Assignment-1

Data Cleaning (Pandas)

By Arijit Dhali [Linkedin]

This dataset helps you to increase the data-cleaning process using the pure Python pandas library.

Columns :

- Age
- Salary
- Rating
- Location
- Established
- Easy Apply
  Here are some questions that you can follow to perform data cleaning and manipulation using the given dataset.

The actual raw .csv data is given below

In [ ]:

```
import pandas as pd
url='https://raw.githubusercontent.com/ArijitDhali/PrepInsta-DA-Week-3/main/Assignment1.csv'
df=pd.read_csv(url,encoding='unicode_escape')
df
```

Out[ ]:

	Index	Age	Salary	Rating	Location	Established	Easy Apply
0	0	44.0	$44k-$99k	5.4	India,In	1999	TRUE
1	1	66.0	$55k-$66k	3.5	New York,Ny	2002	TRUE
2	2	NaN	$77k-$89k	-1.0	New York,Ny	-1	-1
3	3	64.0	$44k-$99k	4.4	India In	1988	-1
4	4	25.0	$44k-$99k	6.4	Australia Aus	2002	-1
5	5	44.0	$77k-$89k	1.4	India,In	1999	TRUE
6	6	21.0	$44k-$99k	0.0	New York,Ny	-1	-1
7	7	44.0	$44k-$99k	-1.0	Australia Aus	-1	-1
8	8	35.0	$44k-$99k	5.4	New York,Ny	-1	-1
9	9	22.0	$44k-$99k	7.7	India,In	-1	TRUE
10	10	55.0	$10k-$49k	5.4	India,In	2008	TRUE
11	11	44.0	$10k-$49k	6.7	India,In	2009	-1
12	12	NaN	$44k-$99k	0.0	India,In	1999	-1
13	13	25.0	$44k-$99k	-1.0	Australia Aus	2019	TRUE
14	14	66.0	$44k-$99k	4.0	Australia Aus	2020	TRUE
15	15	44.0	$88k-$101k	3.0	Australia Aus	1999	-1
16	16	19.0	$19k-$40k	4.5	India,In	1984	-1
17	17	NaN	$44k-$99k	5.3	New York,Ny	1943	TRUE
18	18	35.0	$44k-$99k	6.7	New York,Ny	1954	TRUE
19	19	32.0	$44k-$99k	3.3	New York,Ny	1955	TRUE
20	20	NaN	$44k-$99k	5.7	New York,Ny	1944	TRUE
21	21	35.0	$44k-$99k	5.0	New York,Ny	1946	-1
22	22	19.0	$55k-$66k	7.8	New York,Ny	1988	TRUE
23	23	NaN	$44k-$99k	2.4	New York,Ny	1999	TRUE
24	24	13.0	$44k-$99k	-1.0	New York,Ny	1987	-1
25	25	55.0	$44k-$99k	0.0	Australia Aus	1980	TRUE
26	26	NaN	$55k-$66k	NaN	India,In	1934	TRUE
27	27	52.0	$44k-$99k	5.4	India,In	1935	-1
28	28	NaN	$39k-$88k	3.4	Australia Aus	1932	-1

Problem Solutions:

↓↓↓ Expand to reveal solutions to the problems. [Orderwise]

1. Missing Values:

Question:
Are there any missing values in the dataset, and if so, how should they be handled for each indicator?
Answer:

- Yes, There are missing values in dataset. There are missing values in multiple columns (e.g., `Age, Rating, Established, Easy Apply`).
- Handling: These missing values need to be addressed, either through imputation, removal of rows, or depending on the context.
- However column named `"Easy Apply"` has `-1` which means `False`
- Whereas for columns `"Age"`, `"Rating"`, `"Established"`, we need to replace with mean of Age, mean of Rating and `"Unknown"` values respectively.

