Raw Data to Clean Data Conversion using Python EDA

Once we receive a data set, the 1st step is Data Cleaning

In [3]:
```
import pandas as pd
```

In [4]:
```
pd.__version__
```

Out[4]:
```
'2.2.2'
```

Reading the dataframe

In [6]:
```
emp = pd.read_excel(r'D:\Full Stack Data Scientist and AI\March 27 - EDA Practic
```

In case if we get import error use-

! pip install --upgrade openpyxl

In [8]:
```
emp
```

Out[8]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience#$	34 years	Mumbai	5^00#0	2+
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year
5	Kim	NLP	55yr	Delhi	6000^$0	10+

In [9]:
```
id(emp)   # Address of memory Location
```

Out[9]:
```
1962707824768
```

In [10]:
```
emp.columns
```

Out[10]:
```
Index(['Name', 'Domain', 'Age', 'Location', 'Salary', 'Exp'], dtype='object')
```

In [11]:
```
emp.shape   #6 rows and 6 columns
```

Out[11]:
```
(6, 6)
```

In [12]:
```
emp.head()  # Top 5 rows
```

Out[12]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience#$	34 years	Mumbai	5^00#0	2+
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year

In [13]:
```
emp.tail()    # Bottom 5 rows
```

Out[13]:

	Name	Domain	Age	Location	Salary	Exp
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year
5	Kim	NLP	55yr	Delhi	6000^$0	10+

In [14]:
```
emp.info()     #Information of the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       4 non-null      object
 3   Location  4 non-null      object
 4   Salary    6 non-null      object
 5   Exp       5 non-null      object
dtypes: object(6)
memory usage: 420.0+ bytes
```

- **non-null** means no null values
- Age, Location and Exp has Null values - Age(2 null values), Location(2 null values) and Exp(1 null value)
- memory usage: 420.0+ bytes. Bec it has less data

In [16]:
```
emp
#this is raw data
#whereever we have missing values in the excel, it is replaced by NaN
```

Out[16]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience#$	34 years	Mumbai	5^00#0	2+
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year
5	Kim	NLP	55yr	Delhi	6000^$0	10+

In [17]:
```
emp.isnull()      #Hey python do you find any null values in this dataframe
```

Out[17]:

	Name	Domain	Age	Location	Salary	Exp
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	True	True	False	False
3	False	False	True	False	False	True
4	False	False	False	True	False	False
5	False	False	False	False	False	False

In [18]:
```
emp.isna()   #same as isnull()
```

Out[18]:

	Name	Domain	Age	Location	Salary	Exp
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	True	True	False	False
3	False	False	True	False	False	True
4	False	False	False	True	False	False
5	False	False	False	False	False	False

In [19]:
```
emp.isnull().sum()
```

Out[19]:
```
Name        0
Domain      0
Age         2
Location    2
Salary      0
Exp         1
dtype: int64
```

In [20]:
```
emp.columns
```

Out[20]:
```
Index(['Name', 'Domain', 'Age', 'Location', 'Salary', 'Exp'], dtype='object')
```

In [21]:
```
emp
```

Out[21]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience#$	34 years	Mumbai	5^00#0	2+
1	Teddy^	Testing	45' yr	Bangalore	10%%000	<3
2	Uma#r	Dataanalyst^^#	NaN	NaN	1$5%000	4> yrs
3	Jane	Ana^^lytics	NaN	Hyderbad	2000^0	NaN
4	Uttam*	Statistics	67-yr	NaN	30000-	5+ year
5	Kim	NLP	55yr	Delhi	6000^$0	10+

Data Cleaning or Data Cleansing

- I want to clean every attribute one by one
- replace() - for special characters | extract() - for digits

In [24]:
```
emp['Name']    #  ^#* ->These are regular expressions
```

Out[24]:
```
0      Mike
1    Teddy^
2     Uma#r
3      Jane
4    Uttam*
5       Kim
Name: Name, dtype: object
```

In [25]:
```
emp['Name'] = emp['Name'].str.replace(r'\W','', regex=True)

# '\W' - non-word character (Special Characters/Wild Character)
# '' - Fill with empty
```

In [26]:
```
emp['Name']
```

Out[26]:
```
0     Mike
1    Teddy
2     Umar
3     Jane
4    Uttam
5      Kim
Name: Name, dtype: object
```

Now data is cleansed

In [28]:
```
emp['Domain']
```

Out[28]:
```
0     Datascience#$
1           Testing
2    Dataanalyst^^#
3       Ana^^lytics
4        Statistics
5               NLP
Name: Domain, dtype: object
```

