Python Data Processing Commands

|  |  |  |  |
| --- | --- | --- | --- |
| **Method/Command** | **Source Library** | **Example** | **Description** |
| open() | Inbuilt | open("sample.csv", 'r’) | Open file in read mode and return stream |
| readline() | Inbuilt | infile.readline() | Returns line from file |
| detect() | chardet | chardet.detect(f.read(100000)) | Find encoding and other details by reading 100KB of file data |
| read\_csv() | pandas | pandas.read\_csv(  "sample.csv",  encoding=”ascii”,  sep=",",  on\_bad\_lines="warn",  engine="python"  ) | Read csv file and return data frame |
| head() | pandas | data\_frame.head() | Return top 5 rows of data frame |
| <data frame variable>[<column name>] | pandas | data\_frame['SepalLength'] | Return mentioned column values from data frame |
| to\_numeric() | Pandas | df[col] = pandas.to\_numeric(df[col], errors="coerce") | Convert input to numeric type |
| <data frame variable>[<column name>].median() | Pandas | df["SepalWidth"].median() | Returns median value from the given column’s values |
| fillna() | Pandas | df["SepalWidth"] = df["SepalWidth"].fillna(df["SepalWidth"].median()) | Fill null values of the mentioned column with median value |
| describe() | Pandas | df.describe() | Generate descriptive statistics like min, max, count, mean and others |
| rename() | Pandas | df = df.rename(columns={  "SepalWidth": "SepalWidth2",  "SepalLength": "SepalLength2"  }) | Change column names of data frame |
| writer() | csv | writer = csv.writer(f\_out) | Takes file reference and creates writer object |
| writerow() | csv | writer.writerow(headers) | Adds given variable data to the file through writer object |
| read\_excel() | Pandas | df = pandas.read\_excel("./dataset\_samples/Customer\_Info.xlsx") | Read excel file |
| between() | Pandas | df["AgeValid"] = df["Age"].between(18, 99) | For each row validates age and returns Boolean value |
| contains() | Pandas | df["EmailValid"] = df["Email"].str.contains(r"^[^@]+@[^@]+\.[^@]+$", na=False) | Considers each value of the column as string and checks for mentioned expression. Returns Boolean for each value |
| r"^[^@]+@[^@]+\.[^@]+$" | Inbuilt | df["EmailValid"] = df["Email"].str.contains(r"^[^@]+@[^@]+\.[^@]+$", na=False) | **^** to start of the string  **[^@]+** Match one or more characters that are NOT '@'  **@** Literal '@' character  **[^@]+** Again, match one or more characters that are NOT '@'  **\.** Literal dot (.), escaped so it’s not treated as “any character”  **[^@]+** Match one or more non-@ characters (like “com”)  **$** Anchors the pattern to end of the string |
| astype() | Pandas | df["PurchaseAmount"] = (  df["PurchaseAmount"]  .astype(str)  .str.replace(",", "", regex=False)  .replace("N/A", pd.NA)  .astype(float)  ) | Convert everything to string, float |
| replace() | Pandas | df["PurchaseAmount"] = (  df["PurchaseAmount"]  .astype(str)  .str.replace(",", "", regex=False)  .replace("N/A", pd.NA)  .astype(float)  ) | Replace a string with new string |
| strip() | Pandas | df["Name"] = df["Name"].str.strip() | Trim whitespace from names |
| to\_datetime | Pandas | df["JoinDate"] = pd.to\_datetime(df["JoinDate"], errors="coerce") | Convert JoinDate to date and time |
| timestamp() | Pandas | pandas.timestamp("2022-01-01") | Convert string to date and time |
| to\_excel() | Pandas | df.to\_excel("./dataset\_samples/Customer\_Info\_Cleaned.xlsx", index=False) | Create excel from data frame |
| to\_csv | Pandas | df.to\_csv("./dataset\_samples/Customer\_Info\_Cleaned.csv", index=False, encoding="utf-8") | Create csv from data frame |
| load() | Json | with open("./dataset\_samples/D3\_Customer\_Orders.json") as f:  data = json.load(f) | Creates a list with dictionaries |
| json\_normalize | Pandas | df = pd.json\_normalize(  data,  record\_path=["orders"],  meta=[  ["customer", "id"],  ["customer", "name"],  ["customer", "email"],  ["customer", "address", "city"],  ["customer", "address", "zip"]  ],  meta\_prefix="",  errors="ignore"  ) | Flatten the Orders with Customer Info. This gives you one row per order, with customer details included in each row. |
| columns.str.replace() | Pandas | df.columns = df.columns.str.replace(r"\.", "\_", regex=True) | Replace text in column names |
| Multiplication | Pandas | df["total\_value"] = df["price"] \* df["quantity"] | Multiply column values and create new column |
| groupby() with count() | Pandas | df.groupby("customer\_name")["item"].count() | Count of items ordered by each customer |
| groupby() with sum() | Pandas | df.groupby("customer\_id")["total\_value"].sum() | Total spend per customer |
| Filter Operation | Pandas | df[df["delivery\_status"] == "Delivered"] | Filter only delivered orders |
| Filter Operation with Date | Pandas | df[pd.to\_datetime(df["delivery\_date"], errors="coerce") > "2023-06-03"] | Orders after June 3rd |
| to\_excel() with multiple sheets | Pandas | df\_delivered = df[df["delivery\_status"] == "Delivered"]  df\_pending = df[df["delivery\_status"] == "Pending"]  with pd.ExcelWriter("./dataset\_samples/D3\_Flattened\_Orders\_Split.xlsx") as writer:  df\_delivered.to\_excel(writer, sheet\_name="Delivered", index=False)  df\_pending.to\_excel(writer, sheet\_name="Pending", index=False) | Export to multiple sheets in same excel |
