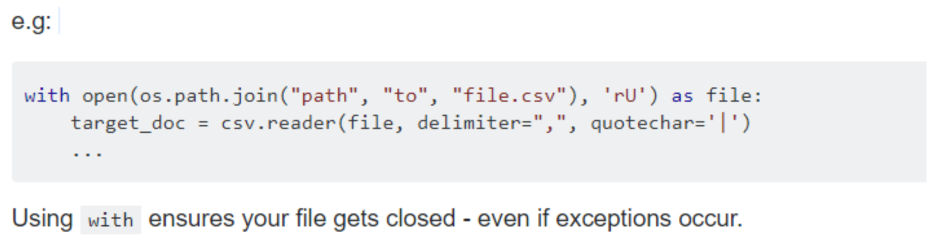
#Part I About path setting

1.Importing csv from a subdirectory in Python

https://stackoverflow.com/questions/10235752/importing-csv-from-a-subdirectory-in-python



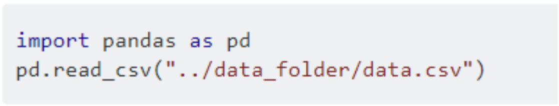
e.g:

```
with open(os.path.join("path", "to", "file.csv"), 'rU') as file:
    target_doc = csv.reader(file, delimiter=",", quotechar='|')
    ...
```

Using `with` ensures your file gets closed - even if exceptions occur.

2.How to open my files in `data_folder` with pandas using relative path?

https://stackoverflow.com/questions/35384358/how-to-open-my-files-in-data-folder-with-pandas-using-relative-path

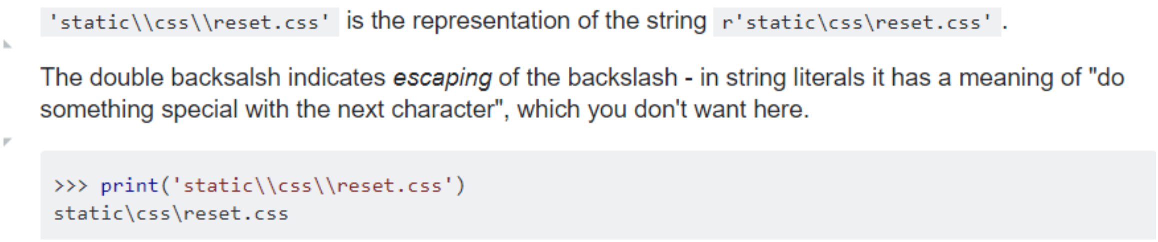


```
import pandas as pd
pd.read_csv("../data_folder/data.csv")
```

3.how to join path

https://stackoverflow.com/questions/17438027/os-path-join-and-os-path-normpath-both-add-double-backwards-slash-on-windows

os.path.join() and os.path.normpath() both add double backwards slash on windows



`'static\\css\\reset.css'` is the representation of the string `r'static\css\reset.css'`.

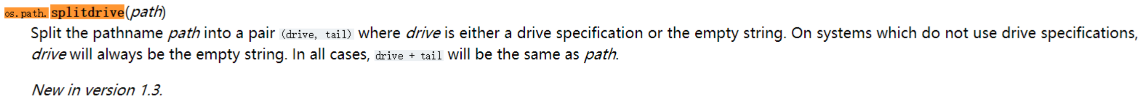
The double backsalsh indicates *escaping* of the backslash - in string literals it has a meaning of "do something special with the next character", which you don't want here.

```
>>> print('static\\css\\reset.css')
static\css\reset.css
```

#Part II system set

1.get the drive letter

https://docs.python.org/2/library/os.path.html



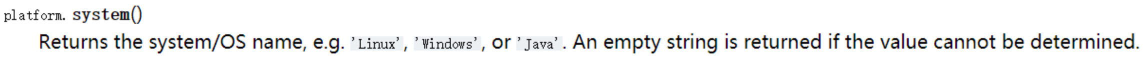
`os.path.splitdrive`(*path*)

Split the pathname *path* into a pair `(drive, tail)` where *drive* is either a drive specification or the empty string. On systems which do not use drive specifications, *drive* will always be the empty string. In all cases, `drive + tail` will be the same as *path*.

New in version 1.3.

