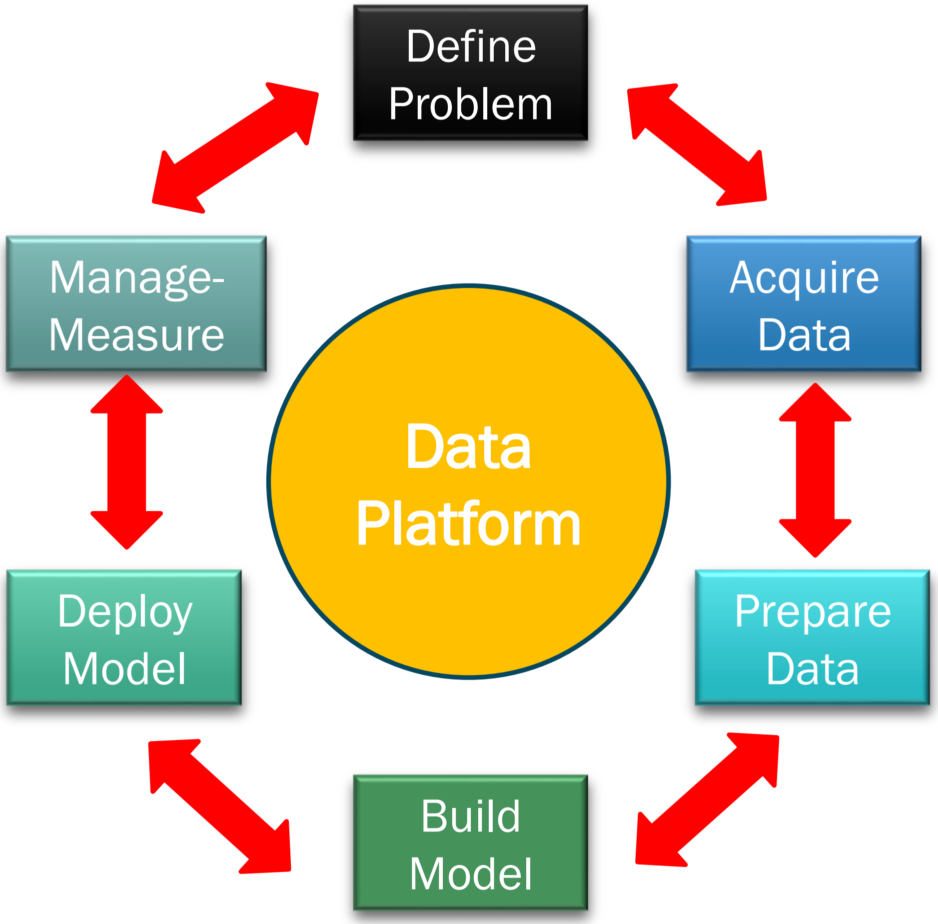
MIDTERM REVIEW

REVIEW TOPICS

- Exploratory Data Analysis process
- Reading in a .csv file into a pandas dataframe
- Data profiling
- Selecting data from a dataframe
- Scatter plots
- Log-log scatter plots
- Mean (average), median, variance, std
- Skew (left/right)
- Pearson r, r^2 correlation
- Pairplots, use of color to separate classes
- Box Plots and their interpretation

BYU

DATA SCIENCE LIFECYCLE



- Identify Data Location
- Manage Metadata
- Extract, Transform, Load
- Join/Merge Tables
- Profile Data
- Data Type and Validation Check
- Regulatory/PII Check

- Clean Data (Missing Values, Outliers, Duplicates, Var names)
- EDA (Univariate, Bivariate, Multivariate analysis, Visualization)
- Prepare Analytic Data Set (ADS) (Standardization/Normalization, Category encoding)
- Feature Engineering

Many times, the EDA is the objective of the project.

BYU

EXPLORATORY DATA ANALYSIS

Process 4.5.1: Exploratory data analysis.

These four steps should be followed in conducting exploratory data analysis:

Step	Description
Step 1: Understand the data	This is done as part of the ”Profile Data” step before analysis. Objective is to understanding the data types, numbers, ranges, overall cleanliness
Step 2: Detect and address Outliers and missing data	The is done in the Data Cleaning stage. Data needs to be cleaned before analysis; otherwise, analysis could be skewed by dirty data.
Step 3: Describe the shape of each feature of the data	Use descriptive univariate statistics and visualization to characterize data distributions for each feature.
Step 4: Identify and address correlations between features	Use multivariate analysis. Assess whether features with high (+/-) correlations can be dropped.

