|  |  |
| --- | --- |
| Panda data\_frame | import numpy as np  import pandas as pd |
| Panda  pivot\_table example | df = pd.read\_csv  ('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df\_1 = df.pivot\_table(  index = ['month', 'day'],  columns = ['rain', 'wind'],  aggfunc = 'mean')  print(df\_1.head(20)) |
| Apply function on panda | series\_1 = pd.Series([6,7,8,9,2,3,4,5] )  series\_2 = series\_1.apply(lambda x: x\*\*2)  print(series\_1)  print(series\_2) |
|  | df = pd.read\_csv(  'https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  print(df.describe())  print(df.columns)  print(df.shape) |
| Multiply element by 2 and use element as index | n = 4  arr = np.arange(1, n+1)  result = np.arange(1, n+1) \*\* 2    s = pd.Series(result, index = arr)  print(s) |
| Sort column values | df = pd.read\_csv(  'https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df\_2 = df.sort\_values(by=['month', 'day'])  print(df\_2.head()) |
| Set the index (header) | df = pd.read\_csv(  'https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df\_2 = df.set\_index('X')  print(df\_2.head()) |
| Print only even numbers ( :: means at interval) | df = pd.read\_csv('  https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df\_2 = df[2::2]  print(df\_2.head(20)) |
| Dataframe iloc and loc.. print all rows and columen tfrom 3 until 6 | df = pd.read\_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df\_2 = df.iloc[:,3:6]  print(df\_2.head(20)) |
| rows 2 to 20 and all column | df\_2 = df.loc[ 2 : 20]  print(df\_2) |
| Function filter where area, wind and temp is > ? | df = pd.read\_csv(  'https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df\_2 = df.loc[(df.area > 0) & (df.wind > 1) & (df.temp > 15), :]  print(df\_2.head(20)) |
| Merge with unique columns unique\_id | df\_1 = pd.read\_csv(  'https://query.data.world/s/vv3snq28bp0TJq2ggCdxGOghEQKPZo')  df\_2 = pd.read\_csv(  'https://query.data.world/s/9wVKjNT0yiRc3YbVJaiI8a6HGl2d74')  df\_3 = pd.merge(df\_1, df\_2, how = 'inner', on = 'unique\_id')  print(df\_3.head(20)) |
| Concat two dataframe df\_1, df\_2 | import warnings  warnings.filterwarnings('ignore')  import pandas as pd  df\_1 = pd.read\_csv(  'https://query.data.world/s/vv3snq28bp0TJq2ggCdxGOghEQKPZo')  df\_2 = pd.read\_csv(  'https://query.data.world/s/9wVKjNT0yiRc3YbVJaiI8a6HGl2d74')  df\_3 = pd.concat([df\_1, df\_2])  print(df\_3.head()) |
| (Two data sets where column is different and aggregate per columns.  Trick is to use column as index and then perform join as it works on index (name) | Given three data frames containing the number of gold, silver, and bronze Olympic medals won by some countries, determine the total number of medals won by each country.  gold.set\_index('Country', inplace = True)  silver.set\_index('Country', inplace = True)  bronze.set\_index('Country', inplace = True)  gold\_silver = gold.add(silver, fill\_value = 0)  gold\_silver\_bronze = gold\_silver.add(bronze, fill\_value = 0)  print(gold\_silver\_bronze.sort\_values(by='Medals', ascending = False))  <https://stackoverflow.com/questions/66162405/using-add-function-to-merge-multiple-dataframes-in-panda> |
| Group by month & day and mean on rain and wind | df = pd.read\_csv('  https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  #Type your command here  df\_1 = df.groupby( ['month', 'day']) ['rain',' wind'].mean()  print(df\_1.head(20)) |
| Create a new column 'XY' | f = pd.read\_csv(  'https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df['XY'] = df['X'] \* df['Y']  print(df.head(20)) |
| Group by index and mean on columns. Pivot\_table alternative of group by clause | df = pd.read\_csv(  'https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')  df\_1 = df.pivot\_table(  index = ['month', 'day'],  columns = ['rain', 'wind'],  aggfunc = 'mean')  print(df\_1.head(20)) |
| Dataset column missing values is not 5 (!=5) | import pandas as pd  marks = pd.read\_csv('https://query.data.world/s/HqjNNadqEnwSq1qnoV\_JqyRJkc7o6O')  marks = marks[marks.isnull().sum(axis=1) !=5]  print(marks.isnull().sum()) |
| Extract date and time | order = pd.read\_csv('https://query.data.world/s/3hIAtsCE7vYkPEL-O5DyWJAeS5Af-7')  order['Order\_Date'] = pd.to\_datetime(order['Order\_Date'])  order['day'] = order['Order\_Date'].dt.day  print(order.head(10)) |
|  |  |
|  | df = pd.read\_csv('/Users/nyashasingh/Downloads/EDA\_census\_2.csv')    df['AREA\_NAME'] = df['AREA\_NAME'].str.replace('State - ','')    columns = ['AREA\_NAME', 'FEMALES', 'L\_FEMALES']    df\_2 = df[columns]  df\_2 = df\_2.groupby(['AREA\_NAME'])[columns].sum()  df\_2 = df\_2.sort\_values(by='FEMALES', ascending = False)  df\_2['LITERACY\_RATE'] = df\_2['L\_FEMALES']\*100/df\_2['FEMALES']  print(df\_2['LITERACY\_RATE']) |
| Calculate lowest literacy rate | df = pd.read\_csv('/Users/nyashasingh/Downloads/EDA\_census\_2.csv')    df['AREA\_NAME'] = df['AREA\_NAME'].str.replace('State - ','')    columns = ['AREA\_NAME', 'PERSONS', 'MALES', 'FEMALES', 'L\_PERSONS', 'L\_MALES', 'L\_FEMALES']    df\_2 = df[columns]  df\_2 = df\_2.groupby(['AREA\_NAME'])[columns].sum()    df\_2['TOTAL'] = (df\_2['PERSONS'] + df\_2['MALES'] + df\_2['FEMALES'])  df\_2['L\_TOTAL'] = (df\_2['L\_PERSONS'] + df\_2['L\_MALES'] + df\_2['L\_FEMALES'])  df\_2['L\_TOTAL\_RATE'] = df\_2['L\_TOTAL'] / df\_2['TOTAL']  df\_3 = df\_2.sort\_values(by=['L\_TOTAL\_RATE'])    df\_3 = df\_3[['L\_TOTAL\_RATE']]    print(df\_3) |
| Age wise summation | import pandas as pd  import numpy as np  df = pd.read\_csv('/Users/nyashasingh/Downloads/EDA\_census\_2.csv')    columns = ['AGE\_GROUP', 'L\_PERSONS', 'L\_MALES', 'L\_FEMALES']    df\_2 = df.sort\_values(by=['AGE\_GROUP'])  df\_2 = df\_2.groupby(['AGE\_GROUP'])[columns].sum()    print(df\_2) |