In [29]:
```
emp['Domain'] = emp['Domain'].str.replace(r'\W','', regex=True)
```

In [30]:
```
emp['Domain']
```

Out[30]:
```
0    Datascience
1        Testing
2    Dataanalyst
3      Analytics
4     Statistics
5            NLP
Name: Domain, dtype: object
```

In [31]:
```
emp
```

Out[31]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34 years	Mumbai	5^00#0	2+
1	Teddy	Testing	45' yr	Bangalore	10%%000	<3
2	Umar	Dataanalyst	NaN	NaN	1$5%000	4> yrs
3	Jane	Analytics	NaN	Hyderbad	2000^0	NaN
4	Uttam	Statistics	67-yr	NaN	30000-	5+ year
5	Kim	NLP	55yr	Delhi	6000^$0	10+

In [32]:
```
emp['Age'] = emp['Age'].str.replace(r'\W','', regex=True)
```

In [33]:
```
emp['Age']  #age is a number
```

Out[33]:
```
0    34years
1       45yr
2        NaN
3        NaN
4       67yr
5       55yr
Name: Age, dtype: object
```

In [34]:
```
#r(r'(\d+)') extracts digits
#r(r'(\\d+)') -> Use double slash, if getting error

#emp['Age'] = emp['Age'].str.extract('(\d+)')

emp['Age'] = emp['Age'].str.extract(r'(\d+)')
```

I got syntax error, when executing - #emp['Age'] = emp['Age'].str.extract('(\d+)') happens because in Python strings, backslashes () are escape characters. So \d is interpreted as a special escape sequence, but \d by itself is not a valid one in standard Python strings.

Use a raw string by adding an r prefix. r'(\d+)' tells Python to treat the backslash literally, not as an escape character.

\d+ is a regular expression that matches one or more digits.

In [36]:
```
emp['Age']
```

Out[36]:
```
0     34
1     45
2    NaN
3    NaN
4     67
5     55
Name: Age, dtype: object
```

In [37]:
```
emp
```

Out[37]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5^00#0	2+
1	Teddy	Testing	45	Bangalore	10%%000	<3
2	Umar	Dataanalyst	NaN	NaN	1$5%000	4> yrs
3	Jane	Analytics	NaN	Hyderbad	2000^0	NaN
4	Uttam	Statistics	67	NaN	30000-	5+ year
5	Kim	NLP	55	Delhi	6000^$0	10+

In [38]:
```
emp['Location']
```

Out[38]:
```
0       Mumbai
1    Bangalore
2          NaN
3     Hyderbad
4          NaN
5        Delhi
Name: Location, dtype: object
```

There are no special character, still apply replace(), in case if any space is there

In [40]:
```
emp['Location'] = emp['Location'].str.replace(r'\W','', regex=True)
```

In [41]:
```
emp['Location']
```

Out[41]:
```
0       Mumbai
1    Bangalore
2          NaN
3     Hyderbad
4          NaN
5        Delhi
Name: Location, dtype: object
```

In [42]:
```
emp['Salary']
```

Out[42]:
```
0     5^00#0
1    10%%000
2    1$5%000
3     2000^0
4     30000-
5    6000^$0
Name: Salary, dtype: object
```

In [43]:
```
emp['Salary'] = emp['Salary'].str.replace(r'\W','', regex=True)
```

In [44]:
```
emp['Salary']
```

Out[44]:
```
0     5000
1    10000
2    15000
3    20000
4    30000
5    60000
Name: Salary, dtype: object
```

- Error message - It will not clean
- Warning message - It will clean

In [46]:
```
emp
```

Out[46]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2+
1	Teddy	Testing	45	Bangalore	10000	<3
2	Umar	Dataanalyst	NaN	NaN	15000	4> yrs
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5+ year
5	Kim	NLP	55	Delhi	60000	10+

In [47]:
```
emp['Exp']
```

Out[47]:
```
0        2+
1        <3
2     4> yrs
3       NaN
4    5+ year
5       10+
Name: Exp, dtype: object
```

In [48]:
```
emp['Exp'] = emp['Exp'].str.extract(r'(\d+)')
```

In [49]:
```
emp['Exp']
```

Out[49]:
```
0      2
1      3
2      4
3    NaN
4      5
5     10
Name: Exp, dtype: object
```

In Company for more attributes, we need to write a loop

- str.extract(r'(\d+)') - same applies to 10k records. Concept is same

In [51]:
```
emp
```

Out[51]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

Now all special characters are removed

In [53]:
```
clean_data = emp.copy()   #New variable is created
```

In [54]:
```
clean_data
```

Out[54]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

- Till now we have raw data and we used regex to clean the data and removed all noisy characters(#$^ etc.) from the dataset
- You can also work in the same way in sql query as well

