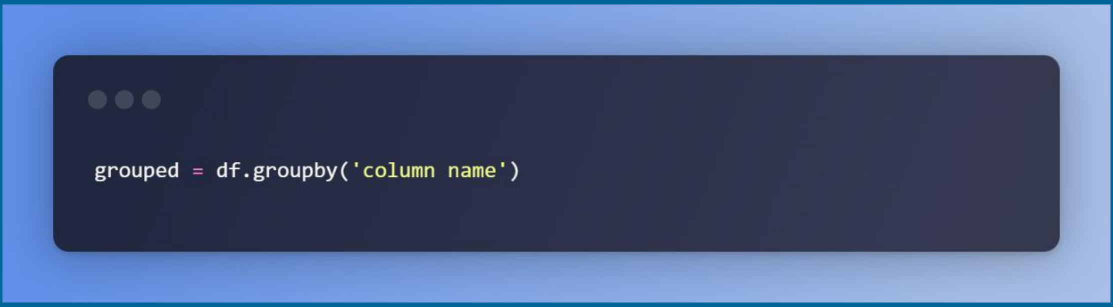
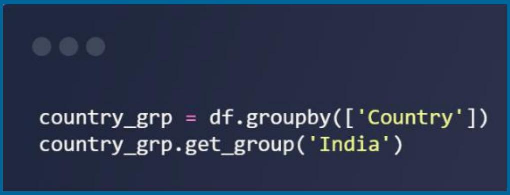
Data Grouping:

The pandas.DataFrame.groupby method is a powerful tool used to group data in a DataFrame based on one or more columns. It allows you to perform aggregate operations, transformations, or filtering operations on subsets of your data. The basic idea is to split the data into groups based on some criteria, apply a function to each group, and then combine the results back into a DataFrame.



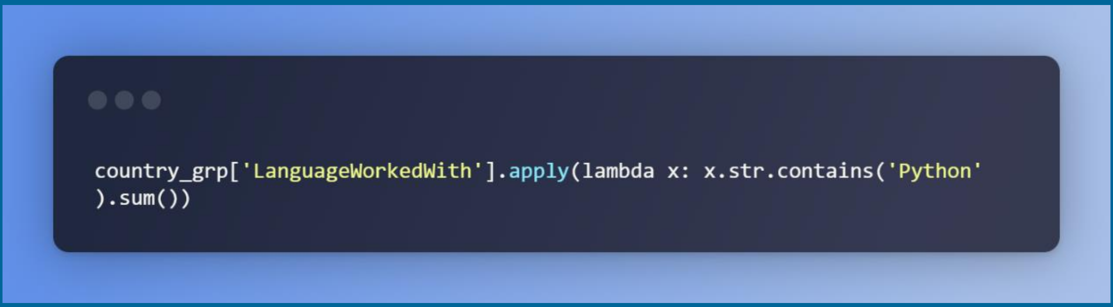
```
grouped = df.groupby('column name')
```

used to group a DataFrame by a specific column and then retrieve the subset of the data corresponding to a particular group.



```
country_grp = df.groupby(['Country'])
country_grp.get_group('India')
```

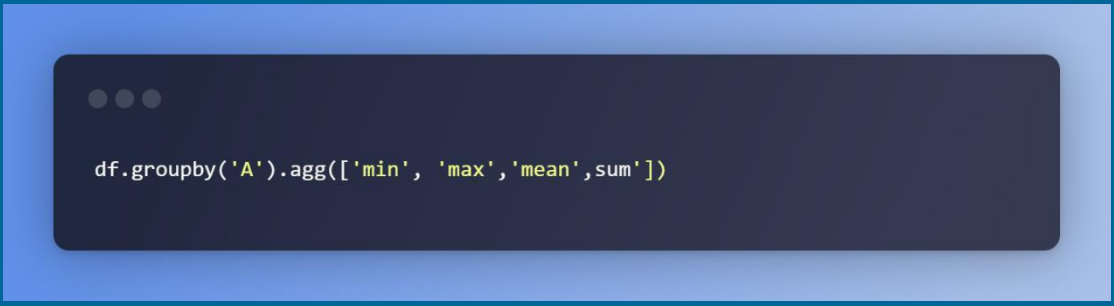
used to calculate the number of times 'Python' is mentioned in the 'LanguageWorkedWith' column for each group in the DataFrame grouped by 'Country'.



```
country_grp['LanguageWorkedWith'].apply(lambda x: x.str.contains('Python'
).sum())
```

