| **Pandas Command** | **Description** | **Return Type** |
| --- | --- | --- |
| pd.read\_csv("file.csv") | Load CSV into a DataFrame | DataFrame |
| df.head(n) | Show first n rows of the DataFrame | DataFrame |
| df.tail(n) | Show last n rows of the DataFrame | DataFrame |
| df.shape | Get the number of rows and columns | Tuple (int, int) |
| df.columns | Get column names | Index (object) |
| df.dtypes | Get data types of columns | Series (dtype) |
| df.index | Get the index labels | Index |
| df.info() | Summary of DataFrame including non-null values | None |
| df.describe() | Summary statistics of numeric columns | DataFrame |
| df["col"] | Access a single column | Series |
| df[["col1", "col2"]] | Access multiple columns | DataFrame |
| df.loc[row\_index] | Access row(s) by label | Series / DataFrame |
| df.iloc[row\_index] | Access row(s) by integer position | Series / DataFrame |
| df.loc[:, "col"] | Access column(s) by label | Series |
| df.at[row, "col"] | Fast scalar access by label | Scalar (value) |
| df.iat[row, col] | Fast scalar access by integer position | Scalar (value) |
| df["col"].isna() | Detect missing values | Series (boolean) |
| df.dropna() | Drop rows with missing values | DataFrame |
| df.dropna(subset=["col1", "col2"]) | Drop rows with NaN in specified columns | DataFrame |
| df.fillna(value) | Fill missing values with a given value | DataFrame |
| df.fillna(method="ffill") | Forward-fill missing values | DataFrame |
| df.isna() | Boolean DataFrame showing NaNs | DataFrame (boolean) |
| df.notna() | Boolean DataFrame showing non-NaNs | DataFrame (boolean) |
| df.isna().sum() | Count NaNs per column | Series (int) |
| df.insert(loc, "col", value) | Insert a new column at specific position | DataFrame |
| df["col"] = value | Add or modify a column | DataFrame |
| df.drop("col", axis=1) | Drop a column | DataFrame |
| df.drop(index) | Drop rows | DataFrame |
| df.duplicated() | Mark duplicate rows | Series (boolean) |
| df.drop\_duplicates() | Remove duplicate rows | DataFrame |
| df["col"].value\_counts() | Count unique values in a column | Series (int) |
| df["col"].unique() | Get unique values from a column | numpy.ndarray |
| df["col"].nunique() | Count unique values | int |
| df.rename(columns={"old": "new"}) | Rename columns | DataFrame |
| df["col"].astype(type) | Change column data type | DataFrame / Series |
| pd.to\_datetime(df["col"]) | Convert column to datetime | Series (datetime) |
| df.dt.year / .month / .day / .weekday | Extract date parts from datetime column | Series |
| df.sort\_values("col") | Sort DataFrame by a column | DataFrame |
| df.sort\_index() | Sort DataFrame by index | DataFrame |
| df.groupby("col") | Group by values in a column | GroupBy object |
| df.groupby("col").agg(func) | Group and aggregate with custom function(s) | DataFrame |
| df.groupby("col").count() | Count entries per group | Series (int) |
| df.groupby("col").mean() | Get mean per group | Series (float) |
| df.groupby("col").sum() | Get sum per group | Series (float) |
| df.groupby("col").nunique() | Count unique values per group | Series (int) |
| df["col"].idxmax() | Index of max value | int |
| df["col"].idxmin() | Index of min value | int |
| df.stack() | Stack columns into a single column (long format) | DataFrame (Series) |
| df.unstack() | Unstack index levels into columns | DataFrame |
| df["col"].str.split(";", expand=True) | Split string column on ; into multiple columns | DataFrame |
| df.explode("col") | Convert list-like column into separate rows | DataFrame |
| df.apply(func) | Apply a function row-wise or column-wise | Series / DataFrame |
| df.applymap(func) | Apply function elementwise to entire DataFrame | DataFrame |
| df.query("col > 5") | Filter rows with query string | DataFrame |
| df[(df["col1"] > 5) & (df["col2"] < 10)] | Filter rows with multiple conditions | DataFrame |
| df.loc[df["col"] == val, "col2"] = new\_val | Conditional assignment | DataFrame |
| pd.concat([df1, df2], axis=0) | Concatenate DataFrames vertically (rows) | DataFrame |
| pd.concat([df1, df2], axis=1) | Concatenate DataFrames horizontally (columns) | DataFrame |
| df.merge(df2, on="key") | Merge two DataFrames on a key column | DataFrame |
| df.merge(df2, how="left") | Left join | DataFrame |
| df.merge(df2, how="outer") | Outer join | DataFrame |
| df.merge(df2, how="cross") | Cartesian join | DataFrame |
| df.pivot(index, columns, values) | Reshape DataFrame with pivot | DataFrame |
| df.pivot\_table(values, index, columns) | Create pivot table with aggregation | DataFrame |
| df.reset\_index(drop=True) | Reset index and optionally drop old index | DataFrame |
| df.set\_index("col") | Set a column as index | DataFrame |
| df.to\_csv("file.csv") | Save DataFrame as CSV | None |
| df.to\_excel("file.xlsx") | Save DataFrame as Excel | None |
| df.to\_json() | Convert DataFrame to JSON string | str |
| df.to\_dict() | Convert DataFrame to dictionary | dict |
| df.corr() | Compute pairwise correlation | DataFrame |
| df.cov() | Compute pairwise covariance | DataFrame |
| df.diff() | Calculate difference between rows | DataFrame |
| df.cumsum() | Cumulative sum over rows or columns | DataFrame |
| df.cumprod() | Cumulative product | DataFrame |
| df.T | Transpose rows and columns | DataFrame |
| df.abs() | Get absolute values | DataFrame |
| df.add(other) | Add another DataFrame or Series | DataFrame |
| df.clip(lower, upper) | Limit values between lower and upper | DataFrame |
| df.equals(other) | Check if two DataFrames are equal | bool |
| df.empty | Check if DataFrame is empty | bool |
| df.sample(n=5) | Random sample of rows | DataFrame |