|  |  |
| --- | --- |
| if any null value for each column | df.isnull.any(axis=0) |
| Sum of missing values for each row | df.isnull().sum(axis=1) |
| Rows having more than 5 missing value | df [ df.isnull().sum(axis=1) > 5 ] |
| Remove price column with nan | df [~np.isnan (df[‘price’]) ] |
| Rows having price column missing | df [np.isnan ( df[‘price’] ) ] |
| Convert car to type category | df['car = df['car'].astype('category') |
| Displaying frequencies of each category  **How many times a column value has occurred** | df['car'].value\_counts() |
| Imputing nans by 2.0 | df.loc[pd.isnull(df['car']) , ['car'] ] = 2  round(100\*( df.isnull().sum() / len(df.index)), 2) |
| Percentage of missing value in each column | import pandas as pd  df = pd.read\_csv(')  round (100 \* (df.isnull().sum() / len(df.index) ) , 2) |
| Percentage of missing value, rows having more than 5 missing value. Trick is to count only until 5 | df = pd.read\_csv()  df2 = df [df.isnull().sum( axis=1) > 5]  df.drop(df2.index, axis = 0, inplace = True)  round (100 \* (df.isnull().sum() / len(df.index) ) , 2)  df = pd.read\_csv()  df = df[df.isnull().sum(axis=1) <= 5]  round(100\*(df.isnull().sum()/len(df.index)), 2) |
| Impute the mean value at all the missing values of the column 'Product\_Base\_Margin' and then print the percentage of missing values in each column | df.loc[ np.isnan(df['Product\_Base\_Margin']), ['Product\_Base\_Margin'] ] =df['Product\_Base\_Margin'].mean()  round(100\*(df.isnull().sum()/len(df.index)), 2) |

* How to clean rows where records are missing

|  |
| --- |
| * Check rows and columns which have missing values (df.isnull().sum(axis=?) * Calculate columns which have high percentage of missing values and drop it * round(100\*(df.isnull().sum()/len(df.index)), 2) * df.drop(‘………….’, axis=1) * Take the rows which have less than 5 missing column values * Calculate percentage len(df[].index) * Remove the column which u don’t want (bcoz price is being predicted)   df[ ~np.isnan(df [‘Price] )]   * Calculate percentage round(100\*(df.isnull().sum()/len(df.index)), 2) * df[‘……...’].describe() => standard deviation is too high so cannot calculate man value remove it * get rid of columns from dataframe   df[~ np.isnan(df[‘Landsize])]   * multiple columns describe() command. 🡺 will print mean, mode, std, quartile   df.loc[ : , [‘Lattitude’, ‘Longitude’]].describe()   * if std deviation is not variying then impute the value   df.loc[np.isnan(df[‘…..’]), [‘……’] ] =df[……….].mean()   * find out the frequencies.   df['car] = df[‘car].astype(‘category’)  df[‘car’].value\_counts()  df.loc [pd.isnull(df[‘Car’]), [‘Car’] ] = 2. #impute the values by 2  round (100 \* (df.isnull().sum() / len(df.index) ), 2)   * len (df.index)/……. 🡺 calculate original percentage of data how much u have lost |

|  |
| --- |
| <https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/expected-return/> |
| Phrasing the conditions more formally, the binomial distribution can be used if, for an experiment:   * The **total number** of trials is **fixed** * Each trial is **binary**, i.e. has **only two possible outcomes**, success and failure * The **probability of success** is the **same** for all the trials |
| 1. **Sampling distribution’s mean** (μ¯X) = **Population mean** (μ) 2. Sampling distribution’s standard deviation (**Standard error**) = σ/√n. (n is sample size, σ is populations’ standard deviation) 3. **For n > 30**, the sampling distribution becomes a **normal distribution** |
| Then, you generalised the whole process. Let’s say you have a sample with **sample size n, mean**\\bar{X}**and standard deviation S**. You learnt that the **y% confidence interval** (i.e. confidence interval corresponding to y% confidence level) for \\mu will be given by the range:  Confidence interval = (\\bar{X}-\\frac{Z^{*}S}{\\sqrt{n}}, \\bar{X}+\\frac{Z^{*}S}{\\sqrt{n}})   Where, **Z\*** is the Z-score associated with a **y% confidence level**. |