**🚀 COMPLETE CHECKLIST — "THINGS TO DO BEFORE MODEL TRAINING"**

We’ll go step-by-step:  
**1️⃣ Understand Data → 2️⃣ Clean Data → 3️⃣ Prepare Features → 4️⃣ Handle Imbalance → 5️⃣ Split + Scale → 6️⃣ Feature Selection → 7️⃣ Validate**

**🔹 1. Understand the Data (Data Understanding & EDA)**

Before doing anything, you must *understand what your data represents.*

**✅ Steps:**

* **Load and inspect**
* df.head(), df.info(), df.describe()
* **Check shape, dtypes, missing values**
* **Understand target variable**: classification/regression? imbalance?
* **Visualize**
  + Histograms, boxplots → distribution
  + Pairplot, heatmap → relationships
  + Countplot → categorical variable distribution
* **Check correlations**
* df.corr()
* **Identify data problems**
  + Duplicates
  + Missing data
  + Wrong data types
  + Outliers
  + Skewness

🧠 *Goal:* build intuition about your data — how features relate to target.

**🔹 2. Data Cleaning**

**✅ Handle Missing Values**

Different approaches depending on data type:

| **Type** | **Techniques** |
| --- | --- |
| Numeric | Mean, median, KNN Imputer, Regression imputation |
| Categorical | Mode, most frequent, or a new category ("Unknown") |
| Advanced | MICE, autoencoders |

🧩 In Python:

from sklearn.impute import SimpleImputer

imp = SimpleImputer(strategy='median')

X['col'] = imp.fit\_transform(X[['col']])

**✅ Handle Duplicates**

df = df.drop\_duplicates()

**✅ Fix Data Types**

Convert categorical to “category”, date to “datetime”, etc.

**✅ Handle Inconsistent Data**

Example: Gender = “M”, “Male”, “male” → standardize to one.

**🔹 3. Outlier Analysis**

As discussed earlier:

* Use **Z-score**, **IQR**, or **Isolation Forest** to find anomalies.
* Then **decide** whether to:
  + Remove (data error)
  + Cap/Floor (Winsorize)
  + Keep (real anomaly like fraud)

**🔹 4. Feature Engineering**

Transform or create new features that add predictive power.

**✅ a. Encoding Categorical Variables**

| **Type** | **Technique** | **Example** |
| --- | --- | --- |
| Nominal (no order) | One-Hot Encoding | Red, Blue, Green |
| Ordinal (ordered) | Label Encoding or custom mapping | Small < Medium < Large |
| High Cardinality | Target Encoding / Frequency Encoding | 1,000+ categories |

from sklearn.preprocessing import OneHotEncoder, LabelEncoder

**✅ b. Feature Transformation**

* **Normalization** → scale values between 0–1
* **Standardization** → mean = 0, std = 1
* **Log / sqrt transform** → handle skewed distributions
* **Box-Cox / Yeo-Johnson** → advanced skew correction

from sklearn.preprocessing import StandardScaler, MinMaxScaler

**✅ c. Feature Creation**

Create features based on domain logic, e.g.:

* Age = CurrentYear - BirthYear
* TotalSpent = Quantity \* UnitPrice
* Text data → word counts, TF-IDF
* Time data → extract month, weekday, hour

**✅ d. Feature Encoding for Date/Time**

Convert datetime columns into:

* Year, Month, Day, Hour
* Weekday/Weekend
* Time since event (duration feature)

**✅ e. Interaction Features**

* Create polynomial or ratio features

from sklearn.preprocessing import PolynomialFeatures

**🔹 5. Feature Scaling**

Used to ensure all features contribute equally to distance-based or gradient-based models.

| **Technique** | **Description** | **When to Use** |
| --- | --- | --- |
| **StandardScaler** | Mean=0, Std=1 | Most ML models |
| **MinMaxScaler** | [0,1] range | Neural networks |
| **RobustScaler** | Uses median and IQR | Outliers present |
| **Normalizer** | Scales each sample to unit norm | Text, vectors |

**🔹 6. Handle Imbalanced Data**

If one class dominates heavily, models get biased.

