# To explore the data visualization and data preprocesing

# Features & Its Types

 A **feature** is an individual measurable property or characteristic of a phenomenon being observed. Choosing informative, discriminating and independent **features** is a crucial step for effective algorithms in pattern recognition, classification and regression.

There are two types of features:

**Continuous Features**

A measurable difference exists between the values continuous features take on.Continuous variables are variables that can have an infinite number of possible values, as opposed to discrete variables which can only have a specified range of values.  Also continuous features are usually a subset of all real numbers. Some example features are:

* Time
* Distance
* Cost
* Temperature

**Categorical Features**

With categorical features, there is a specified number of discrete, possible feature values. These values may or may not have an ordering to them. If they do have a natural ordering, they are called ordinal categorical features. Otherwise if there is no intrinsic ordering, they are called nominal categorical features.

**Nominal**

* Car Models
* Colors
* TV Shows

**Ordinal**

* High-Medium-Low
* 1-10 Years Old, 11-20 Years Old, 30-40 Years Old
* Happy, Neutral, Sad

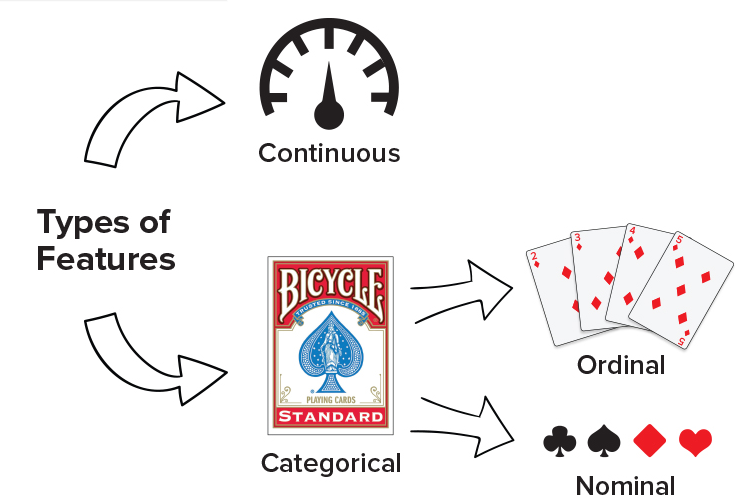
s

Figure 3.1: Types of features

## Visualizations

One of the most rewarding and useful things you can do to understand your data is to visualize it in a pictorial format. Visualizing your data allows you to interact with it, analyze it in a straightforward way, and identify new patterns, making your would-be complex data more accessible and understandable. The way our brains processes visuals like shapes, colors, and lengths makes looking at charts and graphs more intuitive for us than poring over spreadsheets.

### Matplotlib

MatPlotLib is a Python data visualization tool that supports 2D and 3D rendering, animation, UI design, event handling, and more. It only requires you pass in your data and some display parameters and then takes care of all of the rasterization implementation details. For the most part, you will be interacting with MatPlotLib's Pyplot functionality through a Pandas series or dataframe's .plot namespace. Pyplot is a collection of command-style methods that essentially make MatPlotLib's charting methods feel like MATLAB.

## Basic Plots

### Histograms

Histograms are graphical techniques which have been identified as being most helpful for troubleshooting issues. Histograms help you understand the distribution of a feature in your dataset. They accomplish this by simultaneously answering the questions where in your feature's domain your records are located at, and how many records exist there.

Let's go ahead and explore a little bit about how to use histograms and what they can actually do. You've probably seen wheat before, nothing new there, however, it turns out that there's quite a few different varieties of it. A group titled the Polish Academy of Science, particularly their Agrophysics Institute, what they do is they created a dataset that has a bunch of different types of wheat in them. And they x-rayed the wheat, the different specimens of wheat and then they featured the results. So some of the metrics that they curated, include the groove length of the wheat which is this thing over here. They also got the actual kernel length and the kernel width and some other features about the wheat. In addition to that, they also added in some engineered or calculated features such as an asymmetry constant.

