Tab 1

**in database how we have to find the fetures which are dependent and which are independent in prophet or lstm?**

# **Determining Dependent vs. Independent Features for Time-Series Models**

When you’re building a Prophet or LSTM forecast, the first step is to decide which column in your database is the **target** (dependent) variable and which other columns will serve as **covariates** (independent variables). Here’s a step-by-step guide:

## **1. Identify Your Dependent Variable**

* Look for the column you want the model to predict
* Common examples:
  + Sales volume
  + Number of website visits
  + Dengue case counts
* In SQL: it’s the column you SELECT as your y or target

## **2. List Potential Independent Variables**

* Pull in columns that logically influence your target
  + Marketing spend, price, promotions
  + Weather data (temperature, rainfall)
  + Holiday or event flags
* Ensure each candidate:
  + Has values for the entire historical period
  + Can be known (or reasonably forecasted) into the future

## **3. Run Exploratory Feature Analysis**

* Correlation matrix
  + Pearson/Spearman to spot linear relationships
* Cross-correlation function (CCF)
  + Finds lead/lag effects between series
* Granger causality test
  + Checks if past values of X improve prediction of Y
* Visual checks
  + Scatterplots, lag plots, rolling averages

## **4. Implementing in Prophet**

1. Prepare a DataFrame with columns ds (date) and y (target).

Add each independent variable as a regressor:  
 m = Prophet()

m.add\_regressor('promo')

m.add\_regressor('temperature')

m.fit(df\_train)

1. When making future DataFrame, include those same columns out to your forecast horizon.
2. Prophet will treat them as exogenous inputs to improve forecasts.

## **5. Implementing in LSTM**

1. Select your target column for y.
2. Gather independent columns into a feature matrix X.
3. Normalize or scale all features.
4. Create supervised learning sequences:
   * For time step (t), inputs (X\_{t-n+1:t}) and output (y\_t)
   * Use a sliding window approach
5. Build and train your LSTM on (X, y) pairs, then predict future steps by iteratively feeding back.

## **6. Automating & Refining Feature Selection**

* Recursive Feature Elimination (RFE) with a simple regressor
* LASSO or ElasticNet to shrink irrelevant features
* Tree-based models (Random Forest) for feature importances
* Time-series cross-validation to guard against overfitting

## **Next Steps & Tips**

* Sketch out a pipeline:
  1. Extract raw tables with SQL
  2. Join on date keys
  3. Run automated correlation/importance filters
  4. Fit Prophet or LSTM
  5. Validate with hold-out periods
* Consider seasonal indicators or Fourier terms in Prophet for complex seasonality.
* Experiment with multivariate LSTMs vs. univariate ARIMA/SARIMAX as baselines.

What does your database schema look like, and which column are you planning to forecast? Let’s drill into your actual table layout next.

Tab 2

**which feture we have to drop in our data that are not useful to build model**

# **Pruning Useless Features: Which Columns to Drop Before Modeling**

Identifying which columns to remove is as important as choosing the right algorithm. Dropping irrelevant or harmful features speeds up training, reduces overfitting, and often boosts accuracy. Below is a step-by-step framework, with practical checks and code snippets you can adapt to Prophet, LSTM, or any machine-learning pipeline.

## **1. Leverage Domain Knowledge and Initial Data Scan**

* ID Columns  
  + Primary keys, transaction IDs, UUIDs contain no predictive signal. Drop them or use as join keys only.
* Constant or Near-Constant Features  
  + Columns where > 99% of values are identical offer zero discriminative power.
* High Missingness  
  + If > 70–80% of a column is null, impute only if it’s critical; otherwise drop.
* Data Leakage  
  + Future timestamps (e.g., “next\_month\_sales”), target-derived flags, or any column that wouldn’t be known at prediction time must be excluded.

## **2. Statistical & Automated Filters**

1. Low-Variance Filter  
   * Use sklearn’s VarianceThreshold to drop features whose variance falls below a small threshold.
2. Correlation Threshold  
   * Compute pairwise Pearson (for continuous) or Spearman (for monotonic) correlations.
   * If two features correlate > 0.9, keep one and drop the other.
3. High Cardinality  
   * Categorical features with thousands of unique levels can bloat models.
   * Consider target encoding or dropping levels that occur < 1% of the time.
4. Multicollinearity (VIF)

Calculate Variance Inflation Factor (VIF) for each feature:  
 from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

X = df[features]

vif = pd.Series(

[variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])],

index=features

)

high\_vif = vif[vif > 10].index.tolist()

* + Drop features with VIF > 10 iteratively.

1. Feature Importance from Tree-Based Models  
   * Fit a RandomForestRegressor or XGBoost on your training set.

Use SelectFromModel to discard features with importance below a threshold:  
 from sklearn.feature\_selection import SelectFromModel

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n\_estimators=100).fit(X\_train, y\_train)

selector = SelectFromModel(rf, threshold='median', prefit=True)

X\_reduced = selector.transform(X\_train)

1. Univariate Statistical Tests  
   * Continuous vs. Continuous: Pearson’s correlation p-value
   * Continuous vs. Categorical: ANOVA (f-test)
   * Categorical vs. Categorical: Chi-square test
   * Use SelectKBest or SelectPercentile to keep only the most significant features.

