Why deal with missing data?

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi

Deep Learning & Computer Vision Consultant

datacamp

Why does missing data exist?

- Real world data is messy data

> Did you know that 72% of organizations believe that data quality issues hinder customer trust and perception?

[1] [Top 9 Benefits of Data Cleansing for Businesses](https://bit.ly/2QwMrab)

Why does missing data exist?

- Values are missed during data acquisition process
  - Faulty weather sensors during weather analysis
  - Incomplete patient information for medical diagnosis etc.
- Values deleted accidentally
  - Data loss
  - Mistakenly deleted due to human error

In this course, you'll learn

- the significance of treating missing values
- to detect missing values in your messy data
- analyze the types for missingness
- treat the missing values appropriately for
  - numerical
  - time-series
  - categorical values

In this course, you'll learn

- to impute(replace) missing values using simple techniques
- to impute using advanced techniques
- to finally evaluate the best method of treating missing values

Workflow for treating missing values

1. Convert all missing values to null values.
2. Analyze the amount and type of missingness in the data.
3. Appropriately delete or impute missing values.
4. Evaluate & compare the performance of the treated/imputed dataset.

NULL value Operations

`None`

```
None or True # Same for False
True
```

```
None + True # For all operators
TypeError: unsupported operand
None / 3 # For all operators
TypeError: unsupported operand
```

```
type(None)
NoneType
```

`np.nan`

```
import numpy as np
np.nan or True  # Same for False
nan
```

```
np.nan * True # For all operators
nan
np.nan - 3 # For all operators
nan
```

```
type(np.nan)
float
```

NULL value operations

`None`

```
None == None
```

```
True
```

```
np.isnan(None)
```

```
False
```

`np.nan`

```
np.nan == np.nan
```

```
False
```

```
np.isnan(np.nan)
```

```
True
```

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Handling missing values

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

Missing values

- Usually filled with values like `'NA'`, `'-'` or `'.'` etc.

Detect missing values in College dataset

College Dataset

```
college = pd.read_csv('college.csv')
college.head()
```

```
  gradrat             lenroll rmbrd  private stufac    csat   act
0    59.0       5.1761497326  3.75      1.0   10.8       .  21.0
1    52.0       4.7791234931  3.74      1.0   17.7       .  21.0
2    75.0  6.122492809500001     .      1.0   11.4  1052.0  24.0
3    56.0       5.3181199938   4.1      1.0   11.6   940.0  23.0
4    71.0  5.631211781799999     .      1.0   18.3       .  17.0
```

Detect missing values in College dataset

```
college.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 7 columns):
gradrat    200 non-null object
lenroll    200 non-null object
rmbrd      200 non-null object
private    200 non-null float64
stufac     200 non-null object
csat       200 non-null object
act        200 non-null object
dtypes: float64(1), object(6)
```

Detect missing values in College dataset

```
csat_unique = college.csat.unique()
np.sort(csat_unique)
```

```
array(['.', '1000.0', '1006.0', '1010.0', '1013.0', '1020.0', '1024.0',
       '1026.0', '1028.0', '1036.0', '1039.0', '1040.0', '1044.0',
       '1045.0', '1050.0', '1052.0', '1060.0', '1070.0', '1080.0',
       '1092.0', '1096.0', '1109.0', '1111.0', '1120.0', '1139.0',
          ...      ...       ...       ...       ...       ...
          ...      ...       ...       ...       ...       ...
       '940.0', '943.0', '947.0', '950.0', '951.0', '964.0', '970.0',
       '979.0', '980.0', '989.0', '992.0', '994.0', '996.0', '997.0',
       '998.0'], dtype=object)
```

Replace missing values in College dataset

```
college = pd.read_csv('college.csv', na_values='.')
college.head()
```

```
   gradrat   lenroll  rmbrd  private  stufac    csat   act
0     59.0  5.176150   3.75      1.0    10.8     NaN  21.0
1     52.0  4.779123   3.74      1.0    17.7     NaN  21.0
2     75.0  6.122493    NaN      1.0    11.4  1052.0  24.0
3     56.0  5.318120   4.10      1.0    11.6   940.0  23.0
4     71.0  5.631212    NaN      1.0    18.3     NaN  17.0
```

Replace missing values in College dataset

```
college.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 7 columns):
gradrat    187 non-null float64
lenroll    199 non-null float64
rmbrd      114 non-null float64
private    200 non-null float64
stufac     199 non-null float64
csat       105 non-null float64
act        104 non-null float64
dtypes: float64(7)
```

Detect missing values in Diabetes dataset

Pima Indian Diabetes dataset

- contains various clinical diagnostic information of the patients from the Pima community

```
diabetes = pd.read_csv('pima-indians-diabetes.csv')
```

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin	BMI	Diabetes_Pedigree	Age	Class
0	6.0	148.0	72.0	35.0	NaN	33.6	0.627	50	1.0
1	1.0	85.0	66.0	29.0	NaN	26.6	0.351	31	0.0
2	8.0	183.0	64.0	NaN	NaN	23.3	0.672	32	1.0
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21	0.0
4	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33	1.0

Detect missing values in Diabetes dataset

```
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnant              768 non-null float64
Glucose               763 non-null float64
Diastolic_BP          733 non-null float64
Skin_Fold             541 non-null float64
Serum_Insulin         394 non-null float64
BMI                   768 non-null float64
Diabetes_Pedigree     768 non-null float64
Age                   768 non-null int64
Class                 768 non-null float64
dtypes: float64(8), int64(1)
```

Detect missing values in Diabetes dataset

```
diabetes.describe()
```

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin	BMI	Diabetes_Pedigree	Age	Class
count	768.000000	763.000000	733.000000	541.000000	394.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	31.992578	0.471876	33.240885	0.348958
std	3.369578	30.535641	12.382158	10.476982	118.775855	7.884160	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	64.000000	22.000000	76.250000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	29.000000	125.000000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	141.000000	80.000000	36.000000	190.000000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Detect missing values in Diabetes dataset

```
diabetes.BMI[diabetes.BMI == 0]
```

```
9   |    0.0
49  |    0.0
60  |    0.0
81  |    0.0
145 |    0.0
371 |    0.0
426 |    0.0
494 |    0.0
522 |    0.0
684 |    0.0
706 |    0.0
Name: BMI, dtype: float64
```

Replace missing values with NaN

```
diabetes.BMI[diabetes.BMI == 0] = np.nan
diabetes.BMI[np.isnan(diabetes.BMI)]
```

```
9   |   NaN
49  |   NaN
60  |   NaN
81  |   NaN
145|   NaN
371|   NaN
426|   NaN
494|   NaN
522|   NaN
684|   NaN
706|   NaN
Name: BMI, dtype: float64
```

Summary

- detect missing value characters like '.' etc.
- detect the inherent missing values within the data like '0'.
- replace them values with `NaN`

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Analyze the amount of missingness

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

Load Air Quality dataset

Air Quality dataset

- contains the sensor recordings of Ozone, Solar, Temperature and Wind

```
df_air = pd.read_csv('air-quality.csv',
                             parse_dates=['Date'],
                             index_col='Date')
df_air.head()
```

```
             Ozone  Solar  Wind  Temp
Date
1976-05-01   41.0  190.0   7.4    67
1976-05-02   36.0  118.0   8.0    72
1976-05-03   12.0  149.0  12.6    74
1976-05-04   18.0  313.0  11.5    62
1976-05-05    NaN    NaN  14.3    56
```

Nullity DataFrame

- Use either `.isnull()` or `.isna()` methods on the DataFrame

```
airquality_nullity = airquality.isnull()
airquality_nullity.head()
```

```
            Ozone  Solar   Wind   Temp
Date
1976-05-01  False  False  False  False
1976-05-02  False  False  False  False
1976-05-03  False  False  False  False
1976-05-04  False  False  False  False
1976-05-05   True   True  False  False
```

Total missing values

```
airquality_nullity.sum()
```

```
Ozone    37
Solar     7
Wind      0
Temp      0
dtype: int64
```

Percentage of missingness

```
airquality_nullity.mean() * 100
```

```
Ozone    24.183007
Solar     4.575163
Wind      0.000000
Temp      0.000000
dtype: float64
```

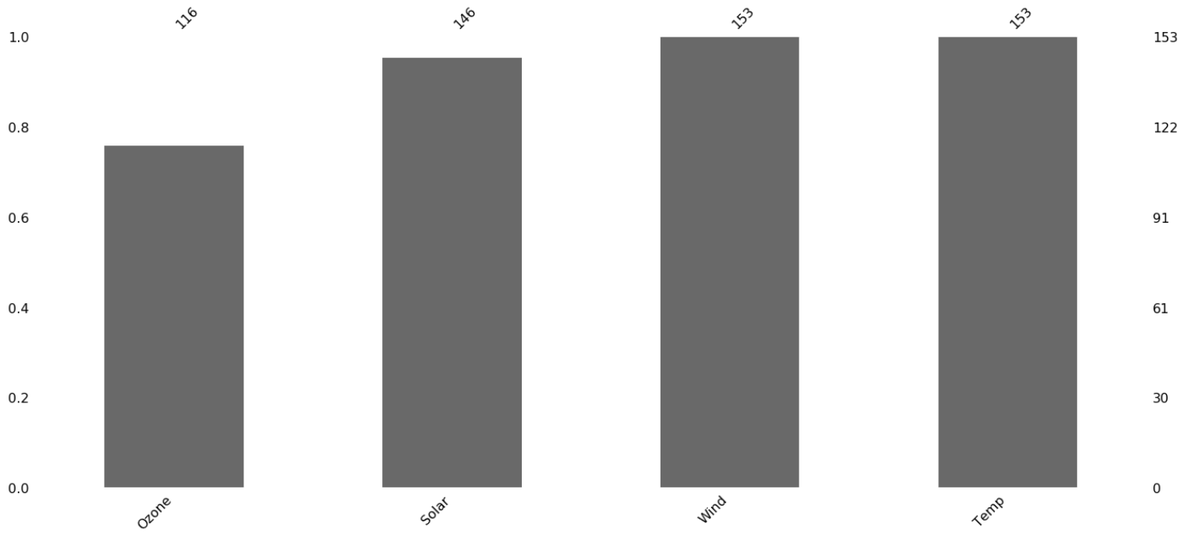
Nullity Bar

Missingno package

- Package for graphical analysis of missing values

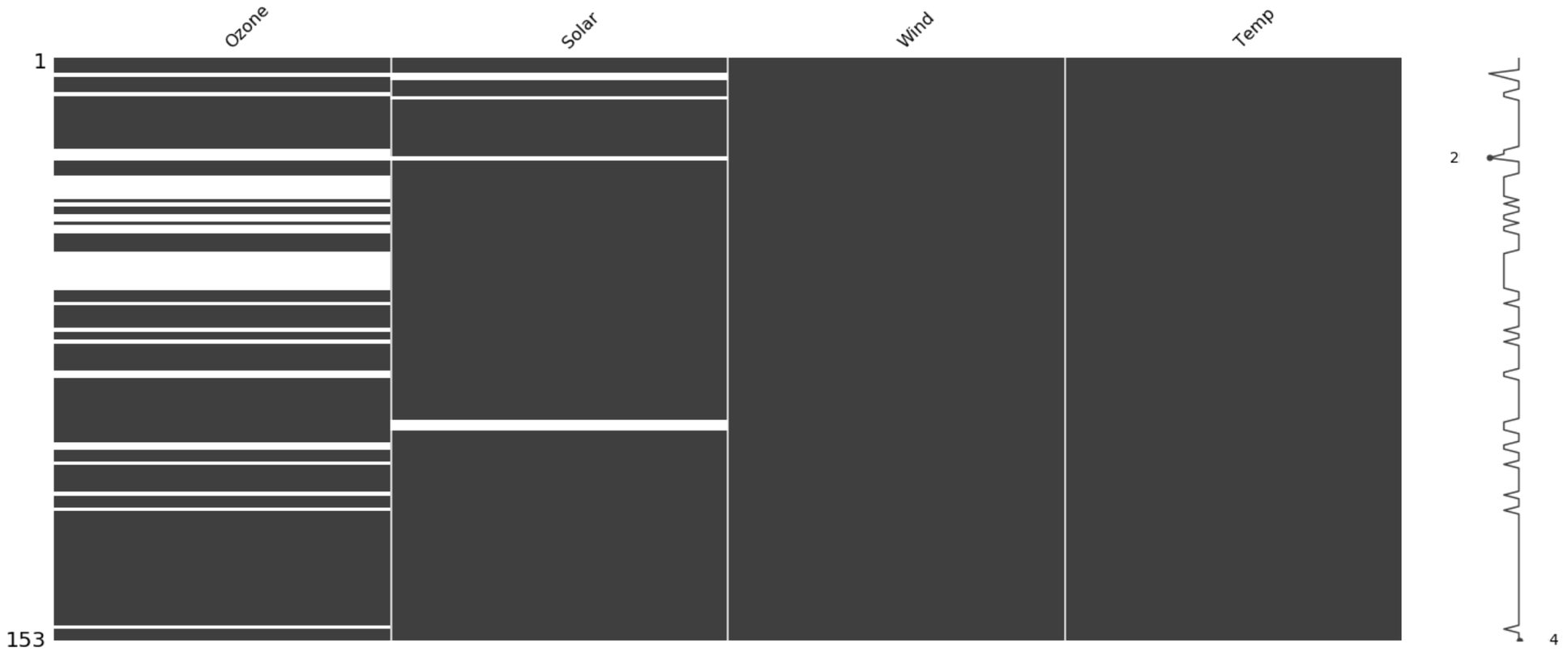
```
import missingno as msno
```

```
msno.bar(airquality)
```



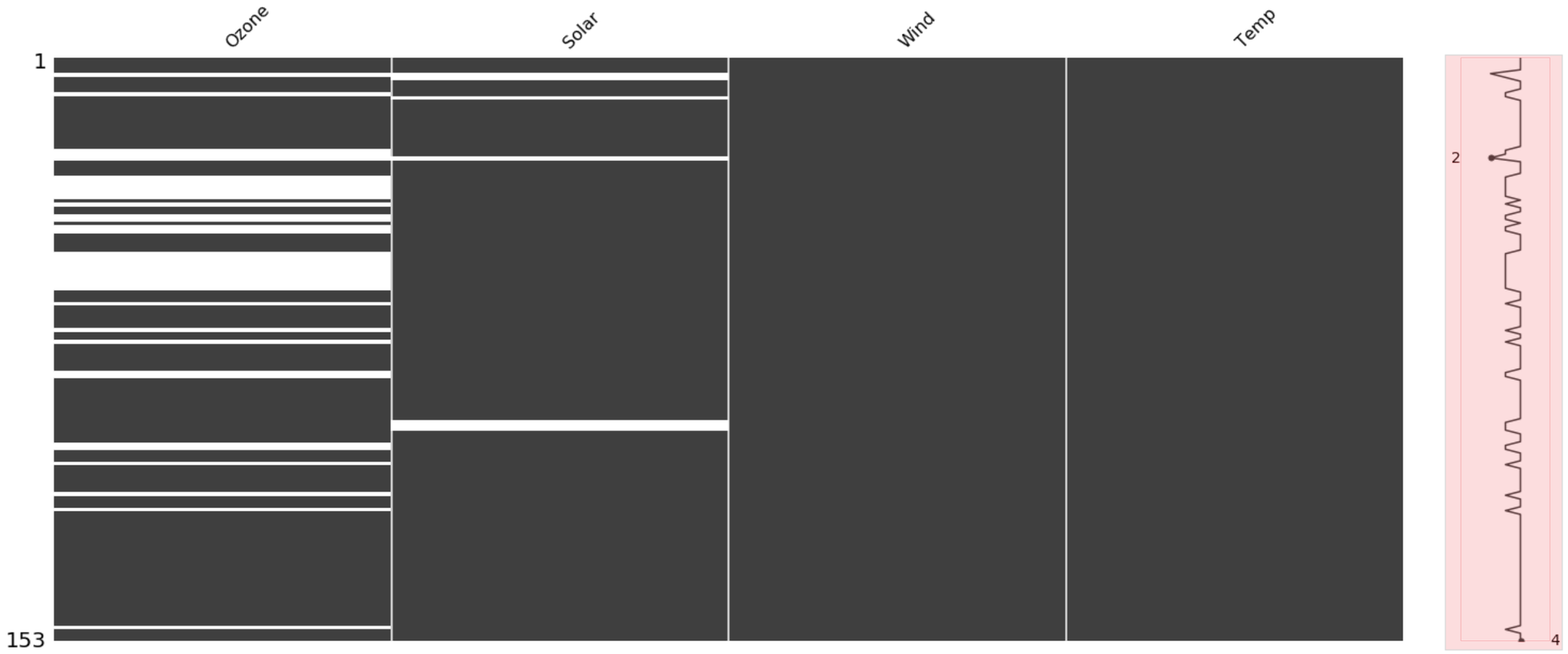
Nullity Matrix

```
msno.matrix(airquality)
```



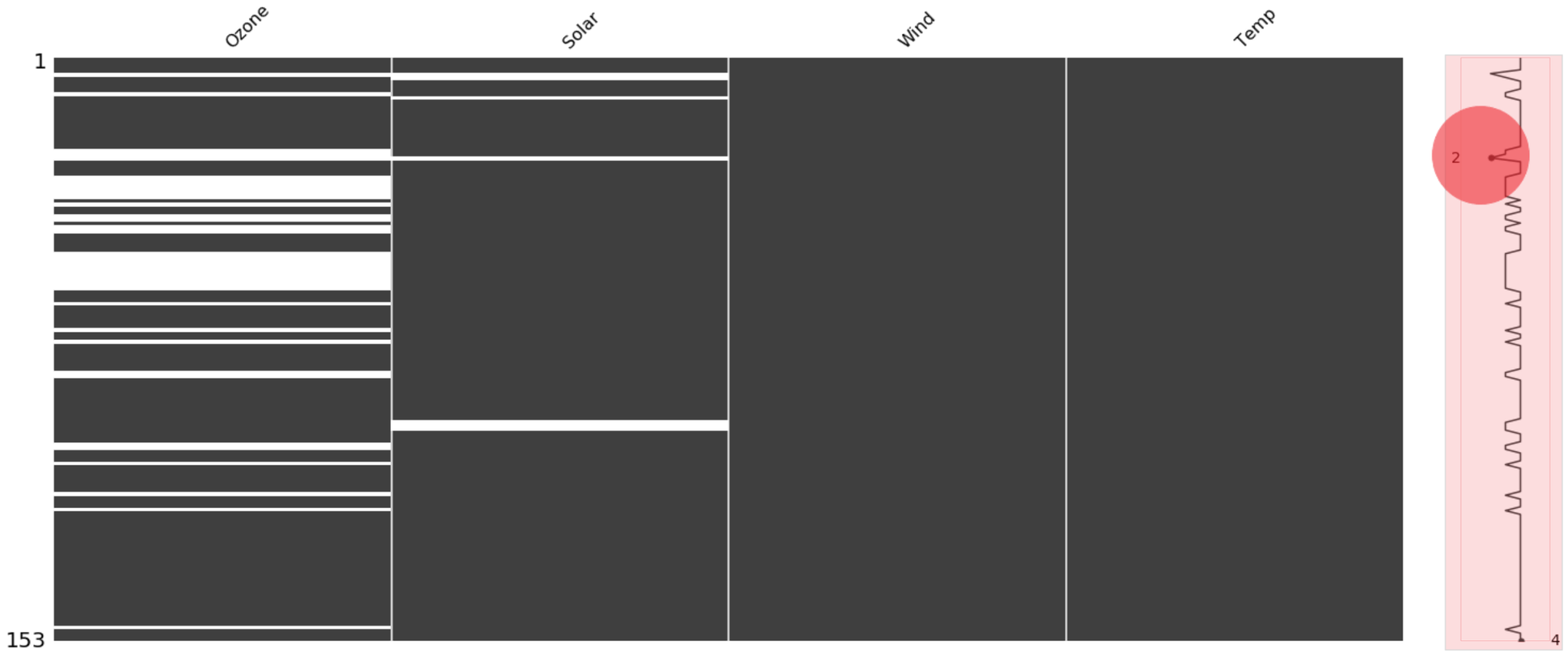
Nullity Matrix

```
msno.matrix(airquality)
```



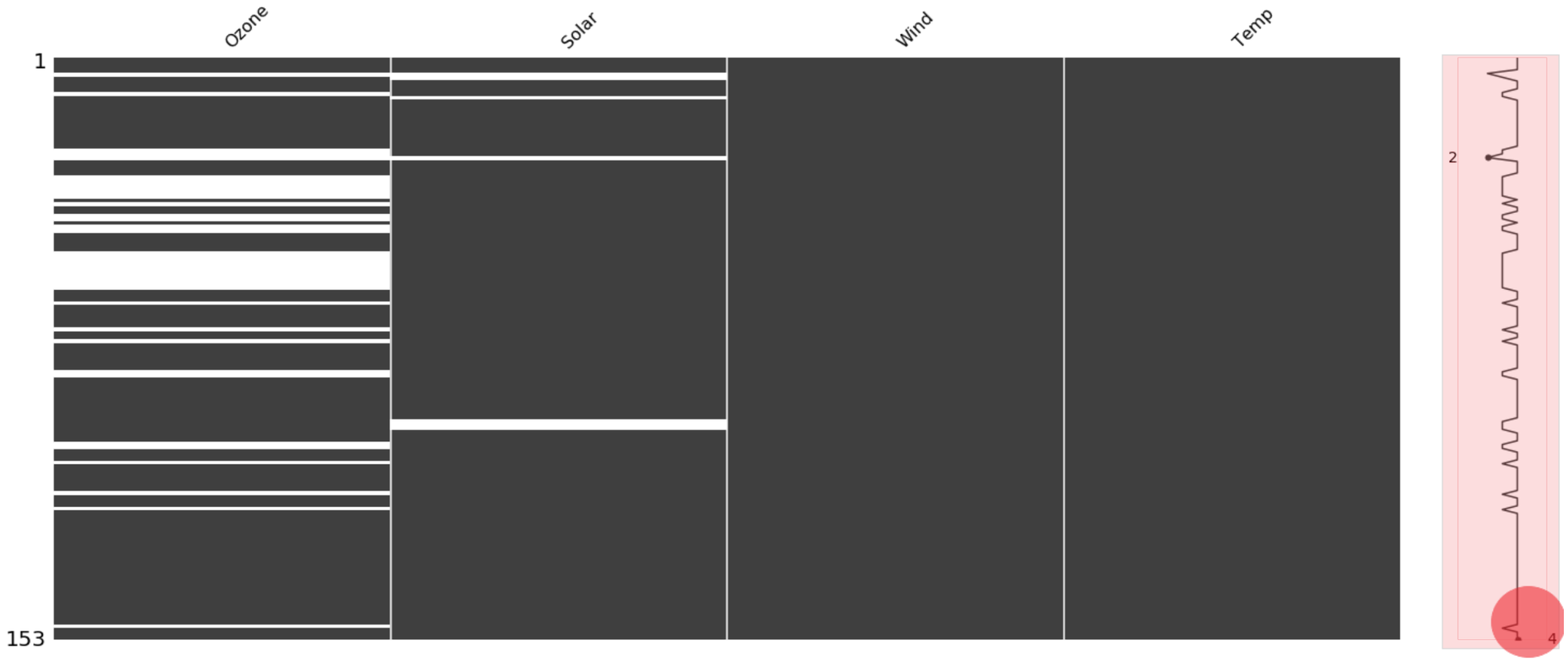
Nullity Matrix

```
msno.matrix(airquality)
```



