Assignment 1

Learning Goals

After successfully completing this assignment, you should be able to:

- represent and manipulate numeric data using `numpy` (https://numpy.org/doc/stable/reference/generated/numpy.array.html) arrays
- represent and manipulate data using `pandas` (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) dataframes
- load data from a .csv file

In [2]:

```
import numpy as np                    # import numpy package under shorthand "np"
import pandas as pd                   # import pandas package under shorthand "pd"
import matplotlib.pyplot as plt
from nose.tools import assert_equal
from numpy.testing import assert_array_equal
```

Representing Data as Numpy Arrays

Data consists of many individual datapoints. Each datapoint is characterized by features and labels. Let us assume that the features of a datapoint is a finite list of numbers $x_1, \ldots, x_n \in \mathbb{R}$. We can represent such a finite list of numbers conveniently using a numeric or `numpy` (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html?highlight=ndarray#numpy.ndarray) array.

For instance, we could have a feature vector `np.array([60.1699, 24.9384])` representing the coordinates of Helsinki, and a label `np.array([5.0])` representing average yearly temperature. In general, the feature vector $\mathbf{x} = \left(x_1, \ldots, x_n\right)^T \in \mathbb{R}^n$ can be represented by a 1 dimensional `numpy` array `x = np.array([x1,...,xn])` .

This course will use mainly `numpy` arrays with 1 or 2 dimensions, representing vectors and matrices, respectively. We represent an $m \times n$ matrix, i.e., with m rows and n columns, using a 2 dimensional `numpy` array with shape `(m,n)` . The Python code `A=np.array([[1,1,1],[2,2,2]])` creates a `numpy` array of shape `(2, 3)` , representing the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$

One key attribute of an `numpy.ndarray` (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html?highlight=ndarray#numpy.ndarray) object `x` is its shape, which is stored in the attribute `x.shape` . The shape `x.shape` is a tuple of integers $s_0, \ldots, s_{d-1}$ that indicates the extent (size/length) of the `numeric` array in different dimensions. The first entry s_0 of the list `x.shape` is the extent in the first dimension (dimension "0"), the second entry s_1 is the extent of `x` in the second dimension (dimension "1"). Note that `np.array([1,2,3]).shape` returns `(3,)` which represents a vector, but `np.array([[1,2,3]]).shape` returns `(1, 3)` which represents a matrix.

For more information:

- [Numpy Documentation (https://numpy.org/doc/stable/index.html)](https://numpy.org/doc/stable/index.html)
- [What is Numpy? (https://numpy.org/doc/stable/user/whatisnumpy.html)](https://numpy.org/doc/stable/user/whatisnumpy.html)
- [Numpy Basics (https://numpy.org/doc/stable/user/basics.html)](https://numpy.org/doc/stable/user/basics.html)
- [Visualization (https://stackoverflow.com/questions/48200911/very-basic-numpy-array-dimension-visualization)](https://stackoverflow.com/questions/48200911/very-basic-numpy-array-dimension-visualization)

In the task below you will be asked to do some simple operations in numpy that will be necessary to know for the duration of the course.

Student Task A1.1

- Create a `numpy` array `x` that represents the vector $\mathbf{x} = \big(1, 2, 3\big)^T$ and another `numpy` array `y` that represents the vector $\mathbf{y} = \big(2, 3, 4\big)^T$.
- Complete the function `sum_matrix` which should read in two `numpy` arrays of the same shape. The function should return a `numpy` array with the same shape of the inputs and whose entries are sums of the corresponding entries in the two input arrays.
- Similar to `sum_matrix`, complete the function `product_matrix` that returns a `numpy` array whose entries are products of the entries of the input `numpy` arrays.

