Last class
Probstat

Feature Engineering & Data Analysis - 2
→ covey code/concepts
→ After-class

→ No class on Saturday ←

Next week: MWF, TTS → SQL

Problem: Loan-Status — Yes / NO

Recap:

- Data Analysis
- Feature engineered:- (prev-class)
  - Total-household-Income
  - { Income + Co-appl-Income

designing new features which are useful

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:08

EDA_FE.ipynb - Colaboratory

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=f2aROZfGMz9t

+ Code + Text RAM Disk

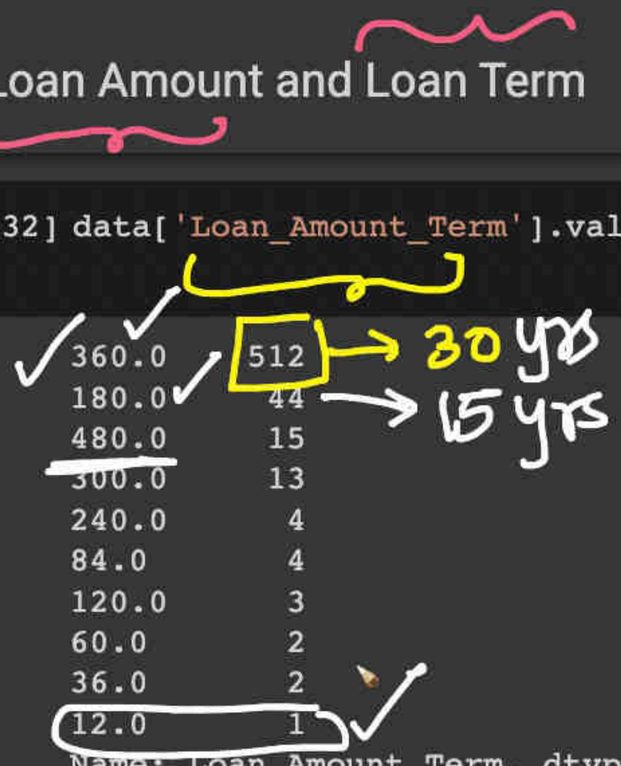
Incomes

[ ] ↳ 11 cells hidden

Loan Amount and Loan Term

[232] data['Loan_Amount_Term'].value_counts()

```
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```



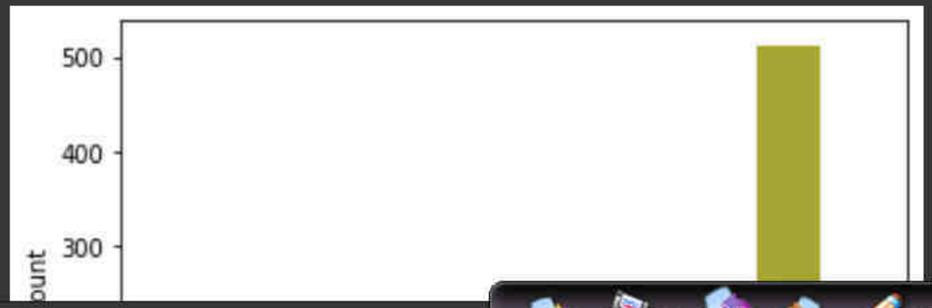
[233] data['Loan_Amount_Term'] = (data['Loan_Amount_Term']/12).astype('float')

3 / 4

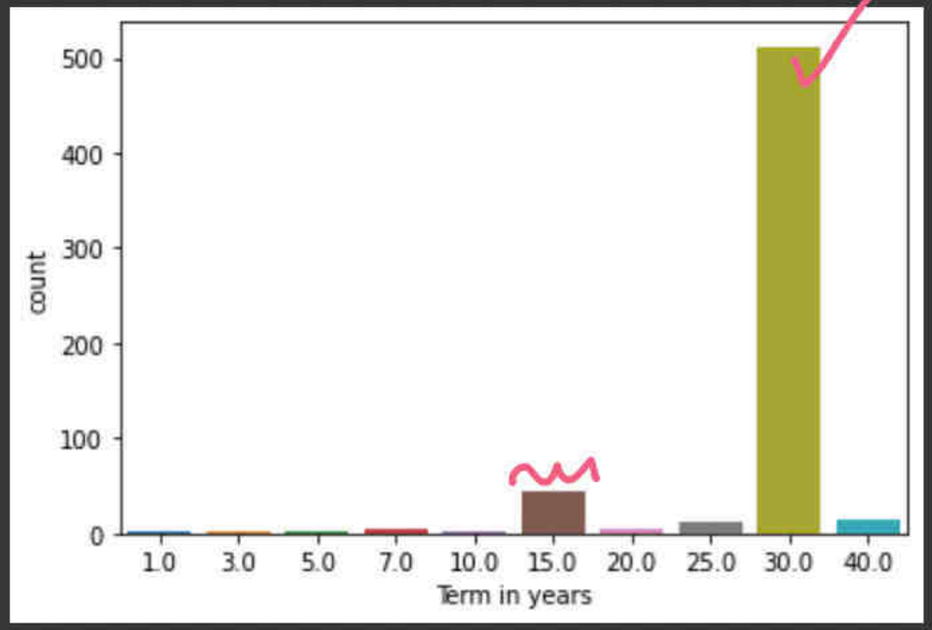
```
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

```
[233] data['Loan_Amount_Term'] = (data['Loan_Amount_Term']/12).astype('float')
```

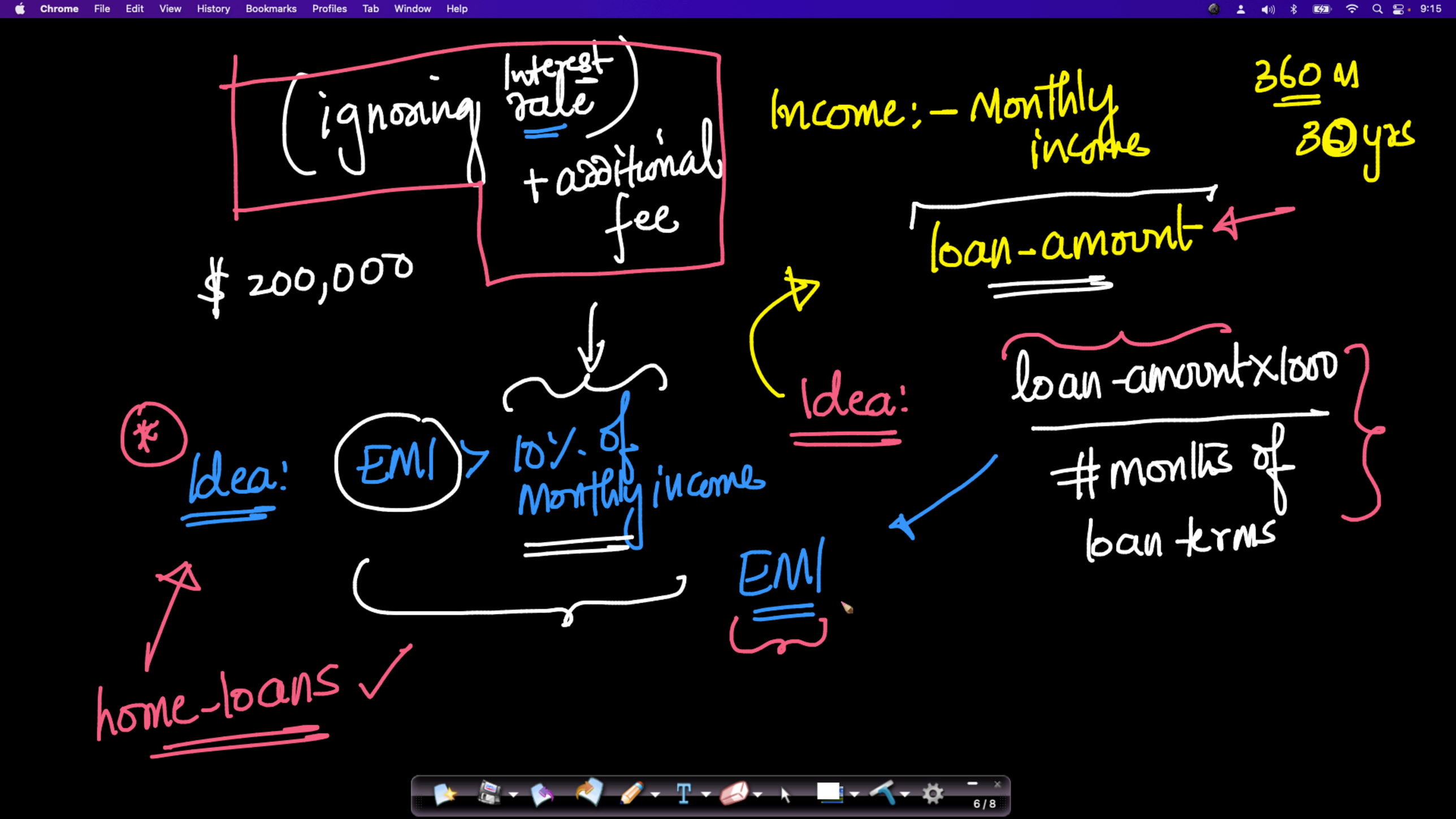
```
sns.countplot(x='Loan_Amount_Term', data=data)
plt.xlabel("Term in years")
plt.show()
# Observation: We can clearly see that more than 90% of the loans were applied for 30 years.
```



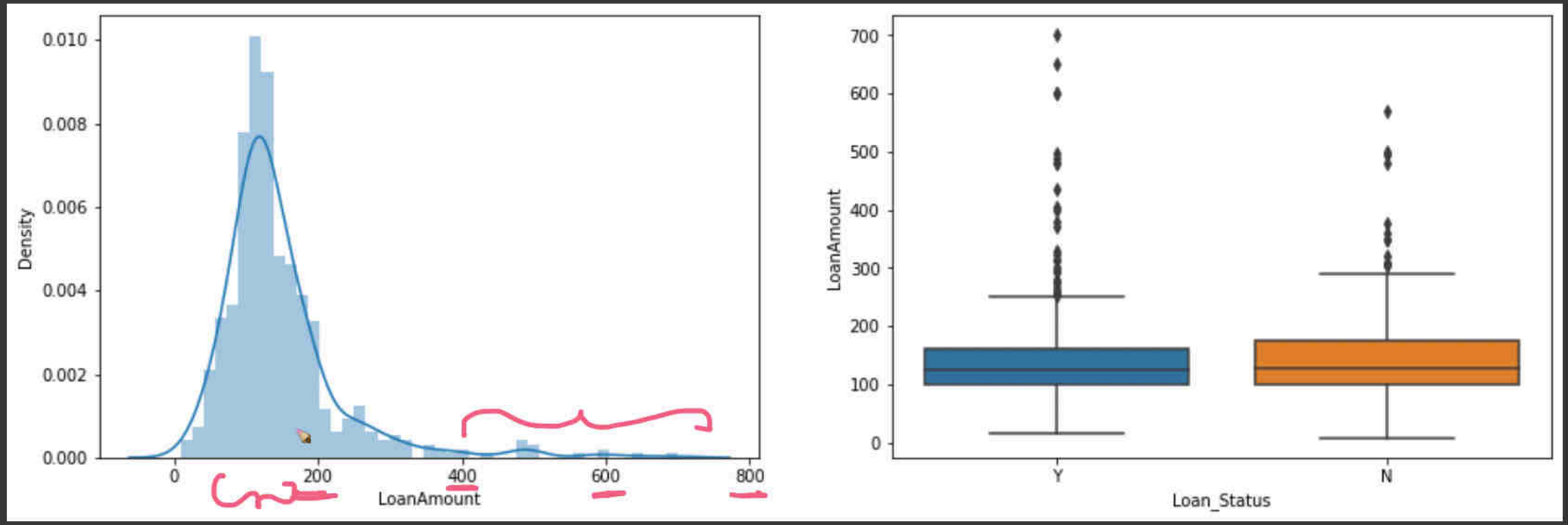
```
[234] sns.countplot(x='Loan_Amount_Term', data=data)
      plt.xlabel("Term in years")
      plt.show()
      # Observation: We can clearly see that more than 90% of the loans were applied for 30 years.
```




```
[235] ### Distribution of "LoanAmount" variable :
```

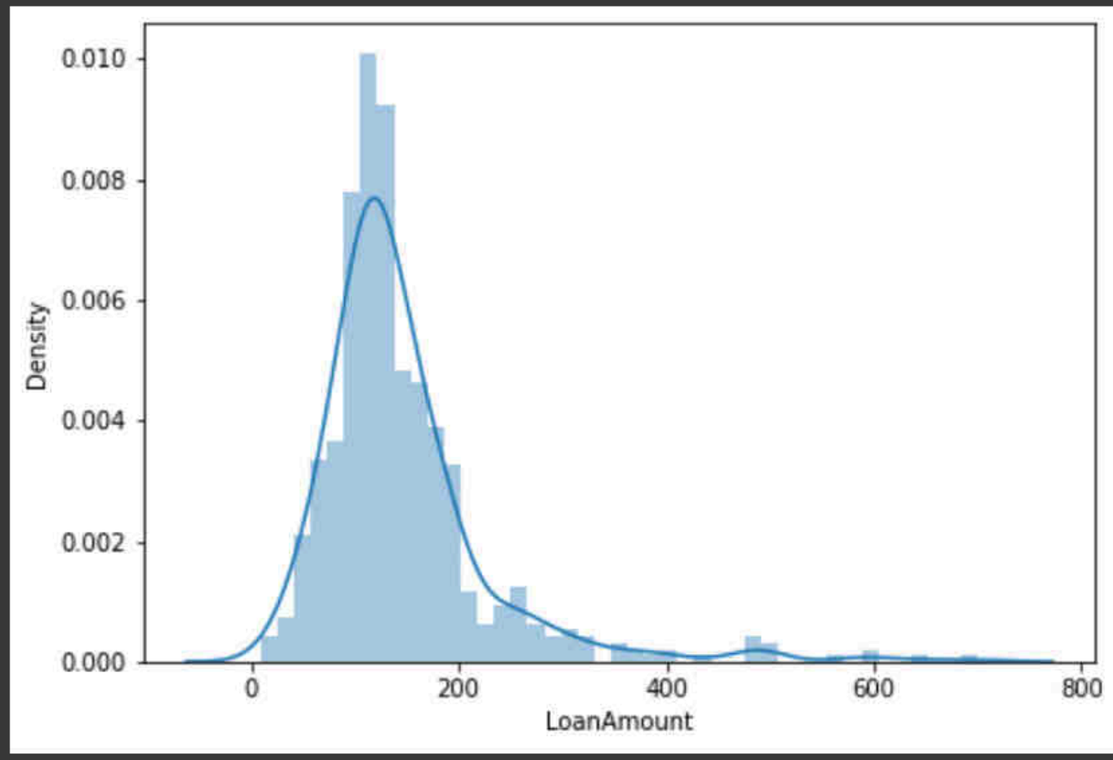

(ignoring Interest rate) + additional fee
$ 200,000
Income :- Monthly income
360 M
30 yrs
loan-amount
Idea:
loan-amount x 1000
months of loan terms
EMI
Idea:
EMI > 10% of Monthly income
EMI
home-loans ✓
6 / 8

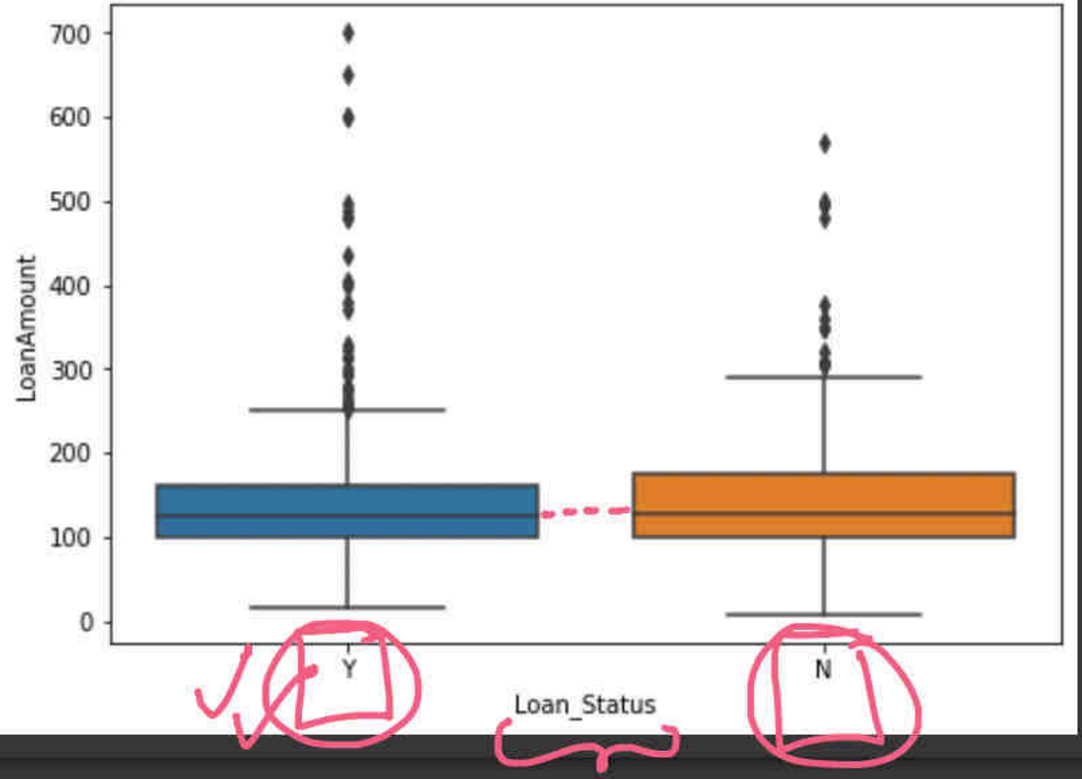
```
plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = 'LoanAmount')

plt.show()
```



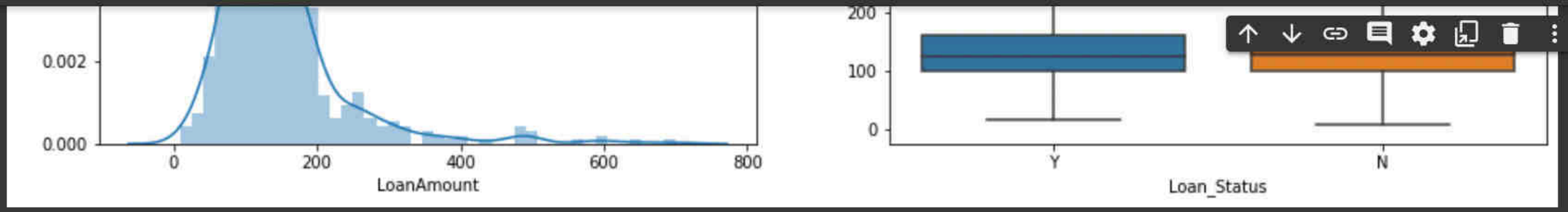

```
[236] # Approximate calc: ignoring interest rates as we dont know that.
      data['Loan_Amount_per_year'] = data['LoanAmount']/data['Loan_Amount_Term']
```

```
plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = 'LoanAmount')

plt.show()
```





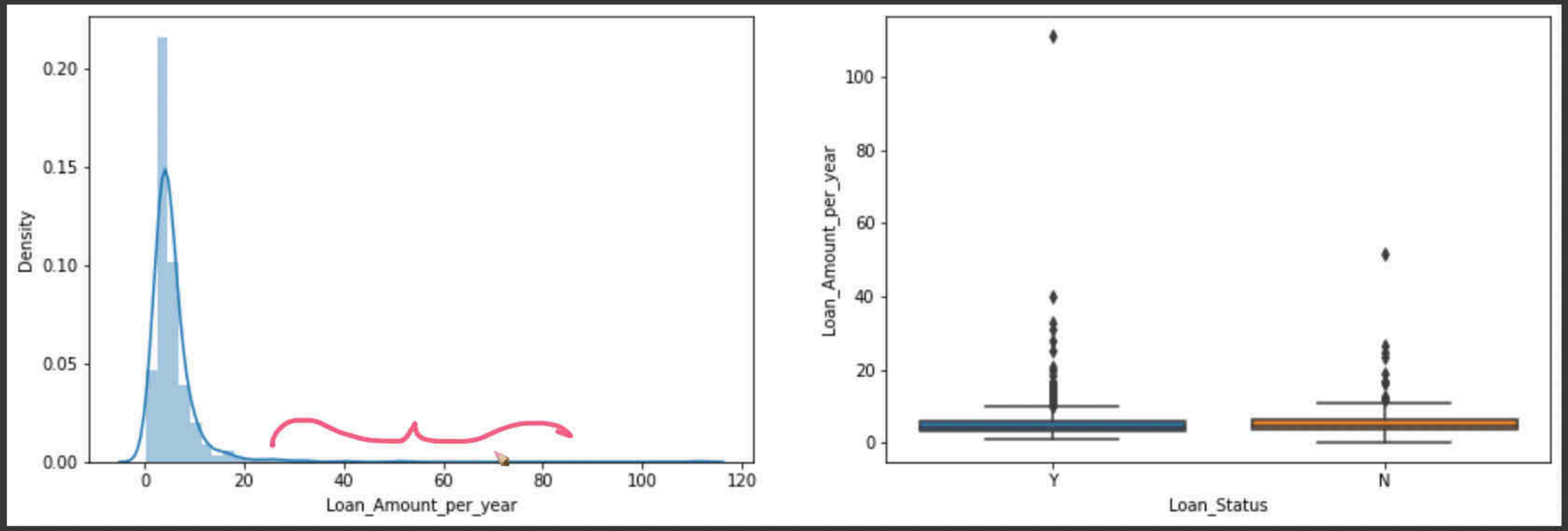

```
[236] # Approximate calc: ignoring interest rates as we dont know that.
      data['Loan_Amount_per_year'] = data['LoanAmount']/data['Loan_Amount_Term']
```