2.get the users system

https://docs.python.org/2/library/platform.html



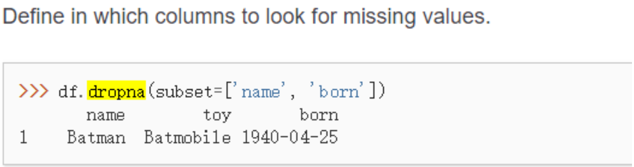
platform.system()

Returns the system/OS name, e.g. 'Linux', 'Windows', or 'Java'. An empty string is returned if the value cannot be determined.

#Part III data_cleaning

1.how to drop na using panda

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html

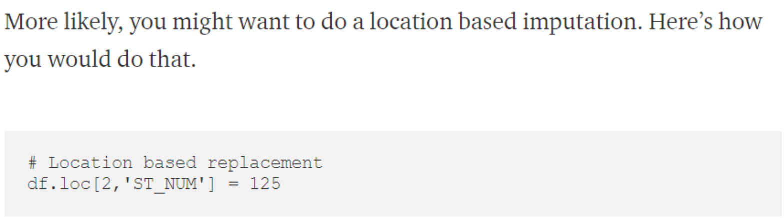


Define in which columns to look for missing values.

```
>>> df.dropna(subset=['name', 'born'])
       name        toy       born
1    Batman  Batmobile 1940-04-25
```

2.remove missing values

https://towardsdatascience.com/data-cleaning-with-python-and-pandas-detecting-missing-values-3e9c6ebcf78b

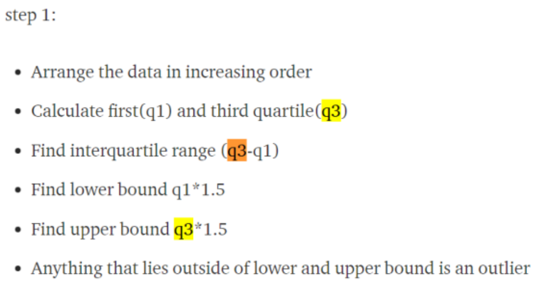


More likely, you might want to do a location based imputation. Here's how you would do that.

```
# Location based replacement
df.loc[2,'ST_NUM'] = 125
```

3.Finding outliers in dataset using python

https://medium.com/datadriveninvestor/finding-outliers-in-dataset-using-python-efc3fce6ce32



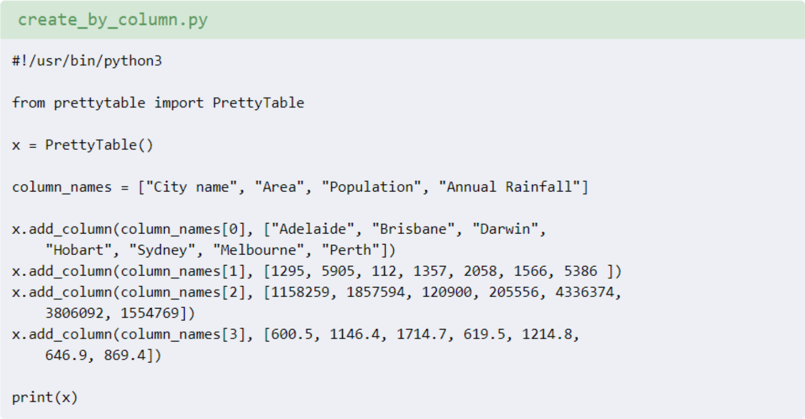
step 1:

- Arrange the data in increasing order
- Calculate first(q1) and third quartile(q3)
- Find interquartile range (q3-q1)
- Find lower bound q1*1.5
- Find upper bound q3*1.5
- Anything that lies outside of lower and upper bound is an outlier

