PANDAS NOTES

1. A Dataframe is Two dimensional data set(rows and column)
2. method chaining. When we select the one column from the dataset , it returns one series, and we can call series function on top of it. And that will return another series on which again we can call the series function, which is called as method chaining
3. Pandas uses rangeindex by default , it will load entire index in memory unless required
4. Index uses hash tables which is very fast in selection
5. DataFrame attribute returned a NumPy n-dimensional array, or ndarray. Most of pandas relies heavily on the ndarray. Beneath the index, columns, and data are NumPy ndarrays

data may be classified as either continuous or categorical. Continuous data is always numeric and represents some kind of measurements, such as height, wage, or salary. Continuous data can take on an infinite number of possibilities. Categorical data, on the other hand, represents discrete, finite amounts of values such as car color, type of poker hand, or brand of cereal.

1. when a column is of the object data type, it signals that the entire column is strings
2. The .info method prints the data type information in addition to the count of non-null values.
3. Selected a single column from a DataFrame returns a Series (that has the same index as the DataFrame).
4. Selecting column from DataFrame .  One syntax uses the index operator and the other uses attribute access (or doqt notation).
5. movies["director\_name"] and movies.director\_name
6. We can also index off of the .loc and .iloc attributes to pull out a Series. The former allows us to pull out by column name, while the latter by position.
7. >>> movies.loc[:, "director\_name"]
8. movies.iloc[:, 1]

test.dtypes –retrun series, type of each column

test.dtypes.value\_counts() –return count of each datatype

>>> movies["director\_name"].index

RangeIndex(start=0, stop=4916, step=1)

>>> movies["director\_name"].dtype

dtype('O')

>>> movies["director\_name"].size

4196

>>> movies["director\_name"].name

'director\_name'

>>> movies["director\_name"].apply(type).unique()

array([<class 'str'>, <class 'float'>], dtype=object)

test.Age.apply(type).unique()

array([<class 'float'>], dtype=object)

Use the .value\_counts method to return the counts of each data type:

>>> movies.dtypes.value\_counts()

float64 13

int64 3

object 12

dtype: int64

Series Method

Same data

>>> movies = pd.read\_csv("data/movie.csv")

>>> director = movies["director\_name"]

>>> fb\_likes = movies["actor\_1\_facebook\_likes"]

>>> director.dtype

dtype('O')

>>> fb\_likes.dtype

dtype('float64')

1. .head() retrun another series
2. .value\_counts()

The data type of the Series usually determines which of the methods will be the most useful. For instance, one of the most useful methods for the object data type Series is .value\_counts, which calculates the frequencies:

1. .size or .shape and .unique()

Counting the number of elements in the Series may be done with the .size or .shape attribute or the built-in len function. The .unique method will return a NumPy array with the unique values:

1. .count()

Additionally, there is the .count method, which doesn't return the count of items, but the number of non-missing values:

1. Basic summary statistics are provided with .min, .max, .mean, .median, and .std
2. .describe()

.describe method to return both the summary statistics and a few of the quantiles at once( all value from step 4)

>>> fb\_likes.describe()

count 4909.000000

mean 6494.488491

std 15106.986884

min 0.000000

25% 607.000000

50% 982.000000

75% 11000.000000

max 640000.000000

**Note**: when we use .describe() method with object data type method will get different output

>>> director.describe()

count 4814

unique 2397

top Steven Spielberg

freq 26

Name: director\_name, dtype: object

1. .quantile() --

The .quantile method calculates the quantile of numeric data. Note that if you pass in a scaler, you will get scalar output, but if you pass in a list, the output is a pandas Series

>>> fb\_likes.quantile(0.2)

510.0

>>> fb\_likes.quantile(

... [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

... )

0.1 240.0

0.2 510.0

0.3 694.0

0.4 854.0

0.5 982.0

0.6 1000.0

0.7 8000.0

0.8 13000.0

0.9 18000.0

Name: actor\_1\_facebook\_likes, dtype: float64

1. .isna()

The .isna method can be used to determine whether each individual value is missing or not

1. .fillna()

It is possible to replace all missing values within a Series with the .fillna method

>>> fb\_likes\_filled = fb\_likes.fillna(0)

>>> fb\_likes\_filled.count()

4916

1. .dropna()

To remove the entries in Series elements with missing values, use the .dropna method

>>> fb\_likes\_dropped = fb\_likes.dropna()

>>> fb\_likes\_dropped.size

4909

NOTE: The .value\_counts

The .value\_counts method is one of the most informative Series methods and heavily used during exploratory analysis, especially with categorical columns. It defaults to returning the counts, but by setting the normalize parameter to True, the relative frequencies are returned instead, which provides another view of the distribution

1. .hasnas

A more direct approach is to inspect the .hasnans attribute

1. .notna()

There exists a complement of .isna; the .notna method, which returns True for all the non-missing

1. Isnull()

isnull method, which is an alias for .isna

Series operations