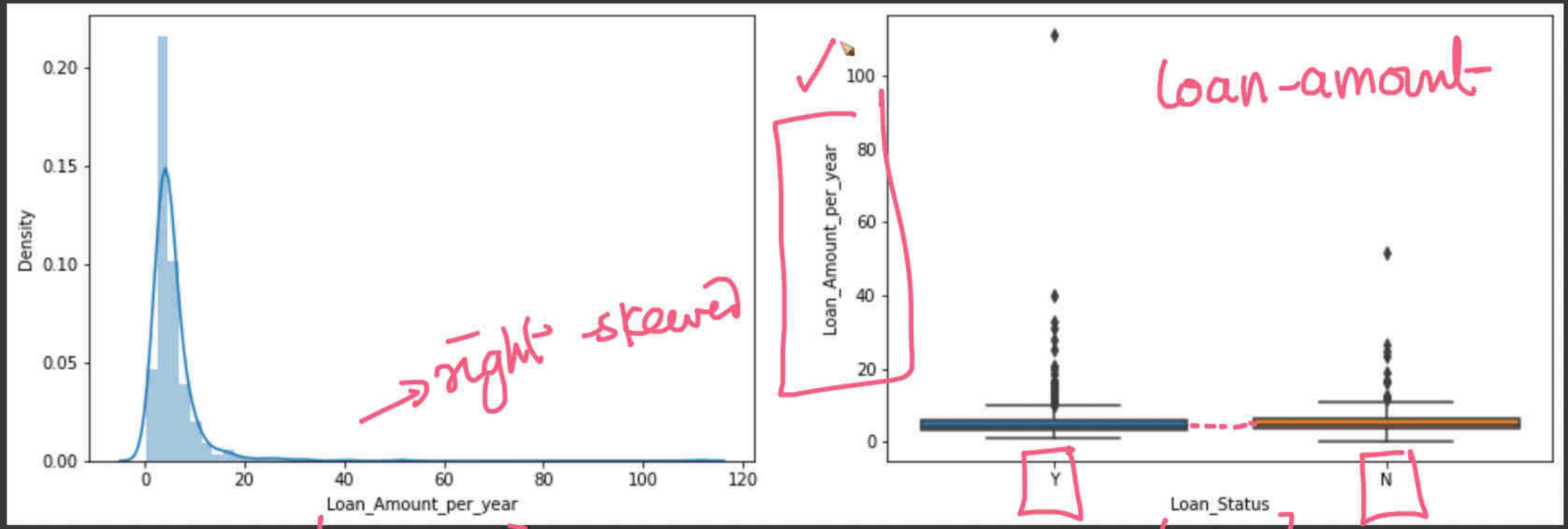
[236]
```
# Approximate calc: ignoring interest rates as we dont know that.
data['Loan_Amount_per_year']  = data['LoanAmount']/data['Loan_Amount_Term']
```

```
plt.figure(figsize=(16,5))
plt.subplot(121)
sns.distplot(data['Loan_Amount_per_year']);

plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = 'Loan_Amount_per_year')

plt.show()
```

```
plt.subplot(121)
sns.distplot(data['Loan_Amount_per_year']);

plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = 'Loan_Amount_per_year')

plt.show()
```

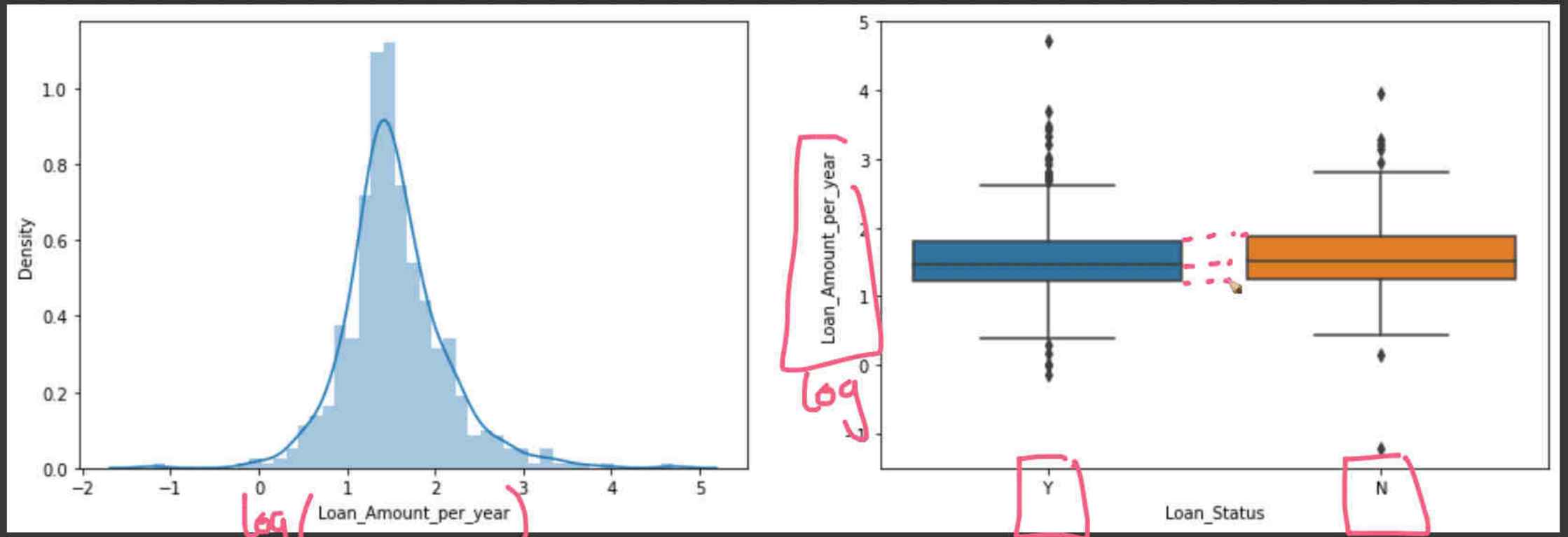


```
sns.boxplot(data=data, x= 'Loan_Status' , y = 'Loan_Amount_per_year' )

plt.show()
```

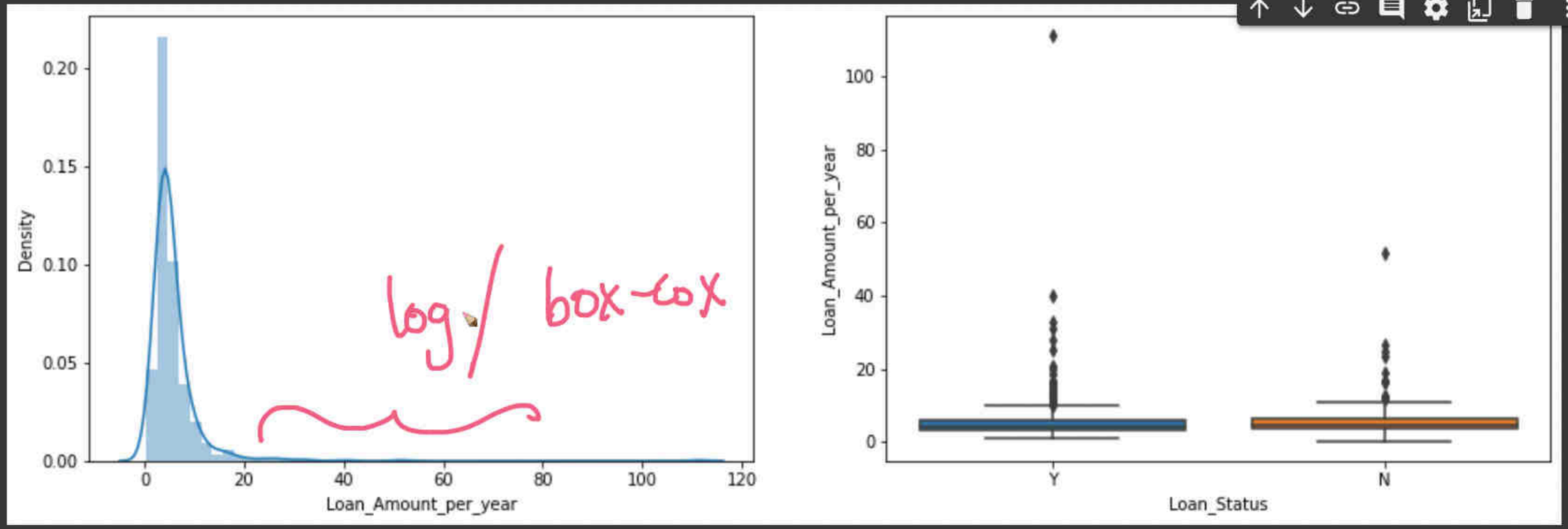



```
[238] # log transform
      plt.figure(figsize=(16,5))
      plt.subplot(121)
      log_loanAmount = np.log(data['Loan Amount per year'])
```

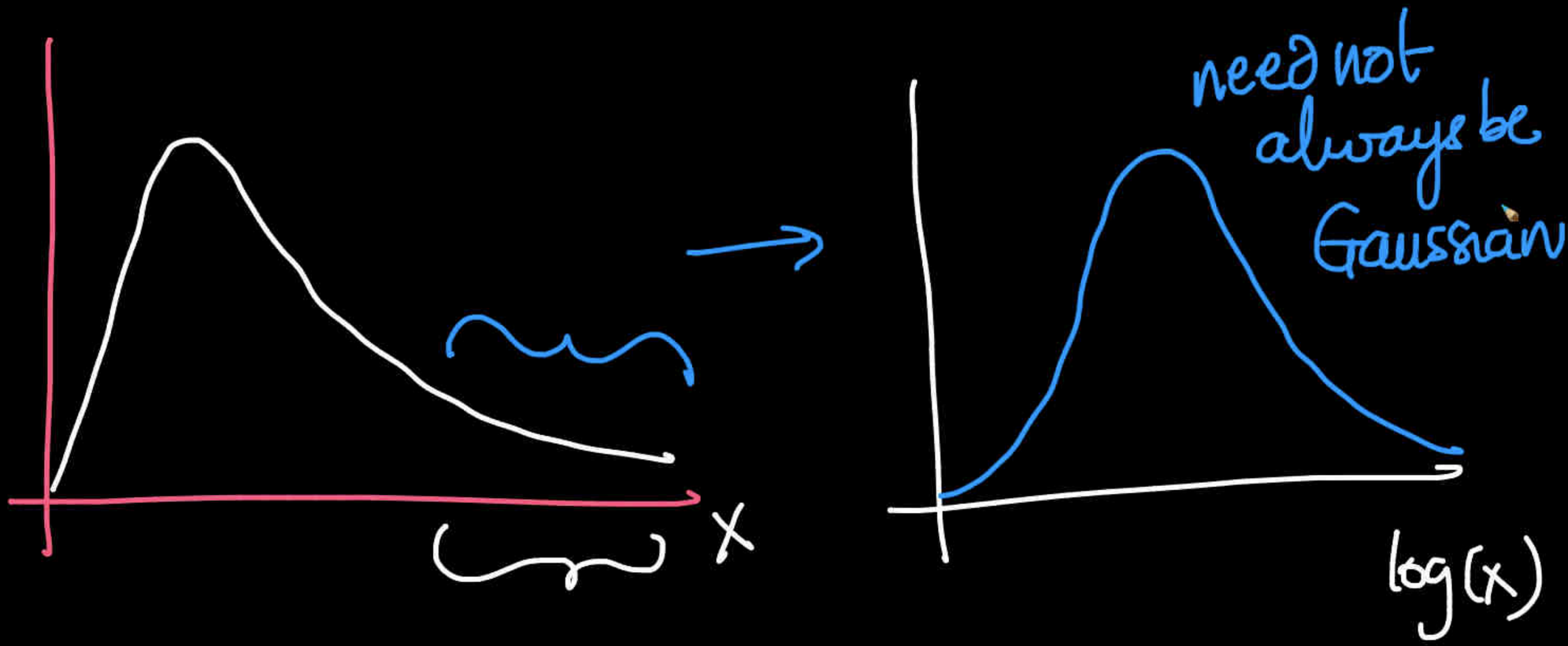
```
    plt.subplot(122)
    sns.boxplot(data=data, x='Loan_Status', y = log_loanAmount)

    plt.show()
```




```
[239] # Feature : Calculate the EMI based on the Loan Amount Per year.
      data['EMI'] = data['Loan_Amount_per_year']*1000/12
```




```
[238] # log transform
      plt.figure(figsize=(16,5))
      plt.subplot(121)
      log_loanAmount = np.log(data['Loan_Amount_per_year'])
      sns.distplot(log_loanAmount)

      plt.subplot(122)
```


need not always be Gaussian
X
log(x)

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

EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

google.com/search?q=plot+log(x)&rlz=1C5CHFA_enIN958IN958&oq=plot+log(x)&aqs=chrome.0.69i59j0i7i30l3j0i30l3j0i5i7i30j0i390l2.3449j0j7&sourceid=chrome&ie=UTF-8

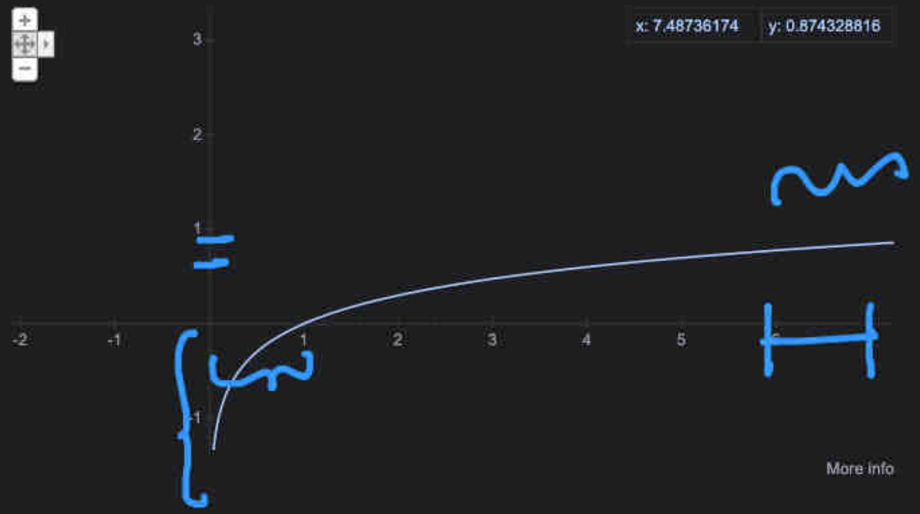
Google

plot log(x)

All Images News Videos Shopping More Tools

About 4,86,00,00,000 results (0.49 seconds)

Graph for log(x)



https://www.rapidtables.com › math › algebra › logarith...

logarithm graph | graph of log(x) - RapidTables.com

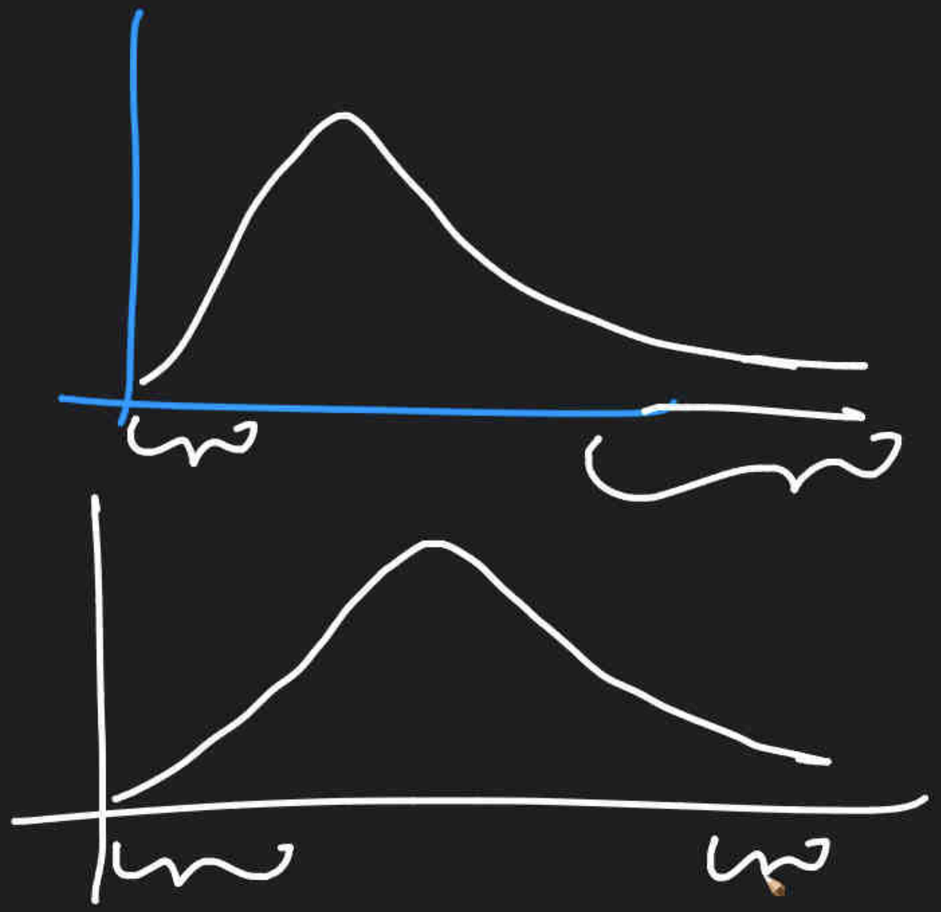
Graph of **log(x)**. **log(x)** function **graph**. Logarithm **graph**. y = f (x) = log10(x). **log(x) graph** properties. **log(x)** is defined for positive values of x. **log(x)** ...

Videos

Draw the graph of `y=-log(x)` when the graph of `y=logx` is ...

YouTube · Doubtnut
31-May-2020

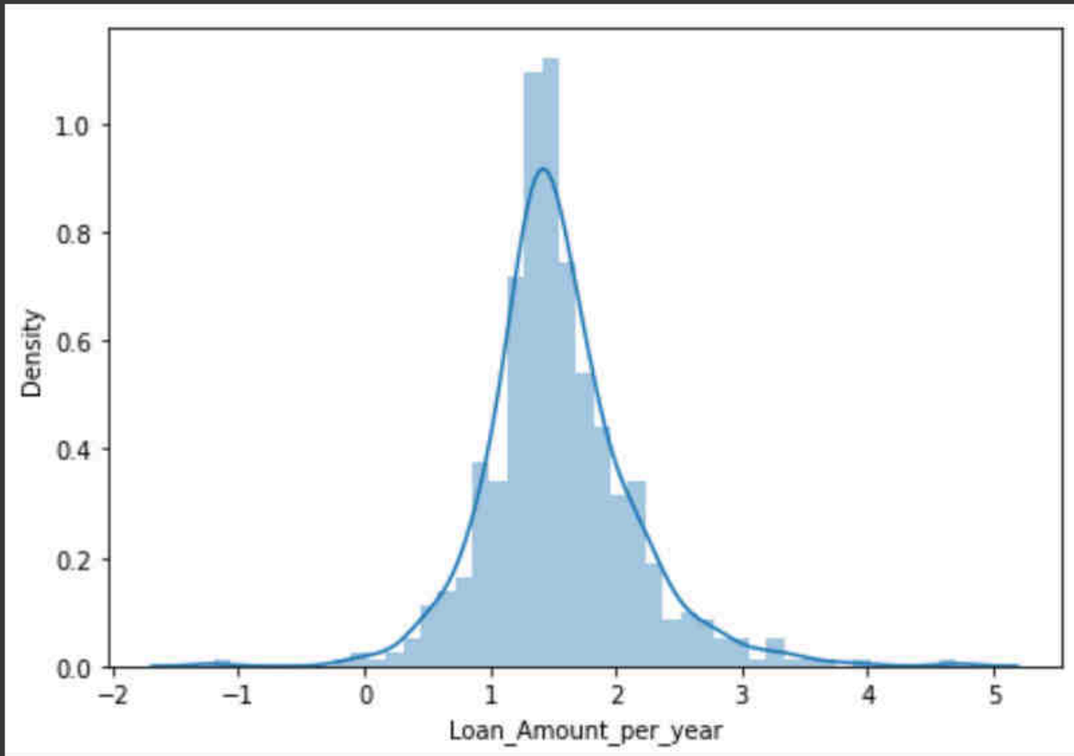
GRAPH OF LOG(X) | GR

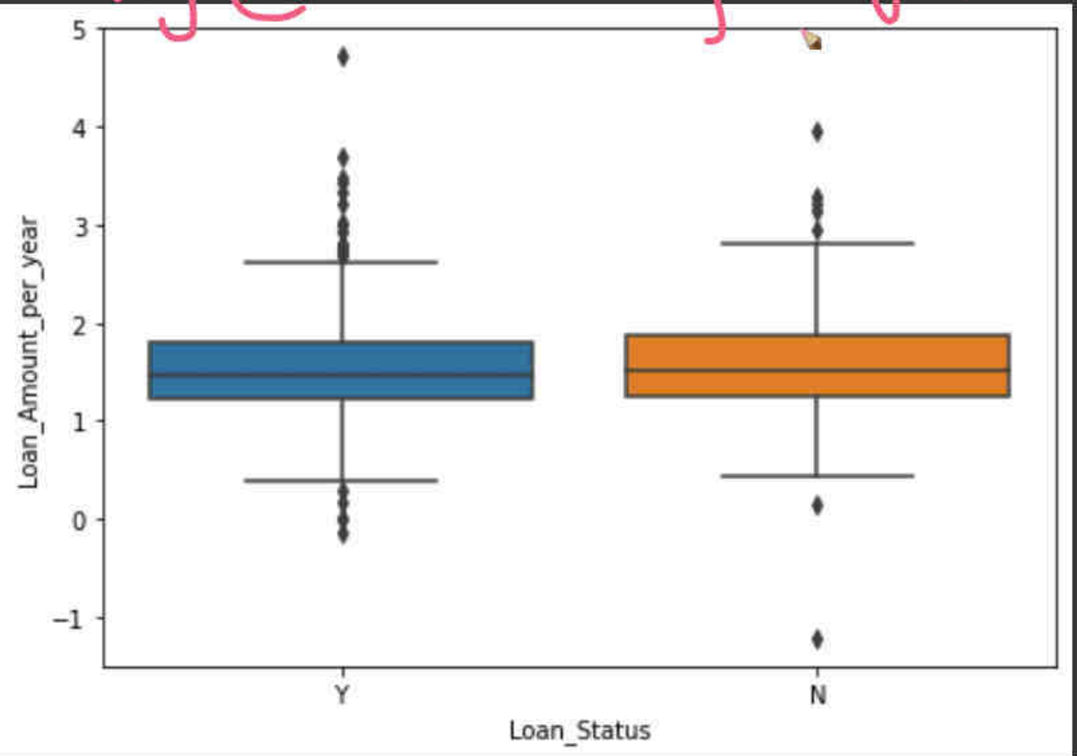


15 / 15

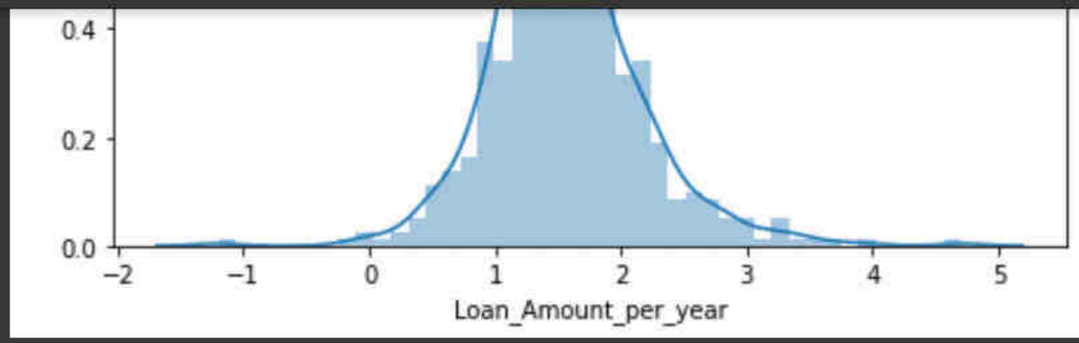
```
plt.subplot(122)
sns.boxplot(data=data, x='Loan_Status', y = log_loanAmount)

plt.show()
```

