# 20 Pandas Functions for 80% of your Data Science Tasks

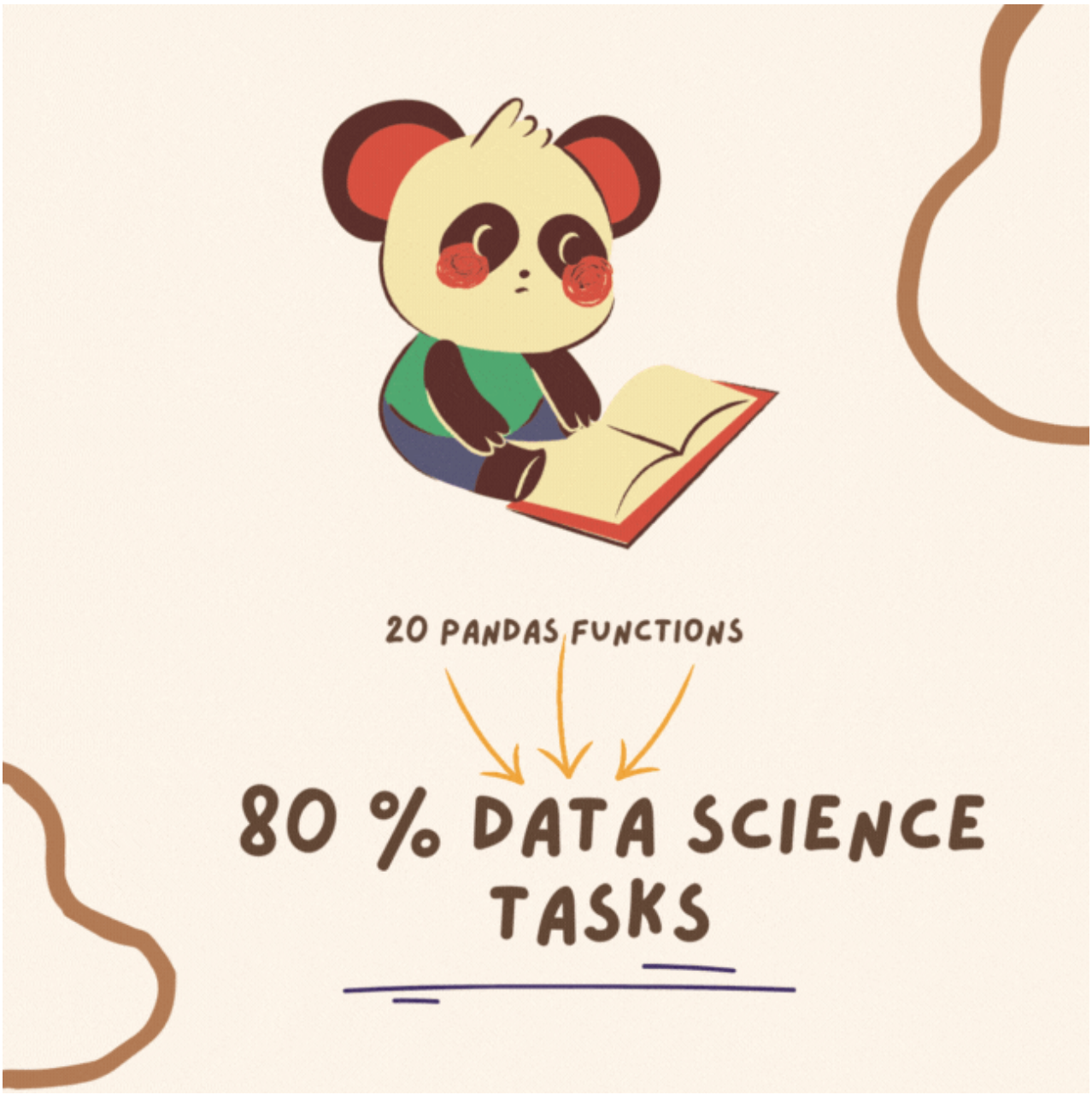
levelup.gitconnected.com/20-pandas-functions-for-80-of-your-data-science-tasks-b610c8bfe63c

30 January 2023

## Master these Functions and Get Your Work Done

Pandas is one of the most widely used libraries in the data science community and it's a powerful tool that can help you with data manipulation, cleaning, and analysis. Whether you're a beginner or an experienced data scientist, this article will provide valuable insights into the most commonly used Pandas functions and how to use them practically.

We will cover everything from basic data manipulation to advanced data analysis techniques, and by the end of this article, you will have a solid understanding of how to use Pandas to make your data science workflow more efficient.