The solution to clean missing values:

```
df["Easy Apply"]=df["Easy Apply"].str.replace('-1','FALSE')        # Replace -1 with False in "Easy Apply Colum
n"
```

```
age_mean=df["Age"].mean()                                         # For Replacing NaN values with mean of Age
age_mean=int(age_mean)
df["Age"] = df["Age"].replace(np.nan, age_mean)

rating_mean=df["Rating"].mean()                                   # For Replacing NaN value with mean of Ratin
g
rating_mean=int(rating_mean)
df["Rating"] = df["Rating"].replace(np.nan, rating_mean)

df["Established"] = df["Established"].replace('-1', "Unknown")    # Replace -1 with Unknown
```

2. Data Types:

Question:
What are the data types of each indicator, and do they align with their expected types (e.g., numerical, categorical)?
Answer:

- `Age:` Mostly numerical (except for some missing values).
- `Salary:` String format (needs conversion to numerical values).
- `Rating:` Floating numerical with some missing values represented as -1.
- `Location:` String format.
- `Established:` Mostly numerical with some missing values.
- `Easy Apply:` Boolean (except for missing values).

The solution to find out the data type of each series:

```
df["Indicator/Column_Name"].dtype           # Returns the fdata type of each indicator
```

3. Outliers:

Question:
Identify potential outliers in numerical indicators (e.g., Age, Salary, Rating). Should outliers be removed or adjusted?
Answer:

- `Age:` Any age that seems unusually high or low compared to the typical range in the dataset could be considered an outlier.
  For example: `13` *is seem to be low outlier and* `66` *seems to be high outlier* `Age`.
- `Salary:` Outliers might include exceptionally high or low salaries compared to the majority of salaries in the dataset.
  For example: `10000` *is seem to be low outlier and* `101000` *seems to be high outlier* `Salary`.
- `Rating:` Values significantly deviating from the general range of ratings could be identified as outliers.
  For example: `0` *is seem to be low outlier and* `7.7` *seems to be high outlier* `Rating`.

The solution to find out the outliers, we use the following algorithm:

```
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    outliers = data[(data < lower_bound) | (data > upper_bound)]
```

4. Salary Formatting:

Question:
Examine the format of the Salary column. Does it require any formatting or standardization for consistent analysis?
Answer:
Salary values are in the format `"$XXk-$XXk"` and require transformation into numerical values.

- Split the `"Salary"` column at `'-'`, forming two columns namely `"Starting Salary"` and `"Ending Salary"`
- Left Strip the `'$'` symbol for both of the columns.
- Replace `'k'` alphabet to `'000'`

Following the the below snippet, we can clean the `"Salary"` column

```
df[["Starting Salary","Ending Salary"]]=df["Salary"].str.split('-',1,expand=True)       # Splits the following
Salary column
df=df.drop(columns=["Salary"])                                        # Delete the Salary column

df["Starting Salary"]=df["Starting Salary"].str.strip("$")            # Left Stripping '$'
df["Ending Salary"]=df["Ending Salary"].str.strip("$")
df["Starting Salary"]=df["Starting Salary"].str.replace('k','000')    # Replace 'k' with '000'
df["Ending Salary"]=df["Ending Salary"].str.replace('k','000')
```

5. Location Standardization:

Question:
Check the consistency of location entries. Do they need standardization, and how can this be achieved?
Answer:
`Locations` have various formats (e.g., "India,In", "New York,Ny", "Australia Aus"). Standardization could involve splitting these into separate columns for `Country` and `Initials`. Let us take a tour of all the countries,

- `India,In` : Here we have to split `India` and `In` at `','`
- `New York,NY` : Here also we need to split `New York` and `Ny` at `','`
- `Australia Aus` : Its a special case where we need to split `Australia` and `Aus` at `' '` but it shouldn't collide with New York, so we need to place them priortywise.