**Techniques:**

| **Category** | **Methods** |
| --- | --- |
| **Resampling** | Oversampling (SMOTE, ADASYN), Undersampling |
| **Algorithmic** | Class weights, cost-sensitive learning |
| **Synthetic Data** | GANs, Variational Autoencoders |

from imblearn.over\_sampling import SMOTE

sm = SMOTE()

X\_res, y\_res = sm.fit\_resample(X, y)

**🔹 7. Dimensionality Reduction (optional but important)**

When you have **too many features**, apply:

| **Method** | **Description** | **Use Case** |
| --- | --- | --- |
| **PCA** | Linear projection to reduce correlated features | Numeric high-dim data |
| **t-SNE / UMAP** | Nonlinear manifold learning | Visualization |
| **Autoencoder** | Neural network-based reduction | Nonlinear tabular/image data |

**🔹 8. Feature Selection**

After feature creation/scaling, remove unhelpful or redundant features.

**Techniques:**

| **Method** | **Description** |
| --- | --- |
| **Filter Methods** | Correlation, Chi-square, Mutual Info |
| **Wrapper Methods** | RFE, Forward/Backward selection |
| **Embedded Methods** | Lasso (L1), Tree-based feature importance |

**🔹 9. Regularization Check**

Apply **L1/L2/ElasticNet** or **Dropout** to prevent overfitting (depending on model).

**🔹 10. Data Splitting**

Split data into:

* **Train**: model learns
* **Validation**: tuning hyperparameters
* **Test**: final unbiased evaluation

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Optional: **Stratified Split** for imbalanced classes.

**🔹 11. Cross-Validation Setup**

Use k-fold cross-validation to evaluate model stability.

from sklearn.model\_selection import cross\_val\_score

| **Type** | **Description** |
| --- | --- |
| **K-Fold** | Split into K parts (commonly 5 or 10) |
| **Stratified K-Fold** | Keeps class ratio same per fold |
| **TimeSeriesSplit** | For time-dependent data |

**🔹 12. Train-Test Leakage Check**

Before training:

* Ensure **no information from test leaks into train** (especially after scaling or encoding).
* Always fit transformers on **train** only, then transform **test**.

**🔹 13. Baseline Model Creation**

Train a simple model first to get a reference accuracy:

* Logistic Regression, Decision Tree, DummyClassifier
* Use as benchmark before complex models (XGBoost, ANN)

**🔹 14. Pipeline Setup (Best Practice)**

Bundle preprocessing + model into one pipeline so it’s reproducible and avoids leakage.

from sklearn.pipeline import Pipeline

pipeline = Pipeline([

('scaler', StandardScaler()),

('model', LogisticRegression())

])

**🔹 15. Data Documentation (Optional but Crucial)**

Keep track of:

* Feature definitions
* Missing value treatments
* Outlier handling rules
* Encoding/scaling methods
* Train-test splits
* Evaluation metrics

This is essential for reproducibility and audits.

**✅ Summary — The 15 Key Steps Before Model Training**

| **Stage** | **What You Do** |
| --- | --- |
| 1️⃣ | Understand data (EDA) |
| 2️⃣ | Clean data (missing, duplicates, types) |
| 3️⃣ | Detect and handle outliers |
| 4️⃣ | Engineer new features |
| 5️⃣ | Encode categorical features |
| 6️⃣ | Scale numeric data |
| 7️⃣ | Handle class imbalance |
| 8️⃣ | Reduce dimensionality (PCA/Autoencoder) |
| 9️⃣ | Select best features |
| 🔟 | Apply regularization |
| 11️⃣ | Split data |
| 12️⃣ | Use cross-validation |
| 13️⃣ | Prevent leakage |
| 14️⃣ | Build baseline |
| 15️⃣ | Create full pipeline |