|  |
| --- |
| **Python applications**: Web dev, machine learning, data science, game dev, cybersecurity, technology automation, cloud orchestration | Interpreter language, bytecode is run by python VM, integrations with C or Java to run parts for faster performance | **Libraries**: Data analysis and manipulation (Pandas), Mathematical functions (NumPy), Data visualizations (Matplotlib, SeaBorn), Machine Learning (Tensorflow, PyTorch), Deep learning (Keras), Scientific Computing (SciPy), Scrapy (web crawling), Interact with SQL database (SQLModel) | **Google colab**: **GPU**: originally for graphics, popular for parallel processing tasks, 1000s of small cores for vector and matrix operations, well suited for deep learning and other compute-intensive workloads; **TPU**: App Specific Integrated Circuits (ASICs) for ML tasks, By Google, Tensor operations, core for neural network computations, FREE but not guaranteed for running long process unless premium paid | **Python virtual env**: Isolated dependency library, easy to switch between projects, reqs.txt enabled repeatability, avoids global installations, security thru isolation, makes testing new lib safer | $pip install virtualenv | $python -m venv deng | (activate) $source deng/bin/activate or C:\>deng\Scripts\activate | **Python code**: type(var) will tell data type, typecasting with str(), float() | **Code** **Naming conventions**: snake case [first\_name] (variable, functions), upper case and underscores [MAX\_LIMIT] (constants), camel case [circularObjects] (Classes), leading with underscores [\_protected\_var] (Protected instance variables), leading with two underscores [\_\_private\_var] (private instance variable) | **Python datatypes**: *Numeric* (Integer [x = 123 or x = 0b0111101 or x = 0x7B], Float [fixed precision, 64-bit values, same as C doubles], Complex No. [z = complex(x,y) z.real, z.imag, z = x+yj]), *Dictionary*, *Boolean*, *Set*, *Sequence Type* (Strings, List, Tuple) | **input**(“What’s your name”) <- will take user i/p | **Error catching** scenarios: try: except: | **Loops in python**: **for** (iterate over sequences like lists, strings, ranges), **while** (Run until a condition becomes false), **Nested loops** (process data in multi-dimensions like matrices) | for i in range(5): | for item in list: | while x < 5: | **range(1,4)** means 1,2,3 | print('\110\145\154\154\157') -> Hello (**octal values**) | **Escape chars**: \', \\, \n, \r, \t, \b, \f, \ooo, \xhh | **String built-in methods**: ***capitalize***(first character to upper case), ***count***(Nbr of times a specified value occurs), ***find***(search specified value and return position), ***index***(search specified value and return 1st position or raise ValueError), ***rindex***(return last position of specific value), ***isalnum***(returns true if all are alphanumeric), ***isupper***(return true if all are upper case), ***join***(elements of iterable to end of string), ***lower***(convert to lower case), ***split***(at specified separator, return list), *strip*(trim leading & trailing white spaces), ***swapcase***(), ***upper***(convert to upper), ***replace***(specified value with new value) | **Data collections**: **List** (ordered, mutable, duplicates, [1, 'a', 6.22], can be empty, .append, .insert(2,'b'), .pop(1) //removes 2nd element in list, returns removed value, .remove(value) , .sort(reverse = True), .copy(old\_list), *joining lists*: [list1 + list2, list2.append(list1) <- need loop, list1.extend(list2)]), **Tuple** (ordered, immutable, duplicates, (), can be empty, cannot sort, .count(element) <- # of times in tuple, .index(element) <- returns 1st location of element), **Dictionary** (key/value, can be nested, ordered, mutable, no duplicate keys, {}, keys can only be string, numbers, tuples, *.items()* returns key, value, *.get(key)* or *[key]*, *.update({‘k’:v})*, *del my\_dict[k]*, *.pop(k)* <- removes key & returns value, *.popitem()* <- removes last item added to dict & returns tuple with k & v, *.keys()* <- returns a list, *for x in my\_dict:* <- will return only keys, *.values()* <- returns a list), **Set** (unordered, mutable, no duplicates, unindexed, {}, no keys, can only be string, numbers, tuples, *.copy()*, *.add(element)* adds to set, *.update(list)* add from other types to set, *.remove(item)* generates KeyError if not present, *.discard(item)* has no error, *.pop()* removes random element, *.clear()* remove all elements, *del my\_set* fully delete variable, *set2.difference(set1)* or **–** [minus sign] <- removes values in set2 that are also in set1 & creates a new set, *set1.difference\_update(set2)* or **-=** [minus equal] <- updates original set1 and returns null, *set1.intersection(set2)* or **&** returns common values as a new set, *set1.union(set2)* or **|** returns all values as a new set, *set1.intersection\_update(set2)* or **&=** updates set1 with common values & returns null, *set1.isdisjoint(set2)* or **<=** <- True if no common values), *set1.issubset(set2)* True if all values of set1 are in set2, *set1.issuperset(set2)* or **>=** True if all values of set2 are in set1, *set1 is set2* True if same object, *set1 is not set2* True if not same object, *value in set1* True if value is present in set1, *value not in set1* True if not present) | **Math operations**: / true division, // floor division (round to negative infinity), \*\* exponential, +, -. Assign operator: =, +=, -=, \*=, /=. **Operation Precedence**: (), \*\*, +x -x, \* / // %, +, -, comparisons, identity, membership (==, !=, >, >=, <, <=, is, is not, in, not in), logical NOT (not) | **If, Else**: if condition1: elif condition2: else: do | If condition in one line: *print("A") if a > b else print("=") if a == b else print("B")* | **Logical operators**: and & (returns True when both are true); OR | (returns True when one is true); not ! (reverse the result); | **Code comments**: # (single line comment), # after code (in-line comment), ''' any text ''' multiple-line comment, first statement in modules, classes, functions, methods | ⌝ is negate, V is OR, ^ is AND | while condition: else: do (elif not supported in while loops) | *Continue* will end current iteration and start next | *Break* will end loop | **Functions**: repeatable code with arguments, returns results; **\* can be used with parameters when number of values are unknown**. (tuples) E.g., *def myfunc(\*kids): print(kids[1])*, parameters *order* is not important in functions, ***keyword arguments*** can be passed with **\*\***, *default value* can be set in function parameter overridden by value passed when calling function, ***position arguments*** *in functions*: *def func(x, /): print(x) func(10)* | Opposite to position argument is *keyword argument*: *def func(\*, x): print(x) func(x=10)* | **Variable scope**: LEGB: ***local***: within a function; ***Enclosing (non-global)***: nested functions, parent function value can be changed with nested function value with ***nonlocal*** keyword making it one level up; ***Global*** scope: defined at top level of script, outside of functions; **Built-in** scope: widest scope for functions & attributes; **Recursion function**: Function calling itself; e.g. Fibonacci; *Base call* is where recursion ends & results passed back to previous call | **Lambda functions**: anonymous function / no-name; e.g. *x = lambda a : a + 10 print(x(5))* <- *easy way to pass function as a argument to another function*; Object oriented: **Classes** (blueprint for objects, define structure, initial state, & object behavior), **Objects** (instance of class), **Instantiation** (initial settings are configured, *constructor* method instantiates), **Methods** (getter & setter, actions object can perform), **Variables** (stored in objects), Inheritance: Parent class (base class), Child class (derived class), call parent with *super()* brings methods & data to child; Iterators: Data lists, tuples, dictionaries can be iterable with **iter()** class, returns data with **iter & next** methods; Polymorphism: Class polymorphism (same name & functions) & Inheritance polymorphism (super methods or override with specific code): simplicity, code reuse, flexibility, common interface; pandas df = pd.read\_csv(“f.csv”) df.head(10) df.describe() df.info(); |
| **Qualitative data** (names, smell, colors), **Quantitative data** (scores, weight, size) | (Statistics) **Categorical** or characteristics (Nominal, Ordinal [w/ order numbers]), **Numerical** (Interval, Ratio) | **Types of data analytics**: (complexity & value graph) Descriptive (live data, real time), Diagnostic (Automated RCA, why), Predictive (What’s likely, based on history), Prescriptive (To do next, goals & objectives) | **AI**: mimic humans, use logic, if-then rules; **ML**: Statistical techniques, machines improve tasks w/ experience; **DL**: s/w to train itself to perform tasks like speech/image recognition w/ vast data; **ML**: **Supervised** (labeled | Regression, Classification | Feedback | use for prediction | decision trees, logistic regression, SVM | known # of classes), **Unsupervised** (Unlabeled | Dimensionality reduction, Clustering & Association | no feedback | use for Analysis | assigns properties to classify | K-Means, hierarchical, apriori | unknown # of classes), **Reinforcement learning** (real-time decisions, Robot nav, learning tasks, skills acquire, game AI); **Series**: 1-Dimentional, fixed-length, like ordered dictionary, sequence of values of same type, w/ labels as *index*; series = pd.Series([1,2,3], index=['a','b','c']) | series.index | series.array | series.to\_dict()| Missing values are **NaN** | series.