The following steps show how to split the `Location` into two seperate columns.

```
# Splitting based on different separators for different locations
def split_location(row):
    location = row['Location']
    if ',' in location:
        parts = location.split(',')
        country = parts[0].strip()
        initials = parts[1].strip()
    elif ' ' in location:
        parts = location.split(' ')
        country = parts[0].strip()
        initials = parts[1].strip()
    else:
        country = location.strip()
        initials = None  # Or set initials as needed for cases with no separator

    return pd.Series({'Country': country, 'Initials': initials})

# Apply the function to split 'Location' into 'Country' and 'Initials'
df[['Country', 'Initials']] = df.apply(split_location, axis=1)
df=df.drop(columns=["Location"])
```

6. Established Column:

Question:
Explore the Established column. Are there any inconsistencies or anomalies that need to be addressed?
Answer:
Yes, Inconsistencies exist in the Established column, with missing values represented as -1. It is converted to string first, then replace -1 with "Unknown" string

It can be solved if emphsize the following snippet:

```
df["Established"] = df["Established"].astype(str)       # Converting them to string data type
df["Established"] = df["Established"].replace('-1', "Unknown") # Replace -1 with Unknown
```

7. Easy Apply Indicator:

Question:
Analyze the Easy Apply column. Does it contain boolean values or need transformation for better analysis?
Answer:
`Easy Apply` column initially consists of string data type. Apart from that, it also consists of missing values which is `"-1"`. However, we consider `-1` as `FALSE` and replace all `-1` with `FALSE`.

Still, it is carries, string data type, which is converted to bool using map function.

Following algorithm is used for that case:

```
df["Easy Apply"]=df["Easy Apply"].str.replace('-1','FALSE')                   # Replace -1 with False
mapping = {'TRUE': True, 'FALSE': False}                                      # Mapped the string to respectiv
e bool value
df['Easy Apply'] = df['Easy Apply'].map(mapping).fillna(False).astype(bool)   # Formatting the column with Boo
lean data type
```

8. Rating Range:

Question:
Investigate the range of values in the Rating column. Does it fall within expected rating scales, and how should outliers be treated?
Answer:
Ratings range from `0 to 7.8`, with missing values represented as `-1`. It is justified that, it lies on a rating scale of 10 points. Moreover, missing values may need addressing.

For that kind of scenarios, we first replace the missing values with `NaN` then find the mean of `"Rating"` column, then replace all NaN values with its mean:

```
rating_mean=df["Rating"].mean()                                   # For Replacing NaN value with mean of Ratin
g
rating_mean=int(rating_mean)
df["Rating"] = df["Rating"].replace(np.nan, rating_mean)
```

9. Age Distribution:

Question:
Check the distribution of values in the Age column. Are there any unusual entries, and how might they impact analysis?
Answer:
`Age` values are present, but some are missing. Apart from that, we can see certain outliers in the `Age` column. Any age that seems unusually high or low compared to the typical range in the dataset could be considered an outlier. For example: `13` is seem to be low outlier and `66` seems to be high outlier Age.

In this way we delete rows where `Age` is `NaN`:

```
df=df.dropna()                              # Delete rows consisting NaN in data frame
```

10. Handling Special Characters:

Question:
Examine all text-based columns (e.g., Location). Are there special characters or inconsistencies that need cleaning?
Answer:
`Location` entries contain commas and spaces that require splitting or cleaning. `Locations` have various formats (e.g., "India,In", "New York,Ny", "Australia Aus"). Standardization could involve splitting these into separate columns for `Country` and `Initials`. Let us take a tour of all the countries,

- `India,In` : Here we have to split `India` and `In` at `','`
- `New York,NY` : Here also we need to split `New York` and `Ny` at `','`
- `Australia Aus` : Its a special case where we need to split `Australia` and `Aus` at `' '` but it shouldn't collide with New York, so we need to place them priortywise.

The following steps show how to split the `Location` into two seperate columns.

```
# Splitting based on different separators for different locations
def split_location(row):
    location = row['Location']
    if ',' in location:
        parts = location.split(',')
        country = parts[0].strip()
        initials = parts[1].strip()
    elif ' ' in location:
        parts = location.split(' ')
        country = parts[0].strip()
        initials = parts[1].strip()
    else:
        country = location.strip()
        initials = None  # Or set initials as needed for cases with no separator

    return pd.Series({'Country': country, 'Initials': initials})

# Apply the function to split 'Location' into 'Country' and 'Initials'
df[['Country', 'Initials']] = df.apply(split_location, axis=1)
df=df.drop(columns=["Location"])
```

`Salary values` are in the format `"$XXk-$XXk"` and require transformation into numerical values.

