5 I DATA MANIPULATION

Jay Klopper MD MMed(Surgery)Cum Laude

The George Washington University

Script for importing data from the following spreadsheet:

```
Workbook: /Users/juanheinklopper/Documents/MATLAB/MATLAB for Data Science/Data/heart.xlsx
Worksheet: heart
```

Auto-generated by MATLAB on 12-Jun-2024 13:49:58

Table of Contents

Set up the Import Options and import the data

```
clear global
opts = spreadsheetImportOptions("NumVariables", 12);

% Specify sheet and range
opts.Sheet = "heart";
opts.DataRange = "A2:L919";

% Specify column names and types
opts.VariableNames = ["Age", "Sex", "ChestPainType", "RestingBP",
"Cholesterol", "FastingBS", "RestingECG", "MaxHR", "ExerciseAngina",
"Oldpeak", "ST_Slope", "HeartDisease"];
opts.VariableTypes = ["double", "categorical", "categorical", "double",
"double", "double", "categorical", "double", "categorical", "double",
"categorical", "categorical"];

% Specify variable properties
opts = setvaropts(opts, ["Sex", "ChestPainType", "RestingECG",
"ExerciseAngina", "ST_Slope", "HeartDisease"], "EmptyFieldRule", "auto");

% Import the data
heart = readtable("/Users/juanheinklopper/Documents/MATLAB/MATLAB for Data
Science/Data/heart.xlsx", opts, "UseExcel", false)
```

```
heart = 918×12 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	49	F	NAP	160	180	0
3	37	M	ATA	130	283	0
4	48	F	ASY	138	214	0
5	54	M	NAP	150	195	0
6	39	M	NAP	120	339	0
7	45	F	ATA	130	237	0
8	54	M	ATA	110	208	0
9	37	M	ASY	140	207	0
10	48	F	ATA	120	284	0
11	37	F	NAP	130	211	0
12	58	M	ATA	136	164	0
13	39	M	ATA	120	204	0
14	49	M	ASY	140	234	0

⋮

```
clear opts
```

Introduction

As in the previous chapter, we used the built-in MATLAB functionality to import a spreadsheet file.

In this chapter we continue our exploration of exploratory data analysis by manipulating the data that we import.

Using indexing to select data

Filtering for only parts of our data for analysis is a very common task in data science. We explored indexing of vectors and matrices in chapter 2. We can apply the same principles to `table` objects. Below, we filter only for the first row (all columns) by using indexing. The colon symbol `:` is shorthand for indicating that we want all (in this cases all the columns). Remember that indexing is in the form *row, column*.

```
% Show the data for the first subject
heart(1,:)
```

```
ans = 1×12 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0

The head function is used to display the first n rows of data. Below, we view the first 5 rows.

```
% Show the data for the first 5 subjects
head(heart,5)
```

```
    Age    Sex    ChestPainType    RestingBP    Cholesterol    FastingBS    RestingECG    MaxHR    Exercis
    ___    ___    _____________    _________    ___________    _________    __________    _____    _______

    40      M          ATA            140           289            0          Normal       172           N
    49      F          NAP            160           180            0          Normal       156           N
    37      M          ATA            130           283            0          ST            98           N
    48      F          ASY            138           214            0          Normal       108           Y
    54      M          NAP            150           195            0          Normal       122           N
```

The `tail` function return the last rows.

```
% Show data for last 5 subjects
tail(heart,5)
```

```
    Age    Sex    ChestPainType    RestingBP    Cholesterol    FastingBS    RestingECG    MaxHR    Exercis
    ___    ___    _____________    _________    ___________    _________    __________    _____    _______

    45      M          TA             110           264            0          Normal       132           N
    68      M          ASY            144           193            1          Normal       141           N
    57      M          ASY            130           131            0          Normal       115           Y
    57      F          ATA            130           236            0          LVH          174           N
    38      M          NAP            138           175            0          Normal       173           N
```

