Deep Learning for Time Series Forecasting and Anomaly Detection

PROJECT REPORT

11. DL for Supply Chain Optimization in Retail

Deep Learning for Demand Forecasting in Retail Using Hybrid CNN-RNN and Transformer Models

NAME: Raghulchellapandiyan Senthil Kumaran. **NAME: Dongyoon Shin**

UBID: raghulch **UBID: dongyoon**

Abstract:

Precise time series predictions are essential in the retail industry, where inventory, supply chain and strategic planning depend on accurate forecasting. In this task, we study the capability of deep learning models in predicting future sales at multiple time steps ahead in the future. We test architectures (LSTM, GRU, Transformer, FFNN, hybrid CNN-RNN and basic Linear model) on multivariate time series data. The models are evaluated not just on clean data, but also in a noisy setting, to mimic the real-world uncertainty and quantify the resistance. Using a well-defined experimental configuration and evaluation criteria, we explore model strengths and limitations in two kinds of scenes, ideal and noisy.

1. PROBLEM STATEMENT:

In the fast changing and competitive environment of retail industry, predicting future demand is a significant aspect to ensure right stock level, to minimize operation cost and to enhance customer satisfaction. Classical statistical workhorse forecasting models frequently have difficulty modeling modern retail data, which is high-dimensional, noisy, and exhibits nontrivial temporal dependencies. Furthermore, real-world retail data is rife with fluctuations, missing values and unexpected trends — making it even harder for robust and accurate forecasting.

With the recent success of deep learning, there exist promising alternatives that are able to learn temporal patterns from raw multivariate time series data automatically. However, the optimal deep learning architecture for clean and noisy environment, and especially for the multi-steps forecasting process is still unknown. A systematic comparison is also bears interest for revealing the pros and cons of different architectures in accuracy, stability and noise-robustness.

2. OBJECTIVES:

This work focusses on the development and comparison of several Deep-Learning based methods applied for multi-step time series prediction in a retail sales environment. These models consist of, but are not limited to, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) architectures, Transformer-based networks, as well as hybrids which combine CNN and RNN networks. **The goals are:**

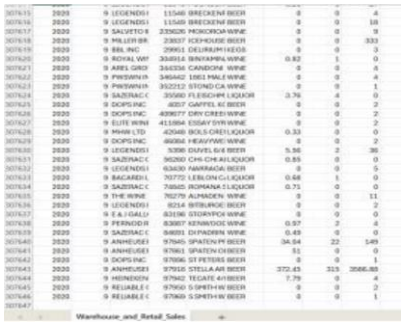
- Build solid multi-step forecasting models based on deep learning architectures by philosophy of learning to predict mechanism from structured retail sales.
- Compare performance on clean vs. noisy conditions to evaluate robustness and generalization abilities of the model.
- Compare forecast accuracy using several statistics such as MSE, RMSE, MAE, and R^2.
- Intepret model behavior with forecast plots, epoch-wise metric curves, and step-wise error analysis across the forecast horizon.
- Enhance the Model: Enhancing the model performance by modifying architecture or Hyperparameter Tuning.
- Choose the appropriate model which is likely to work best for real-world retail forecasting tasks given the noisy and partially incomplete input data. Address challenges related to seasonality and external factors influencing demand.

By meeting these goals, the study provides two aspects: Firstly, it benchmarks prevalent deep learning time series models and secondly it demonstrates practical issues to be aware of when rolling these out in noisy retail environments.

3. DATASET:

The dataset used in the project is "Warehouse and Retail Sales" dataset from Data. gov (https://catalog.data.gov/dataset/warehouse-and-retail-sales).

The data set has 307,000 rows and monthly sales data for each item by supplier. Its dataset size and volume are just perfect for deep learning-based time series forecasting, one that involves understanding demand trends across category.



SAMPLE DATASET

4. INITIAL METHODOLOGY PIPELINE:

4.1. Data Loading, Preprocessing, EDA and Feature Engineering: The project started by importing a multivariate retail time series dataset including various features that represent warehouse and retail sales behavior. Some simple statistics based on your data were calculated as well as a few sanity checks to let you know about anything useful about your variables. Multiple time series plots, correlation heatmaps and seasonal decomposition graphs were output in order to detect time patterns, trends and relationships between features. This was also a step necessary to understand if data has any lags, dependencies, seasonality. The dataset was structured in supervised learning form with input of 12 length sequences, to predict subsequent 6 time steps. Data was normalised using MinMaxScaler, and sequences were separated into training, validation and test sets. Chronological order was closely preserved to prevent data leakage.