- Split the `"Salary"` column at `'-'`, forming two columns namely `"Starting Salary"` and `"Ending Salary"`
- Left Strip the `'$'` symbol for both of the columns.
- Replace `'k'` alphabet to `'000'`

Following the the below snippet, we can clean the `"Salary"` column

```
df[["Starting Salary","Ending Salary"]]=df["Salary"].str.split('-',1,expand=True)       # Splits the following
Salary column
df=df.drop(columns=["Salary"])                                        # Delete the Salary column

df["Starting Salary"]=df["Starting Salary"].str.strip("$")            # Left Stripping '$'
df["Ending Salary"]=df["Ending Salary"].str.strip("$")
df["Starting Salary"]=df["Starting Salary"].str.replace('k','000')    # Replace 'k' with '000'
df["Ending Salary"]=df["Ending Salary"].str.replace('k','000')
```

11. Data Integrity:

Question:
Ensure data integrity by cross-referencing entries. For instance, does the Established column align with the Age column?
Answer:
Yes, Inconsistencies exist between the `Established` and `Age` columns, with missing values causing misalignment.

- The `Established` column denotes the year of establishment for certain records, represented mostly in numerical format but contains missing values indicated by -1.
- The `Age` column refers to the age of individuals or entities, but it also contains missing values.
- We can compare the `Established` column's values (years) with the `Age` column to identify inconsistencies or misalignments. If both columns refer to the same entity, the established year should correlate with the age or else not.

12. Easy Apply Transformation:

Question:
If the Easy Apply column contains non-boolean values, how can it be transformed into a usable format?
Answer:
Some missing values exist in the `Easy Apply` column that may require handling. `Easy Apply` column initially consists of string data type. Apart from that, it also consists of missing values which is `"-1"`. However, we consider `-1` as `FALSE` and replace all `-1` with `FALSE`.

Still, it is carries, string data type, which is converted to bool using map function.

Following algorithm is used for that case:

```
df["Easy Apply"]=df["Easy Apply"].str.replace('-1','FALSE')                   # Replace -1 with False
mapping = {'TRUE': True, 'FALSE': False}                                      # Mapped the string to respectiv
e bool value
df['Easy Apply'] = df['Easy Apply'].map(mapping).fillna(False).astype(bool)   # Formatting the column with Boo
lean data type
```

13. Location Accuracy:

Question:
Assess the accuracy of location entries. Are there misspelled or ambiguous locations that require correction?
Answer:
Entries like `"India,In"` and `"New York,Ny"` could use splitting into separate columns for country and initials at `','`.
Apart from that `"Australia Aus"` could be use splitting into the same columns at `' '`. Also, we need to look out for that if we split at `' '` first then, `New York` would be affected. To avoid it we can use them in `if elif` contional flow, where `','` comes priorly than `' '`.

```
if ',' in location:
        parts = location.split(',')
        country = parts[0].strip()
        initials = parts[1].strip()
    elif ' ' in location:               # Specially for Australia
        parts = location.split(' ')
        country = parts[0].strip()
        initials = parts[1].strip()
    else:
        country = location.strip()
        initials = None  # Or set initials as needed for cases with no separator
```

14. Handling Categorical Data:

Question:
For categorical indicators, consider encoding or transforming them into a format suitable for analysis.
Answer:
Handling categorical data in the `"Location"` column involves:

- **Standardization**: Splitting the column into `"Country"` and `"Initials"` for uniformity and consistency.
- **Cleaning**: Removing special characters and maintaining consistent formatting for locations.
- **Transformation**: Encoding categorical data into numerical representations for analysis, using methods like label encoding.

```
if ',' in location:
        parts = location.split(',')
        country = parts[0].strip()
        initials = parts[1].strip()
    elif ' ' in location:               # Specially for Australia
        parts = location.split(' ')
        country = parts[0].strip()
        initials = parts[1].strip()
    else:
        country = location.strip()
        initials = None  # Or set initials as needed for cases with no separator
```

15. Consistent Rating Scale:

Question:
Ensure a consistent rating scale in the Rating column. Should it be normalized or adjusted for uniform analysis?
Answer:
Outliers might include exceptionally high or low salaries compared to the majority of salaries in the dataset.
For example: `10000` *is seem to be low outlier and* `101000` *seems to be high outlier* `Salary` .
Ratings range from `0 to 7.8` , with missing values represented as `-1` . It is justified that, it lies on a rating scale of 10 points. Moreover, missing values may need addressing.