EDA Techniques

- Missing Values Treatment for Numerical Data

In [57]:
```
clean_data.isnull().sum()
```

Out[57]:
```
Name        0
Domain      0
Age         2
Location    2
Salary      0
Exp         1
dtype: int64
```

- Thus, 3 attributes have missing value
- We have to clean Age, Exp and Location
- Age and Exp are numerical attribute

In [59]:
```
clean_data
```

Out[59]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [60]:
```
clean_data['Age']
```

Out[60]:
```
0     34
1     45
2    NaN
3    NaN
4     67
5     55
Name: Age, dtype: object
```

In [61]:
```
import numpy as np   #Without Numpy we can't impute
```

In [62]:
```
clean_data['Age'] = clean_data['Age'].fillna(np.mean(pd.to_numeric(clean_data['A
```

In [63]:
```
clean_data['Age']
```

Out[63]:
```
0       34
1       45
2    50.25
3    50.25
4       67
5       55
Name: Age, dtype: object
```

- Now missing values are filled
- .isna -> Missing value
- .fillna -> Fills Missing value with Mean Strategy
- np.mean -> Calculates mean
- pd.to_numeric -> Pandas to numeric as by default it takes object

In [65]:
```
clean_data['Exp']
```

Out[65]:
```
0      2
1      3
2      4
3    NaN
4      5
5     10
Name: Exp, dtype: object
```

In [66]:
```
clean_data['Exp'] = clean_data['Exp'].fillna(np.mean(pd.to_numeric(clean_data['E
```

In [67]:
```
clean_data['Exp']
```

Out[67]:
```
0      2
1      3
2      4
3    4.8
4      5
5     10
Name: Exp, dtype: object
```

In [68]:
```
clean_data
```

Out[68]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50.25	NaN	15000	4
3	Jane	Analytics	50.25	Hyderbad	20000	4.8
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [69]:
```
clean_data['Location'].isnull().sum()
```

Out[69]:
```
2
```

In [70]:
```
clean_data['Location']
```

Out[70]:
```
0       Mumbai
1    Bangalore
2          NaN
3     Hyderbad
4          NaN
5        Delhi
Name: Location, dtype: object
```

In [71]:
```
clean_data['Location'] = clean_data['Location'].fillna(clean_data['Location'].mo
```

- .mode()[0] --> Mode strategy. I have to remember this like a formula and have to pass [0] at the end. [0] is mandatory. Mean no problem, but for mode we have to pass 0, do not write 1
- If 0 not written then code will not work. 0 is the index.
- .mode()[0] is applicable only for Categorical Values, not Numerical

In [73]:
```
clean_data['Location']
```

Out[73]:
```
0       Mumbai
1    Bangalore
2    Bangalore
3     Hyderbad
4    Bangalore
5        Delhi
Name: Location, dtype: object
```

Bangalore is picked by Inbuilt parameters and filled inplace of NaN

In [75]:
```
clean_data
```

Out[75]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50.25	Bangalore	15000	4
3	Jane	Analytics	50.25	Hyderbad	20000	4.8
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [76]:
```
emp.info()   #Earlier there wee missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       4 non-null      object
 3   Location  4 non-null      object
 4   Salary    6 non-null      object
 5   Exp       5 non-null      object
dtypes: object(6)
memory usage: 420.0+ bytes
```

In [77]:
```
clean_data.info()      #Now no missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       6 non-null      object
 3   Location  6 non-null      object
 4   Salary    6 non-null      object
 5   Exp       6 non-null      object
dtypes: object(6)
memory usage: 420.0+ bytes
```

- Now data is cleansed.
- All are objects now. But Name, Domain and Location are Category.

Interview ques-> In your project how do you fill missing data in your project?

- Answer - As above

- The above clean data is stored in the system memory. Cannot share Jupiter notebook with boss to present the clean data.

Converting System datatype to User-defined data type

- Object to Integer

- .astype(int) -> For converting System datatype to User-defined data type. If not converted, it will create issue.