**isna**() | series.**notna**() | critical aspect of data analysis is maintain integrity of results while handling missing data; **Dataframes**: 2-Dimentional; rows+columns; dictionary of series sharing same index; csv, excel, json, sql can be loaded as pandas DataFrame; *pd.DataFrame(data)* for dict of lists; *pd.read\_csv('file.csv')* for importing CSV; **df.head(), .tail(), .columns, .info(), .describe();** Access columns: *data['column1']* or *data.column1*; Create new dataframe from csv data: *df2 = pd.DataFrame(csvdata, columns=['column1', 'column2'])*; **Indexes**: hold axis labels + column names; reindexing when reordering data; *df.reindex(columns=[1,2,3])*; **Dropping index**: *df.drop(label, axis=1, inplace=True)* will drop column in same df; **loc**: *df.loc['r']* row index; *df.loc[:,'c1']* column index; df.loc['r1':'r5'] slicing row including 5th row and **iloc**: *df.iloc[[0,2],[0,2]]* access specific rows and columns mentioned; *df.iloc[1:4,3:6]* 2nd row to 4th row & 4th row to 6th row; **Adding dataframes**: df1+df2 works when both have same columns, else filled with NaN | *df1.add(df2, fill\_value=0)* when value don’t exist, fills with 0 (makes it float), .sub, .div, .floordiv, .mul, .pow | **Sending dataframe to functions**: .apply(), .applymap(), .map(); **Sorting dataframes**: *df.sort\_values(by = 'column1', ascending=False)* or *df.sort\_index(ascending=True)*; **Combine datasets**: inner (only common), left (all found in left), right (all found in right), outer (both tables together); *pd.merge(df1,df2, on='column1', how='left')* or *pd.concat([df1, df2], ignore\_index=True)*;  **Descriptive Stats**: Pandas objects have math and stat methods like sum or mean from a series; **Groupby**: *df.groupby('column').sum()*; **Exploratory Data Analysis steps**: Load & confirm data -> Check for missing values (remove, fill, fill w/specific values: Simple [mean/median/mode], KNN [nearest neighbor], Fancy & Iterative *df.at['row', 'column'] = np.nan*) -> check for categorical data -> Address missing data (drop, fill) -> understand distribution/correlation -> identify outliers -> normalize data; **Data science process**: Business understand -> Data understand -> Data Prep -> Modeling -> Evaluation -> Deployment; The *data part in Data science is EDA*.| **Scaling**: **min-max normalization** (0 to 1); useful to bound data to specific range; **Standard normalization**: Z-scaled values; mean of 0 and std of 1; useful when data has varying units or model assumes normally distributed data; **Transform data to scaled values after split of train and test data**; else it *leaks info* and invalidates test results; Use **fit-transform()** on training data; **transform()** on test data; **Matplotlib**: can export into graphics file types & embed in jupyter files; *data = np.random.randint(0,40,size=10) plt.plot(data)* | **Figures**: fig = plt.figure() ax1 = fig.add\_subplot(1,1,1) ax1.hist(np.random.randint(0,5,50), bins=20, color='red', alpha=0.3) ax2.boxplot(np.random.standard\_normal(50)) | **subplots**: fig, axes = plt.subplots(2, 1, figsize=(5,5)) axes[0].plot(np.random.randint(0,5,10)) axes[1].scatter(np.random.randint(0,5,100), np.random.randint(0,5,10)) | **Matplotlib can be called as a method of the pandas dataframe object** (df.plot(color='red')); **Seaborn**: sns.**pairplot**(df\_iris) | ax\_set\_title(), ax.set\_xlabel() | ax.legend() | ax.annotate(); |
| Population: Complete set of data, too large, metrics are parameters, Greek letters; Sample: Subset of population, smaller, manageable group, metrics are statistics, Roman letters, **sample statistics aka point estimators**; **Descriptive statistics** (summarize or describe dataset, no uncertainty, organize/present data in meaningful, results in charts/tables/graphs, *measures of central tendency* [**mean** (symmetric/normal distribution, continuous, x (bar), n for sample, Mu/µ, N for population)/**median** (skewed distribution, has outliers, ordinal data)/**mode** (categorical/nominal data, most common in surveys, unimodal / bimodal / multimodal)], *measure of dispersion/variability* [**range** (diff b/w min & max), **variance** (avg of sq. diff from mean), **std** (sq. root of variance), **MAD** (avg +ve diff from mean), **interquartile