Cross-Validation (Cross-Val)
Objective: partition data into folds to estimate model generalization.
Cross-val tests model on new data. Split data into folds, train on most, test on one, repeat. Avg performance is reliable.
Pros: Better perf estimate than single split; Detects overfitting.
Cons: More time/compute for many folds/large data; Tricky for time-series.

Convex Optimization (Convex-Opt)
Objective: minimize convex $f(x)$ s.t. convex $g_i(x) \leq 0$, affine $h_j(x) = 0$.
Convex opt finds global best in bowl-shaped space. Used in SVMs, regression.
Pros: Guarantees global min; Efficient solvers.
Cons: Not all probs convex, need approx; Heavy for large probs.

Solving LP/QP (Quick Guide)
Form: LP $\min_x c^\top x$ s.t. $Ax \leq b$, $A_{eq}x = b_{eq}$; QP $\min_x \frac{1}{2} x^\top Q x + c^\top x$ s.t. linear constraints, $Q \succeq 0$.
Steps: (1) collect vars into x, write all linear constraints; (2) simplify equalities by eliminating fixed vars; (3) small/medium: use interior-point LP/QP solver; large/sparse: use (projected) gradient / coordinate descent. Example QP: L2-regularized least squares $\min_w \frac{1}{2}\|Xw - y\|_2^2 + \frac{\lambda}{2}\|w\|_2^2$ is a convex QP; closed form $(X^\top X + \lambda I)w = X^\top y$ or solve with CG. KKT: primal feas., dual feas., stationarity, complementary slackness certify optimality.

Gradient Descent (GD)
Update: $\theta^{t+1} = \theta^t - \eta \nabla f(\theta^t)$.
GD updates params opposite grad of loss to min errors.
Pros: Simple to implement; Good for convex.
Cons: Slow on large data (full set/step); Stuck in local min for non-convex.

Stochastic Gradient Descent (SGD)
Update (single sample i): $\theta^{t+1} = \theta^t - \eta \nabla \ell_i(\theta^t)$.
SGD like GD but updates w/one random point, faster/noisier.
Pros: Faster on large data; Escapes local min via noise.
Cons: Noisy, erratic; Needs LR scheduling.

Mini-batch Gradient Descent
Update (batch B_t): $\theta^{t+1} = \theta^t - \eta \frac{1}{|B_t|} \sum_{i \in B_t} \nabla \ell_i(\theta^t)$.
Mini-batch GD updates w/small batches, balances GD/SGD.
Pros: Faster than GD, less noisy than SGD; GPU-efficient.
Cons: Batch size tuning needed; Can stuck in local min.

Data Augmentation
Objective: minimize loss on augmented dataset $\min_\theta \sum_{(x,y) \in \mathcal{D}_{aug}} \ell(y, f_\theta(x))$.
Data aug mods existing ex (rotate, noise) for robust models.
Pros: More data w/o collect; Better gen, esp images.
Cons: May add unreal data; Compute-heavy in train.

Lagrangian
Formulation: $L(w, \lambda, \nu) = f(w) + \sum_i \lambda_i g_i(w) + \sum_j \nu_j h_j(w)$, $\lambda_i \geq 0$.
Lagrangian combines obj func w/constraints via mults for opt pts.
Pros: Solves eq/ineq constraints; Base for SVMs.
Cons: Complex math; Needs KKT checks.

Dual Lagrangian
Dual problem: $\max_{\lambda \geq 0, \nu} g(\lambda, \nu)$ where $g(\lambda, \nu) = \inf_w L(w, \lambda, \nu)$.
Dual reformulates primal, often easier, esp kernels.
Pros: Simplifies computation in many cases; Enables kernel trick for non-linear problems.
Cons: May increase complexity for some formulations; Requires careful handling of dual variables.

K-Nearest Neighbors (KNN)
Objective: no parametric minimization; prediction $\hat{y}(x) = \text{mode}\{y_i : x_i \in \mathcal{N}_k(x)\}$.
KNN classifies a new data point based on the majority label of its 'k' closest neighbors in the training data, using distance metrics like Euclidean.
Pros: Simple and intuitive, no training phase needed; Works well for non-linear data.
Cons: Slow for large datasets (computes distances at prediction time); Sensitive to irrelevant features and noise.

Naive Bayes
Objective: $\min_\theta - \sum_i \log p_\theta(y_i) p_\theta(x_i | y_i)$ with $p(x|y) = \prod_j p(x_j|y)$.
Naive Bayes is a probabilistic classifier that applies Bayes' theorem, assuming features are independent, to predict class probabilities.
Pros: Fast and efficient, especially for high-dimensional data like text; Performs well even with the 'naive' independence assumption.
Cons: Assumption of feature independence often unrealistic; Struggles with zero-probability issues (use smoothing).