20 Pandas Functions for 80 % of Data Science Tasks by Youssef Hosni

**If you want to start a career in data science & AI and do not know how. I offer data science mentoring sessions and long-term career mentoring:**

- Long-term mentoring:
- Mentoring sessions:

## Join Medium with my referral link - Youssef Hosni

## Read every story from Youssef Hosni (and thousands of other writers on Medium). Your membership fee directly supports…

youssefraafat57.medium.com

**If you want to study Data Science and Machine Learning for free, check out these resources:**

- Free interactive roadmaps to learn Data Science and Machine Learning by yourself. Start here:
- The search engine for Data Science learning resources (FREE). Bookmark your favorite resources, mark articles as complete and add study notes.
- Want to learn Data Science from scratch with the support of a mentor and a learning community? Join this Study Circle for free:

## 1. pd.read_csv()

pd.read_csv is a function in the panda's library in Python that is used to read a CSV (Comma Separated Values) file and convert it into a pandas DataFrame.

Example:

```
pandas  pddf = pd.read_csv()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **0** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | SOPHIA | 119 | 1 |
| **1** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | CHLOE | 106 | 2 |
| **2** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMILY | 93 | 3 |
| **3** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | OLIVIA | 89 | 4 |
| **4** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMMA | 75 | 5 |
| **...** | ... | ... | ... | ... | ... | ... |
| **11340** | 2016 | FEMALE | BLACK NON HISPANIC | Saniyah | 10 | 43 |
| **11341** | 2016 | FEMALE | BLACK NON HISPANIC | Skye | 10 | 43 |
| **11342** | 2016 | FEMALE | BLACK NON HISPANIC | Tiana | 10 | 43 |
| **11343** | 2016 | FEMALE | BLACK NON HISPANIC | Violet | 10 | 43 |
| **11344** | 2016 | FEMALE | BLACK NON HISPANIC | Zahra | 10 | 43 |

In this example, the pd.read_csv function reads the file 'data.csv' and converts it into a DataFrame, which is then assigned to the variable 'data'. The contents of the DataFrame can then be printed using the print function.

It has many options like `sep` , `header` , `index_col` , `skiprows` , `na_values` etc.

```
= pd.read_csv(, sep=, header=, index_col=, skiprows=, na_values=)
```

This example reads the CSV file `data.csv` , with `;` as a separator, the first row as the header, the first column as an index, skipping the first 5 rows and replacing `N/A` with NaN.

## 2. df.describe()

The `df.describe()` method in pandas is used to generate summary statistics of various features of a DataFrame. It returns a new DataFrame that contains the count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum of

each numerical column in the original DataFrame.

```
(df.describe())
```

| | Year of Birth | Count | Rank |
|---|---|---|---|
| count | 11345.000000 | 11345.000000 | 11345.000000 |
| mean | 2013.552578 | 32.836844 | 57.656765 |
| std | 1.746367 | 36.981866 | 25.186882 |
| min | 2011.000000 | 10.000000 | 1.000000 |
| 25% | 2012.000000 | 13.000000 | 38.000000 |
| 50% | 2014.000000 | 20.000000 | 59.000000 |
| 75% | 2015.000000 | 35.000000 | 79.000000 |
| max | 2016.000000 | 387.000000 | 102.000000 |

You can also include or exclude certain columns and also include non-numerical columns by passing appropriate arguments to the method.

```
df.describe(include=)
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **count** | 11345.000000 | 11345 | 11345 | 11345 | 11345.000000 | 11345.000000 |
| **unique** | NaN | 3 | 7 | 3016 | NaN | NaN |
| **top** | NaN | FEMALE | WHITE NON HISPANIC | Avery | NaN | NaN |
| **freq** | NaN | 5756 | 3365 | 24 | NaN | NaN |
| **mean** | 2013.552578 | NaN | NaN | NaN | 32.836844 | 57.656765 |
| **std** | 1.746367 | NaN | NaN | NaN | 36.981866 | 25.186882 |
| **min** | 2011.000000 | NaN | NaN | NaN | 10.000000 | 1.000000 |
| **25%** | 2012.000000 | NaN | NaN | NaN | 13.000000 | 38.000000 |
| **50%** | 2014.000000 | NaN | NaN | NaN | 20.000000 | 59.000000 |
| **75%** | 2015.000000 | NaN | NaN | NaN | 35.000000 | 79.000000 |
| **max** | 2016.000000 | NaN | NaN | NaN | 387.000000 | 102.000000 |

```
df.describe(exclude=)
```

| | Gender | Ethnicity | Child's First Name |
|---|---|---|---|
| **count** | 11345 | 11345 | 11345 |
| **unique** | 3 | 7 | 3016 |
| **top** | FEMALE | WHITE NON HISPANIC | Avery |
| **freq** | 5756 | 3365 | 24 |

## 3. df.info()

`df.info()` is a method in pandas that is used to get a concise summary of the DataFrame, including the number of non-null values in each column, the data types of each column, and the memory usage of the DataFrame.

```
(df.info())
```