Aggregation :

You can perform aggregate operations such as sum, mean, count, etc., on each group.

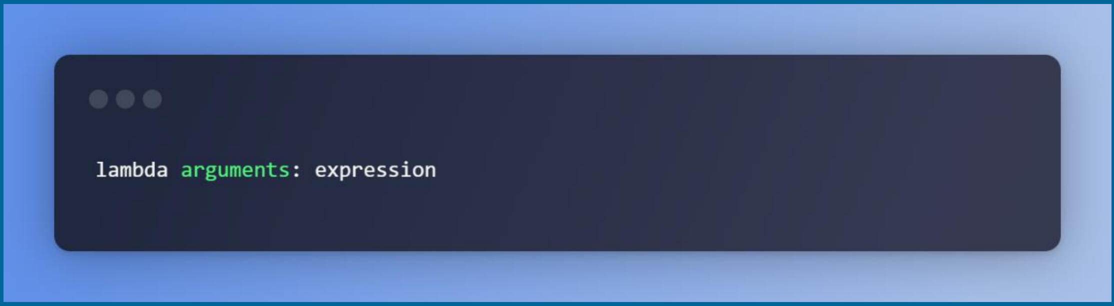


```
df.groupby('A').agg(['min', 'max','mean',sum'])
```

you want to group the DataFrame by column 'A' and then compute the minimum, maximum, mean, and sum for each group.

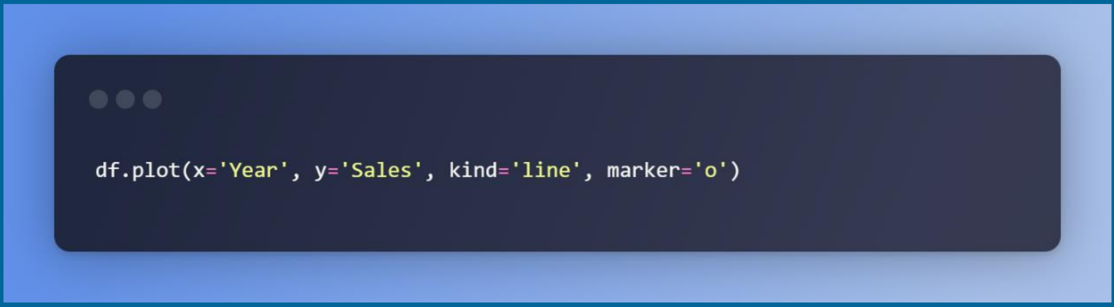
lambda x :

is used to define an anonymous (or unnamed) function. The lambda keyword allows you to create small, throwaway functions on the fly. These functions are often used for simple operations where defining a full function using def might be considered overkill.



```
lambda arguments: expression
```

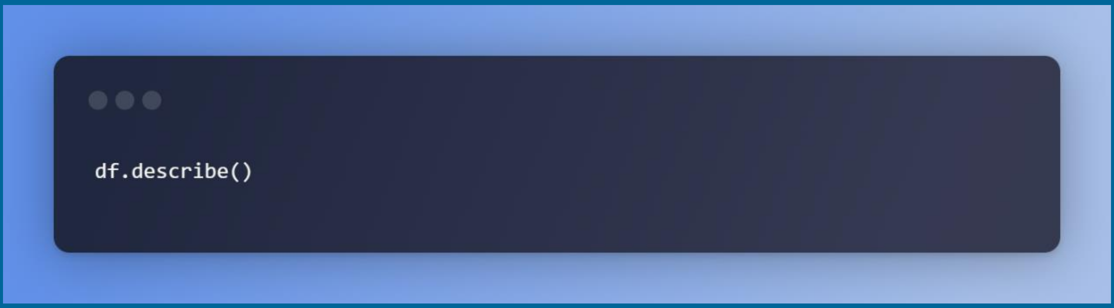
Plot :



```
df.plot(x='Year', y='Sales', kind='line', marker='o')
```

describe()