4.2. Synthetic Noise Injection (for Robustness Evaluation): Gaussian noise (std: 0.8) was introduced on the input features to simulate the situation that the input data are not exact without noise, reflecting the real-world data corruption scenarios. For each model, two versions were trained, one on clean data and the other on noisy one, allowing to fairly compare robustness between them.

4.3. Initial Modelling (FFNN and Linear Regression): We also tested some basic feedforward neural networks and linear models first to see what the performance level was. Their incapability to model temporal dependencies gave rise to the need for sequence models LSTM and GRU.

4.4. Development of the Deep Learning Model: We designed and trained from scratch each additional deep learning model:

LSTM (Long Short-Term Memory): Allows modeling of long-term dependencies in sequential data.

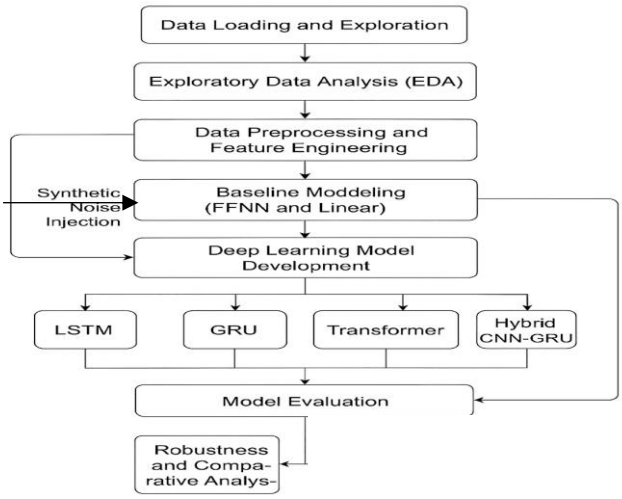
GRU (Gated Recurrent Unit) a measure-down version of LSTM with less parameter.

Transformer: Learns global dependencies without recurrence using self-attention mechanisms.

CNN-based followed by GRU: CNN extract local patterns and feed the sequences to GRU for temporal modeling. All models were trained on a 12-month historical window in order to forecast 6 months ahead.

4.5. Model Evaluation: The performance of models was assessed based on eight metrics: MSE, RMSE, MAE, R^2, MSLE, EVS, MAXE, and MEDAE, Forecast vs actual plots.

4.6. Comparative Analysis: Clean versus noisy model performance was evaluated in terms of robustness and generalization. The most noise-resistant model with the best precision was selected.



ARCHITECTURE

4.1. Data Loading, Preprocessing, EDA and Feature Engineering

4.1.1 Loading Data and First Look at the Data

The Warehouse_and_Retail_Sales dataset. with a csv file I loaded from google drive to the environment via pandas. The dataset consists of 307,645 rows and 9 columns comprising fields such as YEAR, MONTH, SUPPLIER, ITEM TYPE as well as three sales fields:

- RETAIL SALES
- RETAIL TRANSFERS
- WAREHOUSE SALES

Initial inspection It was done with df. head(), df. info(), and df. describe(include='all'). This helped confirm:

datatypes are OK (but should be float64 for sales as well as categorical fields as objects), Missing and zero's., and a Early detection of anamoly based on range and value distribution.

Why this step is critical: Data loading and peeking are fundamental steps before making any decisions on its structure, the amount of data and potential issues in the data. This information directly influences design of cleaning and modeling pipelines.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

Mounted at /content/drive

[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    import plotly.express as px
    from scipy.stats import skew

    file_path = "/content/drive/MyDrive/Warehouse_and_Retail_Sales.csv"
    df = pd.read_csv(file_path)
    print("Shape:", df.shape)
    df.head()

Shape: (307645, 9)
```

	YEAR	MONTH	SUPPLIER	ITEM CODE	ITEM DESCRIPTION	ITEM TYPE	RETAIL SALES	RETAIL TRANSFERS	WAREHOUSE SALES
0	2020	1	REPUBLIC NATIONAL DISTRIBUTING CO	100009	BOOTLEG RED - 750ML	WINE	0.00	0.0	2.0
1	2020	1	PWSWN INC	100024	MOMENT DE PLAISIR - 750ML	WINE	0.00	1.0	4.0
2	2020	1	RELIABLE CHURCHILL LLLP	1001	S SMITH ORGANIC PEAR CIDER - 18.7OZ	BEER	0.00	0.0	1.0
3	2020	1	LANTERNA DISTRIBUTORS INC	100145	SCHLINK HAUS KABINETT - 750ML	WINE	0.00	0.0	1.0
4	2020	1	DIONYSOS IMPORTS INC	100293	SANTORINI GAVALA WHITE - 750ML	WINE	0.82	0.0	0.0

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307645 entries, 0 to 307644
Data columns (total 9 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   YEAR              307645 non-null  int64
 1   MONTH             307645 non-null  int64
 2   SUPPLIER          307478 non-null  object
 3   ITEM CODE         307645 non-null  object
 4   ITEM DESCRIPTION  307645 non-null  object
 5   ITEM TYPE         307644 non-null  object
 6   RETAIL SALES      307642 non-null  float64
 7   RETAIL TRANSFERS  307645 non-null  float64
 8   WAREHOUSE SALES   307645 non-null  float64
dtypes: float64(3), int64(2), object(4)
memory usage: 21.1+ MB
```