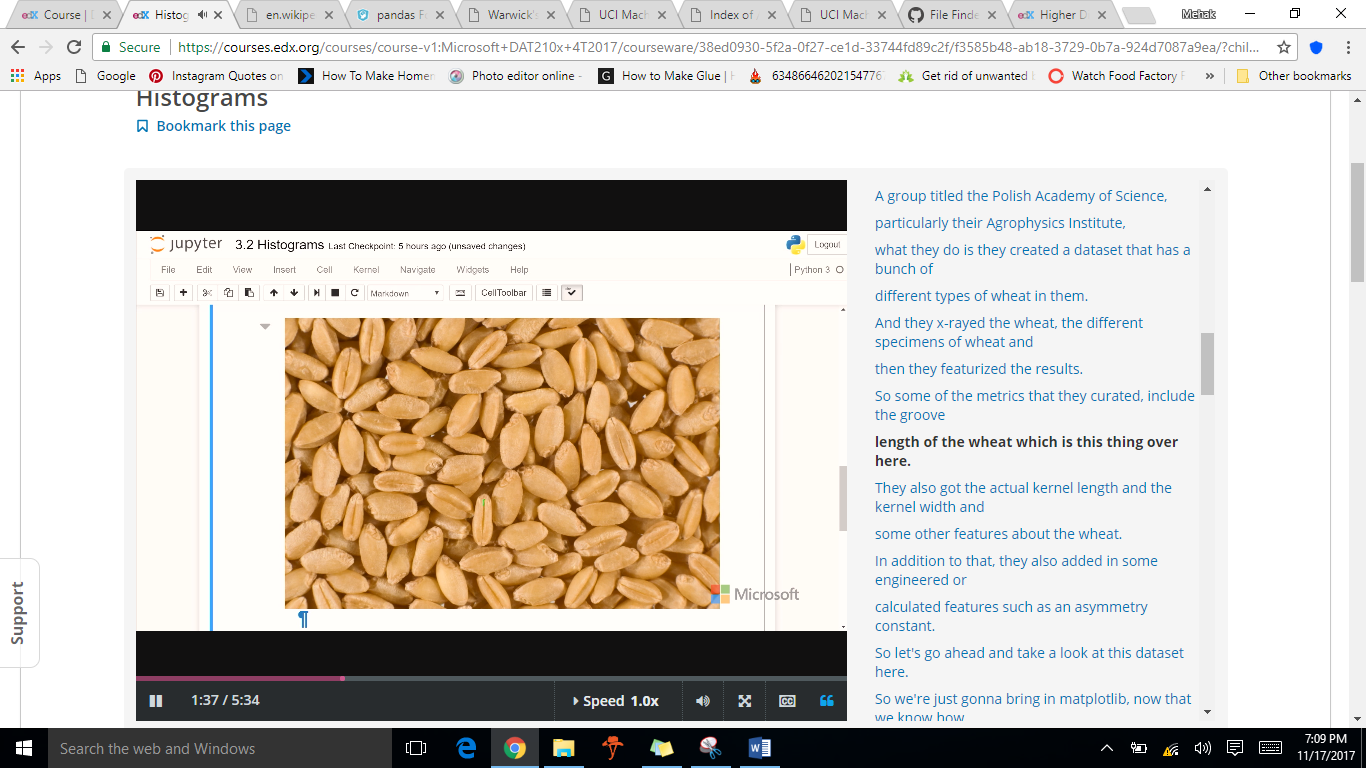


Figure 3.2: Wheat Kernel

DATA SET DESCRIPTION

The examined group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for   
the experiment. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology. The images were recorded on 13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.



|  |
| --- |
| Example 3.1: |
| import pandas as pd  import matplotlib  import matplotlib.pyplot as plt  matplotlib.style.use('ggplot') # Look Pretty, ggplot is Python implementation of the grammar of graphics.  # If the above line throws an error, use plt.style.use('ggplot') instead    df = pd.read\_csv("C:/Users/Mehak/Desktop/DataScience/wheat.csv", index\_col=0)  #Prints columns in present in your csv file  print(df.columns)  #Creates a squence named as my\_series\_series using asymetry column  my\_series = df.asymmetry  #Creates a dataframe named as my\_dataframe using provided columns  my\_dataframe = df[['wheat\_type', 'length', 'asymmetry']]    #Histogram creation of sequence  my\_series.plot.hist(alpha=0.5)  plt.show()  #Histogram creation of dataframe  my\_dataframe.plot.hist(alpha=0.5)  plt.show()  #Histogram creation based on particular condition  df[df.wheat\_type==1].asymmetry.plot.hist(alpha=0.4)#Kama  df[df.wheat\_type==2].asymmetry.plot.hist(alpha=0.4)#Rosa  df[df.wheat\_type==3].asymmetry.plot.hist(alpha=0.4)#Canadian  plt.show() |
| Output: |
| Figure 3.3: Histogram based on one feature  Chart, histogram  Description automatically generated  Figure 3.4: Histogram based on three features  Chart, histogram  Description automatically generated  Figure 3.5: Histogram based on conditions |