For that kind of scenarios, we first replace the missing values with `NaN` then find the mean of `"Rating"` column, then replace all NaN values with its mean:

```
rating_mean=df["Rating"].mean()                                   # For Replacing NaN value with mean of Ratin
g
rating_mean=int(rating_mean)
df["Rating"] = df["Rating"].replace(np.nan, rating_mean)
```

Procedural Coding:

↓↓↓ Expand to view the working principles of code

In [ ]:

```
import pandas as pd
import numpy as np
```

Refering to the file consisting of data in CSV
It was seen that, even it was successful to run the code after uploading the raw data in local machine/drive.
However it was difficult to run the same code on other accounts.
In short, to access this raw csv file and run the code universally, I have uploaded the raw csv file to git and used the raw link here to run the code with ease.

Steps to import .csv file in Google Collab

- Upload the .csv file in Github Repository
- Open the .csv file on Github
- Click on `"Raw"` , present at right side of the page.
- Copy the link for the `"Raw"` file and paste it as `url`
- Pass `url` using `pandas library` to import .csv file in Google Colab

In [ ]:

```
url='https://raw.githubusercontent.com/ArijitDhali/PrepInsta-DA-Week-3/main/Assignment1.csv'
df=pd.read_csv(url,encoding='unicode_escape')
df
```

Well, we will always try to remove the duplicates from a data frame for removing redundancies.
If there are no redundant data, still it is considered to be a good practice.

In [ ]:

```
df=df.drop_duplicates()
df
```

`Easy Apply` column initially consists of string data type. Apart from that, it also consists of missing values which is `"-1"` . However, we consider `-1` as FALSE and replace all `-1` with `FALSE` .
As the column's data type is still string, we will map respective values to respective boolean values.
After that we will replace all missing values, `'-1'` with `NaN`

In [ ]:

```
df["Easy Apply"]=df["Easy Apply"].str.replace('-1','FALSE')
mapping = {'TRUE': True, 'FALSE': False}
df["Established"] = df["Established"].astype(str)
df = df.replace(-1, np.nan)
df
```

As we observe that the data set size is small, dropping rows may occur large deviation of output. To keep the integrity, we will do the following steps:

1. `Age:` Replace NaN values with the `.mean()` of Age column.
2. `Rating:` Replace NaN values with the `.mean()` of Rating column.
3. `Established:` Replace `'-1'` with the `"Unknown"` string.

In [ ]:

```python
age_mean=df["Age"].mean()
age_mean=int(age_mean)
df["Age"] = df["Age"].replace(np.nan, age_mean)

rating_mean=df["Rating"].mean()
rating_mean=int(rating_mean)
df["Rating"] = df["Rating"].replace(np.nan, rating_mean)


df["Established"] = df["Established"].replace('-1', "Unknown")
df
```

Now drop all the rows, where we get all `NULL` / empty values.

In [ ]:

```python
df=df.dropna()
df
```

Spliting the `"Salary"` column at `'-'`, forming two columns namely `"Starting Salary"` and `"Ending Salary"`

In [ ]:

```python
df[["Starting Salary","Ending Salary"]]=df["Salary"].str.split('-',1,expand=True)
df=df.drop(columns=["Salary"])
df
```

Left Strip the `'$'` symbol for both of the columns.
Replace `'k'` alphabet to `'000'`

In [ ]:

```python
df["Starting Salary"]=df["Starting Salary"].str.strip("$")
df["Ending Salary"]=df["Ending Salary"].str.strip("$")
df["Starting Salary"]=df["Starting Salary"].str.replace('k','000')
df["Ending Salary"]=df["Ending Salary"].str.replace('k','000')
df
```

Converting all uncertain data types to respective and needed data types for example, float to int for `Establishment` or String to Bool for `Easy Apply` using map.