Loan-amount X
log (Loan-amount per-year)






```
[239] # Feature : Calculate the EMI based on the Loan Amount Per year.
      data['EMI'] = data['Loan
```




```
[239] # Feature : Calculate the EMI based on the Loan Amount Per year.
      data['EMI'] = data['Loan_Amount_per_year']*1000/12
```

200,000

appl+coapl

```
[240] #Feature : Able_to_pay_EMI
      data['Able_to_pay_EMI'] = (data['TotalIncome']*0.1 > data['EMI']).astype('int')
```

```
[241] sns.countplot(x='Able_to_pay_EMI', data = data, hue = 'Loan_Status')
      #Observation:
      ###There is 50% chance that you may get the loan approved if you cannot pay the EMI.
      ###But there, is a 72% chance that you may get the loan approved if you can pay the EMI.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eadc950>
```

```
sns.countplot(x='Able_to_pay_EMI', data = data, hue = 'Loan_Status')
#Observation:
###There is 50% chance that you may get the loan approved if you cannot pay the EMI.
###But there, is a 72% chance that you may get the loan approved if you can pay the EMI.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eadc950>
```

$\sim \frac{370}{500} =$

Can-pay-EMI

```
[ ]
```

```
sns.countplot(x='Able_to_pay_EMI', data = data, hue = 'Loan_Status')
#Observation:
###There is 50% chance that you may get the loan approved if you cannot pay the EMI.
###But there, is a 72% chance that you may get the loan approved if you can pay the EMI.
```

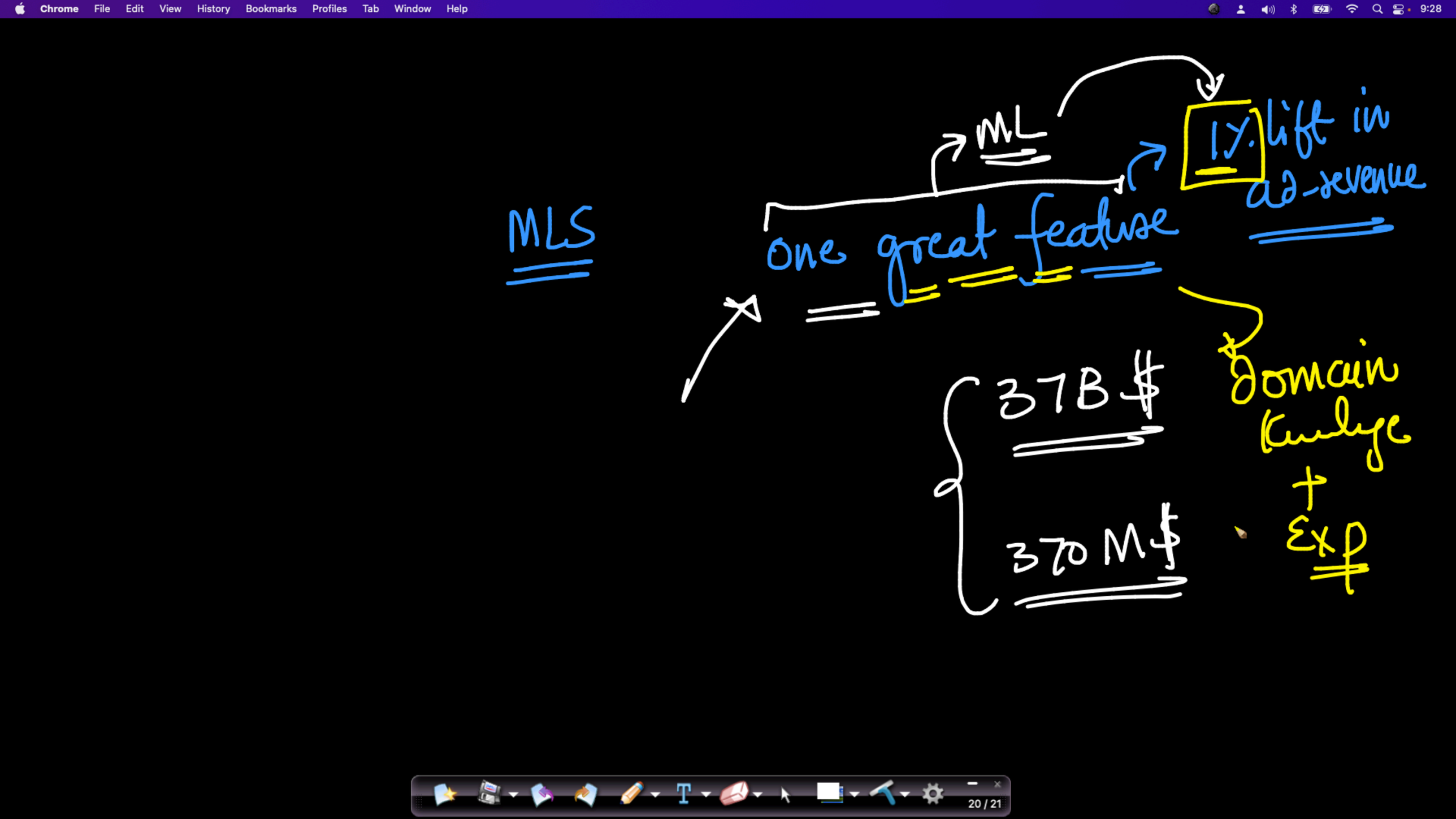
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eadc950>
```

72% P(loan-status = 1 | able to pay EMI = 1)

> P(loan-status = 0 | able to pay EMI = 0)

28%

```
[ ]
```


MLS
ML
1% lift in ad-revenue
one great feature
37B $
370 M $
Domain Knowledge + Exp

+ Code + Text

RAM
Disk

[ ] ↳ 11 cells hidden

Dependents and Loan approval

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

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

```
[243] data['Dependents'].replace('3+',3,inplace=True)
```

```
[244] data['Dependents'] = data['Dependents'].astype('float')
```

```
[245] data.dtypes
```

[ ] ↳ 11 cells hidden

Dependents and Loan approval

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

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

```
[243] data['Dependents'].replace('3+',3,inplace=True)
```

```
[244] data['Dependents'] = data['Dependents'].astype('float')
```

```
[245] data.dtypes
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:30

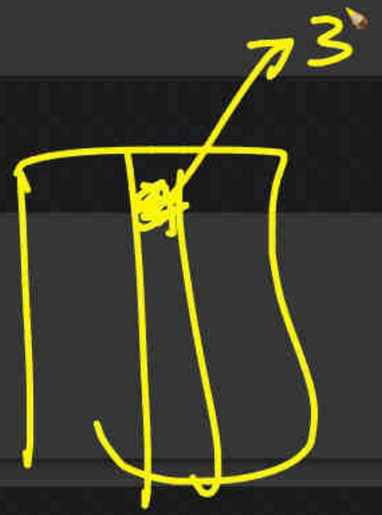
EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=0eGVXudIZgMy

+ Code + Text RAM Disk

[ ] ↳ 11 cells hidden

Dependents and Loan approval




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

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

```
[243] data['Dependents'].replace('3+',3,inplace=True)
```

```
[244] data['Dependents'] = data['Dependents'].astype('float')
```

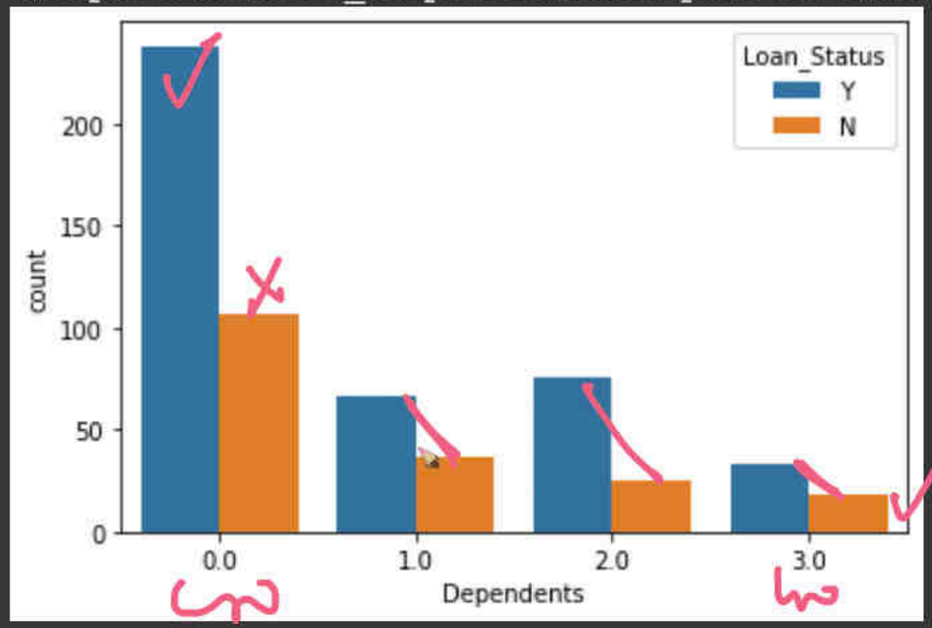
```
[245] data.dtypes
```

23 / 23

```
Able_to_pay_EMI           int64
dtype: object
```

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

      #Observations:

      ## No Dependents and 2 dependents will helps you get loan easily.
```

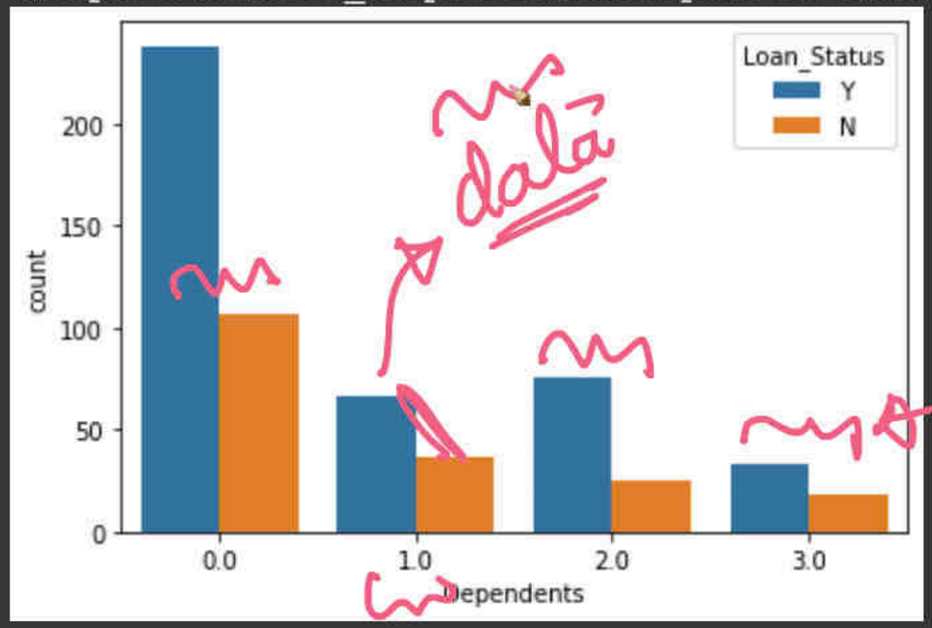
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ecd2790>
```



```
Able_to_pay_EMI           int64
dtype: object
```

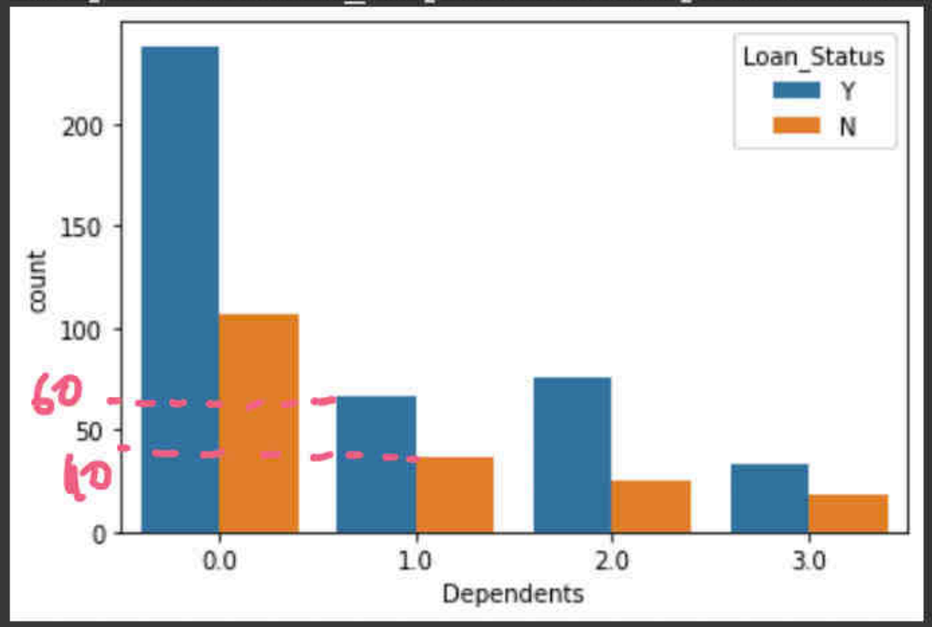
```
[246] sns.countplot(data =data, x = 'Dependents', hue = 'Loan_Status')

      #Observations:

      ## No Dependents and 2 dependents will helps you get loan easily.
```

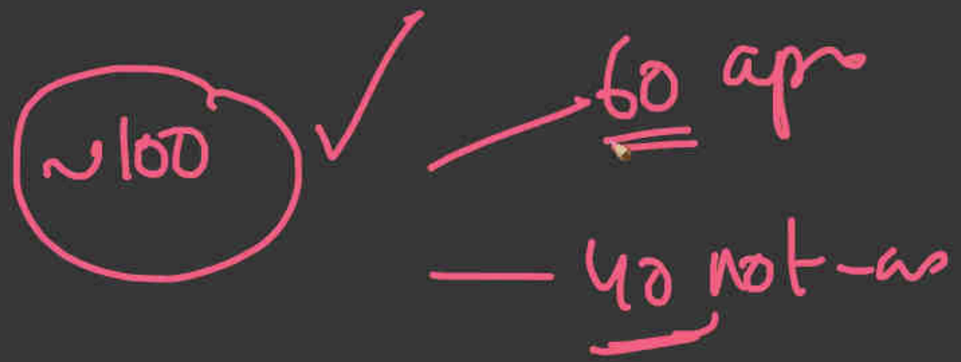
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ecd2790>
```



```
#Observations:

## No Dependents and 2 dependents will helps you get loan easily.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ecd2790>
```



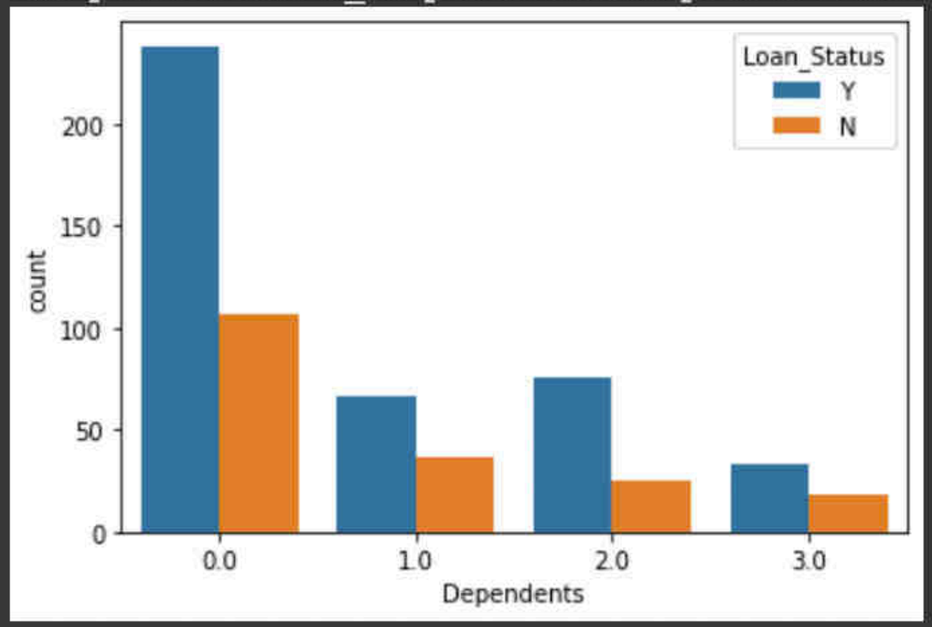


Double-click (or enter) to edit

Credit Score vs Loan Approval

```
#Observations:

## No Dependents and 2 dependents will helps you get loan easily.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ecd2790>
```



Double-click (or enter) to edit

Credit Score vs Loan Approval

t/z test

Loan amount vs approval

1 0

mean-amt

$M_1 - M_0$

$M_1 = M_0$

↳ 6 cells hidden

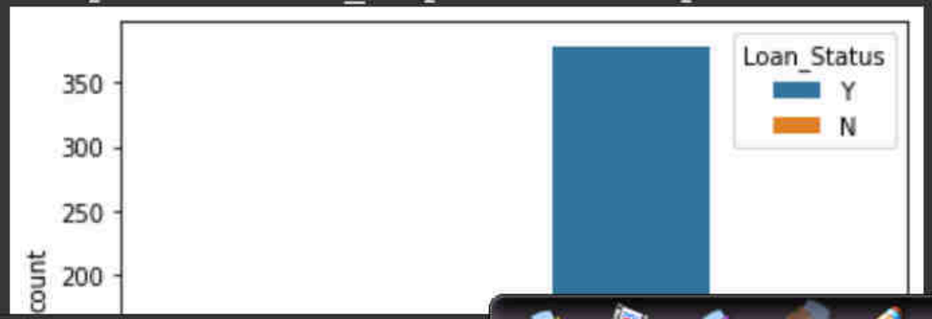
Credit Score vs Loan Approval

```
[247] 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.
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ee68c10>
```



↪ 6 cells hidden

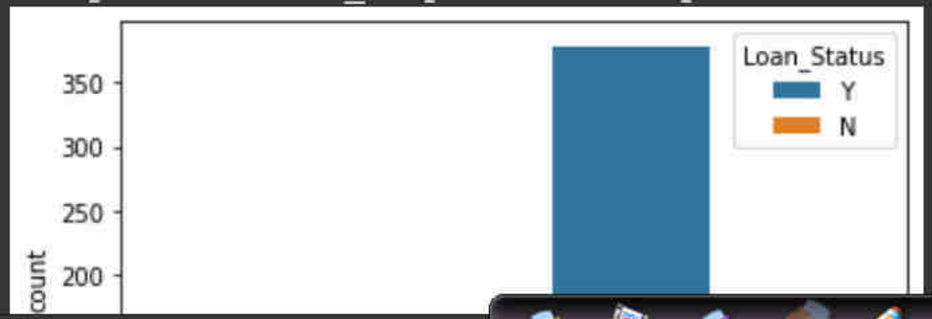
Credit Score vs Loan Approval

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

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

```
[248] 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.
```

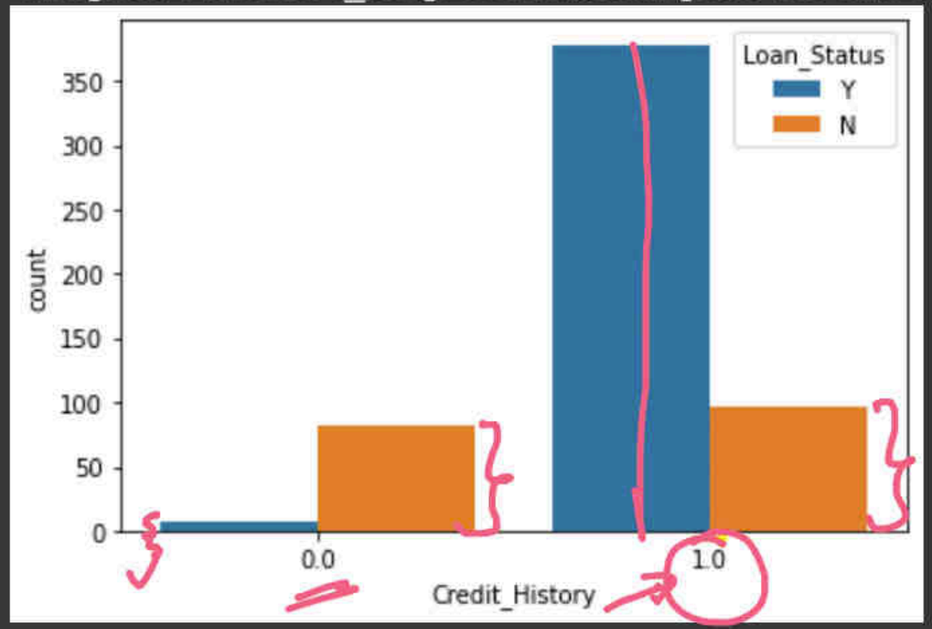
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ee68c10>
```



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

```
[248] 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 0x7f3e2ee68c10>
```