Linear Discriminant Analysis (LDA)
Objective: $\min_{\mu_k, \Sigma, \pi_k} - \sum_i \log(\pi_{y_i} \mathcal{N}(x_i | \mu_{y_i}, \Sigma))$.
LDA projects data onto a lower-dimensional space to maximize class separability, assuming Gaussian distributions and equal covariances.
Pros: Good for dimensionality reduction while preserving class info; Computationally efficient.
Cons: Assumes normality and equal covariances, which may not hold; Linear boundaries only.

Logistic Regression
Objective: $\min_w \sum_i \log(1 + \exp(-y_i w^\top x_i)) + \lambda \|w\|_2^2$.
Logistic Regression models the probability of binary outcomes using a sigmoid function on a linear combination of features.
Pros: Interpretable coefficients show feature importance; Handles binary and multiclass (via one-vs-rest).
Cons: Assumes linear decision boundaries; Sensitive to multicollinearity.

Support Vector Machines (SVM)
Soft-margin primal: $\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C \sum_i \xi_i$ s.t. $y_i(w^\top x_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$.
SVM finds the hyperplane that best separates classes with the maximum margin, using support vectors.
Pros: Effective in high-dimensional spaces; Robust to overfitting with proper regularization.
Cons: Computationally intensive for large datasets; Sensitive to choice of kernel and parameters.

Kernel SVM
Dual: $\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$ s.t. $\sum_i \alpha_i y_i = 0$, $0 \leq \alpha_i \leq C$.
Kernel SVM extends SVM to non-linear data by mapping to higher dimensions via kernels (e.g., RBF) without explicit transformation.
Pros: Handles complex, non-linear boundaries; Versatile with different kernels.
Cons: More computationally expensive; Risk of overfitting if kernel not chosen well.

Linear Regression
Objective: $\min_w \sum_i (y_i - w^\top x_i)^2 = \min_w \|Xw - y\|_2^2$.
Linear Regression fits a line to data by minimizing squared errors, predicting outputs as a linear combination of inputs.
Pros: Simple and interpretable; Fast to train.
Cons: Assumes linearity; poor for complex relationships; Sensitive to outliers.

Ridge Regression
Objective: $\min_w \sum_i (y_i - w^\top x_i)^2 + \lambda \|w\|_2^2$.
Ridge Regression adds L2 regularization to linear regression to shrink coefficients and handle multicollinearity.
Pros: Reduces overfitting and stabilizes estimates; Good for correlated features.
Cons: Includes all features (no selection); Bias introduced by regularization.

Lasso Regression
Objective: $\min_w \sum_i (y_i - w^\top x_i)^2 + \lambda \|w\|_1$.
Lasso Regression uses L1 regularization, which can set some coefficients to zero for feature selection.
Pros: Performs automatic feature selection; Handles multicollinearity.
Cons: Can be unstable with highly correlated features; Bias like Ridge.

Kernel Ridge
Objective: $\min_\alpha \|K\alpha - y\|_2^2 + \lambda \alpha^\top K \alpha$.
Kernel Ridge combines Ridge regression with kernels for non-linear fitting.
Pros: Captures non-linear patterns; Regularization prevents overfitting.
Cons: Computationally heavy for large data; Kernel tuning required.

Support Vector Regression (SVR)
Soft-margin primal: $\min_{w,b,\xi^*} \frac{1}{2}\|w\|^2 + C \sum_i (\xi_i + \xi_i^*)$ s.t. $|y_i - w^\top x_i - b| \leq \epsilon + \xi_i$, $\xi_i, \xi_i^* \geq 0$.
SVR adapts SVM for regression, finding a function that deviates from actual values by at most epsilon.
Pros: Robust to outliers; Effective in high dimensions.
Cons: Sensitive to parameter choice (C, epsilon); Slow for large datasets.

Kernel SVR
Objective: same SVR primal in feature space; dual uses kernel $K(x_i, x_j)$.
Kernel SVR uses kernels for non-linear regression in SVR.
Pros: Handles complex non-linear data; Flexible with kernels.
Cons: Increased complexity and compute; Overfitting risk.

Polynomial Regression
Objective: $\min_w \sum_i (y_i - w^\top \phi(x_i))^2$ with polynomial features $\phi(x)$ (e.g. degree d).
Polynomial Regression fits higher-degree polynomials to capture non-linear trends.
Pros: Simple extension of linear regression; Good for curved relationships.
Cons: Prone to overfitting with high degrees; Extrapolation can be poor.