BYU

SUMMARY: DATA ACQUISITION→ PREPARATION→ EXPLORATION

Data Analysis Processes	Tools
Acquire Data	pd.read_csv(), pd.read_excel(), df.to_csv(index = False), df.to_excel(index = False),
Profile Data	df.head(), df.tail(), df.info(), df.describe(include='all'), df.dtypes, df.shape
Manipulate Data	df.groupby(), df.pivot_table(), df.insert()
Clean Data Missing Data Imputing Data Duplicates Outliers Data Types	df.isnull().sum(), df.isna(), df.fillna(value=), df.drop(), df.drop_duplicates(), df.dropna(), df.replace(), df.astype(),
Exploratory Data Analysis Univariate Multivariate Visualization	df.plot(), plt.show(), df.plot.scatter(), df.plot.box(), plt.hist() plt.subplots(figsize=), sns.histplot(), sns.kdeplot(), sns.boxplot(), sns.violinplot() df.mean(), sns.scatterplot(), sns.swarmplot(), sns.stripplot(), plotly.express
Transform Data	preprocessing.scale(), preprocessing.MinMaxScaler().fit_transform() [Part of Modeling because we need to wait until after train/validation/test split]

PANDAS DATAFRAME

A **Pandas DataFrame** is a two-dimensional, labeled data structure in Python, similar to a table or spreadsheet, that stores data in rows and columns. Each column in a DataFrame can have a different data type (e.g., integers, floats, strings, etc.).

Key features of a DataFrame:

- **Rows and Columns:** Like a table, with rows representing individual records and columns representing variables or features.
- **Labels (Index):** Rows and columns can have labels (names), making it easy to access, slice, or manipulate data.
- **Data Handling:** It can handle missing data and supports arithmetic operations, data filtering, aggregation, and transformation.
- **Data Input/Output:** Can read and write data from various file formats (e.g., CSV, Excel, SQL, etc.).

Example:

python
```python
import pandas as pd

# Create a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'San Francisco', 'Los Angeles']
}

df = pd.DataFrame(data)

print(df)
```

This code would output:

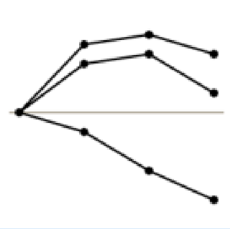
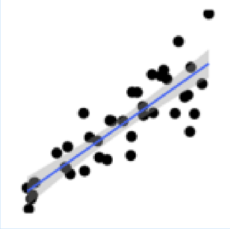
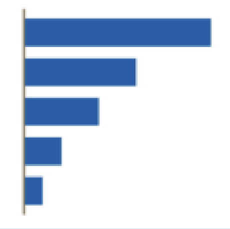
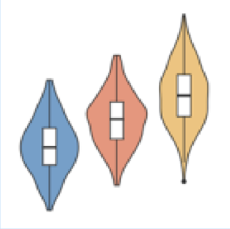
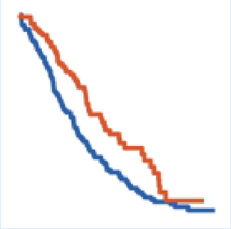
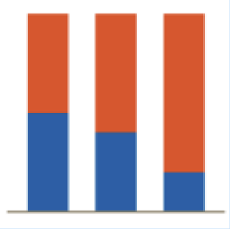
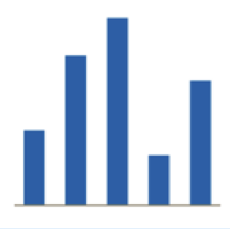
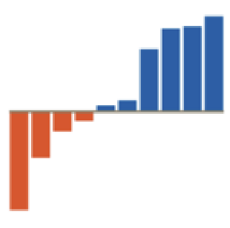
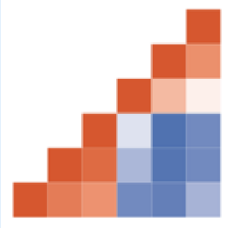
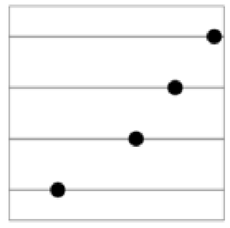
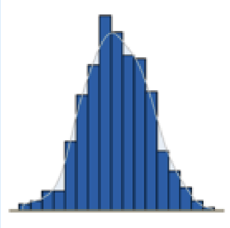
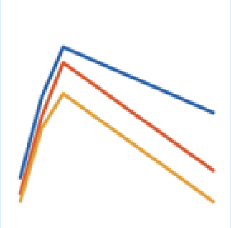
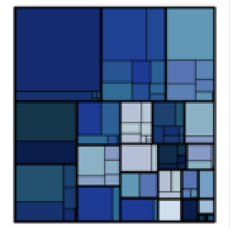
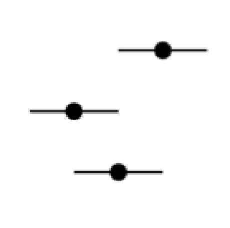
```
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles
```

BYU

SELECTING THE RIGHT BASE GRAPH

Consider if a standard graph can be used by identifying suitable designs based on the:
(i) **purpose** (i.e. message to be conveyed or question to answer) and (ii) **data** (i.e. variables to display).