Next we filter for rows 1 and 3 and all the columns.

```
% Show rows 1 and 3
heart([1 3],:)
```

ans = 2×12 table

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	37	M	ATA	130	283	0

We can use the names of the columns instead of their indices. The names of the columns are passed as a cell array.

```
% Select only the Age and Sex variables
heart(:,{'Age','Sex'})
```

ans = 918×2 table

	Age	Sex
1	40	M
2	49	F
3	37	M
4	48	F

	Age	Sex
5	54	M
6	39	M
7	45	F
8	54	M
9	37	M
10	48	F
11	37	F
12	58	M
13	39	M
14	49	M

⋮

Data filtering by condition

Other than indexing, we can also filter by condition. A condition returns either a true or a false value. We use this fact to return only observations that return as value of true. Below, we return only observations (with all columns selected) by filtering the `Age` column for values in excess of 50.

```
% Filter only those older than 50
heart(heart.Age > 50,:)
```

```
ans = 602×12 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	54	M	NAP	150	195	0
2	54	M	ATA	110	208	0
3	58	M	ATA	136	164	0
4	54	F	ATA	120	273	0
5	60	M	ASY	100	248	0
6	53	M	ASY	124	260	0
7	52	M	ATA	120	284	0
8	53	F	ATA	113	468	0
9	51	M	ATA	125	188	0
10	53	M	NAP	145	518	0
11	56	M	NAP	130	167	0
12	54	M	ASY	125	224	0
13	65	M	ASY	140	306	1

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
14	54	F	ATA	150	230	0

⋮

Next we filter only for females (encoded as F in the Sex column).

```
% Filter only female subjects
heart(heart.Sex == "F",:)
```

```
ans = 193×12 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	49	F	NAP	160	180	0
2	48	F	ASY	138	214	0
3	45	F	ATA	130	237	0
4	48	F	ATA	120	284	0
5	37	F	NAP	130	211	0
6	42	F	NAP	115	211	0
7	54	F	ATA	120	273	0
8	43	F	ATA	120	201	0
9	43	F	TA	100	223	0
10	49	F	ATA	124	201	0
11	53	F	ATA	113	468	0
12	43	F	ATA	150	186	0
13	41	F	ATA	110	250	0
14	48	F	ATA	120	177	1

⋮

Filtering on more than one condition is achieved using logical and. The symbol for this is &. Below we filter for all subjects older than 50 who are female. Note that both conditions must be met.

```
% Filter females older than 50
heart(heart.Age > 50 & heart.Sex == "F",:)
```

```
ans = 115×12 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	54	F	ATA	120	273	0
2	53	F	ATA	113	468	0
3	54	F	ATA	150	230	0

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
4	54	F	NAP	130	294	0
5	52	F	ASY	130	180	0
6	51	F	ATA	160	194	0
7	53	F	ATA	140	216	0
8	52	F	ATA	120	210	0
9	59	F	ATA	130	188	0
10	59	F	ASY	130	338	1
11	52	F	NAP	125	272	0
12	58	F	ATA	180	393	0
13	54	F	ATA	120	230	1
14	61	F	ASY	130	294	0

⋮

Using logical or to filter values whenever any condition is met. Below we filter for subjects older than 50 or who are female, using the | symbol. We need only one of the condition to be true for inclusion in the filtering.

```
% Filter females or those older than 50
heart(heart.Age > 50 | heart.Sex == "F",:)
```

```
ans = 680×12 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	49	F	NAP	160	180	0
2	48	F	ASY	138	214	0
3	54	M	NAP	150	195	0
4	45	F	ATA	130	237	0
5	54	M	ATA	110	208	0
6	48	F	ATA	120	284	0
7	37	F	NAP	130	211	0
8	58	M	ATA	136	164	0
9	42	F	NAP	115	211	0
10	54	F	ATA	120	273	0
11	43	F	ATA	120	201	0
12	60	M	ASY	100	248	0
13	43	F	TA	100	223	0
14	49	F	ATA	124	201	0