#Part IV data_cleaning

1.using pretty table for drawing

http://zetcode.com/python/prettytable/

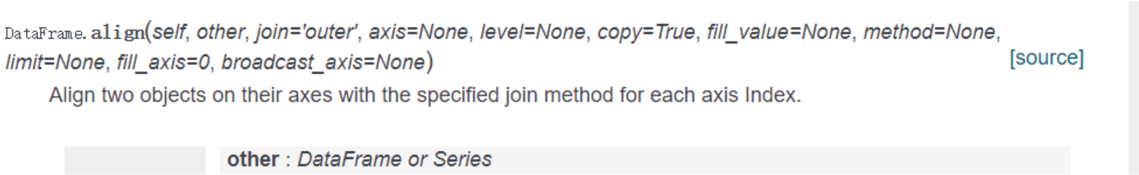


create_by_column.py

```
#!/usr/bin/python3

from prettytable import PrettyTable

x = PrettyTable()

column_names = ["City name", "Area", "Population", "Annual Rainfall"]

x.add_column(column_names[0], ["Adelaide", "Brisbane", "Darwin",
    "Hobart", "Sydney", "Melbourne", "Perth"])
x.add_column(column_names[1], [1295, 5905, 112, 1357, 2058, 1566, 5386 ])
x.add_column(column_names[2], [1158259, 1857594, 120900, 205556, 4336374,
    3806092, 1554769])
x.add_column(column_names[3], [600.5, 1146.4, 1714.7, 619.5, 1214.8,
    646.9, 869.4])

print(x)
```

2.pandas.DataFrame.align¶

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.align.html



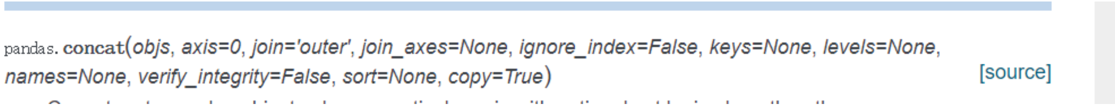
DataFrame.**align**(*self, other, join='outer', axis=None, level=None, copy=True, fill_value=None, method=None, limit=None, fill_axis=0, broadcast_axis=None*) [source]

Align two objects on their axes with the specified join method for each axis Index.

other : *DataFrame or Series*

3. pandas.concat

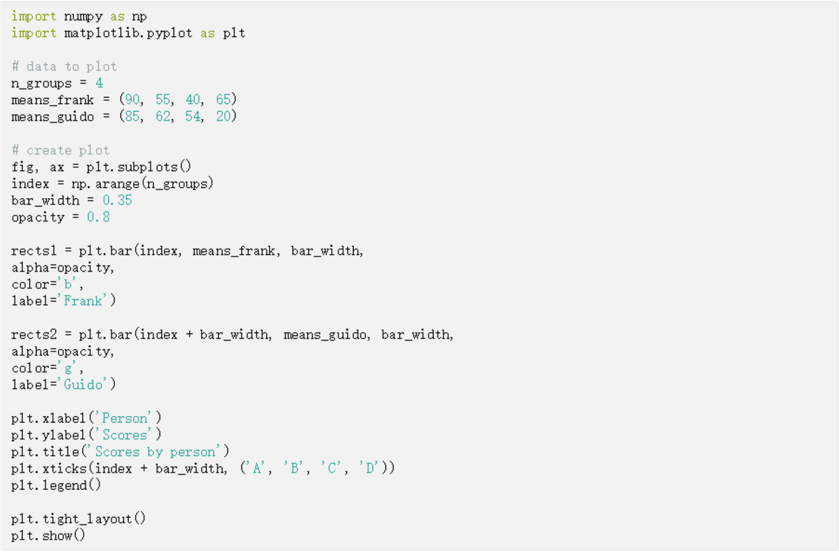
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html



pandas.**concat**(*objs, axis=0, join='outer', join_axes=None, ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, sort=None, copy=True*) [source]

4.compare two data series using this Matplotlib code:

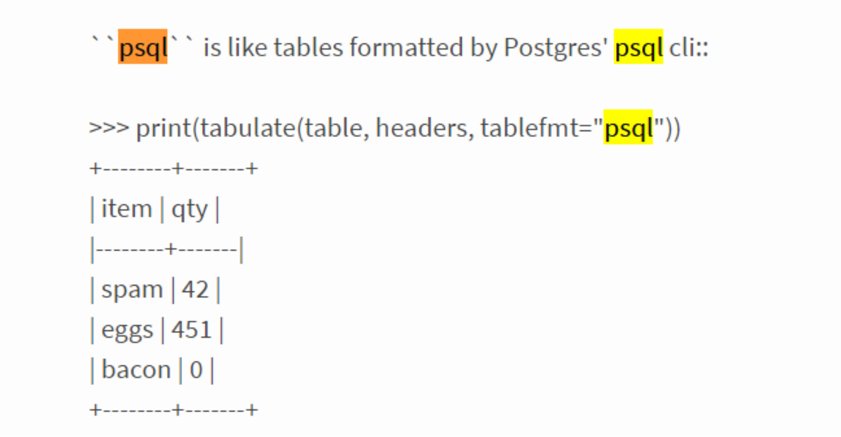
https://pythonspot.com/matplotlib-bar-chart/



