( [Pandas Basics Cheat Sheet]() )
( [Scipy cheat sheet]() / [numpy cheat sheet]() )
( [Pandas Wrangling Cheat Sheet]() )

## Read in Data

**pd.read_csv**

Specifying index column

    pd.read_csv(source, **index_coll=0**)

Specifying column names

    columns_names = ['author', 'claps', 'reading_time', 'link', 'title', 'text']
    pd.read_csv (data,index_col=0, **names=columns_names** )

pd.read_csv("../input/titanic/train.csv", **usecols = ["Pclass", "Sex", "Parch", "Cabin"]**)

# When importing the data we can specify the column which includes the NA values and the way NA is defined in the data
data=pd.read_csv (source,index_col=0,names=columns_names,
**na_values={'column_name':[' NaN']}** )

**read_excel / read_table**

Read json: pd.**read_json**("../input/news-headlines-dataset-for-sarcasm-detection/Sarcasm_Headlines_Dataset.json", lines=True)

```
#importing dataset from Google Drive
from google.colab import drive
drive.mount('/content/drive')
dataset='/content/sample_data/articles.csv'
data=pd.read_csv (dataset)
```

**# Reducing Memory Usage by specifying data types for each column**

For more information refer to **[Reducing Memory Size of Pandas.docs]()**

```
# let's see how much our df occupies in memory
df.memory_usage(deep = True)

# convert to smaller datatypes
df = df.astype({"Pclass":"int8",
                "Sex":"category",
                "Parch": "Sparse[int]", # most values are 0
                "Cabin":"Sparse[str]"}) # most values are NaN

df.memory_usage(deep = True)
```

## Looking at Data

★.head() / .tail() / .describe() / .info()

**Pandas Profiling**

https://pandas-profiling.github.io/pandas-profiling/docs/

- **Essentials**: type, unique values, missing values
- **Quantile statistics** like minimum value, Q1, median, Q3, maximum, range, interquartile range
- **Descriptive statistics** like mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness
- **Most frequent values**
- **Histogram**
- **Correlations** highlighting of highly correlated variables, Spearman, Pearson and Kendall matrices
- **Missing values** matrix, count, heatmap and dendrogram of missing values

```
# Installation

pip install pandas-profiling
or
conda install -c anaconda pandas-profiling
```

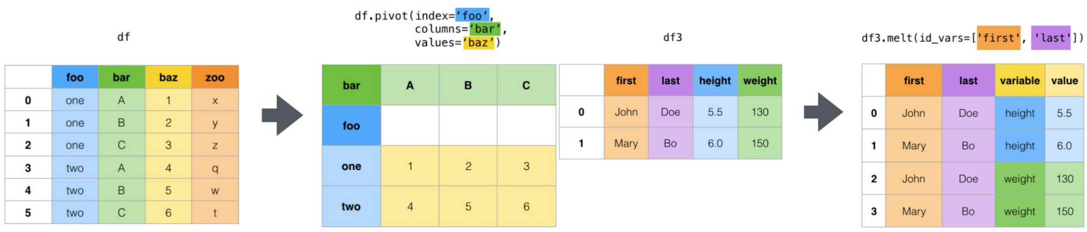
```
#Pandas-Profiling 2.0.0
df.profile_report()

profile = df.profile_report(title='Pandas Profiling Report')
profile.to_file(outputfile="Titanic data profiling.html")

profile = pandas_profiling.ProfileReport(data)
profile
```

## Merging / Pivoting / Changing shape of DataFrames

**merge || Join || Append || concat**
**Pivot || pivot_table**
**Melt || Stack || Unstack**




**# Melt**

```
d = {\
"zip_code": [12345, 56789, 101112, 131415],
"factory": [100, 400, 500, 600],
"warehouse": [200, 300, 400, 500],
"retail": [1, 2, 3, 4]
}

df = pd.DataFrame(d)
df

# location_type is generated automatically from the columns left after specifying id_vars (you can pass a list also)
```

```
df = df.melt(id_vars = "zip_code", var_name = "location_type", value_name = "distance")
df
```