In [296…
```
clean_data['Age'] = clean_data['Age'].astype(int)
```

In [298…
```
clean_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Name      6 non-null      category
 1   Domain    6 non-null      category
 2   Age       6 non-null      int32
 3   Location  6 non-null      category
 4   Salary    6 non-null      int32
 5   Exp       6 non-null      int32
dtypes: category(3), int32(3)
memory usage: 866.0 bytes
```

Interview Ques -> Write down data cleaning pipeline/steps. Tell same as above as it is, interview will clear

In [86]:
```
clean_data['Salary'] = clean_data['Salary'].astype(int)
```

In [87]:
```
clean_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       6 non-null      int32
 3   Location  6 non-null      object
 4   Salary    6 non-null      int32
 5   Exp       6 non-null      object
dtypes: int32(2), object(4)
memory usage: 372.0+ bytes
```

In [88]:
```
clean_data['Exp'] = clean_data['Exp'].astype(int)
```

In [89]:
```
clean_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Name      6 non-null      object
 1   Domain    6 non-null      object
 2   Age       6 non-null      int32
 3   Location  6 non-null      object
 4   Salary    6 non-null      int32
 5   Exp       6 non-null      int32
dtypes: int32(3), object(3)
memory usage: 348.0+ bytes
```

- Now Age, Salary and Exp is converted into Integer

Converting Object to Category

In [92]:
```
clean_data['Name'] = clean_data['Name'].astype('category')
clean_data['Domain'] = clean_data['Domain'].astype('category')
clean_data['Location'] = clean_data['Location'].astype('category')
```

In [93]:
```
clean_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype   
---  ------    --------------  -----   
 0   Name      6 non-null      category
 1   Domain    6 non-null      category
 2   Age       6 non-null      int32   
 3   Location  6 non-null      category
 4   Salary    6 non-null      int32   
 5   Exp       6 non-null      int32   
dtypes: category(3), int32(3)
memory usage: 866.0 bytes
```

- Name, Domain, Location is Category
- Age, Salary, Exp is int32

In [94]:
```
clean_data
```

Out[94]:

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

To share this data with the manager, we have to show it in Excel sheet as client does not have Jupiter, nor he know Python. So, we have to convert Python file into Excel

Missing Value Treatement Completed

In [168…
```
clean_data.to_csv('clean_data.csv')  #Now clean data is stored in my system
```

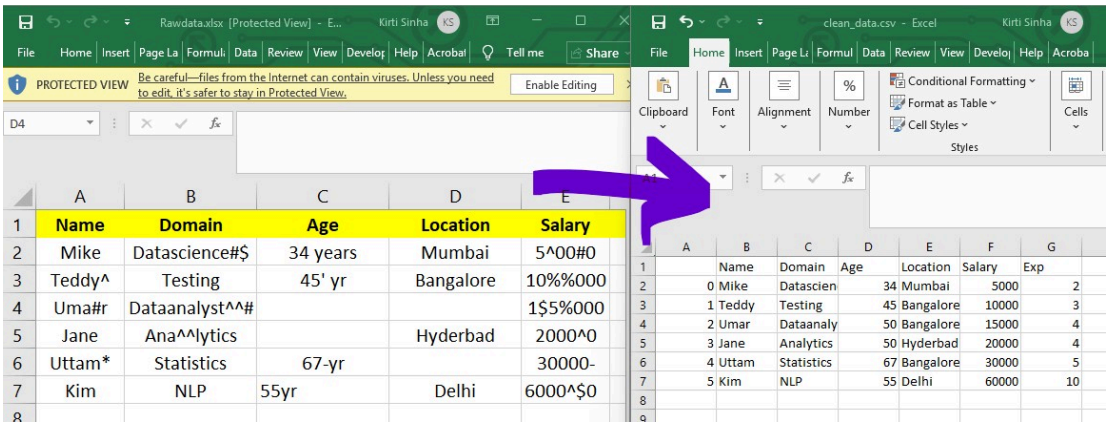
In [178…
```
#To find in which directory, the above file is saved. Path Location

import os
os.getcwd()     #.getcwd() -> Get Current Working Directory
```

Out[178…
```
'C:\\Users\\kirti'
```

In [176…
```
clean_data
```

Out[176...

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10



- This Conversion of Raw Data to Clean data is called Data Cleaning.
- From Clean Data, we will apply Imputations and build machine Learning Model
- Steps -

1.) Raw Data

2.) Clean Data

3.) Transformer

4.) Machine Learning

To understand these steps Python basics are required.