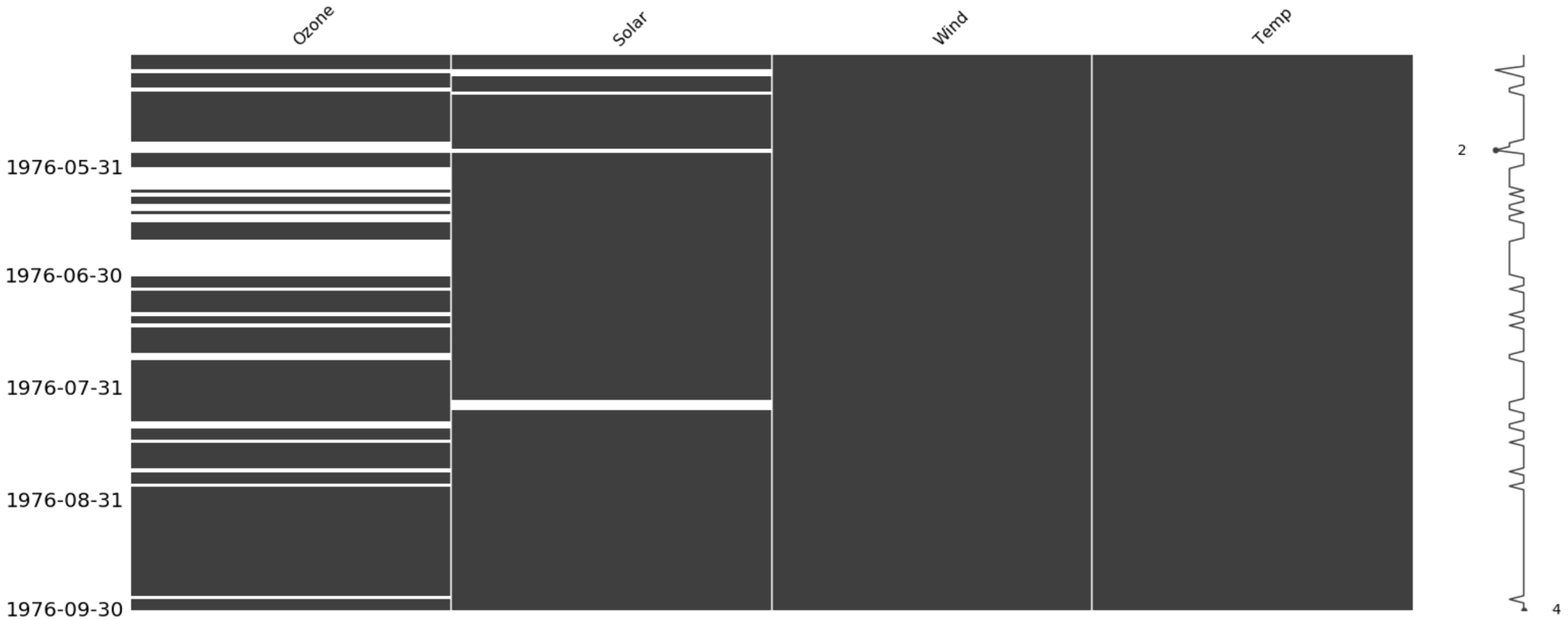
Nullity Matrix

```
msno.matrix(airquality)
```



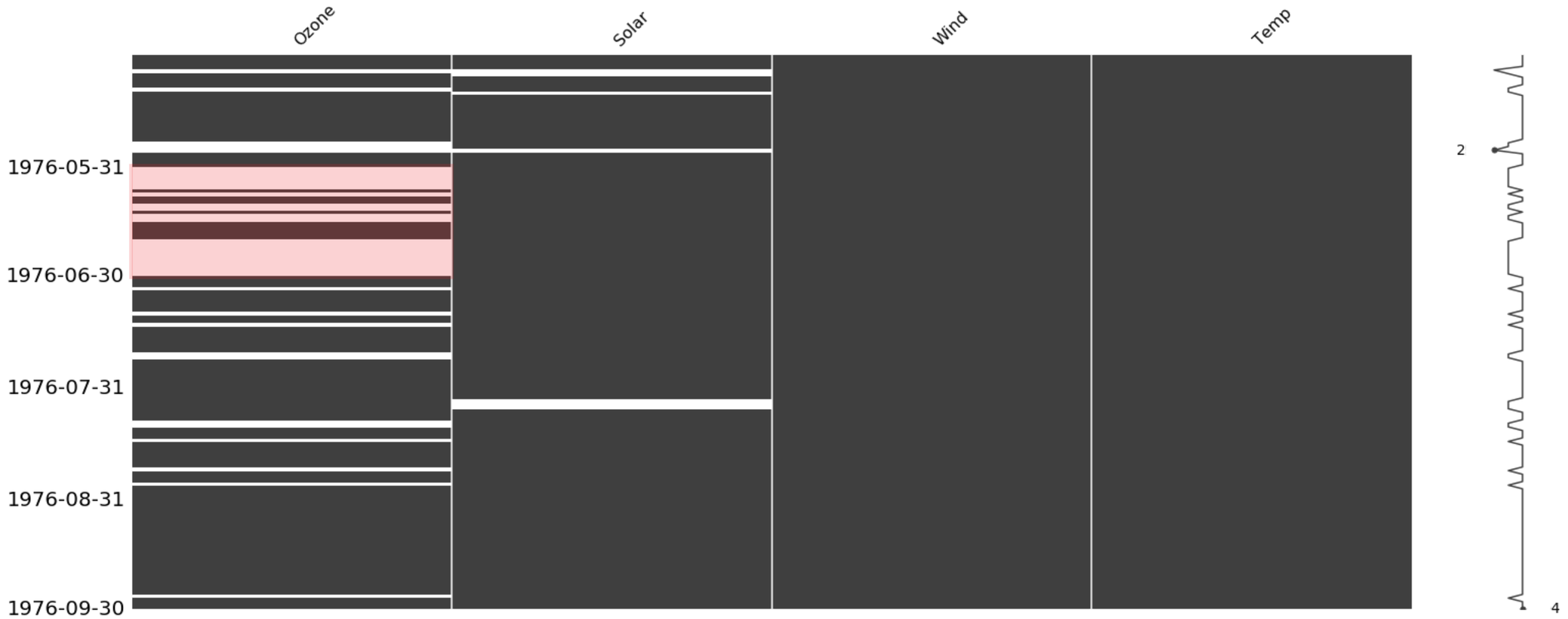
Nullity Matrix for time-series data

```
msno.matrix(airquality, freq='M')
```



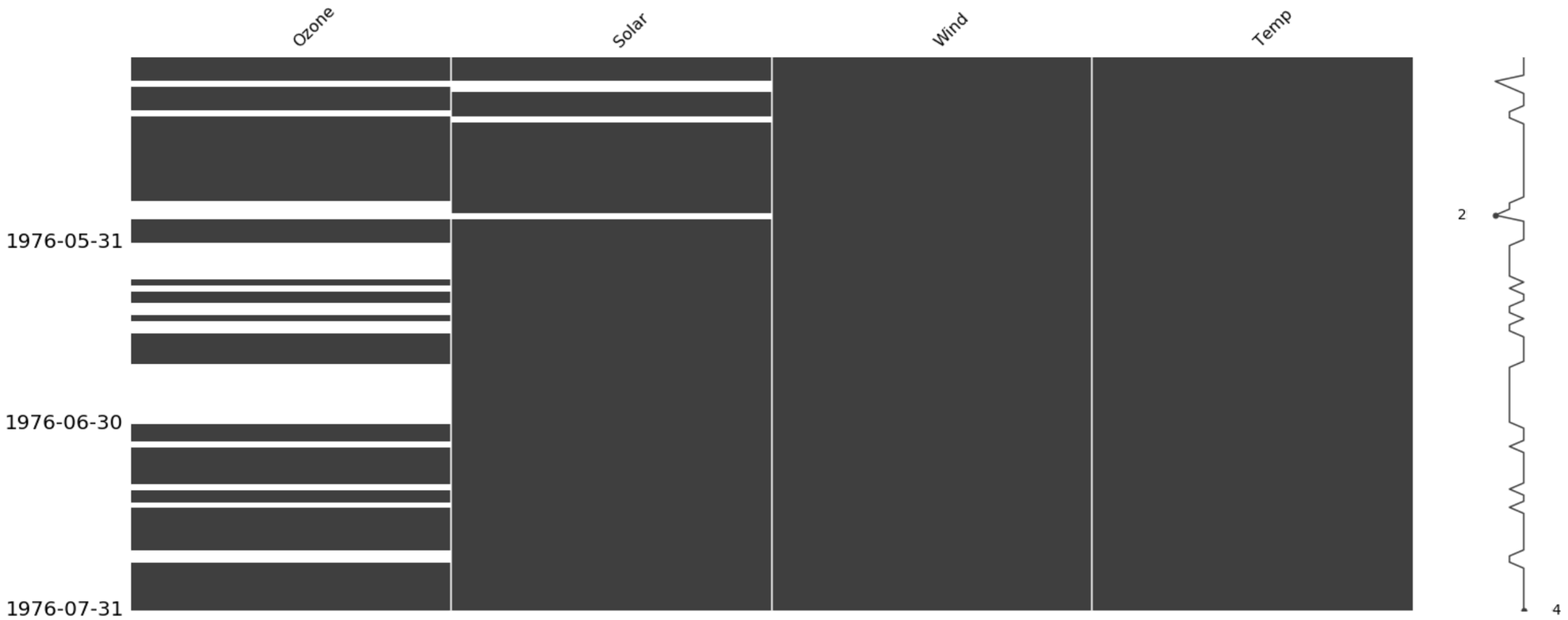
Nullity Matrix for time-series data

```
msno.matrix(airquality, freq='M')
```



Fine tuning the matrix

```
msno.matrix(airquality.loc['May-1976': 'Jul-1976'], freq='M')
```



Summary

In this lesson we learned to analyze

- the amount of missingness numerically
- the amount of missingness graphically
- the percentage of missingness
- the nullity matrix for regular datasets
- the nullity matrix for time-series datasets

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Is the data missing at random?

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

Possible reasons for missing data

Note — (variable → data field or column in a DataFrame)

- Values simply missing at random instances or intervals in a variable
- Values missing due to another variable
- Values missing due to the missingness of the same or another variable

Types of missingness

1. Missing Completely at Random (MCAR)
2. Missing at Random (MAR)
3. Missing Not at Random (MNAR)

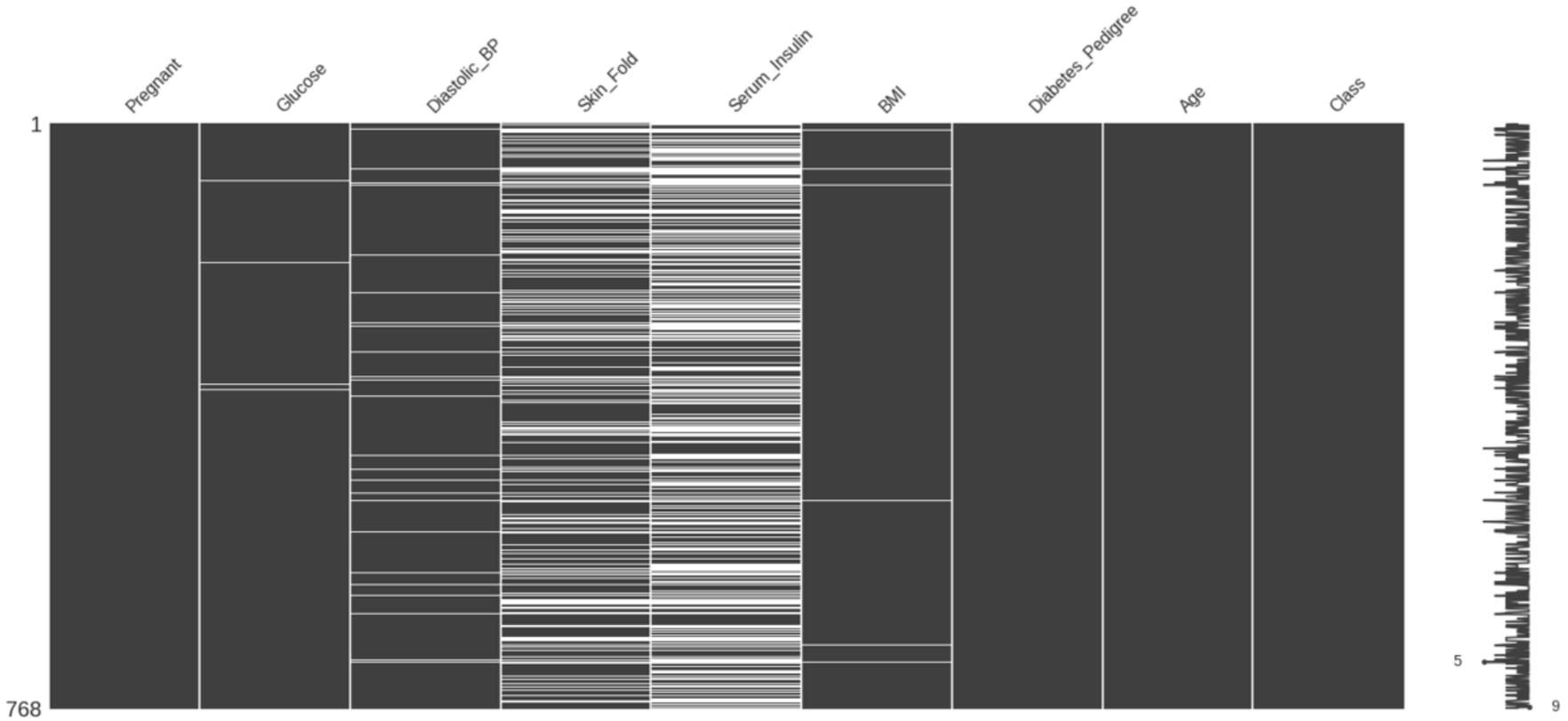
Missing Completely at Random(MCAR)

Definition:

> "Missingness has no relationship between any values, observed or missing"

MCAR - An example

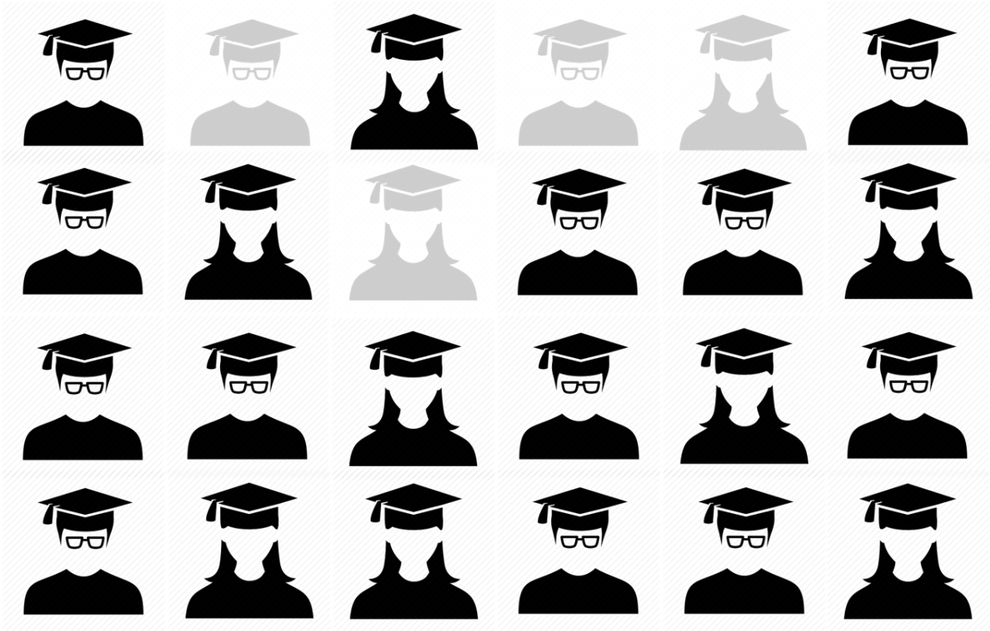
```
msno.matrix(diabetes)
```



Missing at Random(MAR)

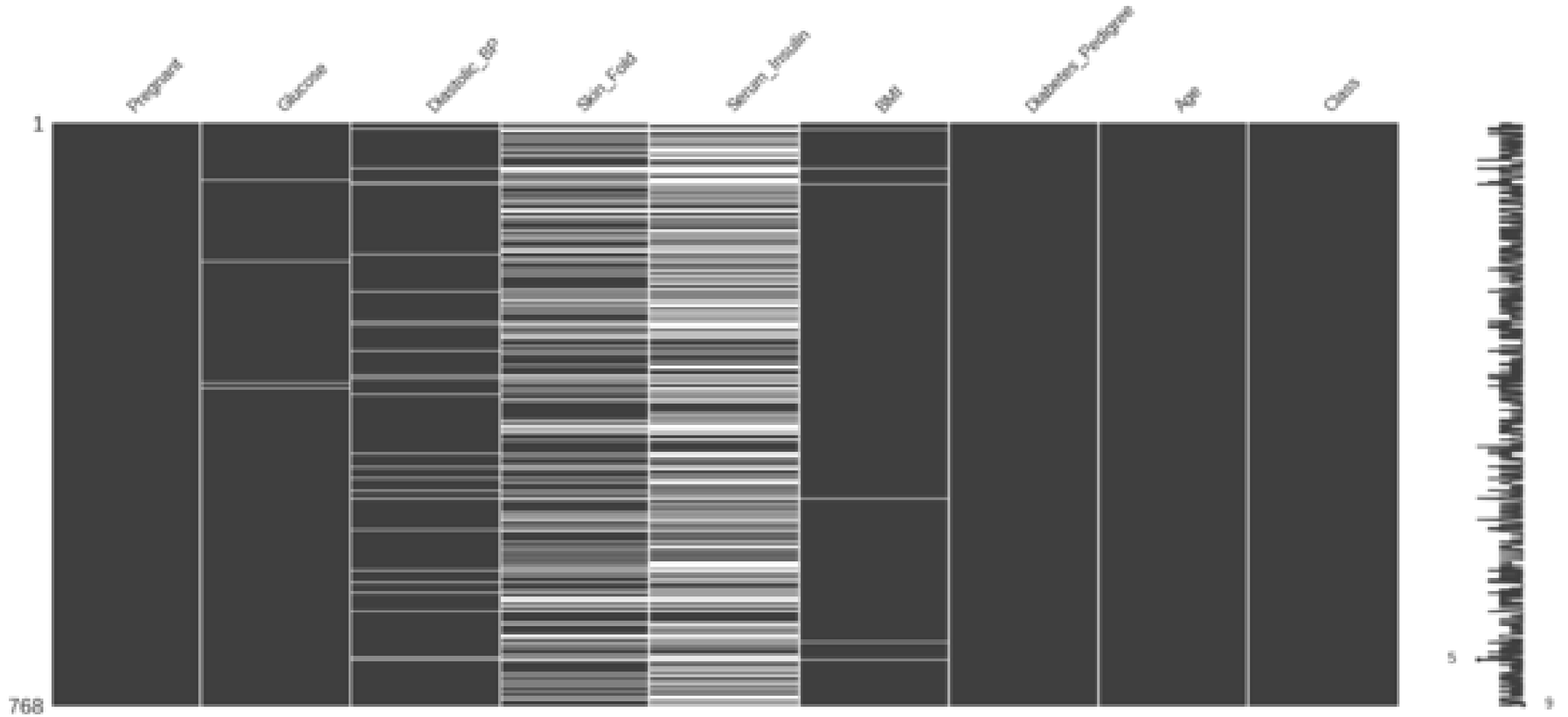
Definition:

> "There is a systematic relationship between missingness and other observed data, but not the missing data"



MAR - An example

```
msno.matrix(diabetes)
```



Missing not at Random(MNAR)