### Unpacking multi index df to single index

**From Feature Engineering Doc**

```
# "train" is the original data and "bureau" is the another numerical data set

bureau_agg =
bureau.drop(columns = ['SK_ID_BUREAU']).groupby('SK_ID_CURR', as_index = False).agg(['count',
'mean', 'max', 'min', 'sum']).reset_index()
bureau_agg.head()
```

| | SK_ID_CURR | DAYS_CREDIT | | | | | CREDIT_DAY_OVERDUE | | | | | DAYS_CREDIT_ENDDATE | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | count | mean | max | min | sum | count | mean | max | min | sum | count | mean | max | min |
| **0** | 100001 | 7 | -735.000000 | -49 | -1572 | -5145 | 7 | 0.0 | 0 | 0 | 0 | 7 | 82.428571 | 1778.0 | -1329.0 |
| **1** | 100002 | 8 | -874.000000 | -103 | -1437 | -6992 | 8 | 0.0 | 0 | 0 | 0 | 6 | -349.000000 | 780.0 | -1072.0 |
| **2** | 100003 | 4 | -1400.750000 | -606 | -2586 | -5603 | 4 | 0.0 | 0 | 0 | 0 | 4 | -544.500000 | 1216.0 | -2434.0 |
| **3** | 100004 | 2 | -867.000000 | -408 | -1326 | -1734 | 2 | 0.0 | 0 | 0 | 0 | 2 | -488.500000 | -382.0 | -595.0 |
| **4** | 100005 | 3 | -190.666667 | -62 | -373 | -572 | 3 | 0.0 | 0 | 0 | 0 | 3 | 439.333333 | 1324.0 | -128.0 |

```
# Unpack multi level index

# List of column names
columns = ['SK_ID_CURR']

# Iterate through the variables names
for var in bureau_agg.columns.levels[0]:
        # Skip the id name
        if var != 'SK_ID_CURR':

                # Iterate through the stat names
                for stat in bureau_agg.columns.levels[1][:-1]:
                        # Make a new column name for the variable and stat
                        columns.append('bureau_%s_%s' % (var, stat))
```

| | SK_ID_CURR | bureau_DAYS_CREDIT_count | bureau_DAYS_CREDIT_mean | bureau_DAYS_CREDIT_max | bureau_DAYS_CREDIT_min | bureau_DAYS_CREDIT_sum |
|---|---|---|---|---|---|---|
| **0** | 100001 | 7 | -735.000000 | -49 | -1572 | -5145 |
| **1** | 100002 | 8 | -874.000000 | -103 | -1437 | -6992 |
| **2** | 100003 | 4 | -1400.750000 | -606 | -2586 | -5603 |
| **3** | 100004 | 2 | -867.000000 | -408 | -1326 | -1734 |
| **4** | 100005 | 3 | -190.666667 | -62 | -373 | -572 |

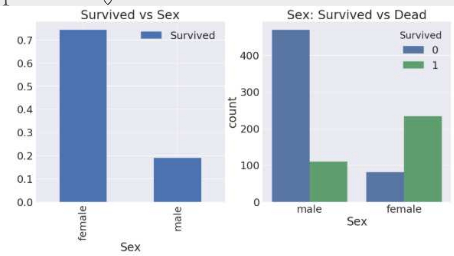
```
# Merge with the training data
train = train.merge(bureau_agg, on = 'SK_ID_CURR', how = 'left')
```

### Various Crosstab Operations

```
pd.crosstab(df_train['Pclass'],
df_train['Survived'],margins=True).style.background_gradient(cmap='summer_r')
```

| Survived | 0 | 1 | All |
|---|---|---|---|
| **Pclass** | | | |
| 1 | 80 | 136 | 216 |
| 2 | 97 | 87 | 184 |
| 3 | 372 | 119 | 491 |
| All | 549 | 342 | 891 |

```
df_train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=True).mean().sort_values(by='Survived', ascending=False).plot.bar()
ax[0].set_title('Survived vs Sex')
sns.countplot('Sex', hue='Survived', data=df_train, ax=ax[1])
ax[1].set_title('Sex: Survived vs Dead')
plt.show()
```




