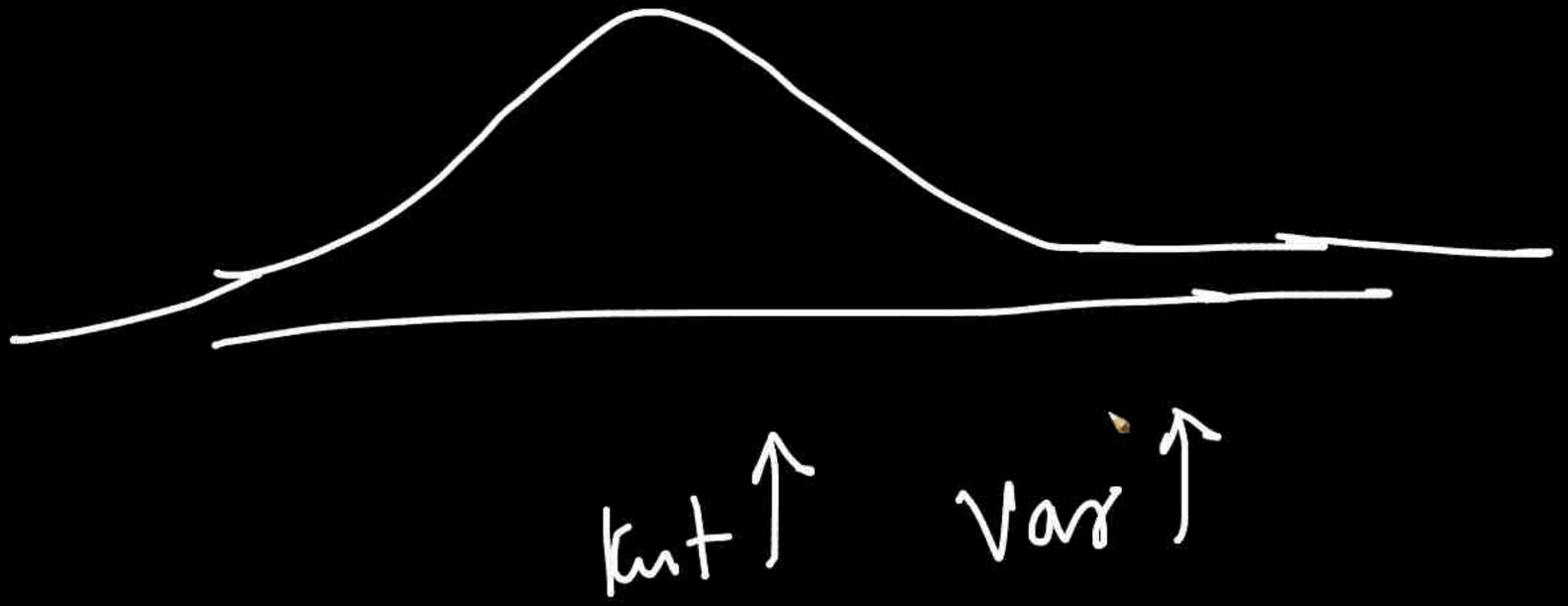
where μ_4 is the fourth central moment and σ is the standard deviation. Several letters are used in the literature to denote the kurtosis. A very common choice is κ, which is fine as long as it is clear that it does not refer to a cumulant. Other choices include γ_2, to be similar to the notation for skewness, although sometimes this is instead reserved for the excess kurtosis.

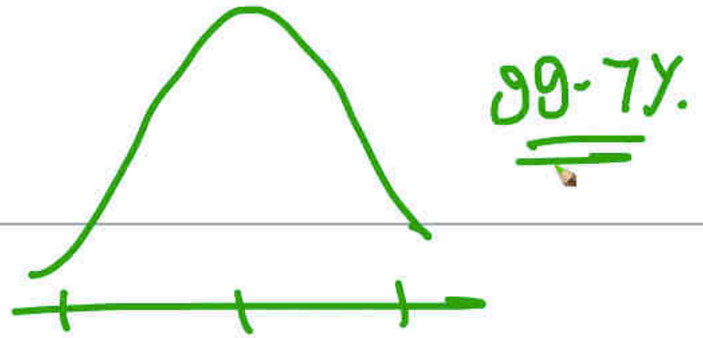
The kurtosis is bounded below by the squared skewness plus 1:[4]:432

$$\frac{\mu_4}{\sigma^4} \geq \left(\frac{\mu_3}{\sigma^3}\right)^2 + 1,$$

where μ_3 is the third central moment. The lower bound is realized by the Bernoulli distribution. There is no upper limit to the kurtosis of a general probability distribution, and it may be infinite.