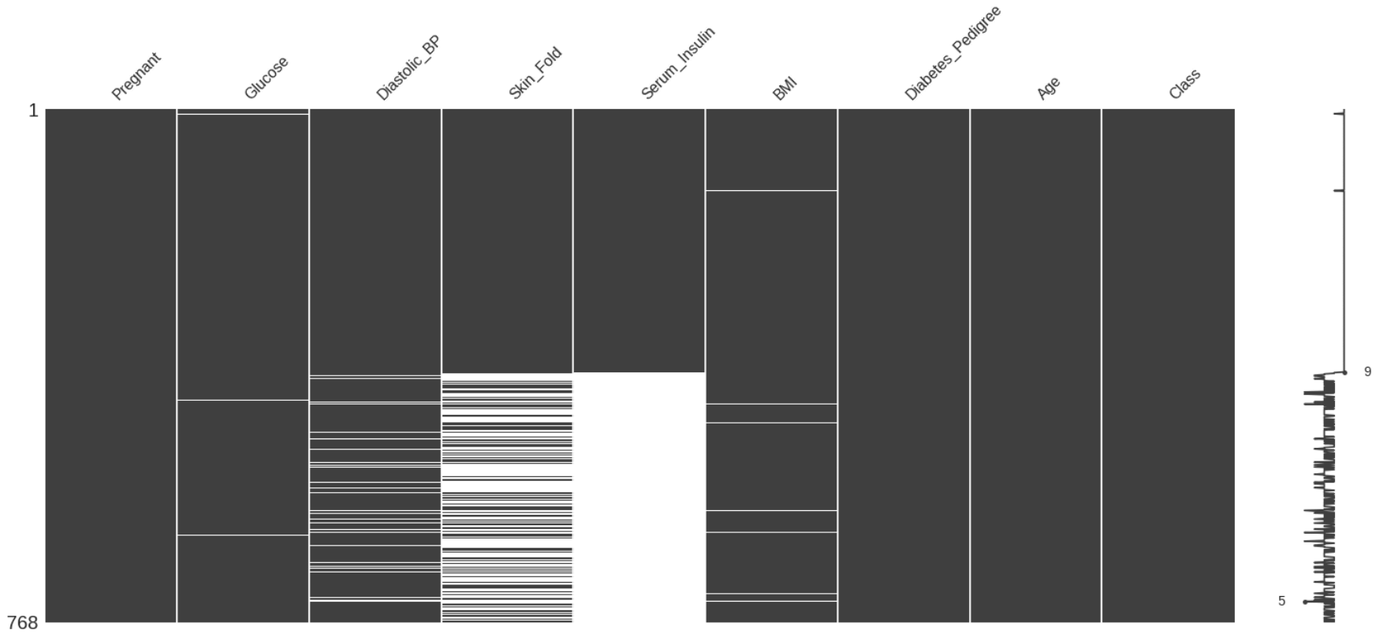
Definition:

> "There is a relationship between missingness and its values, missing or non-missing"

MNAR - An example

- Missingness pattern of the `diabetes` sorted by `Serum_Insulin`

```
sorted = diabetes.sort_values('Serum_Insulin')
msno.matrix(sorted)
```



Summary

- Possible reasons for missingness
  - Missing Completely at Random (MCAR),
  - Missing at Random (MAR) or
  - Missing Not at Random (MNAR)
- Detecting missingness pattern by sorting the variables
- Mapping missingness to MCAR, MAR & MNAR

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Finding patterns in missing data

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

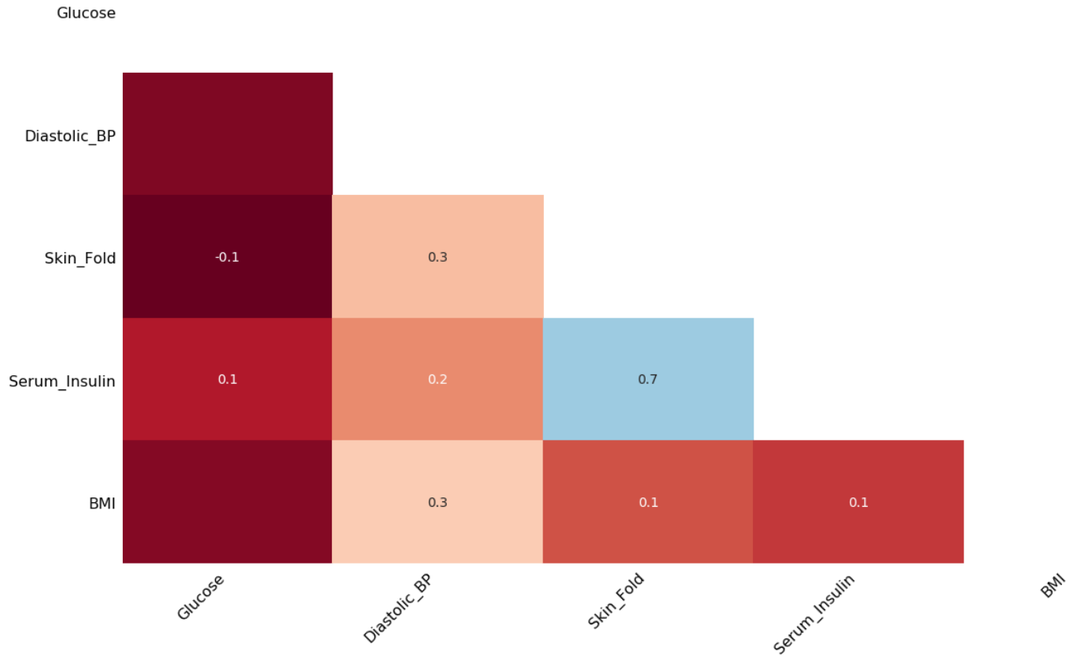
datacamp

Finding correlations between missingness

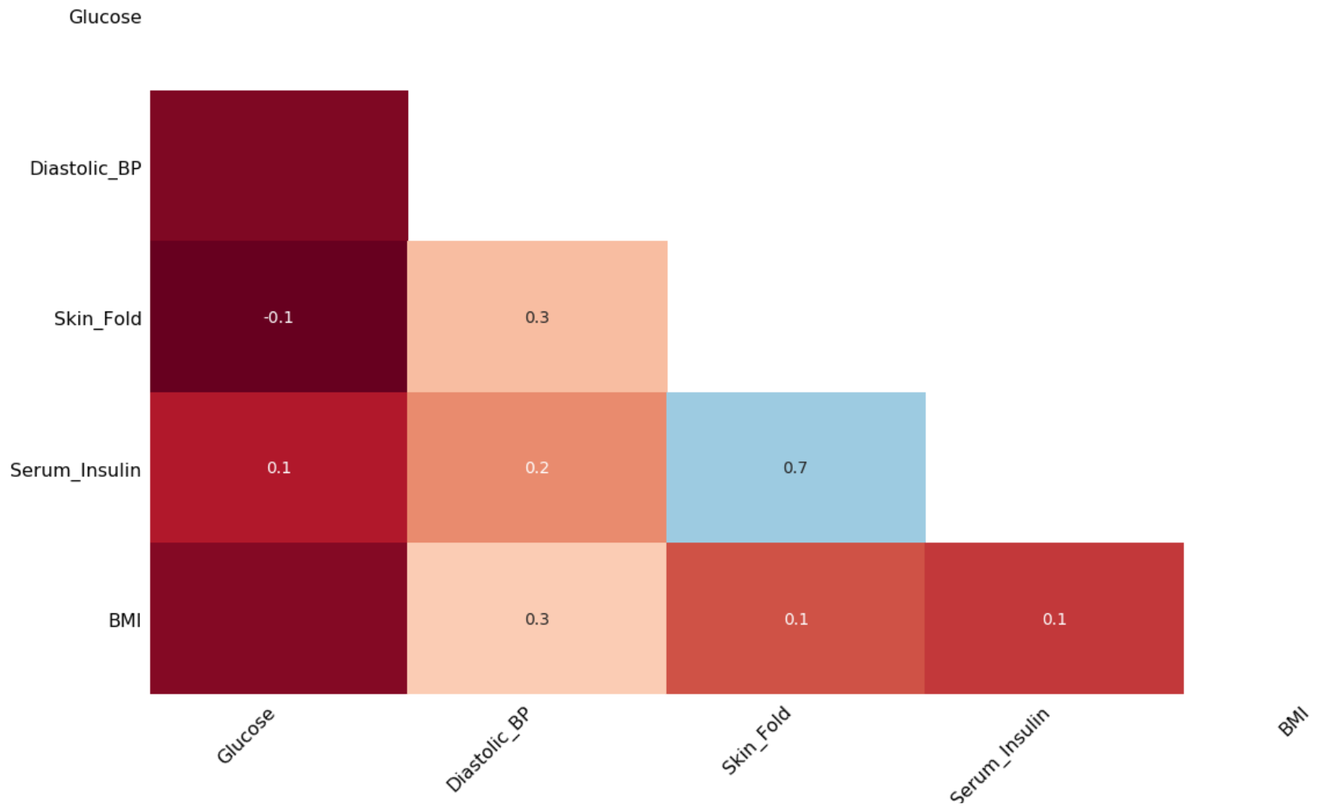
- Missingness heatmap or correlation map
- Missingness dendrogram

Missingness Heatmap

- Graph of correlation of missing values between columns
- Explains the dependencies of missingness between columns



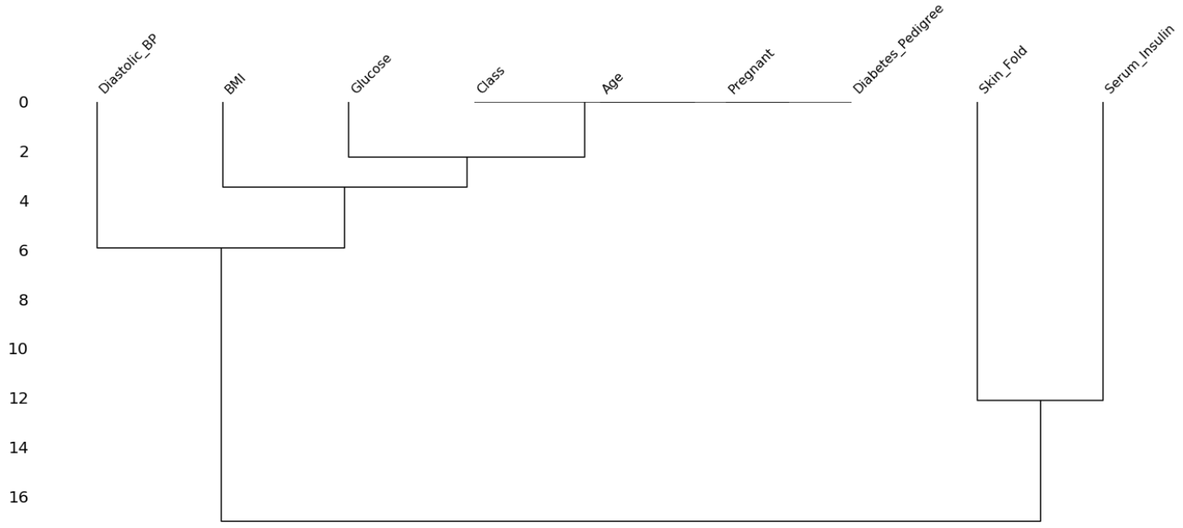
```
import missingno as msno
diabetes = pd.read_csv('pima-indians-diabetes data.csv')
msno.heatmap(diabetes)
```

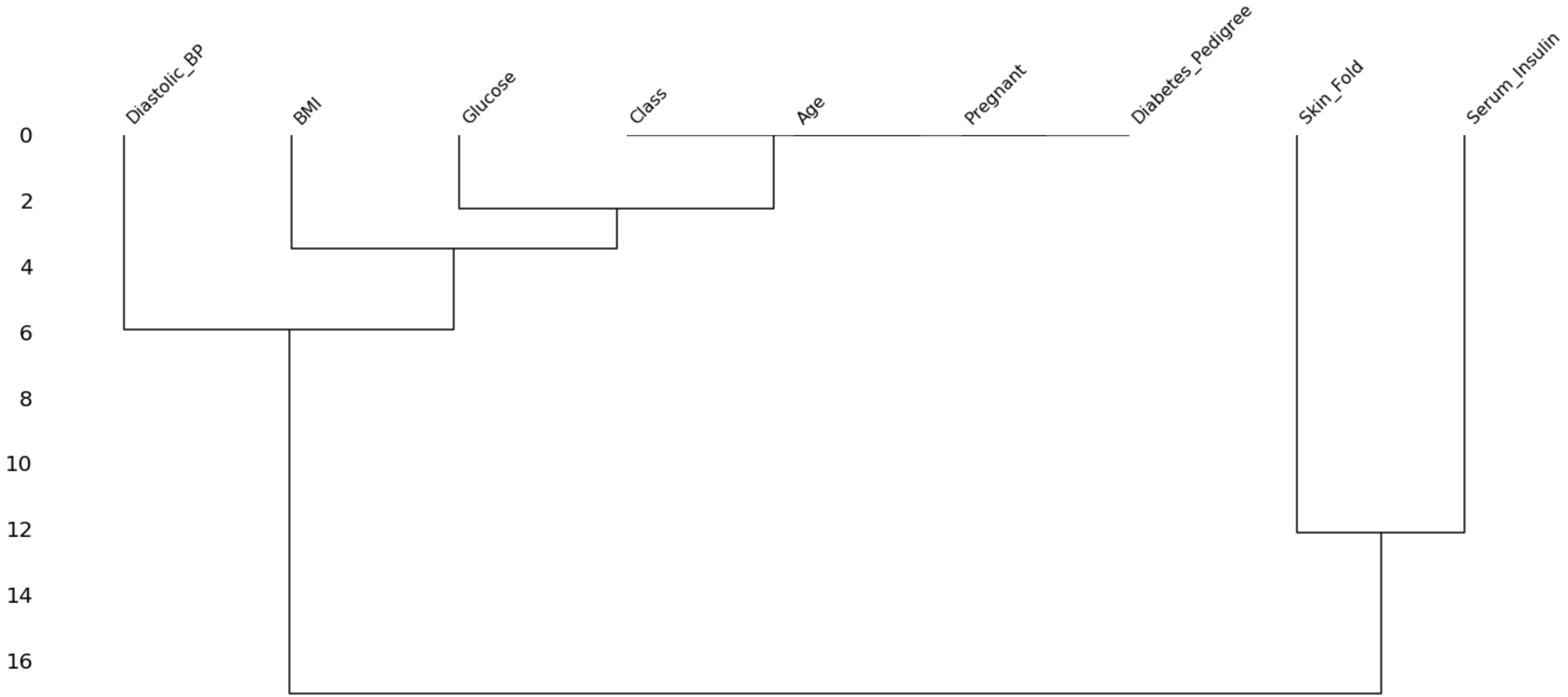


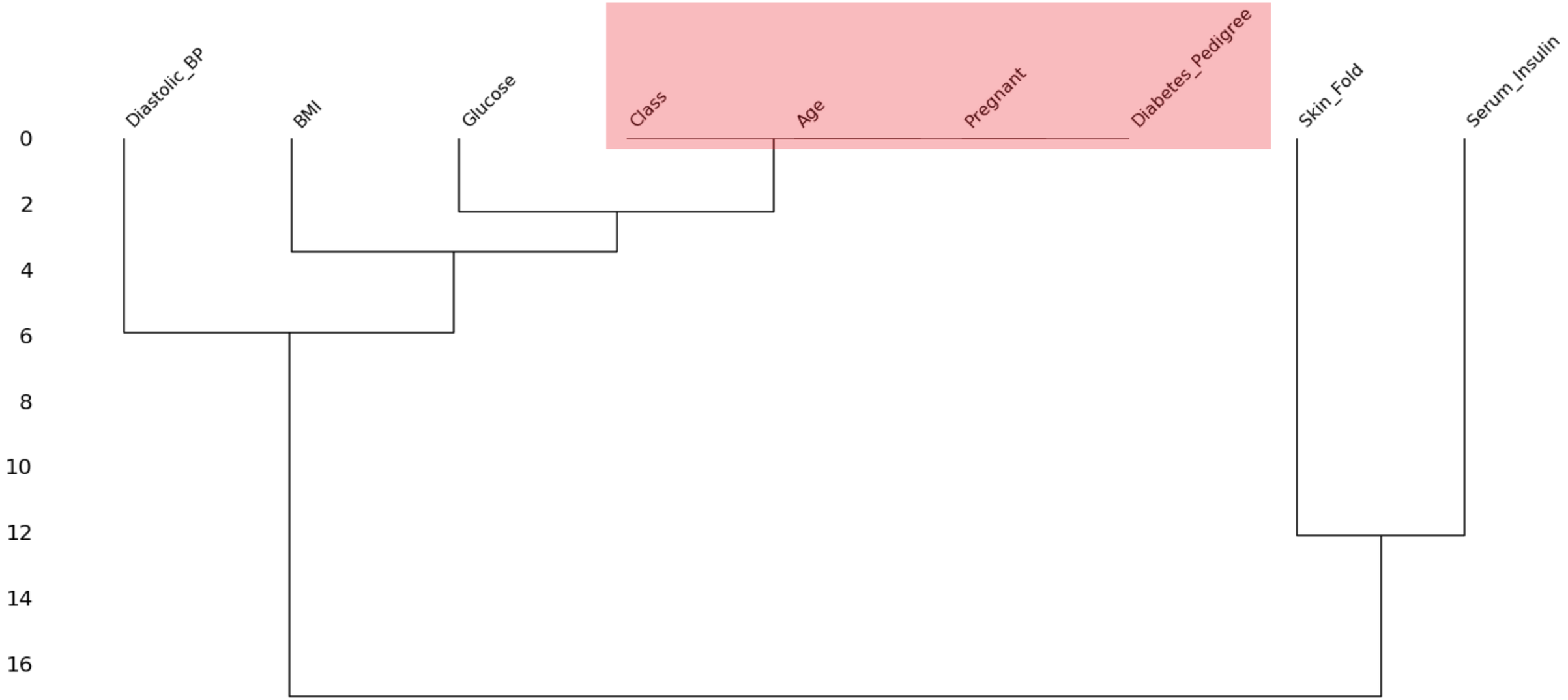
Missingness Dendrogram

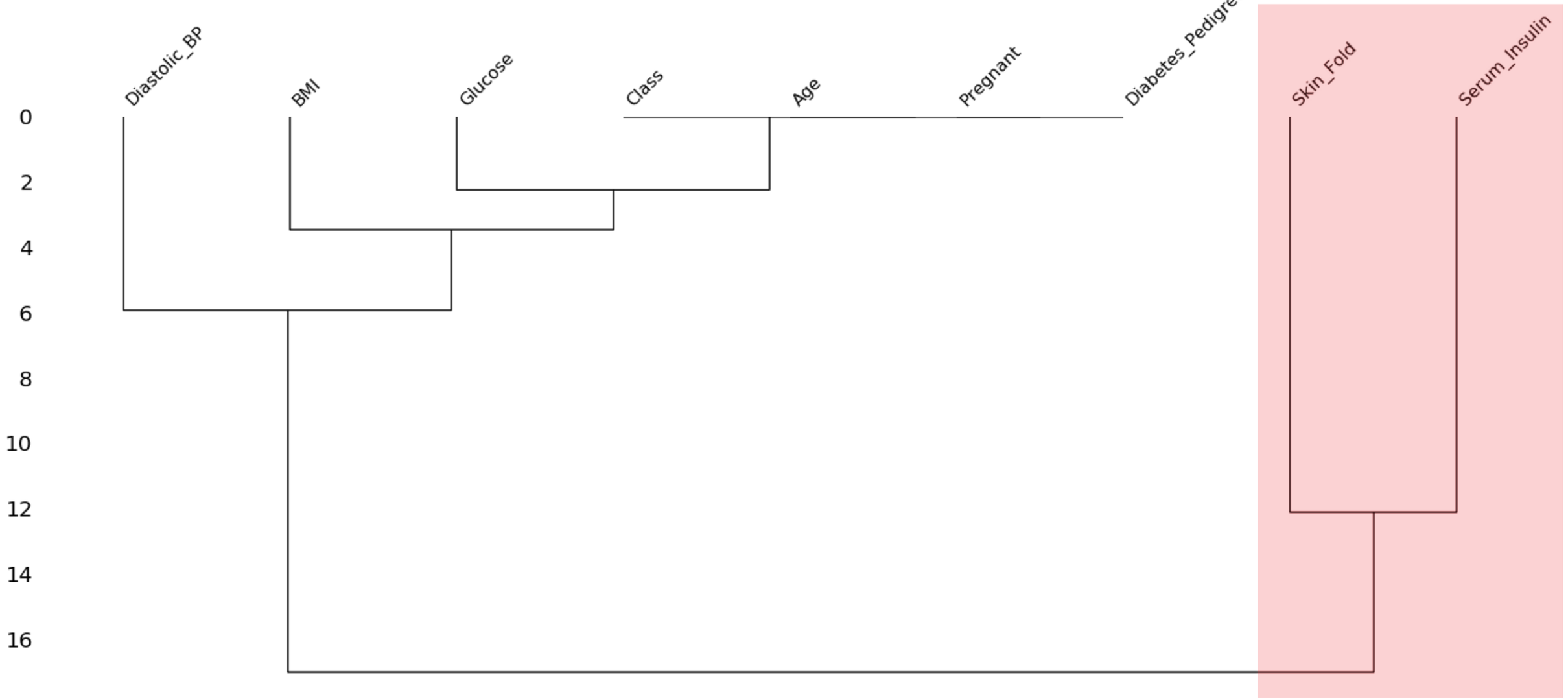
- Tree diagram of missingness
- Describes correlation of variables by grouping them