In [191...
```
import matplotlib.pyplot as plt   # Visualization
import seaborn as sns             # Advanced Visualization
```

In [193...
```
# To ignore warnings

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

In [195...
```
clean_data
```

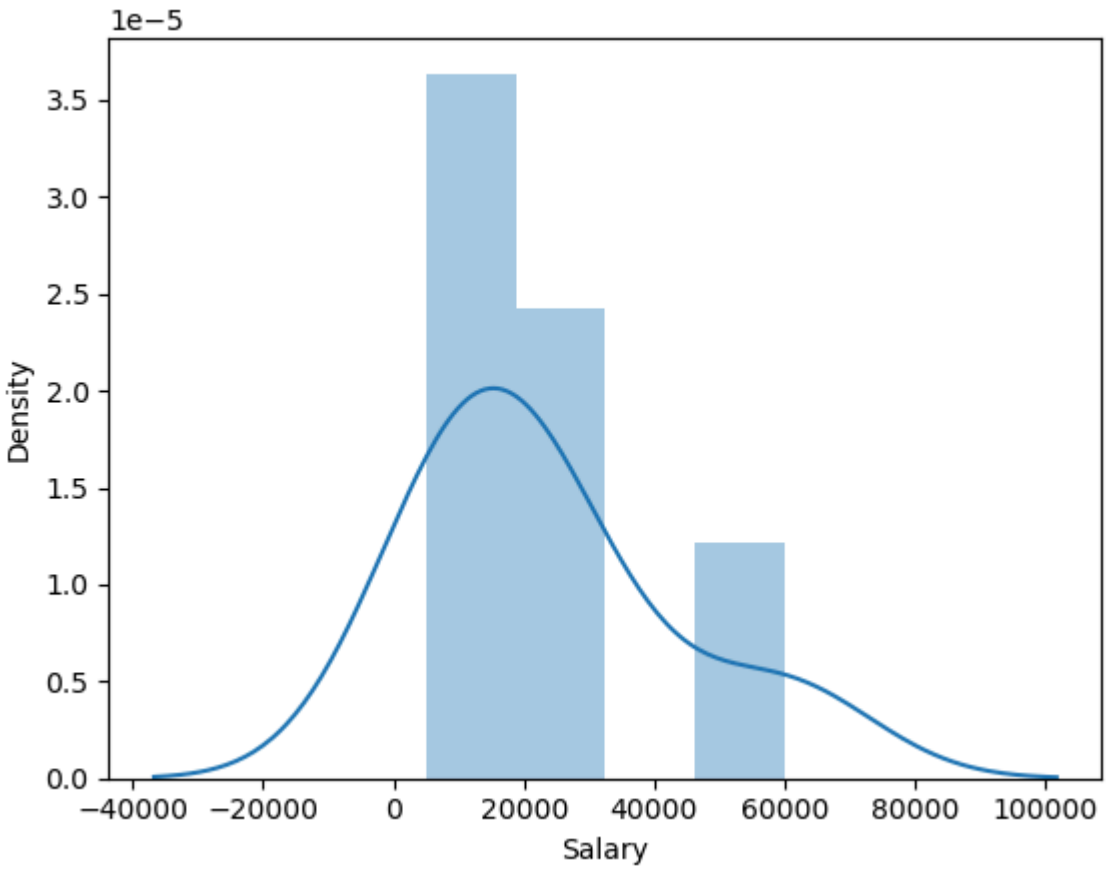