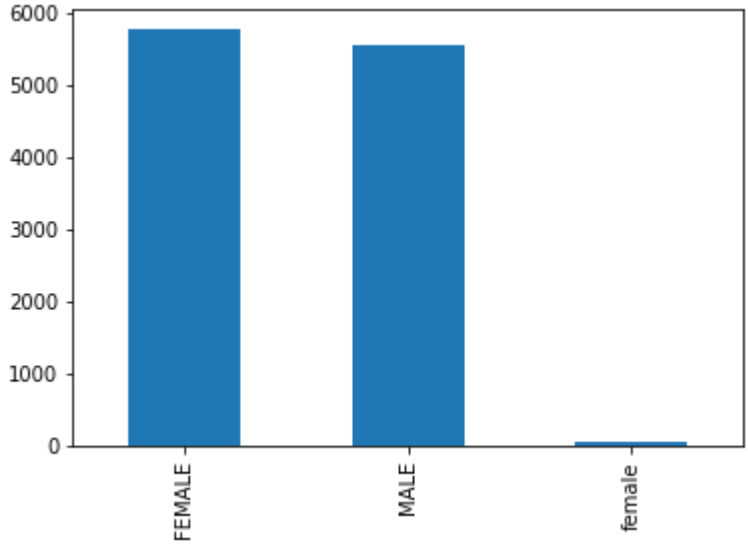
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11345 entries, 0 to 11344
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Year of Birth       11345 non-null  int64 
 1   Gender              11345 non-null  object
 2   Ethnicity           11345 non-null  object
 3   Child's First Name  11345 non-null  object
 4   Count               11345 non-null  int64 
 5   Rank                11345 non-null  int64 
dtypes: int64(3), object(3)
memory usage: 531.9+ KB
None
```

## 4. df.plot()

`df.plot()` is a method in pandas that is used to create various types of plots from a DataFrame. By default, it creates a line plot of all numerical columns in the DataFrame. But you can also pass the argument `kind` to specify the type of plot you want to create. Available options are **line**, **bar**, **barh**, **hist**, **box**, **kde**, **density**, **area**, **pie**, **scatter**, and **hexbin.**

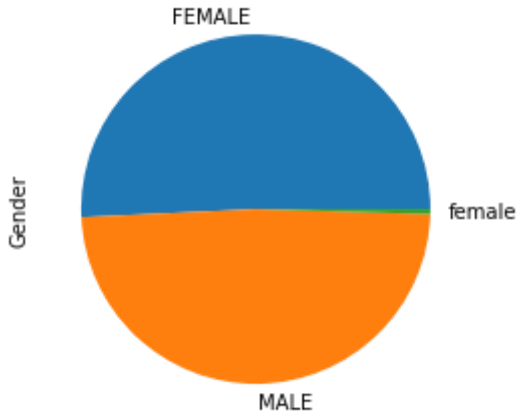
In the examples below I will use the .plot() method to plot numerical and categorical variables. For the categorical variable, I will plot bar and pie plots and for the numerical variable, I will plot box plots. You can try a different types of plots by yourself.

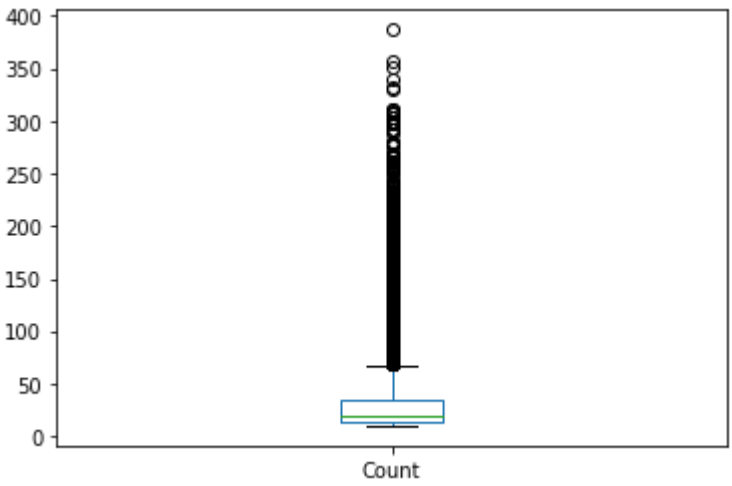
```
df[].value_counts().plot(kind=)
```




```
df[].value_counts().plot(kind=)

[].plot(kind=)
```




It also supports many other options like `title`, `xlabel`, `ylabel`, `legend`, `grid`, `xlim`, `ylim`, `xticks`, `yticks` etc. to customize the plot. You can also use `plt.xlabel()`, `plt.ylabel()`, `plt.title()` etc. after the plot to customize it.

Please keep in mind that `df.plot()` is just a convenient wrapper around the `matplotlib.pyplot` library, so the same customization options available in `matplotlib` are available for `df.plot()`.

## 5. df.iloc()

Pandas' `.iloc()` function is used to select rows and columns by their integer-based index in a DataFrame. It is used to select rows and columns by their integer-based location.

Here are some examples of how you can use it:

```
(df.iloc[])
```

```
Year of Birth                                2011
Gender                                     FEMALE
Ethnicity             ASIAN AND PACIFIC ISLANDER
Child's First Name                         SOPHIA
Count                                         119
Rank                                            1
Name: 0, dtype: object
```