K-Means
Objective: $\min_{\{c_k\}, \{z_i\}} \sum_i \|x_i - c_{z_i}\|^2$ with $z_i \in \{1, \ldots, k\}$.
K-Means partitions data into k clusters by minimizing within-cluster variance, assigning points to nearest centroids.
Pros: Simple and scalable; Fast convergence.
Cons: Needs k specified; sensitive to initialization; Assumes spherical clusters.

Gaussian Mixture Model (GMM)
Objective: $\min_{\pi_k, \mu_k, \Sigma_k} - \sum_i \log \sum_k \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k)$ s.t. $\pi_k \geq 0$, $\sum_k \pi_k = 1$.
GMM models data as a mixture of Gaussian distributions, using EM to estimate parameters.
Pros: Handles elliptical clusters and soft assignments; Probabilistic outputs.
Cons: Slower than K-Means; sensitive to init; Assumes Gaussian components.

Perceptron
Objective (implicit): minimize misclassification by updates $w \leftarrow w + y_i x_i$ on errors.
Perceptron is a single-layer neural network for linear classification, updating weights on errors.
Pros: Basic building block of NNs; Converges for linearly separable data.
Cons: Only linear; no hidden layers; Doesn't handle XOR-like problems.

Multi-Layer Perceptron (MLP)
Objective: $\min_\theta \sum_i \ell(y_i, f_\theta(x_i)) + \lambda \|\theta\|^2$ (cross-entropy/MSE).
MLP adds hidden layers to Perceptron for non-linear learning via backpropagation.
Pros: Universal approximator for functions; Handles complex data.
Cons: Prone to overfitting; needs regularization; Black-box; hard to interpret.

Convolutional Neural Networks (CNN)
Objective: same as MLP, $\min_\theta \sum_i \ell(y_i, f_\theta(x_i)) + Omega(\theta)$, with conv/pooling layers.
CNN uses convolutional layers for feature extraction, ideal for grid data like images.
Pros: Excellent for spatial hierarchies (e.g., images); Parameter sharing reduces compute.
Cons: Requires large data and GPU; Overfits without augmentation.

Sequential Minimal Optimization (SMO)
Objective: solve SVM dual $\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j K_{ij}$ s.t. $0 \leq \alpha_i \leq C$, $\sum_i \alpha_i y_i = 0$ by updating two α at a time.
SMO solves the SVM dual by updating two Lagrange multipliers at a time while keeping constraints satisfied.
Pros: Efficient for large SVM problems; Avoids large QP solvers; Works well with kernels.
Cons: More complex to implement than simple GD; Speed depends on heuristics for picking pairs.

RANSAC
Objective: find parameters θ maximizing inliers while being robust to outliers.
RANSAC repeatedly samples minimal subsets, fits a model, and counts inliers to find a robust fit under many outliers.
Pros: Very robust to outliers; Simple concept; Works well for geometric vision tasks.
Cons: Needs many iterations if inlier ratio low; Requires thresholds and max-iter tuning.

Expectation Maximization (EM)
Objective: maximize $\ell(\theta) = \sum_i \log \sum_Z p(x_i, Z; \theta)$. E-step: compute $Q(\theta|\theta^{old}) = \mathbb{E}_{Z|X, \theta^{old}}[\log p(X, Z; \theta)]$. M-step: $\theta^{new} = \arg\max_\theta Q(\theta|\theta^{old})$.
EM maximizes a latent-variable likelihood by alternating: E-step (compute posteriors/expectations) and M-step (maximize expected complete log-likelihood).
Pros: Handles missing/latent variables naturally; Closed-form updates for models like GMM.
Cons: Converges only to local maxima; Can be slow; Sensitive to initialization.

Dimensionality Reduction
Objective: find mapping $z = f(x)$ preserving variance, distances, or class info.
Dimensionality reduction maps high-dim data to lower-dim space while preserving structure (variance, distances, or class info).
Pros: Reduces storage and computation; Helps visualization and denoising.
Cons: May discard useful information; Choice of method and target dim is non-trivial.

Feature Selection
Objective: $\min_w \sum_i \ell(y_i, f_w(x_i)) + \lambda \|w\|_0$ (NP-hard, relaxed to L1 or greedy).
Feature selection chooses a subset of input features (filter, wrapper, embedded methods) instead of transforming them.
Pros: Improves interpretability; Can reduce overfitting and training time.
Cons: Search can be expensive; Risk of discarding informative but weak features.