In [17]:

```
## create numpy arrays as:
# x = np.array(...)  # input: a list
# y = np.array(...)  # input: a list

# YOUR CODE HERE
x = np.array([[1, 2, 3]]).T  # input: a list
y = np.array([[2, 3, 4]]).T  # input: a list


def sum_matrix(x,y):
    """
    Parameters:
    x -- a numpy array
    y -- a numpy array

    Returns:
    a numpy array representing the element-wise sum of x and y
    """
    # YOUR CODE HERE
    return x+y

def product_matrix(x,y):
    """
    Parameters:
    x -- a numpy array
    y -- a numpy array

    Returns:
    a numpy array representing the element-wise product of x and y
    """
    # YOUR CODE HERE
    return x*y
```

In [18]:

```
# this cell is for tests, please leave it as it is
```

Student Task A1.2

Create a `numpy` array `A` of shape `(2,3)` that represents the 2×3 matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Complete the three functions in the code snippet:

- `first_row` that should return a 1-D `numpy` array that represents the first row of the matrix corresponding to the input array.
- `second_column` that should return 1-D `numpy` array ... second column ...
- `second_row_and_column` that should return a single number which is contained in the second row and second column of the input matrix.

In [30]:

```
## create a 2-D numpy array as:
# A = ...

# YOUR CODE HERE
A = np.array([[1,2,3],[4,5,6]])

## sanity check to avoid major mistakes
assert isinstance(A,np.ndarray)  # check the type of A
assert A.shape==(2,3)  # check the shape of A
```

In [31]:

```
def first_row(A):
    '''
    Parameter:
    A -- a numpy array

    Returns:
    the first row of A
    '''
    # YOUR CODE HERE
    return A[0]

def second_column(A):
    '''
    Parameter:
    A -- a numpy array

    Returns:
    the second column of A
    '''
    # YOUR CODE HERE
    return A[:,1]

def second_row_and_column(A):
    '''
    Parameter:
    A -- a numpy array

    Returns:
    the second row and second column of A, a float
    '''
    # YOUR CODE HERE
    return A[1,1]
```

In [ ]:

```
## this cell is for tests, please leave it as it is
```

Student Task A1.3

- Create a `numpy` array `B` of shape `(2,2)` that represents the matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

- Create a `numpy` array `C` of shape `(2,2)` that represents the matrix

$$\begin{pmatrix} 5 & 6 \\ 6 & 7 \end{pmatrix}$$

- Complete the function `matrix_mult` which reads in `numpy` arrays `B` and `C` and returns a `numpy` array that represents the [matrix multiplication (https://en.wikipedia.org/wiki/Matrix_multiplication)](https://en.wikipedia.org/wiki/Matrix_multiplication) of the matrices represented by `B` and `C`.

In [34]:

```
## create two numpy arrays as:
# B = ...
# C = ...

# YOUR CODE HERE
B = np.array([[1,2],[3,4]])
C = np.array([[5,6],[6,7]])

def matrix_mult(B,C):
    '''
    Parameters:
    B -- a numpy array
    C -- a numpy array

    Returns:
    the result of matrix multiplying of B and C
    '''
    # YOUR CODE HERE
    return B.dot(C)
```

In [ ]:

```
# this cell is for tests
```

Student Task A1.4

Consider the code line `A=np.array([[1,0,0],[0,0,1]])` which creates a `numpy` array `A`. What is the shape of the `numpy` array? (Set the variable `Answer` to the index of the correct answer)

- answer 1: the shape is `(2,1)`.
- answer 2: the shape is `(2,3)`.

In [37]:

```
## set Answer to the index of the correct answer (e.g., Answer = 1 if you think answ
# Answer = ... # either 1 or 2

# YOUR CODE HERE
Answer = 2

assert isinstance(Answer,int) # sanity check the datatype
```

In [ ]:

```
# this cell is for tests
```

Demo

A frequently used method of ndarray is [ndarray.reshape() (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.reshape.html)](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.reshape.html), it returns a new shape to an array without changing its data. The mandatory parameter of `.reshape()` is 'shape' which should be the new shape represented by an int or a tuple of ints, it should be compatible with the original shape. If an integer, then the result will be a 1-D array of that length. One shape dimension can be -1, in this case, the value is inferred from the length of the array and remaining dimensions.