Example plots categorized by purpose

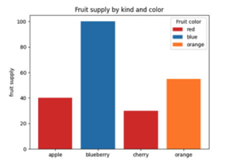
Deviation	Correlation	Ranking	Distribution	Evolution	Part-to-whole	Magnitude
Chg. from baseline	Scatter plot	Horizontal bar chart	Boxplot	Kaplan Meier	Stacked bar chart	Vertical bar chart
						
Waterfall	Heat map	Dotplot	Histogram	Line plot	Tree map	Forest plot
						

MATPLOTLIB

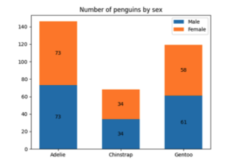
Matplotlib is a popular Python library used for creating static, animated, and interactive visualizations. Matplotlib can produce a variety of plots, such as:

- Line plots
- Bar charts
- Scatter plots
- Histograms
- Pie charts
- 3D plots

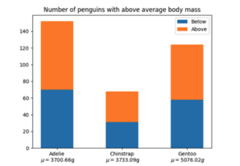
It works closely with other libraries like NumPy for numerical computations and Pandas for handling data structures.



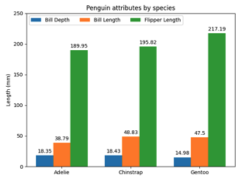
Bar color demo



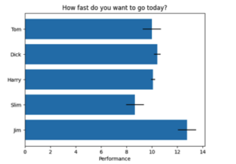
Bar Label Demo



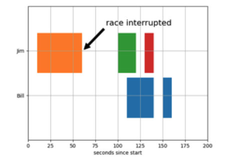
Stacked bar chart



Grouped bar chart with labels

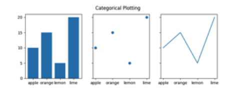


Horizontal bar chart

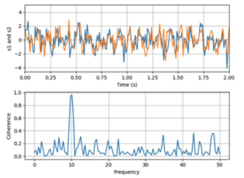


Broken Barh

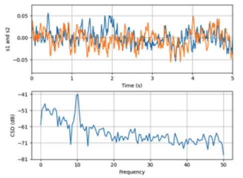
CapStyle



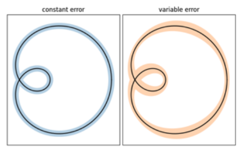
Plotting categorical variables



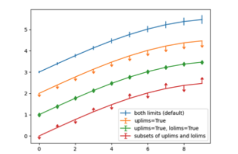
Plotting the coherence of two signals



Cross spectral density (CSD)



Curve with error band



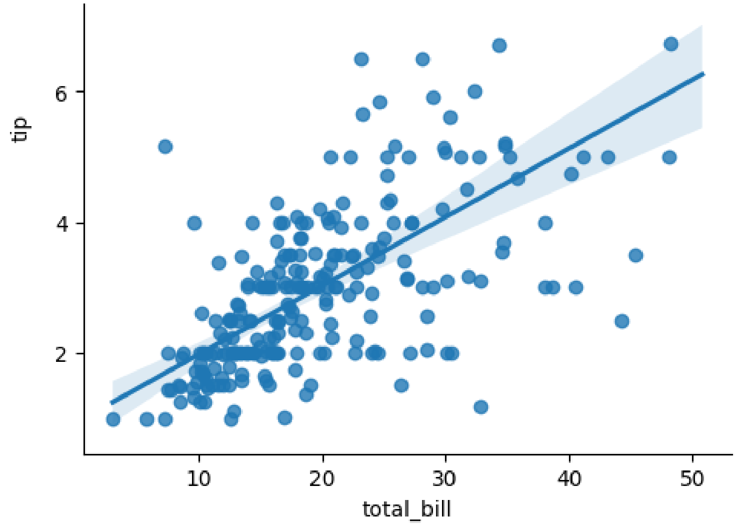
Errorbar limit selection

BYU

SEABORN

- Seaborn is a Python data visualization library built on top of Matplotlib, designed to make it easier to create informative and aesthetically pleasing statistical graphics.
- Seaborn integrates closely with Pandas data structures, which makes it especially powerful for working with data frames and structured data.
- **Plot Types**: Seaborn supports many types of plots, such as:
  - Line plots
  - Bar plots
  - Scatter plots
  - Heatmaps
  - **Pair plots (My favorite!)**
  - Box plots
  - Violin plots