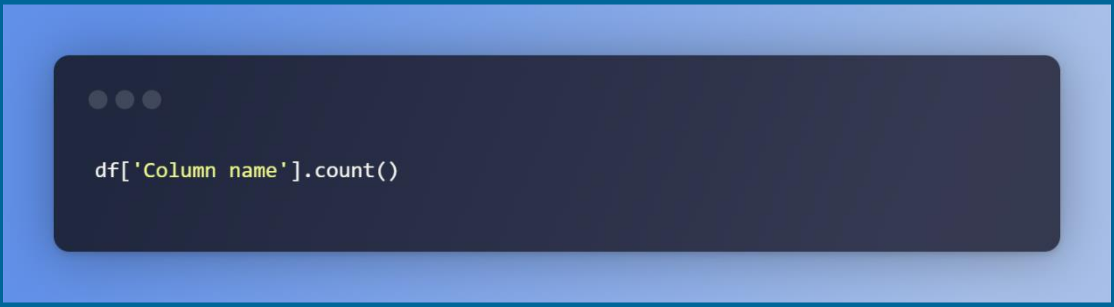
Purpose: To generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution.



```
df.describe()
```

count()

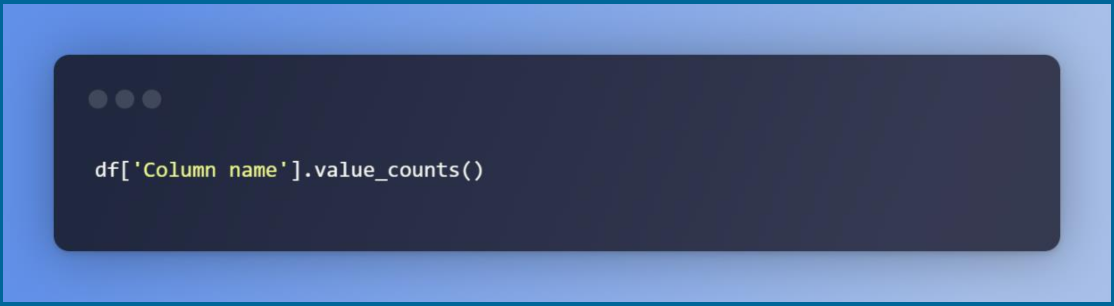
In pandas, the .count() method is used to count the number of non-null values in a specific column of a DataFrame.



```
df['Column name'].count()
```

.value_counts()

count the number of occurrences of each unique value in a specific column of a DataFrame.



```
df['Column name'].value_counts()
```

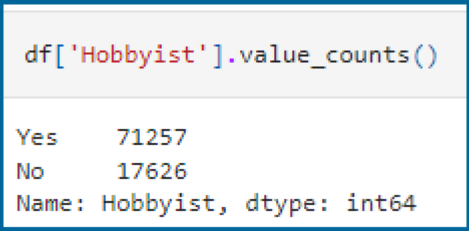
.value_counts(normalize=True) ➔ to display it in percentages

.value_counts():

Provides the count of each unique value in the column.

Returns a Series with unique values as the index and their counts as the values, sorted by frequency.

Excludes NaN values by default but can include them with dropna=False.



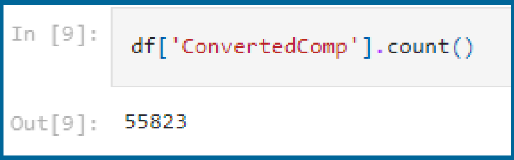
```
df['Hobbyist'].value_counts()

Yes     71257
No      17626
Name: Hobbyist, dtype: int64
```

. count():

Provides the number of non-null values.