Out[195…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

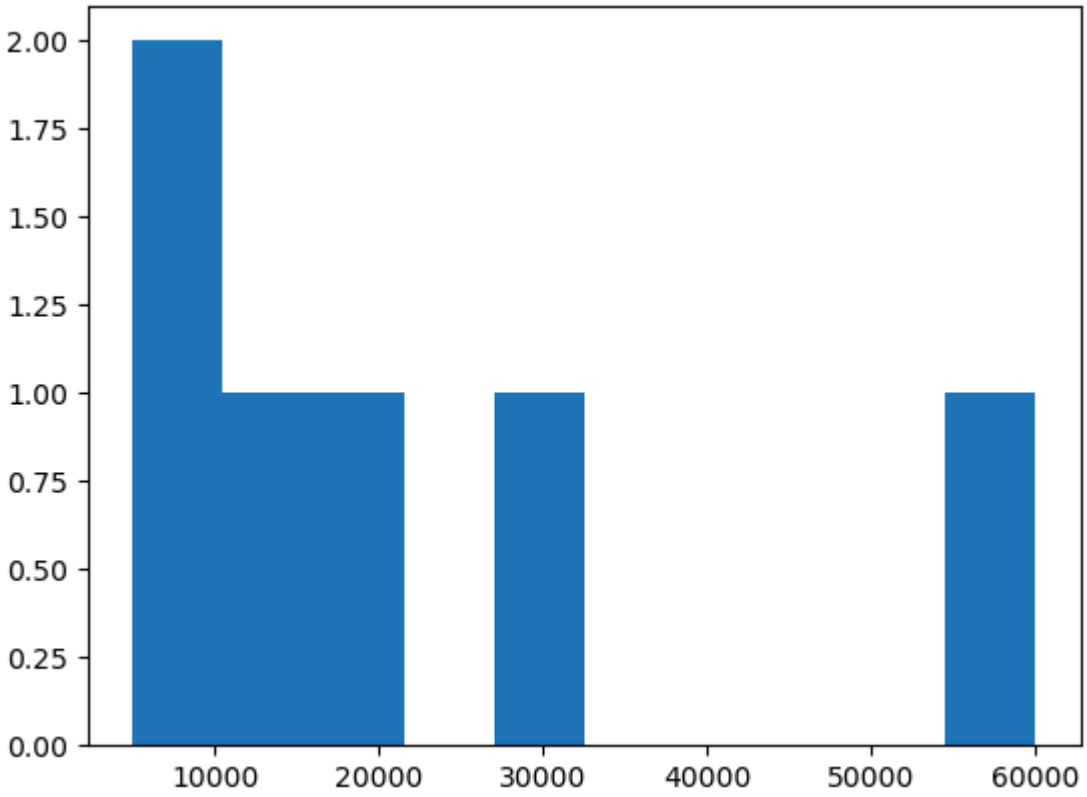
In [197…
```
clean_data['Salary']
```

Out[197…
```
0     5000
1    10000
2    15000
3    20000
4    30000
5    60000
Name: Salary, dtype: int32
```

In [203…
```
vis1 = sns.distplot(clean_data['Salary'])  #Univariate analysis Distribution Plo

