```
pd.read_csv('url', sep='\t', header=None,
 names=['col1','col2'],index_col='col1')
pd.read_excel("path", sheet_name=0/"data")
If header on line 4, header=3 if no blank lines
read_excel() same param as read_csv

df.describe() - numeric cols stats
df.shape - [rows,cols]
df[["col1","col2"]]
df,iloc[-1,0] - last row, 1st col
df.iloc[[-1,3,5], [0,1]]] - last, 4th and 6th row
                            - 1st and 2nd col
df,iloc[:10, 4:] - 1st 10 rows, 5th col +
df['col_1'].value_counts() - unique entries
df.mean(axis=1) - row means
df.idxmin() - row index of lowest
Value in each column
len(df['col1'])
df = df.drop([row#,row#])
df = df.drop(["col1","col2"], axis=1)

df.groupby(by = 'col1').groups
df.groupby(by = 'col1').get_group(value)
df.groupby(by = 'col1').mean(numeric_only=True)
df.loc[:, ['numeric1','numeric2']].groupby(by =
'numeric1').aggregate(['mean','sum','count'])
df.groupby(by='col1').agg(
            {'col2':['max'],
            'col3':['sum']})

df.sort_values(by='col1', ascending=False)
df.to_csv('data.csv')
^ = XOR, & = AND, | = OR
df[(df['col1'] > 5) & (df['col2'] < 15)] filter
df.query("`col1` > 5 & `col2` == 'T'"))
Used ` (backtick) for cols
df.dropna() drop rows w/  NaN
df.fillna(value) replace all Nan w/ value
[] - series, [[]] - dataframe
df.columns = ['new_col1','new_col2']
df = df.rename(columns = {'col_1':'new_col1',
                          'col_2':'new_col2})
melt - wide to long, pivot - long to wide
df.melt(id_vars = ['col_1'],
        value_vars = ['col_2','col_3'],
        var_name = "col2+3",
        value_name = "col2+3_vals")
df.pivot(index = "col_1",
         columns = "col2+3",
         values = "col2+3_vals")
use pivot_table() instead of pivot() if dupes
df.reset_index() - brings back index as col

Use \ for chars in string 'That\'s cool' or 'bin\\usr'
Can also use r"string" to make raw, r"bin\usr"
Use f'string {var1}' to include vars in string
string1+string2 = "string1string2"
s1.find('a') - index of first occurrence
s1.replace('a','b') - replace all 'a' with 'b'
sep.join(s1) - puts sep b/w each element in s1
df['col1'].str.func(): split, strip, findall, contains
Regex: r'^[AEIOU].*[^aeiou]$'
^[AEIOU] - start of s1 has vowel
. - anything, * - 0 or more times
[^] - not, [^aeiou]$ not vowel for last char

df['col1'].cumsum().plot()
df['col1'].cumsum().plot.line(fontsize = 10,
                              linewidth = 3, color = 'r',
ylabel='km')
```

Method	Plot Type
`bar` or `barh`	bar plots
`hist`	histogram
`box`	boxplot
`kde` or `density`	density plots
`area`	area plots
`scatter`	scatter plots
`hexbin`	hexagonal bin plots
`pie`	pie plots

```
pd.merge(df1, df2, how='outer', left/right_index = True, on =
['col1', 'col2','col3'])
pd.merge(left = df1, right = df2, how = 'left', left_on = 'col1',
right_on = 'Col1')
```

* `df.apply(np.sum)`, applies a function **column-wise or row-wise** across a dataframe (the function must be able to accept/return an array)
* `df.map()`, applies a function element-wise (so individually) across **every element** in the data frame (meant for functions that accept/return single values at a time)
* `series.apply()`, same as above for Pandas `Series`.
* `series.map(lambda x: x**2)`, also for Series, but optionally accepts a dictionary as input.

```
pd.Timestamp('July 9, 2005') or
            (year = 2005, month = 7, day = 9)
pd.Period('2005-07-09', freq = 'H') .start/end_time
pt = pd.Timestamp("2005-07-09 12:00")
Span.start_time < pt < span.end_time
pd.date_range('2020-09-01 12:00',
            '2020-09-11 12:00',
            freq='2d')
pd.DateOffset(days=5) - 5 calendar days
pd.period_range('2020-09-01',
             '2020-09-11',
             freq='d')
Can only use Timedelta() with timestamp of
Appropriate time readings (h with 00:00 in it)
df.index.weekday - 0-6
df.index.second - second
df.index.day_name() - weekday name * `'h'`, `'min'` and `'s'` for
hours, minutes,  and seconds (resp.)
df.index.month_name() - month name   * `'B'` for business day
df['datetime'].dt.year -  use dt on series `'MS'` and `'ME'` for
month start and month end (resp.)
dt.resample("1D") - sample once daily
df['cat_col'] = df['cat_col'].astype('category')
df['cat_col'].cat.categories
df['cat_col].nunique()
pd.Categorical(df['cat_col'],
categories=sorted(df['cat_col'].unique(), reverse=True),
ordered=True)
```