**Importing Data**

Any kind of data analysis starts with getting hold of some data. [Pandas](https://pandas.pydata.org/pandas-docs/stable/) gives you plenty of options for getting data into your Python workbook:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | pd.read\_csv(filename) # From a CSV file  pd.read\_table(filename) # From a delimited text file (like TSV)  pd.read\_excel(filename) # From an Excel file  pd.read\_sql(query, connection\_object) # Reads from a SQL table/database  pd.read\_json(json\_string) # Reads from a JSON formatted string, URL or file.  pd.read\_html(url) # Parses an html URL, string or file and extracts tables to a list of dataframes  pd.read\_clipboard() # Takes the contents of your clipboard and passes it to read\_table()  pd.DataFrame(dict) # From a dict, keys for columns names, values for data as lists |

**Exploring Data**

Once you have imported your data into a Pandas dataframe, you can use these methods to get a sense of what the data looks like:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | df.shape() # Prints number of rows and columns in dataframe  df.head(n) # Prints first n rows of the DataFrame  df.tail(n) # Prints last n rows of the DataFrame  df.info() # Index, Datatype and Memory information  df.describe() # Summary statistics for numerical columns  s.value\_counts(dropna=False) # Views unique values and counts  df.apply(pd.Series.value\_counts) # Unique values and counts for all columns  df.describe() # Summary statistics for numerical columns  df.mean() # Returns the mean of all columns  df.corr() # Returns the correlation between columns in a DataFrame  df.count() # Returns the number of non-null values in each DataFrame column  df.max() # Returns the highest value in each column  df.min() # Returns the lowest value in each column  df.median() # Returns the median of each column  df.std() # Returns the standard deviation of each column |

**Selecting**

Often, you might need to select a single element or a certain subset of the data to inspect it or perform further analysis. These methods will come in handy:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | df[col] # Returns column with label col as Series  df[[col1, col2]] # Returns Columns as a new DataFrame  s.iloc[0] # Selection by position (selects first element)  s.loc[0] # Selection by index (selects element at index 0)  df.iloc[0,:] # First row  df.iloc[0,0] # First element of first column |

**Data Cleaning**

If you’re working with real world data, chances are you’ll need to clean it up. These are some helpful methods:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | df.columns = ['a','b','c'] # Renames columns  pd.isnull() # Checks for null Values, Returns Boolean Array  pd.notnull() # Opposite of s.isnull()  df.dropna() # Drops all rows that contain null values  df.dropna(axis=1) # Drops all columns that contain null values  df.dropna(axis=1,thresh=n) # Drops all rows have have less than n non null values  df.fillna(x) # Replaces all null values with x  s.fillna(s.mean()) # Replaces all null values with the mean (mean can be replaced with almost any function from the statistics section)  s.asxtype(float) # Converts the datatype of the series to float  s.replace(1,'one') # Replaces all values equal to 1 with 'one'  s.replace([1,3],['one','three']) # Replaces all 1 with 'one' and 3 with 'three'  df.rename(columns=lambda x: x + 1) # Mass renaming of columns  df.rename(columns={'old\_name': 'new\_ name'}) # Selective renaming  df.set\_index('column\_one') # Changes the index  df.rename(index=lambda x: x + 1) # Mass renaming of index |

**Filter, Sort and Group By**

Methods for filtering, sorting and grouping your data:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | df[df[col] > 0.5] # Rows where the col column is greater than 0.5  df[(df[col] > 0.5) & (df[col] < 0.7)] # Rows where 0.5 < col < 0.7  df.sort\_values(col1) # Sorts values by col1 in ascending order  df.sort\_values(col2,ascending=False) # Sorts values by col2 in descending order  df.sort\_values([col1,col2], ascending=[True,False]) # Sorts values by col1 in ascending order then col2 in descending order  df.groupby(col) # Returns a groupby object for values from one column  df.groupby([col1,col2]) # Returns a groupby object values from multiple columns  df.groupby(col1)[col2].mean() # Returns the mean of the values in col2, grouped by the values in col1 (mean can be replaced with almost any function from the statistics section)  df.pivot\_table(index=col1, values= col2,col3], aggfunc=mean) # Creates a pivot table that groups by col1 and calculates the mean of col2 and col3  df.groupby(col1).agg(np.mean) # Finds the average across all columns for every unique column 1 group  df.apply(np.mean) # Applies a function across each column  df.apply(np.max, axis=1) # Applies a function across each row |

**Joining and Combining**

Methods for combining two dataframes:

Python

|  |  |
| --- | --- |
| 1  2  3 | df1.append(df2) # Adds the rows in df1 to the end of df2 (columns should be identical)  pd.concat([df1, df2],axis=1) # Adds the columns in df1 to the end of df2 (rows should be identical)  df1.join(df2,on=col1,how='inner') # SQL-style joins the columns in df1 with the columns on df2 where the rows for col have identical values. how can be one of 'left', 'right', 'outer', 'inner' |

**Writing Data**

And finally, when you have produced results with your analysis, there are several ways you can export your data:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | df.to\_csv(filename) # Writes to a CSV file  df.to\_excel(filename) # Writes to an Excel file  df.to\_sql(table\_name, connection\_object) # Writes to a SQL table  df.to\_json(filename) # Writes to a file in JSON format  df.to\_html(filename) # Saves as an HTML table  df.to\_clipboard() # Writes to the clipboard |

**Machine Learning**

print(df.shape)

print(df.head(5))

print(df.tail(5))

print(df.info())

print(df.describe())

print(df.iloc[6,1] )

print(df.info())

print(df.isna().sum())

df = df [['company','price']][df.price==df['price'].max()]

print(df)

df = pd.read\_csv("D:\\Automobile\_data.csv")

car\_Manufacturers = df.groupby('company')

toyotaDf = car\_Manufacturers.get\_group('audi')

print(toyotaDf)

print(df['company'].value\_counts())

df = pd.read\_csv("D:\\Automobile\_data.csv")

car\_Manufacturers = df.groupby('company')

priceDf = car\_Manufacturers['company','price'].max()

print(priceDf)

df = pd.read\_csv("D:\\Automobile\_data.csv")

car\_Manufacturers = df.groupby('company')

mileageDf = car\_Manufacturers['company','average-mileage'].mean()

print(mileageDf)

carsDf = pd.read\_csv("D:\\Automobile\_data.csv")

carsDf = carsDf.sort\_values(by=['price', 'horsepower'], ascending=False)

print(carsDf.head(15))

GermanCars = {'Company': ['Ford', 'Mercedes', 'BMV', 'Audi'], 'Price': [23845, 171995, 135925 , 71400]}

carsDf1 = pd.DataFrame.from\_dict(GermanCars)

japaneseCars = {'Company': ['Toyota', 'Honda', 'Nissan', 'Mitsubishi '], 'Price': [29995, 23600, 61500 , 58900]}

carsDf2 = pd.DataFrame.from\_dict(japaneseCars)

carsDf = pd.concat([carsDf1, carsDf2], keys=["Germany", "Japan"])

print(carsDf)

Car\_Price = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'Price': [23845, 17995, 135925 , 71400]}

carPriceDf = pd.DataFrame.from\_dict(Car\_Price)

car\_Horsepower = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'horsepower': [141, 80, 182 , 160]}

carsHorsepowerDf = pd.DataFrame.from\_dict(car\_Horsepower)

carsDf = pd.merge(carPriceDf, carsHorsepowerDf, on="Company")

print(carsDf)