In [ ]:

```python
df["Starting Salary"] = df["Starting Salary"].astype(int)
df["Ending Salary"] = df["Ending Salary"].astype(int)
df["Age"] = df["Age"].astype(int)
df['Easy Apply'] = df['Easy Apply'].map(mapping).fillna(False).astype(bool)
df.reset_index(drop=True, inplace=True)
df
```

- `India,In` : Here we have to split `India` and `In` at `','`
- `New York,NY` : Here also we need to split `New York` and `Ny` at `','`
- `Australia Aus` : Its a special case where we need to split `Australia` and `Aus` at `' '` but it shouldn't collide with New York, so we need to place them priortywise.

After splitting them we will be alloting them to seperate columns namely `Country` and `Initials`.

After converting, we find that there was incorrect indexing due to dropping of data which is now fixed.

In [ ]:

```python
# Splitting based on different separators for different locations
def split_location(row):
    location = row['Location']
    if ',' in location:
        parts = location.split(',')
        country = parts[0].strip()
        initials = parts[1].strip()
    elif ' ' in location:
        parts = location.split(' ')
        country = parts[0].strip()
        initials = parts[1].strip()
    else:
        country = location.strip()
        initials = None  # Or set initials as needed for cases with no separator

    return pd.Series({'Country': country, 'Initials': initials})

# Apply the function to split 'Location' into 'Country' and 'Initials'
df[['Country', 'Initials']] = df.apply(split_location, axis=1)
df=df.drop(columns=["Location"])
df.reset_index(drop=True, inplace=True)
df.drop(columns=['Index'], inplace=True)
df
```

Out[ ]:

	Age	Rating	Established	Easy Apply	Starting Salary	Ending Salary	Country	Initials
0	44	5.4	1999	True	44000	99000	India	In
1	66	3.5	2002	True	55000	66000	New York	Ny
2	39	4.0	Unknown	False	77000	89000	New York	Ny
3	64	4.4	1988	False	44000	99000	India	In
4	25	6.4	2002	False	44000	99000	Australia	Aus
5	44	1.4	1999	True	77000	89000	India	In
6	21	0.0	Unknown	False	44000	99000	New York	Ny
7	44	4.0	Unknown	False	44000	99000	Australia	Aus
8	35	5.4	Unknown	False	44000	99000	New York	Ny
9	22	7.7	Unknown	True	44000	99000	India	In
10	55	5.4	2008	True	10000	49000	India	In
11	44	6.7	2009	False	10000	49000	India	In
12	39	0.0	1999	False	44000	99000	India	In
13	25	4.0	2019	True	44000	99000	Australia	Aus
14	66	4.0	2020	True	44000	99000	Australia	Aus
15	44	3.0	1999	False	88000	101000	Australia	Aus
16	19	4.5	1984	False	19000	40000	India	In
17	39	5.3	1943	True	44000	99000	New York	Ny

18	35	6.7	1954	True	44000	99000	New York	Ny
19	32	3.3	1955	True	44000	99000	New York	Ny
20	39	5.7	1944	True	44000	99000	New York	Ny
21	35	5.0	1946	False	44000	99000	New York	Ny
22	19	7.8	1988	True	55000	66000	New York	Ny
23	39	2.4	1999	True	44000	99000	New York	Ny
24	13	4.0	1987	False	44000	99000	New York	Ny
25	55	0.0	1980	True	44000	99000	Australia	Aus
26	39	4.0	1934	True	55000	66000	India	In
27	52	5.4	1935	False	44000	99000	India	In
28	39	3.4	1932	False	39000	88000	Australia	Aus

With a cherry on top, we will lowercase the Column Headings [It is determined to do at previous step 🥲]