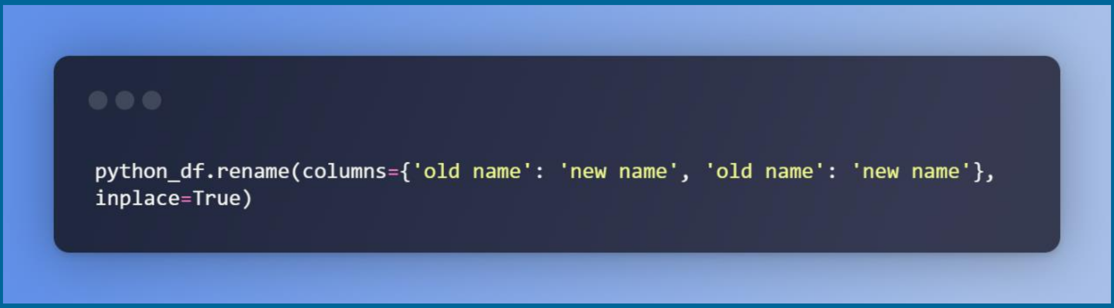
Returns a single integer when used on a Series or a Series when used on a DataFrame.

Does not provide information about the distribution of unique values.



```
In [9]:  df['ConvertedComp'].count()

Out[9]:  55823
```

To rename a column :



```
python_df.rename(columns={'old name': 'new name', 'old name': 'new name'},
inplace=True)
```

Data Cleaning:

drop_duplicates() :

method in pandas is used to remove duplicate rows from a DataFrame or Series. It can be applied to either the entire DataFrame or specific columns, and you can customize its behavior with various parameters.

```
drop_duplicates()
```

drop() :

method is used to remove rows or columns from a DataFrame.

```
drop(columns = "Column name")
```

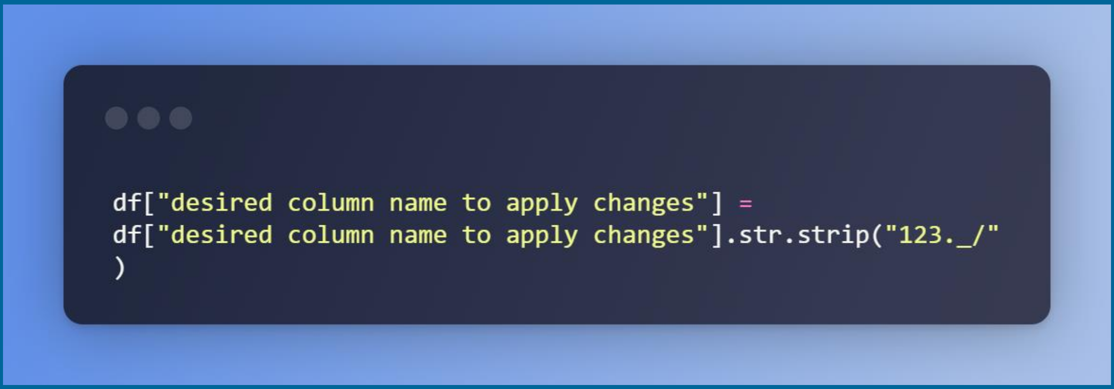
str.strip() :

is used to remove leading and trailing characters (whitespace by default) from a string.