[ ] ↳ 6 cells hidden

Credit Score vs Loan Approval

↳ 3 cells hidden

Missing Values & Data Cleaning

[ ] ↳ 9 cells hidden

Task: features can help us determine loan-status

Categorical to Numerical encoding

Nominal vs Ordinal variables

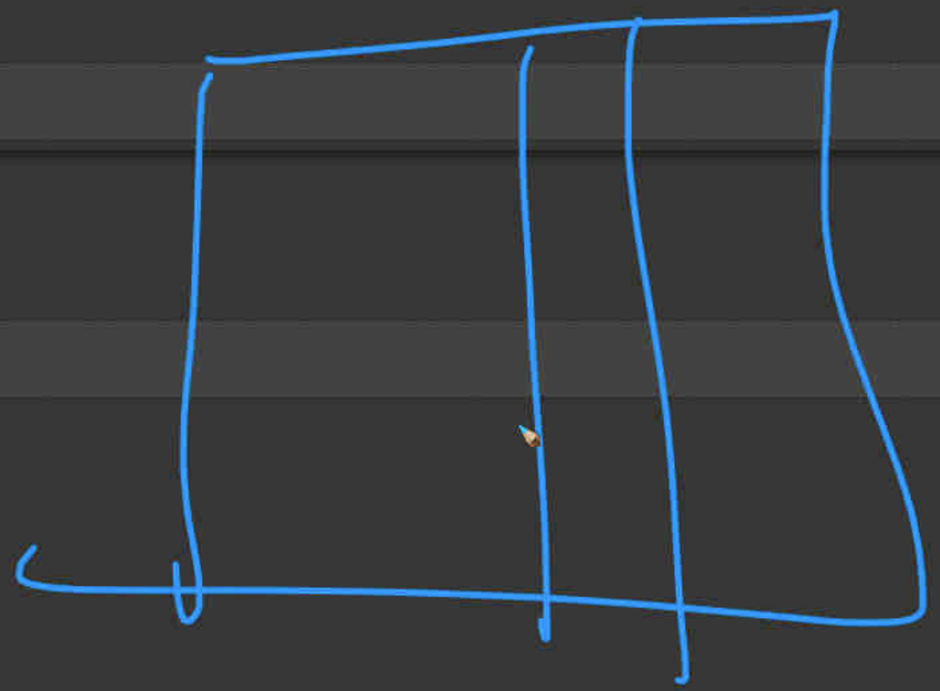
1. One Hot Encoding
2. Label encoding
3. Target Encoding

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

[ ] ↳ 6 cells hidden

Credit Score vs Loan Approval

↳ 3 cells hidden



Missing Values & Data Cleaning

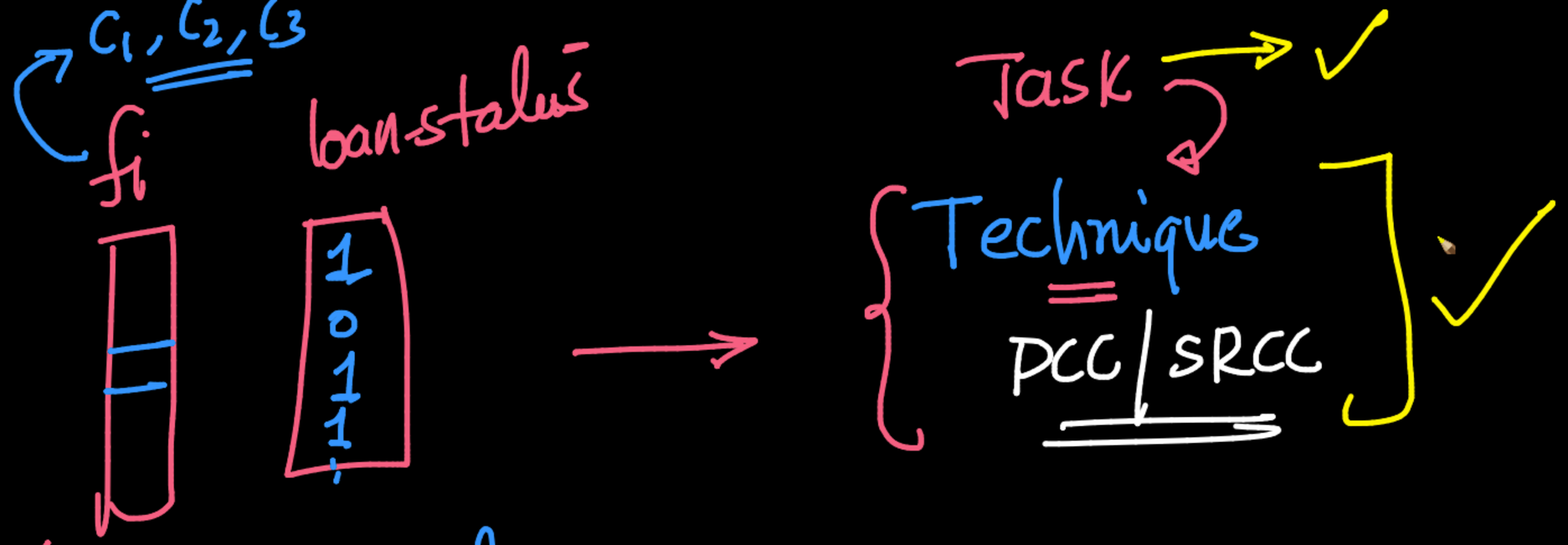
[ ] ↳ 9 cells hidden

Categorical to Numerical encoding

Nominal vs Ordinal variables

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

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



✓ - missing -values

- numerical features

[ ] ↳ 3 cells hidden

Missing Values & Data Cleaning

[249] `data.isna().sum()`

0s

```
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
```

M, F, [illegible]

[ ] ↳ 3 cells hidden

Missing Values & Data Cleaning

[249] `data.isna().sum()`

0s

```
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
```

```
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, keys=['Total', 'Percent'])
    return missing_data_df
```

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

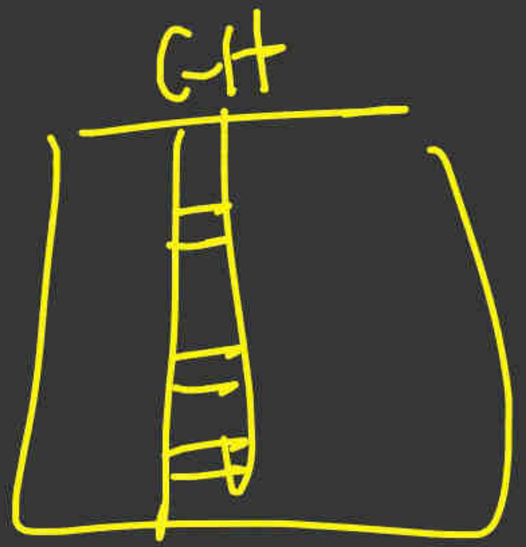
	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192
EMI	36	5.863192
Self_Employed	32	5.211726
LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599

```
[252] # Credit History=2 for n
```

```
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, keys=['Total', 'Percent'])
    return missing_data_df
```

```
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




```
[252] # Credit History=2 for n
```

```
Able_to_pay_EMI          0
dtype: int64
```

[250]
```
# 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
```

```
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

```
    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
```

```
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

```
[250] # 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
```

```
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

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

```
[250] # 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
```

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

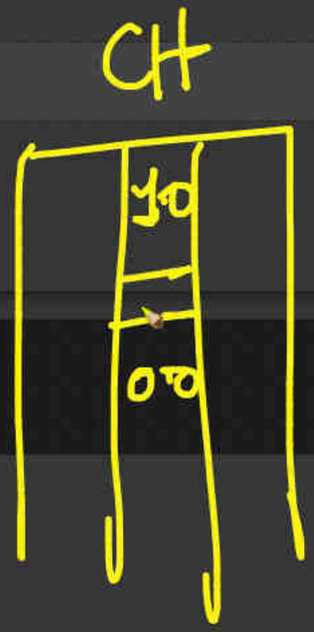
	Total	Percent
Credit_History	50	8.143322
Loan_Amount_per_year	36	5.863192

Loan Amount and Loan Term

[ ] ↳ 11 cells hidden

Dependents and Loan **approval**

[ ] ↳ 6 cells hidden



Credit Score vs Loan Approval

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

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

```
[248] 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.
```

LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599

[252]
```
# Credit History=2 for nan/missing values.
data['Credit_History'] = data['Credit_History'].fillna(2)
```

[253]
```
# Self_Employed = 'Other' for nan
data.Self_Employed.unique()
```

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

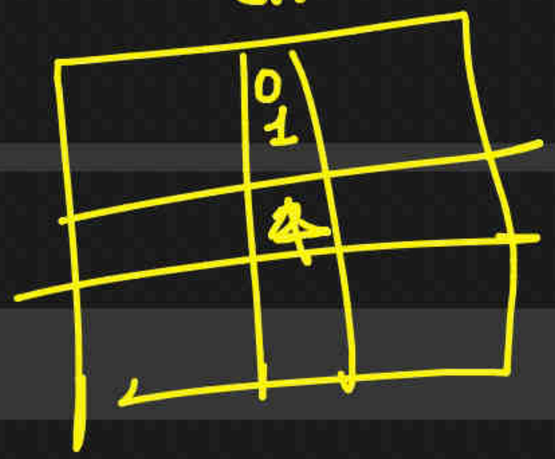
[254]
```
data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

[255]
```
# median imputation for numerical columns.
```

LoanAmount	22	3.583062
Dependents	15	2.442997
Loan_Amount_Term	14	2.280130
Gender	13	2.117264
Married	3	0.488599

imputation

CH → categorical



[252]
```
# Credit History=2 for nan/missing values.
data['Credit_History'] = data['Credit_History'].fillna(2)
```

[253]
```
# Self_Employed = 'Other' for nan
data.Self_Employed.unique()
```

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

[254]
```
data['Self_Employed'] = data['Self_Employed'].fillna('Other')
```

[255]
```
# median imputation for numerical columns.
```

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)
```

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

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

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

Yes
No
other ✓

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

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

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)
```

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

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

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

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

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

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:49

EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=bQzPBynqbx2h

+ Code + Text RAM Disk

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

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

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

```
[255] # 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])))
```

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

      freq_imputer = SimpleImputer(strategy = 'most_frequent')
      for col in cat_missing:
```

scikit-learn

Classical ML

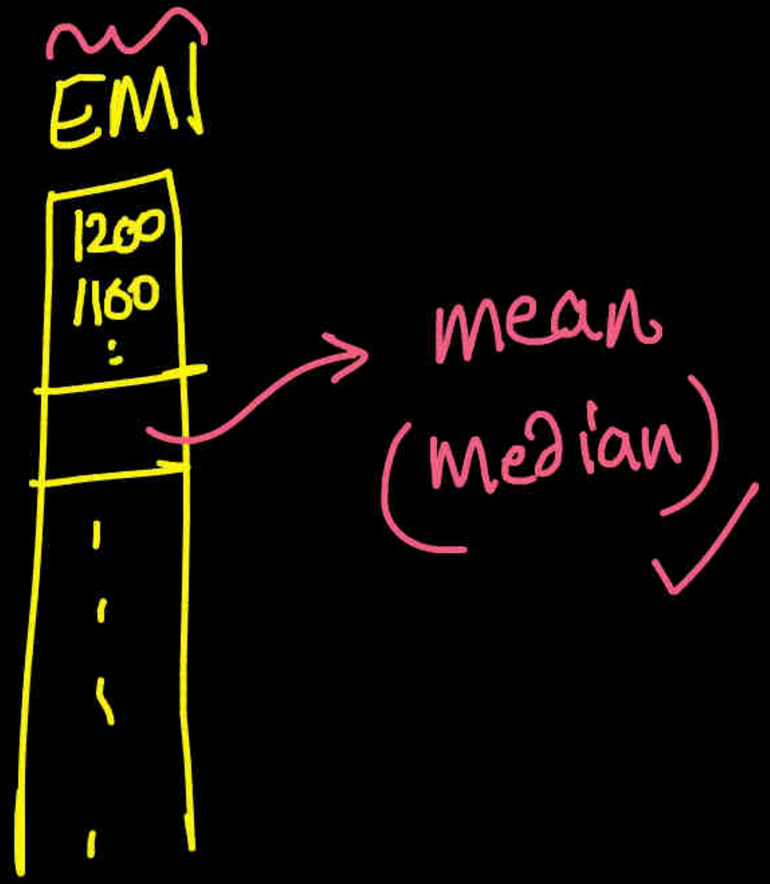
47 / 47

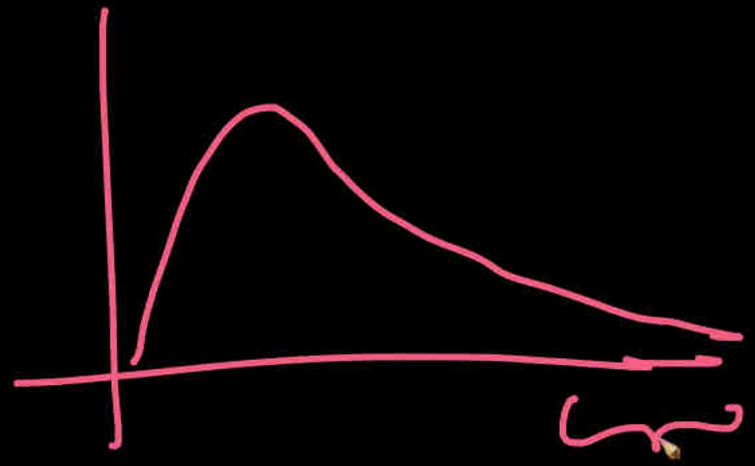
```
[253] # Self_Employed = 'Other' for nan
      data.Self_Employed.unique()

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

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

```
[255] # 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])))
```

```
[256] # 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'] = 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])))
```

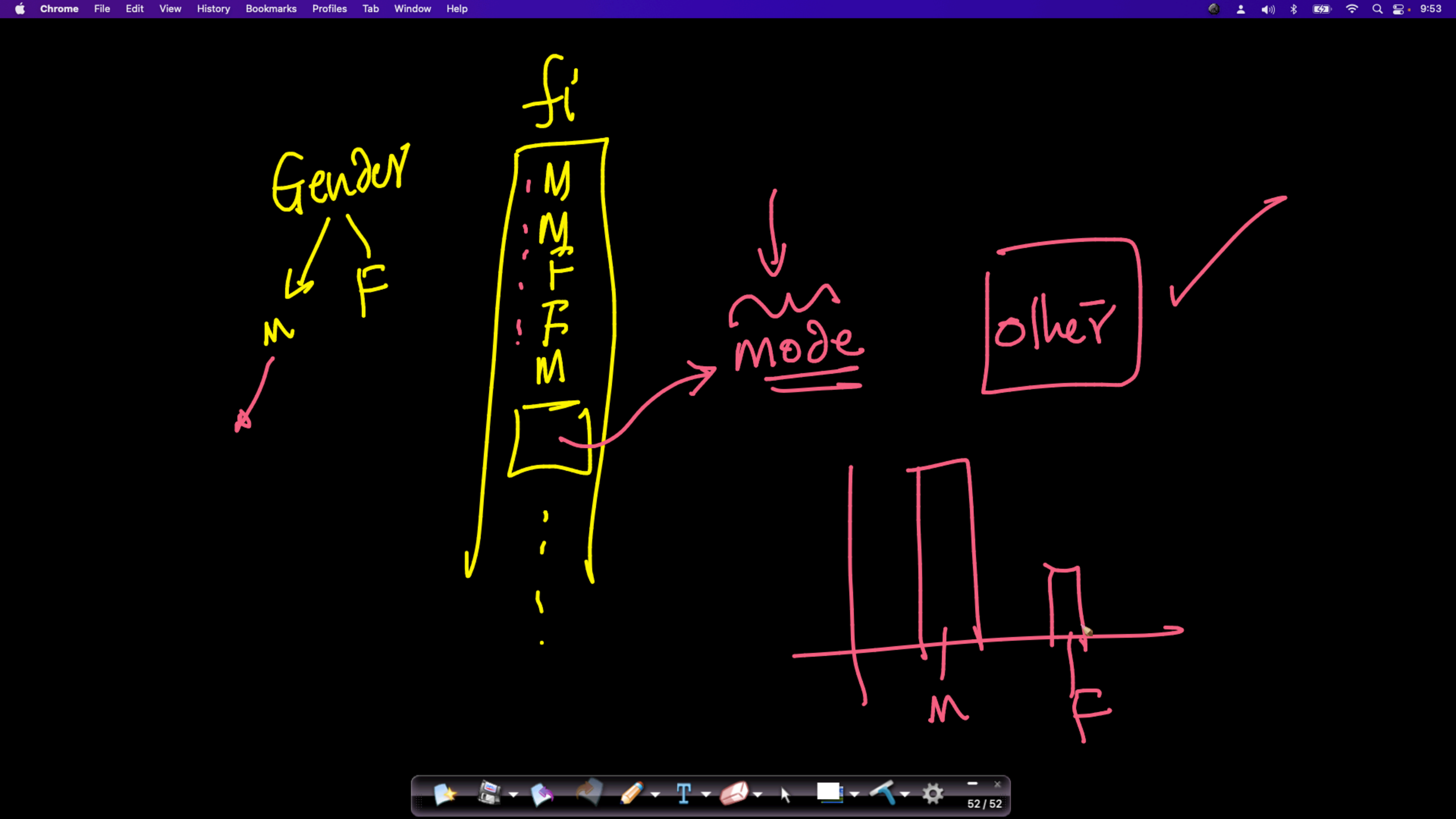
```
missing_df = missing_to_df(data)
missing_df[missing_df['Total'] > 0]
```

```
[ ] 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]
```

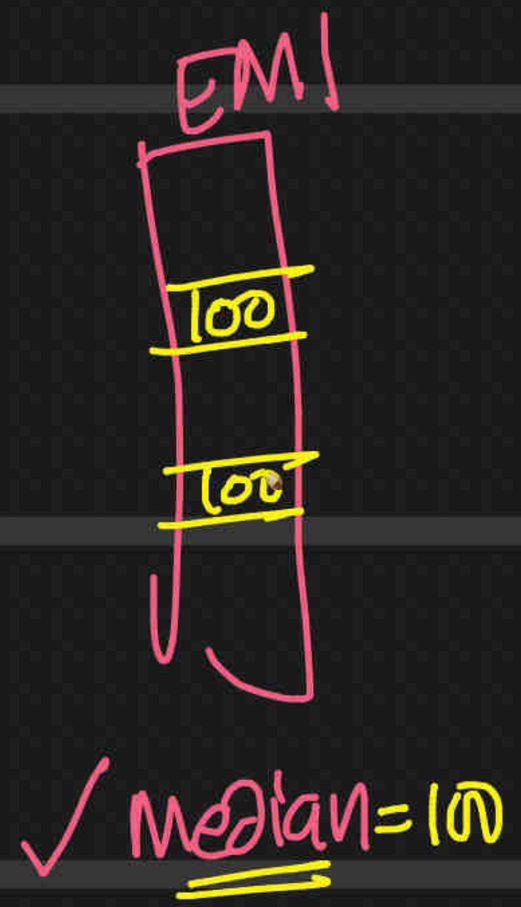

Gender
M
F
fi
M
M
F
F
M
mode
other
M
F

```
[ ] 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]
```



```
[ ] 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]
```

```
[ ] 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])))
```

0, 1, 2, 3

Mode

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

	Total	Percent

Categorical to Numerical encoding

Nominal vs Ordinal variables

```
    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]
```

	Total	Percent

Categorical to Numerical encoding

Nominal vs Ordinal variables

+ Code + Text Reconnect