In [38]:

```
# create a 1-D numpy array
P = np.array([1,2,3,4,5,6,7,8])
print('P:\n',P)
print('The shape of P is: ',P.shape)

# reshape P to a 2-D array, the size of the second dimension is 1, the first dimensi
P_1 = P.reshape((-1,1))
print('\nP_1:\n',P_1)
print('The shape of P_1 is: ',P_1.shape)

# reshape P to a 2-D array with the shape (2,4)
P_2 = P.reshape((2,4))
print('\nP_2:\n',P_2)
print('The shape of P_2 is: ',P_2.shape)
```

```
P:
 [1 2 3 4 5 6 7 8]
The shape of P is:  (8,)

P_1:
 [[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]]
The shape of P_1 is:  (8, 1)

P_2:
 [[1 2 3 4]
 [5 6 7 8]]
The shape of P_2 is:  (2, 4)
```

Student Task A1.5

Your task is to reshape P to a new ndarray P_test with the shape (4,2)

In [39]:

```
## apply the method P.rehshape() to create a new ndarray P_test with the shape (4,2)
# P_test = P.reshape(...) # input: a tuple

# YOUR CODE HERE
P_test = P.reshape((4,2))

print('P_test:\n',P_test)
print('\nThe shape of P_test is: ', P_test.shape)
```

```
P_test:
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]

The shape of P_test is:  (4, 2)
```

In [ ]:

```
# this cell is for tests
```

Processing Data with Pandas

[Pandas (https://pandas.pydata.org/docs/)](https://pandas.pydata.org/docs/) provides the class `DataFrame` . A dataframe is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can think of `DataFrame` a table whose rows represent individual datapoints and whose columns represent different properties (which might be features or labels) of the datapoints.

We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming. You can find some example use cases for `DataFrame` below. For more practice with `pandas` there is great documentation in the [10 minutes to Pandas (https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html)](https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html) notebooks on their website.

In what follows, we will demonstrate the usage of a `DataFrame` on data provided by the Finnish Meteorological Institute (FMI) at [https://en.ilmatieteenlaitos.fi/download-observations (https://en.ilmatieteenlaitos.fi/download-observations)](https://en.ilmatieteenlaitos.fi/download-observations). We have downloaded hourly weather observations at the FMI station Otsjoki Nuorgam during 01.06.2021 and 31.08.2021. The data is stored in the file `air_temp.csv` which is located in the same directory as this notebook.

In [40]:

```
# read in data from the file "air_temp.csv" and store it
# in the DataFrame "df"

df = pd.read_csv('air_temp.csv')

# print the first 5 weather recordings in the DataFrame `df`

df.head(5)
```

Out[40]:

	Year	m	d	Time	Time zone	Air temperature (degC)
0	2021	6	1	00:00	UTC	6.2
1	2021	6	1	01:00	UTC	6.4
2	2021	6	1	02:00	UTC	6.4
3	2021	6	1	03:00	UTC	6.8
4	2021	6	1	04:00	UTC	7.1

In [41]:

```
# print a concise summary of a DataFrame including the index dtype and columns, non-

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2209 entries, 0 to 2208
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Year                    2209 non-null   int64  
 1   m                       2209 non-null   int64  
 2   d                       2209 non-null   int64  
 3   Time                    2209 non-null   object 
 4   Time zone               2209 non-null   object 
 5   Air temperature (degC)  2204 non-null   float64
dtypes: float64(1), int64(3), object(2)
memory usage: 103.7+ KB
```

In [42]:

```
# change column names

df.columns=['year','month', 'day', 'time', 'time_zone','temperature']

# remove rows from dataframe "df" which contain missing values

df = df.dropna(axis=0) #rows are considered as axis 0

# concatenate the 3 columns "year", "month", "day" into a new column "date" in forma

data = df.assign(date = df["year"].astype(str)+'-'+df["month"].astype(str)+'-'+df["c

# remove columns "year", "month", "day", "time_zone" that are not used

data = data.drop(['year','month','day','time_zone'],axis=1)  #columns are axis 1

# switch column order

data = data[['date','time','temperature']]

# print the last 5 weather recordings of the new dataframe

data.tail(5)
```

Out[42]:

	date	time	temperature
2204	2021-8-31	20:00	6.1
2205	2021-8-31	21:00	6.4
2206	2021-8-31	22:00	6.2
2207	2021-8-31	23:00	5.6
2208	2021-9-1	00:00	5.5

In [43]:

```
# print some summary statistics of the rows in "data", such as mean, std, etc.

df.describe()
```

