# Checkpoint 2 Guide: EDA, Feature Selection, and Feature Engineering

## Exploratory Data Analysis (EDA)

* **Understand distributions**
  + Inspect univariate distributions (histograms, KDEs, boxplots).
* **Check for missing data**
  + Quantify missingness, visualize patterns.
  + Tools:
    - [pandas.isna](https://pandas.pydata.org/docs/reference/api/pandas.isna.html)
    - [missingno](https://github.com/ResidentMario/missingno) (matrix, heatmap, dendrogram visualizations)
* **Relationships between variables**
  + Scatterplots, correlation heatmaps, group summaries.
* **Outlier detection**
  + Identify points that may distort modeling.
  + Tools:
    - [scipy.stats.zscore](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html)
    - [sklearn.ensemble.IsolationForest](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html)
    - Boxplots via [seaborn.boxplot](https://seaborn.pydata.org/generated/seaborn.boxplot.html)
* **Time-series and lag structure**
  + Create exploratory lag features *in a notebook only* to inspect autocorrelation, partial autocorrelation, or lag-target scatterplots.
  + Tools:
    - [pandas.DataFrame.shift](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shift.html)
    - [statsmodels.graphics.tsaplots.plot\_acf](https://www.statsmodels.org/stable/generated/statsmodels.graphics.tsaplots.plot_acf.html)
    - [statsmodels.graphics.tsaplots.plot\_pacf](https://www.statsmodels.org/stable/generated/statsmodels.graphics.tsaplots.plot_pacf.html)
    - Interactive plotting with [plotly.express.line](https://plotly.com/python/line-charts/)

## Feature Selection

* **Domain knowledge (primary filter)**
  + Do not include variables just because they exist.
  + Keep features with a plausible, defensible connection to the target.
  + This step combats the **curse of dimensionality**: too many irrelevant predictors leads to overfitting.
* **Leakage check**
  + Drop variables known only post-outcome or that encode the target directly.
* **Low-information features**
  + Drop near-constant or high-missingness columns.
    - **Note:** “Near constant” is not a fixed threshold — it’s contextual. Use domain knowledge to decide whether small variation is meaningful. For example, if request latency varies only from 2.3 ms to 2.4 ms, it’s implausible this explains differences in click-through rate; you should treat it as constant. But a drug dosage shift from 2.3 mg/kg to 2.4 mg/kg may have clinically significant effects and must not be discarded. The same numerical range can be irrelevant in one setting and critical in another.
* **Redundancy checks**
  + Remove highly collinear features; keep one representative.
  + Tools: [pandas.DataFrame.corr](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html), VIF via [statsmodels](https://www.statsmodels.org/stable/generated/statsmodels.stats.outliers_influence.variance_inflation_factor.html)
* **Model-based diagnostics**
  + Use feature importances as *supporting evidence* but not as the main decision tool.
  + Tools: [SelectFromModel](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html), tree-based models (RandomForest\*, XGB\*).

## Feature Engineering

* **Domain-driven transformations**
  + Ratios, differences, interaction terms suggested by subject knowledge.
* **Dimensionality reduction**
  + Tools: [PCA](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html), [UMAP](https://umap-learn.readthedocs.io/en/latest/)
* **Time features**
  + Extract lags, rolling statistics, seasonality components.

## Iteration Between EDA and Feature Work

* **Expect to loop**
  + EDA, feature selection, and feature engineering are not one-time steps. They should inform each other in cycles.
* **Workflow**
  1. Explore data distributions and relationships in EDA.
  2. Propose engineered features based on patterns or domain ideas.
  3. Test them quickly with simple models or diagnostics.
  4. If results are unhelpful (no signal, redundancy, instability), discard or revise.
  5. Go back to EDA with new questions (e.g., “Why didn’t lag-7 help? Maybe seasonality is longer”).
* **Mindset**
  + Don’t hoard features “just in case.” Keep the pipeline clean and only include features that survive scrutiny.
  + Each iteration should strengthen your understanding of the data, not just expand the feature set.

## Deliverables

* **Written / Conceptual**
  + A **data audit summary**: key distributional facts, missingness, correlations.
  + A **list of dropped features** with justification (low information, leakage, redundancy, irrelevance).
  + A **list of engineered features** with rationale (domain knowledge, lags, interactions, transformations).
* **Code / Repo Artifacts**
  + **notebooks/eda.ipynb**:
    - Visualizations and descriptive stats.
    - Missingness analysis.
    - Correlation heatmaps, scatterplots, outlier checks.
    - Exploratory lag analysis (ACF/PACF, lag vs. target plots).
  + **notebooks/feature\_selection.ipynb**:
    - Shows feature elimination decisions (variance threshold, correlation pruning, leakage checks).
    - Logs results in a table (results/feature\_selection.csv).
  + **src/features/transformers.py**:
    - Custom transformers implementing engineered features (e.g., lag features, ratios, domain-driven encodings).
  + **src/features/preprocessing.py**:
    - Main preprocessing pipeline that drops unused features, integrates engineered features, and can be reused in modeling.
  + **notebooks/pipeline\_demo.ipynb**:
    - Demonstrates the pipeline fitting/transforming data.
  + **results/eda/ folder**:
    - Key plots saved as .png or .html.
  + **Updated schema** (schema.json or schema.yaml):
    - Defines available features, their types, and any transformations applied.
  + **(Optional) Tests in tests/test\_pipeline.py**:
    - Verify engineered features are generated correctly.
    - Ensure pipeline respects train/test splits (no future leakage).