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load an example dataset from Seaborn
tips = sns.load_dataset("tips")

# Create a scatter plot with a linear fit
sns.lmplot(x="total_bill", y="tip", data=tips)

# Display the plot
plt.show()
```



BYU

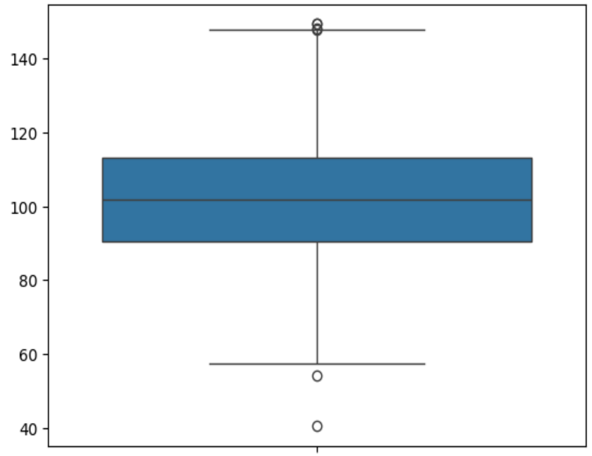
BOX AND WHISKER PLOT

A **boxplot** is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It helps to show the spread and skewness of the data, highlighting potential outliers.

- **Key Elements :**
- 1. **Box**: Spans from Q1 to Q3 (the interquartile range, IQR).
- 2. **Median**: A line inside the box showing the middle of the dataset.
- 3. **Whiskers**: Lines extending from Q1 to the minimum and Q3 to the maximum values within 1.5 times the IQR.
- 4. **Outliers**: Data points that fall outside the whiskers.

By combining these elements, a box plot quickly provides insights into the data's **distribution, variability, and any unusual observations** (like outliers).

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Generate random data
np.random.seed(10)
data = np.random.normal(100, 20, 200)

# Create a boxplot using seaborn
sns.boxplot(data=data)

# Display the plot
plt.show()
```



COVARIANCE VS CORRELATION

Covariance
Covariance measures how the deviation of one variable from its mean is related to the deviation of another variable from its mean
(-∞, +∞)

The mathematical formula for **population covariance** between two variables X and Y is:

$$\mathrm{Cov}(X,Y) = \frac{\sum_{i=1}^{N}(X_i - \mu_X)(Y_i - \mu_Y)}{N}$$

Where:

- $\mathrm{Cov}(X,Y)$ = population covariance between variables X and Y,
- X_i and Y_i = individual data points for X and Y,
- μ_X = population mean of X,
- μ_Y = population mean of Y,
- N = total number of data points in the population.

Correlation
Correlation measures how strongly the two variables are related to each other
[-1, 1]

The mathematical formula for **correlation** (specifically, the **Pearson correlation coefficient**) between two variables X and Y is:

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

Where:

- $\rho_{X,Y}$ = population correlation coefficient between variables X and Y,
- $\mathrm{Cov}(X,Y)$ = covariance between X and Y,
- σ_X = standard deviation of X,
- σ_Y = standard deviation of Y.

BYU

CORRELATION COEFFICIENT

- The **correlation coefficient** is a statistical measure that describes the strength and direction of a relationship between two variables.
- It quantifies how much one variable changes in response to changes in another variable.
- The most common type of correlation coefficient is the Pearson correlation coefficient.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

```python
# Sample data with three variables
data = {
    'x': [3, -0.5, 2, 7],
    'y': [2.5, 0.0, 2, 8],
    'z': [1, 2, 3, 4]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Calculate the correlation matrix for all columns
correlation_matrix = df.corr()

print(correlation_matrix)
```

	x	y	z
x	1.000000	0.984870	0.600143
y	0.984870	1.000000	0.697369
z	0.600143	0.697369	1.000000

r^2 is just the square of the correlation coefficient

BYU

MEAN, VARIANCE, STANDARD DEVIATION

The formula for the **population mean** is:

$$\mu = \frac{\sum_{i=1}^{N} X_i}{N}$$

Where:

- μ = population mean,
- X_i = each individual data point in the population,
- N = the total number of data points in the population.

The formula for **population variance** is:

$$\sigma^2 = \frac{\sum_{i=1}^{N} (X_i - \mu)^2}{N}$$

Where:

- σ^2 = population variance,
- X_i = each individual data point in the population,
- μ = population mean,
- N = total number of data points in the population.