```
(df.iloc[:])
```

```
   Year of Birth  Gender                    Ethnicity Child's First Name  \
0           2011  FEMALE  ASIAN AND PACIFIC ISLANDER             SOPHIA
1           2011  FEMALE  ASIAN AND PACIFIC ISLANDER              CHLOE

   Count  Rank
0    119     1
1    106     2
```

```
(df.iloc[:, ])
```

```
0         2011
1         2011
2         2011
3         2011
4         2011
          ...
11340     2016
11341     2016
11342     2016
11343     2016
11344     2016
Name: Year of Birth, Length: 11345, dtype: int64
```

```
(df.iloc[:, :])
```

```
       Year of Birth  Gender
0                2011  FEMALE
1                2011  FEMALE
2                2011  FEMALE
3                2011  FEMALE
4                2011  FEMALE
...               ...     ...
11340            2016  FEMALE
11341            2016  FEMALE
11342            2016  FEMALE
11343            2016  FEMALE
11344            2016  FEMALE

[11345 rows x 2 columns]
```

```
(df.iloc[, ])
```

```
             FEMALE
```

In the above examples, `df.iloc[0]` selects the first row of the dataframe, `df.iloc[:2]` selects the first two rows, `df.iloc[:, 0]` selects the first column, `df.iloc[:, :2]` selects the first two columns, and `df.iloc[1, 1]` selects the element at the (1, 1) position of the dataframe (second row, second column).

Keep in mind that `.iloc()` only selects rows and columns based on their integer-based index, so if you want to select rows and columns based on their labels you should use `.loc()` method instead as will be shown next.

## 6. df.loc()

Pandas' `.loc()` function is used to select rows and columns by their label-based index in a DataFrame. It is used to select rows and columns by their label-based location.

Here are some examples of how you can use it:

```
(df.loc[:,])
```

```
0         FEMALE
1         FEMALE
2         FEMALE
3         FEMALE
4         FEMALE
           ...
11340     FEMALE
11341     FEMALE
11342     FEMALE
11343     FEMALE
11344     FEMALE
Name: Gender, Length: 11345, dtype: object
```

```
(df.loc[:, [, ]])
```

| | Year of Birth | Gender |
|---|---|---|
| 0 | 2011 | FEMALE |
| 1 | 2011 | FEMALE |
| 2 | 2011 | FEMALE |
| 3 | 2011 | FEMALE |
| 4 | 2011 | FEMALE |
| ... | ... | ... |
| 11340 | 2016 | FEMALE |
| 11341 | 2016 | FEMALE |
| 11342 | 2016 | FEMALE |
| 11343 | 2016 | FEMALE |
| 11344 | 2016 | FEMALE |

[11345 rows x 2 columns]

In the above example, we used `df.loc[:, 'Gender']` to select the column named 'Gender', and `df.loc[:, ['Year of Birth', 'Gender']]` selects the columns named 'Year of Birth' and 'Gender'.

## 7. df.assign()

Pandas' `.assign()` function is used to add new columns to a DataFrame, based on the computation of existing columns. It allows you to add new columns to a DataFrame without modifying the original dataframe. The function returns a new DataFrame with the added columns.

Here is an example of how you can use it:

```
df_new = df.assign(count_plus_5=df[] + )df_new.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank | count_plus_5 |
|---|---|---|---|---|---|---|---|
| **0** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | SOPHIA | 119 | 1 | 124 |
| **1** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | CHLOE | 106 | 2 | 111 |
| **2** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMILY | 93 | 3 | 98 |
| **3** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | OLIVIA | 89 | 4 | 94 |
| **4** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMMA | 75 | 5 | 80 |

In the above example, the first time `df.assign()` is used to create a new column named 'count_plus_5' with the value of count + 5.

It's important to note that the original DataFrame `df` remains unchanged and the new DataFrame `df_new` is returned with the new columns added.

The `.assign()` method can be used multiple times in a chain, allowing you to add multiple new columns to a DataFrame in one line of code.

## 8. df.query()

Pandas' `.query()` function allows you to filter a DataFrame based on a Boolean expression. It allows you to select rows from a DataFrame using a query string similar to SQL. The function returns a new DataFrame containing only the rows that satisfy the Boolean expression.

Here is an example of how you can use it:

```
df_query = df.query()df_query.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **0** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | SOPHIA | 119 | 1 |
| **1** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | CHLOE | 106 | 2 |
| **2** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMILY | 93 | 3 |
| **3** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | OLIVIA | 89 | 4 |
| **4** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMMA | 75 | 5 |

```
df_query  df.df_query.head
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **993** | 2011 | MALE | ASIAN AND PACIFIC ISLANDER | ETHAN | 177 | 1 |
| **994** | 2011 | MALE | ASIAN AND PACIFIC ISLANDER | JAYDEN | 173 | 2 |
| **995** | 2011 | MALE | ASIAN AND PACIFIC ISLANDER | RYAN | 150 | 3 |
| **996** | 2011 | MALE | ASIAN AND PACIFIC ISLANDER | JUSTIN | 110 | 4 |
| **997** | 2011 | MALE | ASIAN AND PACIFIC ISLANDER | LUCAS | 103 | 5 |

In the above example, the first time `df.query()` is used to select rows where the count is greater than 30 and the rank is less than 30, and the second time `df.query()` is used to select rows where gender is 'MALE'.