```
[ ]  # 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]
```

	Total	Percent

Categorical to Numerical encoding

Nominal vs Ordinal variables

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

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

Imputation

- → Common Sense
- → median / Mean
- → Mode
- → model based Imputation (KNN) (later)

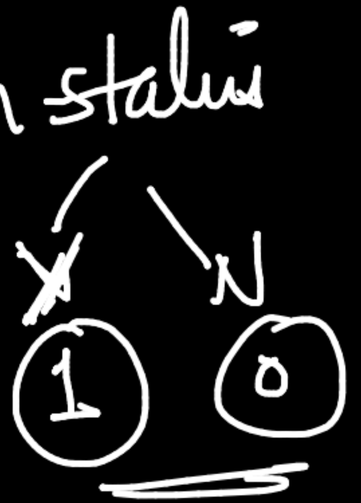
0, 1 [2]

median_imputation

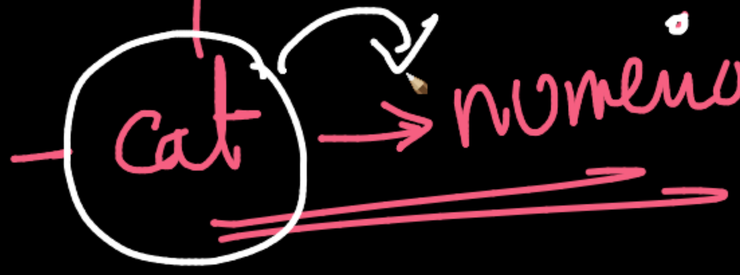
fit transform ... does internally

Task: find most useful features fw loan-status

Tech: Correlation : PCC, SRCC

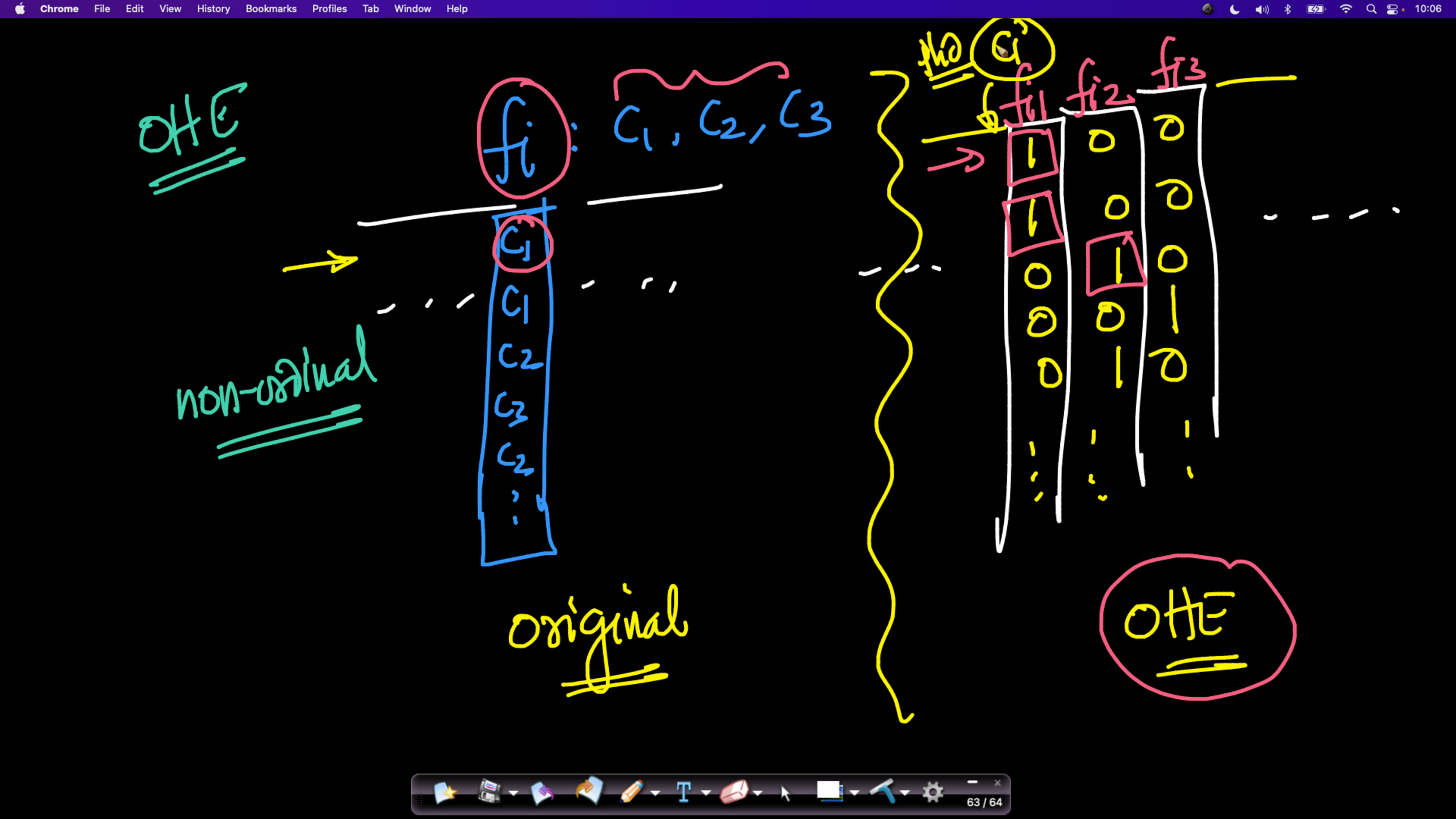


- Impute & avoid missing values



Cat → Num

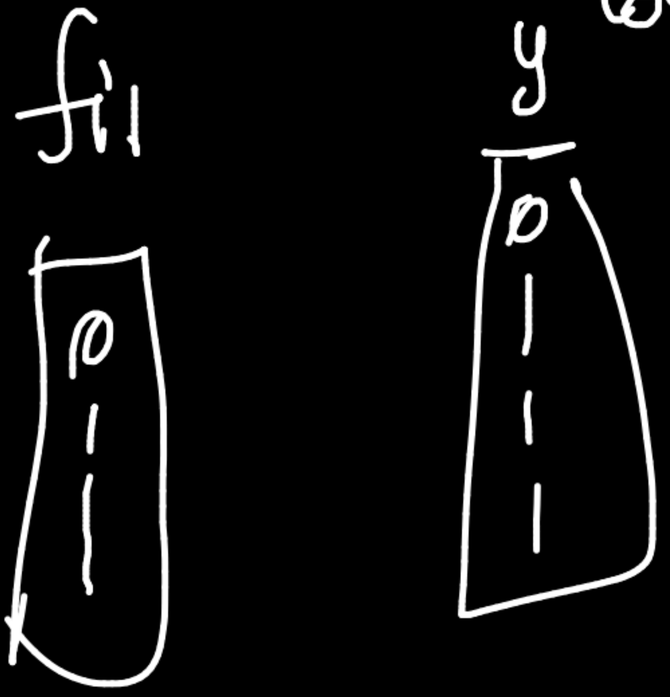
1. OHE → Some ML models
2. Label encoding
3. Target encoding


OHE
fi : C1, C2, C3
C1
C1
C2
C3
C2
non-original
original
No
Ci
fi1
fi2
fi3
OHE

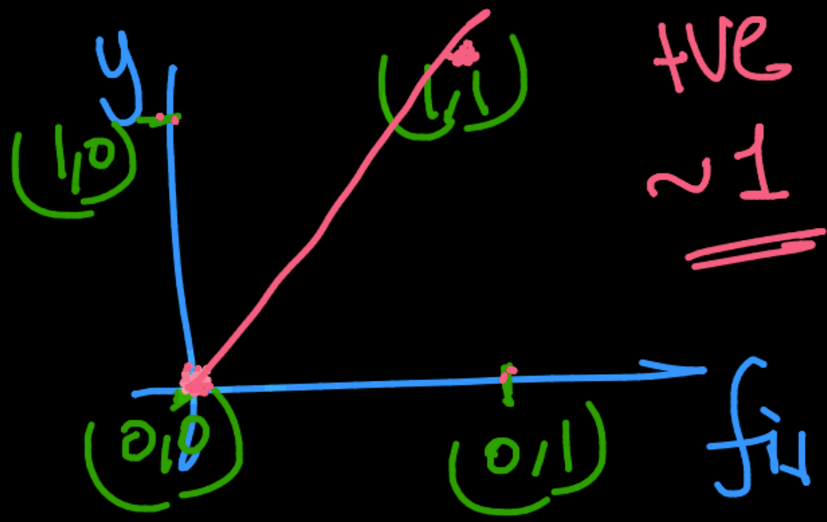
OHE → Lin. Alg + High-dim Geom + ML (later)

Full-power

Simplified



loan-status



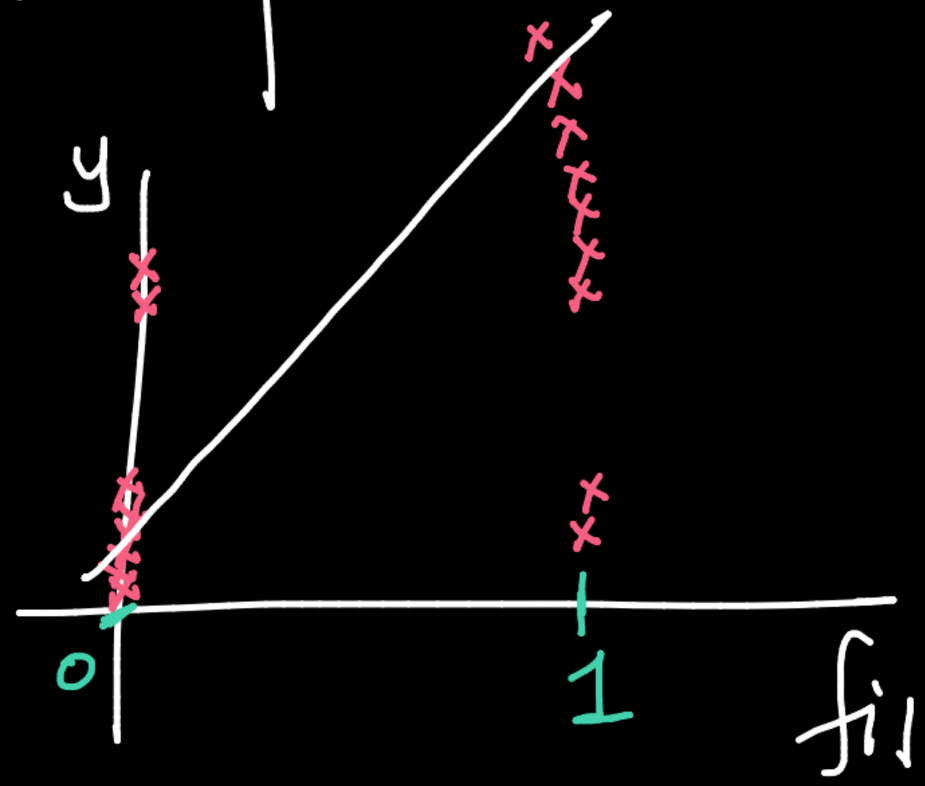
lasso +ve ~1

PCC(fi, y)

Petrol ← f_1

1
0
1
1
0
p
!

y → Car sale prices



PCC : high +ve

high-dim → Curse-of-dim

OHE → Carefully (Log-Reg, Lr-reg)

(later) - ML

binary cat-variables

loan-status

Y N

1 0

✓ Label Encoder

Marital-status

M NM

1 0

{[locality → U SU R

X Gender — M F other

0 1 2

arbitrary order

Q

binary categorical feature!

Loan-status

Y N

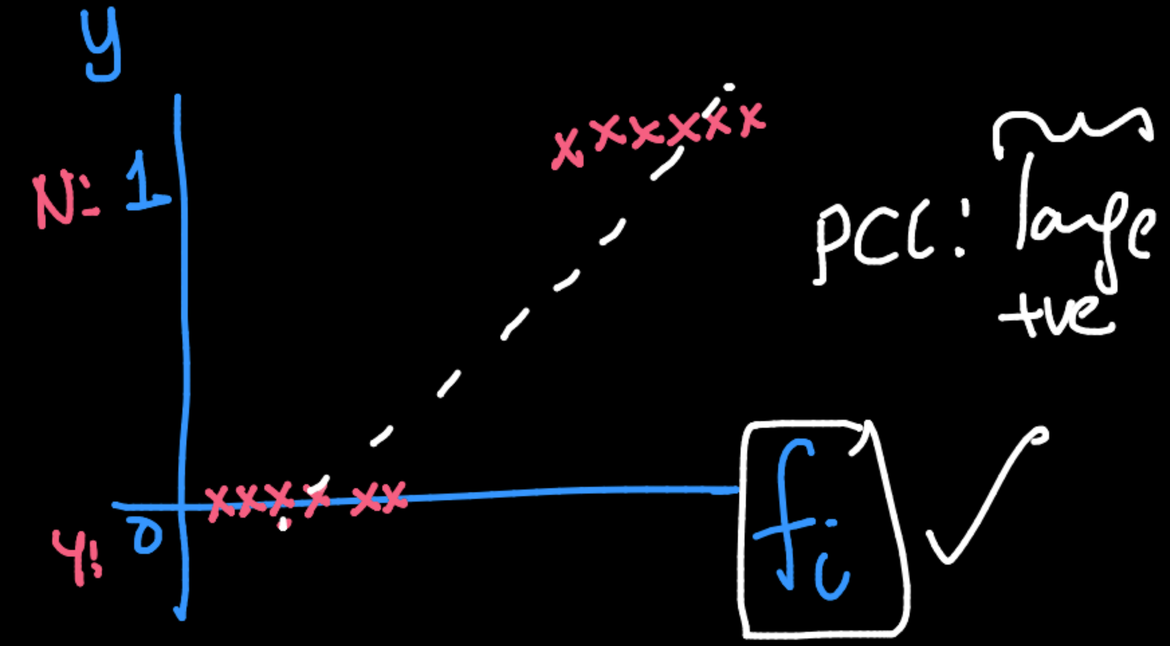
0 1

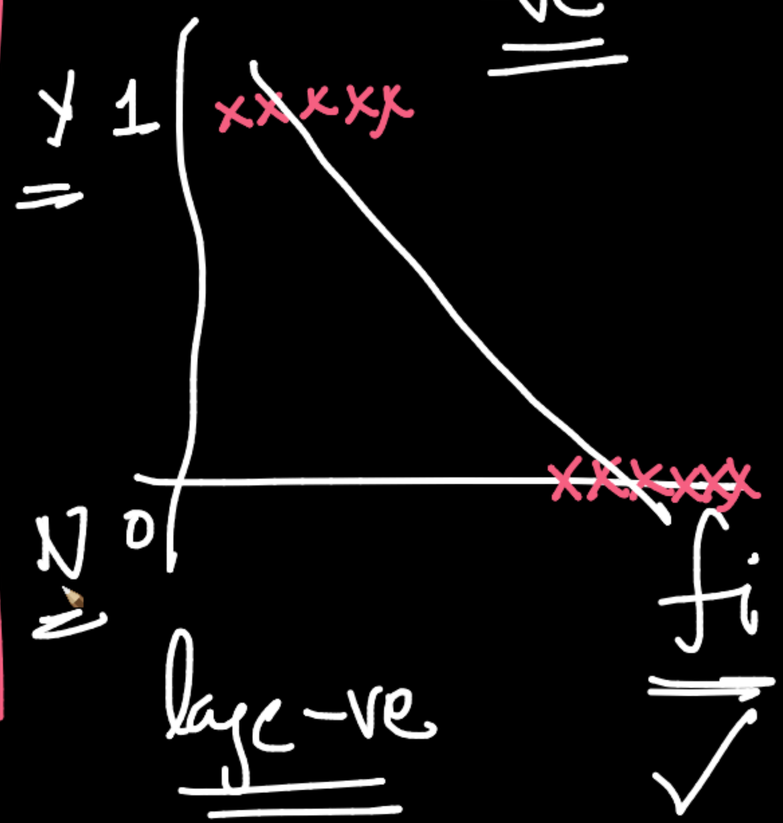
Y N

1 0

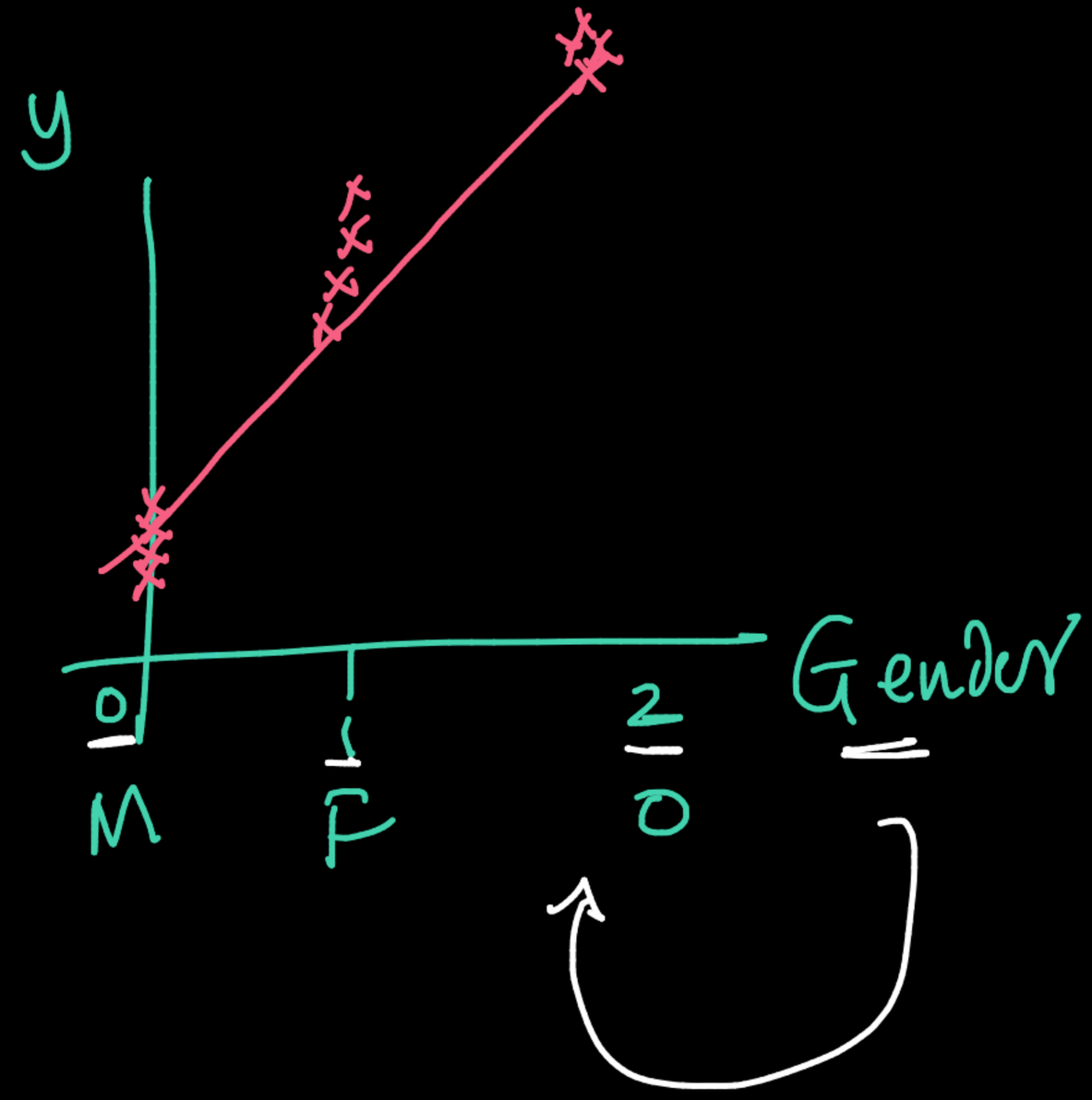
+ve

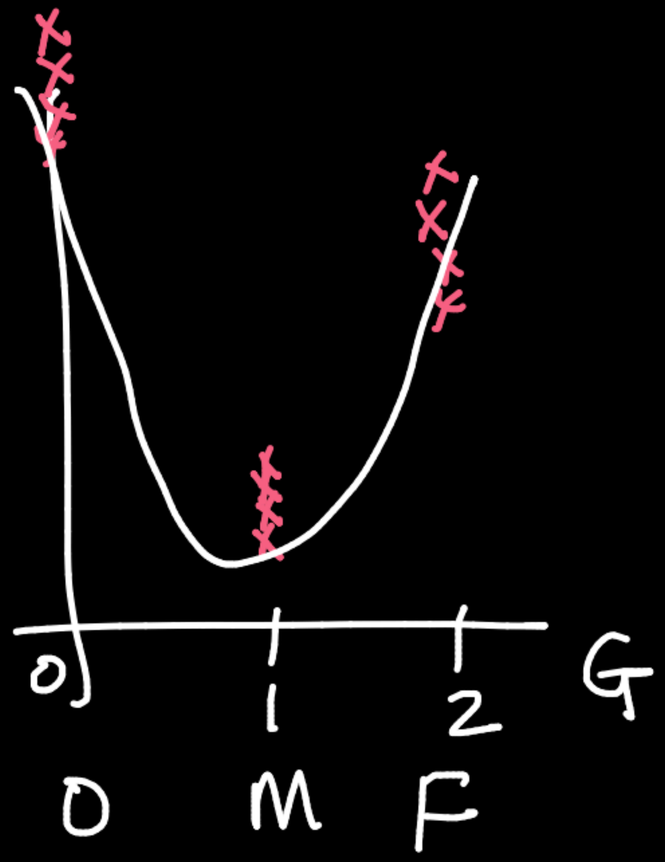
-1 0 +1



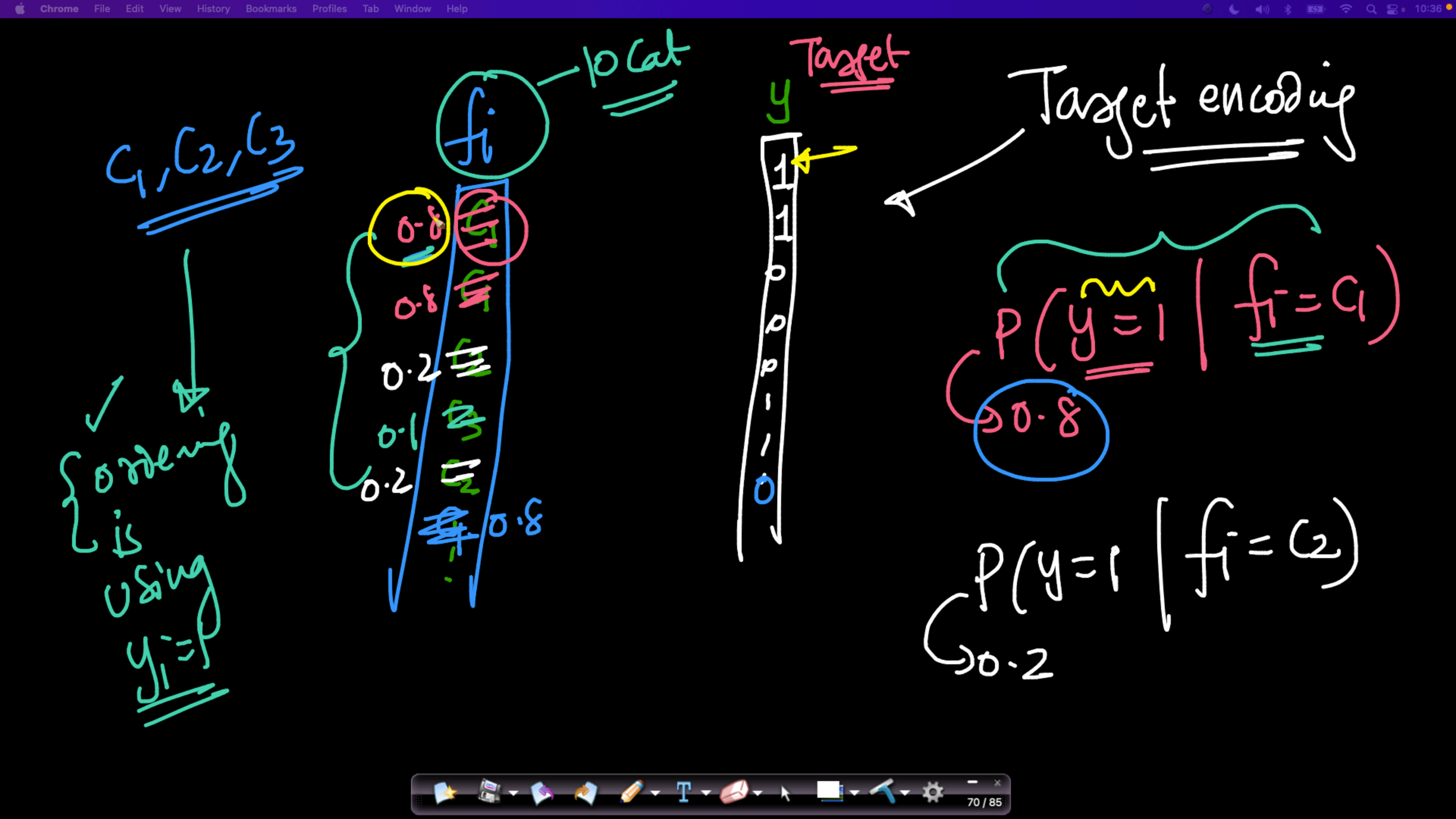


68 / 68





PCC / SLCC


C1, C2, C3
ordering is using yi=p
fi
10 Cat
0.8
0.8
0.2
0.1
0.2
0.8
Target
y
1
1
0
0
0
1
0
Target encoding
P(y=1 | fi=C1)
0.8
P(y=1 | fi=C2)
0.2

$c_2 = 136, 136 \ldots$

138

{Target encoding

126

$c_1 \rightarrow 100, 121, 146 \ldots$

Mean/Median of all y_i's where $f_i = c_1$

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']
```

mean ✓
median
SD
MAD

Loan Status

[ ] ↳ 3 cells hidden

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

[ ] ↳ 3 cells hidden

+ Code + Text Reconnect

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 10:29

EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=p1Bnyafgh7Ka

+ Code + Text Reconnect

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
```

75 / 75

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
```