Out[43]:

	year	month	day	temperature
count	2204.0	2204.000000	2204.000000	2204.000000
mean	2021.0	7.014065	15.837114	12.049365
std	0.0	0.814798	8.863532	5.021324
min	2021.0	6.000000	1.000000	0.500000
25%	2021.0	6.000000	8.000000	8.600000
50%	2021.0	7.000000	16.000000	11.150000
75%	2021.0	8.000000	24.000000	14.700000
max	2021.0	9.000000	31.000000	32.200000

Sometimes we are interested in a specific column of a DataFrame, e.g. we want to use it as feature/label. we can select a single column using `dataframe['column_name']` , this will return a [Series (https://pandas.pydata.org/docs/reference/api/pandas.Series.to_numpy.html#pandas.Series.to_numpy)](https://pandas.pydata.org/docs/reference/api/pandas.Series.to_numpy.html#pandas.Series.to_numpy) object. Series object has a method `Series.to_numpy()` which will give us a NumPy ndarray representing the values in this Series. You will repeatedly use this method through this course.

In [44]:

```
# Let us select only the column "temperature" of the DataFrame "data"
tmp = data['temperature']

print(type(tmp),'\n') # check the type of this object
print(tmp)
```

```
<class 'pandas.core.series.Series'>

0       6.2
1       6.4
2       6.4
3       6.8
4       7.1
       ...
2204    6.1
2205    6.4
2206    6.2
2207    5.6
2208    5.5
Name: temperature, Length: 2204, dtype: float64
```

In [45]:

```
data['temperature'].to_numpy() # extract the values stored in a specific column into
```

Out[45]:

```
array([6.2, 6.4, 6.4, ..., 6.2, 5.6, 5.5])
```

We can select a single row by using dataframe.loc[ ] or dataframe.iloc[ ], the returned object is also Series. dataframe.iloc[ ] and dataframe.loc[ ] in fact can be used in multiple ways to do dataframe slicing, for more details please read [the documentation (https://pandas.pydata.org/docs/reference/frame.html)](https://pandas.pydata.org/docs/reference/frame.html).

In [46]:

```
# select the first weather recording (row) stored in the DataFrame "data"
firstrow = data.iloc[0]  # `0` is the index of the first row

print("The first row: \n",firstrow)

# select the row with row label name `3` by using data.loc[ ]
# NOTE `3` is interpreted as a row label name , not an integer position along the in
# the row label name could be string or other data type, not only int
rowName3 = data.loc[3]
print("\n The row with row label name '3': \n",rowName3)
```

```
The first row: 
 date           2021-6-1
time              00:00
temperature         6.2
Name: 0, dtype: object

 The row with row label name '3': 
 date           2021-6-1
time              03:00
temperature         6.8
Name: 3, dtype: object
```

In [47]:

```
# we can select a subset of a DataFrame on some condition and create a new DataFrame

# create a "newdataset" which consists only of weather recordings in "data" at "time
newdataset= data[data['time'] == '03:00'] ;

# print randomly selected five weather recordings (rows) of "newdataset"
newdataset.sample(5)
```

Out[47]:

	date	time	temperature
411	2021-6-18	03:00	9.3
627	2021-6-27	03:00	8.8
171	2021-6-8	03:00	10.9
2163	2021-8-30	03:00	6.3
2067	2021-8-26	03:00	2.3

Preparing Features and Labels from DataFrame

Demo

Consider the weather observations recorded in `air_temp.csv` and loaded into the dataframe `data` . Let us now demonstrate how to define datapoints, their features and labels based on these weather observations. It is important to note that the choice (definition) of datapoints, their features and labels are design choices.

We like to define a datapoint to represent an entire day, e.g.,

- first datapoint represents the day `2021-06-01` ,
- second datapoint represents the day `2021-06-02` ,
- third datapoint represents the day `2021-06-03` ,
- ...

The total number m of datapoints is the number of days for which `data` contains weather recordings for daytime `01:00` and `12:00` .

We characterize the i-th datapoint (day) using

- the temperature recorded at 01:00 during the ith day as its feature $x^{(i)}$
- the temperature recorded at 12:00 during the ith day as its label $y^{(i)}$

We store the feature values $x^{(i)}, i = 1, \ldots, m$ in a (two-dimensional) numpy array `X_demo` with shape `(m,1)` . The feature value $x^{(i)}$ is stored in the entry `X_demo[i-1,0]` (note that indexing of numpy arrays starts with 0!). The label values $y^{(i)}, i = 1, \ldots, m$ are stored in a (one-dimensional) numpy array `y_demo` with shape `(m,)` . Finally, we generate a scatterplot where the ith datapoint is depicted by a dot located at coordinates $\left(x^{(i)}, y^{(i)}\right)$.