## Apply / map /Applymap / agg / transform Operations

**Applymap**

```
# first let's use applymap to convert to standarize the text
df = df.applymap(lambda x: x.lower() if type(x) == str else x)

mapping = {"male":0, "female":1}

print("PROBLEM: Applies to the whole df but retruns None")
df.applymap(mapping.get)

print("Get the correct result but you have to specify the colums. If you don't want to do this, check the next result")
df[["A", "C"]].applymap(mapping.get)

print("Condtional apply map: if can map --> map else return the same value")
df = df.applymap(lambda x: mapping[x] if x in mapping.keys() else x)
df
```

original df

| | A | B | C | D |
|---|---|---|---|---|
| **0** | Male | x | male | 1 |
| **1** | Female | y | female | 2 |
| **2** | Female | z | male | 3 |
| **3** | Male | A | female | 4 |

**Agg**

```
df.groupby("Pclass").agg(avg_age = ("Age", "mean"),
                         max_age = ("Age", "max"),
                         survival_rate = ("Survived", "mean"))
```

| | avg_age | max_age | survival_rate |
|---|---|---|---|
| **Pclass** | | | |
| **1** | 38.233441 | 80.0 | 0.629630 |
| **2** | 29.877630 | 70.0 | 0.472826 |
| **3** | 25.140620 | 74.0 | 0.242363 |

**Some Frequently Used Apply Operations**

** Extracting month and day information from xxxx/mm/dd string data **

```
df['last_review_month'] = df['last_review'].apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%d").month)
df['last_review_day'] = df ['last_review'].apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%d").day)
```

## Duplicates

**# Dropping Duplicate Columns from data**

```
data = data.loc[:,~data.columns.duplicated()]
```

**# Dropping Duplicate rows from data**

```
df.drop_duplicates(subset=[' ', …], keep='first', inplace=True)
```

## Fixing weird labels & Renaming columns

```
df[x] = df[x].str.replace("$","")
```

```
df[x] = df[x].astype(' ')    'str', 'float64', 'int64' ...
```

```
df = df.rename(columns={"name of column you want to change" : "new column name"})
```

```
d = {"customer": ["A", "B", "C", "D"], "sales":[1100, 950.75, "$400", "$1250.35"]}
df = pd.DataFrame(d)

# Step 1: check the data types
df["sales"].apply(type)

# Step 2: use regex
df["sales"] = df["sales"].replace("[$,]", "", regex = True).astype("float")
df["sales"].apply(type)
```

**Reordering columns by column names (e.g. Q3, Q2, Q6, Q1 to Q1, Q2, Q3, Q6)**

```
Df = df.reindex(sorted(df.columns), axis=1)
```

## Data Types

from pandas.api.types import **CategoricalDtype**

```
cat_type = CategoricalDtype(["bad", "good", "very good", "excellent"], ordered = True)
df["quality"] = df["quality"].astype(cat_type)

print("Now we can use logical sorting.")
df = df.sort_values("quality", ascending = True)

print("We can also filter this as if they are numbers. AMAZING.")
df[df["quality"] > "bad"]
```

**Data types**

determine which variables(features) are categorical and numerical

=> data.select_dtypes(exclude/include=['object'])

*Counter(train.dtypes.values)* ➔ *Counter({dtype('int64'): 49, dtype('float64'): 10})*

```
print("Select numerical columns")
df.select_dtypes(include = "number")

print("Select string columns")
df.select_dtypes(include = "object")
```

```
print("Select datetime columns")
df.select_dtypes(include = ["datetime", "timedelta"])

print("Select miscelaneous")
df.select_dtypes(include = ["number", "object", "datetime", "timedelta"])

print("Select by passing the dtypes you need")
df.select_dtypes(include = ["int8", "int16", "int32", "int64", "float"])
```

## Effective Filtering and Modifying particular values within a dataframe

**Query**

```
# Drop firms that have at least one missing Asset values
df = df.groupby('cik').filter(lambda x: all(pd.notnull(x['Assets'])))
df.query("col == 13")
++++++++++++++++
Pd.query('A > 5 and B < 100')
++++++++++++++++

~~Use a local variable within a query in pandas~~
# create a local variable mean
mean = df["A"].mean()

# now let's use in inside a query of pandas using @
df.query("A > @mean")
```

```
# Reduce
cr1 = df["continent"] == "Europe"
cr2 = df["beer_servings"] > 150
cr3 = df["wine_servings"] > 50
cr4 = df["spirit_servings"] < 60

df[cr1 & cr2 & cr3 & cr4]

IS EQUAL TO

from functools import reduce
criteria = reduce(lambda x, y : x & y , (cr1, cr2, cr3, cr4))
df[criteria]
```