⋮

Change variable names

It is often required to rename variables. The `renamevars` function is used to achieve this. Below we change the `RestingECG` column name to the `Resting_ECG`.

```
% Change RestingECG to Resting_ECG
heart = renamevars(heart,["RestingECG"],["Resting_ECG"])
```

```
heart = 918×12 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	49	F	NAP	160	180	0
3	37	M	ATA	130	283	0
4	48	F	ASY	138	214	0
5	54	M	NAP	150	195	0
6	39	M	NAP	120	339	0
7	45	F	ATA	130	237	0
8	54	M	ATA	110	208	0
9	37	M	ASY	140	207	0
10	48	F	ATA	120	284	0
11	37	F	NAP	130	211	0
12	58	M	ATA	136	164	0
13	39	M	ATA	120	204	0
14	49	M	ASY	140	234	0

⋮

Add new column based on values of another

Adding new column is a task that is often required in data science. We can add any values to a new column. It is most useful, though, to base the values on calculations performed on the values in other columns. Below, we convert the units of the `Cholesterol` column to mmol/L. We also use the `round` function to round off the calculated values to two decimal places.

```
% Add a new column named Cholesterol_mmol
heart.Cholesterol_mmol = round(heart.Cholesterol ./ 38.67,2)
```

```
heart = 918×13 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	49	F	NAP	160	180	0
3	37	M	ATA	130	283	0
4	48	F	ASY	138	214	0
5	54	M	NAP	150	195	0
6	39	M	NAP	120	339	0
7	45	F	ATA	130	237	0
8	54	M	ATA	110	208	0
9	37	M	ASY	140	207	0
10	48	F	ATA	120	284	0
11	37	F	NAP	130	211	0
12	58	M	ATA	136	164	0
13	39	M	ATA	120	204	0
14	49	M	ASY	140	234	0

⋮

Change the values of observations in a column based on conditions

The values in a new column can also be created based on conditions applied to the values of another column. Below we add a new column called `HeartDiseaseClass` and assign the values of the `HeartDisease` column to the new column.

```
% Add a new column called HeartDiseaseClass as a copy of the HeartDisease
% variable
heart.HeartDiseaseClass = heart.HeartDisease
```

```
heart = 918×14 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	49	F	NAP	160	180	0
3	37	M	ATA	130	283	0
4	48	F	ASY	138	214	0
5	54	M	NAP	150	195	0
6	39	M	NAP	120	339	0
7	45	F	ATA	130	237	0
8	54	M	ATA	110	208	0

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
9	37	M	ASY	140	207	0
10	48	F	ATA	120	284	0
11	37	F	NAP	130	211	0
12	58	M	ATA	136	164	0
13	39	M	ATA	120	204	0
14	49	M	ASY	140	234	0

⋮

Next, we use the `ismember` function to filter for rows with a 0 in the new `HeartDisease` class and replace them with the value No.

```
% Change the values in the new column to "No" if the value is 0
heart.HeartDiseaseClass(ismember(heart.HeartDiseaseClass, ["0"])) = "No"
```

```
heart = 918×14 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	49	F	NAP	160	180	0
3	37	M	ATA	130	283	0
4	48	F	ASY	138	214	0
5	54	M	NAP	150	195	0
6	39	M	NAP	120	339	0
7	45	F	ATA	130	237	0
8	54	M	ATA	110	208	0
9	37	M	ASY	140	207	0
10	48	F	ATA	120	284	0
11	37	F	NAP	130	211	0
12	58	M	ATA	136	164	0
13	39	M	ATA	120	204	0
14	49	M	ASY	140	234	0

⋮

We do the same for those with heart disease.

```
% Change the values in the new column to "Yes" if the value is 1
heart.HeartDiseaseClass(ismember(heart.HeartDiseaseClass, ["1"])) = "Yes"
```

```
heart = 918×14 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	49	F	NAP	160	180	0
3	37	M	ATA	130	283	0
4	48	F	ASY	138	214	0
5	54	M	NAP	150	195	0
6	39	M	NAP	120	339	0
7	45	F	ATA	130	237	0
8	54	M	ATA	110	208	0
9	37	M	ASY	140	207	0
10	48	F	ATA	120	284	0
11	37	F	NAP	130	211	0
12	58	M	ATA	136	164	0
13	39	M	ATA	120	204	0
14	49	M	ASY	140	234	0

⋮

Change numerical variable to nominal variable

Categorizing a numerical variable is another common task in data science. Below, we create three classes based on the `Age` column by using the `discretize` function and setting the intervals as left-closed right-open intervals.

