Pandas: Provides data structures (dataframes and series)
Import pandas as pd
Read a csv file using df = pd.read_csv('filename.csv')
Different file types can be read such as Json, excel,etc.
df.info() -> summary of the data
df.describe() -> statistical summary of the data
df.tail() -> last 5 rows of the data
df.head() -> first 5 rows of the data
df.sample() -> random row of the data
Dictionary can be used to create a dataframe and list of lists
df=pd.DataFrame({'a':1})
to access a particular column use
df['column_name'] or df.column_name, the first one is better because it the later can cause issue when the column name is same as some function.
Set a column as index using
df.set_index('column_name')
df.iloc[0, :] -> gets the first row.
df.loc['index_name', :] -> accesses rows by index label.
df.dtypes -> gives data type of the column
df.drop() -> used to drop a row or column
df.isna(), df.isnull() -> used to check Nan values
df.dropna() -> drop the rows or column with Na values
df.fillna() -> fill the missing values with constant, mean,median, etc
df.where() -> replace values in a DataFrame where a condition is False
df.value_counts() -> count the number of unique values
df.apply() -> can be used as a for loop for entire column
pd.cut() -> used to divide data into parts using bins
df.stack() -> converts this DataFrame into a Series with a MultiIndex, unstack() does the opposite
pd.concat() -> join two dataset in row or columns
df.join() -> uses index to join two dataframes
df.merge() -> used for joining on a condition
df.pivot() -> create a new derived dataframe out of a dataframe, it require index, column and values
df.pivot_table() -> similar to pivot but takes extra aggregation to resolve conflicts

Data structure in python:
List = [1,2,3], it is mutable
Tuple = (1,2,3), it is immutable
Dict = {'a':1,'b':2}, it is mutable
Sets = {1,2,3}, it allows insert and remove but do not allow updates.
These data structures can be mixed matched to form new data structures.

Numpy: provides array object for numerical operations.
Import numpy as np
np.array() -> creating a numpy array
np.ones() or np.zeros() -> creating a array with only 1s and 0s
array[row_start:row_stop,column_start_column_stop] -> indexing
statistical functions such as
np.mean(), np.median(), np.std() gives mean, median and standard deviation
np.random.randn() -> randomly selected numbers from standard normal distribution
np.random.seed() -> fix the set of random values

Matplotlib.pyplot: data visualization tool
import matplotlib.pyplot as plt
Plot types → scatter(), hist(), box(), bar() & pie() → pie is NOT a good plot
We can annotate, change labels, change widths colors, change title, etc
subplots() → to plot multiple plots

Seaborn: built on top of matplotlib, better base stats plot & more pandas friendly
import seaborn as sns
Plot types → stripplot(), swarmplot(), violinplot(), boxplot(), kdeplot, histplot(), distplot(), jointplot(), pairplot(), relplot() good for EDA
Pairplot() -> plots scatterplot and histogram for every feature

Distributions: could be uniform, bimodal, poisson, pareto, etc.
Plots help with visual tests on data
Distribution tells us lot of information about the data
Run Sequence (index versus values)
Lag Plot (y_i versus y_{i-1})
The above both help understand any time series relation in the data
QQ Plot -> straight line is normal
histograms
Lambda function -> anonymous functions that can have any number of parameters

Correlation: binds correlation between variables (basis of heatmap)
df.corr(), heatmap is colorful representation of correlation.
OLS Regression: smf.ols('y~x', df).fit()
Can be used to detect outliers

ANOVA testing: if statistically significant difference between 3 or more groups
sm.stats.anova_lm(, typ=2) for typ there are 3 types 1,2 and 3, use them after checking which one is used for what type of problem.
Tukey's HSD: post ANOVA can be used to identify which specific pairs of groups are different from each other.
pairwise_tukeyhsd( )
Hypothesis testing: t-test tests if significant difference between 2 groups
R^2 coefficient of determination, P<threshold then reject NULL hypothesis

Types of errors:
1. Syntax errors: These are errors where the code is not valid Python
2. Runtime errors: These errors are usually logical errors that cause the program to crash
3. Semantic errors: These errors are the hardest to find, because the code is valid, but it doesn't do what you intended
4. Logical errors: These are errors where the code is valid Python, but the logic is wrong
5. Exceptions: These are errors that occur when something goes wrong
6.Warnings: These are messages that Python gives you when something is not necessarily wrong, but might be a problem

Categorical data:
Contingency table are similar to the pivot_table.
mosaic plot: used to plot the contingency table
Chi-square (X^2) tests if statistically significant association or independence between 2 categorical variables
chi2, p, dof, expected = chi2_contingency(ct)

Text Processing:
s.str.lower() -> lowercase the string
s.str.upper() -> uppercase the whole string
s.str.replace() -> replace something with something else
s.str.split() -> split the expression based on the parameter
s.str.extract() -> extract something from the string if it is present or else NaN.
s.str.contains(pattern) -> Boolean value output, if the pattern is in the string or not
s.str.match(pattern) -> Boolean value output, looks for exact pattern

Regex: Regular Expression Powerful tool for pattern matching within strings
spaCy: Fast Extensible NLP package
Import spacy

nlp = spacy.load('en_core_web_sm')
Basic steps: Remove punctuations & special characters & stop words
Counter() -> for frequency of the words
wordcloud2 = WordCloud().generate() -> create a word cloud
contractions is used to convert words such as "I'm" to "I am".
Tokenize -> convert the sentence into smallest part that can be used for the models.
POS tagging -> gives the parts of speech for the words in the sentence.