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

# data to plot
n_groups = 4
means_frank = (90, 55, 40, 65)
means_guido = (85, 62, 54, 20)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, means_frank, bar_width,
alpha=opacity,
color='b',
label='Frank')

rects2 = plt.bar(index + bar_width, means_guido, bar_width,
alpha=opacity,
color='g',
label='Guido')

plt.xlabel('Person')
plt.ylabel('Scores')
plt.title('Scores by person')
plt.xticks(index + bar_width, ('A', 'B', 'C', 'D'))
plt.legend()

plt.tight_layout()
plt.show()
```

5.using tabulate

https://pypi.org/project/tabulate/

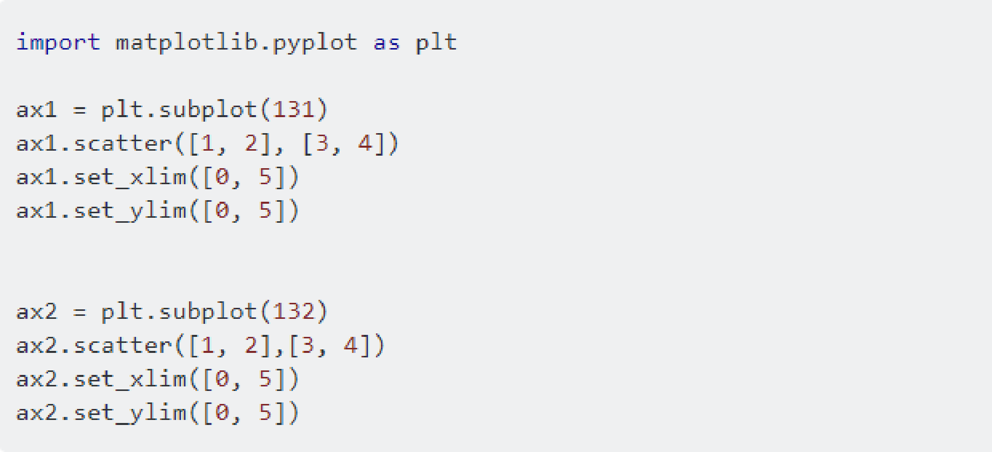


``psql`` is like tables formatted by Postgres' psql cli::

```
>>> print(tabulate(table, headers, tablefmt="psql"))
+--------+-------+
| item | qty |
|--------+-------|
| spam | 42 |
| eggs | 451 |
| bacon | 0 |
+--------+-------+
```

6. draw for scatter

https://pythonspot.com/matplotlib-scatterplot/

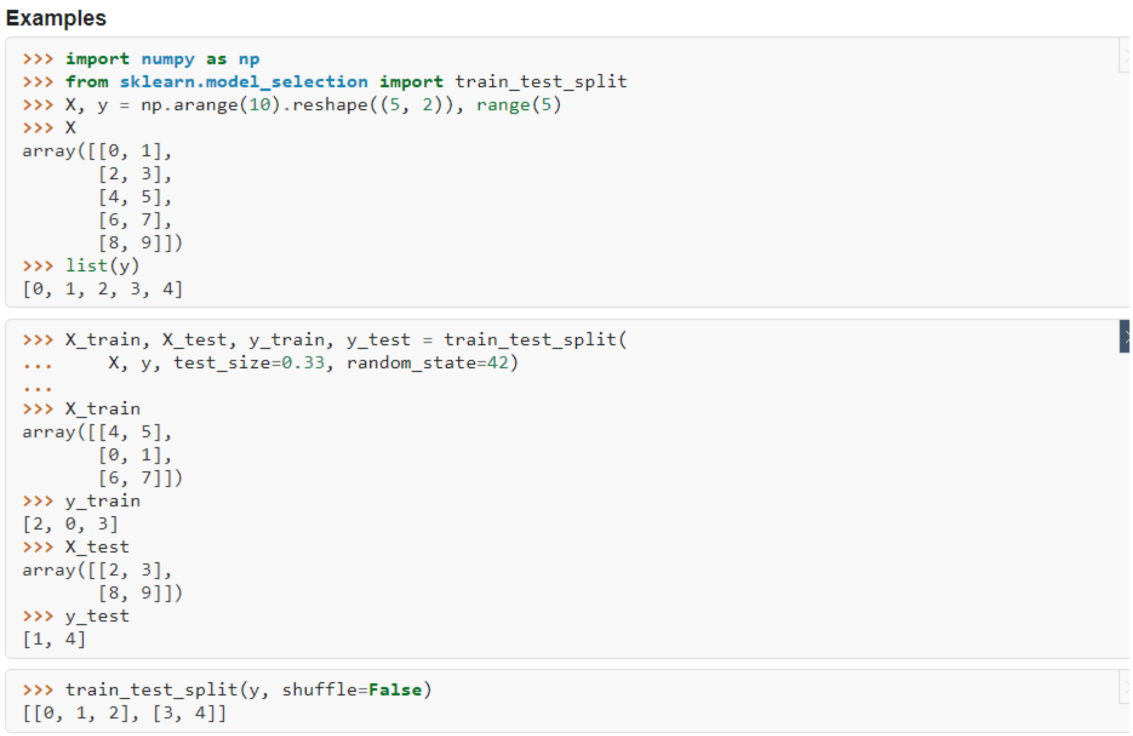