In [ ]:

```
df.columns=df.columns.str.lower().str.replace(' ','_')
df
```

Out[ ]:

	age	rating	established	easy_apply	starting_salary	ending_salary	country	initials
0	44	5.4	1999	True	44000	99000	India	In
1	66	3.5	2002	True	55000	66000	New York	Ny
2	39	4.0	Unknown	False	77000	89000	New York	Ny
3	64	4.4	1988	False	44000	99000	India	In
4	25	6.4	2002	False	44000	99000	Australia	Aus
5	44	1.4	1999	True	77000	89000	India	In
6	21	0.0	Unknown	False	44000	99000	New York	Ny
7	44	4.0	Unknown	False	44000	99000	Australia	Aus
8	35	5.4	Unknown	False	44000	99000	New York	Ny
9	22	7.7	Unknown	True	44000	99000	India	In
10	55	5.4	2008	True	10000	49000	India	In
11	44	6.7	2009	False	10000	49000	India	In
12	39	0.0	1999	False	44000	99000	India	In
13	25	4.0	2019	True	44000	99000	Australia	Aus
14	66	4.0	2020	True	44000	99000	Australia	Aus
15	44	3.0	1999	False	88000	101000	Australia	Aus
16	19	4.5	1984	False	19000	40000	India	In
17	39	5.3	1943	True	44000	99000	New York	Ny
18	35	6.7	1954	True	44000	99000	New York	Ny
19	32	3.3	1955	True	44000	99000	New York	Ny
20	39	5.7	1944	True	44000	99000	New York	Ny
21	35	5.0	1946	False	44000	99000	New York	Ny
22	19	7.8	1988	True	55000	66000	New York	Ny
23	39	2.4	1999	True	44000	99000	New York	Ny
24	13	4.0	1987	False	44000	99000	New York	Ny
25	55	0.0	1980	True	44000	99000	Australia	Aus
26	39	4.0	1934	True	55000	66000	India	In
27	52	5.4	1935	False	44000	99000	India	In
28	39	3.4	1932	False	39000	88000	Australia	Aus

Coding Playground

In [ ]:

```
import pandas as pd                 # We are not really calling pandas everytime
import numpy as np                  # Same case for numoy


# Splitting based on different separators for different locations
def split_location(row):
    location = row['Location']
    if ',' in location:                             # Split at ','
        parts = location.split(',')
        country = parts[0].strip()
        initials = parts[1].strip()
    elif ' ' in location:                           # Elif Split at ' '
        parts = location.split(' ')
        country = parts[0].strip()
        initials = parts[1].strip()
    else:                                            # Else split none
        country = location.strip()
        initials = None  # Or set initials as needed for cases with no separator

    return pd.Series({'Country': country, 'Initials': initials})        # Assigning each series with the values of p
articular row



url='https://raw.githubusercontent.com/ArijitDhali/PrepInsta-DA-Week-3/main/Assignment1.csv'      # To make this cod
e available everywhere
df=pd.read_csv(url,encoding='unicode_escape')            # To ensure proper format of data frame

df=df.drop_duplicates()                                   # Dropping all the duplicate datas if present

df["Easy Apply"]=df["Easy Apply"].str.replace('-1','FALSE')         # Replacing -1 with False
mapping = {'TRUE': True, 'FALSE': False}                            # Converting strings to boolean values
df["Established"] = df["Established"].astype(str)                   # Convert Established to str data type
df = df.replace(-1, np.nan)                                         # Replacing the remaining -1 with NaN Value
```

```
age_mean=df["Age"].mean()
age_mean=int(age_mean)
df["Age"] = df["Age"].replace(np.nan, age_mean)

rating_mean=df["Rating"].mean()
rating_mean=int(rating_mean)
df["Rating"] = df["Rating"].replace(np.nan, rating_mean)


df["Established"] = df["Established"].replace('-1', "Unknown")


df=df.dropna()                                                      # Drop remaining rows where Null value is present

df[["Starting Salary","Ending Salary"]]=df["Salary"].str.split('-',1,expand=True)     # Splitting salary at '-'
df=df.drop(columns=["Salary"])                                                        # No more need of Salary column

df["Starting Salary"]=df["Starting Salary"].str.strip("$")          # Left Striping '$'
df["Ending Salary"]=df["Ending Salary"].str.strip("$")
df["Starting Salary"]=df["Starting Salary"].str.replace('k','000')  # Replacing 'k' with '000'
df["Ending Salary"]=df["Ending Salary"].str.replace('k','000')

df["Starting Salary"] = df["Starting Salary"].astype(int)           # Convert Starting salary to int data type
df["Ending Salary"] = df["Ending Salary"].astype(int)               # Convert Ending salary to int data type
df["Age"] = df["Age"].astype(int)                                   # Convert Age to int data type
df['Easy Apply'] = df['Easy Apply'].map(mapping).fillna(False).astype(bool)   # Convert Easy Apply to int boolean type


# Apply the function to split 'Location' into 'Country' and 'Initials'
df[['Country', 'Initials']] = df.apply(split_location, axis=1)      # Assigning the splitted values of Location to new two columns
df=df.drop(columns=["Location"])                                    # No need of Location column anymore
df.reset_index(drop=True, inplace=True)                             # To restructure the index
df.drop(columns=['Index'], inplace=True)
df.columns=df.columns.str.lower().str.replace(' ','_')
df
```

```
<ipython-input-56-a30de82e046e>:48: FutureWarning: In a future version of pandas all arguments of StringMethods.split except for the argument 'pat' will be keyword-only.
  df[["Starting Salary","Ending Salary"]]=df["Salary"].str.split('-',1,expand=True)     # Splitting salary at '-'
```