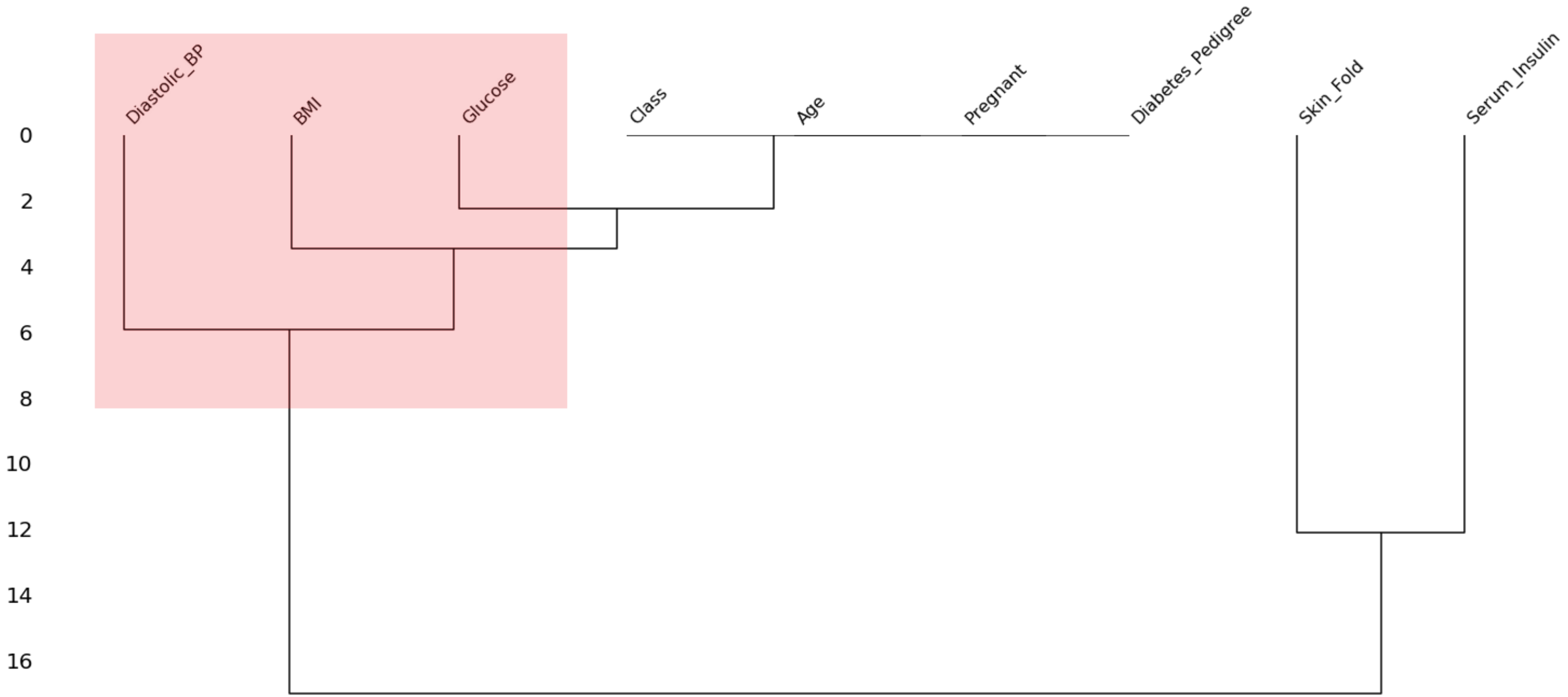
```
msno.dendrogram(diabetes)
```




Diastolic_BP
BMI
Glucose
Class
Age
Pregnant
Diabetes_Pedigree
Skin_Fold
Serum_Insulin
0
2
4
6
8
10
12
14
16


Diastolic_BP
BMI
Glucose
Class
Age
Pregnant
Diabetes_Pedigree
Skin_Fold
Serum_Insulin
0
2
4
6
8
10
12
14
16


Diastolic_BP
BMI
Glucose
Class
Age
Pregnant
Diabetes_Pedigree
Skin_Fold
Serum_Insulin
0
2
4
6
8
10
12
14
16


Diastolic_BP
BMI
Glucose
Class
Age
Pregnant
Diabetes_Pedigree
Skin_Fold
Serum_Insulin
0
2
4
6
8
10
12
14
16

Summary

- Analyze missingness heatmap

```
msno.heatmap(df)
```

- Analayze missingness dendrogram

```
msno.dendrogram(df)
```

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Visualizing missingness across a variable

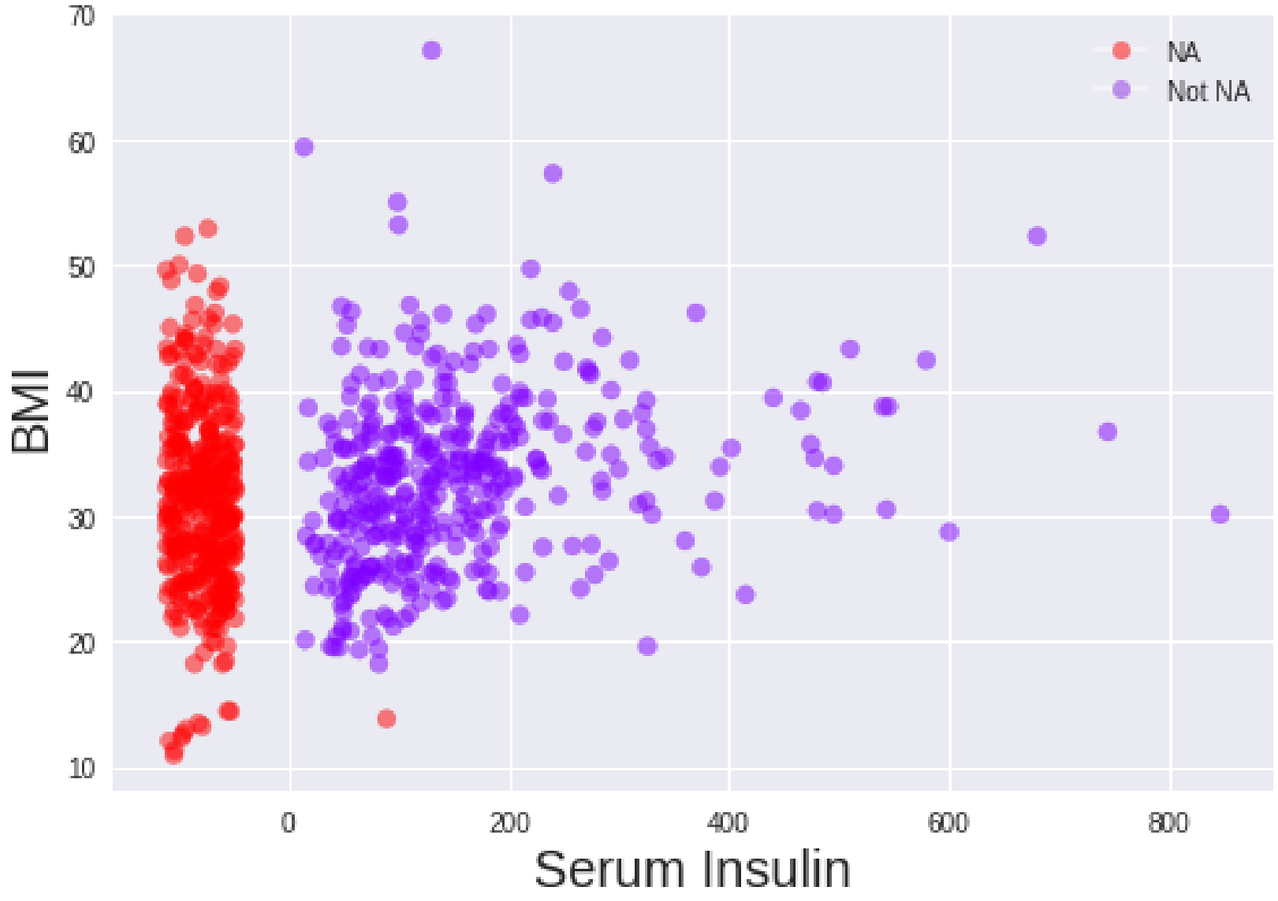
DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