Gender

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

```
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
```

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

Married

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

```
Yes    401
No     213
Name: Married, dtype: int64
```

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

```
1    401
0    213
Name: Married, dtype: int64
```

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

EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

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

+ Code + Text Reconnect

[ ] ↳ 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 sati                                                    dist-packages (from category_encoders)
```

79 / 79

[ ] ↳ 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: statsmodels>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
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: statsmodels>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
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

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')
```

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
```

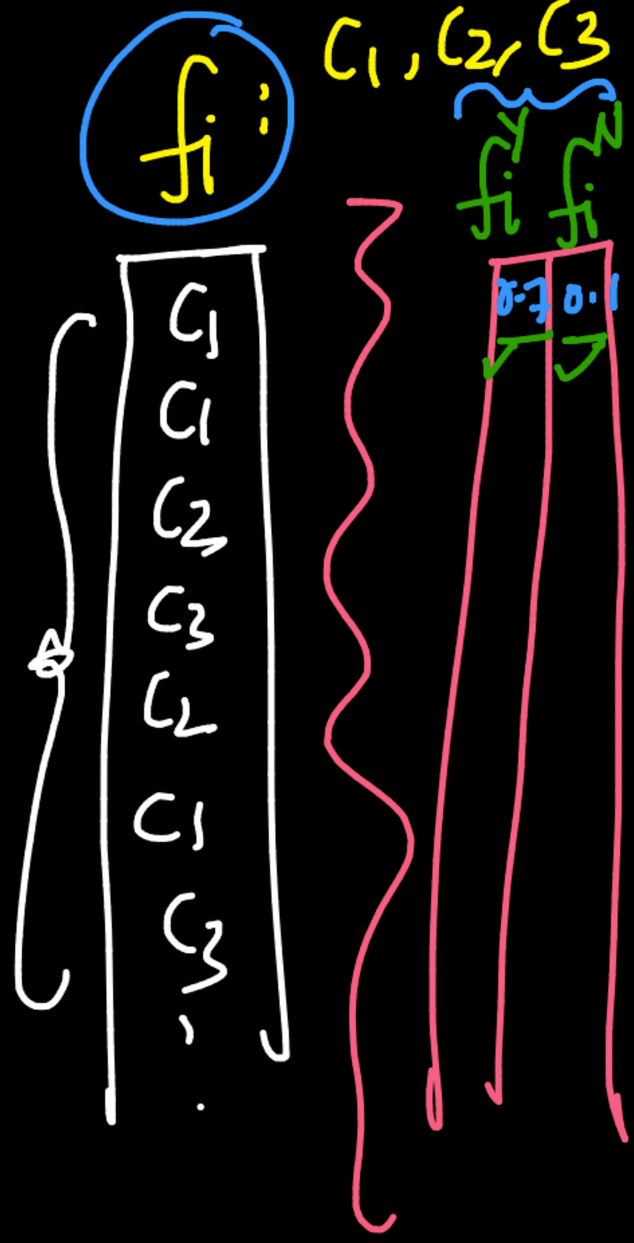
```
[ ] s = (data.dtypes == 'object')
```

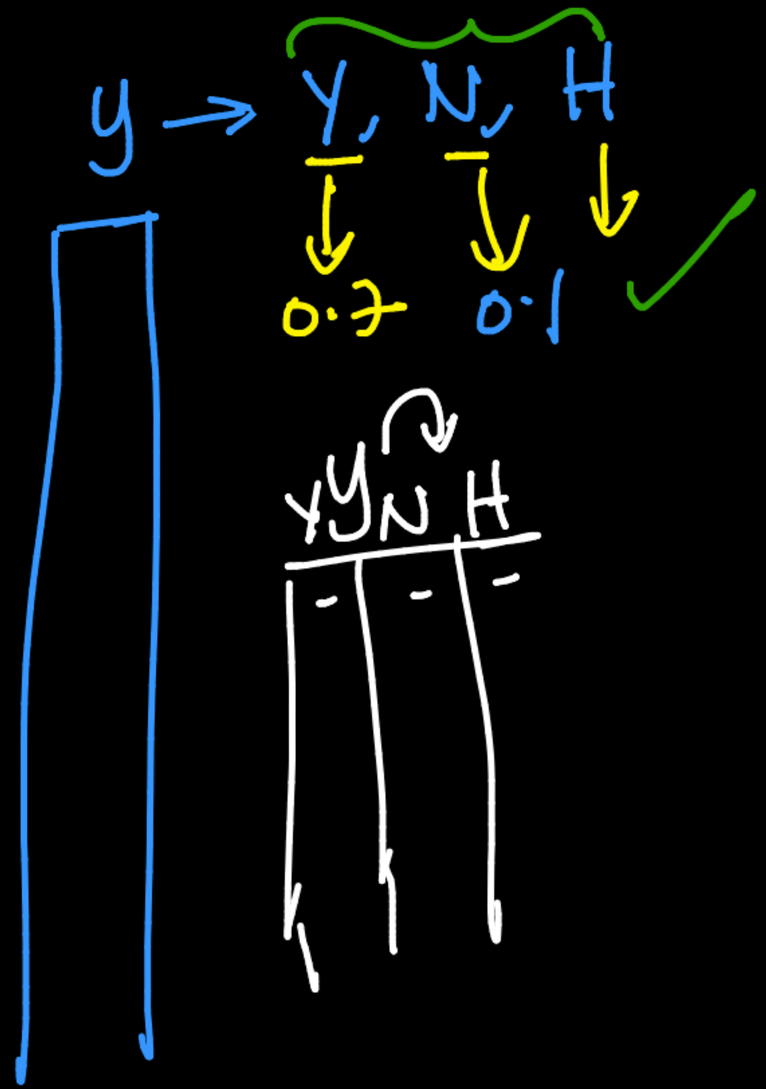
Q

Why not use counts?

$$P(y=Y \mid f_i = c_1) = 0.7$$

$$P(y=N \mid f_i = c_1) = 0.1$$





$y = Y, N$

$0.8 \quad 0.2$

10:48

✓ Missing-values

✓ Cat → Numeric

✓ { (PCC & SRCC) ✓

Standardization :-

$$\frac{x - \mu}{\sigma}$$

↓

simple extension

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:50

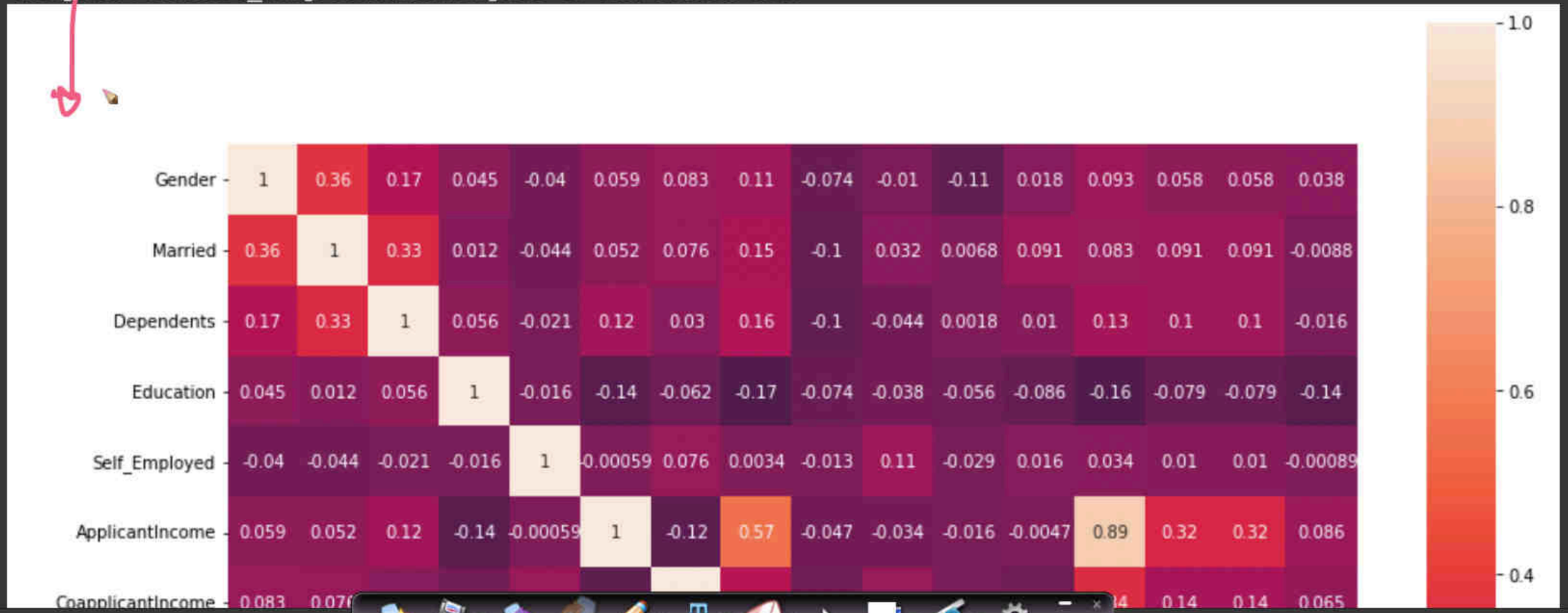
EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=Qk2PGP6bia6d

+ Code + Text Reconnect

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2ed98f50>
```



88 / 88

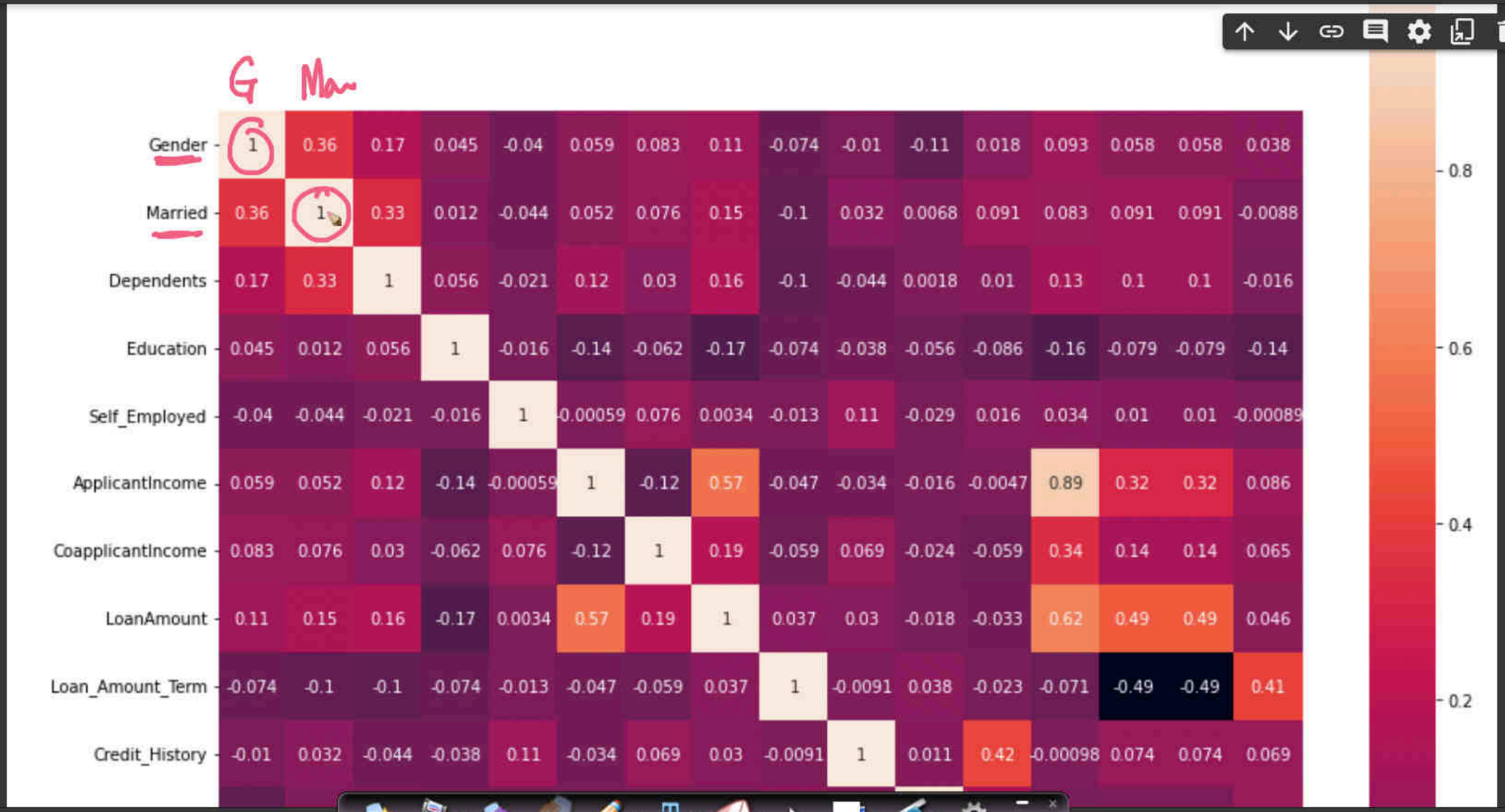
Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:51

EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=Qk2PGP6bia6d

+ Code + Text Reconnect

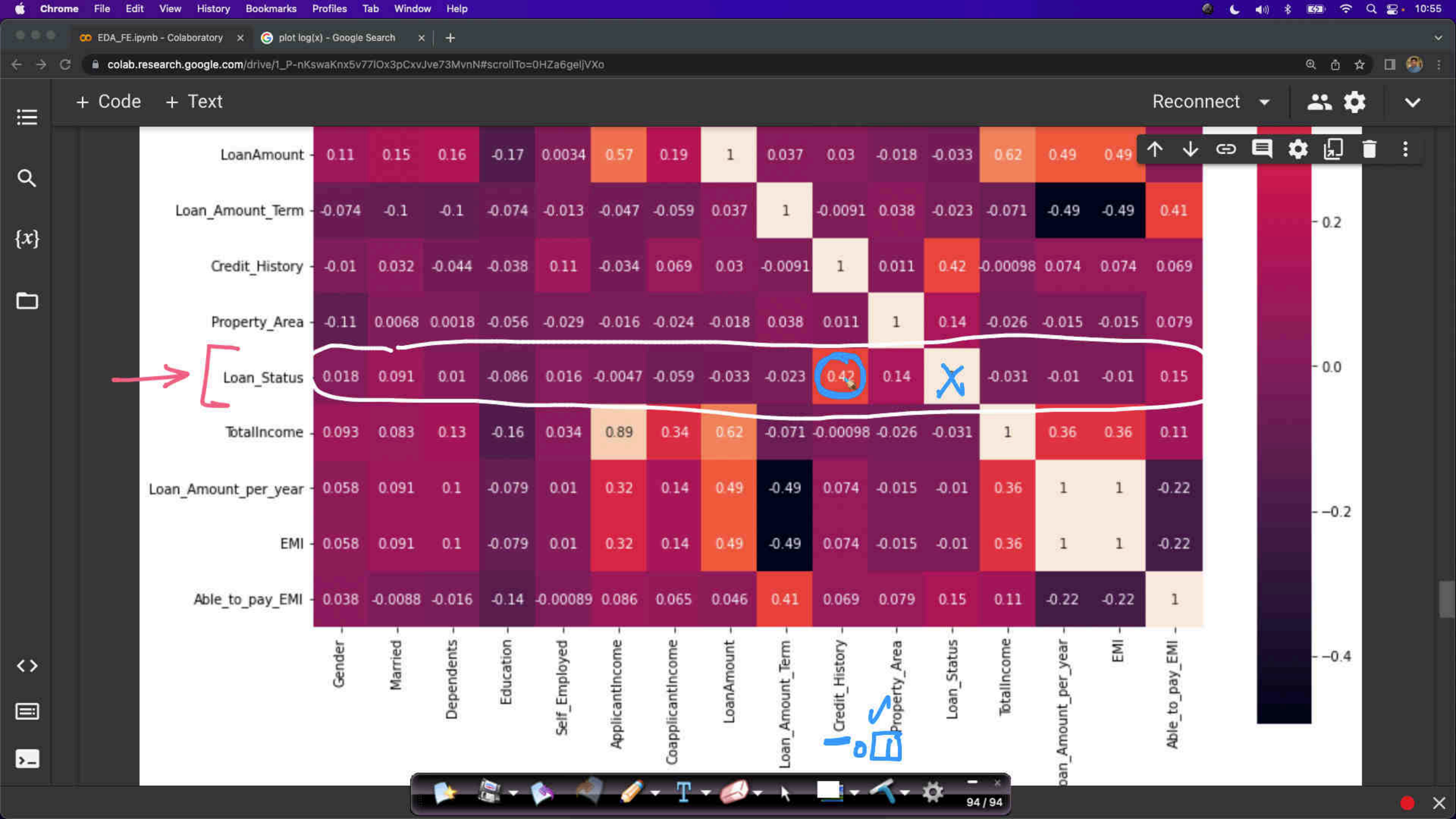
89 / 89

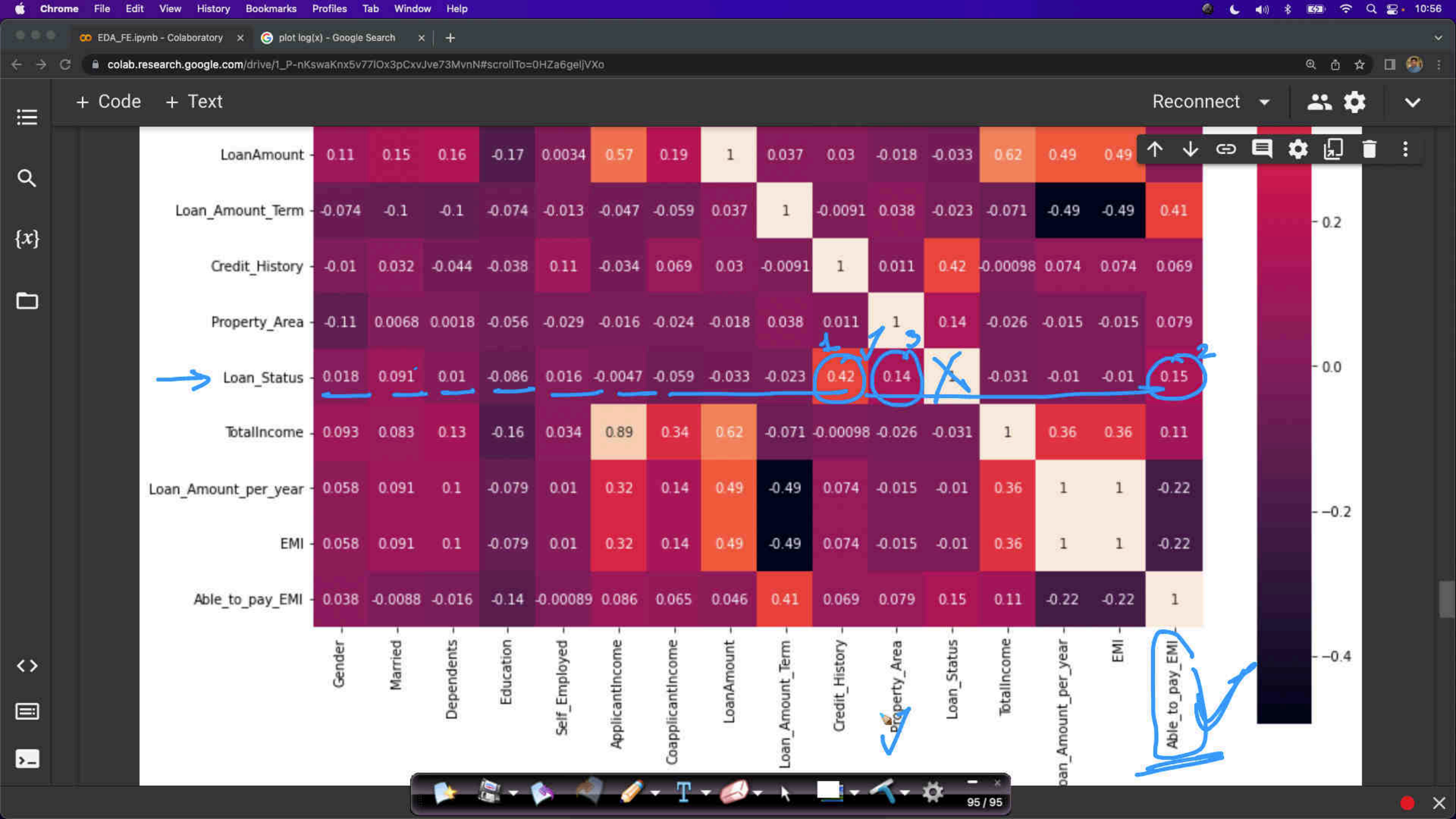

Gender
Married
Dependents
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
1 0.36 0.17 0.045 -0.04 0.059 0.083 0.11 -0.074 -0.01 -0.11 0.018 0.093 0.058 0.058 0.038
0.36 1 0.33 0.012 -0.044 0.052 0.076 0.15 -0.1 0.032 0.0068 0.091 0.083 0.091 0.091 -0.0088
0.17 0.33 1 0.056 -0.021 0.12 0.03 0.16 -0.1 -0.044 0.0018 0.01 0.13 0.1 0.1 -0.016
0.045 0.012 0.056 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
-0.04 -0.044 -0.021 -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
0.059 0.052 0.12 -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
0.083 0.076 0.03 -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
0.11 0.15 0.16 -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
-0.074 -0.1 -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
-0.01 0.032 -0.044 -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
0.8
0.6
0.4
0.2

Loan at/yr = EMI


Loan_Amount_Term -0.074 -0.1 -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 -0.044 -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 0.0018 -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 0.01 -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 0.13 -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 0.1 -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 0.1 -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 -0.016 -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
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
0.2
0.0
−0.2
−0.4


Loan_Amount_Term -0.074 -0.1 -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 -0.044 -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 0.0018 -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 0.01 -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 0.13 -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 0.1 -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 0.1 -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 -0.016 -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
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
0.2
0.0
−0.2
−0.4


Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:55
EDA_FE.ipynb - Colaboratory
plot log(x) - Google Search
colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=0HZa6geljVXo
+ Code
+ Text
Reconnect
LoanAmount 0.11 0.15 0.16 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49
Loan_Amount_Term -0.074 -0.1 -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 -0.044 -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 0.0018 -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 0.01 -0.086 0.016 -0.0047 -0.059 -0.033 -0.023 0.42 0.14 -0.031 -0.01 -0.01 0.15
TotalIncome 0.093 0.083 0.13 -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 0.1 -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 0.1 -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 -0.016 -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
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
0.2
0.0
−0.2
−0.4
94 / 94


Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:56
EDA_FE.ipynb - Colaboratory
plot log(x) - Google Search
colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=0HZa6geljVXo
+ Code
+ Text
Reconnect
LoanAmount 0.11 0.15 0.16 -0.17 0.0034 0.57 0.19 1 0.037 0.03 -0.018 -0.033 0.62 0.49 0.49
Loan_Amount_Term -0.074 -0.1 -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 -0.044 -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 0.0018 -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 0.01 -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 0.13 -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 0.1 -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 0.1 -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 -0.016 -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
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
0.2
0.0
−0.2
−0.4
95 / 95

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:56

EDA_FE.ipynb - Colaboratory | plot log(x) - Google Search

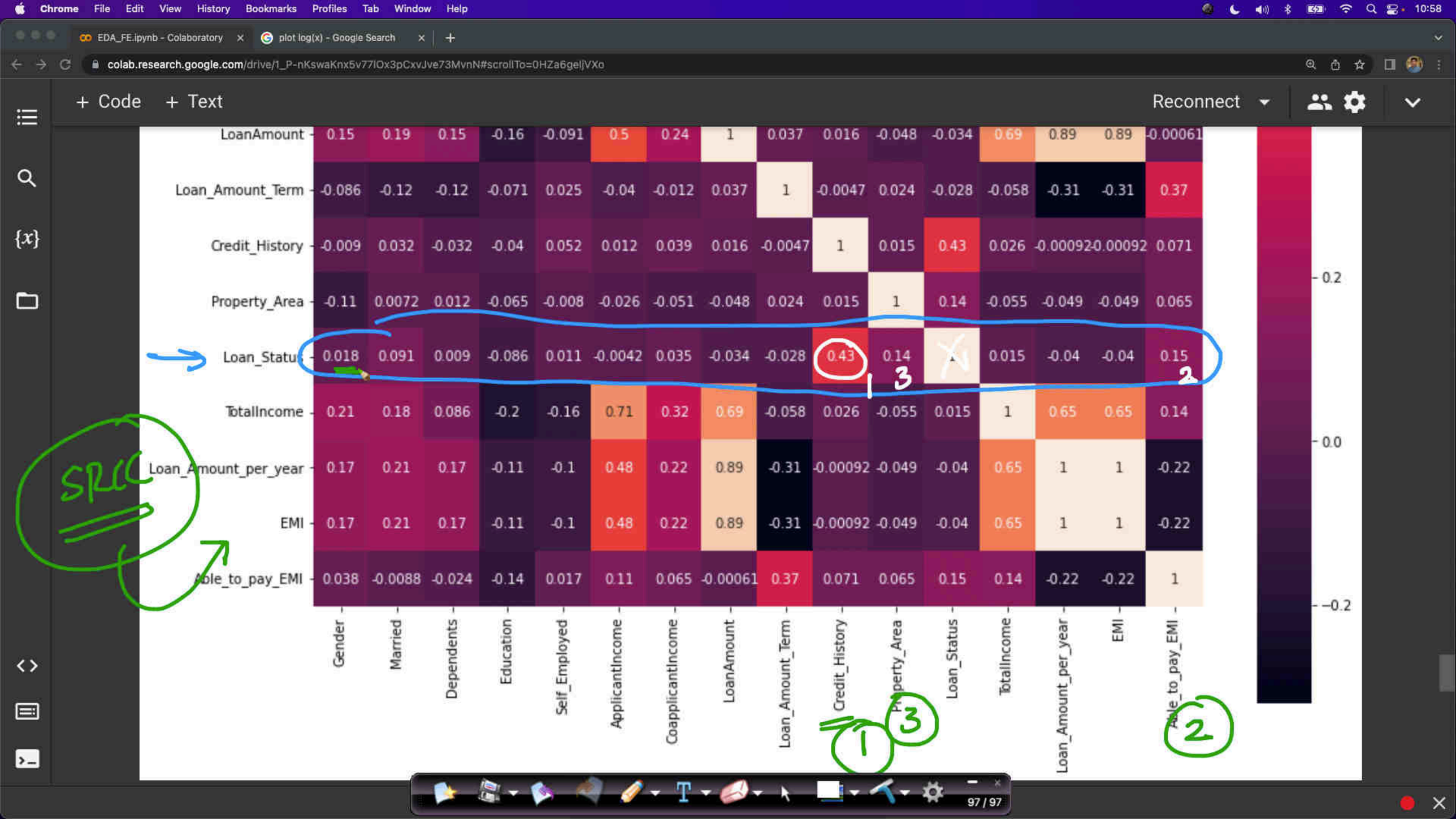
colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=0HZa6geljVXo

+ Code + Text Reconnect

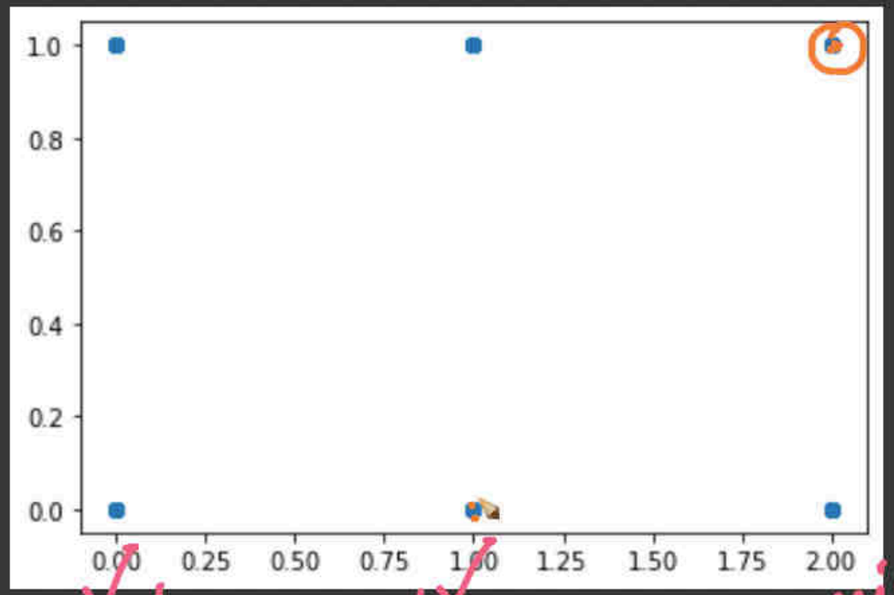
```
#SRCC
plt.figure(figsize=(15, 15))
sns.heatmap(data.corr(method='spearman'), square=True,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2f27f950>
```

96 / 96


Chrome File Edit View History Bookmarks Profiles Tab Window Help
EDA_FE.ipynb - Colaboratory
plot log(x) - Google Search
colab.research.google.com/drive/1_P-nKswaKnx5v77lOx3pCxvJve73MvnN#scrollTo=0HZa6geljVXo
+ Code + Text
Reconnect
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
0.2
0.0
−0.2
SRCC
1
3
2
97 / 97

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



0.41

PCC & SRCC

N

Y

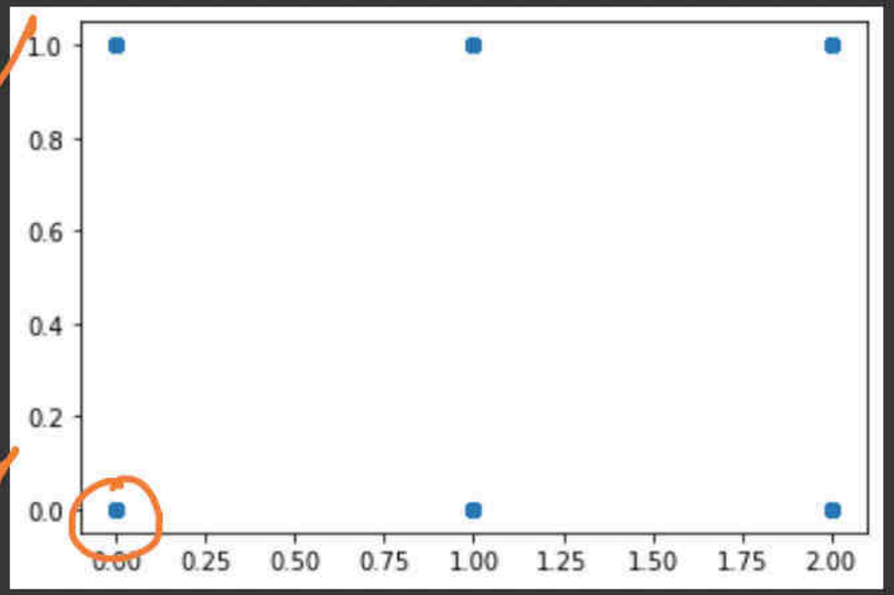
missing

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

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

+ Code + Text Reconnect

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

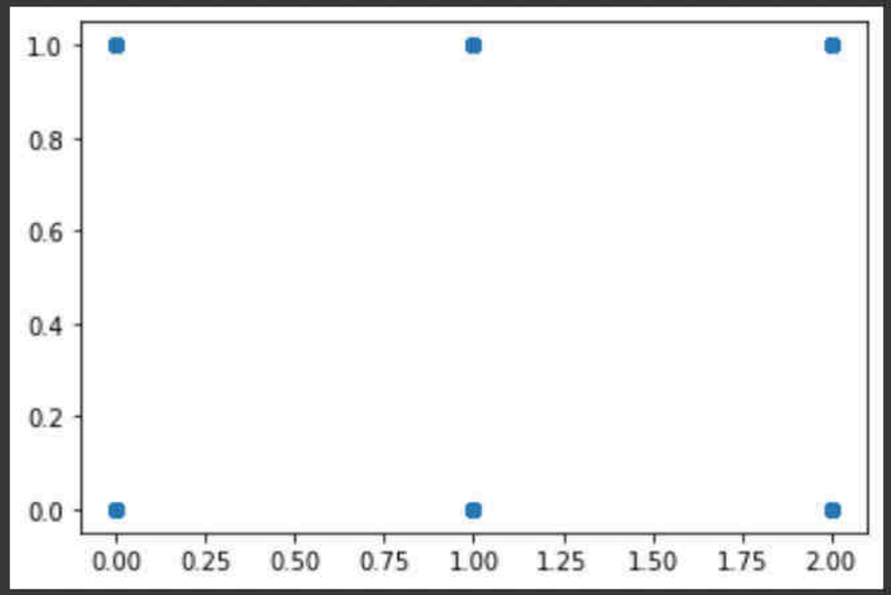


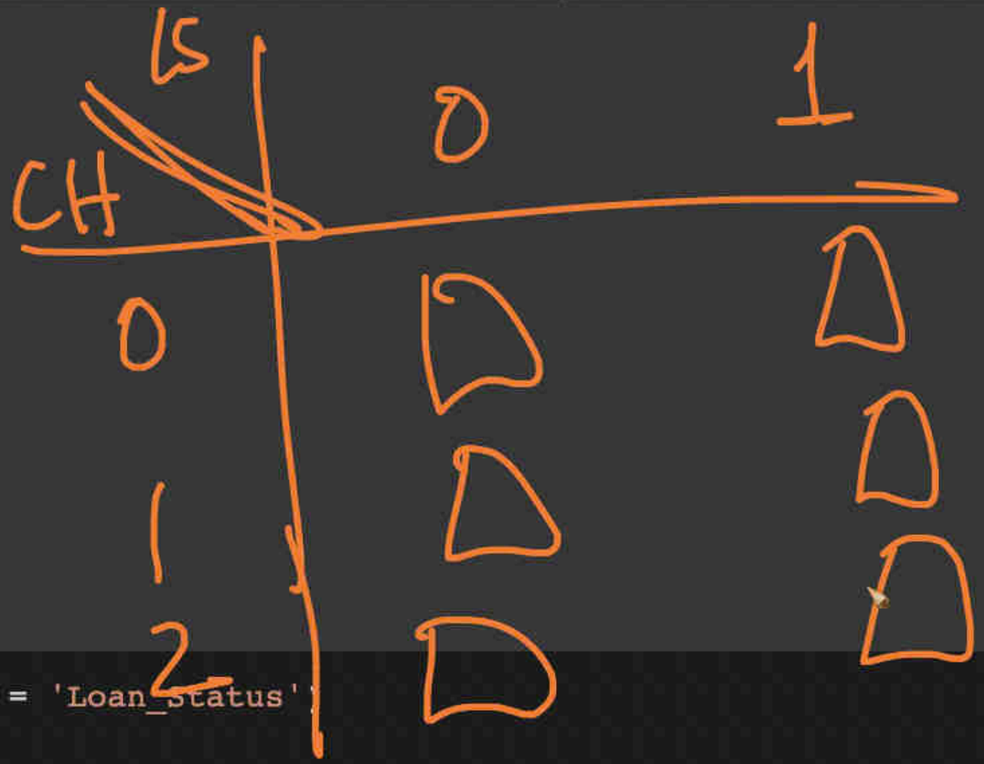
- PCC / SRCC
- plot & Scale

```
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
```

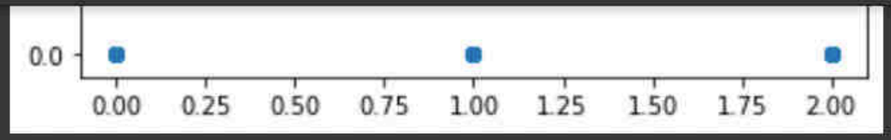





```
sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

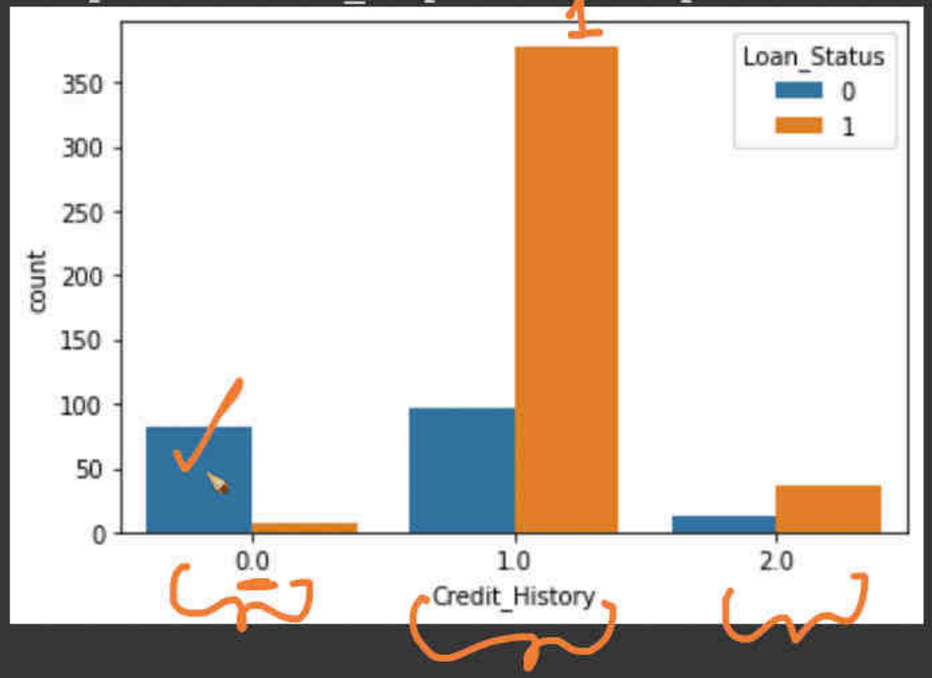
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e2eb7d950>

+ Code + Text Reconnect

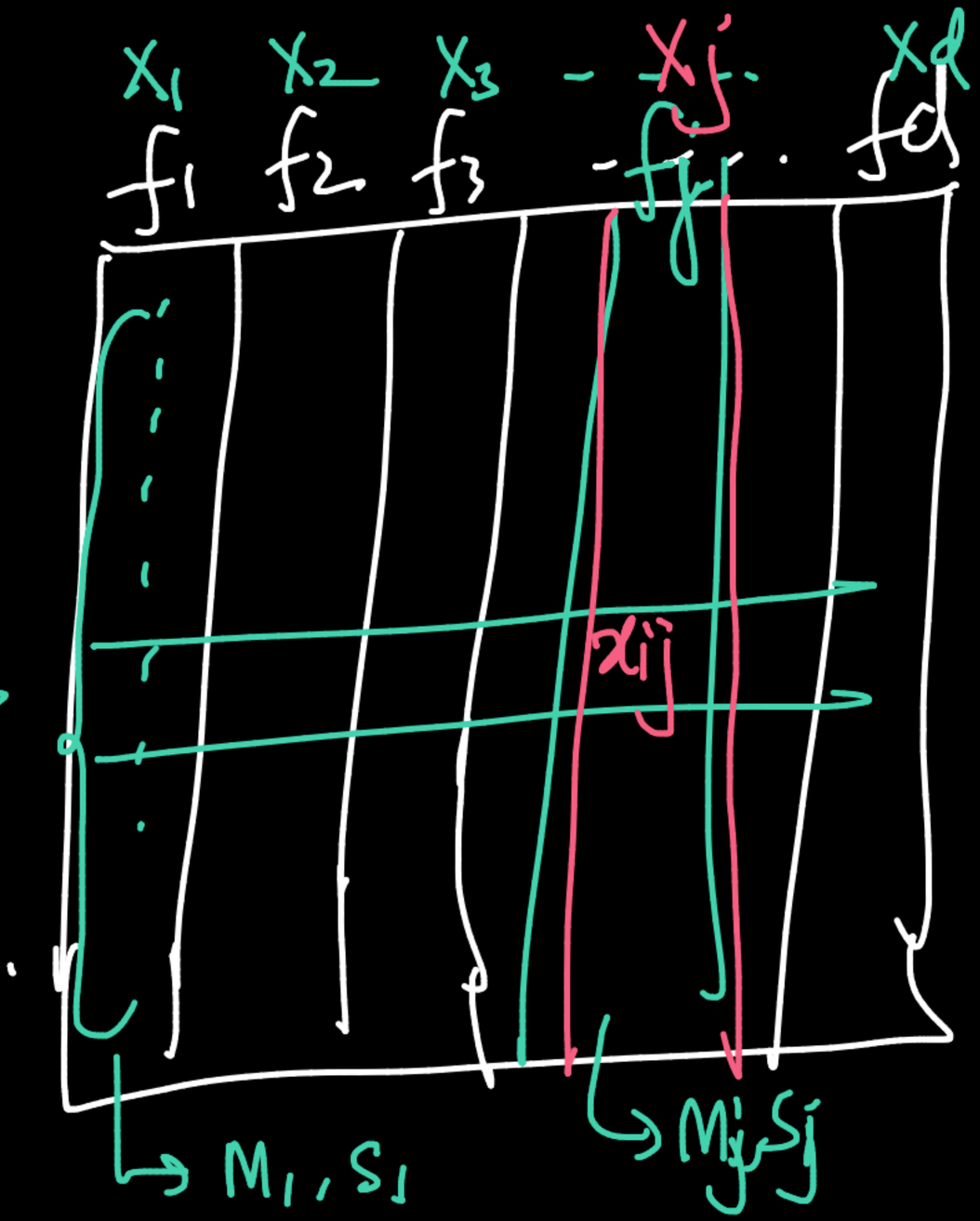