The formula for **population standard deviation** is:

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \mu)^2}{N}}$$

Where:

- σ = population standard deviation,
- σ^2 = population variance,
- X_i = each individual data point in the population,
- μ = population mean,
- N = total number of data points in the population.

BYU

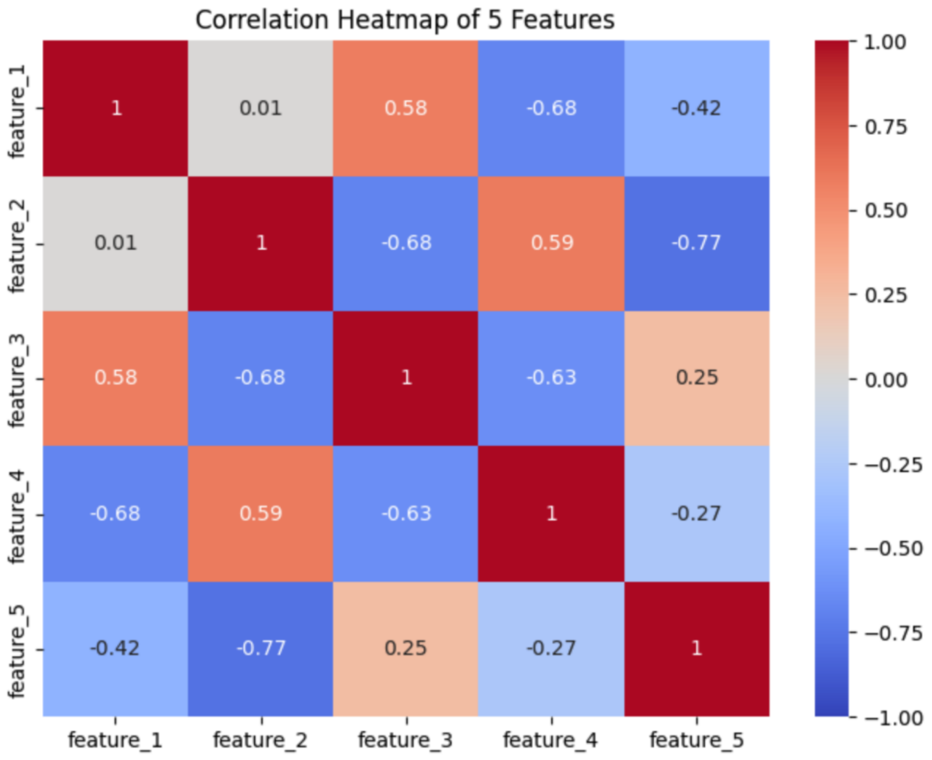
CORRELATION HEATMAP

- A heatmap is an excellent way to visualize the correlation between multiple features.
- For example, you can use a heatmap to display the correlation matrix between five features using seaborn, a Python visualization library built on top of matplotlib.

Steps:

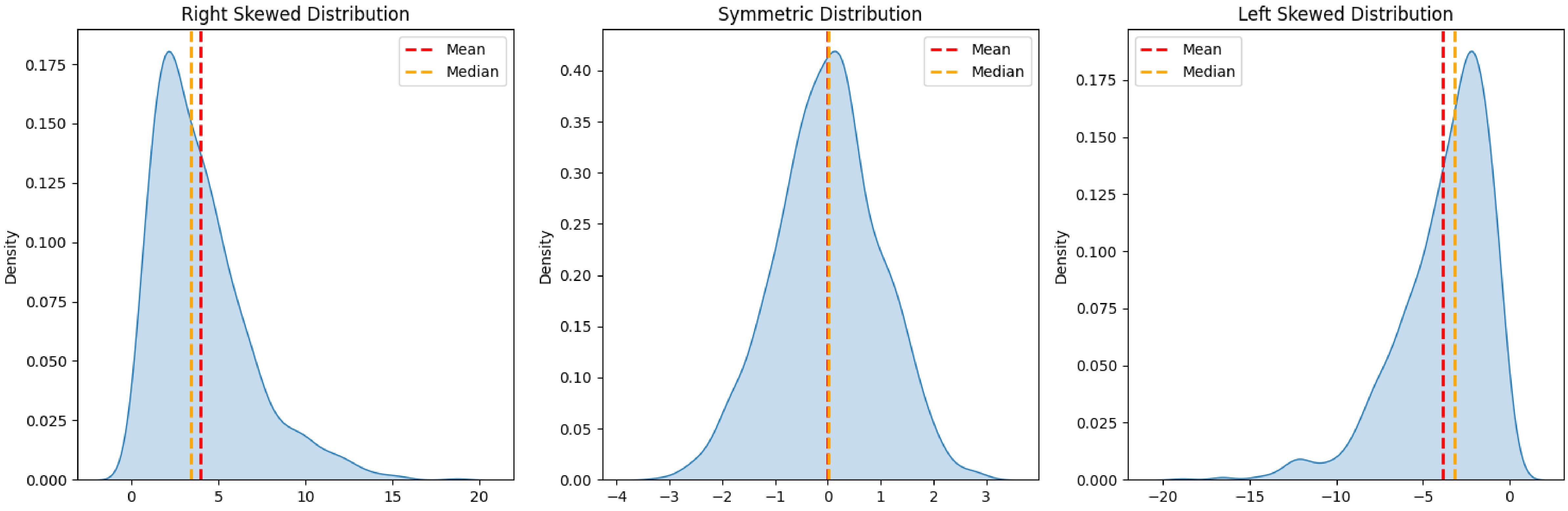
1. Compute the correlation matrix between the features using pandas.
2. Plot the heatmap using seaborn's heatmap () function.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot a