```
% Add a new column to the data table called AgeClass
% Let observation be "Group A" if Age observation is in interval [0,50)
% Let observation be "Group B" if Age observation is in interval [50,75)
% Let observation be "Group C" if Age observation is in interval [75,100)
heart.AgeClass = discretize(heart.Age, [0 50 75 100], 'categorical',
["Group A", "Group B", "Group C"])
```

```
heart = 918×15 table
```

...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
1	40	M	ATA	140	289	0
2	49	F	NAP	160	180	0
3	37	M	ATA	130	283	0
4	48	F	ASY	138	214	0
5	54	M	NAP	150	195	0

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS
6	39	M	NAP	120	339	0
7	45	F	ATA	130	237	0
8	54	M	ATA	110	208	0
9	37	M	ASY	140	207	0
10	48	F	ATA	120	284	0
11	37	F	NAP	130	211	0
12	58	M	ATA	136	164	0
13	39	M	ATA	120	204	0
14	49	M	ASY	140	234	0

⋮

Working with dates and times

Working with dates and times is essential and sometimes very difficult. Most difficulty stems from the data capture in a spreadsheet of database file. The problems that we face with dates and times can be avoided by proper data capture in a consistent format and not using formatting built into spreadsheet and database programs. In this section, we use MATLAB to generate dates and times.

```
% Add day as a string
datetime('June 18, 2024')
```

```
ans = datetime
   18-Jun-2024
```

```
% Current day and time
dt = datetime("now")
```

```
dt = datetime
   18-Jun-2024 14:07:24
```

```
% Year
year(dt)
```

```
ans = 2024
```

```
dt.Year
```

```
ans = 2024
```

```
% Month number
month(dt)
```

```
ans = 6
```

```
%Month name
month(dt,'name')
```

```
ans = 1×1 cell array
    {'June'}
```

```
% Day
day(dt)
```

```
ans = 18
```

```
% Day of the week
day(dt,'dayofweek')
```

```
ans = 3
```

```
% Long day name
day(dt,'name')
```

```
ans = 1×1 cell array
    {'Tuesday'}
```

```
[y,m,d] = ymd(dt)
```

```
y = 2024
m = 6
d = 18
```

```
years = randi(24,20,1) + 2000
```

```
years = 20×1
        2005
        2021
        2015
        2009
        2006
        2006
        2013
        2011
        2018
        2002
     ⋮
```

```
months = randi(12,20,1)
```

```
months = 20×1
    10
     6
     1
     8
     3
     9
     7
     4
    12
    10
     ⋮
```

```
days = randi(28,20,1)
```

```
days = 20×1
```

```
    19
    15
     9
    20
    11
    16
    25
    24
    26
    27
     :
```

```
dates = datetime([years months days],"Format","uuuu-MM-dd")
```

```
dates = 20×1 datetime
2005-10-19
2021-06-15
2015-01-09
2009-08-20
2006-03-11
2006-09-16
2013-07-25
2011-04-24
2018-12-26
2002-10-27
     :
```

```
day(dates,'name')
```

```
ans = 20×1 cell
'Wednesday'
'Tuesday'
'Friday'
'Thursday'
'Saturday'
'Saturday'
'Thursday'
'Sunday'
'Wednesday'
'Sunday'
     :
```

```
% Create a logical array to hold all the weekend dates
idx = isweekend(dates)
```

```
idx = 20×1 logical array
   0
   0
   0
   0
   1
   1
   0
   1
   0
   1
     :
```

```
% Display the weekend dates
```

```
dates(idx)
```

```
ans = 6×1 datetime
2006-03-11
2006-09-16
2011-04-24
2002-10-27
2017-01-01
2014-07-12
```

Next we import a spreadsheet file that contains a date column. This is done from the HOME tab.