You can do it one by one :

```
df["desired column name to apply changes"] =
df["desired column name to apply changes"].str.lstrip("...")

df["desired column name to apply changes"] =
df["desired column name to apply changes"].str.lstrip("/")

df["desired column name to apply changes"] =
df["desired column name to apply changes"].str.rstrip("_")
```

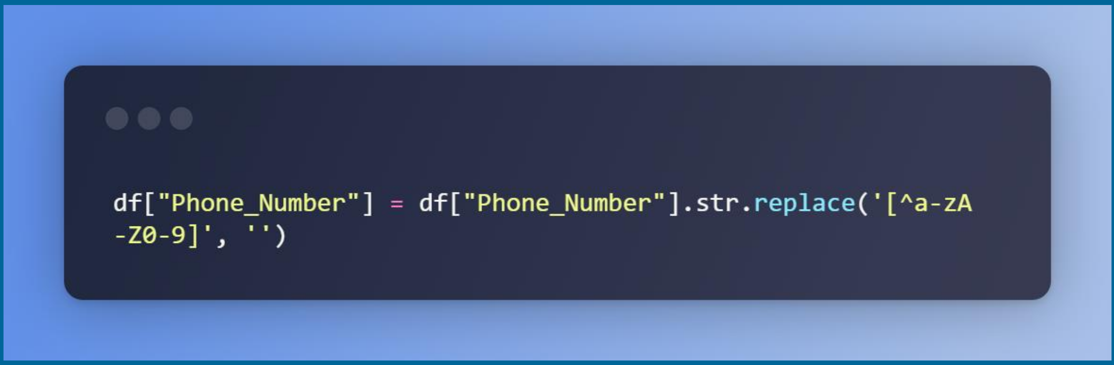
Or :



```
df["desired column name to apply changes"] =
df["desired column name to apply changes"].str.strip("123._/"
)
```

For dealing with phone numbers :

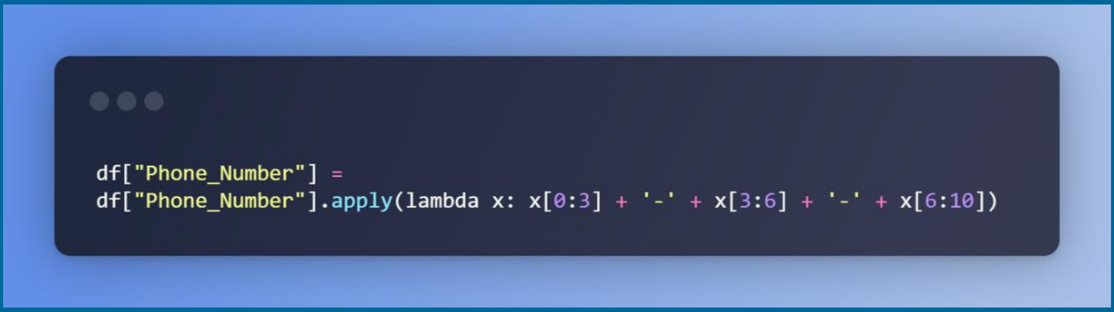
Removing Non-Alphanumeric Characters :



```
df["Phone_Number"] = df["Phone_Number"].str.replace('[^a-zA
-Z0-9]', '')
```

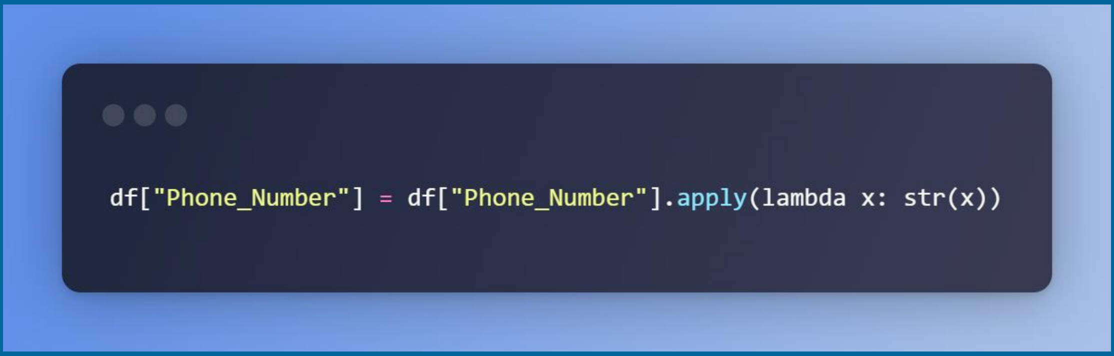
Formatting Phone Numbers :

This line formats the phone numbers into the XXX-XXX-XXXX format. It assumes that after cleaning, the phone number length is exactly 10 digits.