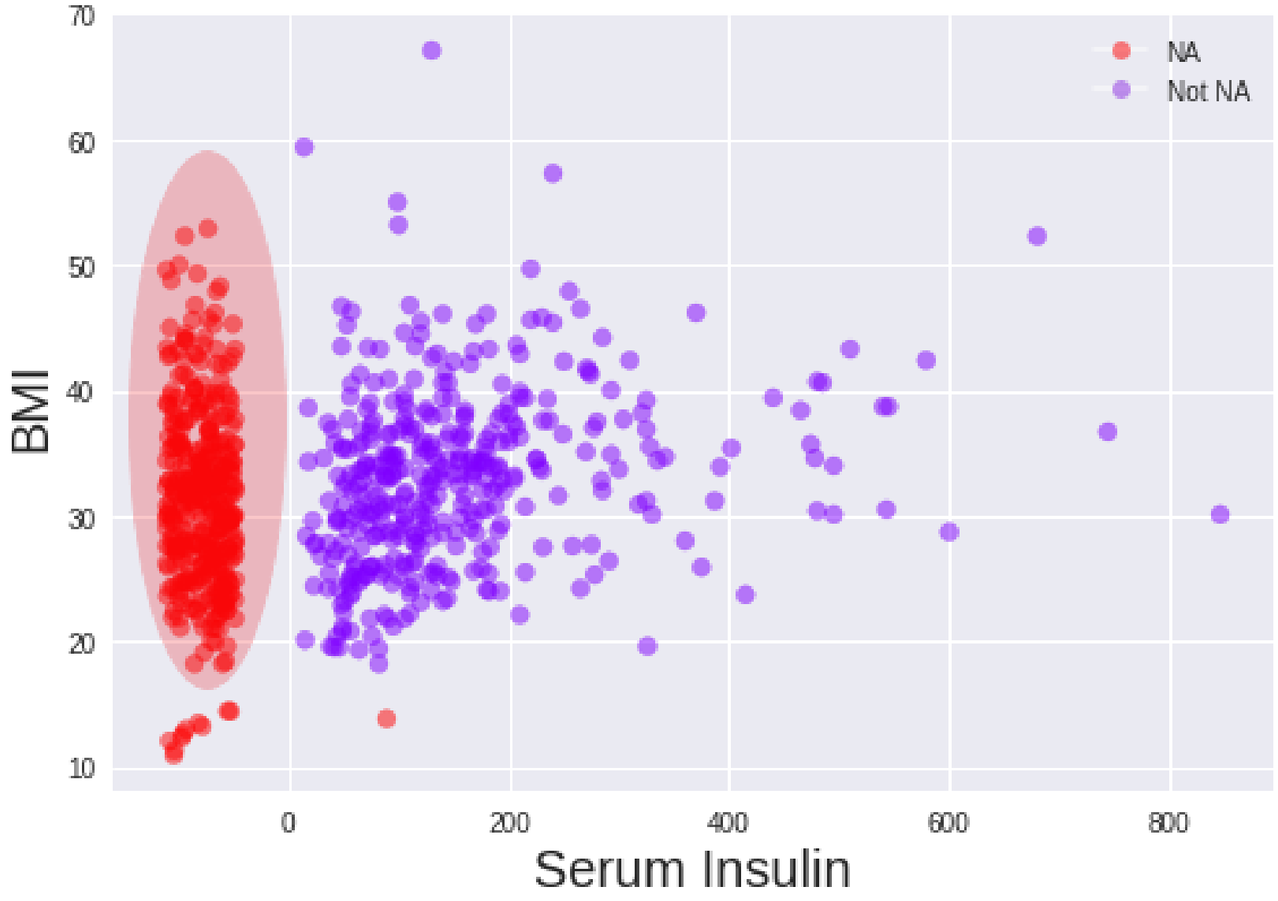
Missingness across a variable

- Visualize how missingness of a variable changes against another variable



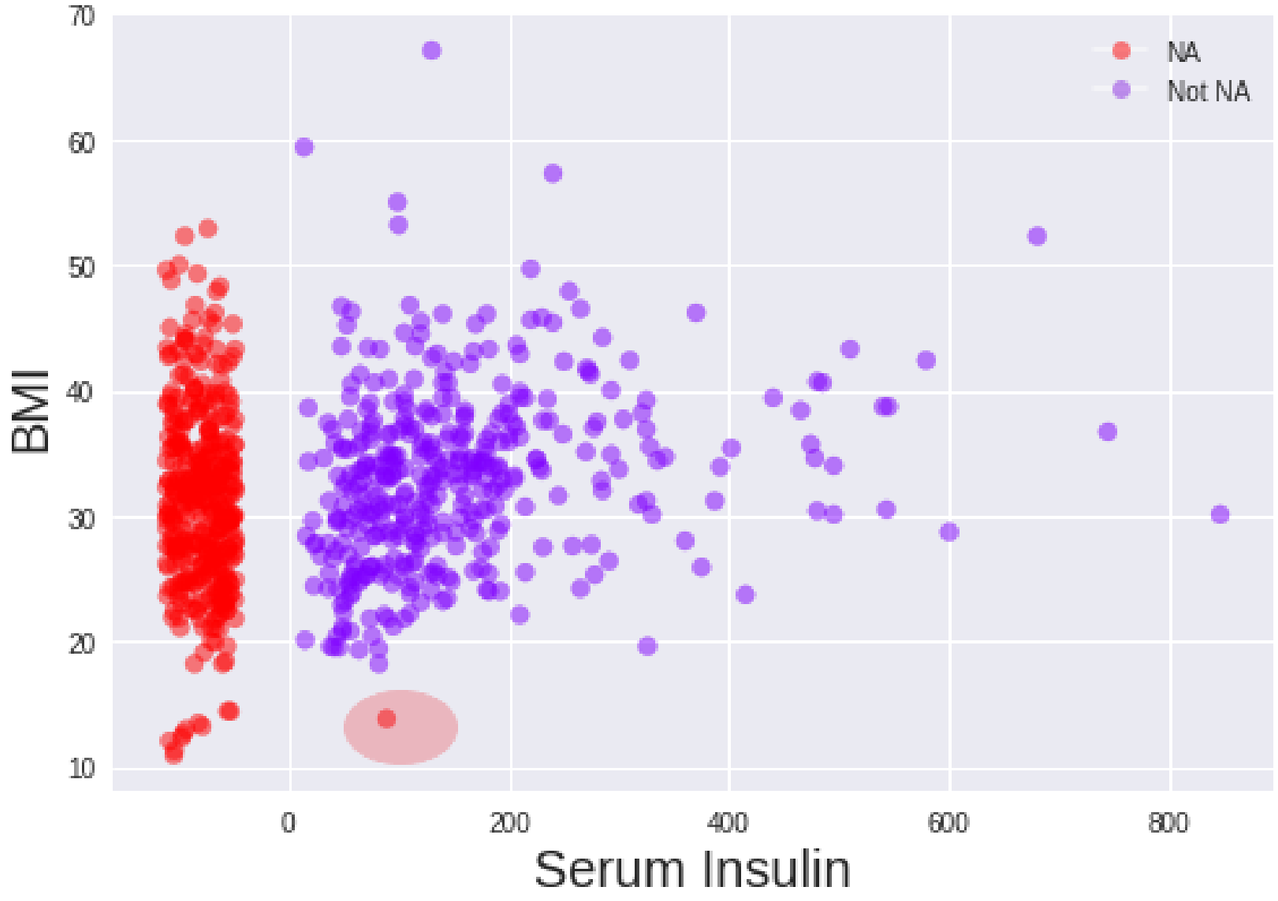
Missingness across a variable

- Visualize how missingness of a variable changes against another variable



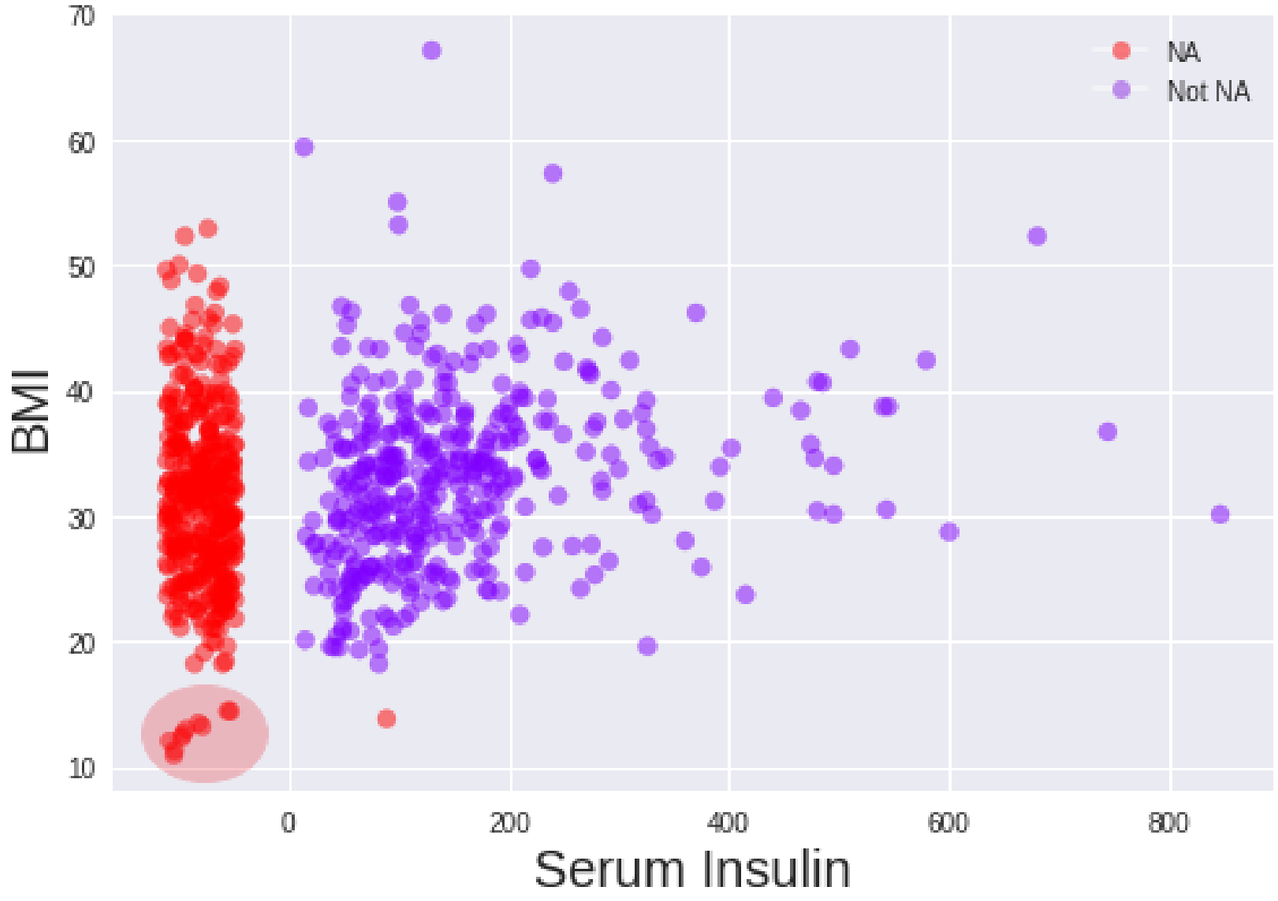
Missingness across a variable

- Visualize how missingness of a variable changes against another variable



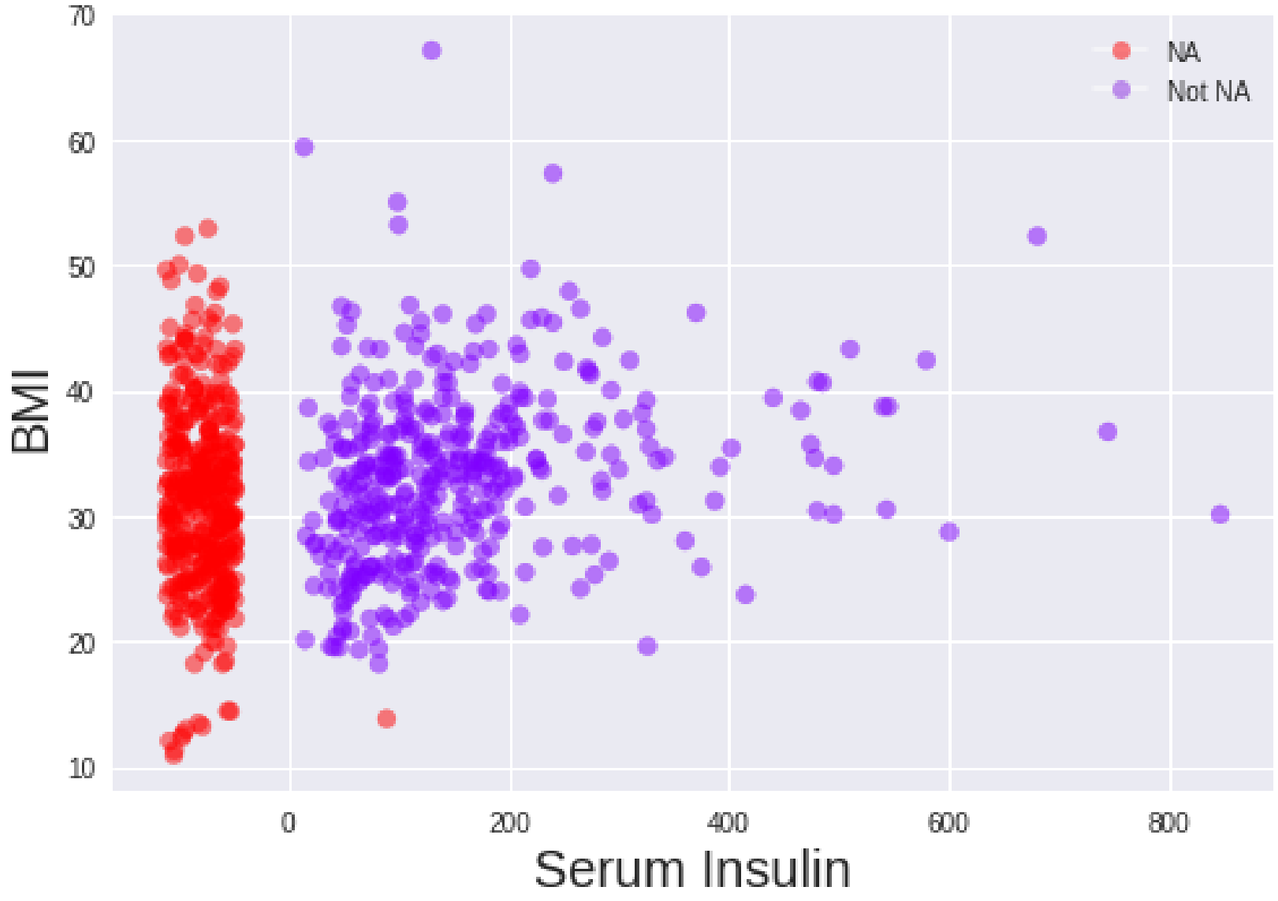
Missingness across a variable

- Visualize how missingness of a variable changes against another variable



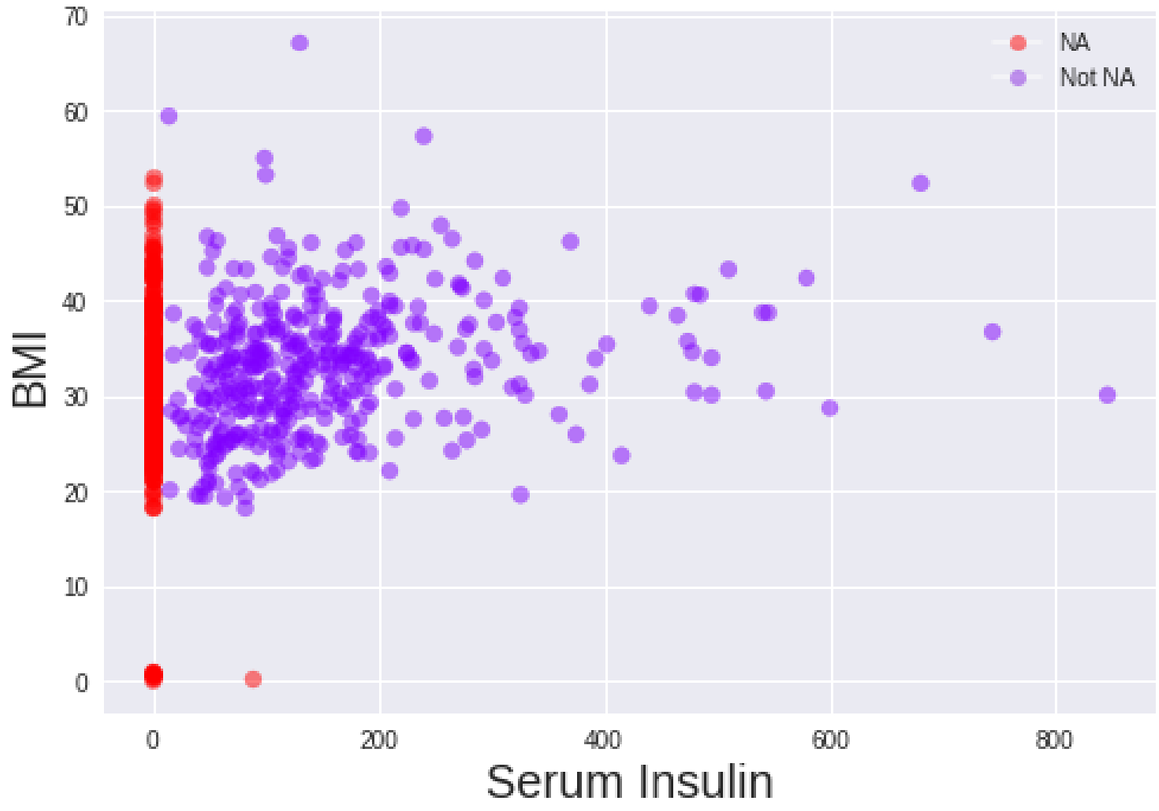
Missingness across a variable

- Visualize how missingness of a variable changes against another variable



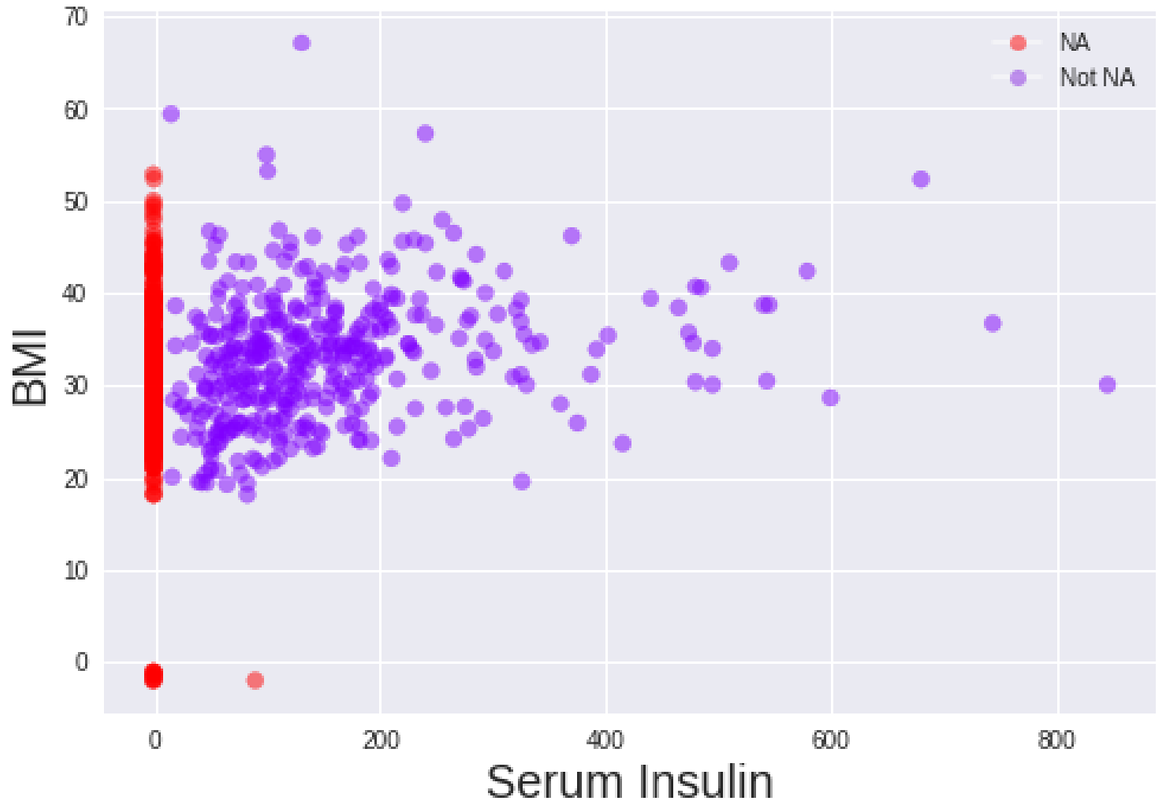
Filling dummy Values

```
from numpy.random import rand

BMI_null = diabetes['BMI'].isnull()
num_nulls = BMI_null.sum()

# Generate random values
dummy_values = rand(num_nulls)
```



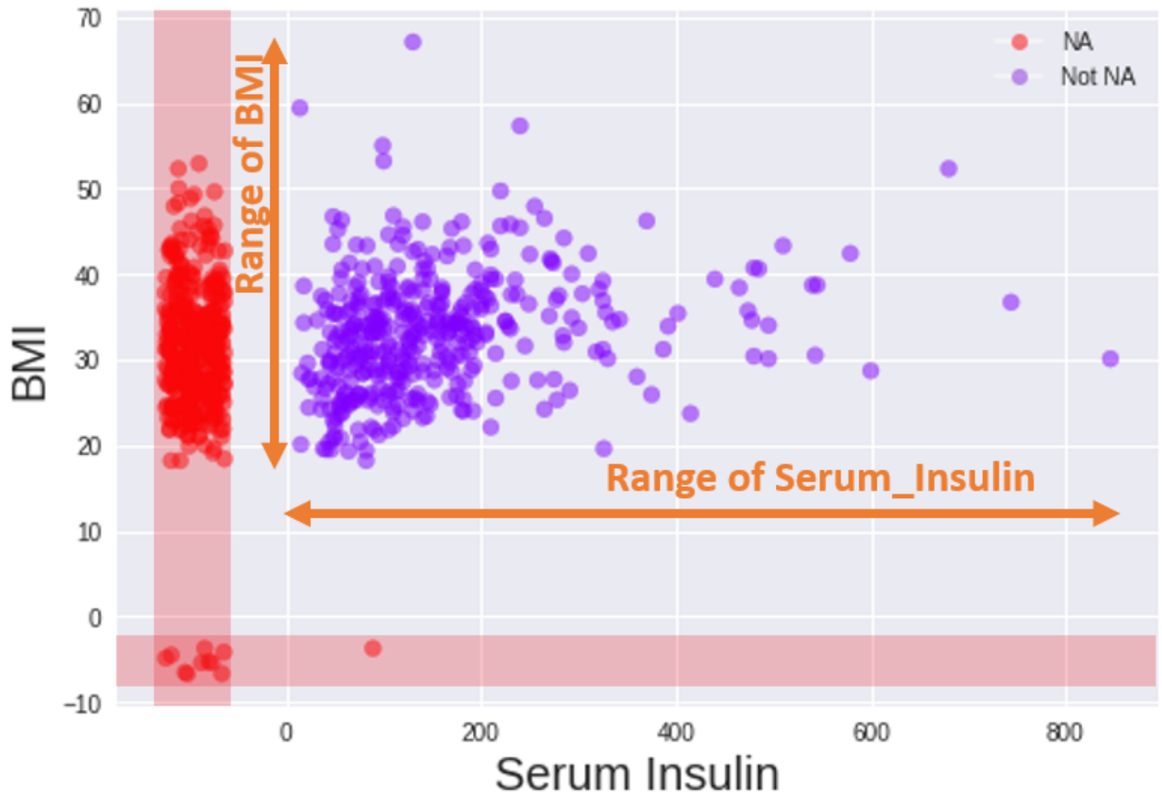
Filling dummy Values

```
from numpy.random import rand

BMI_null = diabetes['BMI'].isnull()
num_nulls = BMI_null.sum()

# Generate random values
dummy_values = rand(num_nulls)
# Shift to -2 & -1
dummy_values = dummy_values - 2
```



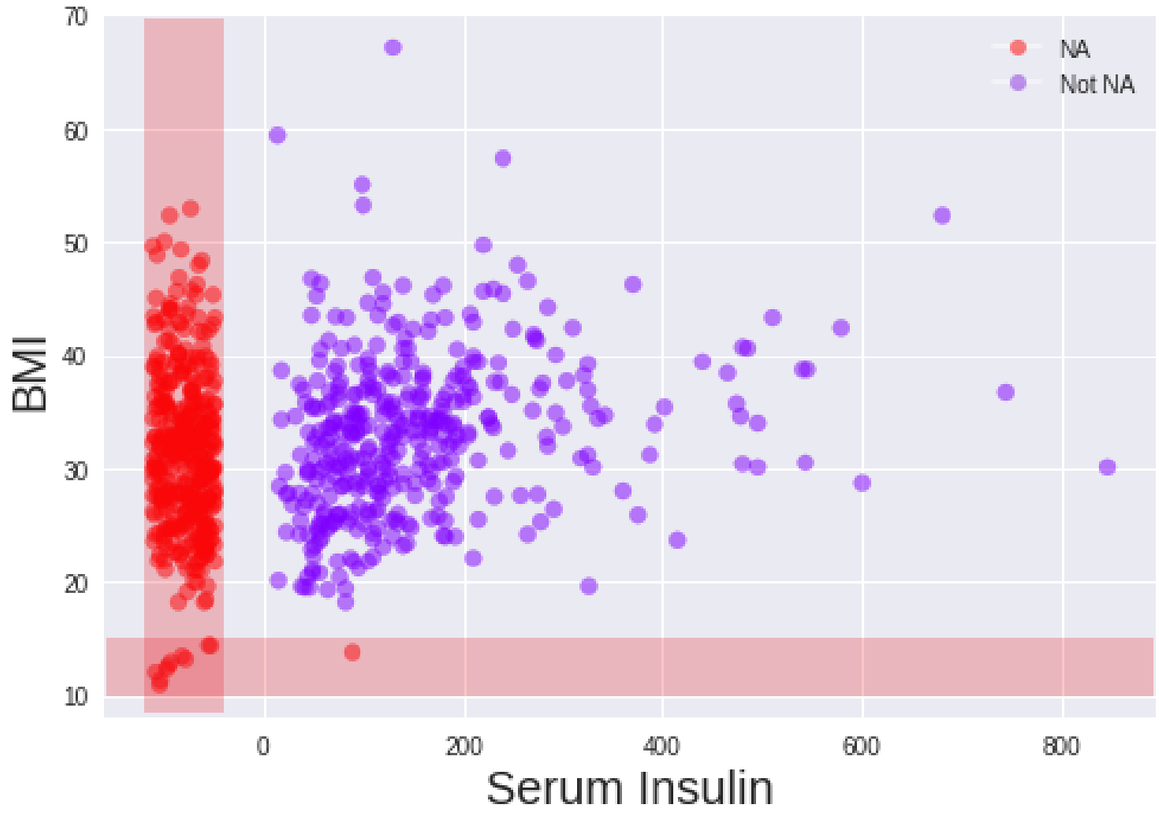
Filling dummy Values

```
from numpy.random import rand

BMI_null = diabetes['BMI'].isnull()
num_nulls = BMI_null.sum()

# Generate random values
dummy_values = rand(num_nulls)
# Shift to -2 & -1
dummy_values = dummy_values - 2
# Scale to 0.075 of Column Range
BMI_range = BMI.max() - BMI.min()
dummy_values = dummy_values * 0.075 * BMI_range
```



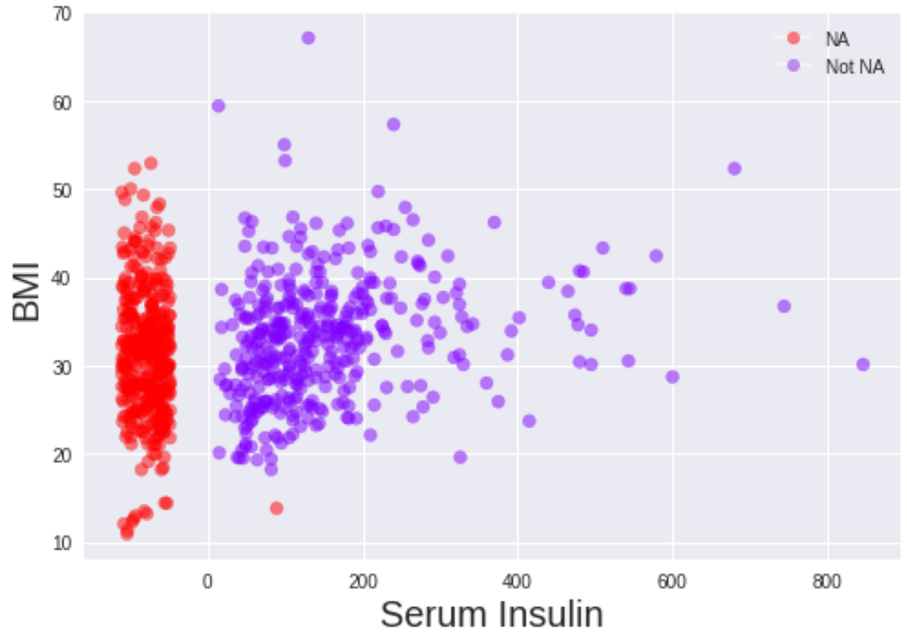
Filling dummy Values

```
from numpy.random import rand

BMI_null = diabetes['BMI'].isnull()
num_nulls = BMI_null.sum()

# Generate random values
dummy_values = rand(num_nulls)
# Shift to -2 & -1
dummy_values = dummy_values - 2
# Scale to 0.075 of Column Range
BMI_range = BMI.max() - BMI.min()
dummy_values = dummy_values * 0.075 * BMI_range
# Shift to Column Minimum
dummy_values = (rand(num_nulls) - 2)
        * 0.075 * BMI_range + BMI.min()
```



```
from numpy.random import rand

def fill_dummy_values(df, scaling_factor):
  # Create copy of dataframe
  df_dummy = df.copy(deep=True)
  # Iterate over each column
  for col in df_dummy:

    # Get column, column missing values and range
    col = df_dummy[col]
    col_null = col.isnull()
    num_nulls = col_null.sum()
    col_range = col.max() - col.min()

    # Shift and scale dummy values
    dummy_values = (rand(num_nulls) - 2)
    dummy_values = dummy_values * scaling_factor * col_range + col.min()

    # Return dummy values
    col[col_null] = dummy_values
  return df_dummy
```

```
# Create dummy dataframe
diabetes_dummy = fill_dummy_values(diabetes)
# Get missing values of both columns for coloring
nullity=diabetes.Serum_Insulin.isnull()+diabetes.BMI.isnull()
# Generate scatter plot
diabetes_dummy.plot(x='Serum_Insulin', y='BMI', kind='scatter', alpha=0.5,
                    c=nullity, cmap='rainbow')
```



Let's practice!

DEALING WITH MISSING DATA IN PYTHON

When and how to delete missing data

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

Types of deletions

1. Pairwise deletion
2. Listwise deletion

Note: Used when the values are MCAR.

Pairwise Deletion

`diabetes` DataFrame

Pregnant	Glucose	Diastolic_BP	...
6	148	72	...
5	NaN	80	...
1	89	66	...
1	NaN	74	...
...	...	...	...
8	183	64	...
6	NaN	68	...

768 rows × 9 columns

```
diabetes['Glucose'].mean()
```

```
121.687
```

```
diabetes.count()
```

```
763
```

```
diabetes['Glucose'].sum() /
       diabetes['Glucose'].count()
```

```
121.687
```

Listwise Deletion or Complete Case

`diabetes` DataFrame

Pregnant	Glucose	Diastolic_BP	...
6	148	72	...
5	NaN	80	...
1	89	66	...
1	NaN	74	...
...	...	...	...
8	183	64	...
6	NaN	68	...

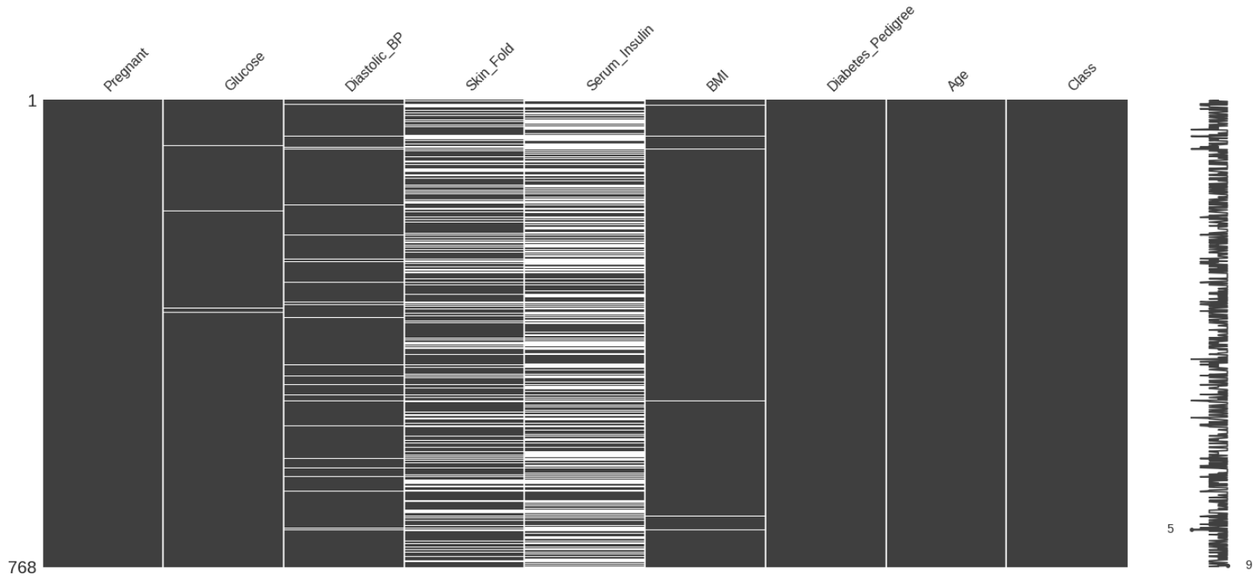
768 rows × 9 columns

```
diabetes.dropna(subset=['Glucose'],
                how='any',
                inplace=True)
```

Deletion in diabetes DataFrame

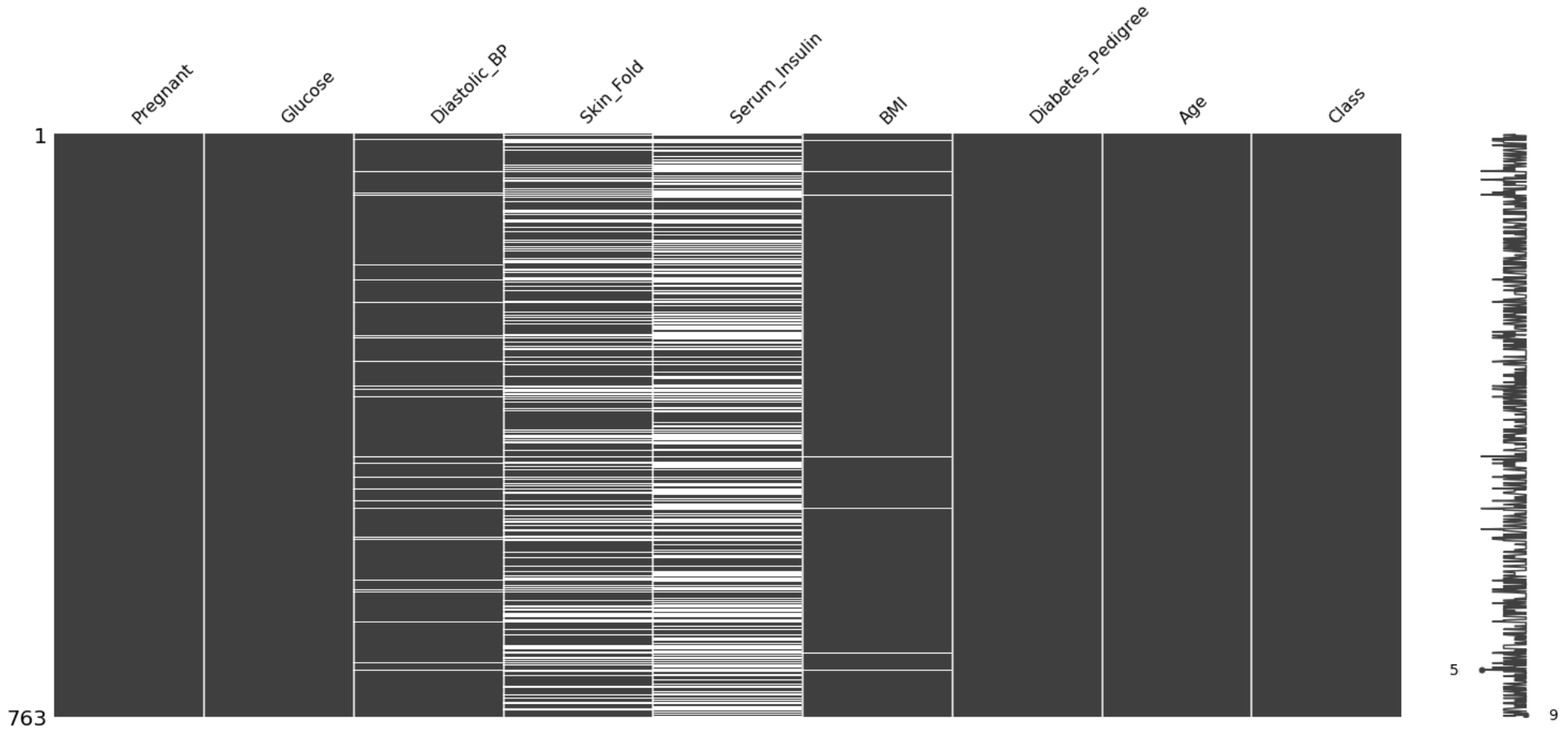
```
msno.matrix(diabetes)
diabetes['Glucose'].isnull().sum()
```

```
5
```