A reason why some authors favor the excess kurtosis is that cumulants are extensive. Formulas related to the extensive property are more naturally expressed in terms of the excess kurtosis. For example, let X_1, ..., X_n be independent random variables for whic[illegible]e defined by the sum of the X_i. The excess





Pearson moments [edit]

The kurtosis is the fourth standardized moment, defined as

$$\operatorname{Kurt}[X] = \operatorname{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\operatorname{E}[(X-\mu)^4]}{(\operatorname{E}[(X-\mu)^2])^2} = \frac{\mu_4}{\sigma^4},$$

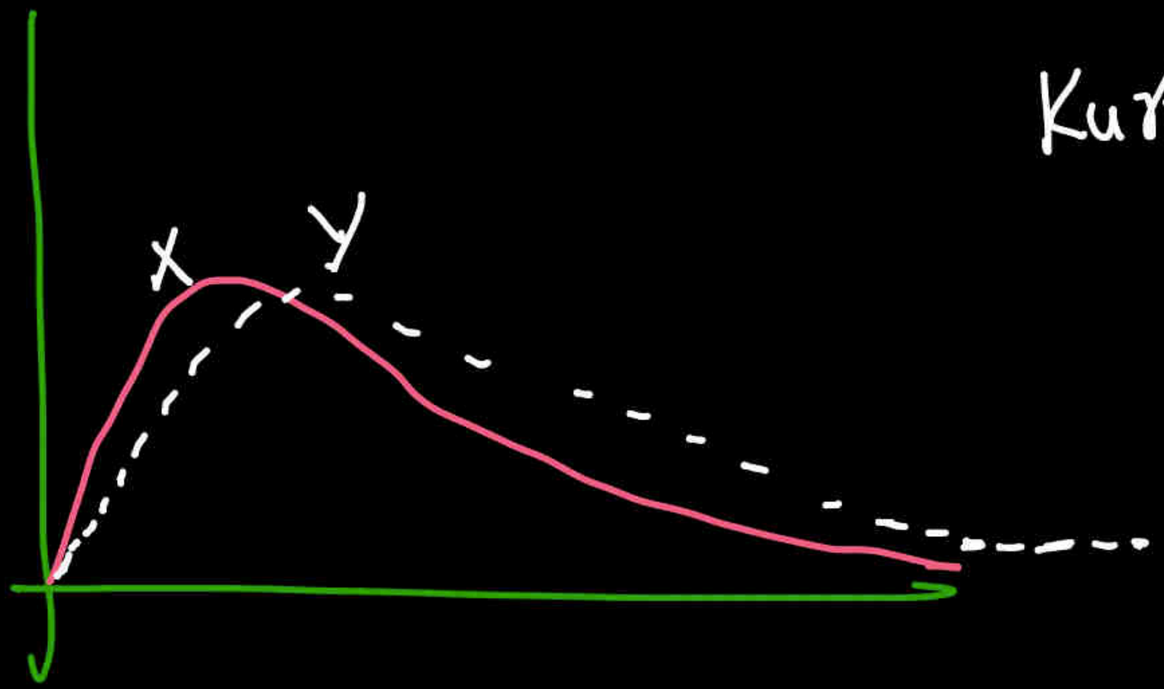
where μ_4 is the fourth central moment and σ is the standard deviation. Several letters are used in the literature to denote the kurtosis. A very common choice is κ, which is fine as long as it is clear that it does not refer to a cumulant. Other choices include γ_2, to be similar to the notation for skewness, although sometimes this is instead reserved for the excess kurtosis.

