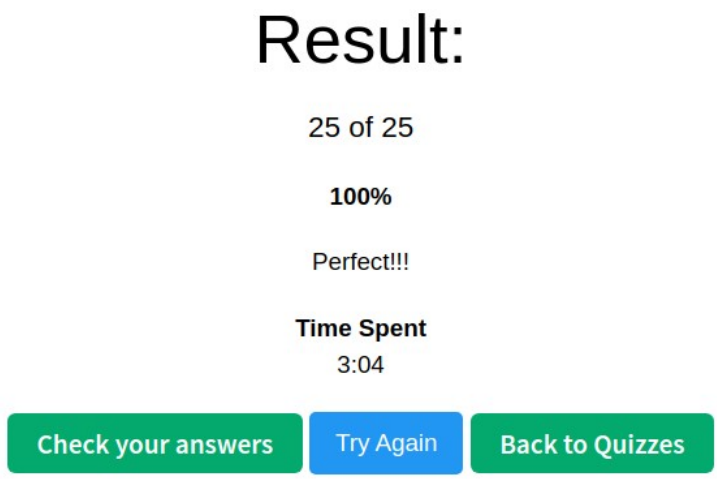
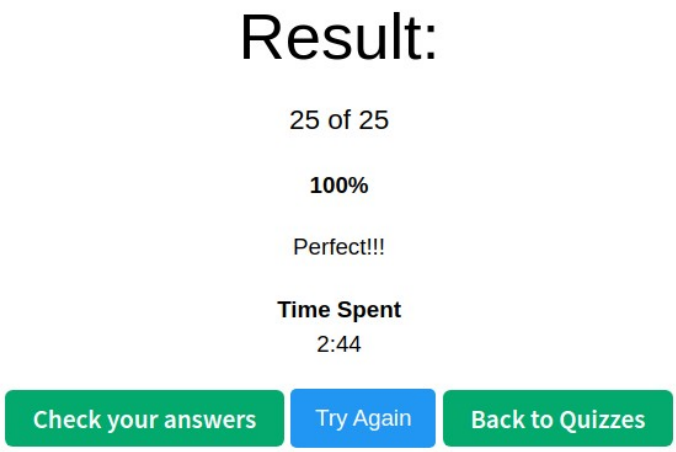
Hw2

NumPy Quiz



Pandas Quiz



Numpy Notes and Pandas Notes on following pages

Numpy Notes

Numpy Tutorial

The array object in NumPy is called `ndarray`

The version string is stored under `__version_` attribute.

```
print(np.__version__) this is a double _
```

The array object in NumPy is called `ndarray`

NumPy has a whole sub module dedicated towards matrix operations called `numpy.mat`

NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array have.

When the array is created, you can define the number of dimensions by using the `ndmin` argument.

Below is a list of all data types in NumPy and the characters used to represent them.

- `i` - integer
- `b` - boolean
- `u` - unsigned integer
- `f` - float
- `c` - complex float
- `m` - timedelta
- `M` - datetime
- `O` - object
- `S` - string
- `U` - unicode string
- `V` - fixed chunk of memory for other type (void)

The NumPy array object has a property called `dtype` that returns the data type of the array

Create an array with data type string: example
`arr = np.array([1, 2, 3, 4], dtype='S')`

For `i`, `u`, `f`, `S` and `U` we can define size as well. `Example dtype='i4'`

The `astype()` function creates a copy of the array, and allows you to specify the data type as a parameter.

`What is a view array used for?`

Every NumPy array has the attribute `base` that returns `None` if the array owns the data.

NumPy arrays have an attribute called `shape` that returns a tuple with each index having the number of corresponding elements.

reshape we can add or remove dimensions or change number of elements in each dimension.

Flattening array means converting a multidimensional array into a 1D array.

We can use `reshape(-1)` to do this.

The function `nditer()` is a helping function that can be used from very basic to very advanced iterations. Iterations mean going through each arrray.

We can use `op_dtypes` argument and pass it the expected datatype to change the datatype of elements while iterating. NumPy does not change the data type of the element in-place (where the element is in array) so it needs some other space to perform this action, that extra space is called buffer, and in order to enable it in `nditer()` we pass `flags=['buffered']`.