Deletion in diabetes DataFrame

```
diabetes.dropna(subset=["Glucose"], how='any', inplace=True)
msno.matrix(diabetes)
```

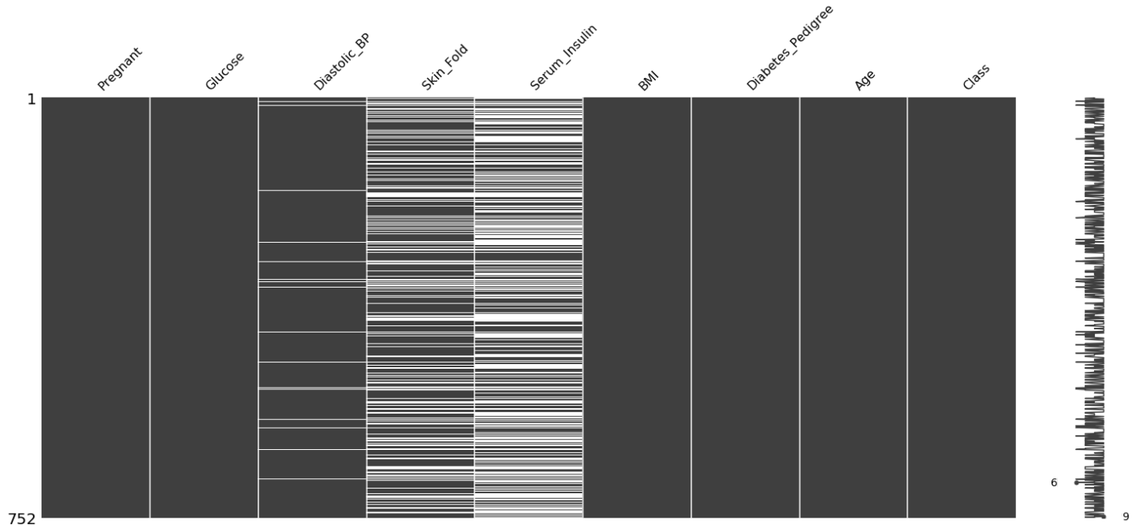


Deletion in diabetes DataFrame

```
diabetes['BMI'].isnull().sum()
```

```
11
```

```
diabetes.dropna(subset=["BMI"], how='any', inplace=True)
msno.matrix(diabetes)
```



Summary

- Pairwise deletion
- Listwise deletion
- Deletion is used only when values are MCAR

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Mean, median & mode imputations

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi

Deep Learning & Computer Vision Consultant

datacamp

Basic imputation techniques

- constant (e.g. 0)
- mean
- median
- mode or most frequent

Mean Imputation

```
from sklearn.impute import SimpleImputer
diabetes_mean = diabetes.copy(deep=True)
mean_imputer = SimpleImputer(strategy='mean')
```

Mean Imputation

```
from sklearn.impute import SimpleImputer
diabetes_mean = diabetes.copy(deep=True)
mean_imputer = SimpleImputer(strategy='mean')
diabetes_mean.iloc[:, :] = mean_imputer.fit_transform(diabetes_mean)
```

Median imputation

```
diabetes_median = diabetes.copy(deep=True)
median_imputer = SimpleImputer(strategy='median')
diabetes_median.iloc[:, :] = median_imputer.fit_transform(diabetes_median)
```

Mode imputation

```
diabetes_mode = diabetes.copy(deep=True)
mode_imputer = SimpleImputer(strategy='most_frequent')
diabetes_mode.iloc[:, :] = mode_imputer.fit_transform(diabetes_mode)
```

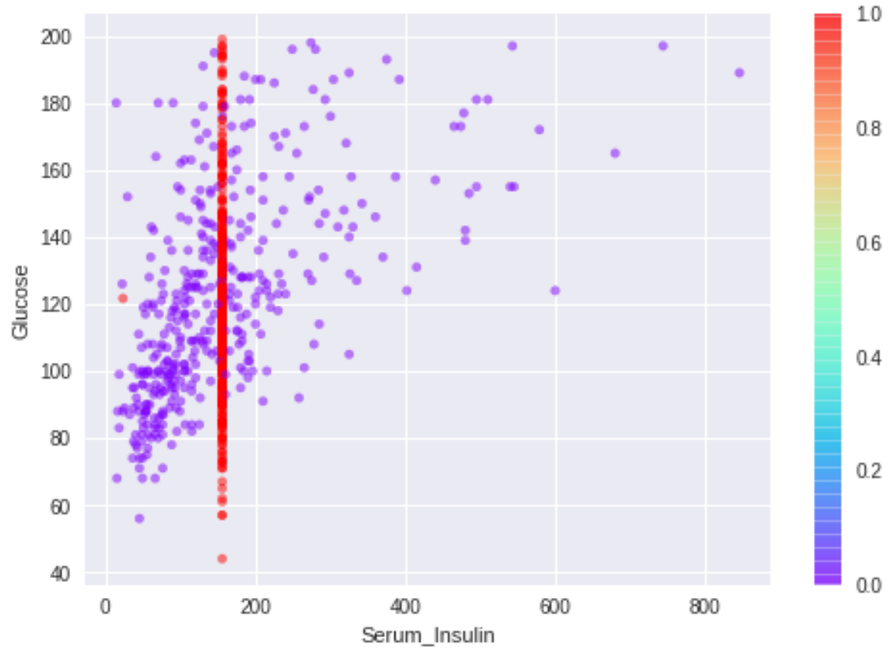
Imputing a constant

```
diabetes_constant = diabetes.copy(deep=True)
constant_imputer = SimpleImputer(strategy='constant', fill_value=0))
diabetes_constant.iloc[:, :] = constant_imputer.fit_transform(diabetes_constant)
```

Scatterplot of imputation

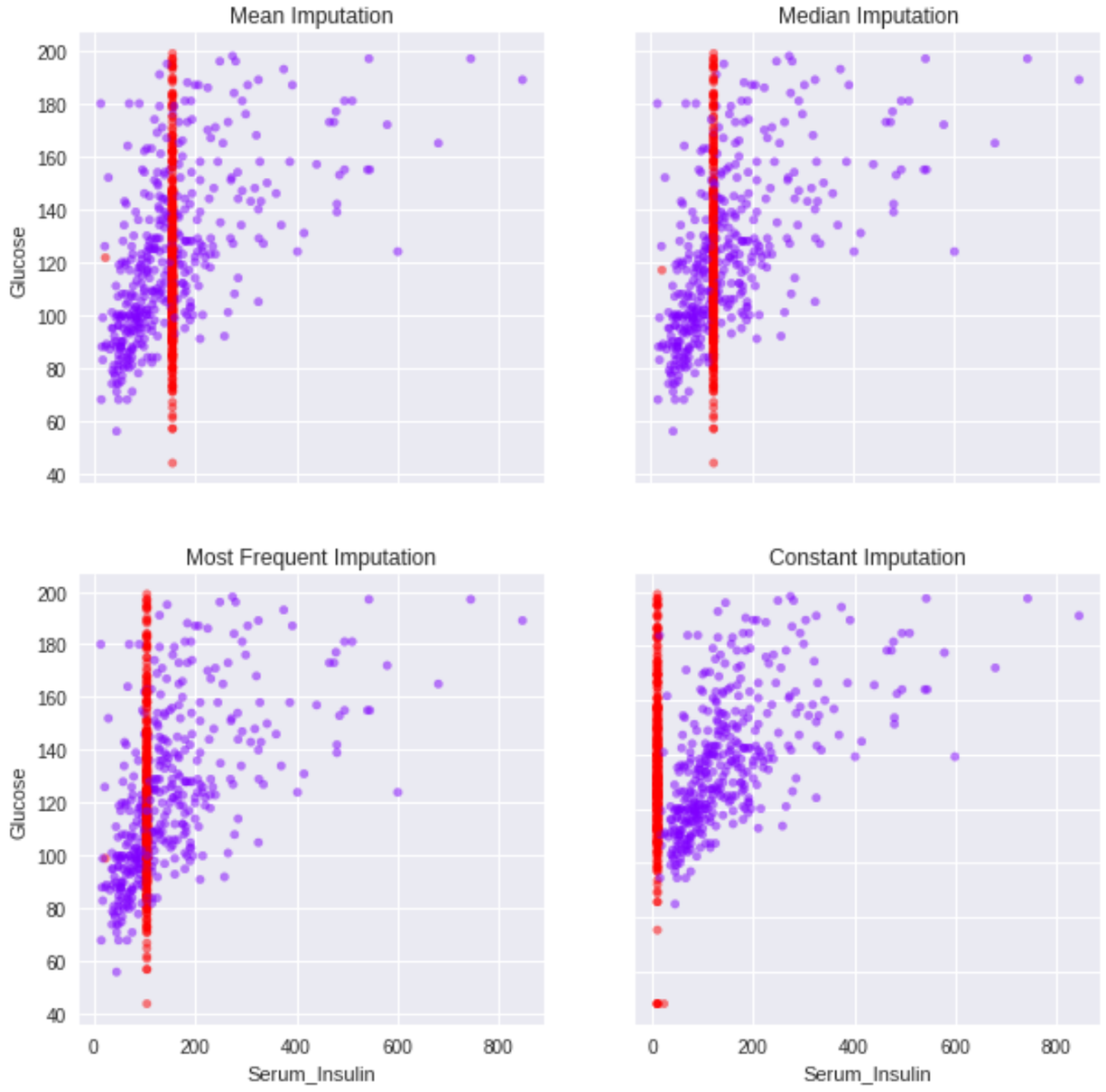
```
nullity = diabetes['Serum_Insulin'].isnull()+diabetes['Glucose'].isnull()
```

```
diabetes_mean.plot(x='Serum_Insulin', y='Glucose', kind='scatter', alpha=0.5,
                   c=nullity, cmap='rainbow', title='Mean Imputation')
```



Visualizing imputations

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
nullity = diabetes['Serum_Insulin'].isnull()+diabetes['Glucose'].isnull()
imputations = {'Mean Imputation': diabetes_mean,
               'Median Imputation': diabetes_median,
               'Most Frequent Imputation': diabetes_mode,
               'Constant Imputation': diabetes_constant}
for ax, df_key in zip(axes.flatten(), imputations):
    imputations[df_key].plot(x='Serum_Insulin', y='Glucose', kind='scatter',
                             alpha=0.5, c=nullity, cmap='rainbow', ax=ax,
                             colorbar=False, title=df_key)
```


Mean Imputation
Median Imputation
Most Frequent Imputation
Constant Imputation
Glucose
Serum_Insulin
200
180
160
140
120
100
80
60
40
0
200
400
600
800

Summary

You learned to

- Impute with statistical parameters like mean, median and mode
- Graphically compare the imputations
- Analyze the imputations

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Imputing time-series data

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

Airquality Dataset

```
import pandas as pd
airquality = pd.read_csv('air-quality.csv', parse_dates='Date',
                                 index_col='Date')
airquality.head()
```

```
             Ozone    Solar    Wind    Temp
Date
1976-05-01    41.0    190.0     7.4    67
1976-05-02    36.0    118.0     8.0    72
1976-05-03    12.0    149.0    12.6    74
1976-05-04    18.0    313.0    11.5    62
1976-05-05     NaN      NaN    14.3    56
```

Airquality Dataset

```
airquality.isnull().sum()
```

```
Ozone    37
Solar     7
Wind      0
Temp      0
dtype: int64
```

```
airquality.isnull.mean() * 100
```

```
Ozone    24.183007
Solar     4.575163
Wind      0.000000
Temp      0.000000
dtype: float64
```

The .fillna() method

The attribute `method` in `.fillna()` can be set to

- `'ffill'` or `'pad'`
- `'bfill'` or `'backwardfill'`

Ffill method

- Replace `NaN` s with last observed value
- `pad` is the same as `'ffill'`

```
airquality.fillna(method='ffill', inplace=True)
```

```
airquality['Ozone'][30:40]
```

```
Date
1976-05-31    37.0
1976-06-01     NaN
1976-06-02     NaN
1976-06-03     NaN
1976-06-04     NaN
1976-06-05     NaN
1976-06-06     NaN
1976-06-07    29.0
1976-06-08     NaN
1976-06-09    71.0
```

```
airquality.fillna(method='ffill',
                  inplace=True)
airquality['Ozone'][30:40]
```

```
Date
1976-05-31    37.0
1976-06-01    37.0
1976-06-02    37.0
1976-06-03    37.0
1976-06-04    37.0
1976-06-05    37.0
1976-06-06    37.0
1976-06-07    29.0
1976-06-08    29.0
1976-06-09    71.0
```

Bfill method

- Replace `NaN` s with next observed value
- `backfill` is the same as `'bfill'`

```
df.fillna(method='bfill', inplace=True)
```

```
airquality['Ozone'][30:40]
```

```
Date          Ozone
1976-05-31     37.0
1976-06-01      NaN
1976-06-02      NaN
1976-06-03      NaN
1976-06-04      NaN
1976-06-05      NaN
1976-06-06      NaN
1976-06-07     29.0
1976-06-08      NaN
1976-06-09     71.0
```

```
airquality.fillna(method='bfill',
                                inplace=True)
airquality['Ozone'][30:40]
```

```
Date          Ozone
1976-05-31     37.0
1976-06-01     29.0
1976-06-02     29.0
1976-06-03     29.0
1976-06-04     29.0
1976-06-05     29.0
1976-06-06     29.0
1976-06-07     29.0
1976-06-08     71.0
1976-06-09     71.0
```

The .interpolate() method

- The `.interpolate()` method extends the sequence of values to the missing values

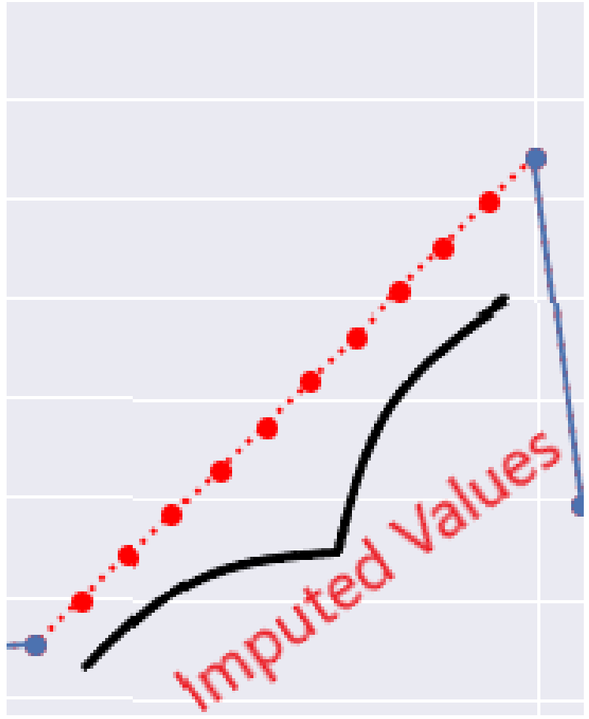
The attribute `method` in `.interpolate()` can be set to

- `'linear'`
- `'quadratic'`
- `'nearest'`

Linear interpolation

- Impute linearly or with equidistant values

```
df.interpolate(method='linear', inplace=True)
```



```
airquality['Ozone'][30:40]
```

```
Date          Ozone
1976-05-31     37.0
1976-06-01      NaN
1976-06-02      NaN
1976-06-03      NaN
1976-06-04      NaN
1976-06-05      NaN
1976-06-06      NaN
1976-06-07     29.0
1976-06-08      NaN
1976-06-09     71.0
```

```
airquality.interpolate(
           method='linear', inplace=True)
airquality['Ozone'][30:40]
```

```
Date          Ozone
1976-05-31     37.0
1976-06-01     35.9
1976-06-02     34.7
1976-06-03     33.6
1976-06-04     32.4
1976-06-05     31.3
1976-06-06     30.1
1976-06-07     29.0
1976-06-08     50.0
1976-06-09     71.0
```

Quadratic interpolation

- Impute the values quadratically

```
df.interpolate(method='quadratic', inplace=True)
```

```
airquality['Ozone'][30:39]
```

```
            Ozone
Date
1976-05-31   37.0
1976-06-01    NaN
1976-06-02    NaN
1976-06-03    NaN
1976-06-04    NaN
1976-06-05    NaN
1976-06-06    NaN
1976-06-07   29.0
1976-06-08    NaN
```

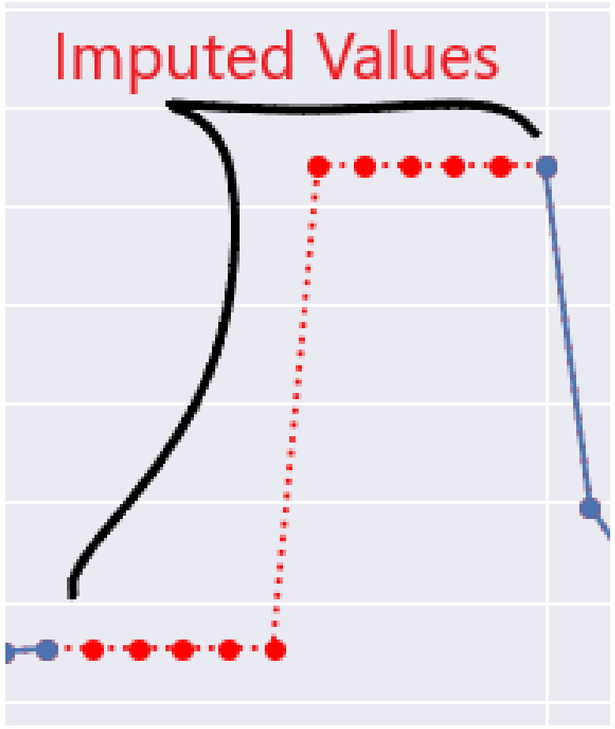
```
airquality.interpolate(
  method='quadratic', inplace=True)
airquality['Ozone'][30:39]
```

```
            Ozone
Date
1976-05-31   37.0
1976-06-01  -38.4
1976-06-02  -79.4
1976-06-03  -85.9
1976-06-04  -62.4
1976-06-06   -2.8
1976-06-07   29.0
1976-06-08   62.2
```

Nearest value imputation

- Impute with the nearest observable value

```
df.interpolate(method='nearest', inplace=True)
```



```
airquality['Ozone'][30:39]
```

```
Date          Ozone
1976-05-31     37.0
1976-06-01      NaN
1976-06-02      NaN
1976-06-03      NaN
1976-06-04      NaN
1976-06-05      NaN
1976-06-06      NaN
1976-06-07     29.0
1976-06-08      NaN
```

```
airquality.interpolate(
  method='nearest', inplace=True)
airquality['Ozone'][30:39]
```