```
import matplotlib.pyplot as plt

ax1 = plt.subplot(131)
ax1.scatter([1, 2], [3, 4])
ax1.set_xlim([0, 5])
ax1.set_ylim([0, 5])


ax2 = plt.subplot(132)
ax2.scatter([1, 2],[3, 4])
ax2.set_xlim([0, 5])
ax2.set_ylim([0, 5])
```

7. for data splitting into training and testing dataset

sklearn.model_selection.train_test_split

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html



Examples

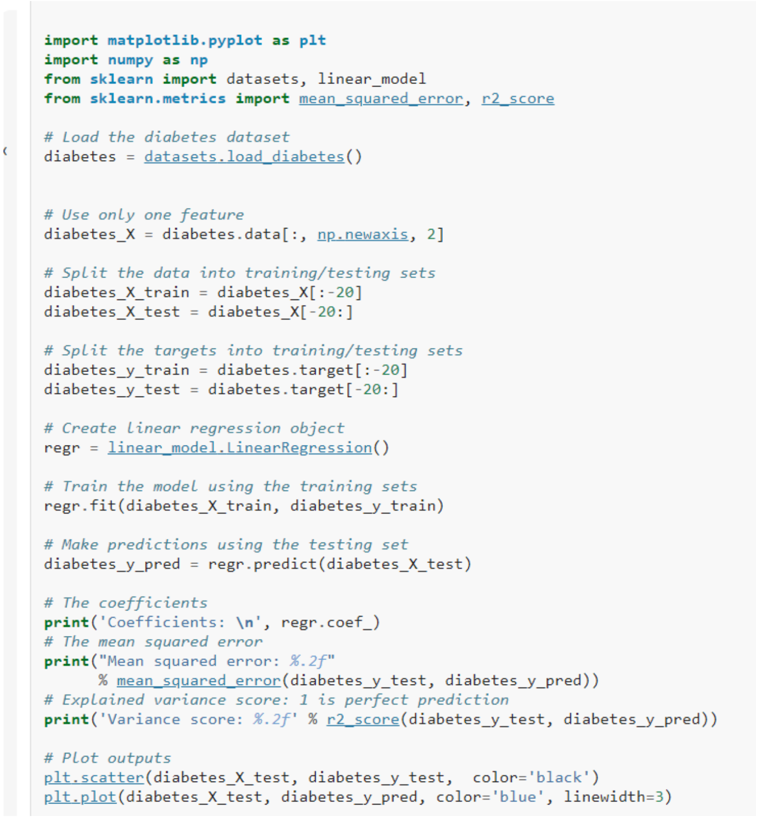
```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

```
>>> train_test_split(y, shuffle=False)
[[0, 1, 2], [3, 4]]
```

8.Linear Regression Example

https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html



```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()


# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
```