# Create a sample DataFram
data = {
    'feature_1': [0.1, 2,
    'feature_2': [0.5, 4,
    'feature_3': [3.2, 0.2
    'feature_4': [3.5, 4.6
    'feature_5': [10, 0.9,
}

df = pd.DataFrame(data)

# Calculate the correlatio
correlation_matrix = df.co

# Plot a heatmap
plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

# Add title
plt.title('Correlation Heatmap of 5 Features')

# Show the plot
plt.show()
```



BYU

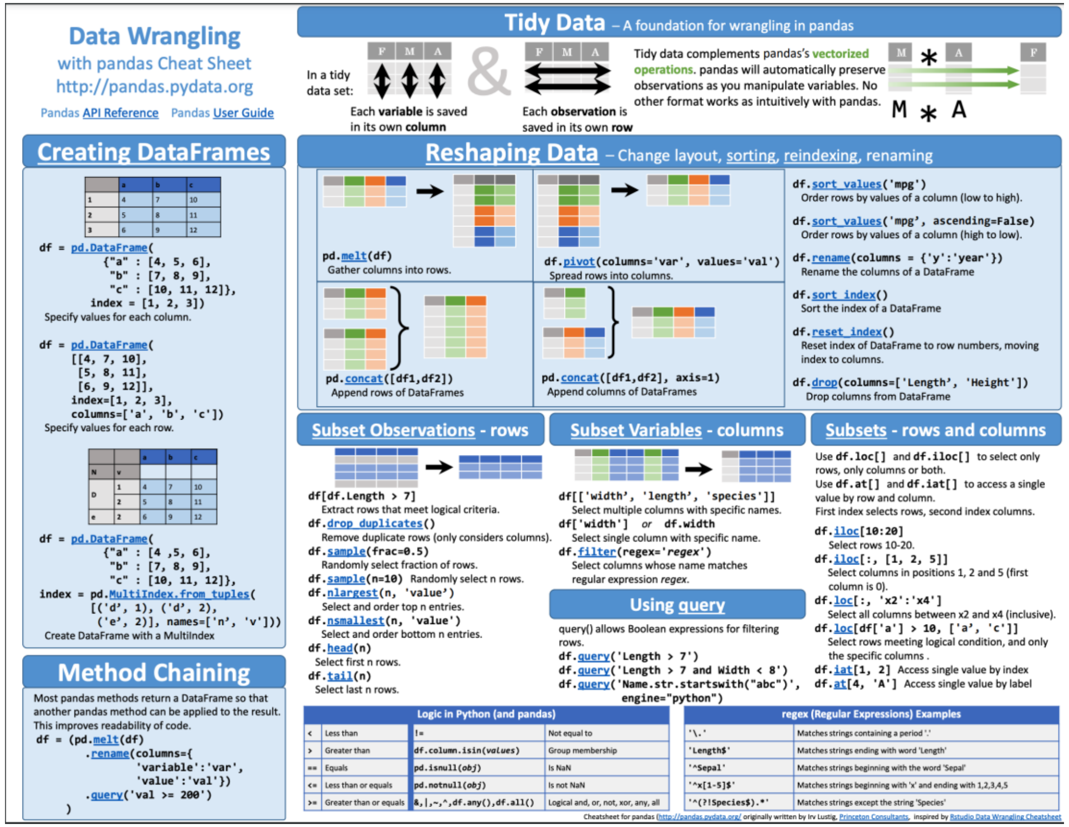
HOW DISTRIBUTION SKEW AFFECTS MEAN AND MEDIAN



https://colab.research.google.com/drive/1HtaxbmvEOvKSUp6ok8m0dEeYmxA3dXKs#scrollTo=u_VK5Iuf18fF

BYU

MOST COMMONLY USED PANDAS OPERATIONS



Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Pandas API Reference   Pandas User Guide

Tidy Data – A foundation for wrangling in pandas

In a tidy data set: Each variable is saved in its own column. Each observation is saved in its own row.

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

Creating DataFrames

```
df = pd.DataFrame(
        {"a" : [4, 5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
     [[4, 7, 10],
      [5, 8, 11],
      [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```
Specify values for each row.