```
% Call file from Workspace
DateData
```

DateData = 125×2 table

	Date	Patients
1	01/07/45	23
2	01/29/45	15
3	01/14/45	32
4	01/22/45	19
5	01/16/45	22
6	01/29/45	25
7	01/12/45	22
8	01/09/45	16
9	01/27/45	23
10	01/15/45	18
11	02/01/45	24
12	02/13/45	14
13	02/17/45	17
14	02/22/45	15

⋮

```
% Compute group summary
frequencyTable = groupsummary(DateData,"Date","monthname")
```

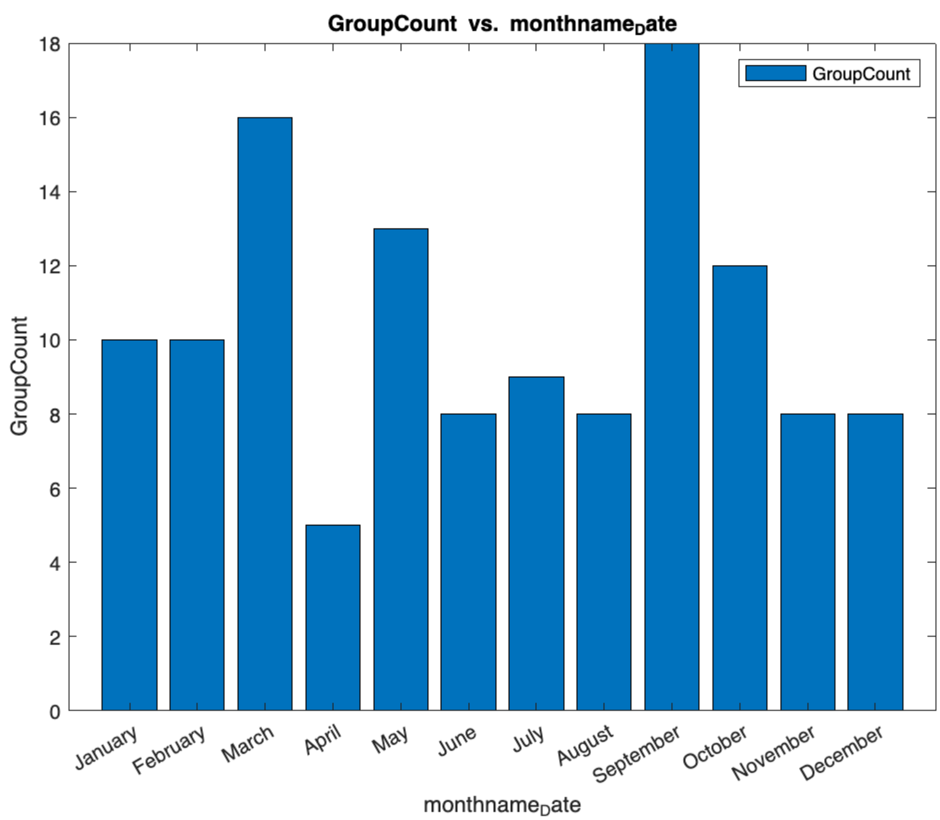
frequencyTable = 12×2 table

	monthname_Date	GroupCount
1	January	10
2	February	10
3	March	16
4	April	5
5	May	13

	monthname_Date	GroupCount
6	June	8
7	July	9
8	August	8
9	September	18
10	October	12
11	November	8
12	December	8

```
% Create bar of selected data
h3 =
bar(frequencyTable.monthname_Date,frequencyTable.GroupCount,"DisplayName","G
roupCount");

% Add xlabel, ylabel, title, and legend
xlabel("monthname_Date")
ylabel("GroupCount")
title("GroupCount vs. monthname_Date")
legend
```



Missing data

Missing data can occur commonly in data sets. Missing data refers to the absence of data points or values in a dataset where they were expected or should have been recorded. This can occur for various reasons such as non-response in surveys, data collection errors, or loss of data during storage. There are different types of missingness.

1. **Missing Completely at Random (MCAR)**: Missing data is considered MCAR if the probability of a data point being missing is completely independent of any observed or unobserved data in the dataset. In other words, the missingness is unrelated to the values of any variables. For instance, if survey responses are missing purely due to random technical errors, the data is MCAR

2. **Missing at Random (MAR)**: Missing data is considered MAR if the probability of a data point being missing is related to some of the observed data but not the missing data itself. That means the missingness can be explained by other measured variables in the dataset. For example, if older individuals are less likely to answer a particular survey question, but this tendency is consistent across all questions, the data is MAR if age information is recorded

3. **Missing Not at Random (MNAR)**: Missing data is considered MNAR if the probability of a data point being missing is related to the value of the missing data itself. This means that the missingness is systematically related to the unobserved value. For instance, if people with higher incomes are less likely to report their income, the data is MNAR.

In this section we only explore the use of the Clean Missing Data app. First we note that the `nan` keyword, which is short for not a number is used to indicate missing data.

```
% Use the nan keyword
nan
```

```
ans = NaN
```

We create a column vector of random integers and then overwrite the values at indices $3, 11$, and 14 with the `nan` keyword.

```
% Create a column vector of 20 random integers assigned to the variable
% array
array = randi(10,20,1)+20
```

```
array = 20×1
    28
    27
    30
    28
    27
    30
    21
    23
    30
    28
     ⋮
```

```
% Overwrite observations 3, 11, and 14 with missing data
array([3 11 14]) = nan
```

```
array = 20×1
    28
    27
   NaN
    28
    27
    30
    21
    23
    30
    28
     ⋮
```

We cannot calculate statistics such as the mean with this vector as the values held at indices $3, 11$, and 14 are unknown.

```
% Calculate the mean
mean(array)
```

```
ans = NaN
```

The simplest task is to remove observations with missing data. This might be a problem when there are many missing observations. Instead, the Clean Missing Data app can be used to impute the missing data.

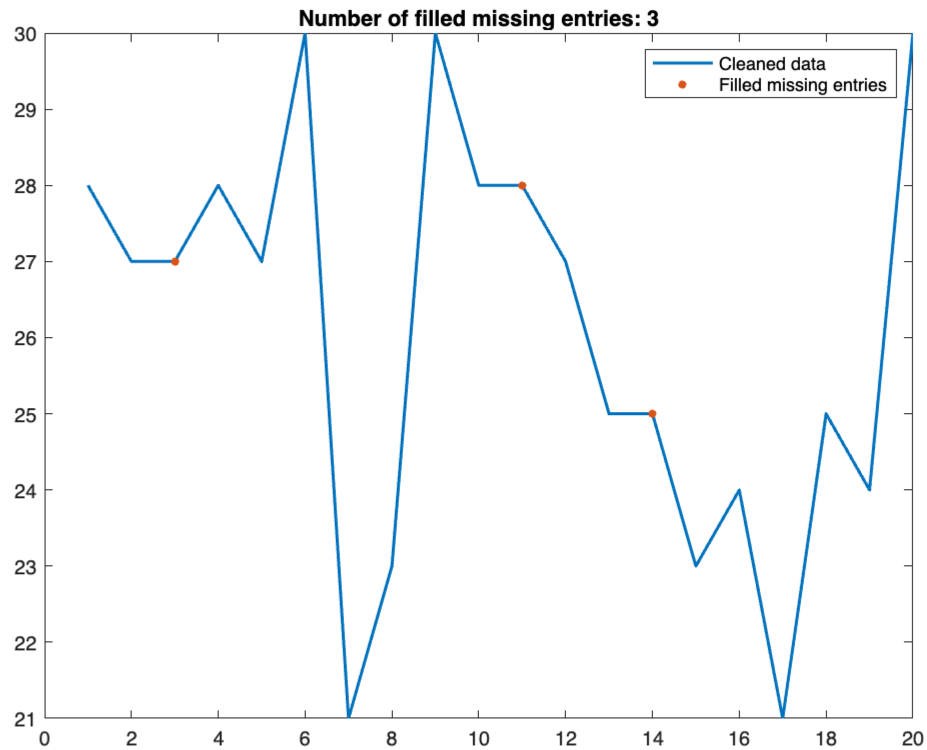
```
% Fill missing data
[cleanedArray,missingIndices] = fillmissing(array,"previous");

% Display results
figure

% Plot cleaned data
plot(cleanedArray,"SeriesIndex",1,"LineWidth",1.5,"DisplayName","Cleaned
data")
hold on

% Plot filled missing entries
plot(find(missingIndices),cleanedArray(missingIndices),".","MarkerSize",12,
...
    "SeriesIndex",2,"DisplayName","Filled missing entries")
title("Number of filled missing entries: " + nnz(missingIndices))

hold off
legend
```




```
clear missingIndices
```

The mean can now be calculated.

```
% Calculate the mean
mean(cleanedArray)
```

```
ans = 26.0500
```