```
Date          Ozone
1976-05-31     37.0
1976-06-01     37.0
1976-06-02     37.0
1976-06-03     37.0
1976-06-04     29.0
1976-06-05     29.0
1976-06-06     29.0
1976-06-07     29.0
1976-06-08     29.0
```

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Visualizing time-series imputations

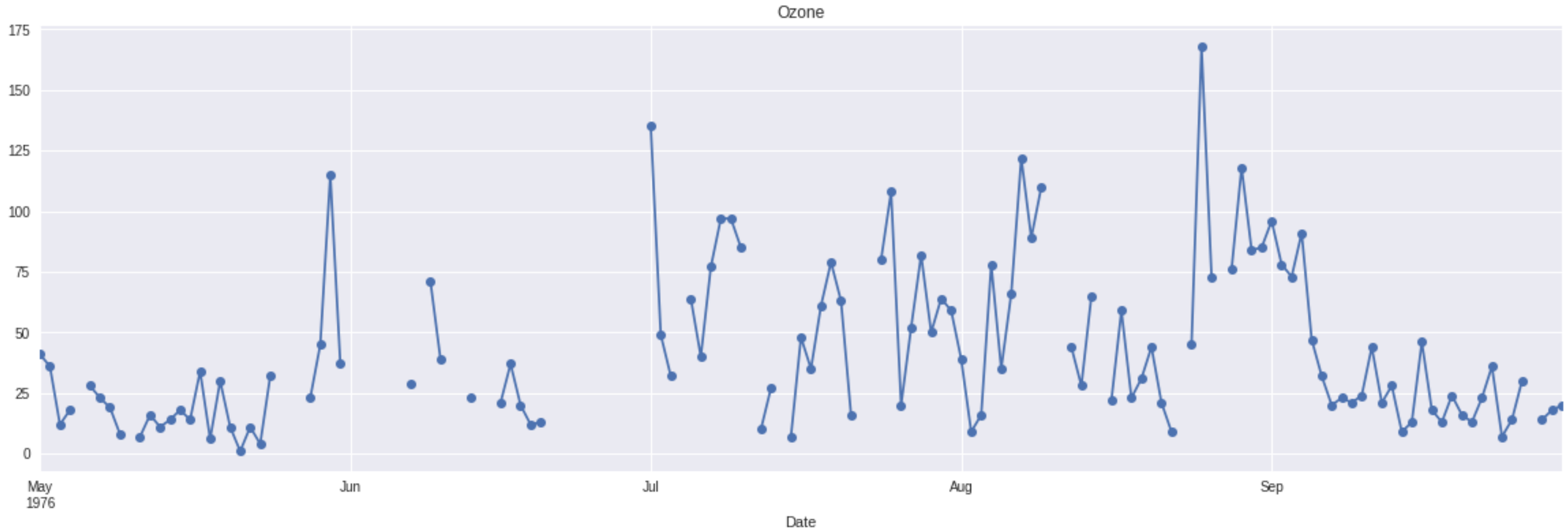
DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Learning

datacamp

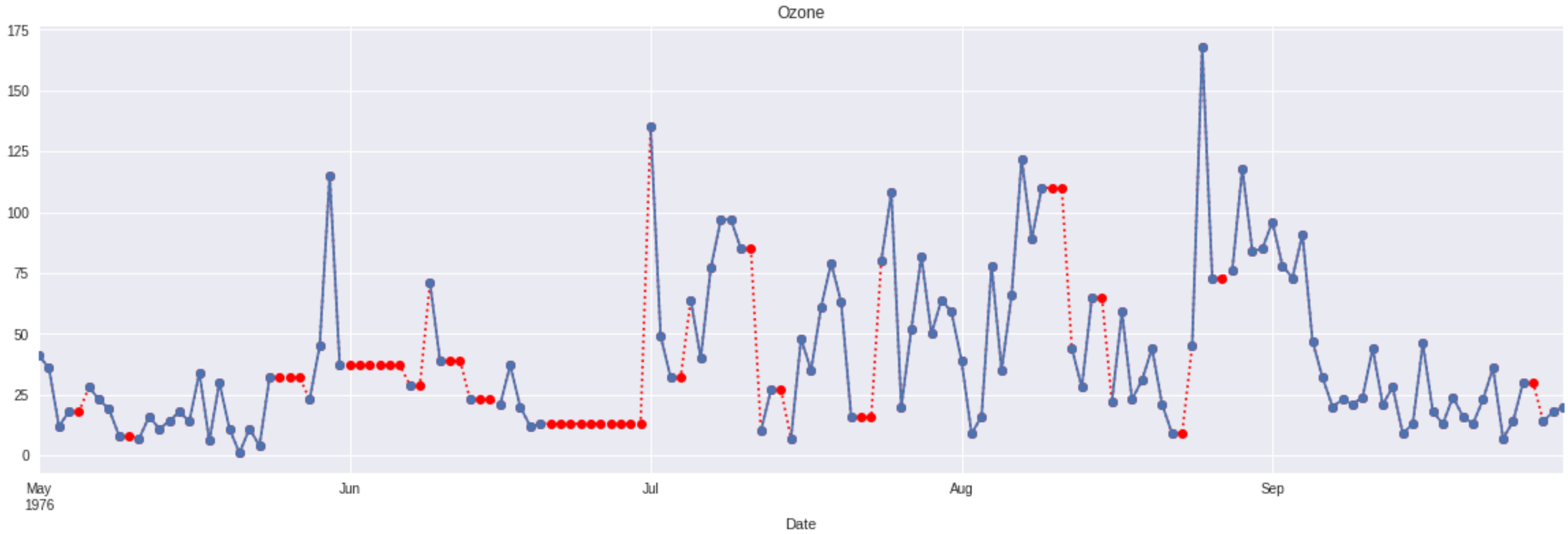
Air quality time-series plot

```
airquality['Ozone'].plot(title='Ozone', marker='o', figsize=(30, 5))
```



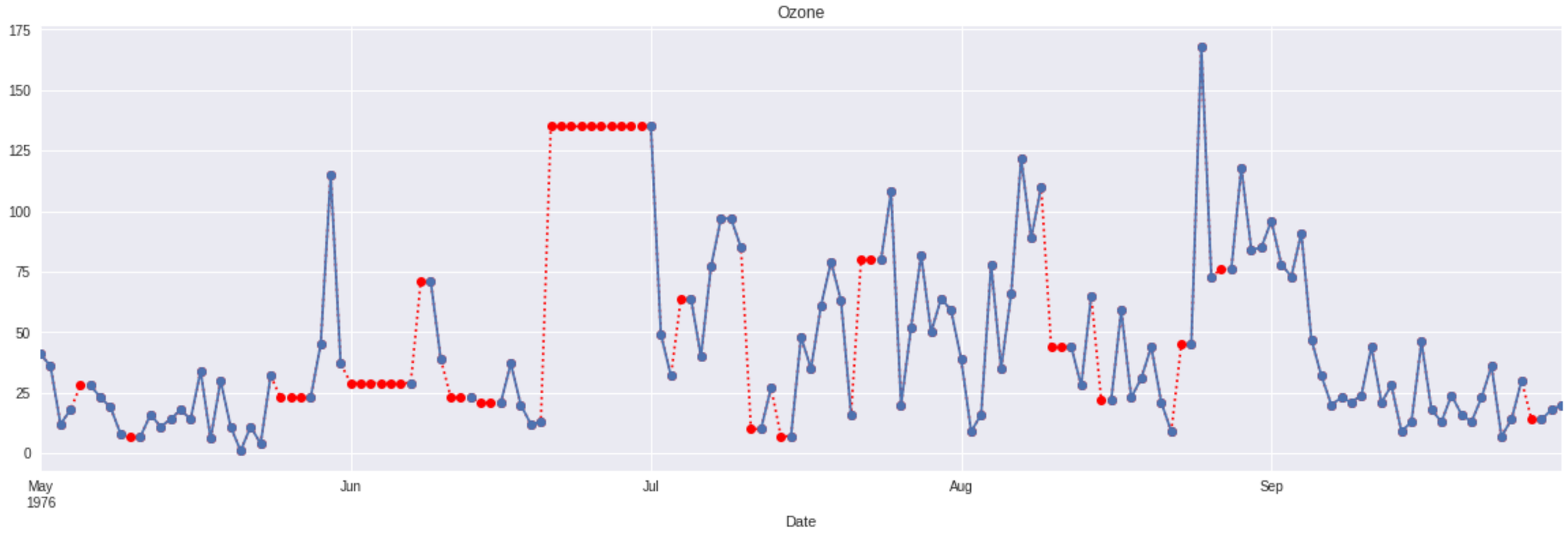
Ffill Imputation

```
ffill_imp['Ozone'].plot(color='red', marker='o', linestyle='dotted', figsize=(30, 5))
airquality['Ozone'].plot(title='Ozone', marker='o')
```



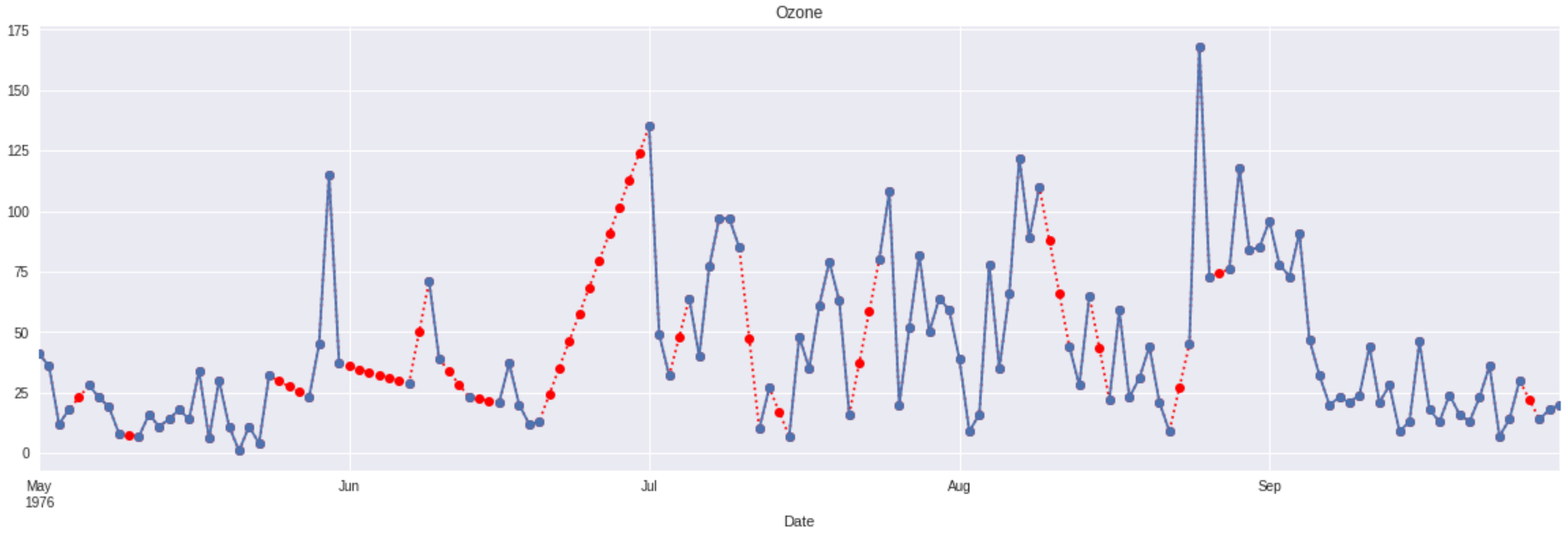
Bfill Imputation

```
bfill_imp['Ozone'].plot(color='red', marker='o', linestyle='dotted', figsize=(30, 5))
airquality['Ozone'].plot(title='Ozone', marker='o')
```



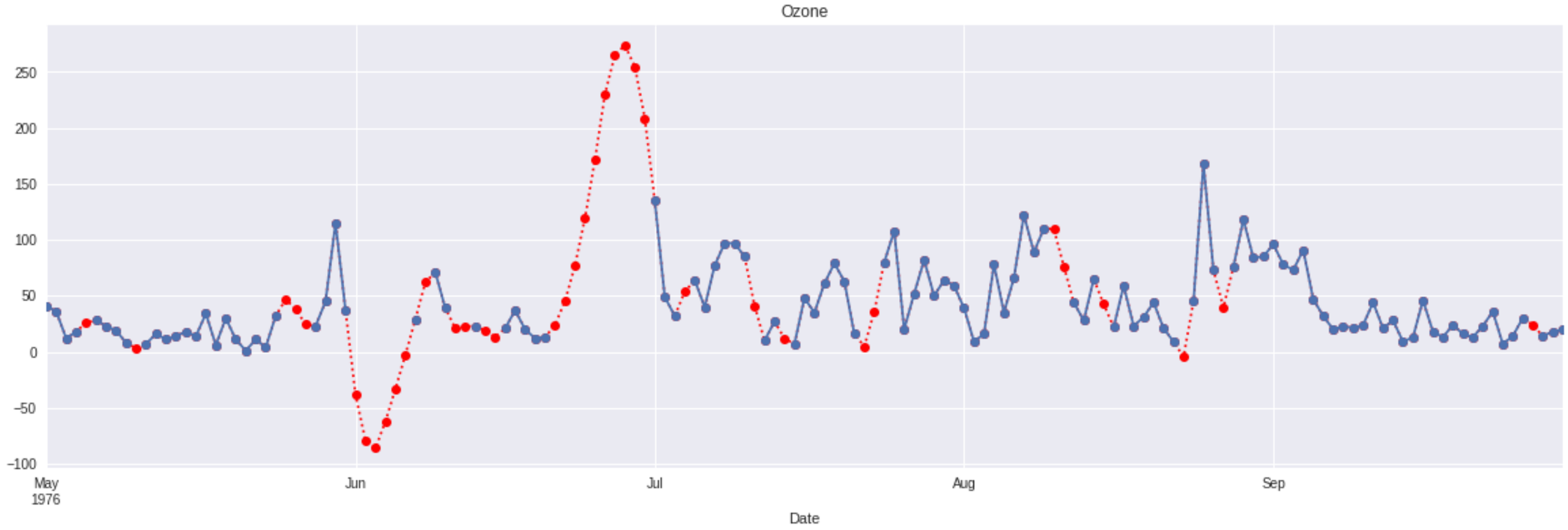
Linear Interpolation

```
linear_interp['Ozone'].plot(color='red', marker='o', linestyle='dotted', figsize=(30, 5
airquality['Ozone'].plot(title='Ozone', marker='o')
```



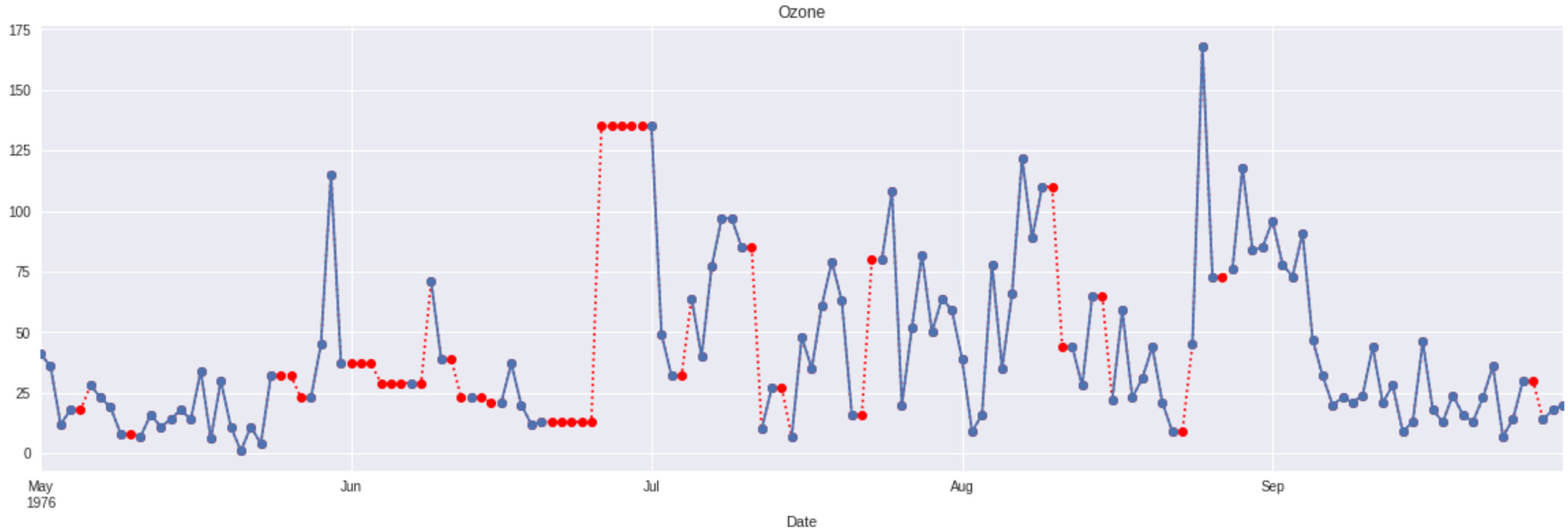
Quadratic Interpolation

```
quadratic_interp['Ozone'].plot(color='red', marker='o', linestyle='dotted', figsize=(30, 5))
airquality['Ozone'].plot(title='Ozone', marker='o')
```



Nearest Interpolation

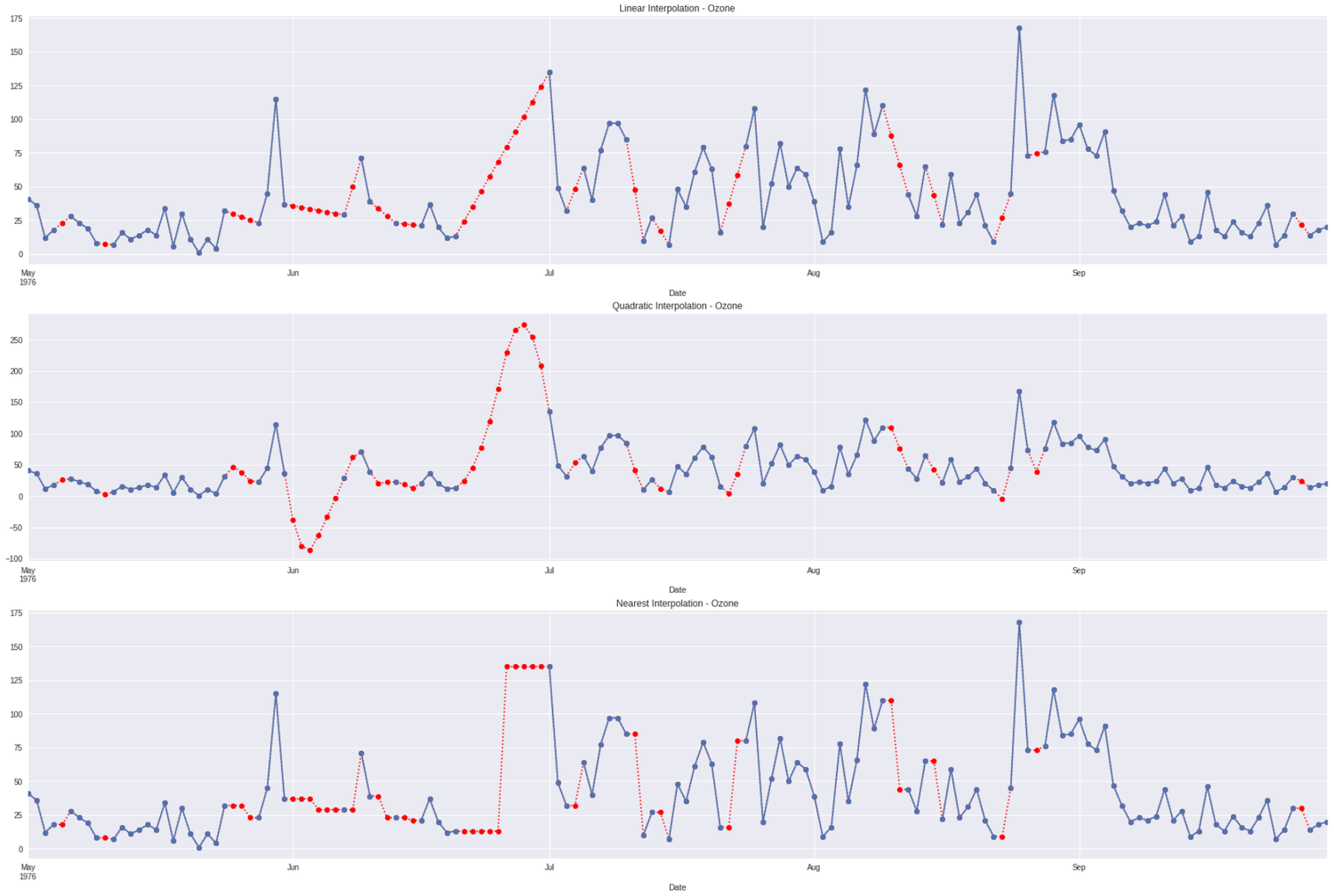
```
nearest_interp['Ozone'].plot(color='red', marker='o', linestyle='dotted', figsize=(30, 
airquality['Ozone'].plot(title='Ozone', marker='o')
```



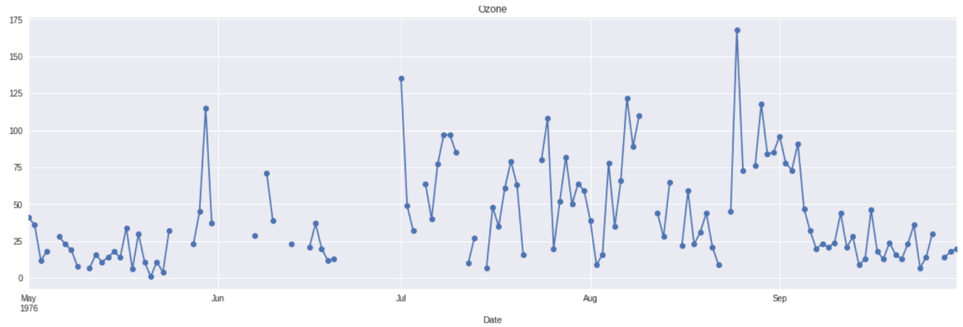
A comparison of the interpolations

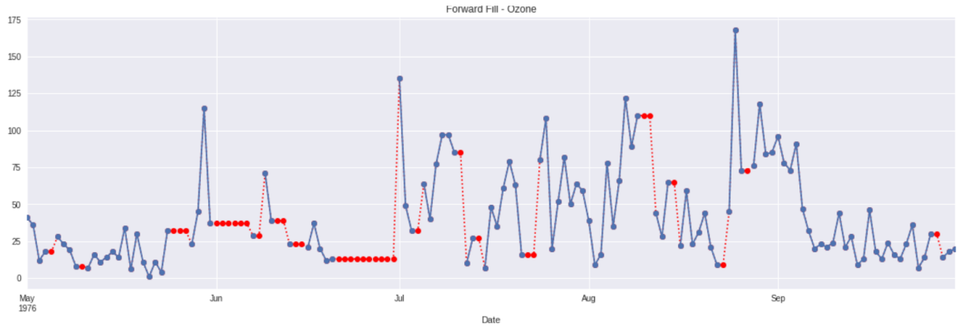
```
# Create subplots
fig, axes = plt.subplots(3, 1, figsize=(30, 20))

# Create interpolations dictionary
interpolations = {'Linear Interpolation': linear_interp,
                  'Quadratic Interpolation': quadratic_interp,
                  'Nearest Interpolation': nearest_interp}