| len() | Inbuilt | summary = {  "Total Orders": len(df),  "Unique Customers": df["customer\_id"].nunique(),  "Total Spend": df["total\_value"].sum()  } | Get number of records of data frame |
| nunique() | Pandas | df["customer\_id"].nunique() | Unique Customers count |
| sum() | Pandas | df["total\_value"].sum() | Sum of the given column values |
| sort\_values() | Pandas | df\_sorted= df.sort\_values(ascending=False, by="TotalSales").head(3) **OR** df\_sorted=df.sort\_values("TotalSales", ascending=False).head(3) **OR** df.sort\_values("TotalSales", ascending=False, inplace=True) | Sort values in data frame. If memory is less, inplace helps to modify the existing df |
| read\_excel() | Pandas | df = pd.read\_excel("monthly\_sales.xlsx", sheet\_name="Sheet2") | Read only specific sheet |
| read\_excel() with parsing dates | Pandas | df = pd.read\_excel(  "employee\_records.xlsx",  parse\_dates=["JoiningDate", "LastLogin"]  ) | Read excel file along by parsing dates fields |
| read\_excel() all sheets | Pandas | dfs = pd.read\_excel("file.xlsx", sheet\_name=None) | Read all sheets of excel. It returns dictionary of sheets. Each key is sheet name and value is data frame |
| query() | Pandas | df\_filtered = df.query(  "LastLogin > '2023-07-01' and Status in ['Remote', 'Present']"  ) | Filter rows |
| parse() | xml.etree.ElementTree | tree = ET.parse("./dataset\_samples/D4\_Employees.xml") | Load and parse XML |
| getroot() | xml.etree.ElementTree | tree.getroot() | Get root of xml |
| tag | xml.etree.ElementTree | root.tag | Gives root tag value |
| findall() | xml.etree.ElementTree | for emp in root.findall("employee") | Get all values under employee |
| get() | xml.etree.ElementTree | emp.get("id") | Get attribute value of node |
| find() | xml.etree.ElementTree | name\_tag = emp.find("name") | Get sub-elements, get name |
| text | xml.etree.ElementTree | name\_tag.text | Get value of the tag |
| attrib | xml.etree.ElementTree | "title" in name\_tag.attrib | Gives dictionary of element’s attributes |
| findall() | xml.etree.ElementTree | root.findall(".//employee[@id='E002']") | XPath-style filtering — find specific employee by ID |
| ET.fromstring() | xml.etree.ElementTree | tree\_from\_string = ET.ElementTree(ET.fromstring(xml\_string)) | Tree from string |
| append() | xml.etree.ElementTree | root.append(new\_emp) | Add new element to root |
| write() | xml.etree.ElementTree | tree.write("./dataset\_samples/D4\_Employees.xml", encoding="utf-8", xml\_declaration=True) | Update file/if file not present create new |
| ET.Element() | xml.etree.ElementTree | new\_emp = ET.Element("employee", id="E004") | Create new element |
| ET.SubElement() | xml.etree.ElementTree | ET.SubElement(new\_emp, "name", title="Lead").text = "Kiran" | Add sub-element |
| dataframe() | Pandas | df = pandas.DataFrame({  'id': [1, 2, 3],  'name': ['Sandeep', 'Aditi', 'Rohan']  }) | DataFrame creation |
| to\_parquet() | Pandas | df.to\_parquet('./dataset\_samples/D5\_sample.parquet', index=False) | Write DataFrame to a Parquet file (columnar format, efficient for analytics) |
| read\_parquet() | Pandas | df\_parquet = pd.read\_parquet('./dataset\_samples/D5\_sample.parquet') | Read the Parquet file into a DataFrame |
| to\_csv with compression | Pandas | df.to\_csv('./dataset\_samples/D5\_compressed\_data.csv.gz', index=False, compression='gzip') | Write DataFrame to a GZIP-compressed CSV file |
| read\_csv with compression | Pandas | df\_gzip = pd.read\_csv('./dataset\_samples/D5\_compressed\_data.csv.gz', compression='gzip') | Read the GZIP-compressed CSV file |
| dict() | Inbuilt | compression\_opts = dict(method='zip', archive\_name='nested.csv') | Define zip compression options: specify filename inside the archive |
| to\_csv with compression options | Pandas | df.to\_csv('./dataset\_samples/D5\_archive.zip', index=False, compression=compression\_opts) | Write DataFrame into a ZIP archive containing 'nested.csv' |
| read\_csv of zip file | Pandas | df\_zip = pd.read\_csv('./dataset\_samples/D5\_archive.zip') | Read CSV file from ZIP archive using zip:// protocol. Zip:// is used for multiple files, if single file its not required |
| ZipFile(), write() | Zipfile | with zipfile.ZipFile('./dataset\_samples/D5\_multi\_archive.zip', 'w') as zipf:  zipf.write('./dataset\_samples/D5\_compressed\_data.csv.gz', arcname='compressed.csv.gz')  zipf.write('./dataset\_samples/D5\_sample.parquet', arcname='data.parquet')  zipf.write('./dataset\_samples/D5\_archive.zip', arcname='original\_archive.zip') | Create a zip archive with multiple files |
| guess\_type() | mimetypes | file\_path = './dataset\_samples/D5\_compressed\_data.csv.gz'  print(mimetypes.guess\_type(file\_path)) | Detect file compression format |
| ParquetFile() | pyarrow.parquet | metadata = pq.ParquetFile('./dataset\_samples/D5\_sample.parquet').metadata  print(metadata.schema) | Gives schema, like type of fields it contains |