Linear Dimensionality Reduction
Objective: $\max_W \operatorname{tr}(W^\top SW)$ s.t. $W^\top W = I$ (PCA), or ratio of between/within scatter (LDA).
Linear DR finds projections $z = W^\top x$ that keep most variance or class separation (e.g., PCA, LDA).
Pros: Simple and fast; Often has eigenvalue/eigenvector closed forms.
Cons: Only captures linear structure; Fails on curved manifolds (non-linear relations).

Singular Value Decomposition (SVD)
Objective: $X = U\Sigma V^\top$, best rank-k approx: $\min_{\operatorname{rank}(X_k) \leq k} \|X - X_k\|_F^2$ solved by $X_k = U_k \Sigma_k V_k^\top$.
SVD: $X = U\Sigma V^\top$, with orthogonal U, V and singular values in Σ.
Pros: Basis of PCA and low-rank approximations; Optimal rank-k approximation in Frobenius norm.
Cons: Expensive on very large matrices; Often needs truncated or randomized SVD.

Principal Component Analysis (PCA)
Objective: $\max_W \sum_i \|W^\top x_i\|^2$ s.t. $W^\top W = I$ (find top eigenvectors of covariance).
PCA finds directions of maximum variance (eigenvectors of covariance, or top right-singular vectors of X).
Pros: Unsupervised linear DR; Decorrelates features; Often improves downstream methods.
Cons: Components are linear and not label-aware; Sensitive to scaling and outliers.

How to Perform PCA
Given data matrix $X \in \mathbb{R}^{n \times d}$ (rows = samples, columns = features).
1. **Center (and scale) data**: compute mean $\mu = \frac{1}{n} \sum_i x_i$, set $\tilde{x}_i = x_i - \mu$; optionally standardize each feature to unit variance.
2. **Compute covariance or use SVD**: covariance $S = \frac{1}{n} \tilde{X}^\top \tilde{X}$ and solve eigenproblem $Sw_k = \lambda_k w_k$; or compute SVD $\tilde{X} = U\Sigma V^\top$ and take columns of V as principal directions.
3. **Sort components**: sort eigenvalues (or singular values) descending; choose top k such that $\sum_{j=1}^{k} \lambda_j / \sum_{j=1}^{d} \lambda_j$ reaches desired variance (e.g. 95%).
4. **Project data**: form $W_k = [w_1, \ldots, w_k]$ and compute low-dim representation $Z = \tilde{X} W_k \in \mathbb{R}^{n \times k}$ (rows are principal component scores).
5. **(Optional) Reconstruct**: approximate original data by $\hat{X} = ZW_k^\top + \mu$ for visualization or compression error analysis.
Notes: Always center data; scaling is crucial if features have different units. Use truncated/randomized SVD for large, sparse, or high-dimensional X.

Kernel PCA
Objective: same as PCA in feature space; eigendecomposition of centered kernel matrix K.
Kernel PCA applies PCA in an implicit feature space using a kernel matrix instead of the covariance of raw features.
Pros: Captures non-linear structure; Works with same kernels as Kernel SVM.
Cons: Needs storing and eigendecomposing $N \times N$ kernel matrix; Less interpretable than standard PCA.

Whitening
Objective: enforce $\operatorname{Cov}(z) = I$ via linear transform $z = \Sigma^{-1/2} U^\top x$ after PCA.
Whitening transforms data so that it has zero mean and identity covariance (decorrelated, unit variance). Often done after PCA.
Pros: Removes linear correlations; Useful preprocessing for some models and ICA.
Cons: Can amplify noise in low-variance directions; Requires good covariance estimate.

Looking at Learning Curves
Objective: plot train and test error vs. training set size or training iterations to diagnose bias/variance.
Healthy curve: high-loss start, rapid decreases, gradually flattens as model converges
Underfitting: Decrease slightly, flat at hight loss
Large learning rate, small batch, noisy data: Decrease with oscillation
Too small learning rate, bad initialization: Initially flat then drop
Overfitting: Good decreasing for training, val/test loss drop then increase
Pros: Helps diagnose high-bias vs. high-variance; Guides whether to get more data or change model complexity.
Cons: Requires repeated training; Interpretation can be ambiguous with noisy curves.

PCA, Kernel PCA, Whitening
Objectives: PCA max variance in input; Kernel PCA max variance in feature space; Whitening enforce $\operatorname{Cov}(z) = I$.
Similarities: All linear transforms in some space; Used for preprocessing and dimensionality reduction.
Differences: PCA linear in input space; Kernel PCA non-linear via kernels; Whitening rescales to identity covariance (often after PCA) instead of just keeping top-variance directions.