The kurtosis is bounded below by the squared skewness plus 1:[4]:432

$$\frac{\mu_4}{\sigma^4} \geq \left(\frac{\mu_3}{\sigma^3}\right)^2 + 1,$$

where μ_3 is the third central moment. The lower bound is realized by the Bernoulli distribution. There is no upper limit to the kurtosis of a general probability distribution, and it may be infinite.

A reason why some authors favor the excess kurtosis is that cumulants are extensive. Formulas related to the extensive property are more naturally expressed in terms of the excess kurtosis. For example, let X_1, ..., X_n be independent random variables for which [illegible] defined by the sum of the X_i. The excess



$$Kurt(x) < Kurt(y)$$

Kunt

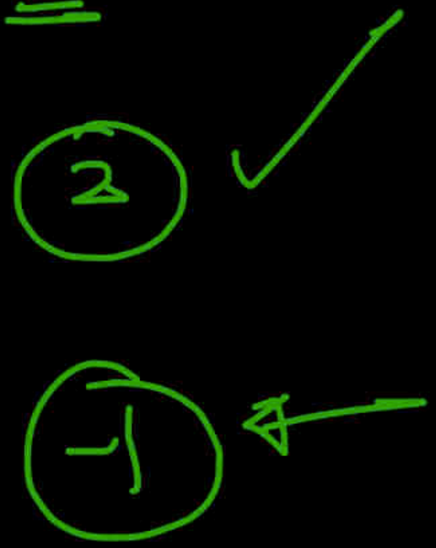
Normal → 3

X →

Y →

Ex Kunt = Kunt - 3

0



Q

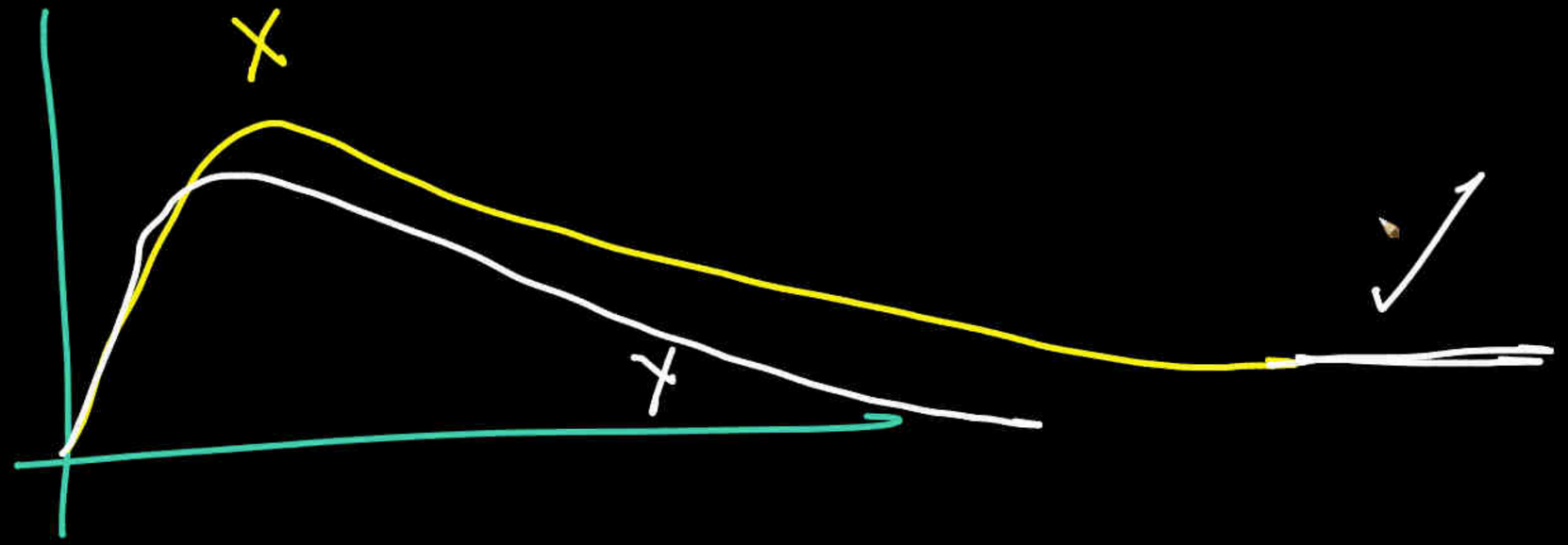
X

Y

why?