Word Embedding:
Word2Vec → finds similarity btw 2 words. Gives a vector to every unique word. useful to build and train word embeddings.
Sentiment analysis use NLTK or spaCy
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sent_analyzer = SentimentIntensityAnalyzer()

scikit-learn:open-source ML library
Study the Data and EDA Understanding the data before applying any model.
Handle Missing Values -> Depending on the context and the extent of missing data drop or impute them.
Scale Numerical Data-> Scaling numerical features.
Encode Categorical Data: one-hot encoding or label encoding to make it suitable for machine learning algorithms.
Feature Selection -> Choosing important features.
Choose a Model -> The choice of model based on the problem.
Cross-Validation -> This is used to assess the performance of the model.
Performance Metrics -> Metrics such as mean absolute error (MAE), mean squared error (MSE), R-squared (R^2), precision, and recall are used to evaluate the model's predictions.
Model Tuning -> Hyperparameter tuning is conducted to find the most effective model settings.
Test the Model -> The final step is to evaluate the model's performance on unseen test data to estimate how well it will perform on new data.
Pipeline Creation -> A pipeline streamlines the process by encapsulating all preprocessing steps and the model.

Dimension Reduction: reduce complexity while preserving essential info.
manifold.MDS() -> MDS
PCA is used to linear preserves global structure.
PCA.components_ -> % of information (explained variance captured info by model)
t-SNE -> Nonlinear, emphasizes local relationships vs global components, perplexity parameter can be tuned for optimum clustering.

Clustering:
Hierarchical Clustering, it does not assume any number of clustering.
Two types of uproach bottom up and top bottom.
from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering()
Dendrograms are used to visualize this clustering.
K-means clustering creates K clusters using nearest centroid.
KMeans(n_clusters =)
It doesn't work great with data with many outliers.
Elbow silhouette score how close each point is to its centroid, the higher the score the better. It ranges between -1 to 1.
silhouette plot looking like a knife

pipelines: pipeline can be inside another pipeline.
Impute -> scale/ OneHotEncoder -> PCA(depending on the problem) -> Clustering/Classifier

Steps for classification:
1)EDA
2) NA impute: Numerical- mean/median/constant, Categorical- OneHot Encode
3)Train-test-split
4) Choose classifier: Naive Bayes, SVM, Decision Trees, Random Forests, etc. (Ruff tree)
5) Evaluate: Accuracy score, confusion matrix, classification report. Can check feature importance.
6) Tune hyperparameters: grid search, cross validation search
7) Evaluate the model, understand the output.
Ensemble can use multiple models and use voting classifier in the end, this can increase the model capabilities. It does take a little more time to train as it has multiple models.
Stacking the models is also an ensemble method where there is a final model which works on the output of the base models.

Logistic Regression: Used for binary classification but can be extended to multiclass problems
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
SVM: Designed for binary classification but also capable of handling multiclass problems using strategies such as one-vs-one or one-vs-rest.
from sklearn import svm
model = svm.SVC()
Decision trees: Easy to implement but have tendency to over fit.
This problem is solved by random forest, which has ensemble of many trees in a voting classifier.
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
Naïve Bayesian Classifier: It is the most basic probability model based on Baye's rule, it is easy to implement but has an assumption of class conditional independence.
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()

Evaluation of classifier:
accuracy = (True Positives + True Negatives) / All Samples
precision = True Positives / (True Positives + False Positives)
true positive rate (recall, sensitivity) = True Positives / (True Positives + False Negatives)
F1 = 2 * (Precision * Recall) / (Precision + Recall)
false positive rate (fall-out) = False Positives / (False Positives + True Negatives)
specificity = True Negatives / (True Negatives + False Positives)
false positive rate = 1 – specificity
There are many other evaluation matrix.
Roc curve is a plot of FPR on the x-axis and TPR on the y axis, the area under the curse if closer to 1, the better the model.

Dask: Parallel & distributed tasks, works on large amount of data
from dask.distributed import Client
client = Client(n_workers=4)
import dask.dataframe as dd
specify the tasks during data split
use .compute() -> lazy evaluation
use visualize() to see task graph
map-reduce -> HDFS (similar to .apply())
client.shutdown()
Use Dask if the data is very large, if it can be done using pandas do it with Pandas.