HINT: Reshape `X_demo` into a 2-D array by using array.reshape(-1, 1). This asks numpy to make the second dimension length one and automatically calculate the needed length of the first dimension so that the feature fits in the container which expects a 2-D array. (e.g.,the .fit() method of [LinearRegression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearR](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearR)

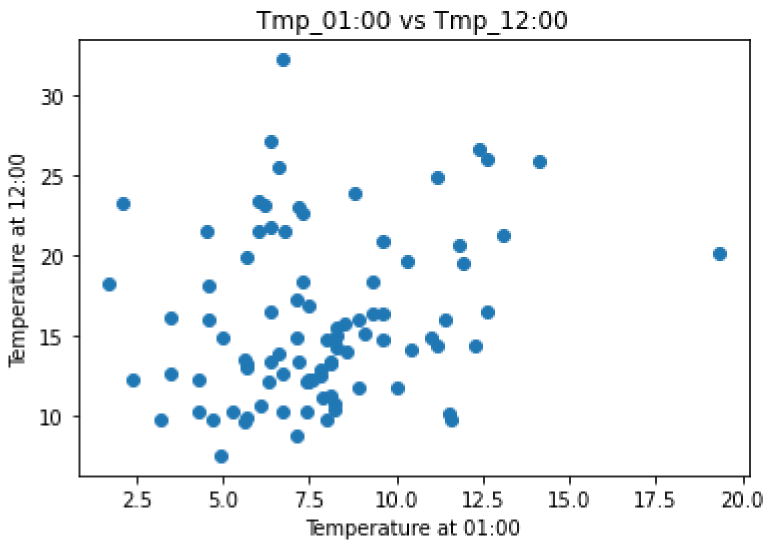
In [48]:

```
# create a list containing the dates for which at least one recording is contained i
dates = data['date'].unique()

features = []   # list used for storing features of datapoints
labels = []     # list used for storing labels of datapoints

m = 0    # number of datapoints created so far

# iterate through the list of dates for which we have weather recordings
for date in dates:
    datapoint = data[(data['date']==date)]  # select weather recordinds correspondi
    row_f = datapoint[(datapoint.time=='01:00')]    # select weather recording at ti
    row_l = datapoint[(datapoint.time=='12:00')]    # select weather recording at ti
    if len(row_f)==1 and len(row_l)==1:
        feature = row_f['temperature'].to_numpy()[0]  # extract the temperature reco
        label = row_l['temperature'].to_numpy()[0]    # extract the temperature reco
        features.append(feature)                     # add feature to list "features"
        labels.append(label)                         # add label to list "labels"
        m = m+1

X_demo = np.array(features).reshape(m,1)  # convert a list of len=m to a ndarray anc
y_demo = np.array(labels) # convert a list of len=m to a ndarray

print("number of datapoints:",m)
print("the shape of the feature matrix is: ",X_demo.shape)
print('the shape of the label vector is: ',y_demo.shape)
```

```
number of datapoints: 92
the shape of the feature matrix is:  (92, 1)
the shape of the label vector is:  (92,)
```

In [49]:

```
# visualize the datapoints
fig = plt.figure()  #create a figure

ax = fig.add_subplot(1, 1, 1) #add an axes object to the figure

ax.scatter(X_demo,y_demo) #plot a scatterplot in the axes to visualize the datapoint
ax.set_xlabel('Temperature at 01:00') # set the label of x axis
ax.set_ylabel('Temperature at 12:00') #
ax.set_title('Tmp_01:00 vs Tmp_12:00')

plt.show()

# one line of code `plt.scatter(X_demo,y_demo)` without creating figure and axes obj
# can also realize a scatter plot, but it's worth getting yourself familiar with the
```



Student Task A1.6

Consider the weather observations recorded in `air_temp.csv` and loaded into the dataframe `data` . We define a datapoint to represent an entire day,

- First datapoint represents the day `2021-06-01` ,
- Second datapoint represents the day `2021-06-02` ,
- Third datapoint represents the day `2021-06-03` ,
- ...