range** (middle 50% of dataset)], *measure of distribution* [**symmetric** (bell curve), **skewness** (degree of asymmetry, **zero skew** = symmetric, **+ve skew** = right skew, tail is long on right, e.g. income distribution, **-ve skew** = left skew e.g. grades in easy test), **kurtosis** (tailedness or presence of extreme values, not about peak curve, mesokurtic = 3, leptokurtic >3, platykurtic <3)]) vs **Inferential statistics** (inference from sample, generalize to population, compare/test/predict future, result is probability scores, goes beyond data available, hypothesis test, variance analysis); **Parametric** (ratio/interval, mean) vs. **Non-Parametric** (ordinal/nominal, median, not affected by outliers); **Population variance** () vs. **Sample variance** (Bessel’s correction to divide by N-1 [as 1 degree of freedom is used up in sample mean] to **ensure unbiased estimate** of population variance from sample); **Z-Score**: # of std from the mean for a given datapoint; data value < mean = z-score is -ve; data value > mean = z-score is +ve; **Empirical Rule**: estimate spread of data in normal distribution; ~68% within 1 std; 95% within 2 std; 99.7% within 3 std; **Outliers**: reasons: measurement error, data entry error, natural error, intentional error, sampling error; Z-score method: outlier is +/-3 std from mean; IQR method for outliers; **Steps for Hypothesis testing**: Formulate Hypothesis (H₀: Null hypothesis, default assumption, no effect | H₁: Alternate hypothesis, trying to prove a difference or effect) -> Choose test type (**t-test**: compare means of 2 groups; **ANOVA**: compare means of > 2 groups; **Chi-squared**: for categorical data; **Mann Whitney U-Test**: compare 2 independent samples if coming from same population, non-parametric) -> Set Significance level (typical α = 0.05, 5% chance of incorrectly rejecting null-hypo, Type-1 error; False positive) -> Calculate Test statistic & P value (if p < α, reject the null hypo)| *p-value is probability of obtaining results under null hypothesis, tells how likely the data is on null hypothesis*; *P-value cutoffs* for rejecting null hypothesis are **5% or 1%**; when *n=30, t-statistics is 2 and 2.75* for each cutoff. Test statistic (W) is between 0 and 1. W close or equal to 1, data follows normal distribution; W close to 0, data is not normal distribution. **Linear Regression**: predict quantitative response, y = mx + b, m = slope, b = y intercept, OLS (Ordinary least squares); RSS (Residual Sum of squares): ; **Irreducible Error (Noise)**: inherent variability in data that cannot be predicted, no matter how good model is; Reasons: Unmeasured variables, Inherent stochasticity (randomness), Measurement errors; **Reducible Error**: error that can be reduced w/ better model; **Bias**: error due to assumptions in model; **Variance**: error due to model sensitivity to fluctuations in training set; High variance will overfit the training data (captures training data + noise); **Standardized error**: average amount that estimated mean differs from actual mean; When x and y are unrelated, **t-distribution** within **n-2 degrees of freedom**; If within 2 standardized errors, we fail to reject null hypothesis. Alpha(𝛼) of 0.05 represents reject region is more than 2 SEs from median of 0, so rejecting the null hypothesis or x, y are independent variables. **Set Theory**: object by itself, collection of similar objects to analyze or perform ops; {1,2,3}, unique, unordered, defined; *Capital letters* to represent sets; a ∈ D (element a is in set D). Empty set ∅ or {} or |A|=0. Size of set is |A|; Subset (⊆ or ⊂) & superset; |A| ≤ |B|. Bc means B complement, means elements not in B, but outside of B | A∩B=B∩A | A U B = B U A (all) | A Δ B (symmetric difference, excluding intersection) | A – B or A \ B (difference) | **Associate & Distributive properties**: A∩(B∩C)=(A∩B)∩C | A∩(B∪C)=(A∩B)∪(A∩C) | A∪(B∩C)=(A∪B)∩(A∪C) | DeMorgan’s laws: (A ∪ B)c = Ac ∩ Bc | (A ∩ B)c = Ac ∪ Bc | **Set Functions**: referred as **mapping**; *Domain*: set of all possible i/p to function; *Codomain*: all possible o/p of function; *Range*: actual set of o/p by function from elements in domain (subset of codomain); **Injective**: one-to-one; every element in domain maps to unique element in codomain; no repeats in codomain; **Surjective**: every element in codomain has at least one element from domain; covers entire codomain; **Bijective**: both injective & surjective; have inverses, unique pairing on domain & codomain; Euclidean distance: Straight line between two points in a space, shortest, used in special distance, machine