**loc / iloc**

Let's say "A" is the first column name. The following filters all give the same results

```
# using loc --> labels
df.loc[0, "A"]

# using iloc --> position
df.iloc[0, 0]

# mixing labels and position with loc
df.loc[0, df.columns[0]]

# mixing labels and position with loc
df.loc[df.index[0], "A"]
```

```
# mixing labels and position with iloc
df.iloc[0, df.columns.get_loc("A")]

# mixing labels and position with iloc
df.iloc[df.index.get_loc(0), 0]
```

**Modifying particular values that satisfy certain conditions with new values**

```
# replace K with 1000 and create 'claps' column an int64 type #

# Casting column 'claps' values as str
data['claps'].astype('str')

# Storing under K True/False for all values in 'Claps' column to locate values that end with 'K'
K = data['claps'].str.endswith('K')

# Storing under old_value all values in 'claps' column that end with K
old_value = data.loc[K,'claps']

# replacing 'K' in old_value with '000',
casting the values as a float and multiplying by 1000 to get the real values
new = old_value.str.replace('K','').astype('float') *1000

# replacing old_value with the new ones in the claps column and casting the data in the column as integer
data.loc[:,'claps']=data.loc[:,'claps'].replace(np.array(old_value),np.array(new)).astype(int)
```

## Missing Values

**Identifying Missing Value**

• Missing Value Count and % of missing values out of Total # of Observation

```
total = numeric_features.isnull().sum().sort_values(ascending=False)
percent =
(numeric_features.isnull().sum()/numeric_features.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1,join='outer', keys=['Total Missing Count', '% of Total Observations'])
missing_data.index.name =' Numeric Feature'
```

```
for col in df_train.columns:
    msg = 'column: {:>10}\t Percent of NaN value: {:.2f}%'.format(col, 100 *
(df_train[col].isnull().sum() / df_train[col].shape[0]))
    print(msg)
```

```
# MSNO Library

msno.matrix(df=df_train.iloc[:, :], figsize=(8, 8), color=(0.8, 0.5, 0.2))
msno.bar(df=df_train.iloc[:, :], figsize=(8, 8), color=(0.8, 0.5, 0.2))
```

```
def missing_data(data):
    total = data.isnull().sum()
    percent = (data.isnull().sum()/data.isnull().count()*100)
    tt = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    types = []
    for col in data.columns:
        dtype = str(data[col].dtype)
```

```
            types.append(dtype)
        tt['Types'] = types
    return(np.transpose(tt))

missing_data(train_df)
```

```
# Youhan 님
for col in df_train.columns:
    msg = 'column: {:>10}\t Percent of NaN value: {:.2f}%'.\
    format(col, 100 * (df_train[col].isnull().sum() / df_train[col].shape[0]))
    print(msg)
```

```
# From Home Credit Default Risk Competition
# Function to calculate missing values by column

def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns
```

**Handling Missing Values**

• Deletion : When data is missing completely at random (list wise, pairwise)

• Filling in with Mean/Mode/Median

```
df['LoanAmount'] = df.groupby(['Gender','Education','Married','Dependents','Self_Employed']).\
transform(lambda x: x.fillna(x.mean()))['LoanAmount']

from sklearn.impute import SimpleImputer
# Imputing with the mean or mode
mean_imp = SimpleImputer (missing_values=-1 or nan, strategy='mean', axis=0)
mode_imp = SimpleImputer (missing_values=-1 or nan, strategy='most_frequent', axis=0)
train['ps_reg_03'] = mean_imp.fit_transform(train[['ps_reg_03']]).ravel( )
train['ps_car_12'] = mean_imp.fit_transform(train[['ps_car_12']]).ravel( )
train['ps_car_14'] = mean_imp.fit_transform(train[['ps_car_14']]).ravel( )
train['ps_car_11'] = mode_imp.fit_transform(train[['ps_car_11']]).ravel( )
```

• various fill-ins

# normal fillna || df[' '] = df['  '].fillna( )

# fillna with groupby || df['Scheduled Hours'] = df.groupby('EEOC Job Classification')['Scheduled Hours'].bfill().ffill()