Exam: 3

2



Q

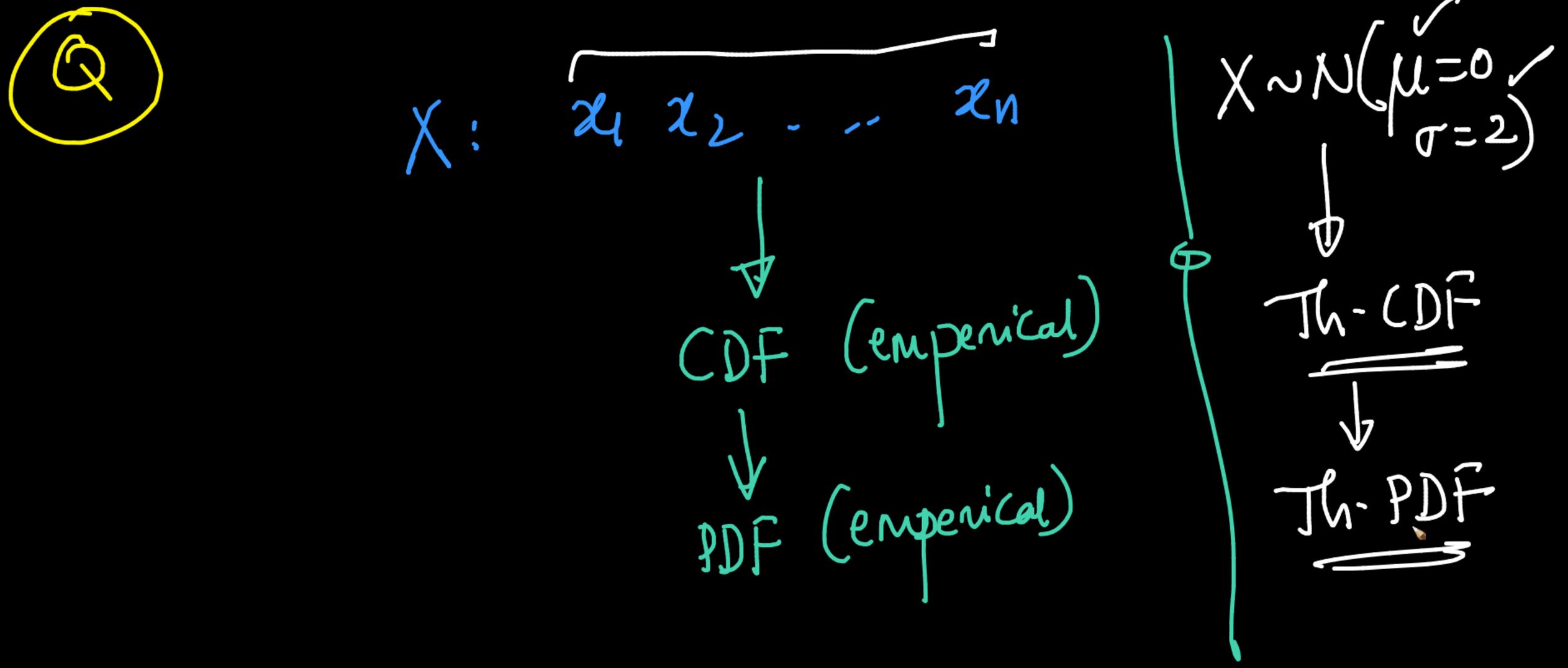
X : mean [lb ub] ← sample of size 'n'

95% C.I of pop-mean is [lb ub]

95% C.I

10K

Using sample sizes of 'n', if we repeatedly compute the the C.I there is a 95% prob of the μ lying in these C.I's


Q
X:
x_1 x_2 . . .. x_n
CDF (empirical)
PDF (empirical)
$X \sim N(\mu=0, \sigma=2)$
Th- CDF
Th- PDF