learning; Manhattan distance: City Block Distance: only horizontal or vertical move, use in grid-like env, routing algorithm; Minkowski Distance: has ‘p’; when p=1, it is Manhattan; when p=2, it is Euclidean distance; p is infinity, it becomes chebshev distance, use in flexible distance metrics; Cosine Similarity: cosine of angle between two vectors, focus on orientation, than magnitude; -1 to 1, 1 = vectors are identical, 0=orthogonal, -1=vectors are opposite, use in text similarity, recommendation systems; Hamming distance: 2 strings or vectors of equal length; # of positions at which symbols are different (bits or chars); minimum substitutions req to transform one string to another; used in error detection in data transmit, spell check; Jaccard Distance: dissimilarity b/w 2 sets, based on **jaccard index** (similarity b/w 2 sets), looking at intersection & union; different sizes; 0 (identical) to 1 (different); 1-(intersection)/(union); variable lengths allows in sets; **Probability**: study randomness & uncertainty, *foundation in cybersecurity*; **Sample space**: set of possible results/outcomes from random experiment; referred as S or Ω (omega). **Permutation**: arrange elements in specific order; **Combination**: Select items from large set, where order doesn’t matter; **Joint probability**: P(2 or more events occur simultaneously), P(A ⋂ B); **Independent events**: 2 or more events that do not influence each other’s outcomes; P(A)xP(B); **Conditional probability**: P(event occur, given that another event already occurred), P(A|B), probability of A given B; P(A|B) = P(A ⋂ B)/P(B); 2 ideas leading to Bayes Theorem: Conditional & Total probability; **Bayes Rule**: how to update probability of hypothesis based on new evidence, given prior knowledge; |
| Confusion matric: False negative: Type II error | False positive: Type I error; **Specificity**: TN/(TN+FP) | **Recall**: TP/(TP+FN) | **Precision**: TP/(TP+FP) | **-ve predictive value**: TN/(TN+FN) | **Accuracy**: (TP+TN)/(TP+TN+FP+FN) | F1 = 2x(Precision x Recall)/(Precision + Recall)) | Accuracy: proportion of total # of correct predictions | Precision: ratio of total # of correct +ve and total predicted +ve aka positive predictive value (Tripe-P rule) | Recall: (Sensitivity or True +ve rate (**TPR**): total # of actual positives that were identified correctly | Specificity: proportion of total # of actual negative that were correctly identified (True -ve rate, **TNR**) | F1 score: (harmonic mean) combines precision & recall performance; **ROC (Receiver Operating Characteristics)**: TPR = TP/(TP+FN); FPR = FP/(FP+TN) or (1 – specificity); displays trade-off b/w TPR & FPR; *Benefits*: Robustness to imbalanced datasets, Threshold agnostic, Comparative analysis (higher AUC is better); Train-Test-Split; **stratified split**: training & test datasets are representative of class distribution; K-fold cross validation: training error rate is different from test error rate; often underestimated; can be stratified; **Outliers**: causes: measurement error, data entry error, natural error, intentional error, sampling error; **Z-score method to find outliers**: outlier is +/- 3 std from mean; Mahalanobis Distance: For outlier detection on high-dimensional datasets when correlation is significant; sensitive to overall data distribution; ; Outliers are detected using thresholds (computed using **chi-squared distribution**); Points with distances > threshold are flagged as outliers; Chi-squared distribution = squared mahalanobis distance; e.g. 95% confidence level on outliers identified; **IQR method to find outliers**: Inter-Quartile Range: Q3 – Q1; univariate data; non-parametric; may miss outliers in complex distributions; Lower bound: Q1 – 1.5xIQR; Upper Bound: Q3=1.5xIQR; **Decision tree**: *Root node* (depth=0, level=0) -> *Decision node* -> *Leaf node*; tree-based for regression & classification; works well with noisy or missing data; fast at runtime; effective when interpretability is paramount; Depth: length of longest path from root to leaf; level: each step down; parent node, child node; terminal node: same as leaf nodes; do not split further; Splitting: dividing into two or more sub-nodes; Pruning: remove branches from tree; prevents overfitting; Internal node: same as decision node; test condition, directs data flow; sibling node: same parent; Sub-tree: intermediate node and descendants; ASM: *Attribute Selection Measure*: metric for best split e.g. Gini impurity, entropy, MSE for regression, info gain for classification trees; Tree width: Max # of leaves in tree; **Ensemble approach in ML**: reduce