### 2D Scatter Plots

2D scatter plots are used to visually inspect if a correlation exist between the charted features. Both axes of a 2D scatter plot represent a distinct, numeric feature. They don't have to be continuous, but they must at least be ordinal since each record in your dataset is being plotted as a point with its location along the axes corresponding to its feature values. Without ordering, the position of the plots would have no meaning.

It is possible that either a negative or positive correlation exist between the charted features, or alternatively, none at all. The correlation type can be assessed through the overall diagonal trending of the plotted points.

Positive and negative correlations may further display a linear or non-linear relationship. If a straight line can be drawn through your scatter plot and most of points seem to stick close to it, then it can be said with a certain level of confidence that there is a linear relationship between the plotted features. Similarly, if a curve can be drawn through the points, there is likely a non-linear relationship. If neither a curve nor line adequately seems to fit the overall shape of the plotted points, chances are there is neither a correlation nor relationship between the features, or at least not enough information at present to determine.

Dataset for the next example is taken from UCI Machine Learning Repository.



|  |
| --- |
| Example 3.2: |
| import pandas as pd  import matplotlib  import matplotlib.pyplot as plt    matplotlib.style.use('ggplot') # Look Pretty  # If the above line throws an error, use plt.style.use('ggplot') instead    #Creation of dataframe from csv file  df = pd.read\_csv("C:/Users/Mehak/Desktop/DataScience/Concrete\_Data.csv", index\_col=0)  #prints columns in file  print(df.columns)  #Rename Columns  df.columns=['Slag','Ash','Water','Superplasticizer','Coarse Aggregate','Fine Aggregate','Age','Strength']  #Print new column names  print(df.columns)  #Creates scatter plots based on different features  df.plot.scatter(x='Slag', y='Strength')  plt.show()  df.plot.scatter(x='Water', y='Strength')  plt.show()  df.plot.scatter(x='Ash', y='Strength')  plt.show() |
| Output: |
| Figure 3.6: 2D Scatter plot based on slag & strength    Figure 3.6: 2D Scatter plot based on water & strength    Figure 3.6: 2D Scatter plot based on ash & strength |

### 3D Scatter Plots

There surely is a way to visualize the relationship between three variables simultaneously. That way is through 3D scatter plots. Unfortunately, the Pyplot member of Pandas data frames don't natively support the ability to generate 3D plots, so for the sake of your visualization repertoire, you're going to learn how to make them directly with MatPlotLib.

|  |
| --- |
| Example 3.3: |
| import pandas as pd  import matplotlib  import matplotlib.pyplot as plt  from mpl\_toolkits.mplot3d import Axes3D #for creation of 3d plots    matplotlib.style.use('ggplot') # Look Pretty  # If the above line throws an error, use plt.style.use('ggplot') instead    #Creation of dataframe from csv file  df = pd.read\_csv("C:/Users/Mehak/Desktop/DataScience/Concrete\_Data.csv", index\_col=0)  #prints columns in file  print(df.columns)  #Rename Columns  df.columns=['Slag','Ash','Water','Superplasticizer','Coarse Aggregate','Fine Aggregate','Age','Strength']  #Print new column names  print(df.columns)  #creates figure  fig = plt.figure()  ax = fig.add\_subplot(111, projection='3d')#"1x1 grid, first subplot"  #OR USE ax = fig.gca(projection='3d'),gca(Get the current axes, creating one if necessary)  ax.set\_xlabel('Slag')  ax.set\_ylabel('Water')  ax.set\_zlabel('Strength')    ax.scatter(df.Slag, df.Water, df.Strength, c='b', marker='.')  plt.show() |
| Output: |
| Figure 3.7: 3D Scatter plot |

## Higher Dimensionality Visualizations

Scatter plots are effective in communicating data by mapping a feature to spatial dimensions, which we understand intuitively. However, you and I are limited in that we lose the ability to easily and passively comprehend an image past three spatial dimensions. It takes a great deal of thought and even more creativity to push the envelope any further. You can introduce a time dimension using animations, but it really doesn't get much better than that.