Sometimes we require corresponding index of the element while iterating, the `ndenumerate()` method can be used for those usecases.

We pass a sequence of arrays that we want to join to the `concatenate()` function, along with the axis. If axis is not explicitly passed, it is taken as 0.

We pass a sequence of arrays that we want to join to the `stack()` method along with the axis. If axis is not explicitly passed it is taken as 0.

NumPy provides a helper function: `hstack()` to stack along rows.

NumPy provides a helper function: `vstack()` to stack along columns.

NumPy provides a helper function: `dstack()` to stack along height, which is the same as depth.

We use `array_split()` for splitting arrays, we pass it the array we want to split and the number of splits.

An alternate solution is using `hsplit()` opposite of `hstack()`

Similar alternates to `vstack()` and `dstack()` are available as `vsplit()` and `dsplit()`.

To search an array, use the `where()` method.

There is a method called `searchsorted()` which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order. The `searchsorted()` method is assumed to be used on sorted arrays.

By default the left most index is returned, but we can give `side='right'` to return the right most index instead.

The NumPy ndarray object has a function called `sort()`, that will sort a specified array. This method returns a copy of the array, leaving the original array unchanged.

If the value at an index is `True` that element is contained in the filtered array, if the value at that index is `False` that element is excluded from the filtered array.

example

```
import numpy as np
arr = np.array([41, 42, 43, 44])
```

```
x = [True, False, True, False]
newarr = arr[x]
print(newarr)
```

```
[41, 43]
```

NumPy Random

NumPy offers the `random` module to work with random numbers.

he random module's `rand()` method returns a random float between 0 and 1
The `rand()` method also allows you to specify the shape of the array.

The `randint()` method takes a `size` parameter where you can specify the shape of an array.

The `choice()` method allows you to generate a random value based on an array of values.

The `choice()` method takes an array as a parameter and randomly returns one of the values The `choice()` method also allows you to return an array of values.

Add a `size` parameter to specify the shape of the array.

Probability Density Function: A function that describes a continuous probability. i.e. probability of all values in an array.

A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a permutation of [1, 2, 3] and vice-versa.

The NumPy Random module provides two methods for this: `shuffle()` and `permutation()`. The `shuffle()` method makes changes to the original array. The `permutation()` method *returns* a re-arranged array (and leaves the original array un-changed).

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

Normal Distribution

The Normal Distribution is one of the most important distributions.

It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss.

It fits the probability distribution of many events, eg. IQ Scores, Heartbeat etc.

Use the `random.normal()` method to get a Normal Data Distribution.

It has three parameters:

`loc` - (Mean) where the peak of the bell exists.

`scale` - (Standard Deviation) how flat the graph distribution should be.

`size` - The shape of the returned array.

Binomial Distribution

Binomial Distribution is a Discrete Distribution.

It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails.

It has three parameters:

`n` - number of trials.

`p` - probability of occurence of each trial (e.g. for toss of a coin 0.5 each).

`size` - The shape of the returned array.

Discrete Distribution:The distribution is defined at separate set of events, e.g. a coin toss's result is discrete as it can be only head or tails whereas height of people is continuous as it can be 170, 170.1, 170.11 and so on.

The main difference is that normal distribution is continous whereas binomial is discrete, but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.

Poisson Distribution

Poisson Distribution is a Discrete Distribution.

It estimates how many times an event can happen in a specified time. e.g. If someone eats twice a day what is probability he will eat thrice?

It has two parameters:

`lam` - rate or known number of occurences e.g. 2 for above problem.

`size` - The shape of the returned array.

he difference is very subtle it is that, binomial distribution is for discrete trials, whereas poisson distribution is for continuous trials.

But for very large `n` and near-zero `p` binomial distribution is near identical to poisson distribution such that `n * p` is nearly equal to `lam`.

Uniform Distribution

Used to describe probability where every event has equal chances of occuring.

E.g. Generation of random numbers.

It has three parameters:

`a` - lower bound - default 0 .0.

`b` - upper bound - default 1.0.

`size` - The shape of the returned array.

Logistic Distribution

Logistic Distribution is used to describe growth.

Used extensively in machine learning in logistic regression, neural networks etc.

It has three parameters:

`loc` - mean, where the peak is. Default 0.

`scale` - standard deviation, the flatness of distribution. Default 1.

`size` - The shape of the returned array.

Both distributions are near identical, but logistic distribution has more area under the tails. ie. It representage more possibility of occurence of an events further away from mean.

For higher value of scale (standard deviation) the normal and logistic distributions are near identical apart from the peak.

Multinomial Distribution

Multinomial distribution is a generalization of binomial distribution.

It describes outcomes of multi-nomial scenarios unlike binomial where scenarios must be only one of two. e.g. Blood type of a population, dice roll outcome.

It has three parameters:

`n` - number of possible outcomes (e.g. 6 for dice roll).

`pvals` - list of probabilties of outcomes (e.g. [1/6, 1/6, 1/6, 1/6, 1/6, 1/6] for dice roll).

`size` - The shape of the returned array.

Note: Multinomial samples will NOT produce a single value! They will produce one value for each `pval`.

Note: As they are generalization of binomial distribution their visual representation and similarity of normal distribution is same as that of multiple binomial distributions.

Exponential Distribution

Exponential distribution is used for describing time till next event e.g. failure/success etc.

It has two parameters:

`scale` - inverse of rate (see lam in poisson distribution) defaults to 1.0.

`size` - The shape of the returned array.

Poisson distribution deals with number of occurences of an event in a time period whereas exponential distribution deals with the time between these events.

Chi Square Distribution

Chi Square distribution is used as a basis to verify the hypothesis.

It has two parameters:

`df` - (degree of freedom).

`size` - The shape of the returned array.

Rayleigh Distribution

Rayleigh distribution is used in signal processing.

It has two parameters:

`scale` - (standard deviation) decides how flat the distribution will be default 1.0).

`size` - The shape of the returned array.

At unit stddev the and 2 degrees of freedom rayleigh and chi square represent the same distributions.

Pareto Distribution

A distribution following Pareto's law i.e. 80-20 distribution (20% factors cause 80% outcome).

It has two parameter:

`a` - shape parameter.

`size` - The shape of the returned array

Zipf distritutions are used to sample data based on zipf's law.

Zipf's Law: In a collection the nth common term is 1/n times of the most common term. E.g. 5th common word in english has occurs nearly 1/5th times as of the most used word.

It has two parameters:

`a` - distribution parameter.

`size` - The shape of the returned array.

NumPy ufuncs

Adding the elements of two functon can be done with Python's `zip()` function or with ufuncs `add(x, y)`

How To Create Your Own ufunc

To create you own ufunc, you have to define a function, like you do with normal functions in Python, then you add it to your NumPy ufunc library with the `frompyfunc()` method.

The `frompyfunc()` method takes the following arguments:

1. `function` - the name of the function.
2. `inputs` - the number of input arguments (arrays).
3. `outputs` - the number of output arrays.

Check the type of a function to check if it is a ufunc or not.

A ufunc should return `<class 'numpy.ufunc'>`.

To test if the function is a ufunc in an if statement, use the `numpy.ufunc` value (or `np.ufunc` if you use np as an alias for numpy):

Arithmetic Conditionally: means that we can define conditions where the arithmetic operation should happen.

All of the discussed arithmetic functions take a `where` parameter in which we can specify that condition.

The `add()` function sums the content of two arrays, and return the results in a new array.

The `subtract()` function subtracts the values from one array with the values from another array, and return the results in a new array.

The `multiply()` function multiplies the values from one array with the values from another array, and return the results in a new array.

The `divide()` function divides the values from one array with the values from another array, and return the results in a new array.

The `power()` function rises the values from the first array to the power of the values of the second array, and return the results in a new array.

Both the `mod()` and the `remainder()` functions return the remainder of the values in the first array corresponding to the values in the second array, and return the results in a new array.

The `divmod()` function return both the quotient and the the mod. The return value is two arrays, the first array contains the quotient and second array contains the mod.

Both the `absolute()` and the `abs()` functions functions do the same absolute operation element-wise but we should use `absolute()` to avoid confusion with python's inbuilt `math.abs()`

Rounding Decimals

There are primarily five ways of rounding off decimals in NumPy:

- truncation
- fix
- rounding
- floor
- ceil

Remove the decimals, and return the float number closest to zero. Use the `trunc()` and `fix()` functions.

The `around()` function increments preceding digit or decimal by 1 if >=5 else do nothing.

E.g. round off to 1 decimal point, 3.16666 is 3.2

The `floor()` function rounds off decimal to nearest lower integer.
Note: The `floor()` function returns floats, unlike the `trunc()` function who returns integers.

The ceil() function rounds off decimal to nearest upper integer.

Logs

NumPy provides functions to perform log at the base 2, e and 10.

We will also explore how we can take log for any base by creating a custom ufunc.

All of the log functions will place -inf or inf in the elements if the log can not be computed.

Use the `log2()` function to perform log at the base 2. Note: The `arange(1, 10)` function returns an array with integers starting from 1 (included) to 10 (not included).

Use the `log10()` function to perform log at the base 10.

Use the `log()` function to perform log at the base e.

NumPy does not provide any function to take log at any base, so we can use the `frompyfunc()` function along with inbuilt function `math.log()` with two input parameters and one output parameter:

sum() is summation over n elements

If you specify `axis=1`, NumPy will sum the numbers in each array.

Cummulative sum means partially adding the elements in array.

E.g. The partial sum of [1, 2, 3, 4] would be [1, 1+2, 1+2+3, 1+2+3+4] = [1, 3, 6, 10].

Perfom partial sum with the `cumsum()` function

To find the product of the elements in an array, use the `prod()` function. If you specify `axis=1`, NumPy will return the product of each array.

Cummulative product means taking the product partially.

E.g. The partial product of [1, 2, 3, 4] is [1, 1*2, 1*2*3, 1*2*3*4] = [1, 2, 6, 24]

Perfom partial sum with the `cumprod()` function.

A discrete difference means subtracting two successive elements.

E.g. for [1, 2, 3, 4], the discrete difference would be [2-1, 3-2, 4-3] = [1, 1, 1]

To find the discrete difference, use the `diff()` function.

lcm() - The Lowest Common Multiple is the least number that is common multiple of both of the numbers.

To find the Lowest Common Multiple of all values in an array, you can use the `reduce()` method.

The `reduce()` method will use the ufunc, in this case the `lcm()` function, on each element, and reduce the array by one dimension.

gcd() - The GCD (Greatest Common Denominator), also known as HCF (Highest Common Factor) is the biggest number that is a common factor of both of the numbers.

To find the Highest Common Factor of all values in an array, you can use the `reduce()` method.

The `reduce()` method will use the ufunc, in this case the `gcd()` function, on each element, and reduce the array by one dimension.

NumPy provides the ufuncs `sin()`, `cos()` and `tan()` that take values in radians and produce the corresponding sin, cos and tan values.

deg2rad() - convert degrees to radians

rad2deg() - convert radians to degrees

NumPy provides ufuncs `arcsin()`, `arccos()` and `arctan()` that produce radian values for corresponding sin, cos and tan values given.

NumPy provides the `hypot()` function that takes the base and perpendicular values and produces hypotenues based on pythagoras theorem

NumPy provides the ufuncs `sinh()`, `cosh()` and `tanh()` that take values in radians and produce the corresponding sinh, cosh and tanh values

Numpy provides ufuncs `arcsinh()`, `arccosh()` and `arctanh()` that produce radian values for corresponding sinh, cosh and tanh values given.

We can use NumPy's `unique()` method to find unique elements from any array. E.g. create a set array, but remember that the set arrays should only be 1-D arrays. Eliminates repeat elements in 1-D array.

To find the unique values of two arrays, use the `union1d()` method.

To find only the values that are present in both arrays, use the `intersect1d()` method. Note: the `intersect1d()` method takes an optional argument `assume_unique`, which if set to True can speed up computation. It should always be set to True when dealing with sets.

To find only the values in the first set that is NOT present in the seconds set, use the `setdiff1d()` method. Note: the `setdiff1d()` method takes an optional argument `assume_unique`, which if set to True can speed up computation. It should always be set to True when dealing with sets.

To find only the values that are NOT present in BOTH sets, use the `setxor1d()` method. Note: the `setxor1d()` method takes an optional argument `assume_unique`, which if set to True can speed up computation. It should always be set to True when dealing with sets.

Pandas Notes

Pandas is usually imported under the `pd` alias.

Series() - A Pandas Series is like a column in a table.

With the `index` argument, you can name your own labels.
`Ex: calories = {"day1": 420, "day2": 380, "day3": 390}`

You can also use a key/value object, like a dictionary, when creating a Series. To select only some of the items in the dictionary, use the `index` argument and specify only the items you want to include in the Series

Data sets in Pandas are usually multi-dimensional tables, called DataFrames. df

Series is like a column, a DataFrame is the whole table

DataFrame is like a table with rows and columns.

Pandas use the `loc` attribute to return one or more specified row(s)

With the `index` argument, you can name your own indexes.

Use the named index in the `loc` attribute to return the specified row(s).

If your data sets are stored in a file, Pandas can load them into a DataFrame.

```
import pandas as pd

df = pd.read_csv('data.csv')
```

```
print(df)
```

```
example with a csv file
```

Tip: use `to_string()` to print the entire DataFrame.

```
Big data sets are often stored, or extracted as JSON.
```

JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.

In our examples we will be using a JSON file called 'data.json'.

JSON = Python Dictionary

JSON objects have the same format as Python dictionaries.

If your JSON code is not in a file, but in a Python Dictionary, you can load it into a DataFrame directly

One of the most used method for getting a quick overview of the DataFrame, is the `head()` method.

The `head()` method returns the headers and a specified number of rows, starting from the top. Note: if the number of rows is not specified, the `head()` method will return the top 5 rows.

There is also a `tail()` method for viewing the last rows of the DataFrame.

The `tail()` method returns the headers and a specified number of rows, starting from the bottom.

The DataFrames object has a method called `info()`, that gives you more information about the data set.

The `info()` method also tells us how many Non-Null values there are present in each column

```
Data cleaning means fixing bad data in your data set.
```

Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

Note: By default, the `dropna()` method returns a *new* DataFrame, and will not change the original.

The result from the converting in the example above gave us a NaT value, which can be handled as a NULL value, and we can remove the row by using the `dropna()` method.

If you want to change the original DataFrame, use the `inplace = True` argument:

Note: Now, the `dropna(inplace = True)` will NOT return a new DataFrame, but it will remove all rows containg NULL values from the original DataFrame.

Another way of dealing with empty cells is to insert a new value instead.

This way you do not have to delete entire rows just because of some empty cells.

The `fillna()` method allows us to replace empty cells with a value:

`A common way to replace empty cells, is to calculate the mean, median or mode value of the column.`

Pandas uses the `mean()` `median()` and `mode()` methods to calculate the respective values for a specified column:

`Cells with data of wrong format can make it difficult, or even impossible, to analyze data.`

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

Let's try to convert all cells in the 'Date' column into dates.

Pandas has a `to_datetime()` method for this:

Another way of handling wrong data is to remove the rows that contains wrong data.

drop() -

To discover duplicates, we can use the `duplicated()` method.

The `duplicated()` method returns a Boolean values for each row:

To remove duplicates, use the `drop_duplicates()` method.
Remember: The `(inplace = True)` will make sure that the method does NOT return a *new* DataFrame, but it will remove all duplicates from the *original* DataFrame.

A great aspect of the Pandas module is the `corr()` method.

The `corr()` method calculates the relationship between each column in your data set.
Note: The `corr()` method ignores "not numeric" columns.

`Result Explained`

The Result of the `corr()` method is a table with a lot of numbers that represents how well the relationship is between two columns.

The number varies from -1 to 1.

1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.

0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.

-0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.

0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

What is a good correlation? It depends on the use, but I think it is safe to say you have to have at least `0.6` (or `-0.6`) to call it a good correlation.

Perfect Correlation:

We can see that "Duration" and "Duration" got the number `1.000000`, which makes sense, each column always has a perfect relationship with itself.

Good Correlation:

"Duration" and "Calories" got a `0.922721` correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long work out.

Bad Correlation:

"Duration" and "Maxpulse" got a `0.009403` correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.

Pandas uses the `plot()` method to create diagrams.

We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.

Scatter Plot

Specify that you want a scatter plot with the `kind` argument:

```
kind = 'scatter'
```

A scatter plot needs an x- and a y-axis.

Use the `kind` argument to specify that you want a histogram:

```
kind = 'hist'
```

A histogram needs only one column.

A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?