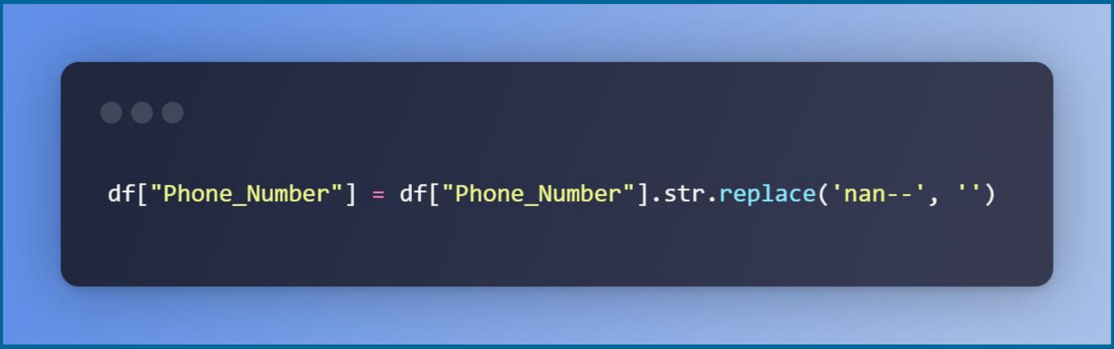
```
df["Phone_Number"] =
df["Phone_Number"].apply(lambda x: x[0:3] + '-' + x[3:6] + '-' + x[6:10])
```

Converting to String :



```
df["Phone_Number"] = df["Phone_Number"].apply(lambda x: str(x))
```

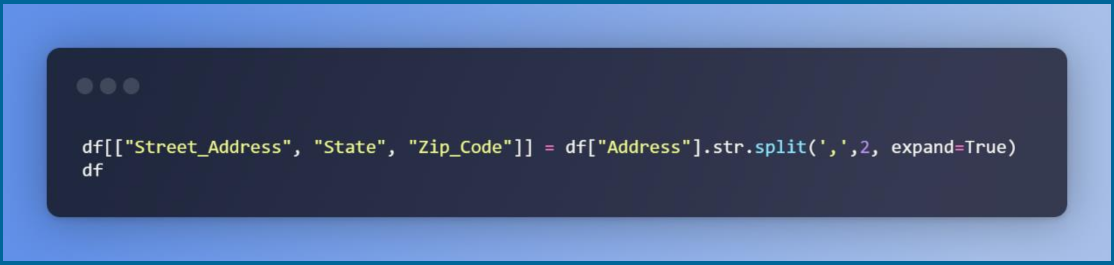
Handling Specific Cases :



```
df["Phone_Number"] = df["Phone_Number"].str.replace('nan--', '')
```

---

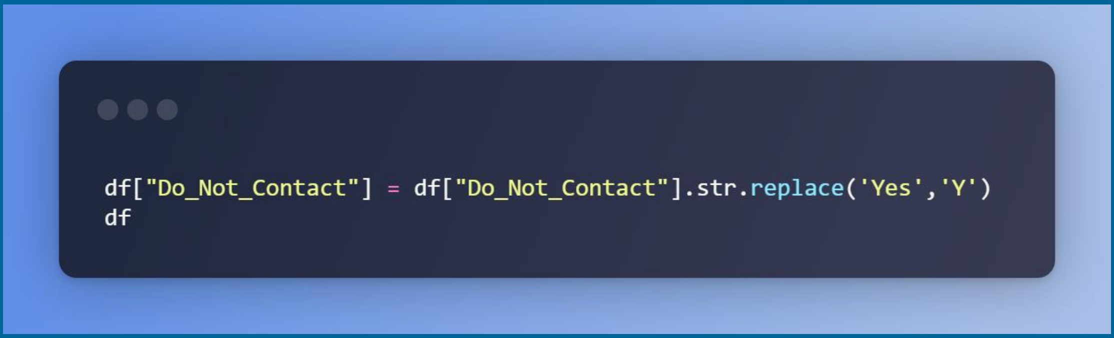
split an address column in a DataFrame into separate columns



```
df[["Street_Address", "State", "Zip_Code"]] = df["Address"].str.split(',',2, expand=True)
df
```