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
     )
```

Reshaping Data – Change layout, sorting, reindexing, renaming

`pd.melt(df)` Gather columns into rows.

`df.pivot(columns='var', values='val')` Spread rows into columns.

`pd.concat([df1,df2])` Append rows of DataFrames

`pd.concat([df1,df2], axis=1)` Append columns of DataFrames

`df.sort_values('mpg')` Order rows by values of a column (low to high).

`df.sort_values('mpg', ascending=False)` Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})` Rename the columns of a DataFrame

`df.sort_index()` Sort the index of a DataFrame

`df.reset_index()` Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length', 'Height'])` Drop columns from DataFrame

Subset Observations - rows

`df[df.Length > 7]` Extract rows that meet logical criteria.

`df.drop_duplicates()` Remove duplicate rows (only considers columns).

`df.sample(frac=0.5)` Randomly select fraction of rows.

`df.sample(n=10)` Randomly select n rows.

`df.nlargest(n, 'value')` Select and order top n entries.

`df.nsmallest(n, 'value')` Select and order bottom n entries.

`df.head(n)` Select first n rows.

`df.tail(n)` Select last n rows.

Subset Variables - columns

`df[['width', 'length', 'species']]` Select multiple columns with specific names.

`df['width']` or `df.width` Select single column with specific name.

`df.filter(regex='regex')` Select columns whose name matches regular expression regex.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use df.loc[] and df.iloc[] to select only rows, only columns or both.

Use df.at[] and df.iat[] to access a single value by row and column.

First index selects rows, second index columns.

`df.iloc[10:20]` Select rows 10-20.

`df.iloc[:, [1, 2, 5]]` Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[:, 'x2':'x4']` Select all columns between x2 and x4 (inclusive).

`df.loc[df['a'] > 10, ['a', 'c']]` Select rows meeting logical condition, and only the specific columns.

`df.iat[1, 2]` Access single value by index

`df.at[4, 'A']` Access single value by label

Logic in Python (and pandas)

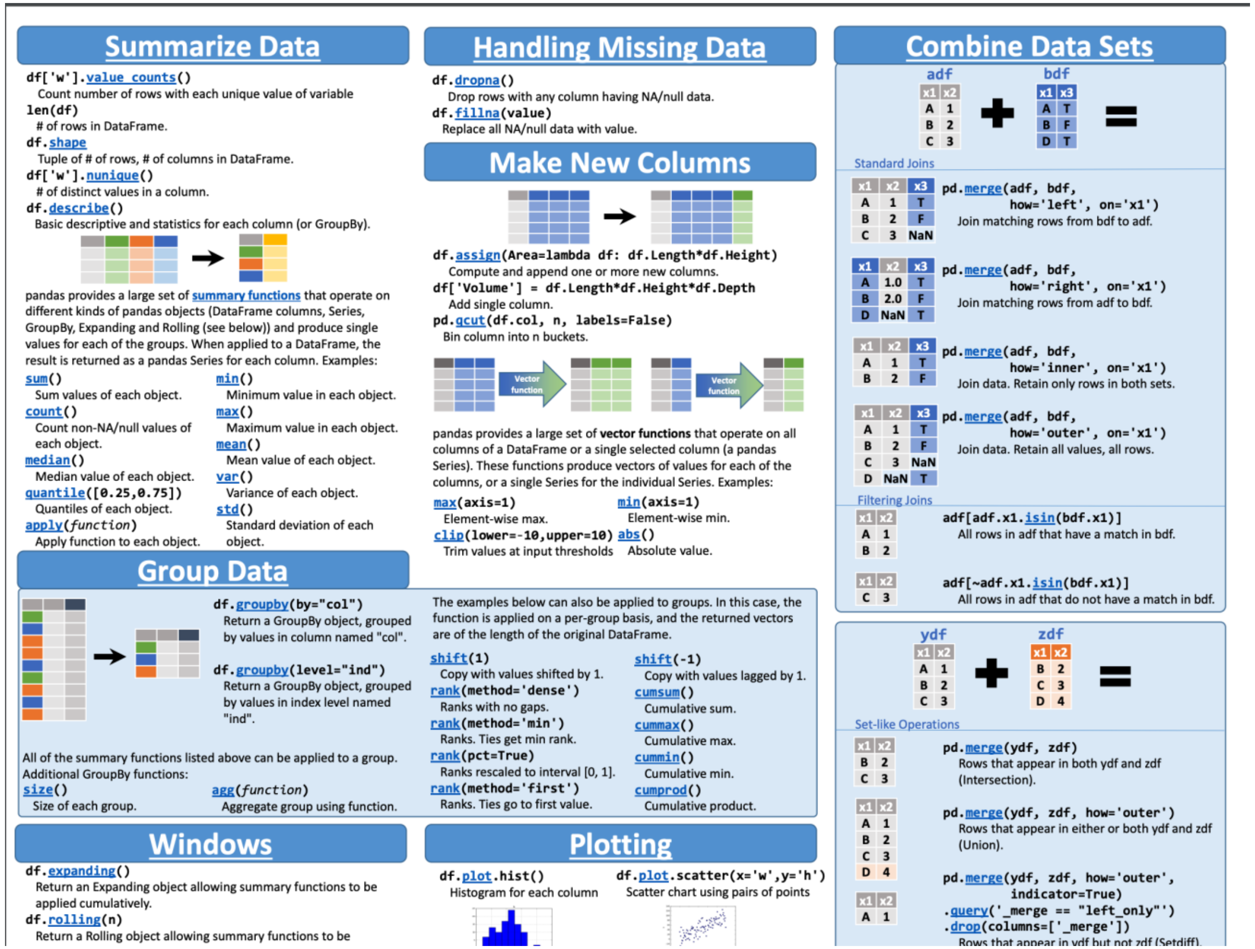
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equals	pd.isnull(obj)	Is NaN
<=	Less than or equals	pd.notnull(obj)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

Cheatsheet for pandas (http://pandas.pydata.org/) originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf



Summarize Data

`df['w'].value_counts()` Count number of rows with each unique value of variable

`len(df)` # of rows in DataFrame.

`df.shape` Tuple of # of rows, # of columns in DataFrame.

`df['w'].nunique()` # of distinct values in a column.

`df.describe()` Basic descriptive and statistics for each column (or GroupBy).