$n = 40$

SD

$n = 90$

$\sigma \uparrow$

$n \uparrow$

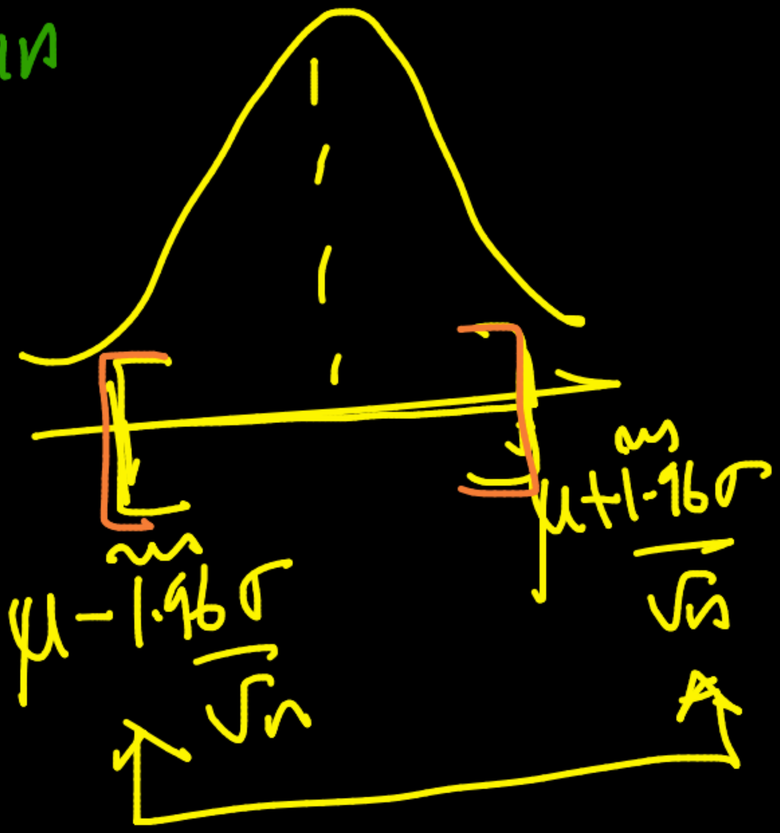
$\sqrt{n} \uparrow$

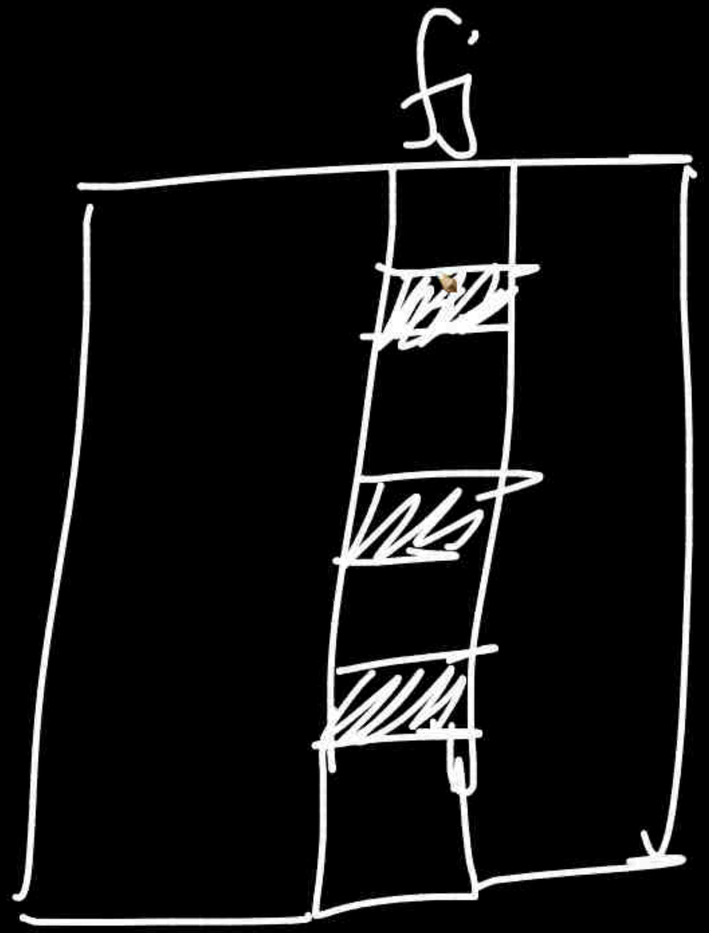
$\frac{1}{\sqrt{n}} \downarrow$

CLT

$\bar{X} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$

$\bar{X}$ ↑ Sample-mean

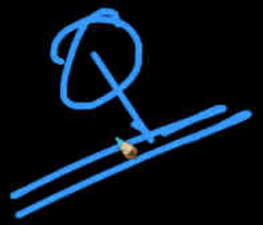



fi

Q

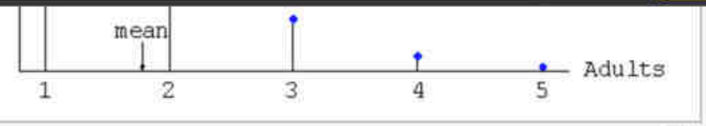
t-dist (?)

$N(0,1)$ = z-dist



Kurtosis $= E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$

Definition [edit]



Distribution of adult residents across US households

Fisher's moment coefficient of skewness [edit]

The skewness of a random variable X is the third standardized moment $\tilde{\mu}_3$, defined as:[4][5]

$$\tilde{\mu}_3 = \mathrm{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{\mu_3}{\sigma^3} = \frac{\mathrm{E}\left[(X-\mu)^3\right]}{(\mathrm{E}[(X-\mu)^2])^{3/2}} = \frac{\kappa_3}{\kappa_2^{3/2}}$$

where μ is the mean, σ is the standard deviation, E is the expectation operator, μ_3 is the third central moment, and κ_t are the t-th cumulants. It is sometimes referred to as **Pearson's moment coefficient of skewness**,[5] or simply the **moment coefficient of skewness**,[4] but should not be confused with Pearson's other skewness statistics (see below). The last equality expresses skewness in terms of the ratio of the third cumulant κ_3 to the 1.5th power of the second cumulant κ_2. This is analogous to the definition of kurtosis as the fourth cumulant normalized by the square of the second cumulant. The skewness is also sometimes denoted Skew[X].