# fillna with the most frequent value of that variable

df['Scheduled Hours'] = df['Scheduled Hours'].fillna(df['Scheduled Hours'].value_counts().index[0])

• Filling in null values based on some condition of other columns

df['check'] = np.where(df['Position Effective Date']!=df['Hire Date'], 'Yes', 'No')
df.loc[df['Promotion'].isnull(),'Promotion'] = df['check']

• Prediction Model: Use data without missing values as training set and rows with missing values as test set

• KNN Imputation

• Using Domain Knowledge to fillna

train['Item_Weight']=\
train.groupby(['Item_Identifier'])['Item_Weight'].ffill().bfill()

• '_x'로 끝나는 column들에서 null value 인 것들을 같은 이름이지만 '_y'로 끝나는 column의 value로 채우기

```
for col in df.columns[df.columns.str.endswith('_x')].tolist():
    df.loc[df[col].isnull(),col] = df[col[:-2]+"_y"]
```

## Outlier Detection & Handling

• Detecting Outliers

Boxplot ,Histogram, Scatter plot
Beyond the range of -1.5 x IQR to 1.5 x IQR
Capping method: Any value which out of range of 5th and 95th percentile can be considered as outlier
Data points three or more standard deviation away from mean are considered outlier
Distance: Mahalanobis' distance, Cook's D

**Handling Outliers**

| |
|---|
| Delete: Data entry error, data processing error or outlier observations are very small in numbers |
| Binning<br>data['Global Rank Bin'] = pd.cut(data['Global Rank'], 20, include_lowest =True) # creating bins |
| Transformation of Variables<br><br>Log Transformation<br>df_train['Fare'] = df_train['Fare'].map(lambda i: np.log(i) if i > 0 else 0) |
| Treat separately: If number of outliers is significant. Treat them as two different groups, build individual model and combine output |
| |

## OTHER / MISC

**Explode: Create one row for each item in a list**

```
print("Using explode to generate new rows for each player.")
df1 = df.explode("Players")
```

| | Team | Players |
|---|---|---|
| 0 | FC Barcelona | [Ter Stegen, Semedo, Piqué, Lenglet, Alba, Rak... |
| 1 | FC Real Madrid | [Courtois, Carvajal, Varane, Sergio Ramos, Men... |

```
df1
print("Reverse this operation with groupby and agg")
df["Imploded"] = df1.groupby(df1.index)["Players"].agg(list)
df
```

**Original df**

**When you want to find unusual values in a column**
df.name of column.value_counts().sort_index(ascending=False))
df.describe( )

**Binning (transforming numerical feature to categorical feature)**
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20, 70, num = 11))
pd.qcut

**Number of unique classes /values in each object / cat features**
# Number of unique classes in each object column
app_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)

**Remove features in the training data that are not in the testing data (often because one hot encoding creates more features in the trainin data for some categorical variables with categories not represented in the testing data)**
train_labels = app_train['TARGET']

# Align the training and testing data, keep only columns present in both dataframes
app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)

# Add the target back in
app_train['TARGET'] = train_labels

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)

**Count the number of words (seprarated by " ") in each row of that text column**

```
df["Words"] = df["Title"].str.count(" ") + 1
```

**Split String column into multiple columns**
df[["first", "middle", "last"]] = df["name"].str.split(" ", expand = True)
df["city"] = df["location"].str.split(",", expand = True)[0]

| | name | location |
|---|---|---|
| 0 | John Artur Doe | Los Angeles, CA |
| 1 | Jane Ann Smith | Washington, DC |
| 2 | Nico P | Barcelona, Spain |

9]:

| | name | location | first | middle | last | city |
|---|---|---|---|---|---|---|
| 0 | John Artur Doe | Los Angeles, CA | John | Artur | Doe | Los Angeles |
| 1 | Jane Ann Smith | Washington, DC | Jane | Ann | Smith | Washington |
| 2 | Nico P | Barcelona, Spain | Nico | P | None | Barcelona |

**Cumsum**
df["running_total"] = df["item"].cumsum()
df["running_total_by_person"] = df.groupby("salesperson")["item"].cumsum( )