## **3. Practical Thresholds & Example Table**

| **Criterion** | **Common Threshold** | **Action** |
| --- | --- | --- |
| Missing proportion | > 0.8 | Drop column |
| Unique value ratio | > 0.99 (constant) | Drop column |
| Pairwise correlation (abs) | > 0.9 | Keep one of each correlated pair |
| VIF | > 10 | Drop high-VIF features iteratively |
| Feature importance | Below median | Drop low-importance features |

## **4. Putting It All Together: Sample Pipeline**

import pandas as pd

from sklearn.feature\_selection import VarianceThreshold, SelectFromModel, SelectKBest, f\_regression

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from sklearn.ensemble import RandomForestRegressor

# 1. Load and domain-filter

df = pd.read\_csv('data.csv')

df.drop(['id', 'uuid', 'future\_flag'], axis=1, inplace=True)

# 2. Drop high-missing and near-constant

miss\_pct = df.isnull().mean()

df.drop(miss\_pct[miss\_pct > 0.8].index, axis=1, inplace=True)

sel\_var = VarianceThreshold(threshold=1e-4)

\_ = sel\_var.fit\_transform(df.select\_dtypes('number'))

const\_cols = df.select\_dtypes('number').columns[~sel\_var.get\_support()]

df.drop(const\_cols, axis=1, inplace=True)

# 3. Correlation filter

corr = df.corr().abs()

upper = corr.where(~np.tril(np.ones(corr.shape)).astype(bool))

to\_drop = [col for col in upper.columns if any(upper[col] > 0.9)]

df.drop(to\_drop, axis=1, inplace=True)

# 4. VIF

X = df.drop('target', axis=1)

vif = pd.Series([variance\_inflation\_factor(X.values, i)

for i in range(X.shape[1])],

index=X.columns)

for col in vif[vif > 10].index:

df.drop(col, axis=1, inplace=True)

# 5. Tree-based importance

y = df['target']

X = df.drop('target', axis=1)

rf = RandomForestRegressor().fit(X, y)

sel = SelectFromModel(rf, threshold='median', prefit=True)

X\_final = sel.transform(X)

## **5. Next Steps**

* After pruning, always retrain and validate your Prophet or LSTM model on the reduced set.
* Monitor if dropping any feature degrades performance—sometimes weak features still carry complementary signal.
* Consider dimensionality-reduction (PCA) if you still have hundreds of correlated inputs.

What does your raw schema look like, and which columns are you uncertain about? Share a snippet and we can recommend exactly which to keep or drop.

Tab 3

# **Pruning Columns for an LSTM Pipeline**

Before you build your LSTM, you want to remove or transform any columns that either leak future information, are redundant, or add noise. Here’s a pragmatic recipe:

## **1. Which Columns to Drop (and Why)**

* **Redundant date parts** You’ll reindex by week\_start\_date, so drop year and weekofyear.
* **High‐missing NDVI** ndvi\_ne has ~20% missing; either impute it carefully or drop if you don’t want heavy imputation.
* **Identifiers & Pure Categories** city can be one-hot encoded if you want city‐specific patterns; otherwise drop it.
* **Highly correlated sensors** Many “reanalysis” and “station” temps track the same signal. We’ll remove one from each pair with correlation > 0.95.

## **2. Code Example**

import pandas as pd

import numpy as np

# 1. Load data

df = pd.read\_csv('dengue\_features.csv', parse\_dates=['week\_start\_date'])

# 2. Drop redundant date columns

df.drop(['year', 'weekofyear'], axis=1, inplace=True)

# 3. Handle high-missing features (>20% missing)

miss\_frac = df.isnull().mean()

high\_miss = miss\_frac[miss\_frac > 0.20].index.tolist()

df.drop(high\_miss, axis=1, inplace=True)

# 4. Optional: encode city

# df = pd.get\_dummies(df, columns=['city'], drop\_first=True)

df.drop('city', axis=1, inplace=True)

# 5. Impute remaining gaps (simple forward-fill here)

df.sort\_values('week\_start\_date', inplace=True)

df.fillna(method='ffill', inplace=True)

# 6. Remove highly correlated columns

corr = df.drop('total\_cases', axis=1).corr().abs()

upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))

to\_drop = [col for col in upper.columns if any(upper[col] > 0.95)]

df.drop(to\_drop, axis=1, inplace=True)

print("Dropped features:", high\_miss + to\_drop + ['year','weekofyear','city'])

print("Remaining features:", df.columns.tolist())

## **3. Next Steps**

* **Scaling**: Apply MinMaxScaler or StandardScaler across your remaining numeric features.
* **Windowing**: Convert df into supervised sequences (X, y) for your LSTM.
* **Validation**: Use a rolling‐forecast CV to ensure you didn’t drop a weak but valuable predictor.

Ready to build your LSTM on this cleaned set?