It's important to note that the original DataFrame `df` remains unchanged and the new DataFrame `df_query` is returned with the filtered rows.

The `.query()` method can be used with any valid Boolean expression and it's useful when you want to filter a DataFrame based on multiple conditions or when the conditions are complex and hard to express using the standard indexing operators.

Also, keep in mind that the `.query()` method is slower than boolean indexing, so if performance is critical, you should use boolean indexing instead.

## 9. df.sort_values()

Pandas' `.sort_values()` function allows you to sort a DataFrame by one or multiple columns. It sorts the DataFrame based on the values of one or more columns, in ascending or descending order. The function returns a new DataFrame sorted by the specified column(s).

Here is an example of how you can use it:

```
df_sorted = df.sort_values(by=)df_sorted.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **5672** | 2014 | FEMALE | ASIAN AND PACIFIC ISLANDER | Priscilla | 10 | 40 |
| **4344** | 2013 | FEMALE | HISPANIC | Raquel | 10 | 78 |
| **4343** | 2013 | FEMALE | HISPANIC | Rachel | 10 | 78 |
| **4342** | 2013 | FEMALE | HISPANIC | Paula | 10 | 78 |
| **4341** | 2013 | FEMALE | HISPANIC | Nyah | 10 | 78 |

```
df_sorted = df.sort_values(by=, ascending=)df_sorted.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **7236** | 2014 | MALE | WHITE NON HISPANIC | Yidel | 10 | 102 |
| **7224** | 2014 | MALE | WHITE NON HISPANIC | Erik | 10 | 102 |
| **7235** | 2014 | MALE | WHITE NON HISPANIC | Shraga | 10 | 102 |
| **7234** | 2014 | MALE | WHITE NON HISPANIC | Nico | 10 | 102 |
| **7233** | 2014 | MALE | WHITE NON HISPANIC | Myles | 10 | 102 |

```
df_sorted = df.sort_values(by=[, ])df_sorted.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **141** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | AIZA | 10 | 38 |
| **142** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | ALEXA | 10 | 38 |
| **143** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | ALISHA | 10 | 38 |
| **144** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | ANGIE | 10 | 38 |
| **145** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | ANNABELLE | 10 | 38 |

In the above example, the first time `df.sort_values()` is used to sort the DataFrame by 'Count' in ascending order, the second time is used to sort by 'Rank' in descending order, and the last time it's used to sort by multiple columns 'Count' and 'Rank'.

It's important to note that the original DataFrame `df` remains unchanged and the new DataFrame `df_sorted` is returned with the sorted values.

The `.sort_values()` method can be used with any column(s) of the DataFrame and it's useful when you want to sort the DataFrame based on multiple columns, or when you want to sort the DataFrame by a column in descending order.

## 10. df.sample()

Pandas' `.sample()` function allows you to randomly select rows from a DataFrame. It returns a new DataFrame containing the randomly selected rows. The function takes several parameters that allow you to control the sampling process, such as the number of rows to return, and whether or not to sample with replacement and seed for reproducibility.

Here is an example of how you can use it:

```
df_sample = df.sample(n=, replace=, random_state=)df_sample
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **10441** | 2016 | MALE | WHITE NON HISPANIC | Grant | 22 | 87 |
| **8113** | 2015 | FEMALE | ASIAN AND PACIFIC ISLANDER | Alexandra | 13 | 40 |

```
df_sample = df.sample(n=, replace=, random_state=)df_sample
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **235** | 2011 | FEMALE | BLACK NON HISPANIC | ANIYA | 19 | 36 |
| **5192** | 2013 | MALE | HISPANIC | Ezequiel | 11 | 94 |
| **905** | 2011 | FEMALE | WHITE NON HISPANIC | GOLDA | 14 | 77 |

```
df_sample = df.sample(n=, replace=, random_state=, axis=)df_sample
```

| | Ethnicity | Gender |
|---|---|---|
| **0** | ASIAN AND PACIFIC ISLANDER | FEMALE |
| **1** | ASIAN AND PACIFIC ISLANDER | FEMALE |
| **2** | ASIAN AND PACIFIC ISLANDER | FEMALE |
| **3** | ASIAN AND PACIFIC ISLANDER | FEMALE |
| **4** | ASIAN AND PACIFIC ISLANDER | FEMALE |
| **...** | ... | ... |
| **11340** | BLACK NON HISPANIC | FEMALE |
| **11341** | BLACK NON HISPANIC | FEMALE |
| **11342** | BLACK NON HISPANIC | FEMALE |
| **11343** | BLACK NON HISPANIC | FEMALE |
| **11344** | BLACK NON HISPANIC | FEMALE |

11345 rows × 2 columns

In the above example, the first time `df.sample()` is used to randomly select 2 rows without replacement, the second time is used to randomly select 3 rows with replacement and the last time is used to randomly select 2 columns without replacement.

It's important to note that the original DataFrame `df` remains unchanged and the new DataFrame `df_sample` is returned with the randomly selected rows.

The `.sample()` method can be useful when you want to randomly select a subset of the data for testing or validation, or when you want to randomly select a sample of rows for further analysis. The `random_state` parameter is useful for reproducibility and the `axis=1` parameter allows you to select columns.

## 11. df.isnull()

The `isnull()` method in Pandas returns a DataFrame of the same shape as the original DataFrame, but with `True` or `False` values indicating whether each value in the original DataFrame is missing or not. Missing values, such as `NaN` or `None`, will be `True` in the resulting DataFrame, while non-missing values will be `False`.