---

replace() :

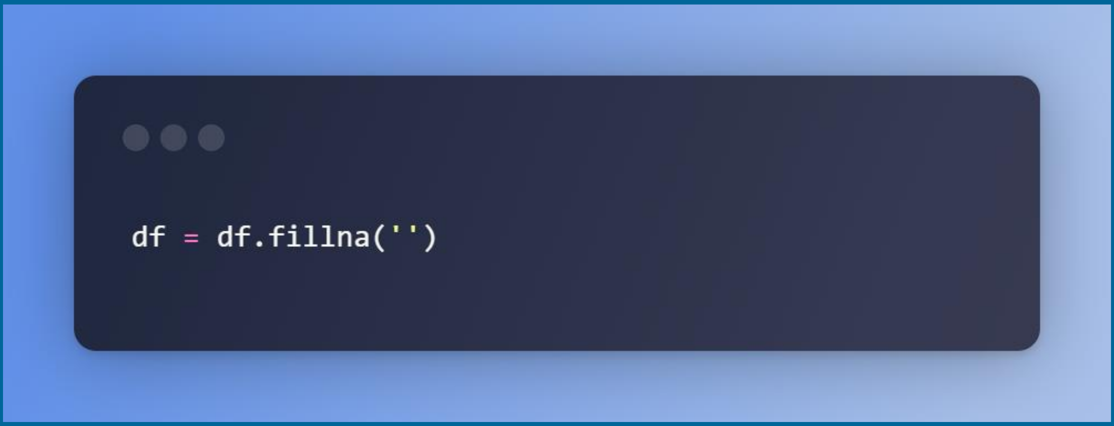


```
df["Do_Not_Contact"] = df["Do_Not_Contact"].str.replace('Yes','Y')
df
```

---

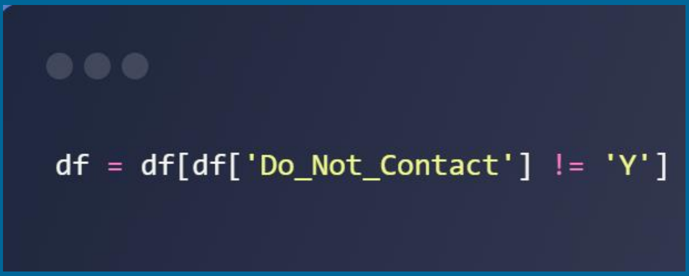
fillna()

This method replaces NaN (Not a Number) values in the DataFrame with an empty string ''. This is generally used to handle missing values in a more systematic way.



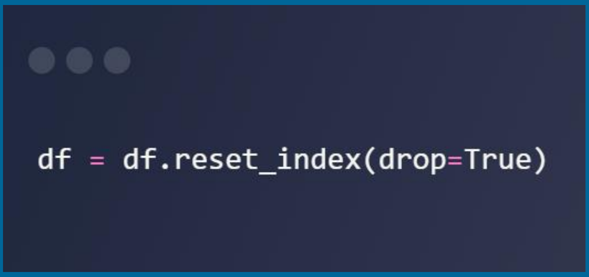
```
df = df.fillna('')
```

to remove rows from the DataFrame where the value in the "Do_Not_Contact" column is 'Y'



```
df = df[df['Do_Not_Contact'] != 'Y']
```

The drop=True parameter ensures that the old index is not added as a new column in the DataFrame.



```
df = df.reset_index(drop=True)
```