variance; **Bagging**: averaging (regression) or voting (classification) set of observations reduces variance; low overfitting risk; training multiple instances of same model independently on different subsets of training data; e.g. random forest, multiple decision trees are trained on bootstrapped datasets; **Boosting**: sequential training on multiple instances of same model; final prediction is weighted combinations of all model outputs; reduce bias by focusing on weakness of previous model; works well for imbalanced datasets; e.g. AdaBoost (Adaptive boosting), Gradient Boosting; high overfitting risk; **Boosting process**: initialize – Calculate error – update weights – train next model – combine model – repeat – final model; ***Unsupervised ML***: Isolation Forest (iForest): identify outliers, anomalies in datasets; high-dimensional datasets, where z-score or IQR may struggle. Tree-based method; isolates data points that are different; uses decision trees but for isolating anomalies/outliers rather than classification or regression; fewer splits on forest have high isolation score; efficient, no assumptions (non-parametric), works with *non-linear data*; **IF process**: Random sample/Tree building – Path length calculation – Anomaly scoring (*points with higher score/shorter path length are anomalies*) – Classification (anomalies or normal based on thresholds); **Path length calculation**: # of splits to isolate it on isolation tree; **Average path length**: for each datapoint across all trees; s(x,n) = 2 power(- E(h(x))/c(n)); between 0 and 1; E(h(x)) small means isolated quickly; **Contamination parameter**: expected portion of outliers in dataset; guideline, not a strict requirement; **DBSCAN**: density-based spatial clustering of app w/ noise; identify clusters in data by grouping points closely packed; useful on arbitrary-shared clusters; **Core points** (min # of neighboring points within specified distance) **| Border points** (with epsilon distance of core points; do not have enough neighbors to be considered core) **| Noise points** (do not belong to any cluster)**;** *Pros*: Do not require # of clusters predefined, detect noise & clusters, outliers do not affect model, works w/ cluster of arbitrary shapes, only 2 parameters to tune; *Cons*: sensitive to **epsilon & minPts**, struggle /w varying density clusters; DBSCAN **applied in** Geospatial data, Anomaly detection, Image segmentation; **DBSCAN algo steps**: Initialize parameters – every point in dataset – find neighbor points – check current point is core point (if # of neighbor points (+self) > minPts, it is core point; if neighbor points < minPts, it is noise); *Expand the cluster* (recursively visit all neighbor points, if not visited earlier, mark it part of current cluster; if already core points, continue), *Repeat* until all unvisited points are processed; **Hyperparameter tuning**: done before the training; *minPts* (at least D+1, increases with dimensions, for noisy data use high minPts as only dense region form cluster, small clusters = low minPts) & *epsilon* (k = # of clusters = minPts – 1, **elbow** method (to determine epsilon) in plot when sorted distance b/w point to kth nearest neighbor). **SVM**: Can be for both classification (SVC) or regression (SVR). **Large margin classification**: SVM classifier fitting widest street b/w class (2 parallel lines). In p-dimensional space, its **hyperplane** (flat affine of dimension p-1). hyperplane=line in 2D, plane in 3D; **margin**=distance from solid line (hyperplane) to either of two parallel lines; support vectors are data points on margins. **Hard margin classification**: strictly imposing observation (no matter which side of hyperplane) is assigned a class; No room for misclassification; *2 issues*: works only when data is linearly separable; sensitive to outliers; impractical; **Soft margin**: allow some observations to be on incorrect side of margin or hyperplane; observations on wrong side of hyperplane (training observations) misclassified by SVC; **C**: (for soft margin): high C -> strict; low C -> relax; trade-off b/w margin and error; ε=0, correct side; ε>0,wrong side or margin; ε>1, wrong side of hyperplane; **SVM kernel options**: **polynomial** (relationship b/w features); **Radial Basis Fn (guassian**; default choice; no clear understand of data distribution); **Sigmoid** (suspect the data behaves like neural n/w); **Kernel trick**: helps in high-dim space non-linear boundary; **K>2**: multiclass OCSVM; one-class per class; ensemble method; cluster-based; **feature engineering;** **OCSVM**: adapt from SVM; params: nu (**v**); |