# Visualize each interpolation
for ax, df_key in zip(axes, interpolations):
        interpolations[df_key].Ozone.plot(color='red', marker='o',
                                          linestyle='dotted', ax=ax)
        airquality.Ozone.plot(title=df_key + ' - Ozone', marker='o', ax=ax)
```

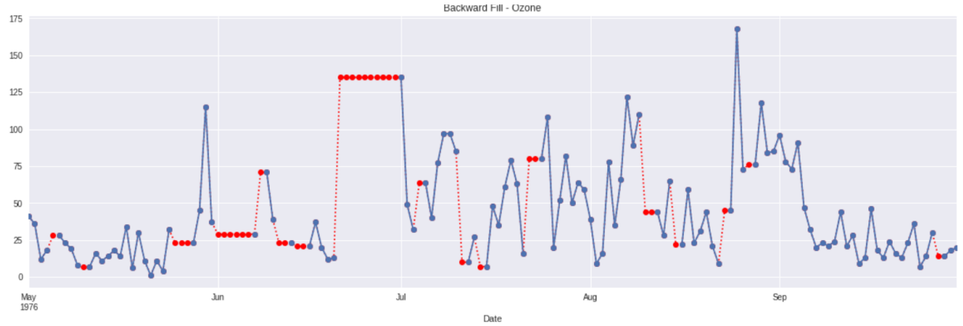
A comparison of the interpolations

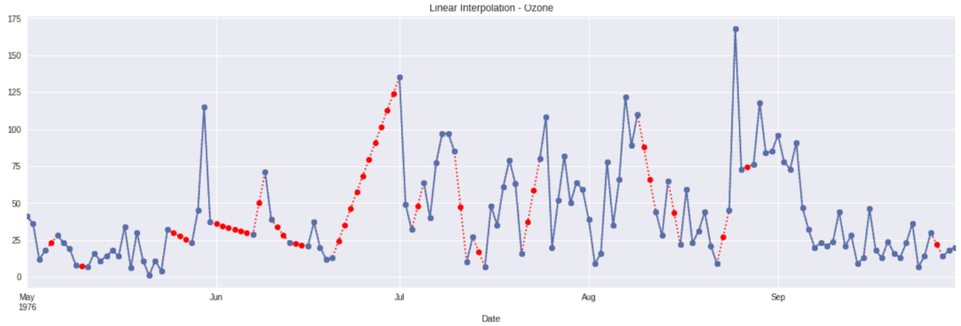


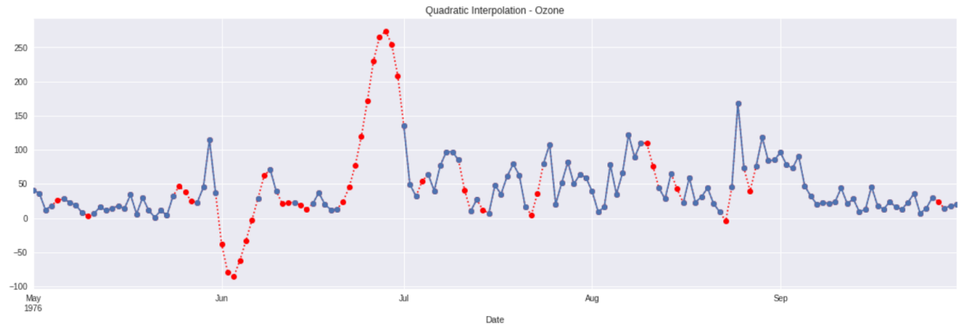
A comparison of imputation techniques

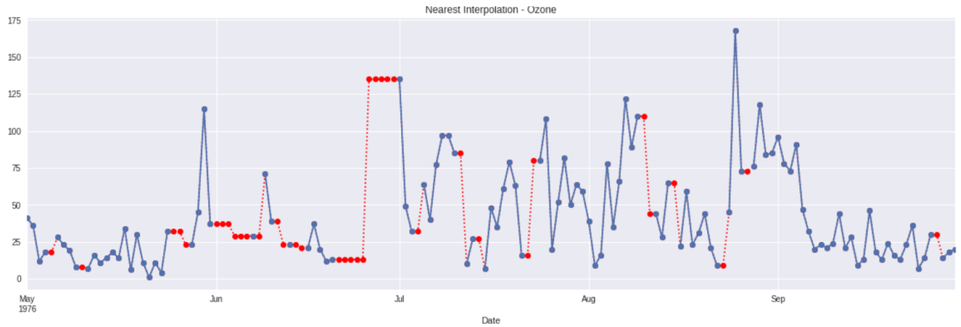












Summary

- Time-series plot of imputed DataFrame
- Comparison of imputations

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Imputing using fancyimpute

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

fancyimpute package

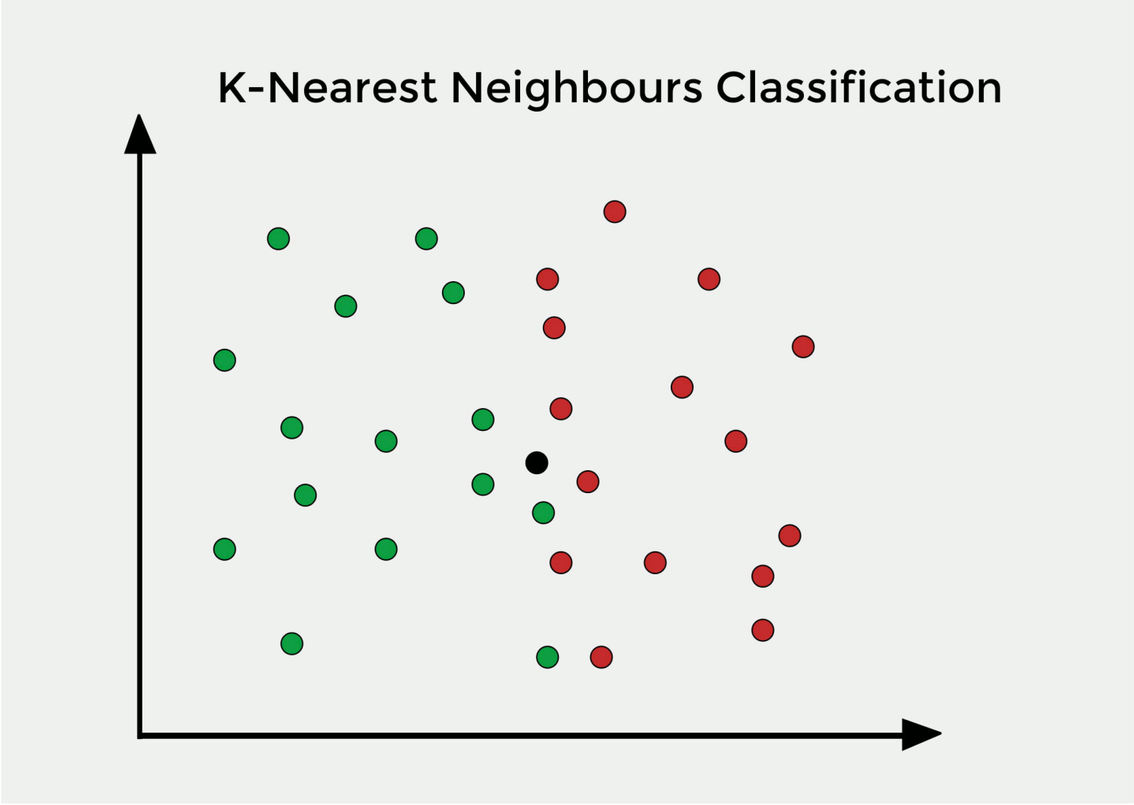
- Package contains advanced techniques
- Uses machine learning algorithms to impute missing values
- Uses other columns to predict the missing values and impute them

Fancyimpute imputation techniques

- KNN or K-Nearest Neighbor
- MICE or Multiple Imputation by Chained Equations

K-Nearest Neighbor Imputation

- Select K nearest or similar data points using all the non-missing features
- Take average of the selected data points to fill in the missing feature



K-Nearest Neighbor Imputation

```
from fancyimpute import KNN
knn_imputer = KNN()
diabetes_knn = diabetes.copy(deep=True)
diabetes_knn.iloc[:, :] = knn_imputer.fit_transform(diabetes_knn)
```

Multiple Imputations by Chained Equations (MICE)

- Perform multiple regressions over random sample of the data
- Take average of the multiple regression values
- Impute the missing feature value for the data point

Multiple Imputations by Chained Equations(MICE)

```
from fancyimpute import IterativeImputer
MICE_imputer = IterativeImputer()
diabetes_MICE = diabetes.copy(deep=True)
diabetes_MICE.iloc[:, :] = MICE_imputer.fit_transform(diabetes_MICE)
```

Summary

- Using Machine Learning techniques to impute missing values
- KNN finds most similar points for imputing
- MICE performs multiple regression for imputing
- MICE is a very robust model for imputation

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Imputing categorical values

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

Complexity with categorical values

- Most categorical values are strings
- Cannot perform operations on strings
- Necessity to convert/encode strings to numeric values and impute

Conversion techniques

ONE-HOT ENCODER

Color	Color_Red	Color_Green	Color_Blue
Red	1	0	0
Green	0	1	0
Blue	0	0	1
Red	1	0	0
Blue	0	0	1
Blue	0	0	1

ORDINAL ENCODER

Color	Value
Red	0
Green	1
Blue	2
Red	0
Blue	2
Blue	2

Imputation techniques

- Fill with most frequent category
- Impute using statistical models like KNN

Users profile data

```
users = pd.read_csv('userprofile.csv')
users.head()
```

```
    smoker    drink_level    dress_preference ambience    hijos        activity     budg
0    False    abstemious        informal      family    independent    student      medi
1    False    abstemious        informal      family    independent    student      low
2    False    social drinker    formal        family    independent    student      low
3    False    abstemious        informal      family    independent    professional medi
4    False    abstemious     no preference    family    independent    student      medi
```

Ordinal Encoding

```
from sklearn.preprocessing import OrdinalEncoder

# Create Ordinal Encoder
ambience_ord_enc = OrdinalEncoder()

# Select non-null values in ambience
ambience = users['ambience']
ambience_not_null = ambience[ambience.notnull()]
reshaped_vals = ambience_not_null.values.reshape(-1, 1)
# Encode the non-null values of ambience
encoded_vals = ambience_ord_enc.fit_transform(reshaped_vals)
# Replace the ambience column with ordinal values
users.loc[ambience.notnull(), 'ambience'] = np.squeeze(encoded_vals)
```

Ordinal Encoding

```
# Create dictionary for Ordinal encoders
ordinal_enc_dict = {}

# Loop over columns to encode
for col_name in users:
    # Create ordinal encoder for the column
    ordinal_enc_dict[col_name] = OrdinalEncoder()

    # Select the nin-null values in the column
    col = users[col_name]
    col_not_null = col[col.notnull()]
    reshaped_vals = col_not_null.values.reshape(-1, 1)

    # Encode the non-null values of the column
    encoded_vals = ordinal_enc_dict[col_name].fit_transform(reshaped_vals)

    # Replace the values in the column with ordinal values
```

Imputing with KNN

```
users_KNN_imputed = users.copy(deep=True)

# Create MICE imputer
KNN_imputer = KNN()

users_KNN_imputed.iloc[:, :] = np.round(KNN_imputer.fit_transform(imputed))

for col in imputed:
    reshaped_col = imputed[col].values.reshape(-1, 1)
    users_KNN_imputed[col] = ordinal_enc[col].inverse_transform(reshaped_col)
```

Summary

Steps to impute categorical values

- Convert non-missing categorical columns to ordinal values
- Impute the missing values in the ordinal DataFrame
- Convert back from ordinal values to categorical values

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Evaluation of different imputation techniques

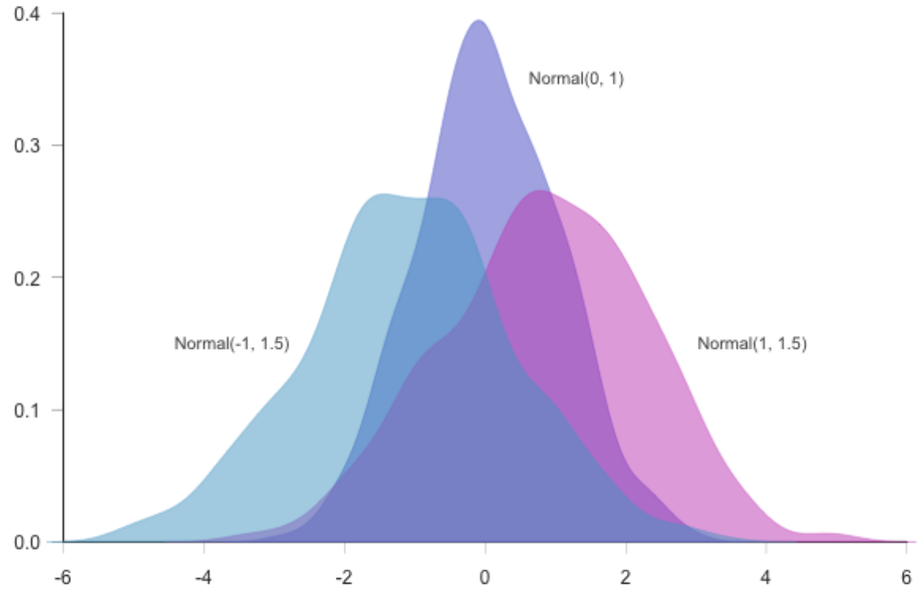
DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

datacamp

Evaluation techniques

- Imputations are used to improve model performance.
- Imputation with maximum machine learning model performance is selected.
- Density plots explain the distribution in the data.
- A very good metric to check bias in the imputations.



Fit a linear model for statistical summary

```
import statsmodels.api as sm


diabetes_cc = diabetes.dropna(how='any')
X = sm.add_constant(diabetes_cc.iloc[:, :-1])
y = diabetes_cc['Class']
lm = sm.OLS(y, X).fit()
```

```
print(lm.summary())
```

```
Summary:                            OLS Regression Results
==============================================================================
Dep. Variable:                  Class   R-squared:                       0.346
Model:                            OLS   Adj. R-squared:                  0.332
Method:                 Least Squares   F-statistic:                     25.30
Date:                Wed, 10 Jul 2019   Prob (F-statistic):           2.65e-31
Time:                        15:03:19   Log-Likelihood:                -177.76
No. Observations:                 392   AIC:                             373.5
Df Residuals:                     383   BIC:                             409.3
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
<hr />--------------------------------------------------------------------------------
const             -1.1027      0.144     -7.681      0.000      -1.385      -0.820
Pregnant           0.0130      0.008      1.549      0.122      -0.003       0.029
Glucose            0.0064      0.001      7.855      0.000       0.005       0.008
Diastolic_BP    5.465e-05      0.002      0.032      0.975      -0.003       0.003
Skin_Fold          0.0017      0.003      0.665      0.506      -0.003       0.007
Serum_Insulin     -0.0001      0.000     -0.603      0.547      -0.001       0.000
BMI                0.0093      0.004      2.391      0.017       0.002       0.017
Diabetes_Pedigree  0.1572      0.058      2.708      0.007       0.043       0.271
Age                0.0059      0.003      2.109      0.036       0.000       0.011
```

R-squared and Coefficients

```
lm.rsquared_adj
```

```
0.33210
```

```
lm.params
```

```
const               -1.102677
Pregnant             0.012953
Glucose              0.006409
Diastolic_BP         0.000055
Skin_Fold            0.001678
Serum_Insulin       -0.000123
BMI                  0.009325
Diabetes_Pedigree    0.157192
Age                  0.005878
dtype: float64
```

Fit linear model on different imputed DataFrames

```
# Mean Imputation
X = sm.add_constant(diabetes_mean_imputed.iloc[:, :-1])
y = diabetes['Class']
lm_mean = sm.OLS(y, X).fit()
# KNN Imputation
X = sm.add_constant(diabetes_knn_imputed.iloc[:, :-1])
lm_KNN = sm.OLS(y, X).fit()
# MICE Imputation
X = sm.add_constant(diabetes_mice_imputed.iloc[:, :-1])
lm_MICE = sm.OLS(y, X).fit()
```

Comparing R-squared of different imputations

```
print(pd.DataFrame({'Complete': lm.rsquared_adj,
                    'Mean Imp.': lm_mean.rsquared_adj,
                    'KNN Imp.': lm_KNN.rsquared_adj,
                    'MICE Imp.': lm_MICE.rsquared_adj},
                   index=['R_squared_adj']))
```

```
                Complete   Mean Imp.    KNN Imp.   MICE Imp.
R_squared_adj   0.332108    0.313781    0.316543    0.317679
```

Comparing coefficients of different imputations

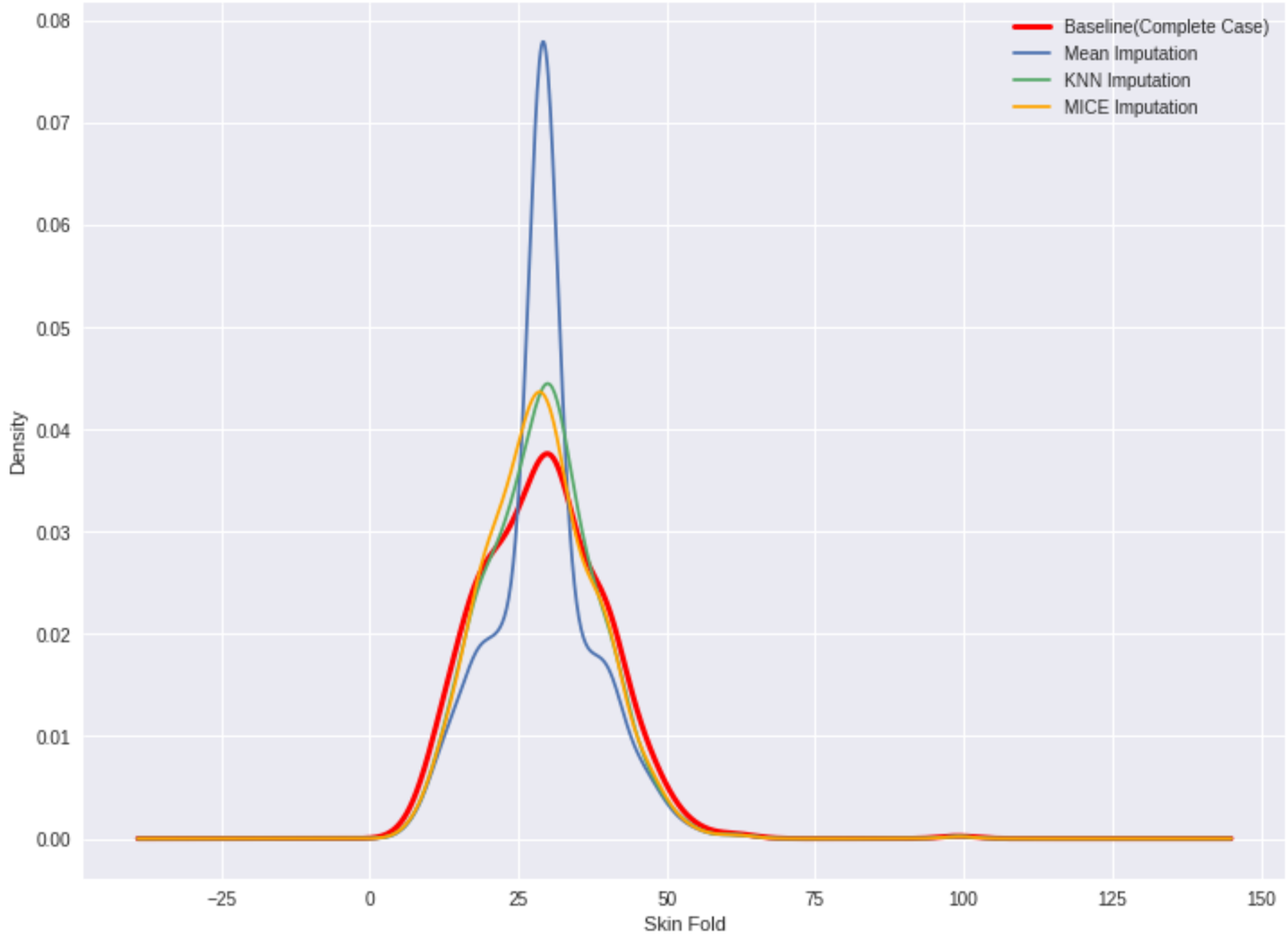
```
print(pd.DataFrame({'Complete': lm.params,
                    'Mean Imp.': lm_mean.params,
                    'KNN Imp.': lm_KNN.params,
                    'MICE Imp.': lm_MICE.params}))
```

```
                    Complete   Mean Imp.    KNN Imp.   MICE Imp.
const              -1.102677   -1.024005   -1.028035   -1.050023
Pregnant            0.012953    0.020693    0.020047    0.020295
Glucose             0.006409    0.006467    0.006614    0.006871
Diastolic_BP        0.000055   -0.001137   -0.001196   -0.001317
Skin_Fold           0.001678    0.000193    0.001626    0.000807
Serum_Insulin      -0.000123   -0.000090   -0.000147   -0.000227
BMI                 0.009325    0.014376    0.013239    0.014203
Diabetes_Pedigree   0.157192    0.129282    0.128038    0.129056
Age                 0.005878    0.002092    0.002046    0.002097
```

Comparing density plots

```
diabetes_cc['Skin_Fold'].plot(kind='kde', c='red', linewidth=3)
diabetes_mean_imputed['Skin_Fold'].plot(kind='kde')
diabetes_knn_imputed['Skin_Fold'].plot(kind='kde')
diabetes_mice_imputed['Skin_Fold'].plot(kind='kde')

labels = ['Baseline (Complete Case)', 'Mean Imputation', 'KNN Imputation',
          'MICE Imputation']
plt.legend(labels)
plt.xlabel('Skin Fold')
```

Comparing density plots



Summary

- Applying linear model from the statsmodels package
- Comparing the coefficients and standard errors
- Comparing density plots

Let's practice!

DEALING WITH MISSING DATA IN PYTHON

Conclusion

DEALING WITH MISSING DATA IN PYTHON

Suraj Donthi
Deep Learning & Computer Vision Consultant

Chapter 1

- Null Value operations
- Detecting missing values
- Replacing missing values
- Analyzing amount of missingness

Chapter 2

- Types of missingness
  - MCAR
  - MAR
  - MNAR
- Correlations of missingness
  - Heatmaps
  - Dendrograms
- Visualize missingness across a variable
- Deleting missing values

Chapter 3

- Imputation techniques
- Treating time-series data
- Graphical comparison of imputed time-series data

Chapter 4

- Advanced imputation techniques
  - KNN
  - MICE
- Imputing categorical data
- Evaluating and comparing the different imputations

Congratulations!!

DEALING WITH MISSING DATA IN PYTHON