```
df.isnull()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | False |
| **1** | False | False | False | False | False | False |
| **2** | False | False | False | False | False | False |
| **3** | False | False | False | False | False | False |
| **4** | False | False | False | False | False | False |
| **...** | ... | ... | ... | ... | ... | ... |
| **11340** | False | False | False | False | False | False |
| **11341** | False | False | False | False | False | False |
| **11342** | False | False | False | False | False | False |
| **11343** | False | False | False | False | False | False |
| **11344** | False | False | False | False | False | False |

11345 rows × 6 columns

## 12. df.fillna()

The `fillna()` method in Pandas is used to fill in missing values in a DataFrame with a specified value or method. By default, it replaces missing values with `NaN`, but you can specify a different value to use instead as shown below:

- `value` : Specifies the value to use to fill in the missing values. Can be a scalar value or a dict of values for different columns.
- `method` : Specifies the method to use for filling in missing values. Can be 'ffill' (forward-fill) or 'bfill' (backward-fill) or 'interpolate'(interpolate values) or 'pad' or 'backfill'
- `axis` : Specifies the axis along which to fill in missing values. It can be 0 (rows) or 1 (columns).
- `inplace` : Whether to fill in the missing values in place (modifying the original DataFrame) or to return a new DataFrame with the missing values filled in.
- `limit` : Specifies the maximum number of consecutive missing values to fill.
- `downcast` : Specifies a dictionary of values to use to downcast the data types of columns.

```
# fill missing values with 0
df.fillna(0)

# forward-fill missing values (propagates last valid observation forward to next)
df.fillna(method='ffill')

# backward-fill missing values (propagates next valid observation backward to
last)
df.fillna(method='bfill')

df.interpolate()
```

It is important to note that the `fillna()` method returns a new DataFrame with the missing values filled in and does not modify the original DataFrame in place. If you want to modify the original DataFrame, you can use the `inplace` parameter and set it to `True`.

```
df.fillna(, inplace=)
```

## 13. df.dropna()

`df.dropna()` is a method used in the Pandas library to remove missing or null values from a DataFrame. It removes rows or columns from the DataFrame where at least one element is missing.

You can remove all rows containing at least one missing value by calling `df.dropna().`

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

If you want to remove only the columns that contain at least one missing value you can use `df.dropna(axis=1)`

```
df = df.dropna(axis=)
```

You can also set `thresh` parameter to keep only the rows/columns that have at least `thresh` non-NA/null values.

```
df = df.dropna(thresh=)
```

## 14. df.drop()

`df.drop()` is a method used in the Pandas library to remove rows or columns from a DataFrame by specifying the corresponding labels. It can be used to drop one or multiple rows or columns based on their labels.

You can remove a specific row by calling `df.drop()` and passing the index label of the row you want to remove, and the axis parameter set to 0 (default is 0)

```
df_drop = df.drop()
```

This would remove the first row of the DataFrame.

You can also drop multiple rows by passing a list of index labels:

```
df_drop = df.drop([,])
```

This would remove the first and second rows of the DataFrame.

Similarly, you can drop columns by passing the labels of the columns you want to remove and setting the `axis` parameter to 1:

```
df_drop = df.drop([, ], axis=)
```

| | Year of Birth | Gender | Ethnicity | Child's First Name |
|---|---|---|---|---|
| **0** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | SOPHIA |
| **1** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | CHLOE |
| **2** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMILY |
| **3** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | OLIVIA |
| **4** | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMMA |

## 15. pd.pivot_table()

`pd.pivot_table()` is a method in the Pandas library that is used to create a pivot table from a DataFrame. A pivot table is a table that summarizes and aggregates data in a more meaningful and organized way, by creating a new table with one or more columns as the index, one or more columns as values, and one or more columns as attributes.