If σ is finite, μ is finite too and skewness can be expressed in terms of the non-central moment E[X^3] by expanding the previous formula,

$$\begin{aligned}\tilde{\mu}_3 &= \mathrm{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] \\ &= \frac{\mathrm{E}[X^3] - 3\mu\,\mathrm{E}[X^2] + 3\mu^2\,\mathrm{E}[X] - \mu^3}{}\end{aligned}$$

Definition [edit]

Distribution of adult residents across US households

Fisher's moment coefficient of skewness [edit]

The skewness of a random variable X is the third standardized moment $\tilde{\mu}_3$, defined as:[4][5]

$$\tilde{\mu}_3 = \mathrm{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{\mu_3}{\sigma^3} = \frac{\mathrm{E}\left[(X-\mu)^3\right]}{(\mathrm{E}[(X-\mu)^2])^{3/2}} = \frac{\kappa_3}{\kappa_2^{3/2}}$$

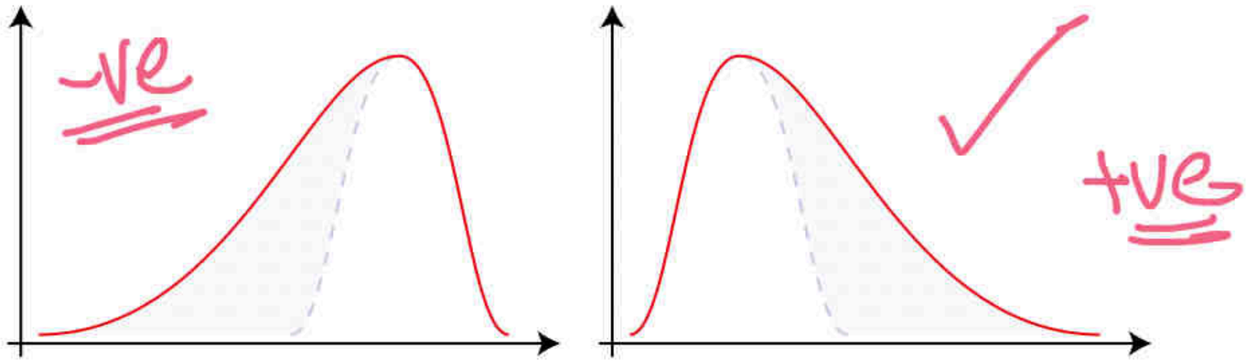
where μ is the mean, σ is the standard deviation, E is the expectation operator, μ_3 is the third central moment, and κ_t are the t-th cumulants. It is sometimes referred to as **Pearson's moment coefficient of skewness**,[5] or simply the **moment coefficient of skewness**,[4] but should not be confused with Pearson's other skewness statistics (see below). The last equality expresses skewness in terms of the ratio of the third cumulant κ_3 to the 1.5th power of the second cumulant κ_2. This is analogous to the definition of kurtosis as the fourth cumulant normalized by the square of the second cumulant. The skewness is also sometimes denoted Skew[X].