The total number m of datapoints is the number of days for which `data` contains weather recordings for daytime `11:00` and `12:00` .

We characterize the i-th datapoint (day) using

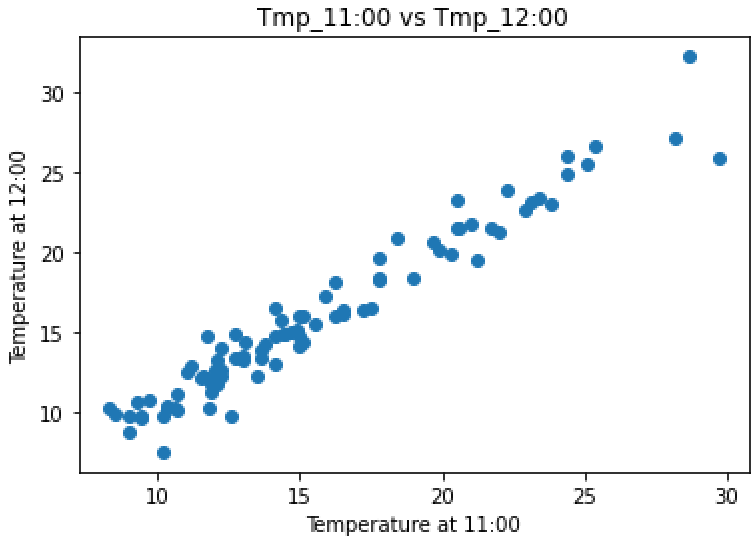
- The temperature recorded at 11:00 during the ith day as its feature $x^{(i)}$
- The temperature recorded at 12:00 during the ith day as its label $y^{(i)}$

Store the feature values in a numpy array `X` of shape `(m,1)` and the label values in a numpy array `y` of shape `(m,)` .

In [51]:

```
## Generate your own datapoints with feature X = “tmp at 11:00” and label y = “tmp a

# YOUR CODE HERE
# create a list containing the dates for which at least one recording is contained i
dates = data['date'].unique()

features = []   # list used for storing features of datapoints
labels = []     # list used for storing labels of datapoints

m = 0    # number of datapoints created so far

# iterate through the list of dates for which we have weather recordings
for date in dates:
    datapoint = data[(data['date']==date)]  # select weather recordinds correspondir
    row_f = datapoint[(datapoint.time=='11:00')]    # select weather recording at ti
    row_l = datapoint[(datapoint.time=='12:00')]    # select weather recording at ti
    if len(row_f)==1 and len(row_l)==1:
        feature = row_f['temperature'].to_numpy()[0]  # extract the temperature recc
        label = row_l['temperature'].to_numpy()[0]    # extract the temperature recc
        features.append(feature)                     # add feature to list "features"
        labels.append(label)                         # add label to list "labels"
        m = m+1

X = np.array(features).reshape(m,1)  # convert a list of len=m to a ndarray and resh
y = np.array(labels) # convert a list of len=m to a ndarray

print("number of datapoints:",m)
print("the shape of the feature matrix is: ",X_demo.shape)
print('the shape of the label vector is: ',y_demo.shape)

# visualize the datapoints
fig = plt.figure()  #create a figure

ax = fig.add_subplot(1, 1, 1) #add an axes object to the figure

ax.scatter(X,y) #plot a scatterplot in the axes to visualize the datapoints
ax.set_xlabel('Temperature at 11:00') # set the label of x axis
ax.set_ylabel('Temperature at 12:00') #
ax.set_title('Tmp_11:00 vs Tmp_12:00')

plt.show()

# one line of code `plt.scatter(X_demo,y_demo)` without creating figure and axes obj
# can also realize a scatter plot, but it's worth getting yourself familiar with the

# Sanity check to help you detect major mistakes
assert np.isclose(X[0,0],14.1), 'Feature matrix is incorrect'
assert X.shape == (91,1), 'The shape of feature matrix is incorrect'
assert y.shape == (91,), 'The shape of label vector is incorrect'
```

```
number of datapoints: 91
the shape of the feature matrix is:  (92, 1)
the shape of the label vector is:  (92,)
```



In [ ]:

```
#this cell is for tests, please leave it as it is.
```