pandas provides a large set of summary functions that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()` Sum values of each object.

`count()` Count non-NA/null values of each object.

`median()` Median value of each object.

`quantile([0.25,0.75])` Quantiles of each object.

`apply(function)` Apply function to each object.

`min()` Minimum value in each object.

`max()` Maximum value in each object.

`mean()` Mean value of each object.

`var()` Variance of each object.

`std()` Standard deviation of each object.

Group Data

`df.groupby(by="col")` Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")` Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()` Size of each group.

`agg(function)` Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)` Copy with values shifted by 1.

`rank(method='dense')` Ranks with no gaps.

`rank(method='min')` Ranks. Ties get min rank.

`rank(pct=True)` Ranks rescaled to interval [0, 1].

`rank(method='first')` Ranks. Ties go to first value.

`shift(-1)` Copy with values lagged by 1.

`cumsum()` Cumulative sum.

`cummax()` Cumulative max.

`cummin()` Cumulative min.

`cumprod()` Cumulative product.

Windows

`df.expanding()` Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)` Return a Rolling object allowing summary functions to be

Handling Missing Data

`df.dropna()` Drop rows with any column having NA/null data.

`df.fillna(value)` Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)` Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth` Add single column.

`pd.qcut(df.col, n, labels=False)` Bin column into n buckets.

pandas provides a large set of vector functions that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)` Element-wise max.

`clip(lower=-10,upper=10)` Trim values at input thresholds

`min(axis=1)` Element-wise min.

`abs()` Absolute value.

Plotting

`df.plot.hist()` Histogram for each column

`df.plot.scatter(x='w',y='h')` Scatter chart using pairs of points

Combine Data Sets

Standard Joins

`pd.merge(adf, bdf, how='left', on='x1')` Join matching rows from bdf to adf.

`pd.merge(adf, bdf, how='right', on='x1')` Join matching rows from adf to bdf.

`pd.merge(adf, bdf, how='inner', on='x1')` Join data. Retain only rows in both sets.

`pd.merge(adf, bdf, how='outer', on='x1')` Join data. Retain all values, all rows.

Filtering Joins

`adf[adf.x1.isin(bdf.x1)]` All rows in adf that have a match in bdf.

`adf[~adf.x1.isin(bdf.x1)]` All rows in adf that do not have a match in bdf.

Set-like Operations

`pd.merge(ydf, zdf)` Rows that appear in both ydf and zdf (Intersection).

`pd.merge(ydf, zdf, how='outer')` Rows that appear in either or both ydf and zdf (Union).

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

BYU