In the example below we will create a pivot table with **Ethnicity** as the index and aggregate the sum of the count. This is used to know the count of each **Ethnicity** in the dataset.

```
pivot_table = pd.pivot_table(df, =, =, aggfunc=)pivot_table.head()
```

| | Count |
|---|---|
| Ethnicity | |
| ASIAN AND PACI | 9185 |
| ASIAN AND PACIFIC ISLANDER | 41500 |
| BLACK NON HISP | 8802 |
| BLACK NON HISPANIC | 45179 |
| HISPANIC | 122805 |

You can also include more columns in the pivot table by specifying multiple index and values parameters and also include multiple aggfunc functions.

```
pivot_table = pd.pivot_table(df, index=[,], values=  , aggfunc=
[,])pivot_table.head()
```

| | | sum | count |
|---|---|---|---|
| | | Rank | Rank |
| Ethnicity | Gender | | |
| ASIAN AND PACI | FEMALE | 5741 | 171 |
| | MALE | 7383 | 164 |
| ASIAN AND PACIFIC ISLANDER | FEMALE | 24097 | 777 |
| | MALE | 35389 | 835 |
| | female | 1432 | 50 |
| BLACK NON HISP | FEMALE | 5516 | 159 |
| | MALE | 7528 | 169 |
| BLACK NON HISPANIC | FEMALE | 26878 | 848 |
| | MALE | 38857 | 856 |
| HISPANIC | FEMALE | 103488 | 1704 |
| | MALE | 117006 | 1610 |
| WHITE NON HISP | FEMALE | 22414 | 341 |
| | MALE | 20932 | 296 |
| WHITE NON HISPANIC | FEMALE | 115302 | 1756 |
| | MALE | 122153 | 1609 |

## 16. df.groupby()

`df.groupby()` is a method in the Pandas library that is used to group rows of a DataFrame based on one or multiple columns. This allows you to perform aggregate operations on the groups, such as calculating the mean, sum, or count of the values in each group.

`df.groupby()` returns a `GroupBy` object, which you can then use to perform various operations on the groups, such as calculating the sum, mean, or count of the values in each group.

Let's see an example using the birth names dataset:

```
grouped = df.groupby()(grouped.mean())
```

The output of the above code would be:

| Gender | Year of Birth | Count | Rank |
|---|---|---|---|
| FEMALE | 2013.569493 | 28.489750 | 52.716470 |
| MALE | 2013.558043 | 37.474093 | 63.052537 |
| female | 2011.000000 | 19.560000 | 28.640000 |

```
grouped = df.groupby(['Gender', 'Ethnicity'])

(grouped.())
```

The output of the above code would be:

| Gender | Ethnicity | Year of Birth | Count | Rank |
|---|---|---|---|---|
| FEMALE | ASIAN AND PACI | 344052 | 4072 | 5741 |
| | ASIAN AND PACIFIC ISLANDER | 1564965 | 17055 | 24097 |
| | BLACK NON HISP | 319908 | 3693 | 5516 |
| | BLACK NON HISPANIC | 1707706 | 19288 | 26878 |
| | HISPANIC | 3430969 | 52225 | 103488 |
| | WHITE NON HISP | 686092 | 10954 | 22414 |
| | WHITE NON HISPANIC | 3536414 | 56700 | 115302 |
| MALE | ASIAN AND PACI | 329968 | 5113 | 7383 |
| | ASIAN AND PACIFIC ISLANDER | 1681670 | 23467 | 35389 |
| | BLACK NON HISP | 340028 | 5109 | 7528 |
| | BLACK NON HISPANIC | 1723760 | 25891 | 38857 |
| | HISPANIC | 3241744 | 70580 | 117006 |
| | WHITE NON HISP | 595552 | 12850 | 20932 |
| | WHITE NON HISPANIC | 3240376 | 64559 | 122153 |
| female | ASIAN AND PACIFIC ISLANDER | 100550 | 978 | 1432 |

## 17. df.transpose()

`df.transpose()` is a method in the Pandas library used to transpose the rows and columns of a DataFrame? This means that the rows become columns and the columns become rows.

```
# Transpose the DataFrame
df_transposed = df.transpose()

df_transposed.head()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 11335 | 11336 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Year of Birth | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | ... | 2016 | 2016 |
| Gender | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | ... | FEMALE | FEMALE |
| Ethnicity | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ... | BLACK NON HISPANIC | BLACK NON HISPANIC |
| Child's First Name | SOPHIA | CHLOE | EMILY | OLIVIA | EMMA | ISABELLA | TIFFANY | ASHLEY | FIONA | ANGELA | ... | Mikayla | Mila |
| Count | 119 | 106 | 93 | 89 | 75 | 67 | 54 | 52 | 48 | 47 | ... | 10 | 10 |

5 rows × 11345 columns

It can also be done using `T` attributes on the dataframe. `df.T` will do the same as `df.transpose()`.

```
df_transposed = df.Tdf_transposed.head()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 11335 | 11336 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Year of Birth | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | 2011 | ... | 2016 | 2016 |
| Gender | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | FEMALE | ... | FEMALE | FEMALE |
| Ethnicity | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ASIAN AND PACIFIC ISLANDER | ... | BLACK NON HISPANIC | BLACK NON HISPANIC |
| Child's First Name | SOPHIA | CHLOE | EMILY | OLIVIA | EMMA | ISABELLA | TIFFANY | ASHLEY | FIONA | ANGELA | ... | Mikayla | Mila |
| Count | 119 | 106 | 93 | 89 | 75 | 67 | 54 | 52 | 48 | 47 | ... | 10 | 10 |

5 rows × 11345 columns

## 18. df.merge()

`df.merge()` is a pandas function that allows you to combine two DataFrames based on one or more common columns. It is similar to SQL JOINs. The function returns a new DataFrame that contains only the rows where the values in the specified columns match between the two DataFrames.