#Plotting the graph using 1 variable is called Univariate analysis
# minimum employee salary is 5000, maximum is 60,000
```



In [201…
```
vis2 = plt.hist(clean_data['Salary'])       # #Univariate analysis. This plots h
```

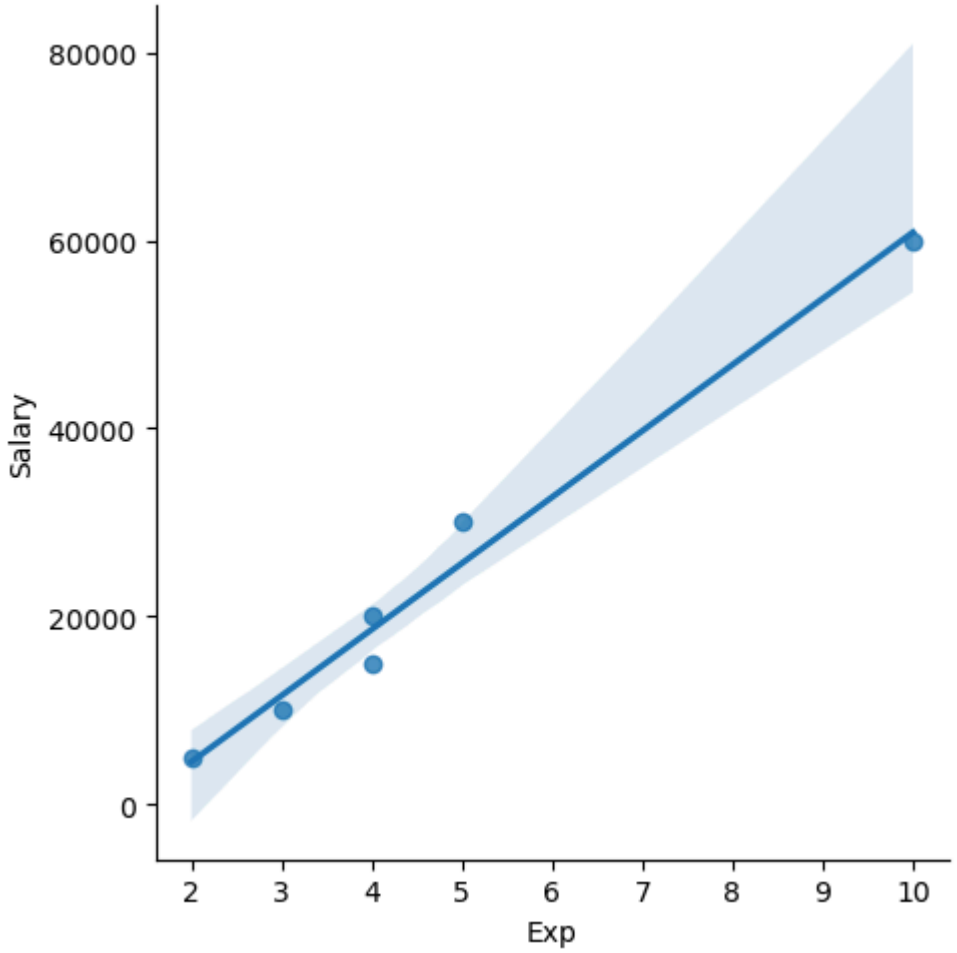


In [205…
```
clean_data
```

Out[205…

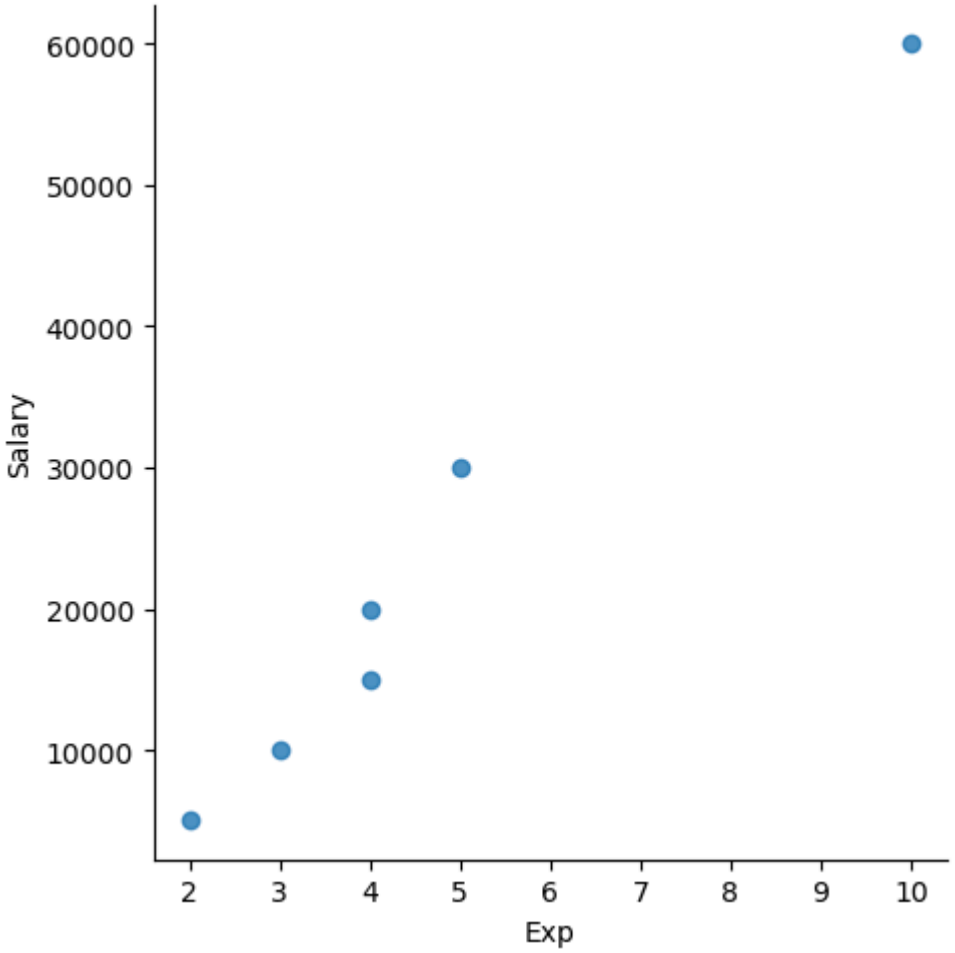
	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [215…
```
vis3 = sns.lmplot(data = clean_data, x = 'Exp', y = 'Salary') # Bivariate Analys