Real world datasets often have tens of features, if not more. Sparse datasets can have tens of thousands of features. What are your visualization options if when you have a dataset with more than three dimensions?

### Parallel Coordinates

Parallel coordinate plots are similar to scatter plots in that each axis maps to the ordered, numeric domain of a feature. But instead of having axes aligned in an orthogonal manner, parallel coordinates get their name due to their axes being arranged vertically and in parallel. All that is just a fancy way of saying parallel coordinates are a bunch of parallel, labeled, numeric axes.

Each graphed observation is plotted as a polyline, a series of connected line segments. The joints of the polyline fall on each axis. Since each axis maps to the domain of a numeric feature, the resulting polyline fully describes the value of each of the observation's features.

Parallel coordinates are a useful charting technique you'll want to add the exploring section of your course map. They are a higher dimensionality visualization technique because they allow you to easily view observations with more than three dimensions simply by tacking on additional parallel coordinates. However at some point, it becomes hard to comprehend the chart anymore due to the sheer number of axes and also potentially due to the number of observations. If you data has more than 10 features, parallel coordinates might not do it for you.

Parallel coordinates are useful because polylines belonging to similar records tend to cluster together. To graph them with Pandas and MatPlotLib, you have to specify a feature to group by (it can be non-numeric). This results in each distinct value of that feature being assigned a unique color when charted. Here's an example of parallel coordinates using SciKit-Learn's Iris dataset.

|  |
| --- |
| Example 3.4: |
| from sklearn.datasets import load\_iris  from pandas.tools.plotting import parallel\_coordinates    import pandas as pd  import matplotlib.pyplot as plt  import matplotlib    # Look pretty...  matplotlib.style.use('ggplot')  # If the above line throws an error, use plt.style.use('ggplot') instead    # Load up SKLearn's Iris Dataset into a Pandas Dataframe  data = load\_iris() # <class 'sklearn.utils.Bunch'>  df = pd.DataFrame(data.data, columns=data.feature\_names) #creates dataframe    df['target\_names'] = [data.target\_names[i] for i in data.target]# creates a series named as target name in df based on the class of each observation is stored in the .target attribute of the dataset.    # Parallel Coordinates Start Here:  plt.figure()  parallel\_coordinates(df, 'target\_names')  plt.show() |
| Output: |
| Figure 3.8: Parallel coordinates |

Pandas' parallel coordinates interface is extremely easy to use, but use it with care. It only supports a single *scale* for all your axes. If you have some features that are on a small scale and others on a large scale, you'll have to deal with a compressed plot. For now, your only three options are to:

* Normalize your features before charting them
* Change the scale to a log scale

### Andrew’s Curve

An Andrews plot, also known as Andrews curve, helps you visualize higher dimensionality, multivariate data by plotting each of your dataset's observations as a curve. The feature values of the observation act as the coefficients of the curve, so observations with similar characteristics tend to group closer to each other. Due to this, Andrews’s curves have some use in outlier detection.

Andrew’s curves are a method for visualizing multidimensional data by mapping each observation onto a function. This function is defined as:

https://4.bp.blogspot.com/-0n23UhHOB9w/VEEpDZsoZUI/AAAAAAAAA2k/uQCQAoaLGlM/s400/andrewscurve.png

For implementation details of andrew’s cure in python follow the link: <https://glowingpython.blogspot.com/2014/10/andrews-curves.html>

Here's an example of parallel coordinates using SciKit-Learn's Iris dataset.

|  |
| --- |
| Example 3.5: |
| from sklearn.datasets import load\_iris  from pandas.tools.plotting import andrews\_curves    import pandas as pd  import matplotlib.pyplot as plt  import matplotlib    # Look pretty...  matplotlib.style.use('ggplot')  # If the above line throws an error, use plt.style.use('ggplot') instead    # Load up SKLearn's Iris Dataset into a Pandas Dataframe  data = load\_iris()  df = pd.DataFrame(data.data, columns=data.feature\_names)  df['target\_names'] = [data.target\_names[i] for i in data.target]    # Andrews Curves Start Here:  plt.figure()  andrews\_curves(df, 'target\_names')  plt.show() |
| Output: |
| Figure 3.9: Andrew’s curve |

### IMSHOW

One last higher dimensionality, visualization-technique you should know how to use is MatPlotLib's .imshow() method. This command generates an image based off of the normalized values stored in a *matrix*, or rectangular array of float64s. The properties of the generated image will depend on the dimensions and contents of the array passed in:

* An [X, Y] shaped array will result in a grayscale image being generated
* A [X, Y, 3] shaped array results in a full-color image: 1 channel for red, 1 for green, and 1 for blue
* A [X, Y, 4] shaped array results in a full-color image as before with an extra channel for alpha

Besides being a straightforward way to display .PNG and other images, the .imshow() method has quite a few other use cases. When you use the .corr() method on your dataset, Pandas calculates a correlation matrix for you that measures how close to being linear the relationship between any two features in your dataset are. Correlation values may range from -1 to 1, where 1 would mean the two features are perfectly positively correlated and have identical slopes for all values.  -1 would mean they are perfectly negatively correlated, and have a negative slope for one another, again being linear. Values closer to 0 mean there is little to no linear relationship between the two variables at all (e.g., pizza sales and plant growth), and so the further away from 0 the value is, the stronger the relationship between the features.

|  |
| --- |
| Example 3.6: |
| import pandas as pd  import matplotlib.pyplot as plt  import matplotlib  import matplotlib.image as mpimg  img=mpimg.imread("C:/Users/Mehak/Desktop/DataScience/Capture.png")  print(img.shape)  plt.imshow(img)  #plt.imshow(img,aspect=0.5) |
| Output: |
| Figure 3.10: IMSHOW for images |

## Exercise

For this assignment, you'll be using the [seeds](https://archive.ics.uci.edu/ml/datasets/seeds) data set, generated by recording X-Ray measurements of various wheat kernels.



### Histograms

Question 1: Write python code that

1. Loads the seeds dataset into a dataframe.
2. Creates a slice of your dataframe that only includes the **area** and **perimeter** features
3. Creates another slice that only includes the **groove** and **asymmetry** features
4. Creates a histogram for the 'area and perimeter' slice, and another histogram for the 'groove and asymmetry' slice. Set the optional display parameter: **alpha**=0.75

Once you're done, run your code and then answer the following questions about your work:

1. Looking at your first plot, the histograms of area and perimeter, which feature do you believe more closely resembles a Gaussian / normal distribution?
2. In your second plot, does the groove or asymmetry feature have more variance?

### 2D Scatter Plots

Question 2: Write python code that

1. Loads up the seeds dataset into a dataframe
2. Create a 2d scatter plot that graphs the **area** and **perimeter** features
3. Create a 2d scatter plot that graphs the **groove** and **asymmetry** features
4. Create a 2d scatter plot that graphs the **compactness** and **width** features

Once you're done, answer the following questions about your work:

1. Which of the three plots seems to totally be lacking any correlation?
2. Which of the three plots has the most correlation?

### 3D Scatter Plots

Question 3: Write python code that

1. Loads up the seeds dataset into a dataframe. You should be very good at doing this by now.
2. Graph a 3D scatter plot using the **area**, **perimeter**, and **asymmetry** features. Be sure to label your axes, and use the optional display parameter **c**='red'.
3. Graph a 3D scatter plot using the **width**, **groove**, and **length** features. Be sure to label your axes, and use the optional display parameter **c**='green'.

Once you're done, answer the following questions about your work.

1. Which of the plots seems more compact / less spread out?
2. Which of the plots were you able to visibly identify two outliers within, that stuck out from the samples?

### Parallel Coordinates

Question 4: Write python code that

1. Loads up the seeds dataset into a dataframe
2. Drop the **area**, and **perimeter** features from your dataset. Use .drop method on data frame to drop specified columns
3. Plot a parallel coordinates chart, grouped by the **wheat\_type** feature. Be sure to set the optional display parameter **alpha** to 0.4

Once you're done, answer the following questions about your work.

1. Which class of wheat do the two outliers you found previously belong to?
2. Which feature has the largest spread of values across all three types of wheat?

### Andrew’s Plot

Question 5: Write python code that

1. Loads up the seeds dataset into a dataframe
2. Plot anandrew’s curve chart, grouped by the **wheat\_type** feature. Be sure to set the optional display parameter **alpha** to 0.4

Once you're done, answer the following questions about your work.

1. Are your outlier samples still easily identifiable in the plot?

### IMSHOW

Question 6: Write python code that

1. Loads up any image of your choice, into a dataframe.
2. Print shape and type of the object holding image.
3. Plot image using **imshow.**