**Sorting with "Sorted"**
Let's say you have some data that looks like new_corrs = [ ('ang', 3), ('hang, 5.3) ….. ]
new_corrs = sorted(new_corrs, key = lambda x: abs(x[1]), reverse = True)

**Other Useful HotKeys / Options / Styling**
Pd.assign(new column name = lambda x: x['A'] * 100 / sum(x['A']))
pd.set_option("display.max_rows",5)
pd.set_option("display.max_columns",3)
pd.set_option('display.width', 1000)

```
pd.set_option('display.date_dayfirst', True)
pd.reset_option('^display.', silent=True) # restore to default

# add some more formattin
(df.style.format(fd)
  .hide_index()
  .highlight_min("sales", color ="red")
  .highlight_max("sales", color ="green")
  .background_gradient(subset = "sales_100", cmap ="Blues")
  .bar("customers", color = "lightblue", align = "zero")
  .set_caption("A df with different stylings")
)
```

| time | sales | customers | sales_100 |
|---|---|---|---|
| 01/01/00 | $6.00 | 11 | 600 |
| 01/01/00 | $3.00 | 18 | 300 |
| 01/01/00 | $5.00 | 12 | 500 |
| 01/01/00 | $1.00 | 17 | 100 |
| 01/01/00 | $11.00 | 11 | 1100 |
| 01/01/00 | $9.00 | 11 | 900 |
| 01/01/00 | $16.00 | 17 | 1600 |
| 01/01/00 | $4.00 | 13 | 400 |
| 01/01/00 | $11.00 | 15 | 1100 |
| 01/01/00 | $13.00 | 1 | 1300 |

**Transformation of Variables**

• Log Transformation

```
df_train['Fare'] = df_train['Fare'].map(lambda i: np.log(i) if i > 0 else 0)
```

**Looking at Target (y) Label**

```
f, ax = plt.subplots(1,2, figsize=(18,8))

df_train['Survived'].value_counts().plot.pie(explode=[0,0.1],
                                             autopct='%1.1f%%', ax=ax[0],
                                             shadow=True)
ax[0].set_title('Pie plot - Survived')
ax[0].set_ylabel('')
sns.countplot('Survived', data=df_train, ax=ax[1])
ax[1].set_title('Count plot - Survived')
```

**Creating Dummy Datasets**

**# Time Series Dummy**

```
# Solution 1
number_or_rows = 365*24 # hours in a year
pd.util.testing.makeTimeDataFrame(number_or_rows, freq="H")
```

| | A | B | C | D |
|---|---|---|---|---|
| **2000-01-01 00:00:00** | 0.548268 | -1.810959 | -0.197603 | -0.223416 |
| **2000-01-01 01:00:00** | -0.514501 | 0.407318 | -0.108549 | 1.384783 |
| **2000-01-01 02:00:00** | -0.430572 | -0.232883 | 1.261089 | -0.042892 |
| **2000-01-01 03:00:00** | -0.538359 | -1.182248 | -1.041456 | -0.721104 |
| **2000-01-01 04:00:00** | 0.610743 | 1.854269 | 0.802882 | 1.192621 |

```
# Solution 2
num_cols = 2
cols = ["sales", "customers"]
df = pd.DataFrame(np.random.randint(1, 20, size = (number_or_rows, num_cols)), columns=cols)

df.index = pd.util.testing.makeDateIndex(number_or_rows, freq="H")
df
```

| | sales | customers |
|---|---|---|
| **2000-01-01 00:00:00** | 8 | 13 |
| **2000-01-01 01:00:00** | 14 | 10 |
| **2000-01-01 02:00:00** | 13 | 19 |
| **2000-01-01 03:00:00** | 3 | 5 |
| **2000-01-01 04:00:00** | 13 | 19 |
| **...** | ... | ... |

```
print("Contains random values")
df1 = pd.util.testing.makeDataFrame() # contains random values
df1
print("Contains missing values")
df2 = pd.util.testing.makeMissingDataframe() # contains missing values
df2
print("Contains datetime values")
df3 = pd.util.testing.makeTimeDataFrame() # contains datetime values
```

```
df3
print("Contains mixed values")
df4 = pd.util.testing.makeMixedDataFrame() # contains mixed values
df4
```