Here is an example of how to use `df.merge()` to combine two DataFrames based on a common column:

```
# Create the first DataFrame
df1 = pd.DataFrame({'key': ['A', 'B', 'C', 'D'],
'value': [1, 2, 3, 4]})


# Create the second DataFrame
df2 = pd.DataFrame({'key': ['B', 'D', 'E', 'F'],
'value': [5, 6, 7, 8]})


# Merge the two DataFrames on the 'key' column
merged_df = df1.merge(df2, on='key')

(merged_df)
```

As you can see the two dataframe are merged on the key column and the column name is appended with _x and _y for the left and right dataframe respectively.

| | key | value_x | value_y |
|---|---|---|---|
| 0 | B | 2 | 5 |
| 1 | D | 4 | 6 |

You can also use left, right and outer join by passing how = 'left', how = 'right', or how = 'outer' respectively.

You can also merge multiple columns by passing a list of columns to the on parameter.

```
merged_df = df1.merge(df2, on=[,])
```

You can also specify a different column name to merge by using left_on and right_on parameters.

```
merged_df = df1.merge(df2, left_on=, right_on=)
```

It's worth noting that the `merge()` function has many options and parameters that allow you to control the behavior of the merge, such as how to handle missing values, whether to keep all rows or only those that match and what columns to merge on.

## 19. df.rename()

`df.rename()` is a pandas function that allows you to change the name of one or more columns or rows in a DataFrame. You can use the `columns` parameter to change the column names, and the `index` parameter to change the row names.

```
df_rename = df.rename(columns={: })df_rename.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | count | Rank |
|---|---|---|---|---|---|---|
| 0 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | SOPHIA | 119 | 1 |
| 1 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | CHLOE | 106 | 2 |
| 2 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMILY | 93 | 3 |
| 3 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | OLIVIA | 89 | 4 |
| 4 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMMA | 75 | 5 |

You can also use a dictionary to rename multiple columns at once:

```
df_rename = df.rename(columns={: , :})df_rename.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | count | rank |
|---|---|---|---|---|---|---|
| 0 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | SOPHIA | 119 | 1 |
| 1 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | CHLOE | 106 | 2 |
| 2 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMILY | 93 | 3 |
| 3 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | OLIVIA | 89 | 4 |
| 4 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMMA | 75 | 5 |

You can also rename the index similarly:

```
df_rename = df.rename(index={:,:,:})df_rename.head()
```

| | Year of Birth | Gender | Ethnicity | Child's First Name | Count | Rank |
|---|---|---|---|---|---|---|
| first | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | SOPHIA | 119 | 1 |
| second | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | CHLOE | 106 | 2 |
| third | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMILY | 93 | 3 |
| 3 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | OLIVIA | 89 | 4 |
| 4 | 2011 | FEMALE | ASIAN AND PACIFIC ISLANDER | EMMA | 75 | 5 |

## 20. df.to_csv()

`df.to_csv()` is a method used in the Pandas library to export a DataFrame to a CSV file. CSV stands for "Comma Separated Values" and it is a popular file format for storing data in a tabular form.

For example, let's say we want to save `df` that you want to export to a CSV file. You can export the DataFrame to a CSV file by calling `df.to_csv()` and passing the file name as a string:

```
df.to_csv()
```

This will save the DataFrame to a file named data.csv in the current working directory. You can also specify the path of the file by passing it to the method:

```
df.to_csv()
```

You can also specify the separator used in the CSV file by passing the sep parameter. By default, it's set to ",".

```
df.to_csv(, sep=)
```

It is also possible to only save specific columns of the DataFrame by passing the list of column names to the columns parameter, and also to save only specific rows by passing a boolean mask to the index parameter.

```
df.to_csv(, columns=[,])
```

You can also use the **index** parameter to specify whether to include or exclude the index of the dataframe in the exported CSV file.

```
df.to_csv(, index=)
```

This will exclude the index of the dataframe in the exported CSV file.

You can also use `na_rep` parameter to replace missing values in the exported CSV file with a specific value.

```
df.to_csv(, na_rep=)
```

In summary here are the 20 pandas functions that can be used to accomplish most of your data tasks:

1. pd.read_csv()
2. df.describe()
3. df.info()
4. df.plot()
5. df.iloc()
6. df.loc()
7. df.assign()
8. df.query()
9. df.sort_values()
10. df.sample()
11. df.isnull()
12. df.fillna()
13. df.dropna()
14. df.drop()
15. pd.pivot_table()
16. df.groupby()
17. df.transpose()
18. df.merge()
19. df.rename()
20. df.to_csv()