```
[ ]  sns.countplot(data =data, x = 'Credit_History', hue = 'Loan_Status')
```

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



Column Standarization and Normalization



$\frac{x_i - \mu}{\sigma}$: std

Column std

$x_{ij} = \frac{x_{ij} - M_j}{S_j}$

Col. Std → ML (optimization; FI ...)

↳ features which are in differing scales/ units

(*)

h, w

Std. normal var

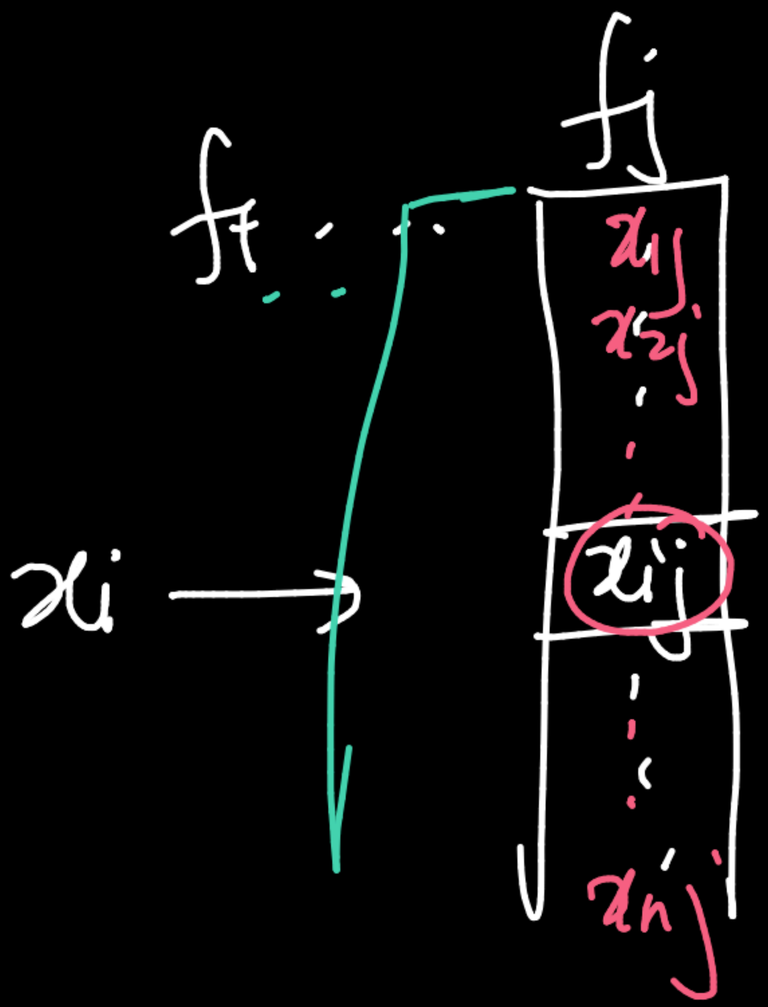
$$\frac{x - \mu}{\sigma}$$

✓ col. std / standard scaling

↓

{ mean-centering & var-scaling } per col

Min Max scaling
Normalization



filter

maxj , minj

0 & 1

$$x_{ij} = \frac{x_{ij} - Min_j}{Max_j - Min_j}$$

x_{ij} → 0 & 1

Min Max Scaling → not widely used in ML

↳ Computer vision (DL)

Column Standarization and Normalization

- Mean centering and Variance scaling (Stndard 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 (Stndard 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	0.1
1	0.472343	0.728816	0.253470	-0.528362	-0.174052	-0.134412	-0.038732	-0.211241	0.1
2	0.472343	0.728816	-0.737806	-0.528362	-0.586643	-0.393747	-0.554487	-0.948996	0.1
3	0.472343	0.728816	-0.737806	1.892641	-0.174052	-0.462062	0.251980	-0.306435	0.1
4	0.472343	-1.372089	-0.737806	-0.528362	-0.174052	0.097728	-0.554487	-0.056551	0.1

$$X' = \frac{X - \mu}{\sigma}$$

> any-dist

$\mu = 0$

$\sigma = 1$

std:

↳ mean = 0

std = 1

for each column

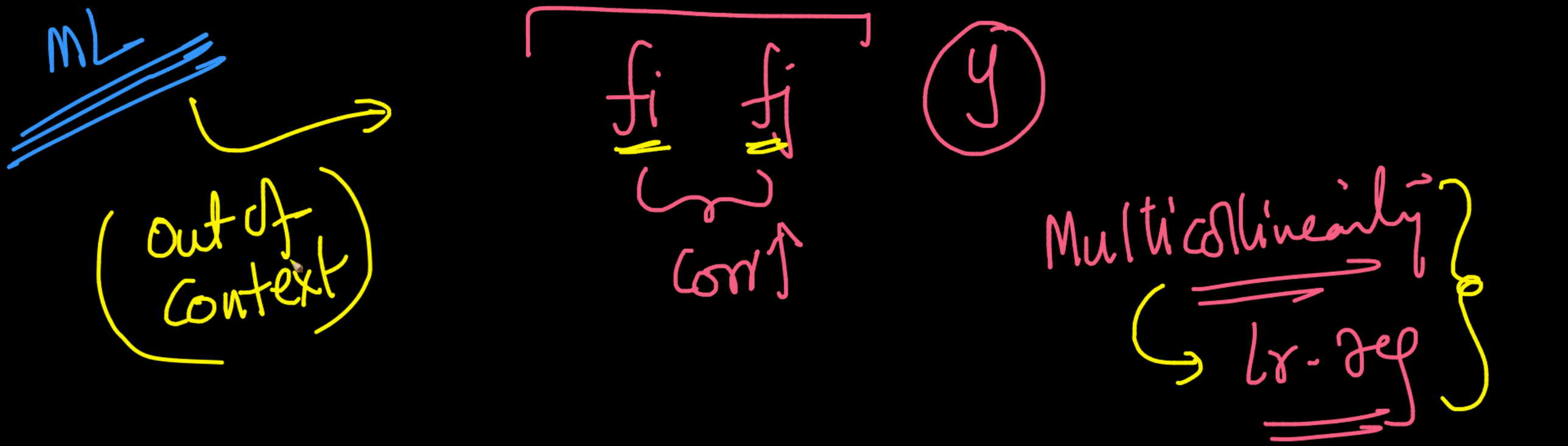
most

norm

↳ 0 - 1 for each col

v. few

CV


ML
out of context
fi
fj
corr
y
Multicollinearity
lr - reg

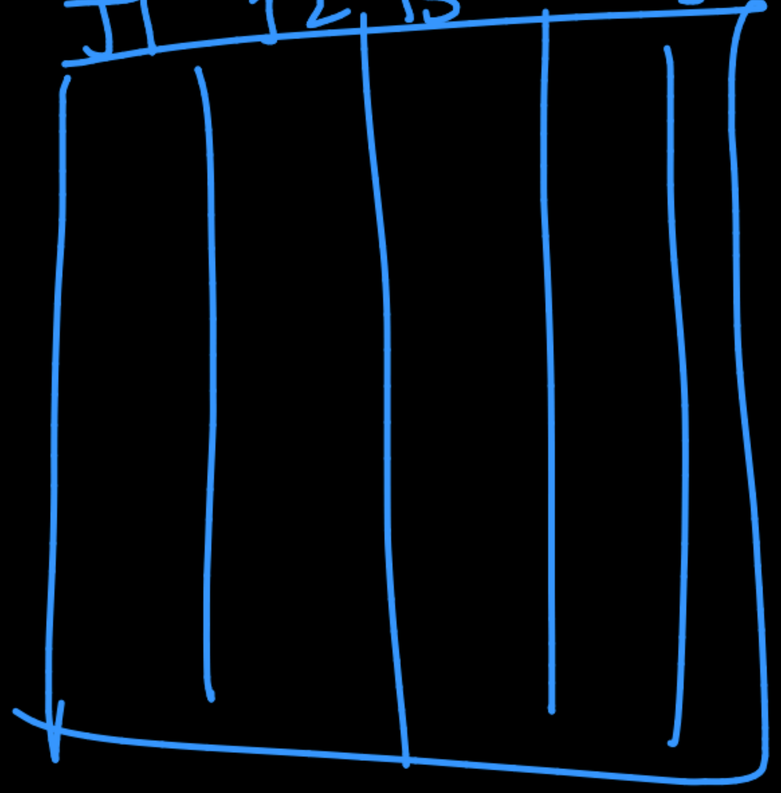
Assessments ✓

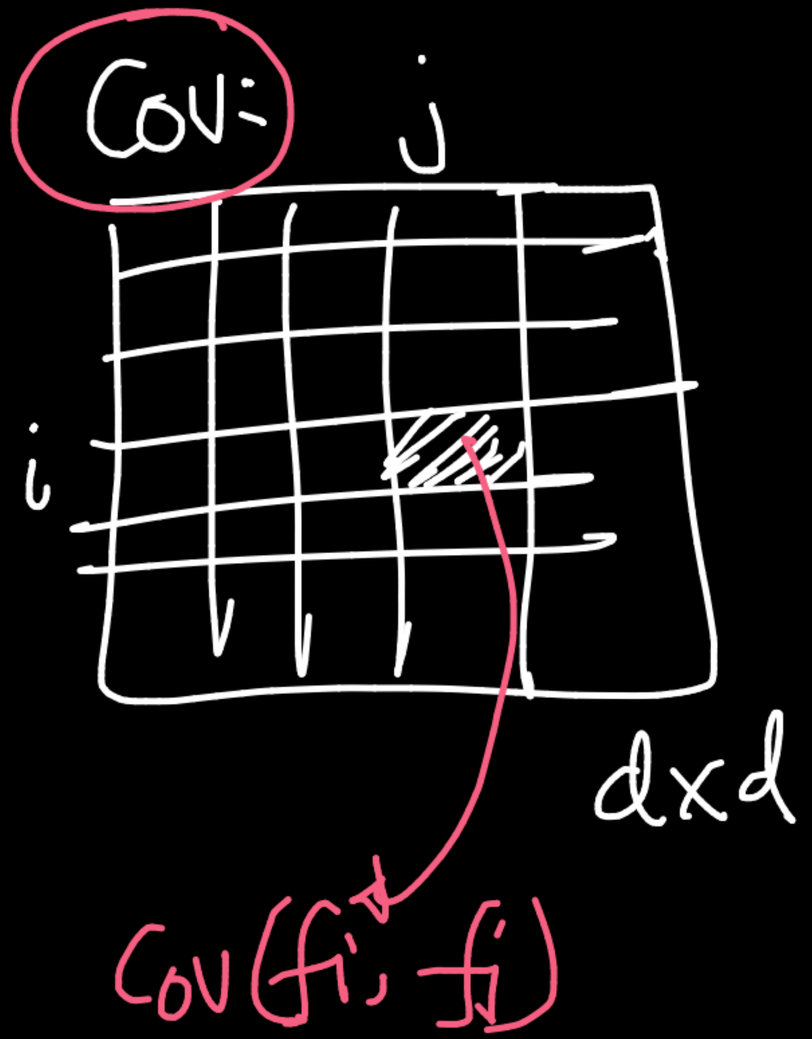
Case-studies → solve reasonably well for 1 or 2 ✓

Cov (X, Y)

f_1 f_2 f_3 ... f_d

$D =$





RANSAC

Denoising AutoEncoders

K-NN imputation

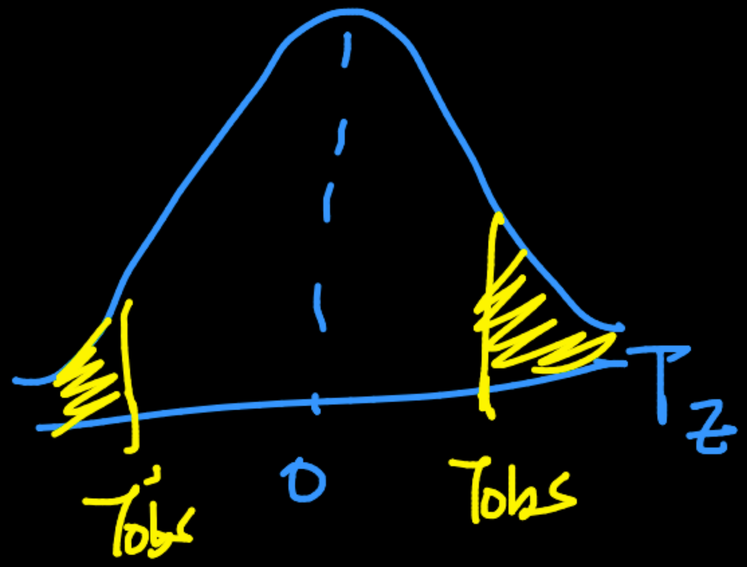
Outliers → Model

TTS or MWF

(Q)

2-tailed z/t test

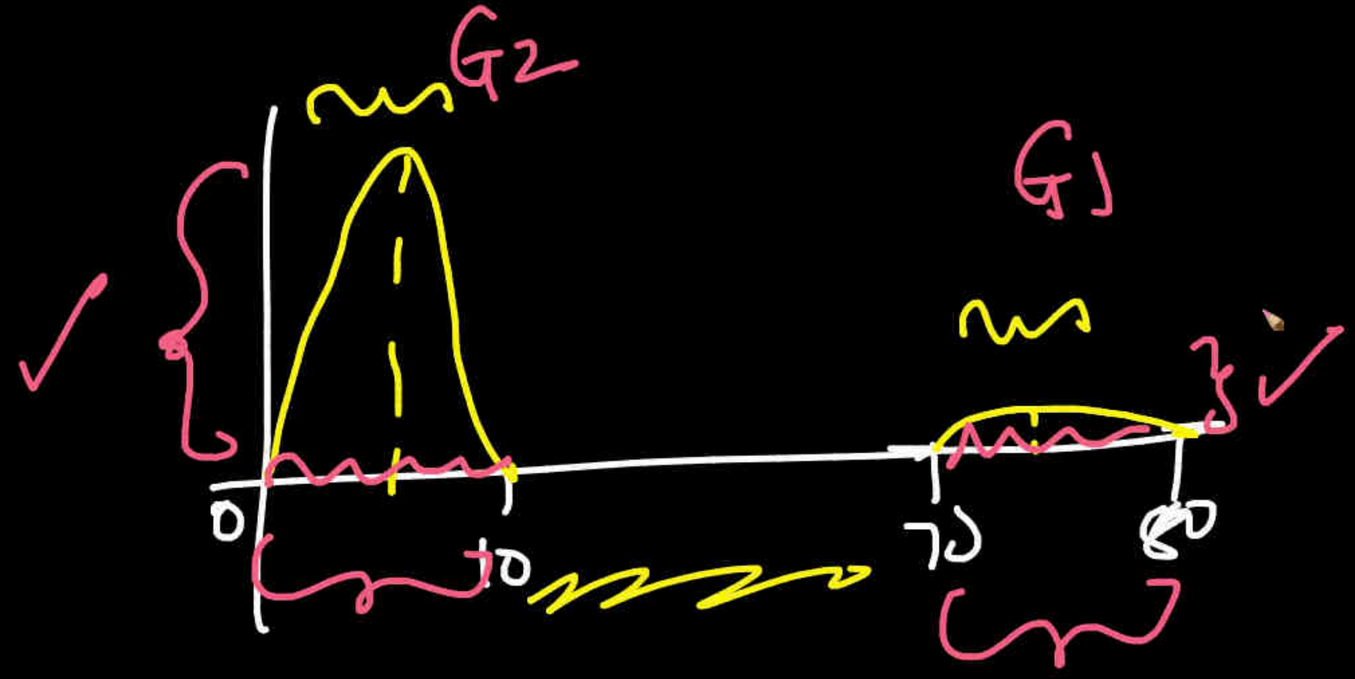
$T_{obs} = \frac{M_1 - M_2}{\sqrt{\cdots}}$ or $\frac{M_2 - M_1}{\sqrt{\cdots}} = T_{obs}'$



p-value: 2 tails

99% — (0 -10)

1%. — (70-80)



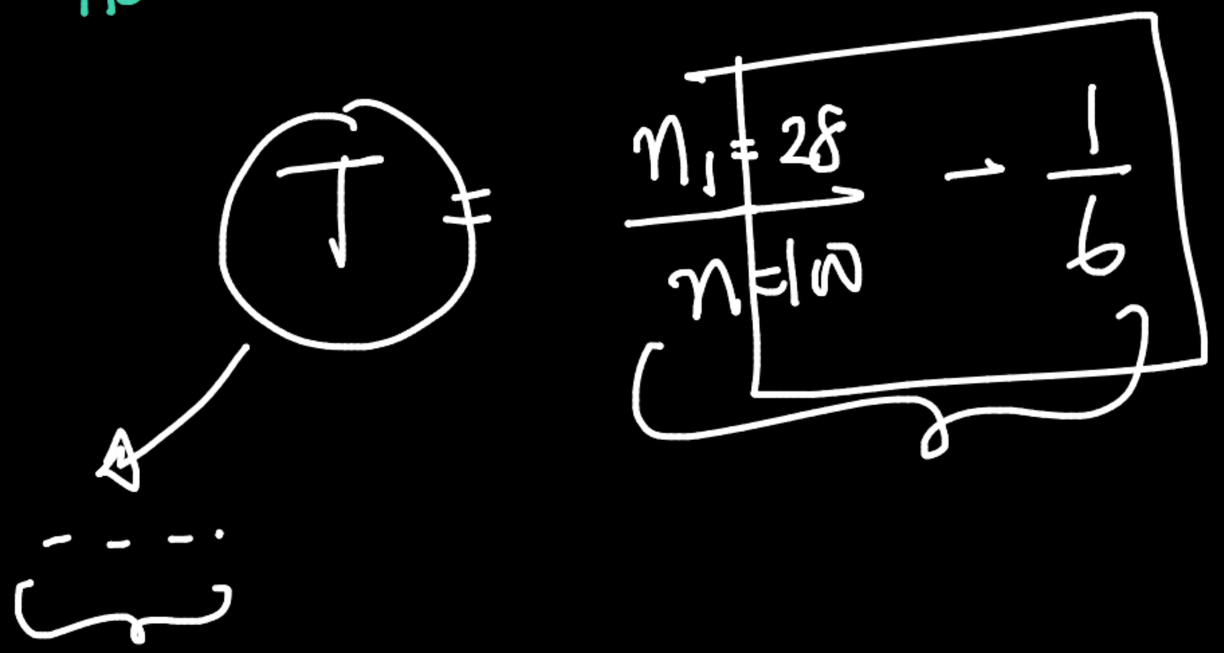
dice

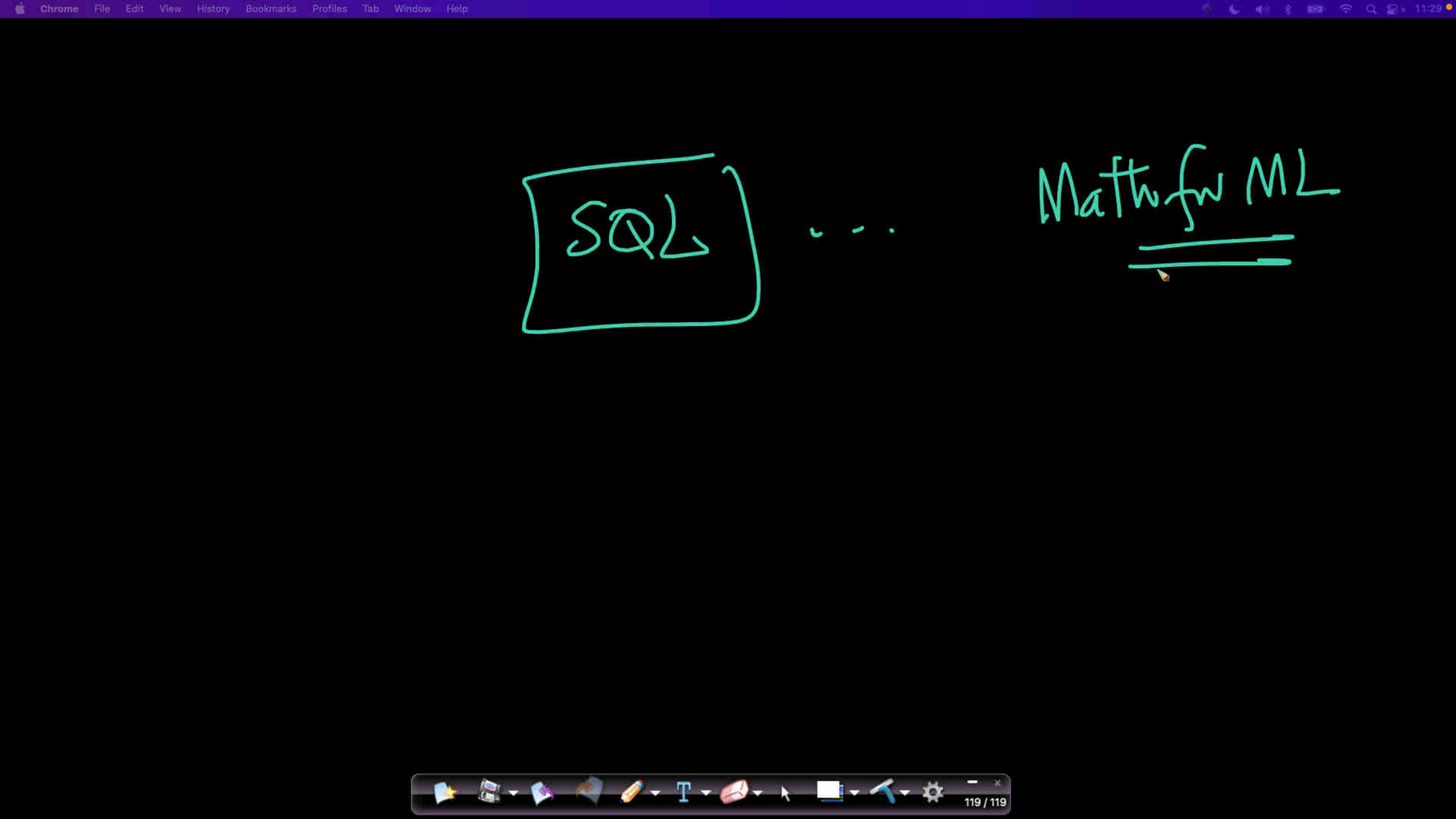
$\frac{1}{6}$

H_0: not biased

H_a: biased towards 1

1 vs not one

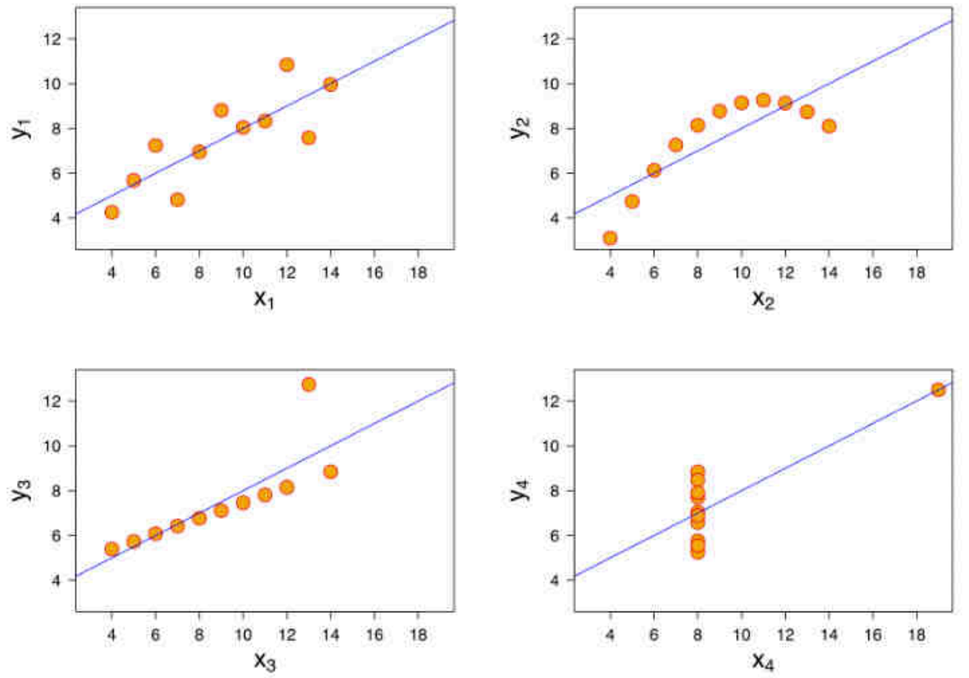



Chrome File Edit View History Bookmarks Profiles Tab Window Help
SQL
. . .
Math for ML
119 / 119

Anscombe's quartet

From Wikipedia, the free encyclopedia

Anscombe's quartet comprises four data sets that have nearly identical simple descriptive statistics, yet have very different distributions and appear very different when graphed. Each dataset consists of eleven (*x*,*y*) points. They were constructed in 1973 by the statistician Francis Anscombe to demonstrate both the importance of graphing data when analyzing it, and the effect of outliers and other influential observations on statistical properties. He described the article as being intended to counter the impression among statisticians that "numerical calculations are exact, but graphs are rough."[1]



All four sets are identical when examined using simple summary siderably when graphed

Contents [hide]

120 / 120

Data [edit]