If σ is finite, μ is finite too and skewness can be expressed in terms of the non-central moment E[X^3] by expanding the previous formula,

$$\begin{aligned}\tilde{\mu}_3 &= \mathrm{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] \\ &= \frac{\mathrm{E}[X^3] - 3\mu\,\mathrm{E}[X^2] + 3\mu^2\,\mathrm{E}[X] - \mu^3}{}\end{aligned}$$

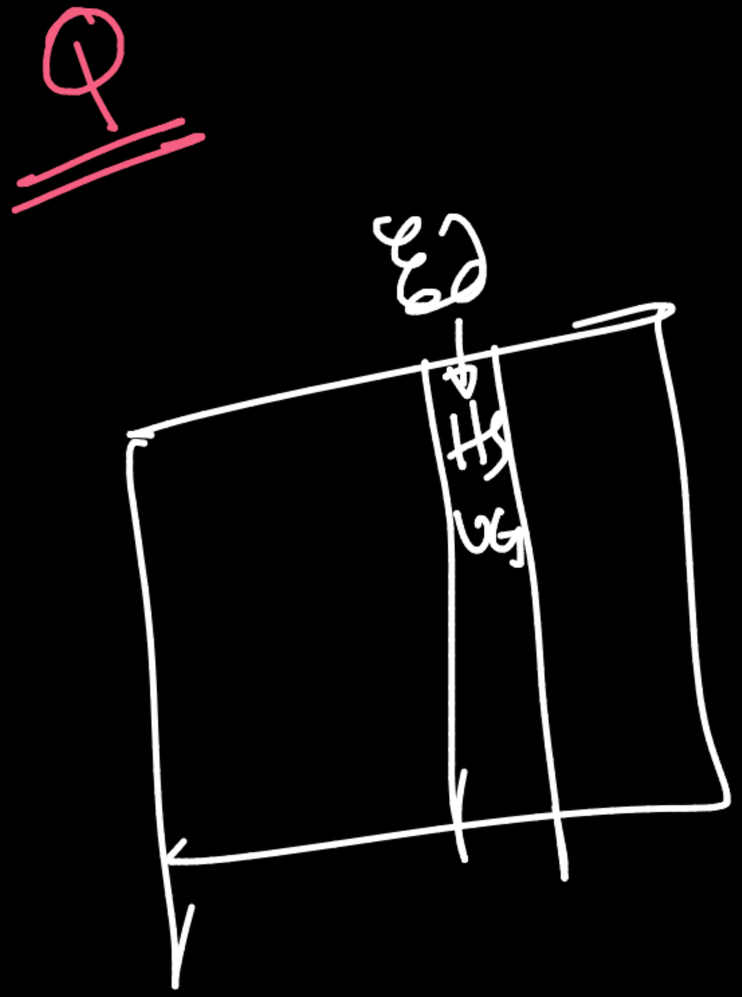
be skewed or leaning to the right; *left* instead refers to the left tail being drawn out and, often, the mean being skewed to the left of a typical center of the data. A left-skewed distribution usually appears as a *right-leaning* curve.[1]

2. *positive skew*: The right tail is longer; the mass of the distribution is concentrated on the left of the figure. The distribution is said to be *right-skewed*, *right-tailed*, or *skewed to the right*, despite the fact that the curve itself appears to be skewed or leaning to the left; *right* instead refers to the right tail being drawn out and, often, the mean being skewed to the right of a typical center of the data. A right-skewed distribution usually appears as a *left-leaning* curve.[1]



Negative Skew Positive Skew

Skewness in a data series may sometimes be observed not only graphically but by simple inspection of the values. For instance, consider the numeric sequence (49, 50, 51), whose values are evenly distributed around a central value of 50. We can transform this sequence into a negatively skewed distribution by adding a value far below the mean, which is probably a negative outlier, e.g. (40, 49, 50, 51). Therefore, the mean of the sequence becomes 47.5, and the median is 49.5. Based on the formula of nonparametric skew, defined as $(\mu - \nu)/\sigma$, the skew is negative. Similarly, we can make the sequence pos n is probably a positive outlier, e.g. (49, 50,



Ed: HS, UG, PG, Phd

1 → HS

2 → UG

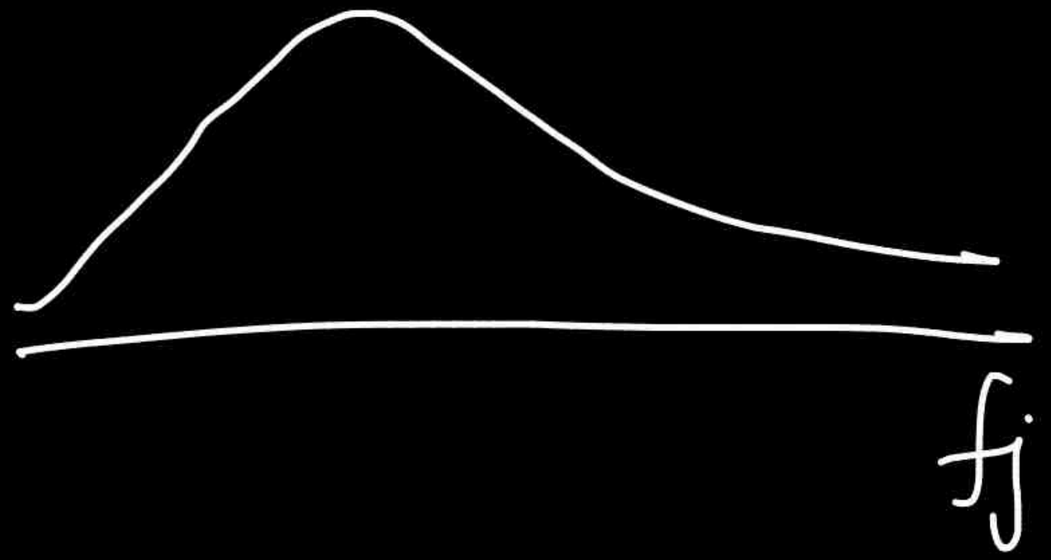
3 → PG

4 → Phd

5 → HS

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0


fj


fj

$$E(X) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$x_1 \; x_2 \; \cdots \; x_n$

PMF: $p(X = k) \quad \forall k$

$$E(X) = \sum_{k} k \cdot p(X = k)$$

$$\hookrightarrow \int_{K} k \cdot p(k) \, dk$$

Math 20 – Inequalities of Markov and Chebyshev

Often, given a random variable X whose distribution is unknown but whose expected value μ *is* known, we may want to ask how likely it is for X to be 'far' from μ, or how likely it is for this random variable to be 'very large.' This would give us some idea of the spread of the distribution, though perhaps not a complete picture.

Proposition 1 (Markov's Inequality)**.** *Let X be a random variable that takes only nonnegative values. Then for any positive real number a,*

$$P(X \geq a) \leq \frac{E(X)}{a}$$

provided $E(X)$ exists.

For example, Markov's inequality tells us that as long as X doesn't take negative values, the probabili[illegible] which we can see by setting a = [illegible]iable is at least k times its expected value is at most $\frac{1}{k}$. Notice that the only things we assumed about this random variable are that it can't be negative and has finite mean; we don't need to know