Out[ ]:

	age	rating	established	easy_apply	starting_salary	ending_salary	country	initials
0	44	5.4	1999	True	44000	99000	India	In
1	66	3.5	2002	True	55000	66000	New York	Ny
2	39	4.0	Unknown	False	77000	89000	New York	Ny
3	64	4.4	1988	False	44000	99000	India	In
4	25	6.4	2002	False	44000	99000	Australia	Aus
5	44	1.4	1999	True	77000	89000	India	In
6	21	0.0	Unknown	False	44000	99000	New York	Ny
7	44	4.0	Unknown	False	44000	99000	Australia	Aus
8	35	5.4	Unknown	False	44000	99000	New York	Ny
9	22	7.7	Unknown	True	44000	99000	India	In
10	55	5.4	2008	True	10000	49000	India	In
11	44	6.7	2009	False	10000	49000	India	In
12	39	0.0	1999	False	44000	99000	India	In
13	25	4.0	2019	True	44000	99000	Australia	Aus
14	66	4.0	2020	True	44000	99000	Australia	Aus
15	44	3.0	1999	False	88000	101000	Australia	Aus
16	19	4.5	1984	False	19000	40000	India	In
17	39	5.3	1943	True	44000	99000	New York	Ny
18	35	6.7	1954	True	44000	99000	New York	Ny
19	32	3.3	1955	True	44000	99000	New York	Ny
20	39	5.7	1944	True	44000	99000	New York	Ny
21	35	5.0	1946	False	44000	99000	New York	Ny
22	19	7.8	1988	True	55000	66000	New York	Ny
23	39	2.4	1999	True	44000	99000	New York	Ny
24	13	4.0	1987	False	44000	99000	New York	Ny
25	55	0.0	1980	True	44000	99000	Australia	Aus
26	39	4.0	1934	True	55000	66000	India	In
27	52	5.4	1935	False	44000	99000	India	In
28	39	3.4	1932	False	39000	88000	Australia	Aus

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29 entries, 0 to 28
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   age              29 non-null     int64  
 1   rating           29 non-null     float64
 2   established      29 non-null     object 
 3   easy_apply       29 non-null     bool   
 4   starting_salary  29 non-null     int64  
 5   ending_salary    29 non-null     int64  
 6   country          29 non-null     object 
 7   initials         29 non-null     object 
dtypes: bool(1), float64(1), int64(3), object(3)
memory usage: 1.7+ KB
```