# lmplot -> Linear Plot, 2yr exp : salary = 5k, 3yr exp: Salary = 10k. It is a p
# range is 0 to 1
# There are 1 outlier -> the last one
```



In [211…

```
vis3 = sns.lmplot(data = clean_data, x = 'Exp', y = 'Salary', fit_reg = False)

# fit_reg = False removes the linear line
```



In [217…
```
clean_data[:]          # Slicing
```

Out[217…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [219…
```
clean_data[0:6:2]
```

Out[219…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
2	Umar	Dataanalyst	50	Bangalore	15000	4
4	Uttam	Statistics	67	Bangalore	30000	5

In [221…
```
clean_data[::-1]
```

Out[221…

	Name	Domain	Age	Location	Salary	Exp
5	Kim	NLP	55	Delhi	60000	10
4	Uttam	Statistics	67	Bangalore	30000	5
3	Jane	Analytics	50	Hyderbad	20000	4
2	Umar	Dataanalyst	50	Bangalore	15000	4
1	Teddy	Testing	45	Bangalore	10000	3
0	Mike	Datascience	34	Mumbai	5000	2

In [223…
```
clean_data.columns
```

Out[223…
```
Index(['Name', 'Domain', 'Age', 'Location', 'Salary', 'Exp'], dtype='object')
```

Dependent Variable

In [225…
```
clean_data
```

Out[225…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

Salary is the Dependent Variable.

- We can create multiple linear equations -> y = x1 + x2 + x3
- This is regression because dependent variable is of continuous nature -> 5k, 10k, 15k...
- Thus, we apply regression algorithm and not classification

Besides Salary all are Independent Variable

In [238…
```
X_iv = clean_data[['Name', 'Domain', 'Age', 'Location', 'Exp']]  #x is the indep
```

In [240…
```
X_iv
```

Out[240…

	Name	Domain	Age	Location	Exp
0	Mike	Datascience	34	Mumbai	2
1	Teddy	Testing	45	Bangalore	3
2	Umar	Dataanalyst	50	Bangalore	4
3	Jane	Analytics	50	Hyderbad	4
4	Uttam	Statistics	67	Bangalore	5
5	Kim	NLP	55	Delhi	10

The above variables are Independent Variables

In [243…
```
y_dv = clean_data[['Salary']]     #y dependent variable
```

In [245…
```
y_dv
```

Out[245…

	Salary
0	5000
1	10000
2	15000
3	20000
4	30000
5	60000

Variables created so far ->

In [247…
```
emp   #We have Null Values in Emp
```

Out[247…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	NaN	NaN	15000	4
3	Jane	Analytics	NaN	Hyderbad	20000	NaN
4	Uttam	Statistics	67	NaN	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [250…
```
clean_data    #This is a Cleans data set
```

Out[250…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [256…
```
X_iv          #Only Independent Variables
```

Out[256…

	Name	Domain	Age	Location	Exp
0	Mike	Datascience	34	Mumbai	2
1	Teddy	Testing	45	Bangalore	3
2	Umar	Dataanalyst	50	Bangalore	4
3	Jane	Analytics	50	Hyderbad	4
4	Uttam	Statistics	67	Bangalore	5
5	Kim	NLP	55	Delhi	10

In [258…
```
y_dv          #Dependent variable
```

Out[258…

	Salary
0	5000
1	10000
2	15000
3	20000
4	30000
5	60000

In [260…
```
clean_data
```

Out[260…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

Machine + Learning = Machine Learning

- Machine does not understand Mike, Teddy etc.
- Thus, Machine will Imputate it
- Let's apply Transformers

In [271…
```
imputation = pd.get_dummies(clean_data, dtype = int)  #.get_dummies -> Create a
```

In [285…
```
imputation

# We feed the below data in the form of 0 and 1 to Machine learning as all the v
```

Out[285…

	Age	Salary	Exp	Name_Jane	Name_Kim	Name_Mike	Name_Teddy	Name_Umar
0	34	5000	2	0	0	1	0	0
1	45	10000	3	0	0	0	1	0
2	50	15000	4	0	0	0	0	1
3	50	20000	4	1	0	0	0	0
4	67	30000	5	0	0	0	0	0
5	55	60000	10	0	1	0	0	0

- Now columns are increased. There were 6 names, thus 6 columns/variables are created.
- Name_Mike and Mike is 1, rest of them are 0
- Name_Teddy and Teddy is 1, rest of them 0
- Domain_Datascience and Datascience is 1, rest of them are 0
- Location_Delhi and Delhi is 1, rest of them are 0

In [287…
```
clean_data
```

Out[287…

	Name	Domain	Age	Location	Salary	Exp
0	Mike	Datascience	34	Mumbai	5000	2
1	Teddy	Testing	45	Bangalore	10000	3
2	Umar	Dataanalyst	50	Bangalore	15000	4
3	Jane	Analytics	50	Hyderbad	20000	4
4	Uttam	Statistics	67	Bangalore	30000	5
5	Kim	NLP	55	Delhi	60000	10

In [289…
```
len(clean_data)
```

Out[289…
```
6
```

In [291…
```
imputation.columns
```

Out[291…
```
Index(['Age', 'Salary', 'Exp', 'Name_Jane', 'Name_Kim', 'Name_Mike',
       'Name_Teddy', 'Name_Umar', 'Name_Uttam', 'Domain_Analytics',
       'Domain_Dataanalyst', 'Domain_Datascience', 'Domain_NLP',
       'Domain_Statistics', 'Domain_Testing', 'Location_Bangalore',
       'Location_Delhi', 'Location_Hyderbad', 'Location_Mumbai'],
      dtype='object')
```

In [293…
```
len(imputation.columns)
```

Out[293…
```
19
```

In [ ]: