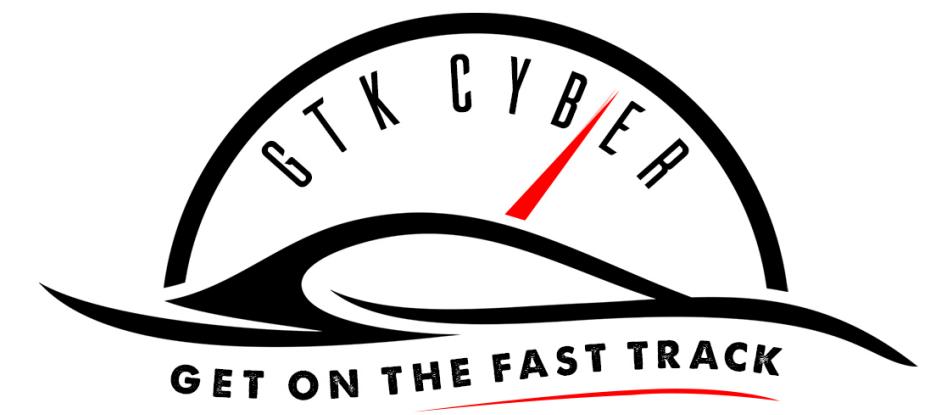
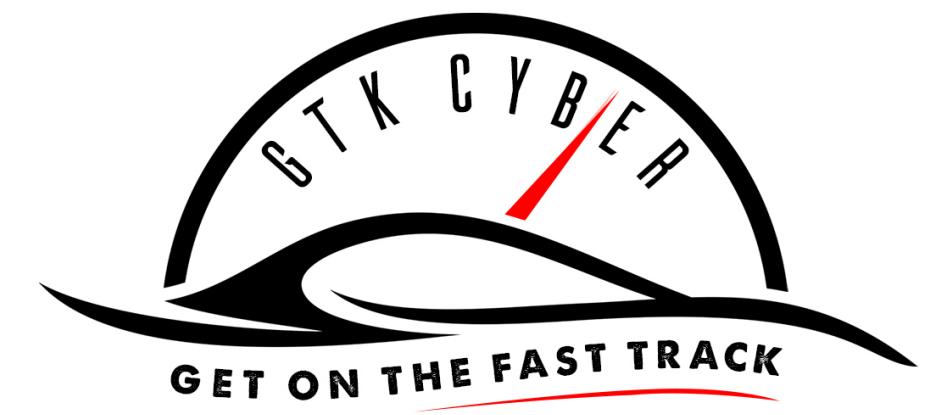


Module 1: Exploratory Data Analysis in One Dimension

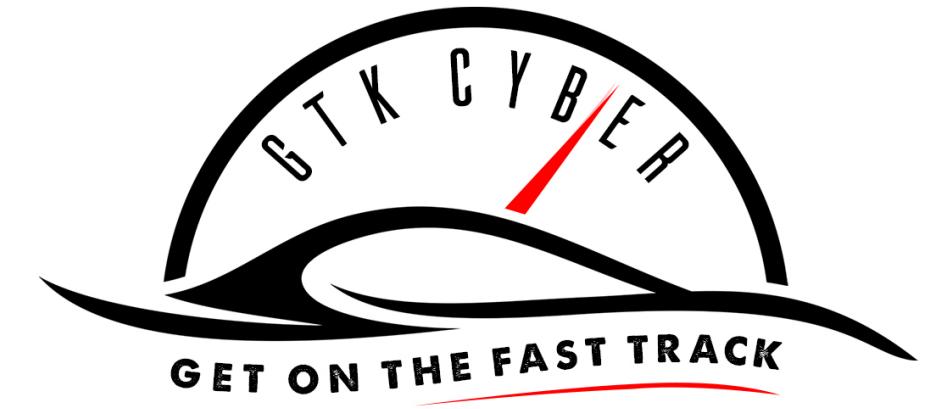
GET ON THE FAST TRACK



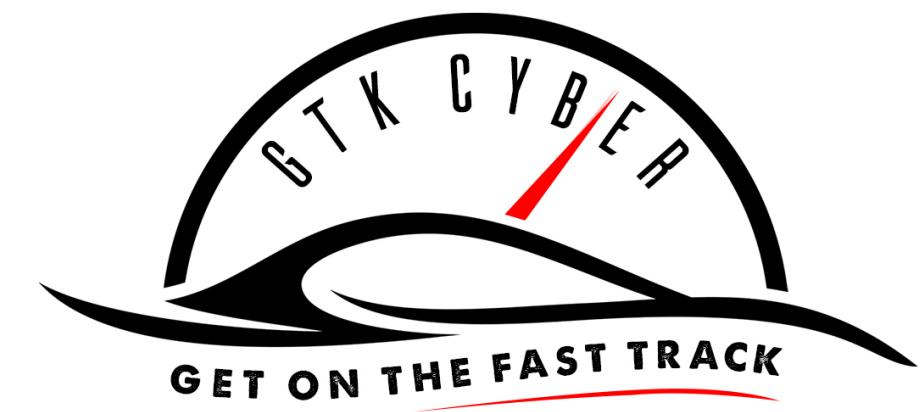
Vectorized Data Structures



What are they?

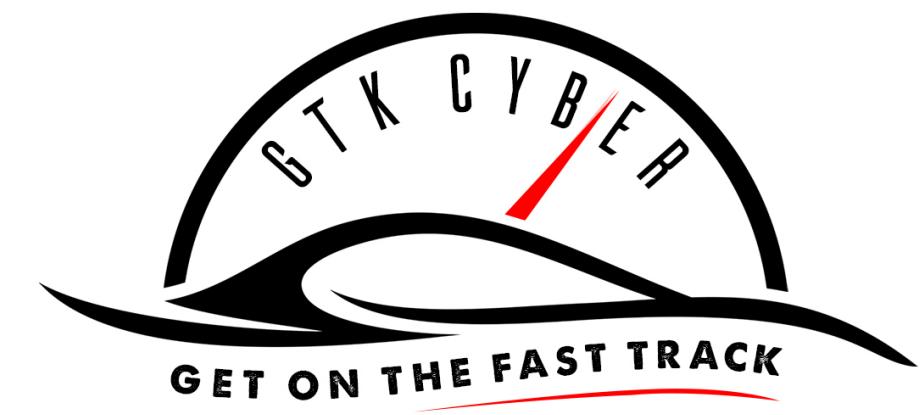


Vectorized Data Structures let you
perform operations on your data
all at once

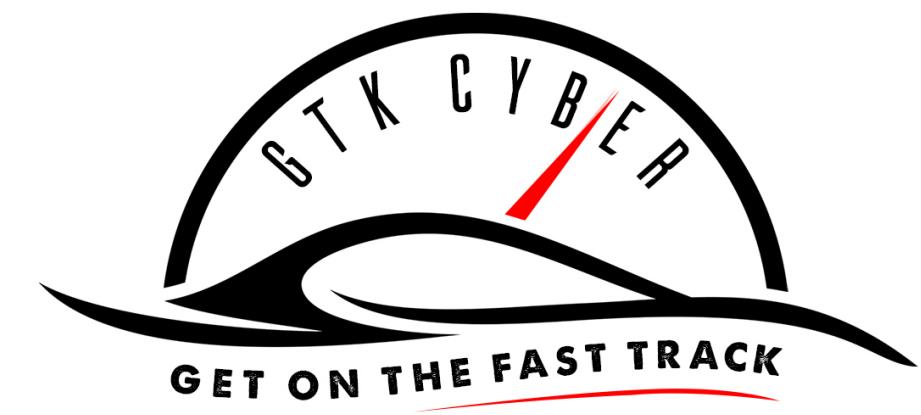


Advantages of Vectorized Data Structures

- More readable and simpler code
- Allows the module to optimize for CPU and Memory usage
- Pass serialized objects from one system to another

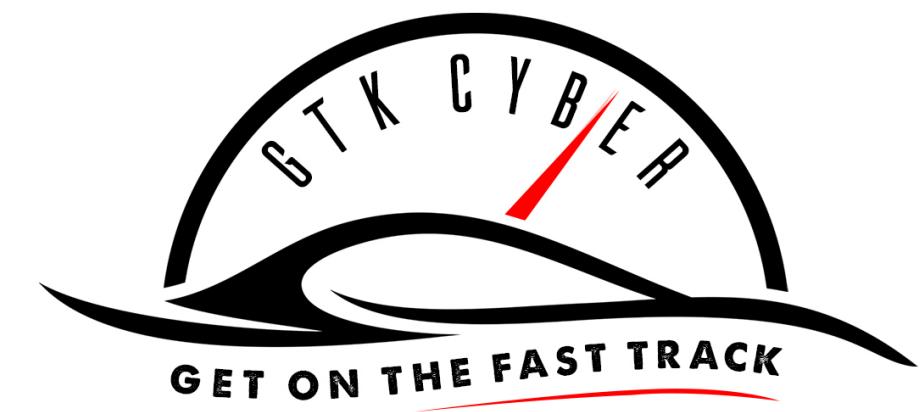


```
for x in range(0, len( data )):  
    data[ x ] = data[ x ] + 1
```

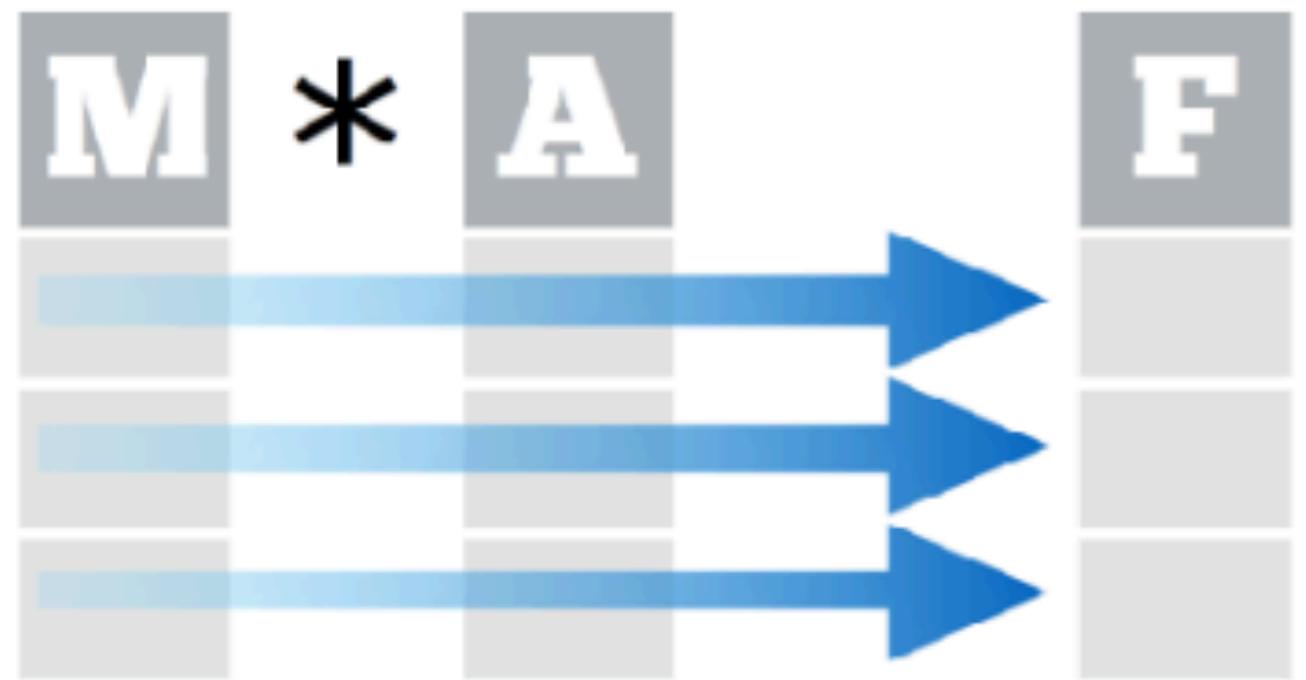


```
for x in range(0, len( data )):  
    data[ x ] = data[ x ] + 1
```

odds = evens + 1



$$F = M * A$$



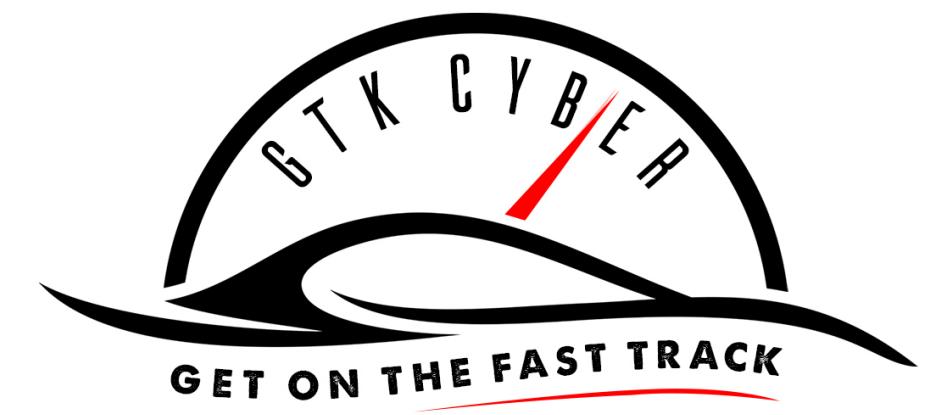
Each **variable** is saved
in its own **column**



&



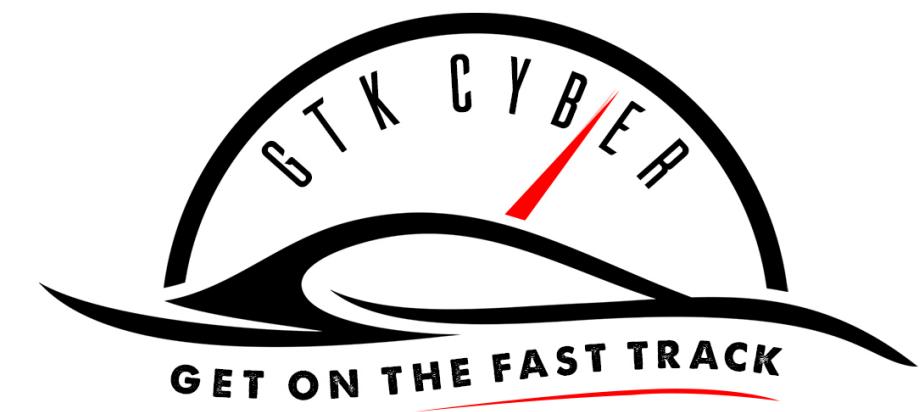
Each **observation** is
saved in its own **row**



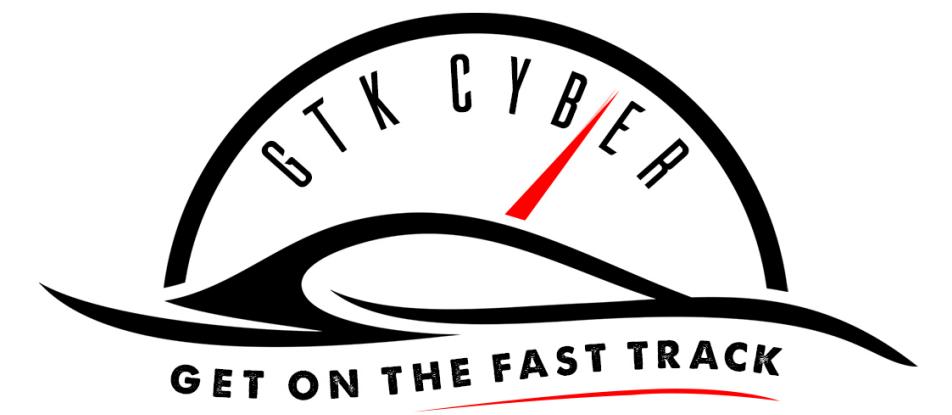
No loops



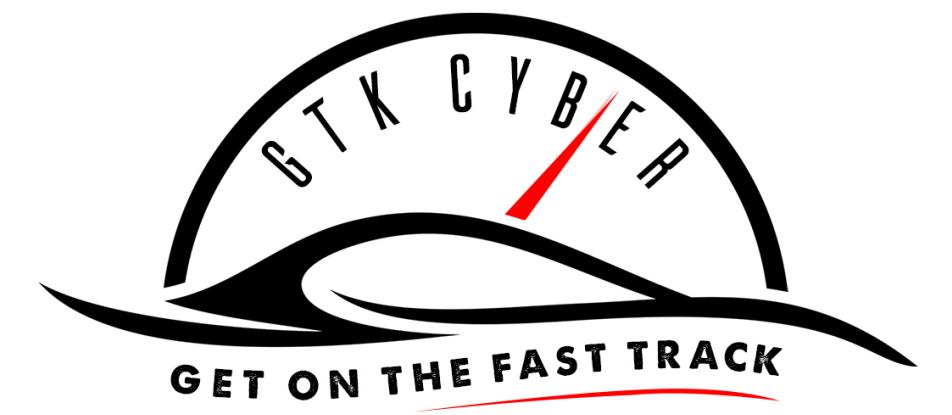




Dimensions	Name	Description
1	Series	Indexed 1 dimensional data structure
1	Timeseries	Series using timestamps as an index
2	DataFrame	A two dimensional table
3	Panel	A three dimensional mutable data structure



The Series Object



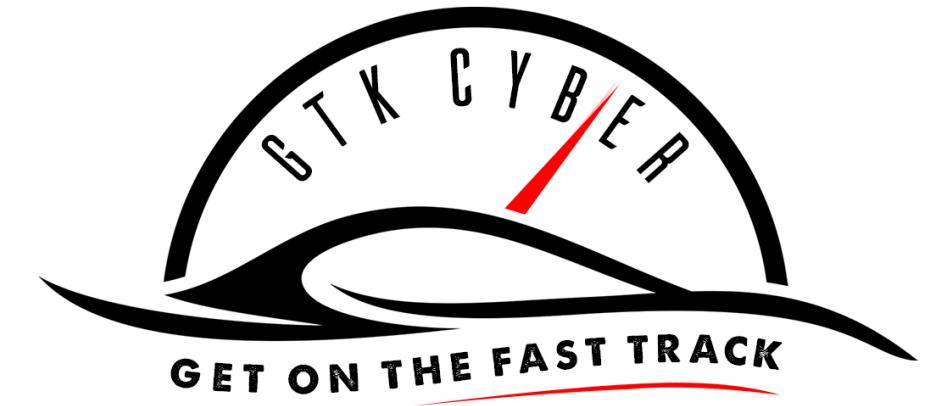
A **Series** is like a list



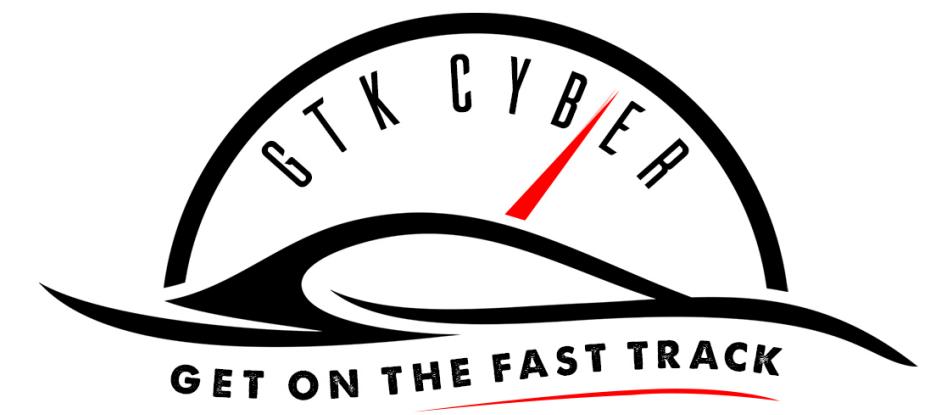
A **Series** is like a list or a
dictionary



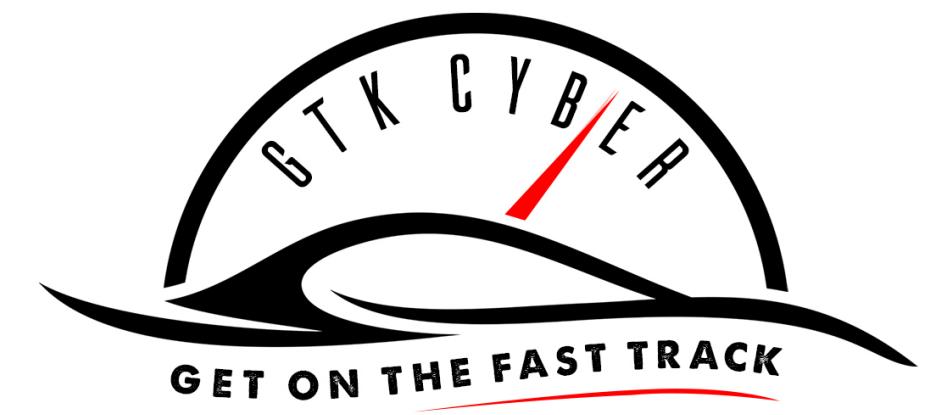
A **Series** is like a list or a
dictionary **but BETTER!!**



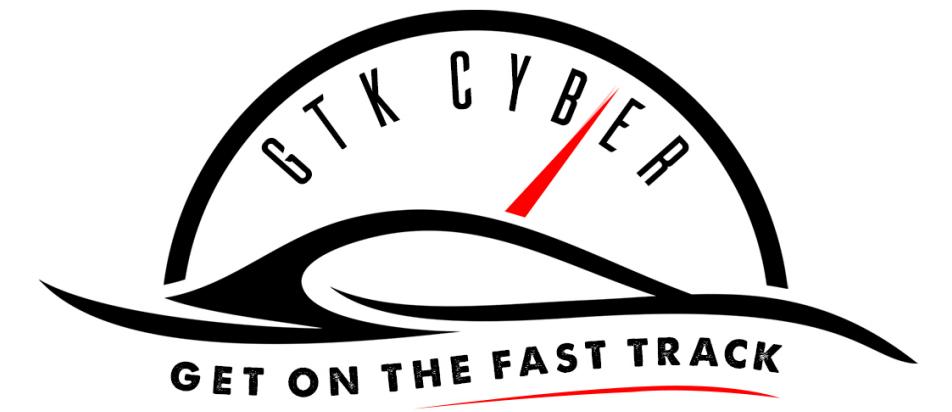
The **Series** is the **primary building block** for all the other data structures we will discuss



```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```



```
myData = pd.Series( <data>, index=<index> )
```



Creating a Series

```
series1 = pd.Series( [ 'a' , 'b' , 'c' , 'd' , 'e' ] )
```

```
series2 = pd.Series( { 'firstName' : 'Charles' ,  
'lastName' : 'Givre' } )
```



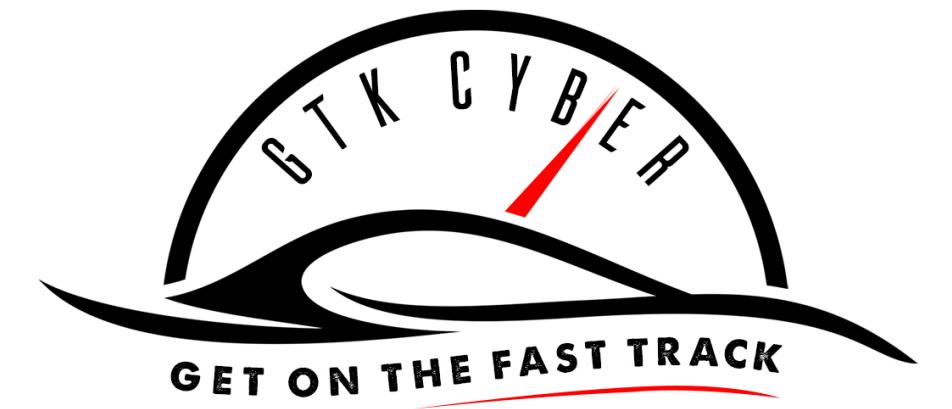
Accessing a Series

```
series1 = pd.Series( [ 'a' , 'b' , 'c' , 'd' , 'e' ] )
```

```
print( series1 )
```

0	a
1	b
2	c
3	d
4	e

dtype: object



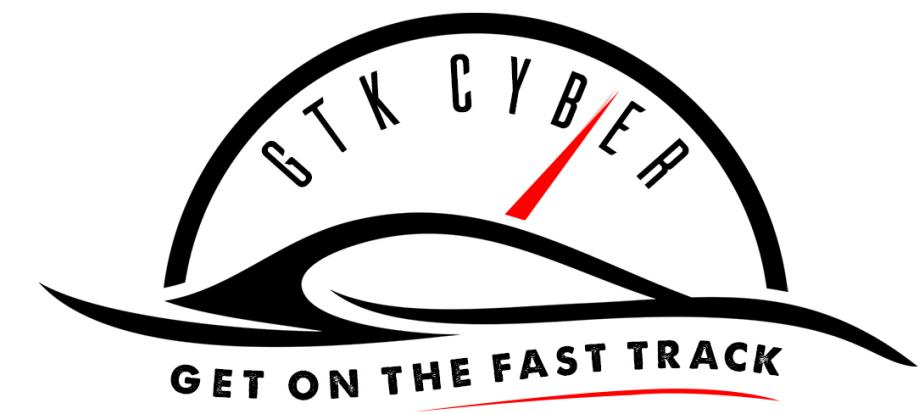
Accessing a Series

You can access items in a Pandas Series by index, similar to a python list. Just as in a regular list, indexing starts at zero.

```
series1 = pd.Series( [ 'a' , 'b' , 'c' , 'd' , 'e' ] )
```

```
print( series1[3] )
```

d



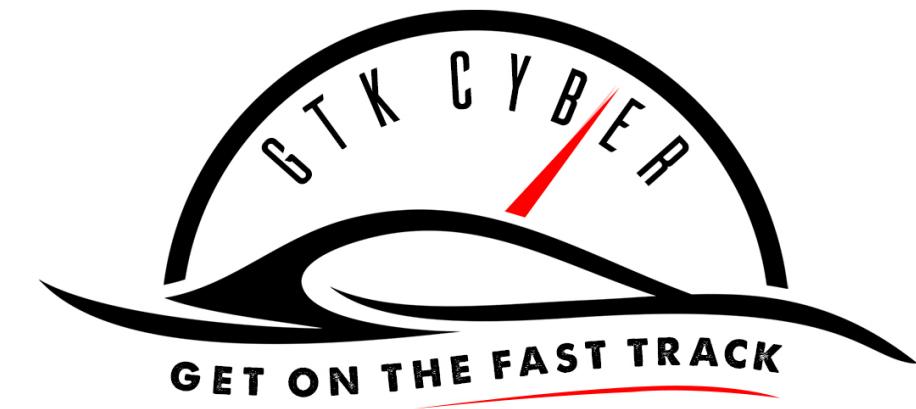
Accessing a Series: Slicing

```
series1 = pd.Series( [ 'a' , 'b' , 'c' , 'd' , 'e' ] )
```

```
print( series1[1:3] )
```

1	b
2	c

dtype: object



Accessing a Series

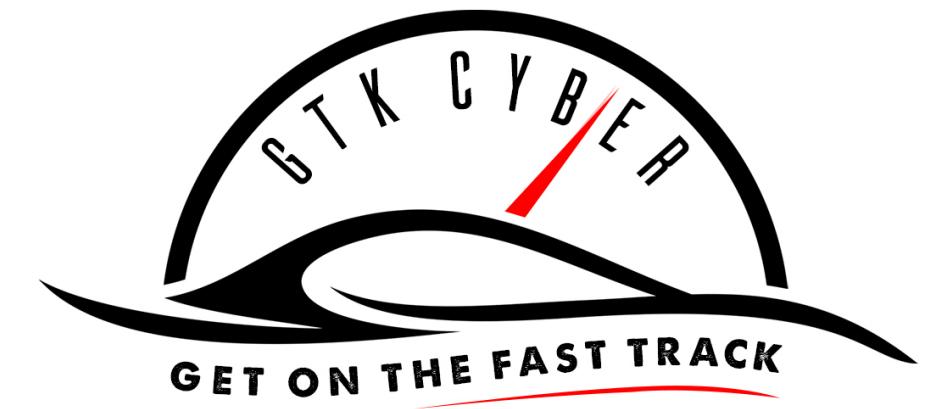
```
series1 = pd.Series( [ 'a' , 'b' , 'c' , 'd' , 'e' ] )
```

```
print( series1[1:3] )
```

1	b
2	c

dtype: object

```
series2 = series1[1:3]  
series2.reset_index( drop=True )
```

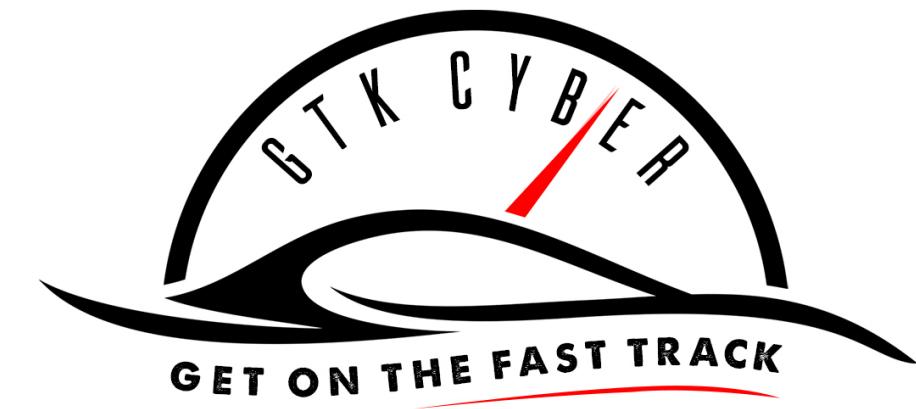


Accessing a Series

```
record = pd.Series(  
    { 'firstname': 'Charles',  
      'lastname': 'Givre',  
      'middle': 'classified' } )
```

```
record[ 'firstname' ]
```

Charles

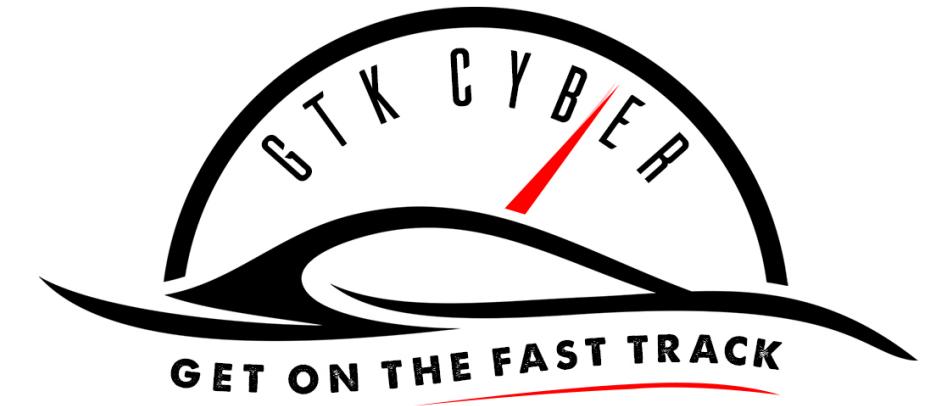


Accessing a Series

```
record = pd.Series(  
    {'firstname': 'Charles',  
     'lastname': 'Givre',  
     'middle': 'classified' } )
```

```
record[ 'firstname', 'lastname' ]
```

```
firstname      Charles  
lastname       Givre  
dtype: object
```

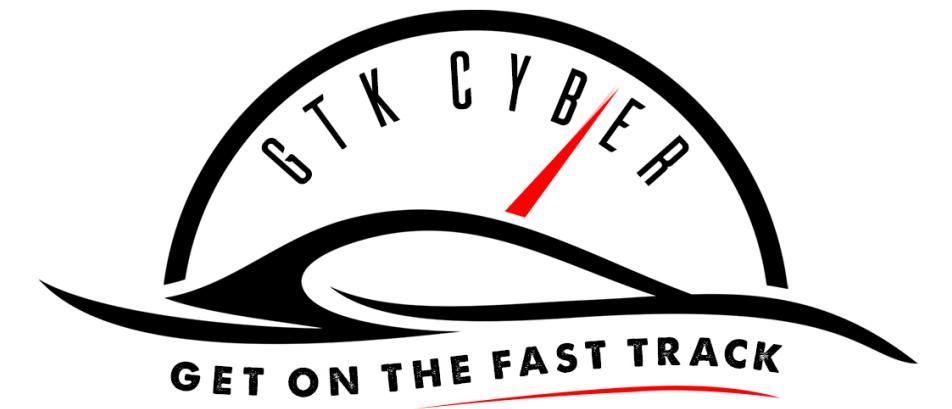


Accessing a Series

```
record = pd.Series(  
    {'firstname': 'Charles',  
     'lastname': 'Givre',  
     'middle': 'classified' } )
```

```
record[0]
```

Charles



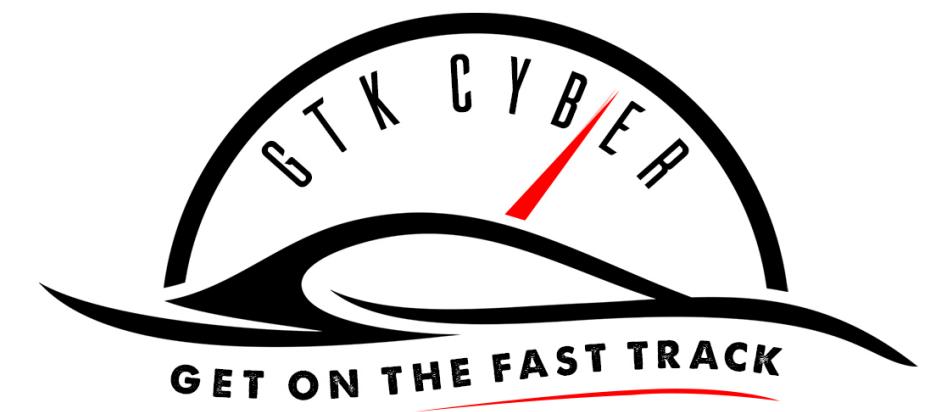
Accessing a Series

```
randomNumbers = pd.Series(  
    np.random.random_integers(1, 100, 50) )
```

```
randomNumbers.head()
```

```
0      48  
1      34  
2      84  
3      85  
4      58
```

```
dtype: int64
```



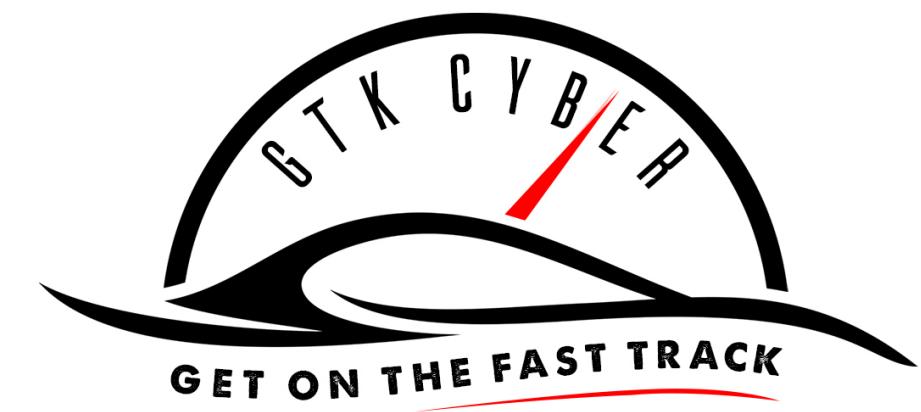
Accessing a Series

```
randomNumbers = pd.Series(  
    np.random.randint(1, 100, 50) )
```

```
randomNumbers.tail( 7 )
```

```
43      66  
44      66  
45      43  
46      55  
47      99  
48      82  
49      19
```

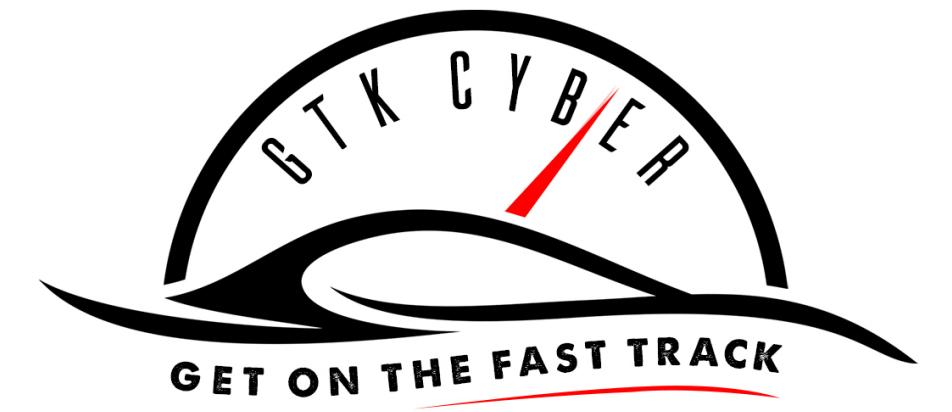
```
dtype: int64
```



Filtering Data in a Series

```
randomNumbers [ <boolean condition> ]  
randomNumbers [ randomNumbers < 10 ]
```

```
12      5  
21      2  
24      1  
27      1  
29      1  
dtype: int64
```



```
record.str.contains('Cha')
```

```
firstname      True
```

```
lastname      False
```

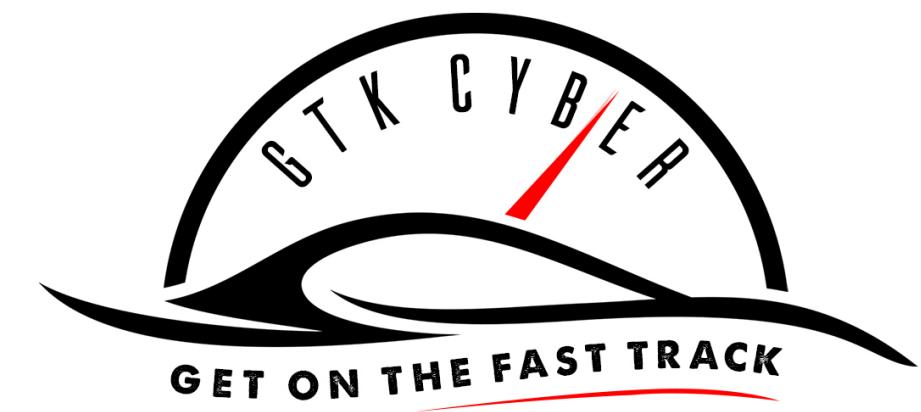
```
middle        False
```

```
dtype: bool
```

```
record[ record.str.contains('Cha') ]
```

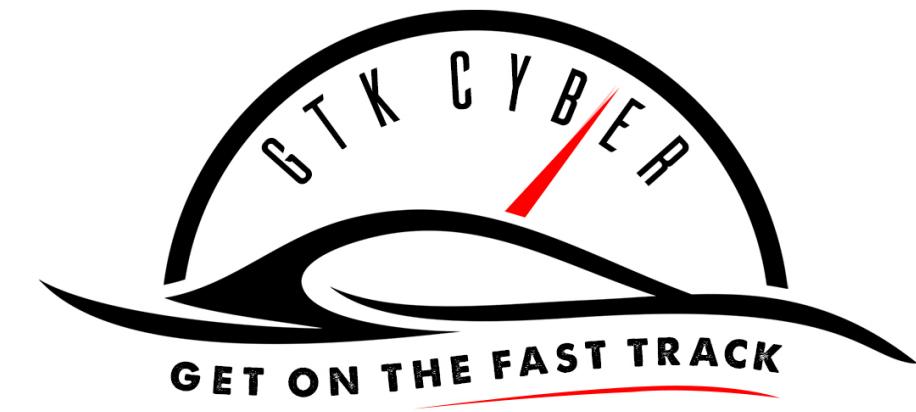
```
firstname    Charles
```

```
dtype: object
```



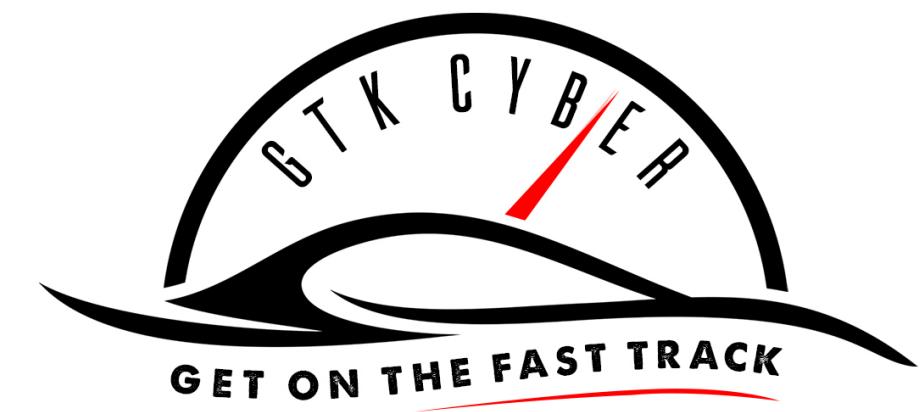
String Functions

Function	Explanation
Series.str.contains(<pattern>)	Returns true/false if text matches a pattern
Series.str.count(<pattern>)	Returns number of occurrences of a pattern in a string
Series.str.extract(<pattern>)	Returns matching groups from a string
Series.str.find(<string>)	Returns index of first occurrences of a substring (Note: not regex)
Series.str.findall(<pattern>)	Returns all occurrences of a regex
Series.str.len()	Returns the length of text
Series.str.replace(<pat>, <replace>)	Replaces matches with a replacement string



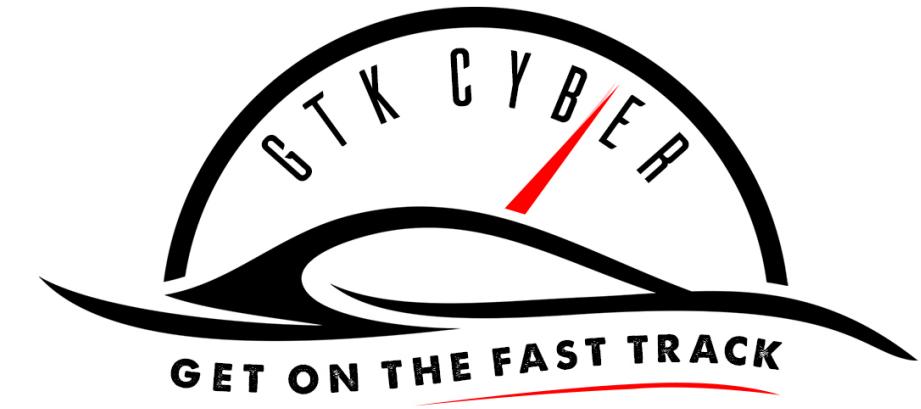
Date/Time Operations

```
pd.to_datetime(<times>, format='<format string>')
```



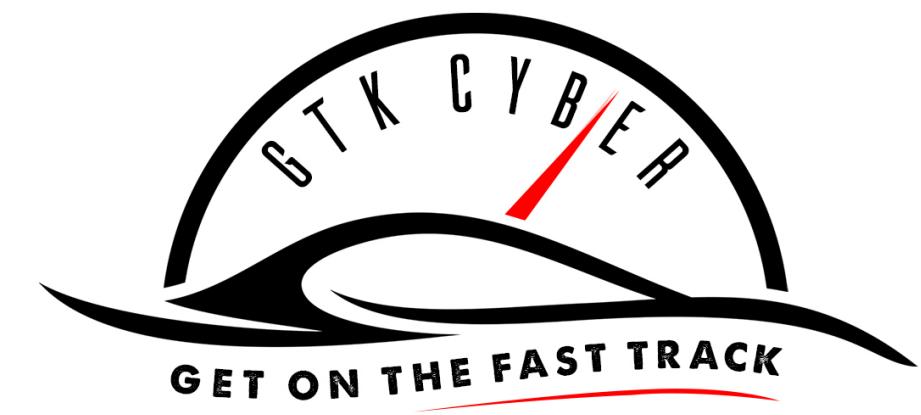
Selected Date/Time Operations

Function	Explanation
<code>series.dt.year</code>	Returns the year of the date time
<code>series.dt.month</code>	Returns the month of the date time
<code>series.dt.day</code>	Returns the days of the date time
<code>series.dt.weekday</code>	Returns the day of the week
<code>series.dt.is_month_start</code>	Returns true if it is the first day of the month
<code>series.dt.strftime()</code>	Returns an array of formatted strings specified by a <code>date_format</code>

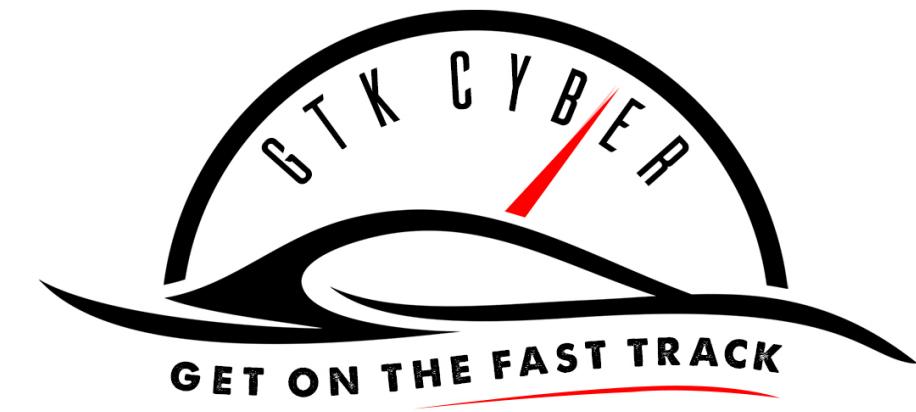


Manipulating Data

Pandas Data Structures allow you
to **perform operations**
on your **entire data set at once.**

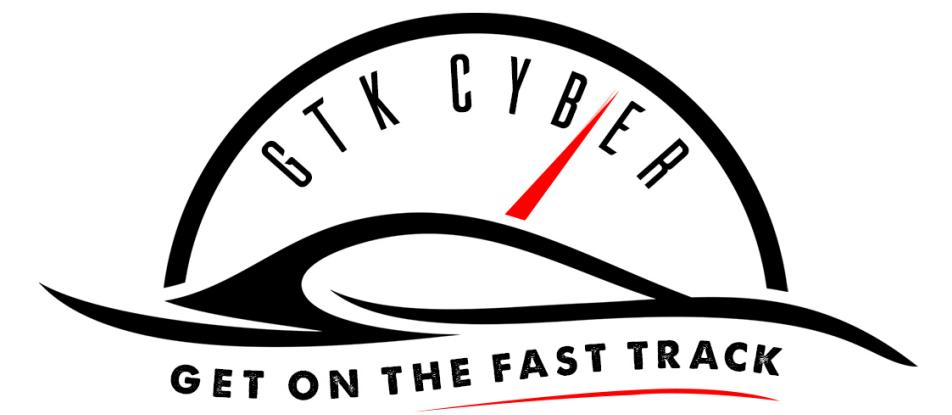


```
for x in range(0, len( data )):  
    data[ x ] = data[ x ] + 1
```

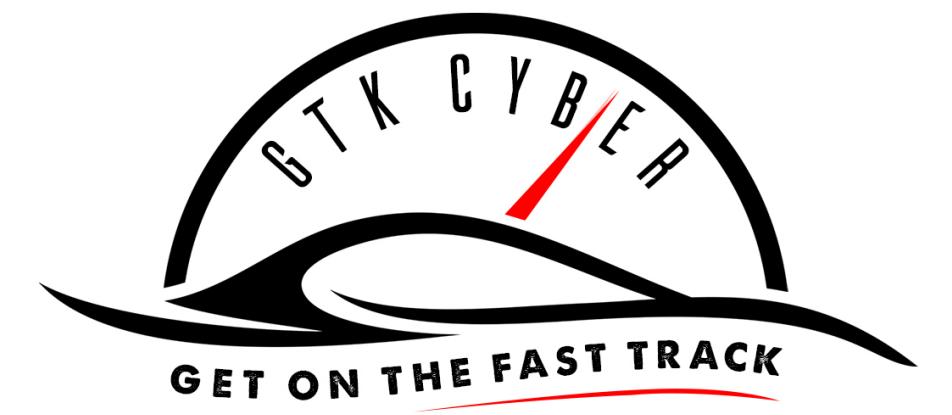


```
for x in range(0, len( data )):  
    data[ x ] = data[ x ] + 1
```

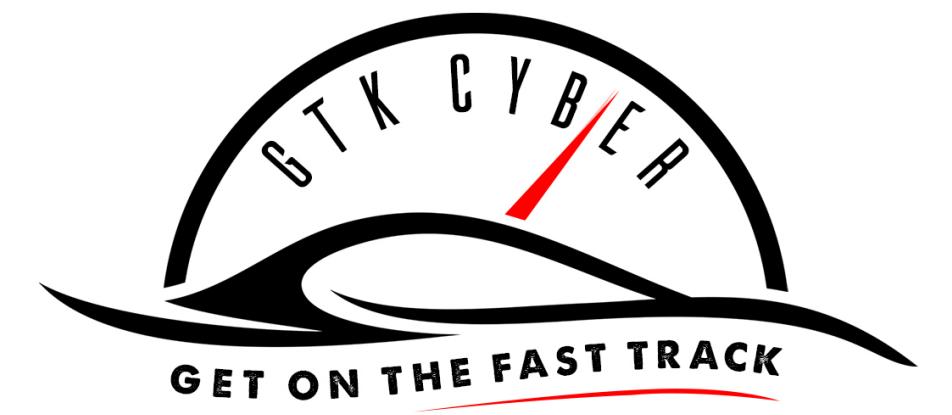
odds = evens + 1



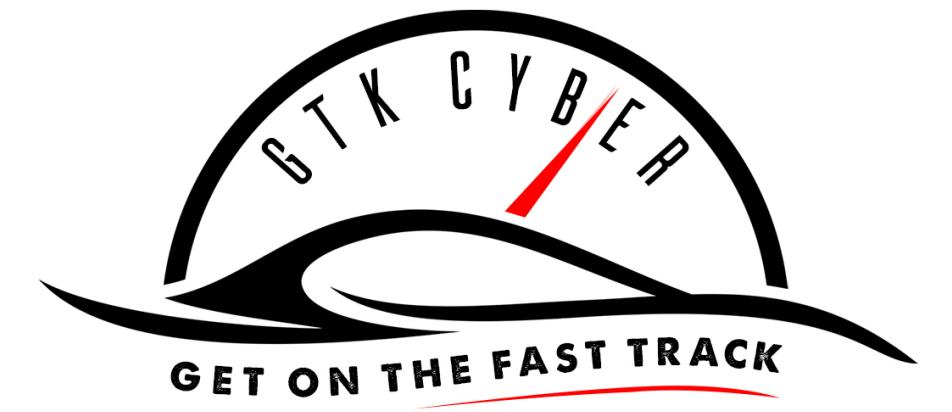
```
combinedSeries = series1.add( series2 )
```



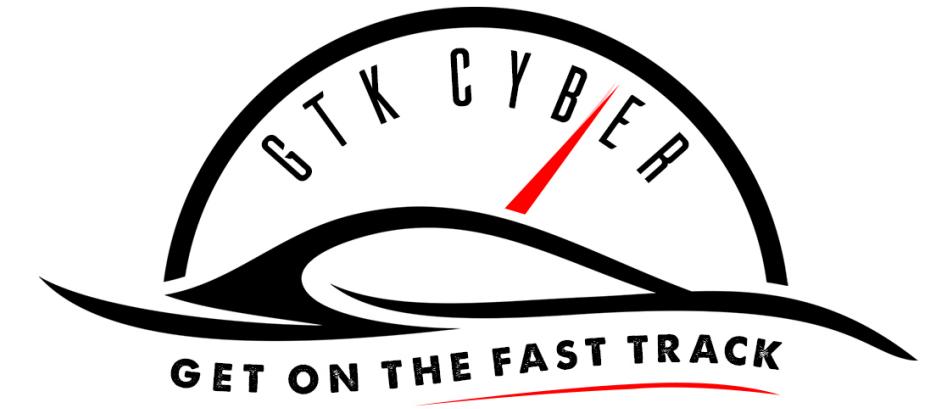
```
def addTwo( n ):  
    return n + 2  
  
odds.apply( addTwo )
```



```
odds.apply( lambda x: x + 1 )
```

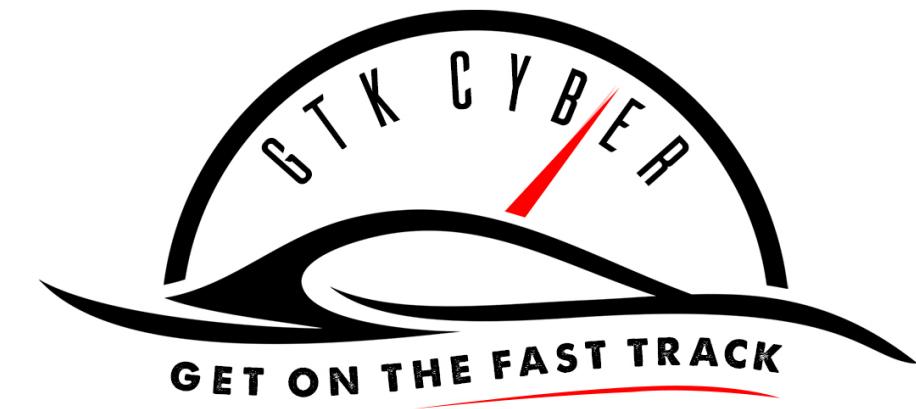


Questions?



In Class Exercise

Please take 20 minutes and complete
Worksheet 1.1: Working with One Dimensional Data

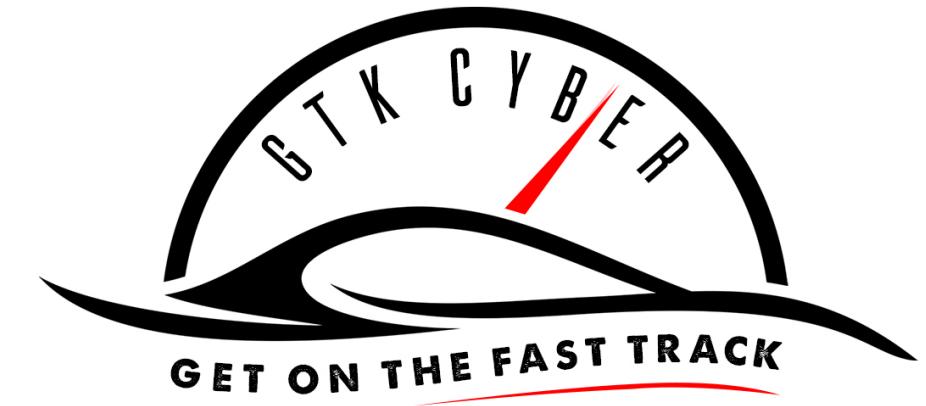


Exercise 1

```
email_series = pd.Series(emails)
filtered_emails =
email_series[email_series.str.contains('.edu')]
```

```
2      mdixon2@cmu.edu
11     jjonesb@arizona.edu
15     eberryf@brandeis.edu
```

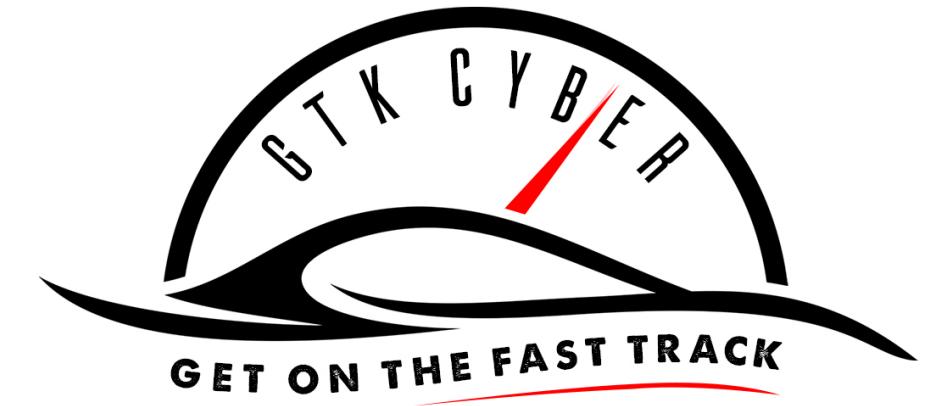
```
accounts = filtered_emails.str.split('@').str[0]
2      mdixon2
11     jjonesb
15     eberryf
dtype: object
```



Exercise 2

```
pounds = pd.Series( weights )
```

```
kilos = pounds.apply( poundsToKilograms )
```

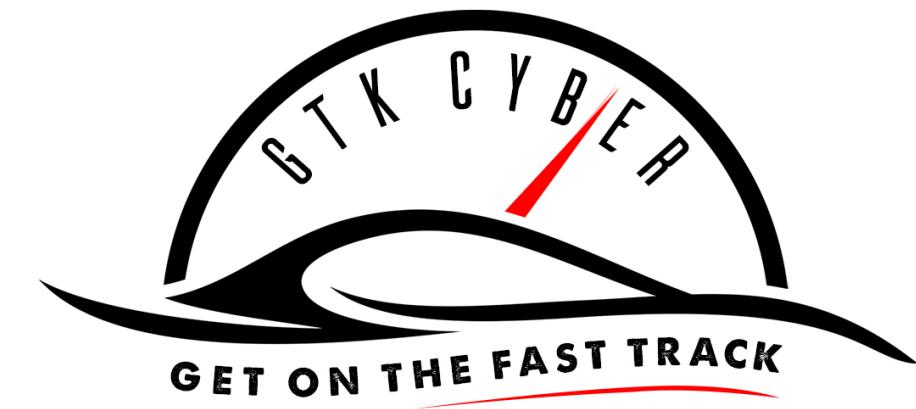


Exercise 3

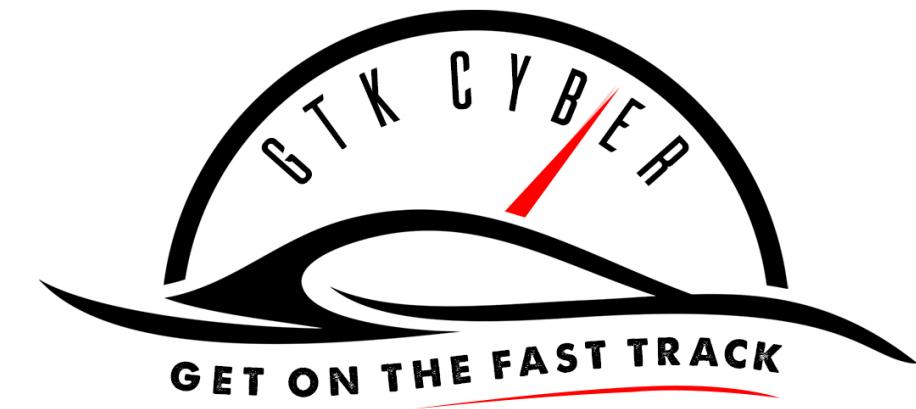
```
IPData = pd.Series( hosts )
privateIPs = IPData[
    IPData.apply(
        lambda x : ip_address(x).is_private
    )
]
```

Explore Your Data





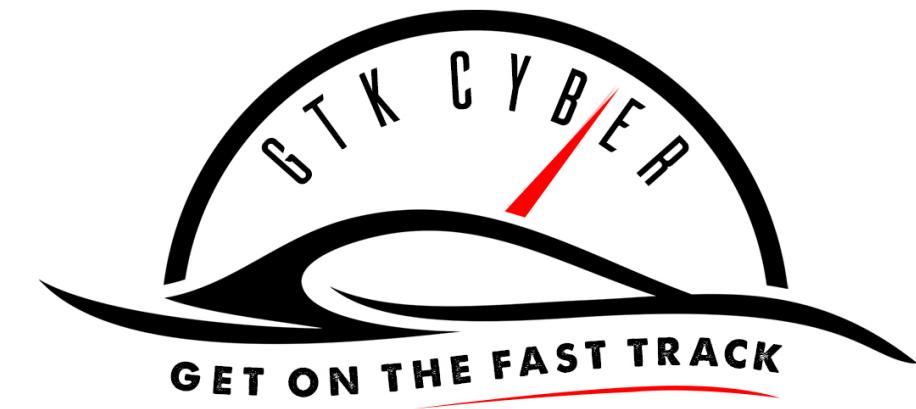
- **Mean:** The mean is the average value of a set of numbers
- **Median:** The median is the middle value of an ordered set of numbers
- **Mode:** The mode is the value from a set which is repeated the most frequently.
- **Range:** Difference between minimum and maximum
- **Standard Deviation:** Measures dispersion in a data set. Close to zero indicates little dispersion.



Intro Stats: IQR

The **interquartile range** is a measure of where the “middle fifty” is in a data set. Where a range is a measure of where the beginning and end are in a set, **an interquartile range is a measure of where the bulk of the values lie.**

$\text{IQR} = \text{data.quantile}(0.75) - \text{data.quantile}(0.25)$

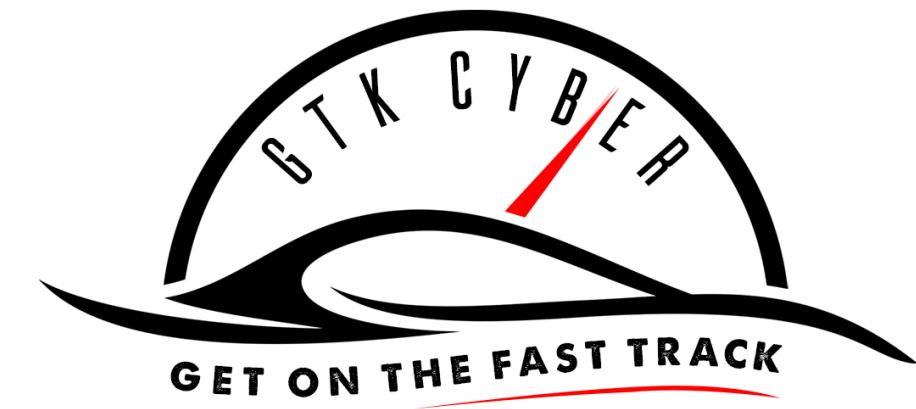


Intro Stats: Variance

The **variance** of a set of values, σ^2 , is the average of the square of the difference of the values in the dataset from the dataset's mean.

$$\sigma^2 = \frac{\sum_{i=1}^n (x_i - \mu)^2}{n}$$

`data.var(axis=0)`



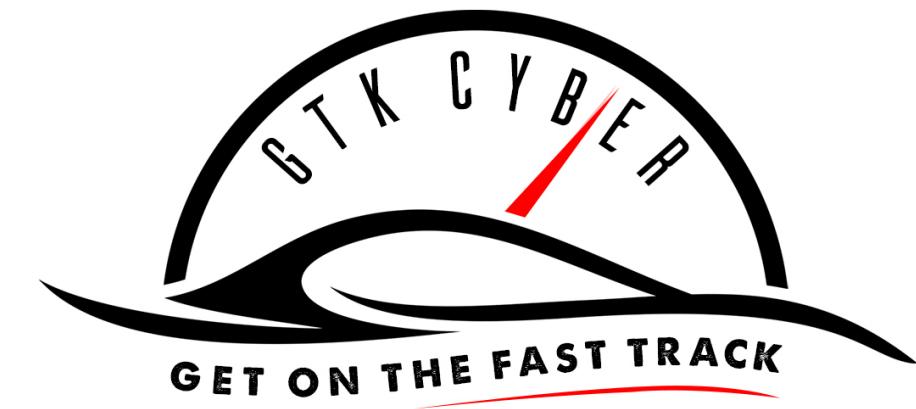
Intro Stats: Std. Deviation

The standard deviation, σ , is the square root of the variance:

We use the **standard deviation** much more regularly than the **variance** because it is on the same scale as the original data:

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \mu)^2}{n}}$$

`data.std(axis=0)`

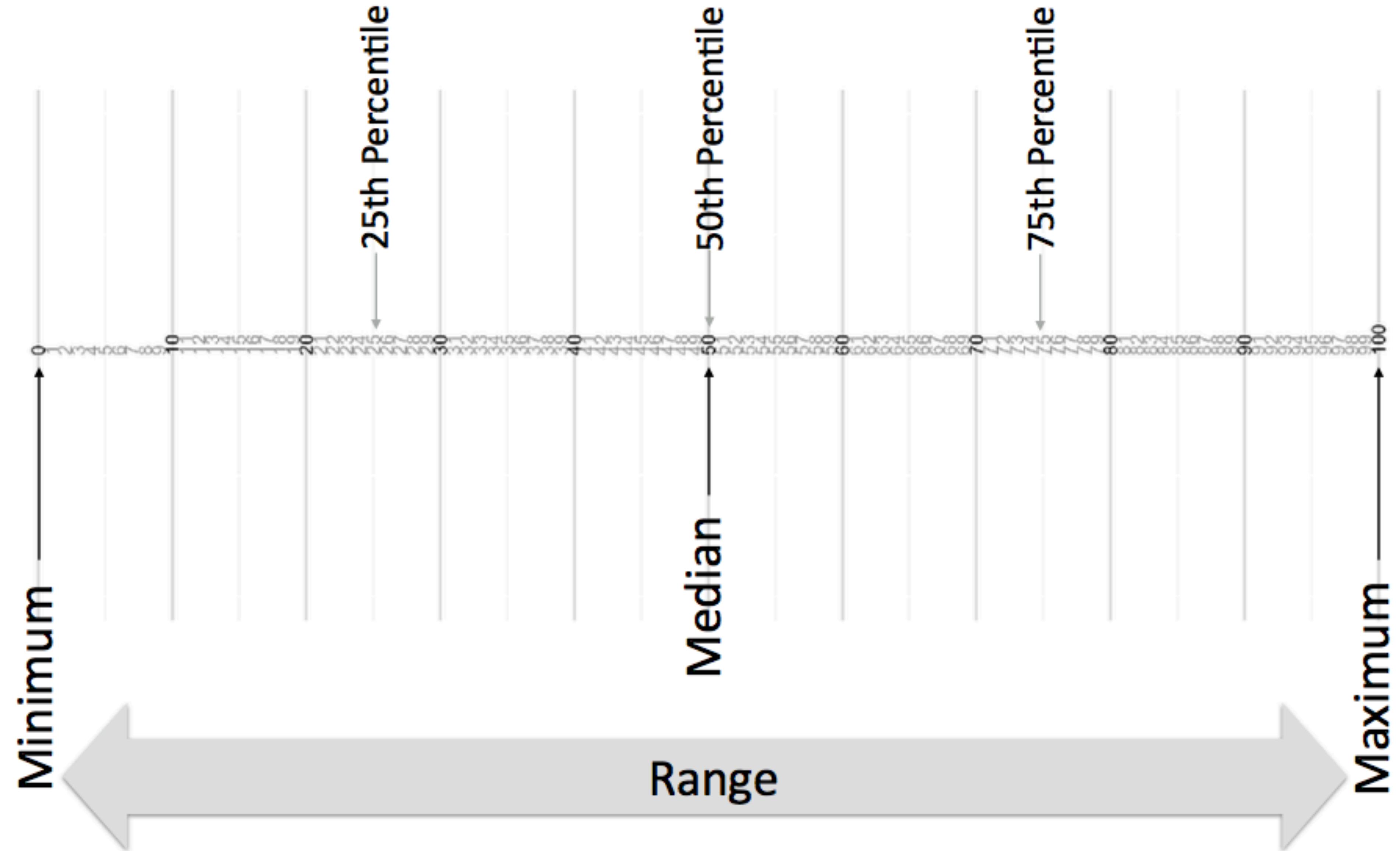
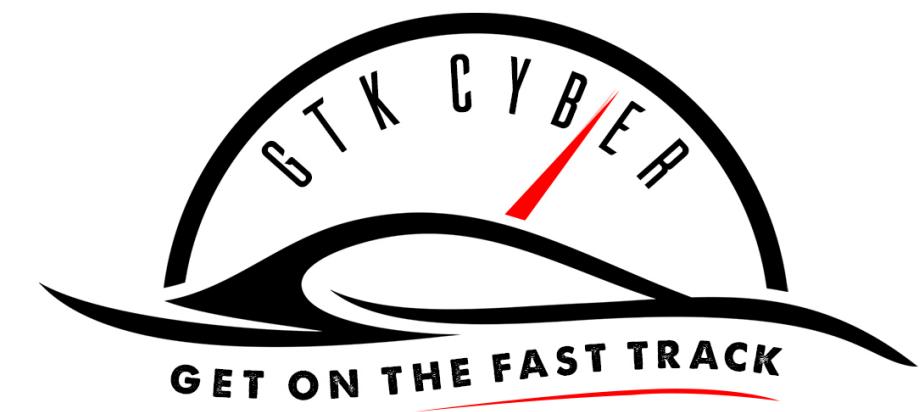


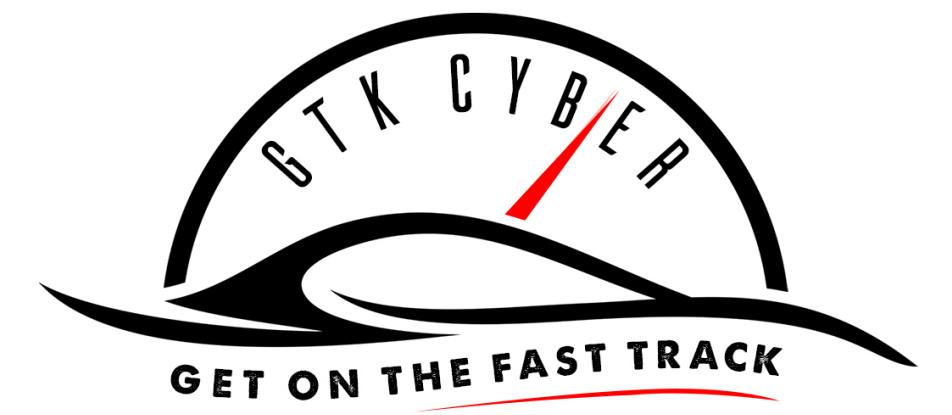
Intro Stats: Covariance

Covariance, like the variance, is a measure of spread, however it also measures how closely two datasets track each other.

- **Covariance** is a squared quantity, so it is not on the same scale as the mean
- **Covariance** of different pairs of variables can have completely different scales

$$Cov(x, y) = \frac{\sum_{i=1}^n (x_i - \mu_x)(y_i - \mu_y)}{n}$$

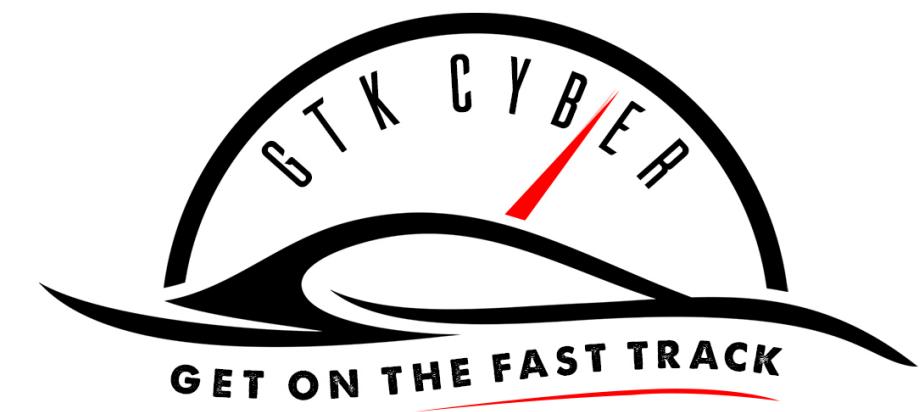




Tukey 5 Number Summary

- **Minimum:** The smallest value in the dataset
- **Lower Quartile:** Smallest 25% of the dataset
- **The Median:** The middle value of the dataset
- **Upper Quartile:** The largest 25% of the dataset
- **Maximum:** The largest value in the dataset





Series.abs()

Absolute Value of the Series

Series.count()

Returns number of non-empty values in the series

Series.max()

Returns maximum value in the Series

Series.mean()

Returns the mean of a Series

Series.median()

Returns the median of a Series

Series.min()

Returns the minimum value in a Series

Series.mode()

Take a guess..

Series.quantile([q])

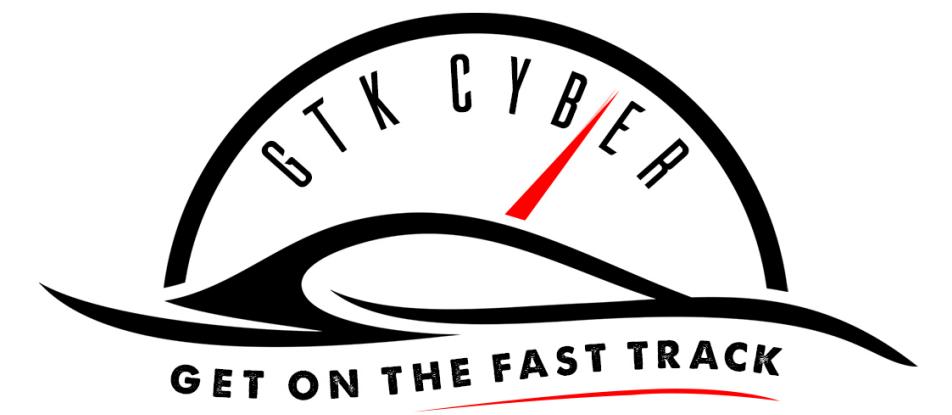
Returns the quantiles of a Series

Series.sum

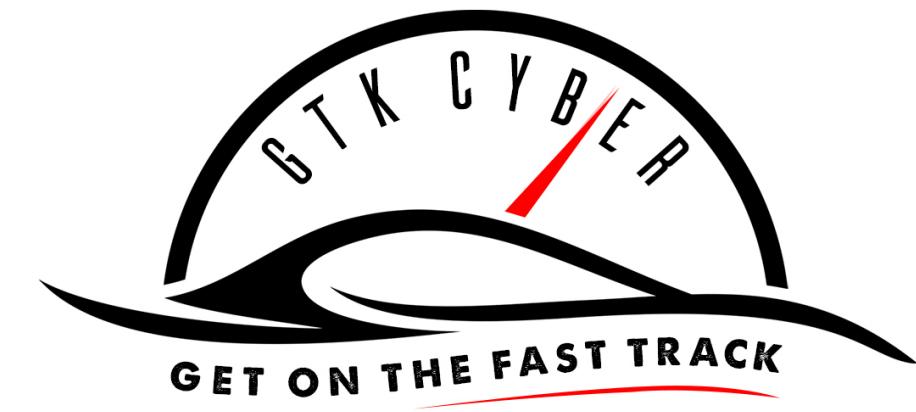
Returns the sum of a series

Series.std

Returns the standard deviation of a Series

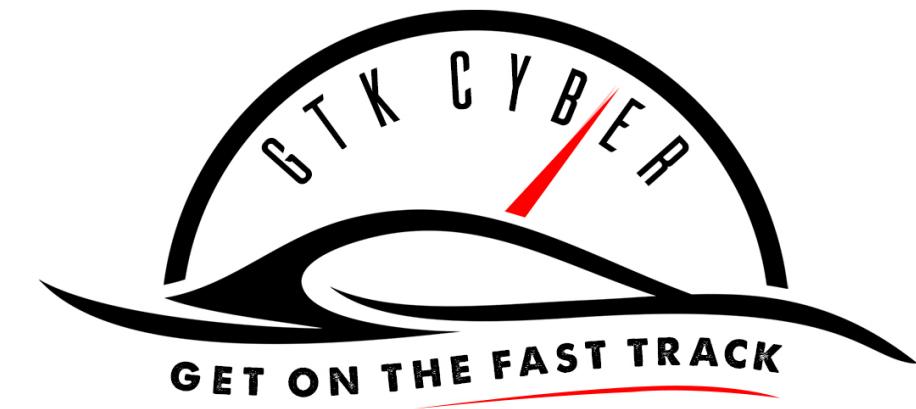


Series.describe()



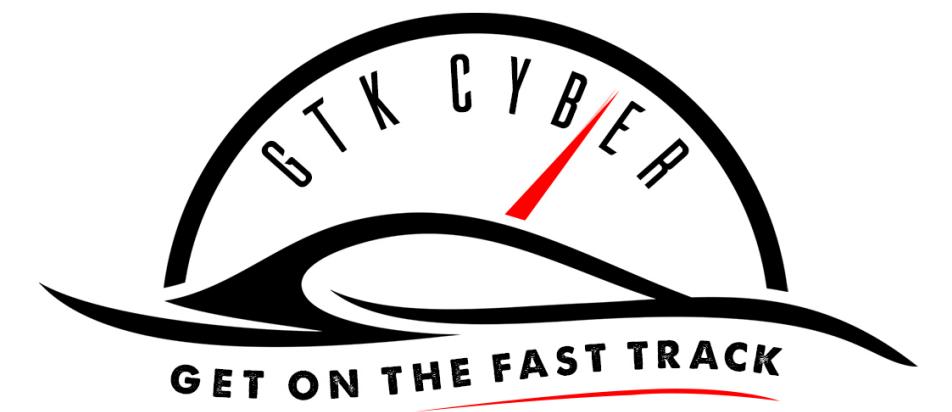
Series.describe()

```
>>> random_numbers.describe()
count      50.000000
mean       50.620000
std        30.102471
min        1.000000
25%       25.500000
50%       54.000000
75%       73.000000
max        99.000000
dtype: float64
```



```
names = pd.Series(  
    [ 'Jim', 'Bob',  
    'Bob', 'Steve', 'Jim', 'Jane', 'Steph', 'Emma', 'Rachel  
    ' ] )
```

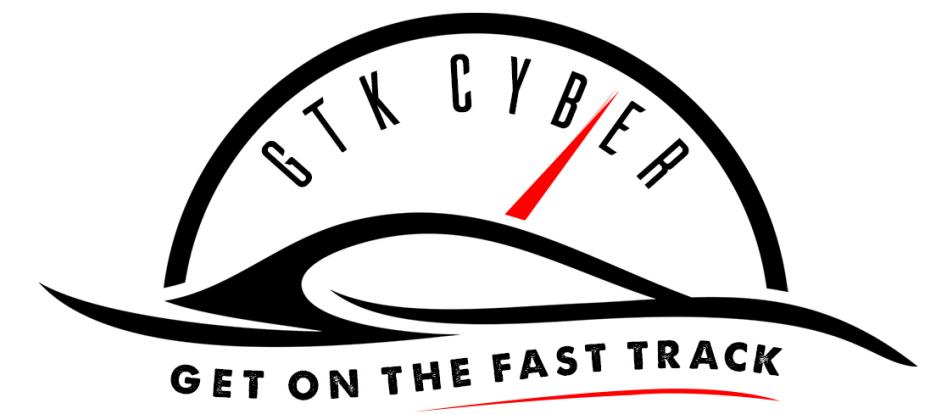
names.**describe()**



```
names = pd.Series(  
    [ 'Jim', 'Bob',  
    'Bob', 'Steve', 'Jim', 'Jane', 'Steph', 'Emma', 'Rachel  
    ' ] )
```

names.**describe()**

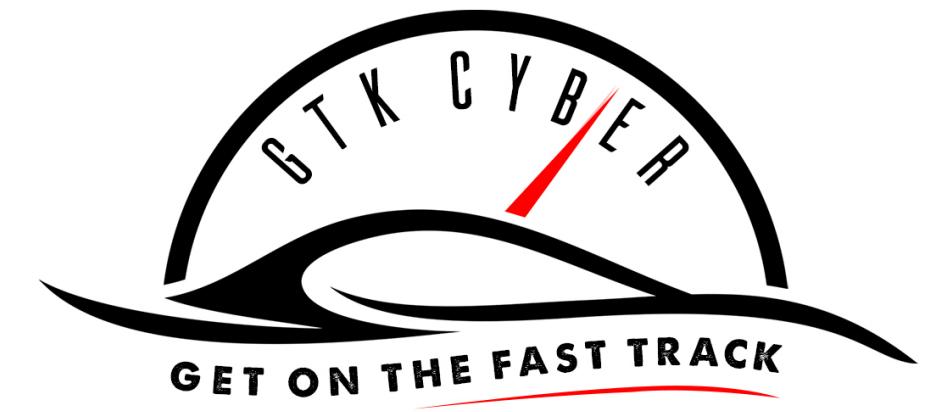
```
count          9  
unique         7  
top           Jim  
freq           2  
dtype: object
```



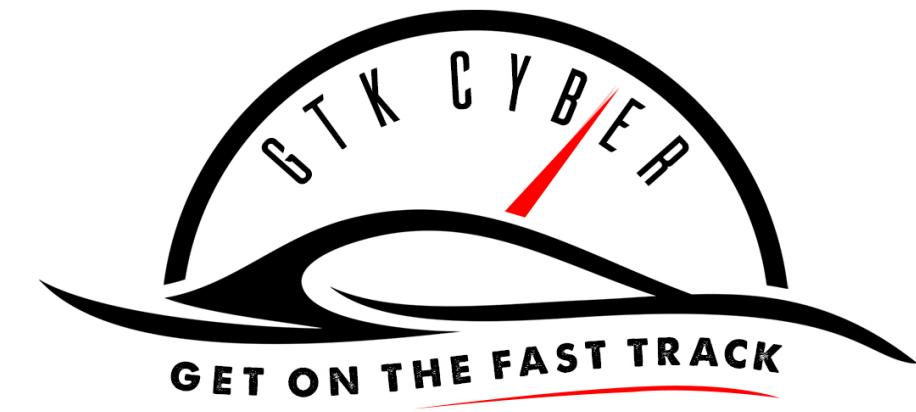
Finding Unique Values



```
unique_nums = []
for key, value in random_numbers.iteritems():
    if value not in unique_nums:
        unique_nums.append(value)
```



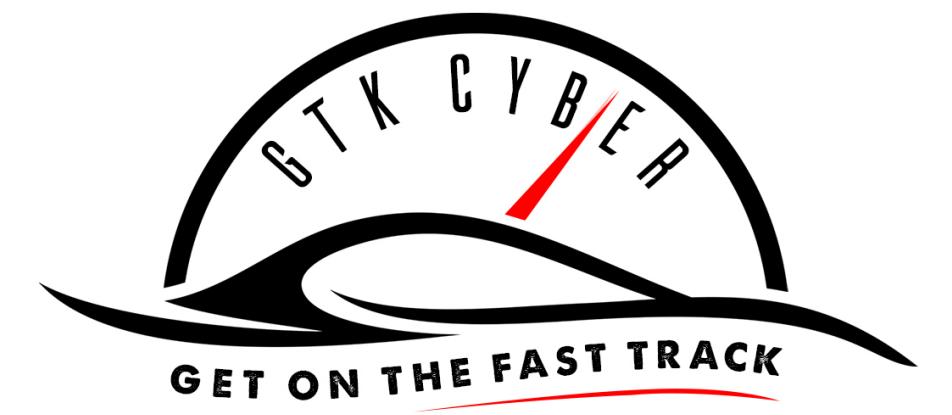
NO!!!



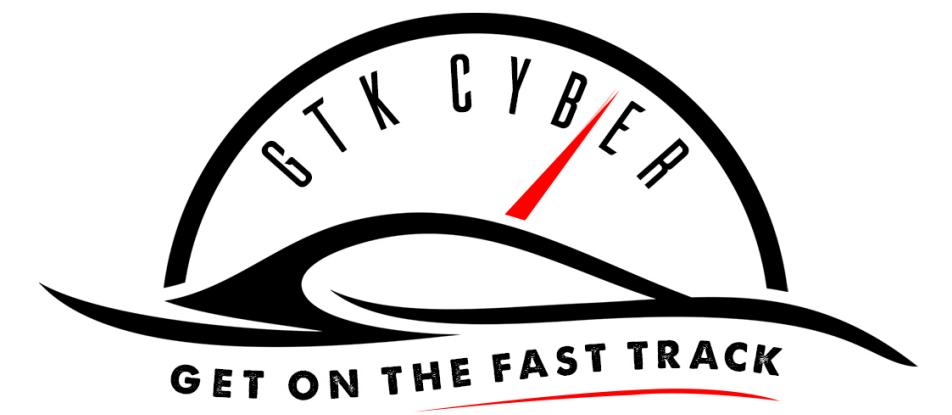
series.drop_duplicates()
series.unique()



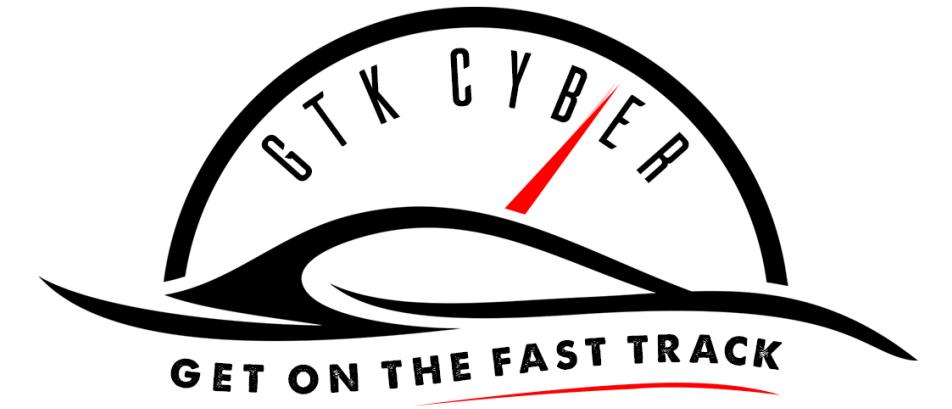
Counting Unique Occurrences



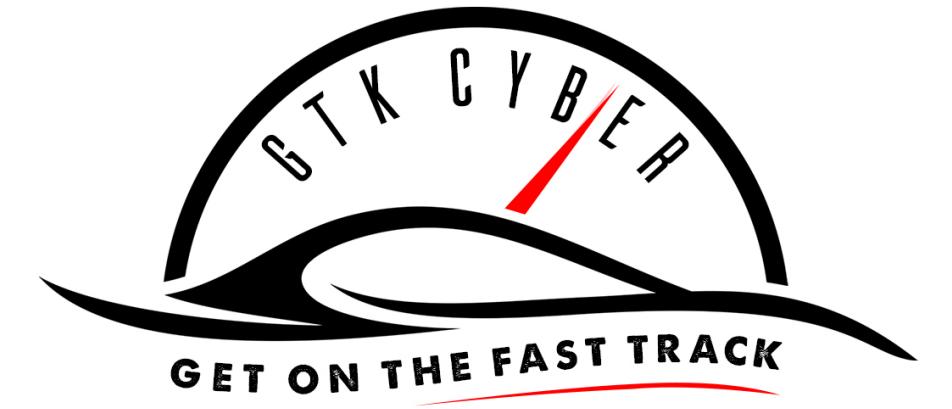
`series.value_counts()`



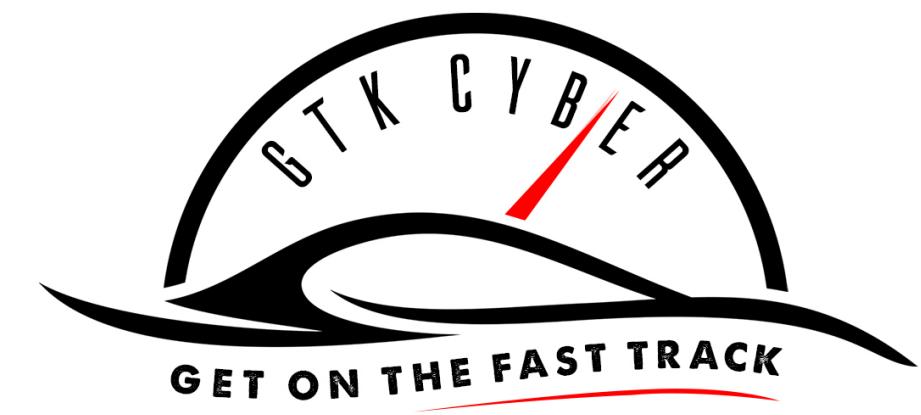
`series.value_counts(bins=5)`



```
series.value_counts(bins=5,  
normalize=True)
```

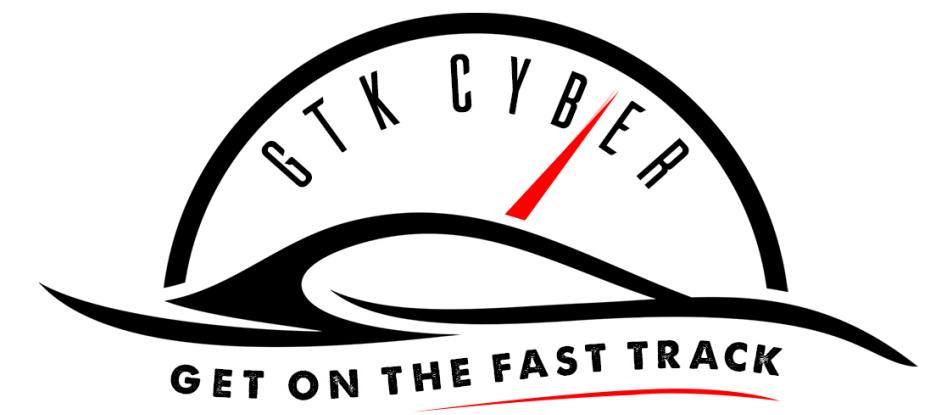


```
series.value_counts(bins=5,  
normalize=True,  
dropna=False)
```

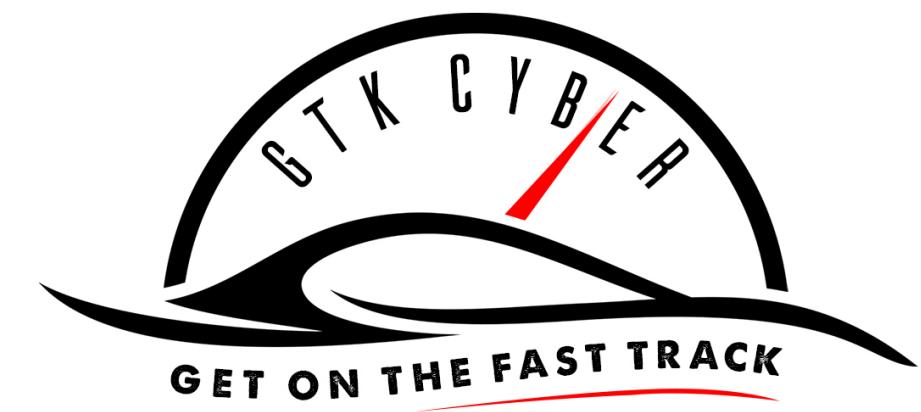


```
In [7]: area_codes2.value_counts()
```

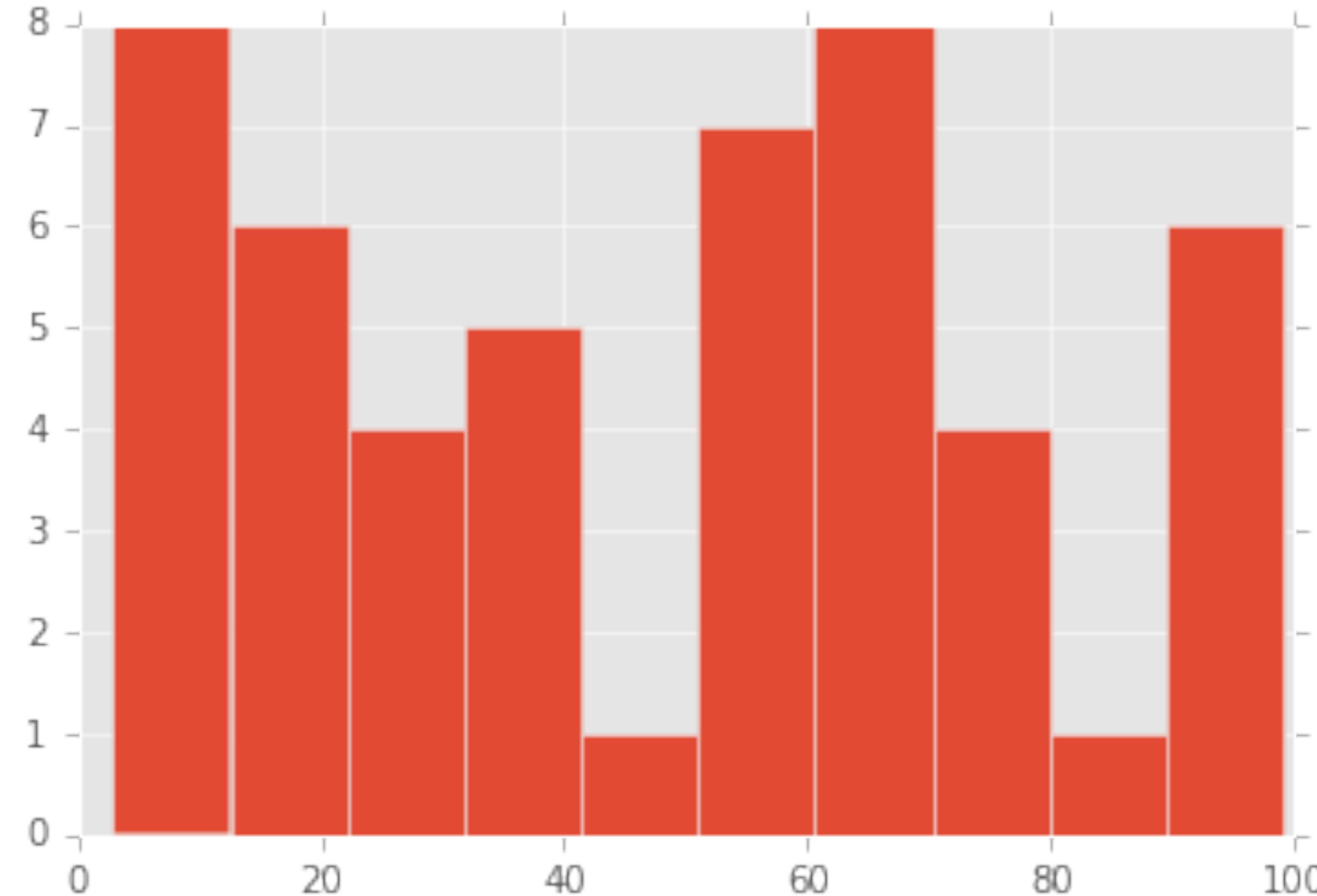
```
Out[7]: 833      7  
        811      5  
        822      5  
        899      3  
        844      3  
        855      2  
dtype: int64
```



series.hist()

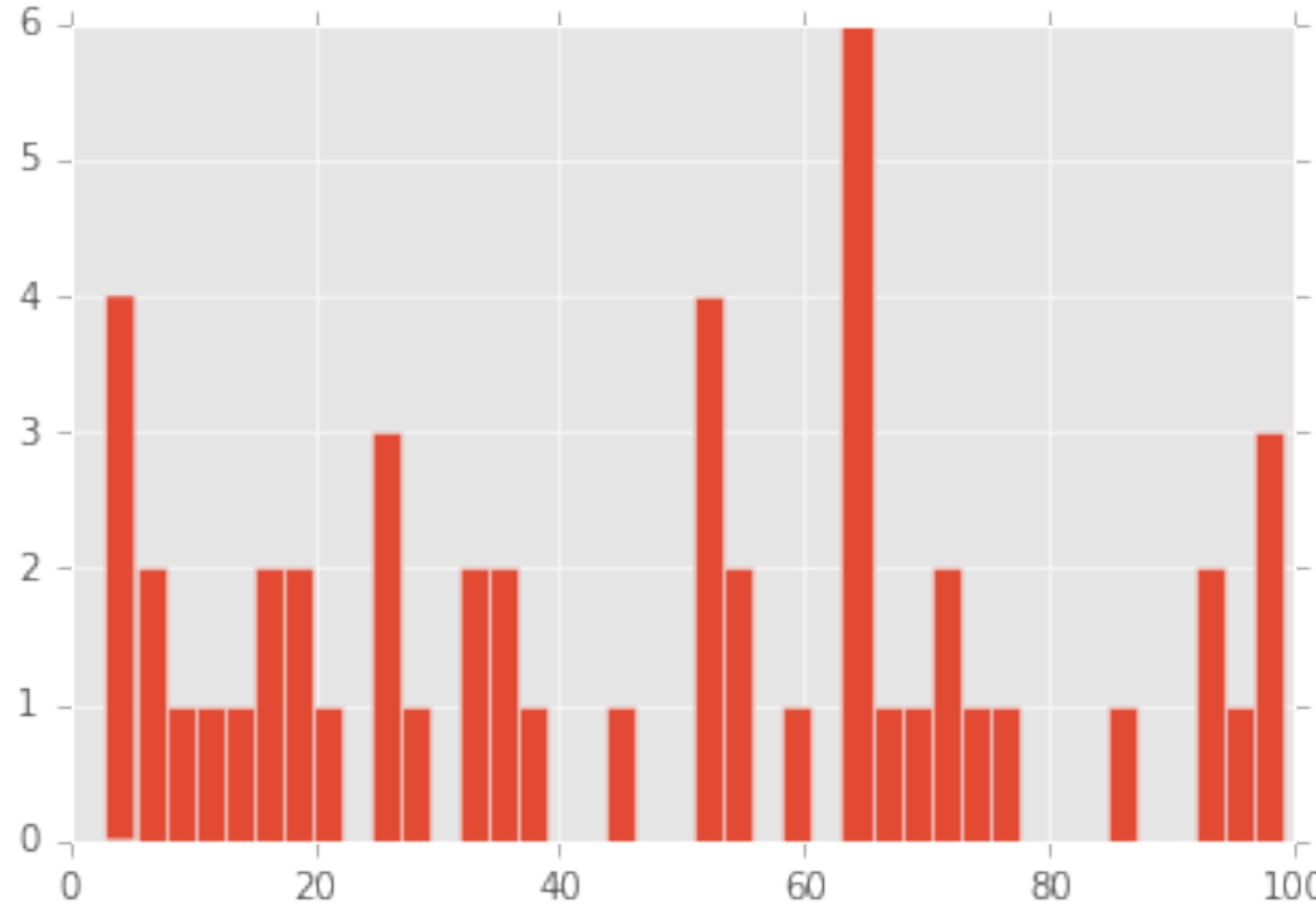


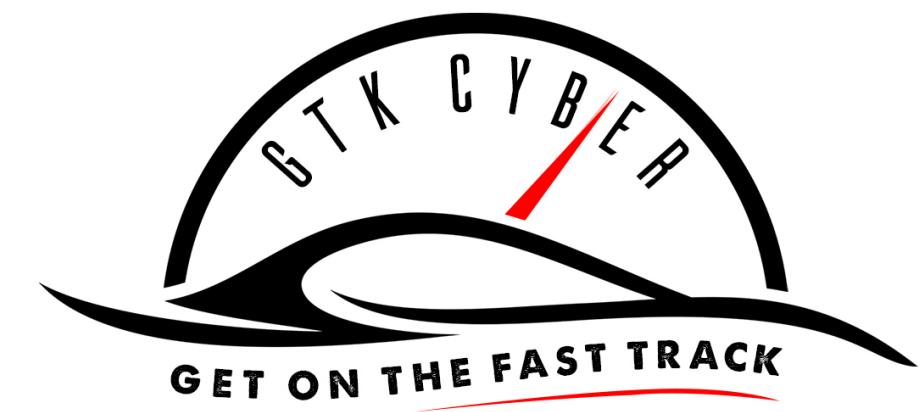
series.hist()



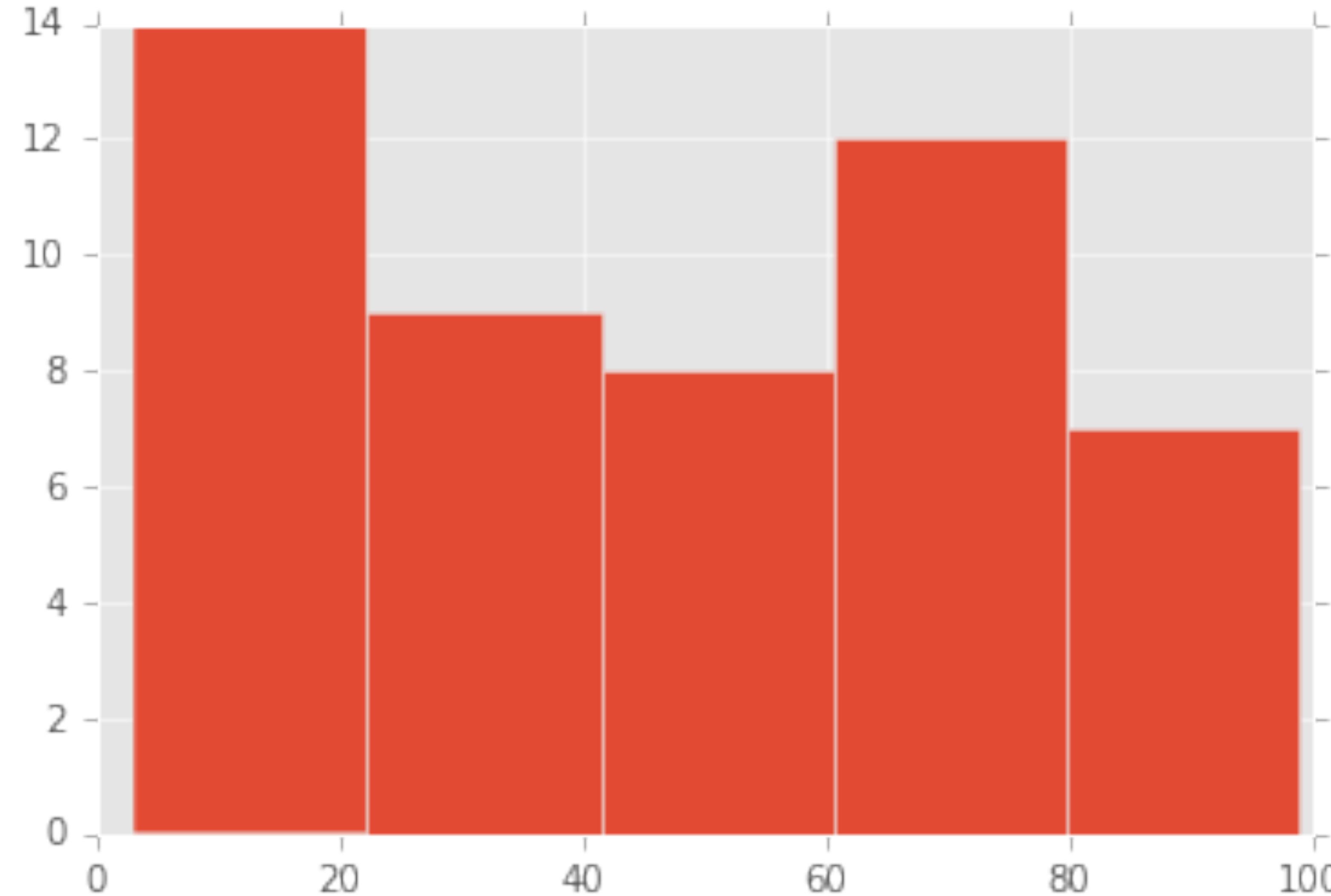


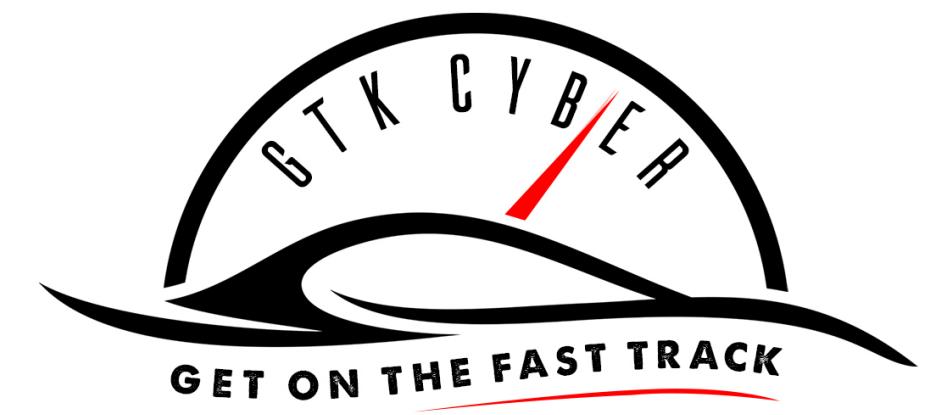
series.hist(bins=40)



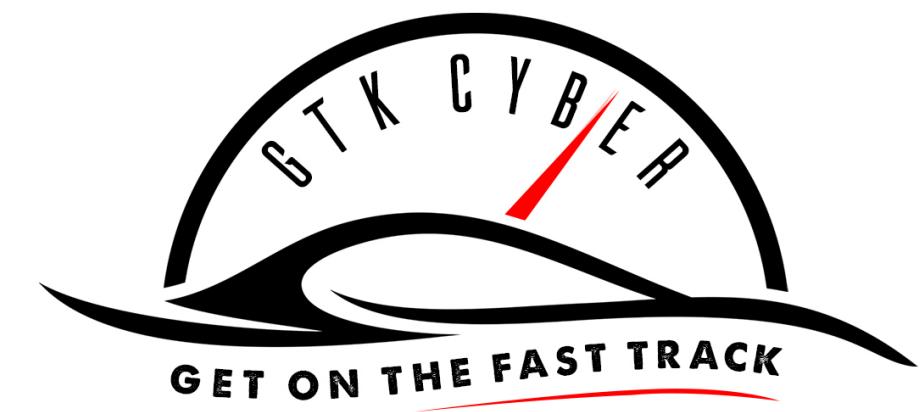


series.hist(bins=5)

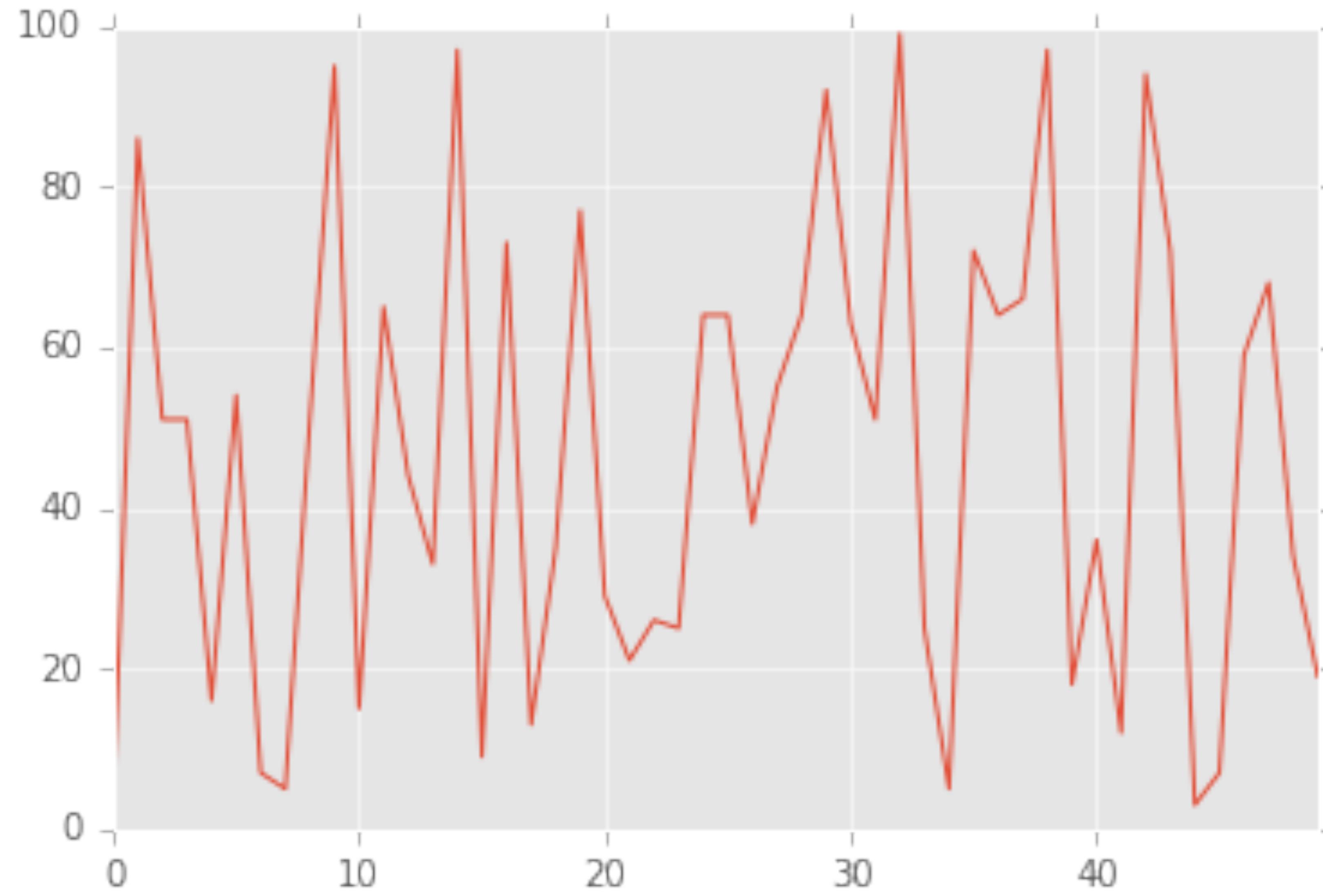


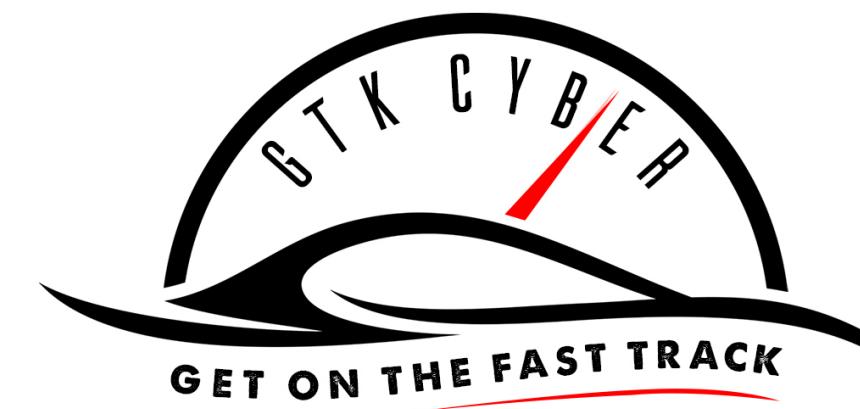


series.plot()

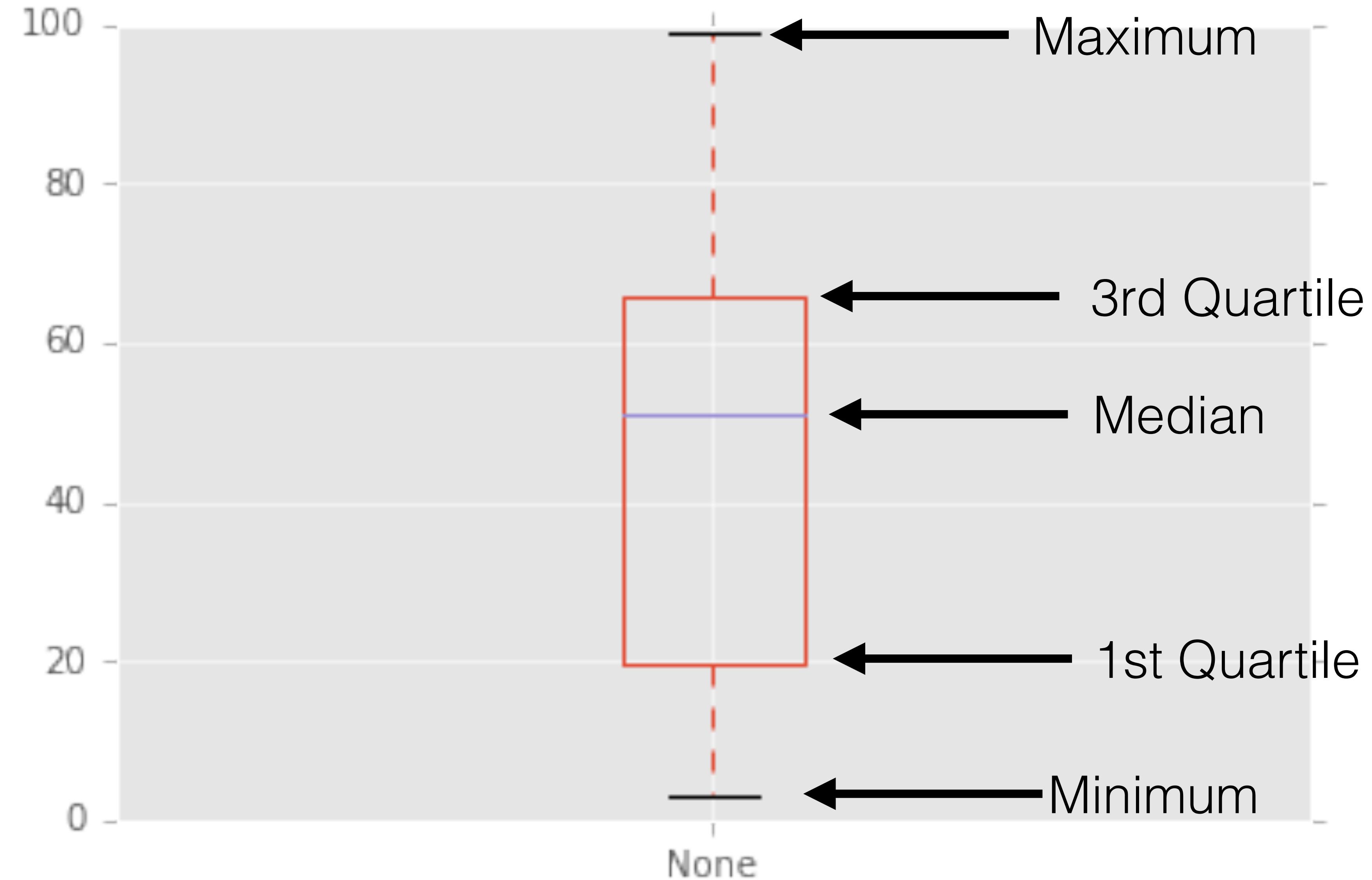


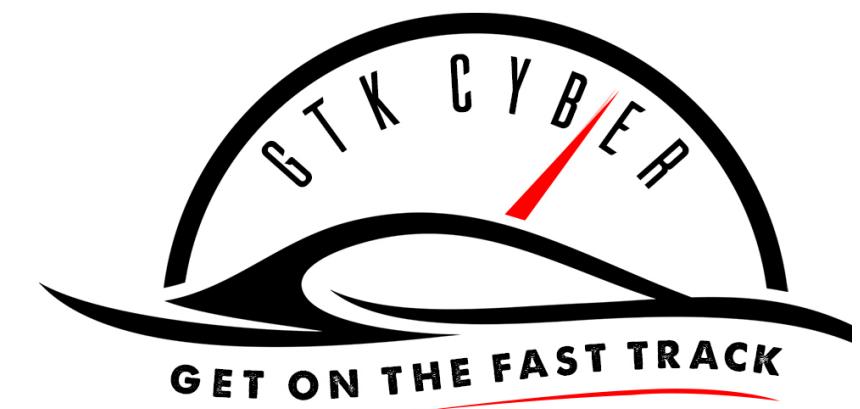
series.plot()





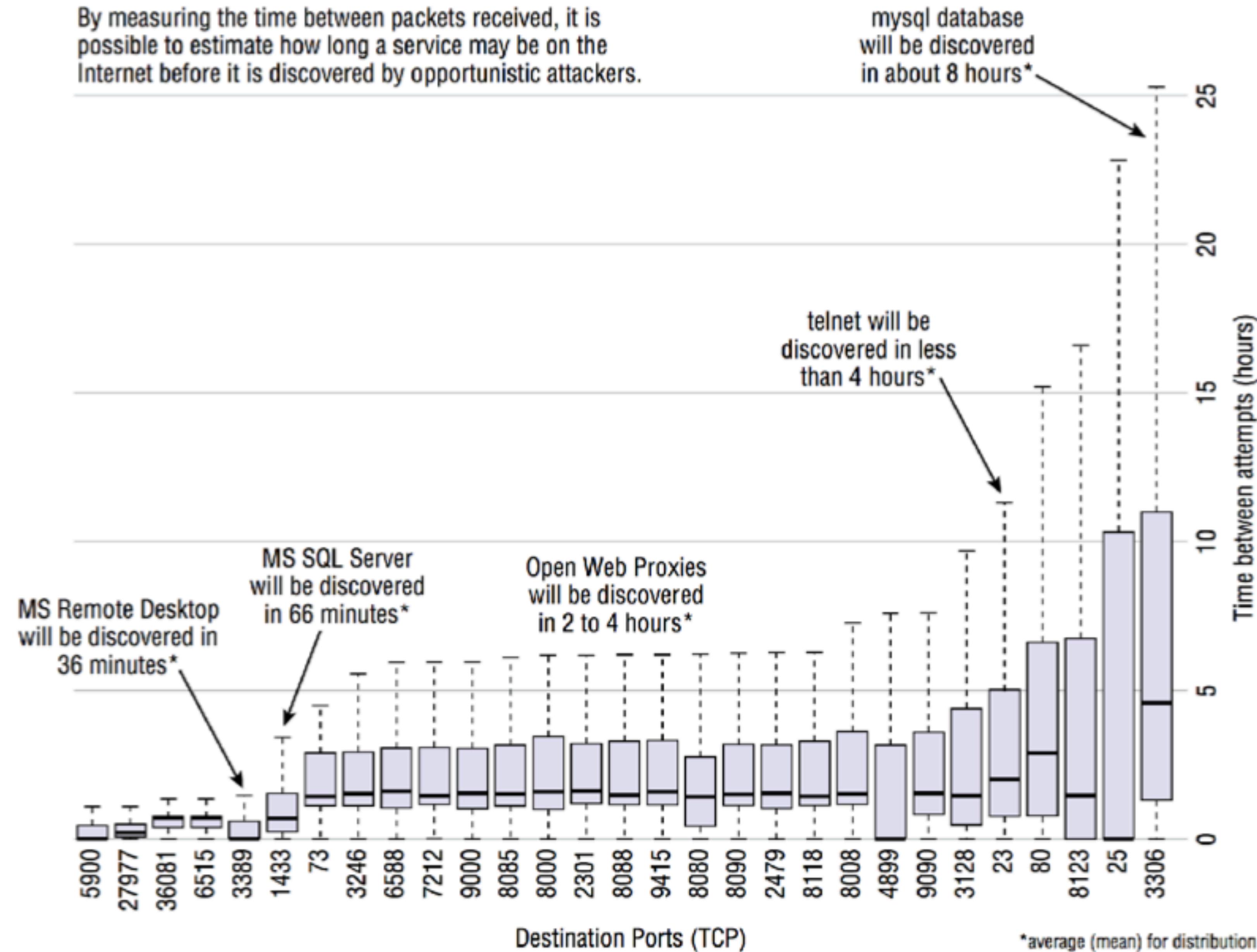
series.plot(kind='box')

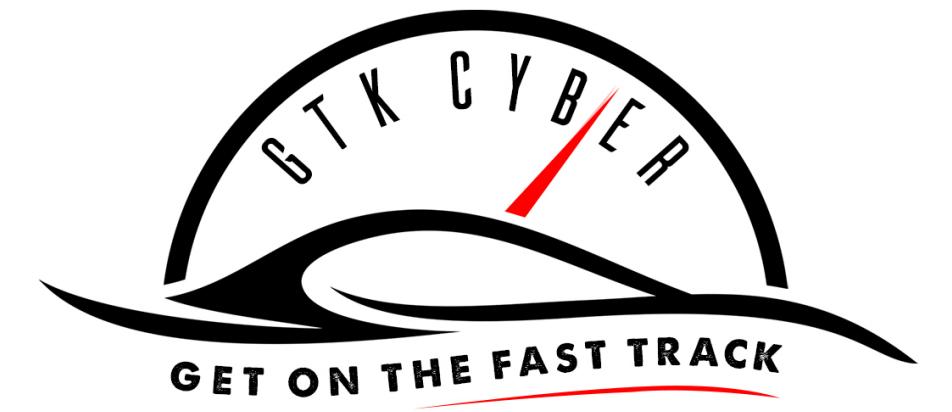




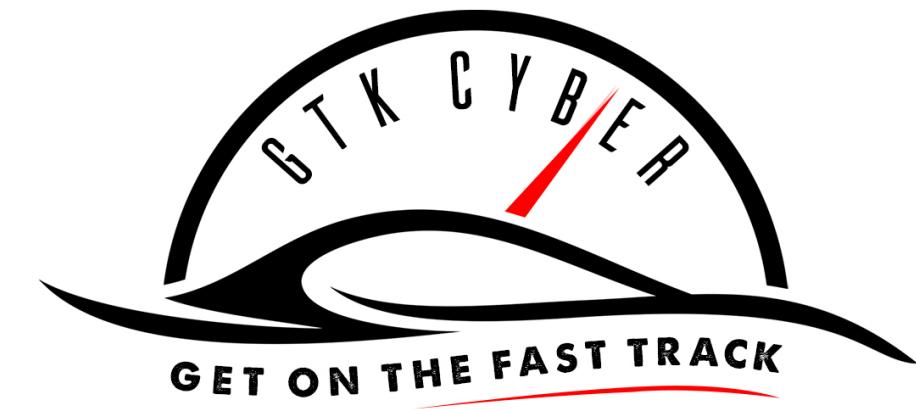
How long will a service go undiscovered by opportunistic attackers?

By measuring the time between packets received, it is possible to estimate how long a service may be on the Internet before it is discovered by opportunistic attackers.





Correlations between Datasets

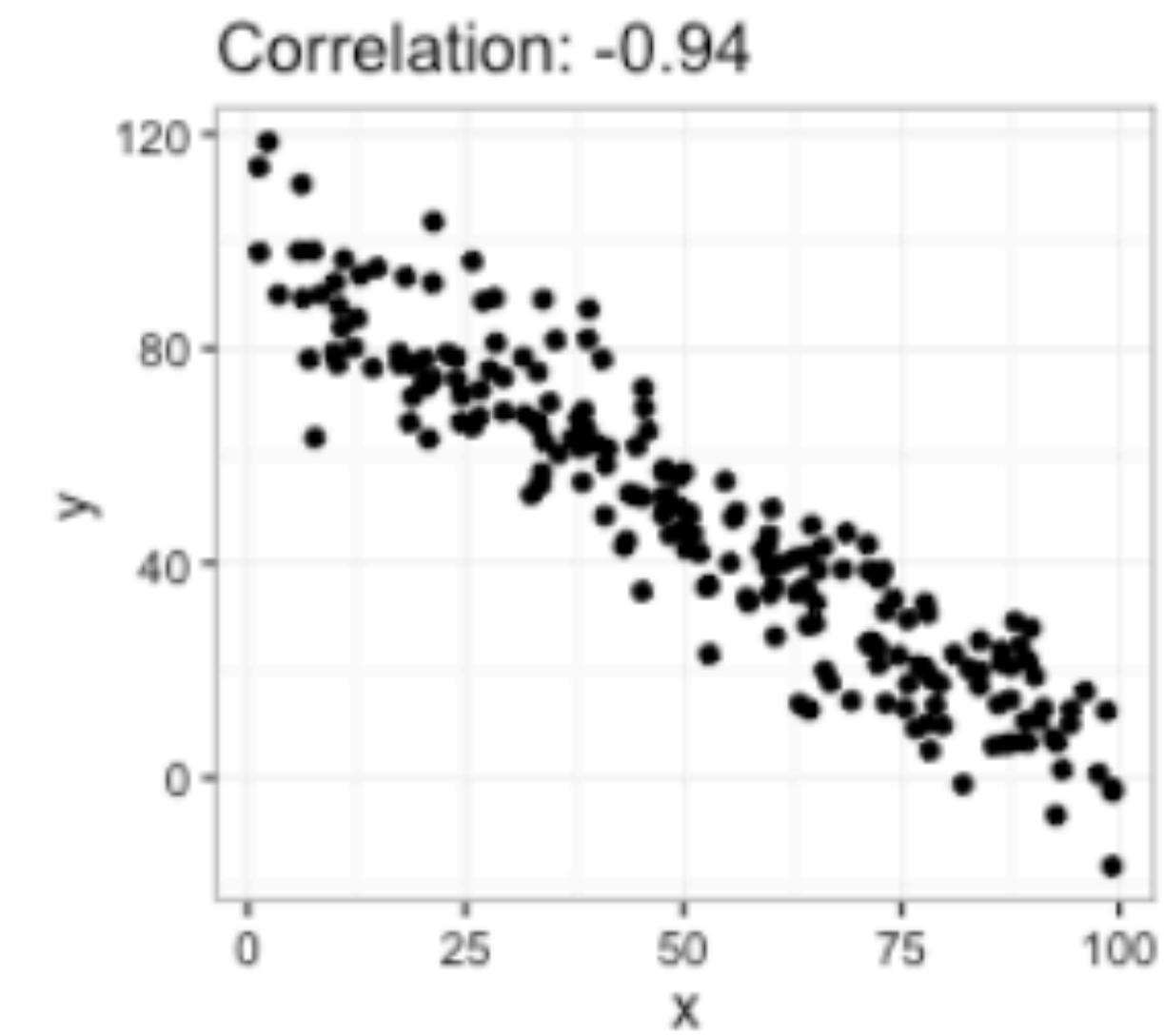
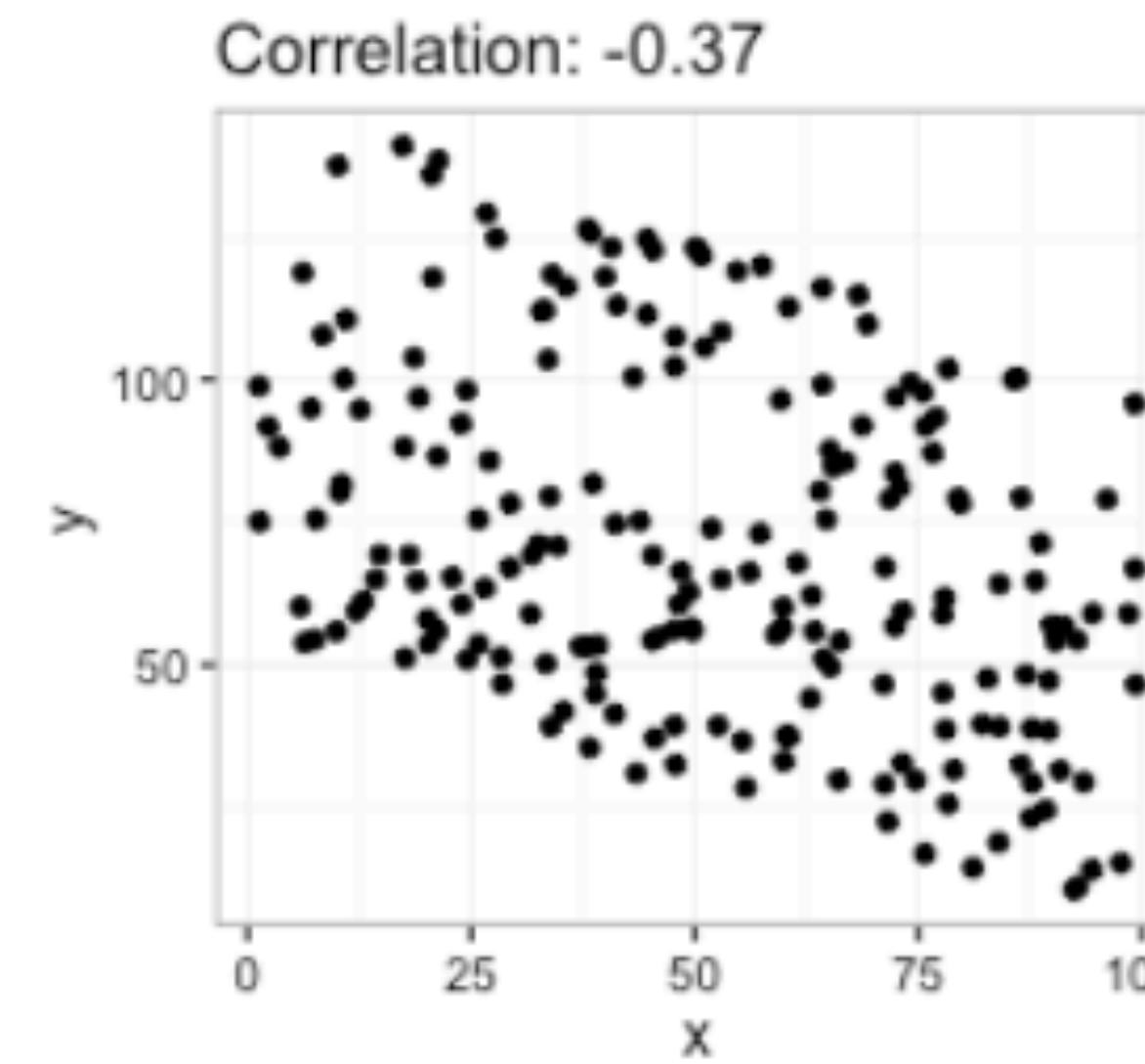
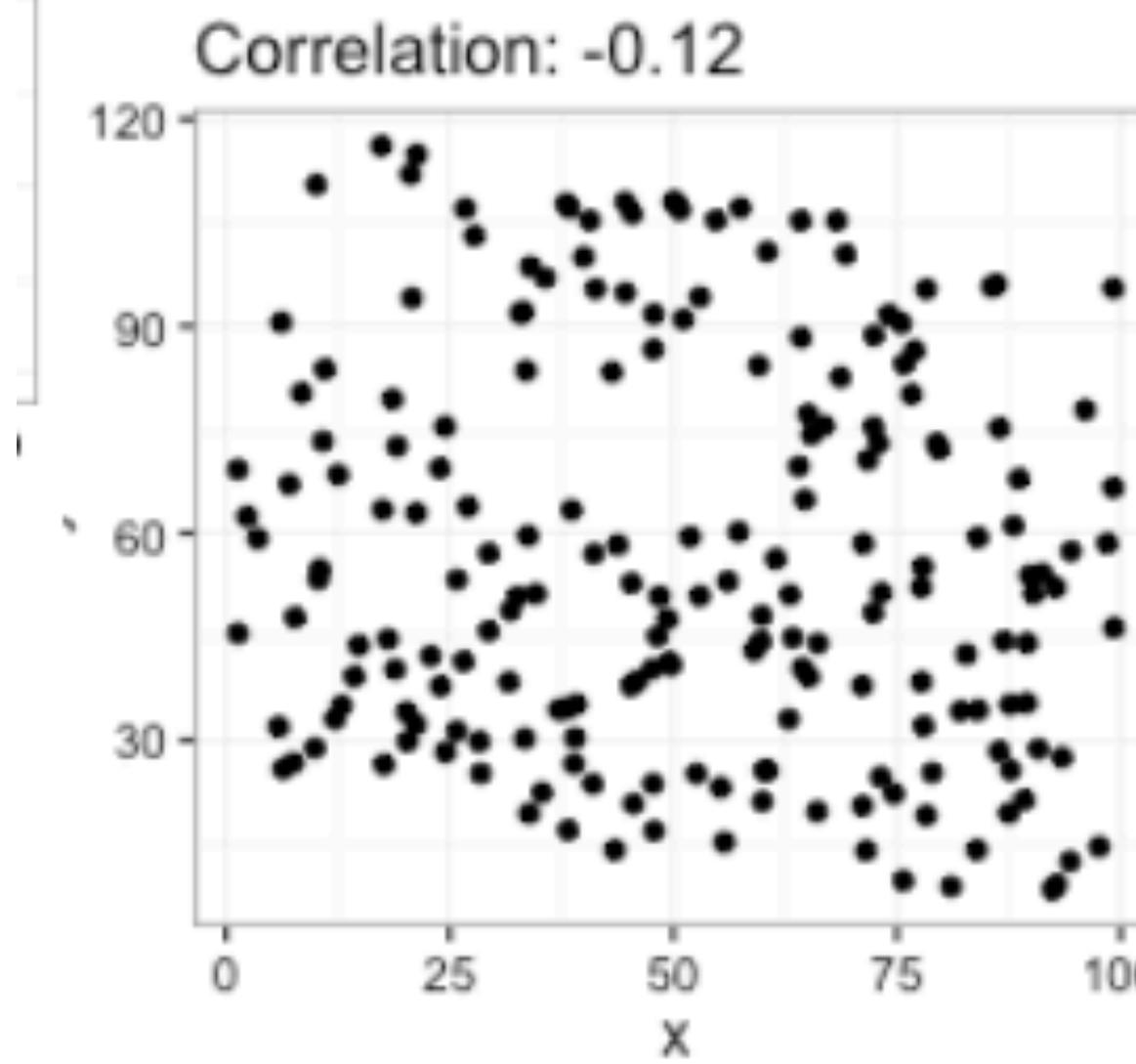
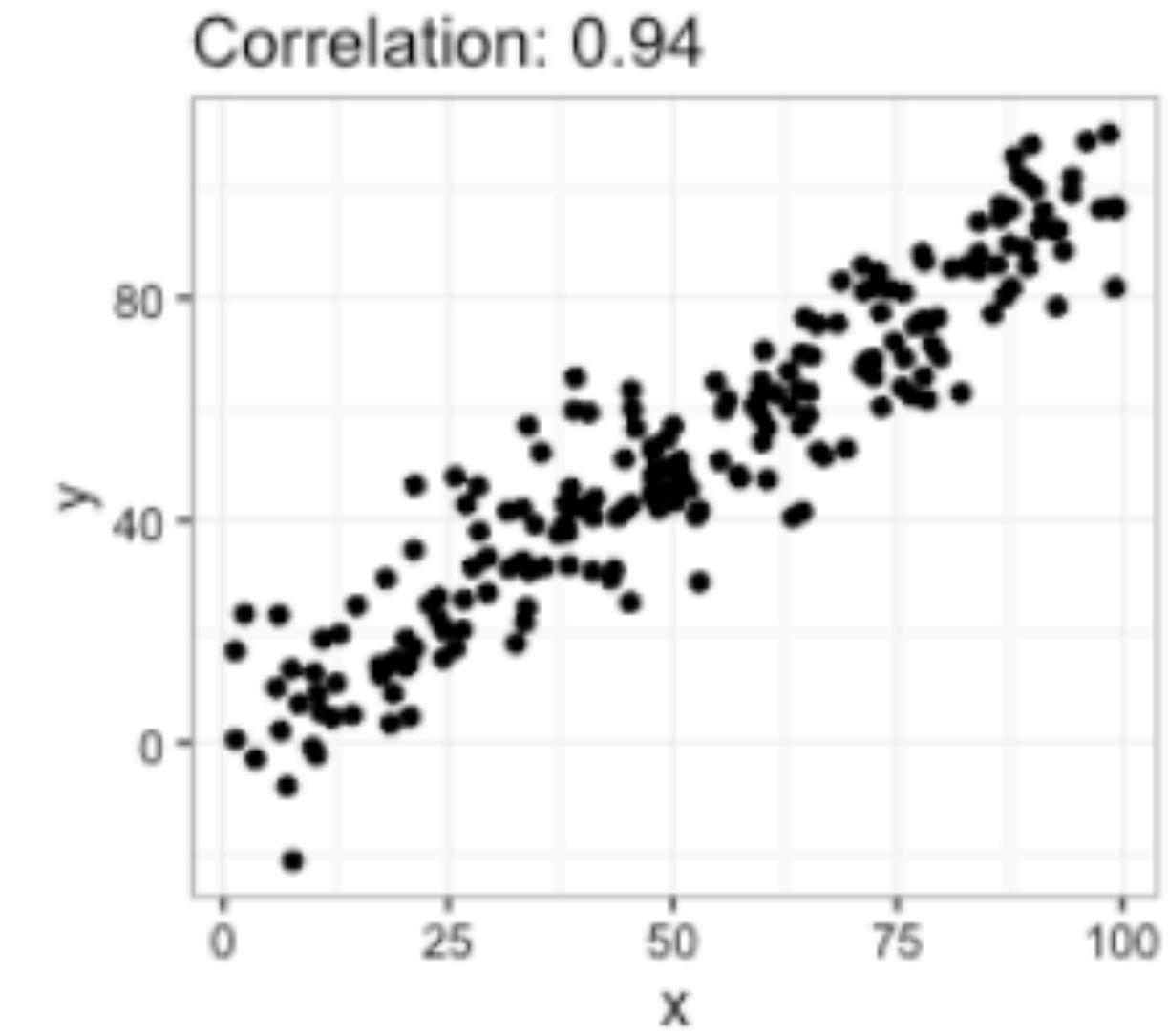
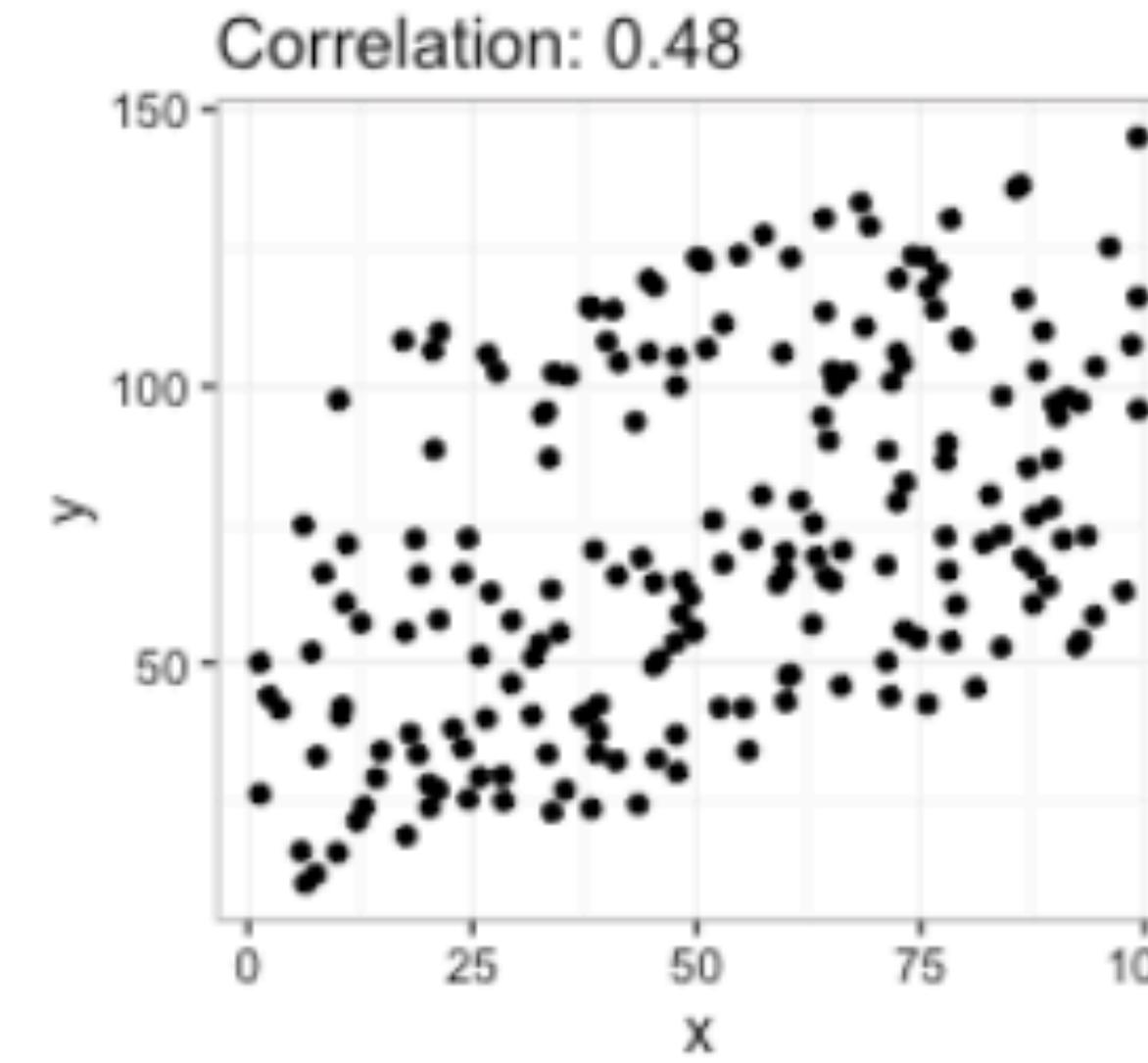
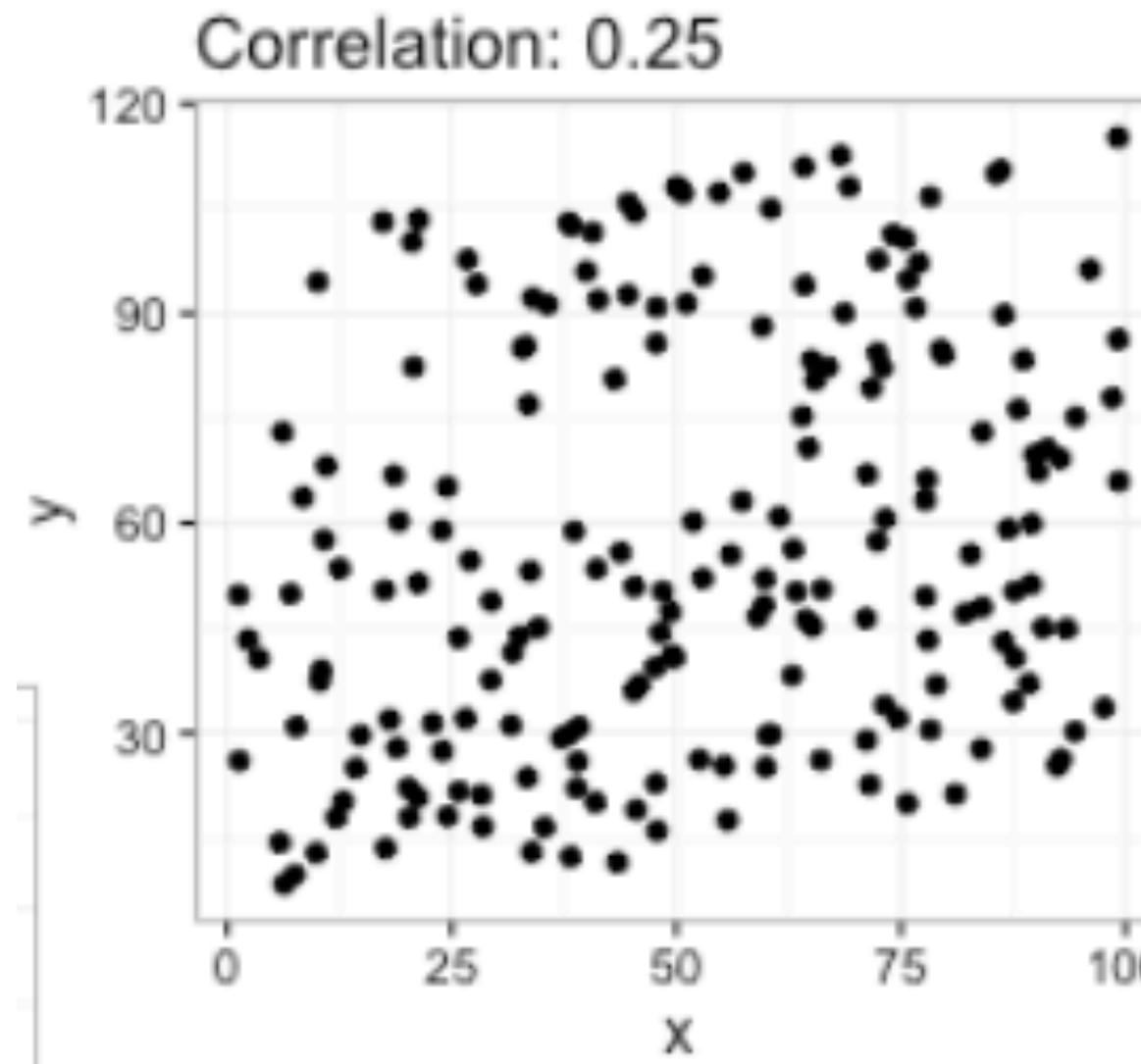


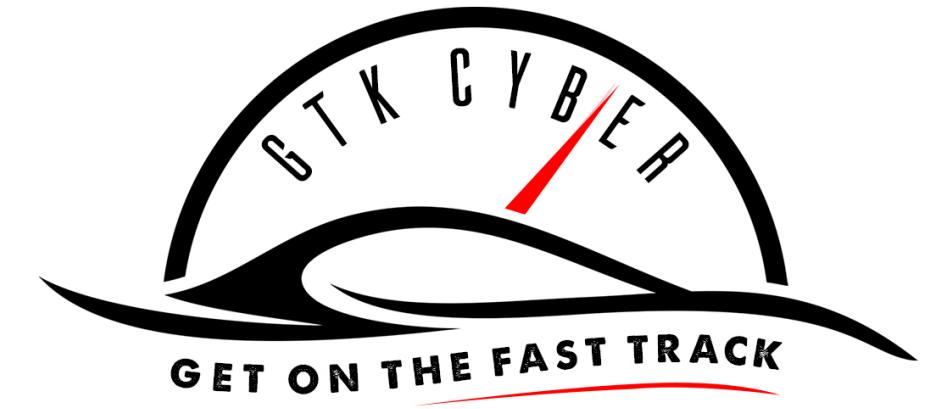
Correlations

- Measurement of the relationship between **two continuous variables**
- Output is between -1 and 1, with 1 being perfect correlation, -1 is perfect negative correlation, 0 is no correlation.
- Correlation measurement is referred to as r .



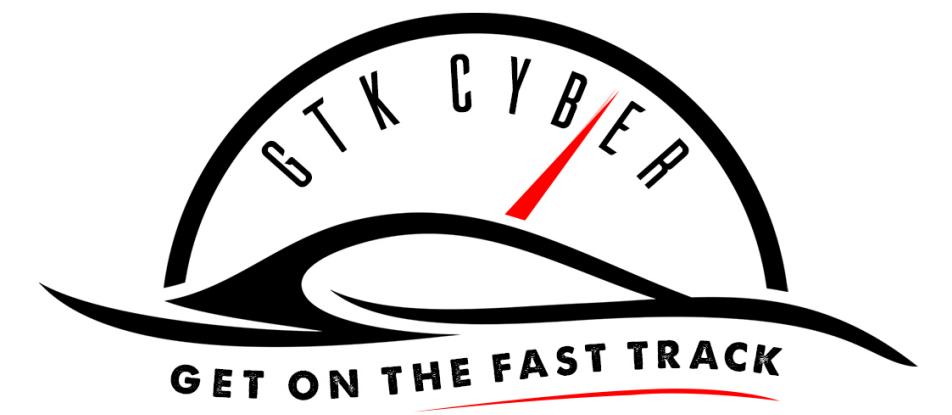
Correlations



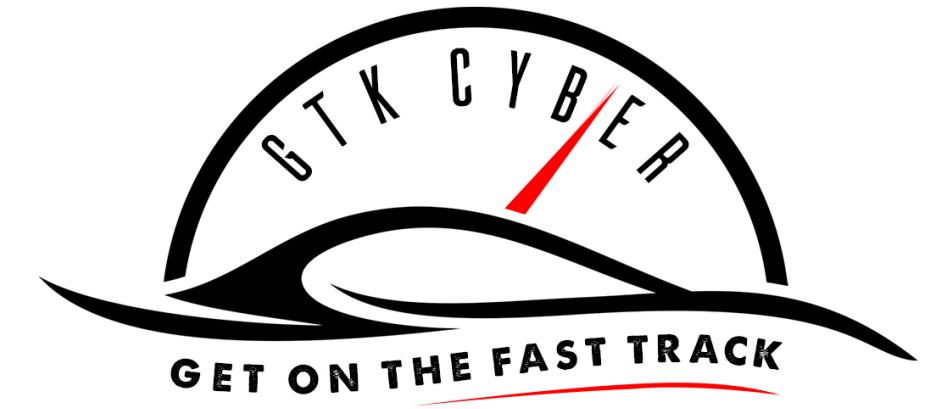


Correlations

```
series1.corr( series2 )
```



Questions?



In Class Exercise

Please take 30 minutes and complete
Worksheet 1.2: Exploring One Dimensional Data