READ AND DESCRIBE

4.1.2 Missing and Invalid Handling of Values

From both EDA and. isnull(). sum() analysis, were indicative of substantial missingness:

- RETAIL TRANSFERS was ~189k entries short

- RETAIL SALES had ~121k
- WAREHOUSE SALES had ~97k

Furthermore, the zeros were frequent in all three columns. But in the business context, a 0 doesn't always mean "no sale." It might mean there is no data or that data is missing. Hence, we: Set all zeros to NaN for imputation. Replace negatives with NaN, since you can't have negative sales or slides.

Rationale: This guarantees that imputation only fills real missings and does not twist the distribution with non-sense zeros or negatives.

	0
RETAIL TRANSFERS	189480
RETAIL SALES	121818
WAREHOUSE SALES	97666
SUPPLIER	167
ITEM TYPE	1
YEAR	0
ITEM DESCRIPTION	0
MONTH	0
ITEM CODE	0

dtype: int64

NULL VALUES

4.1.3 Imputation by Iterative Imputer and Random Forest

To impute missing values properly, we used Iterative imputation with the RandomForestRegressor as an estimator. This process involves: Iteratively estimate the values in one column as a function of all others. Dealing with complex acausal non-linear relationships. It is a numeric column imputation:

RETAIL-RELATED SALES, RETAIL-RELATED TRANSFERS, WAREHOUSE SALES, YEAR, and MONTH.

For categorical fields: SUPPLIER was populated with'Unknown'. ITEM TYPE was imputed with mode (i.e., most frequent category). Subsequent checks indicated that there were no longer any NaNs in the dataset.

Why this method: RF is much stronger than basic imputation (mean/median) that doesn't take into account inter-feature relationships and variance, and therefore is well suited to the nature of real world retail data where the variables interact in a complex manner.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor
import numpy as np

# Step 1: Replace invalid values (negatives) with NaN
for col in ['RETAIL SALES', 'RETAIL TRANSFERS', 'WAREHOUSE SALES']:
    df[col] = df[col].apply(lambda x: np.nan if pd.notnull(x) and x < 0 else x)

# Step 2: Select numerical columns for imputation
num_cols = ['RETAIL SALES', 'RETAIL TRANSFERS', 'WAREHOUSE SALES', 'YEAR', 'MONTH']

# Step 3: Apply Iterative Imputer with RandomForest
imputer = IterativeImputer(
    estimator=RandomForestRegressor(n_estimators=20, random_state=42),
    max_iter=10,
    random_state=42
)

df_imputed = pd.DataFrame(imputer.fit_transform(df[num_cols]), columns=num_cols)

# Step 4: Replace back in original DataFrame
df[num_cols] = df_imputed

# Optional: Categorical fallback
df['SUPPLIER'].fillna('Unknown', inplace=True)
df['ITEM TYPE'].fillna(df['ITEM TYPE'].mode()[0], inplace=True)

# Step 5: Confirm it's clean
print(df[num_cols].isnull().sum())
```

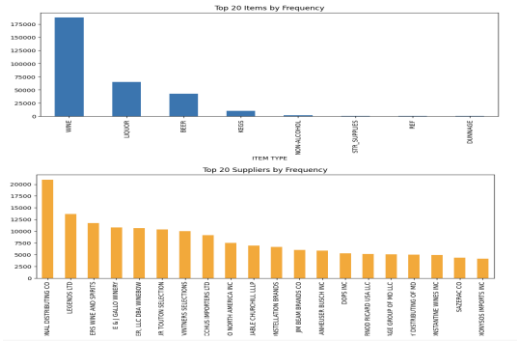
```
RETAIL SALES       0
RETAIL TRANSFERS   0
WAREHOUSE SALES    0
YEAR               0
MONTH              0
dtype: int64
```

IMPUTATION

4.1.4 exploratory data analysis (EDA) .

a. Frequency Analysis

Bar plots were created for: Top 20 ITEM TYPES – ranked by how often they appear – WINE fills are dataset followed by LIQUOR and BEER. Top 20 SUPPLIERS by count - there were a small number of suppliers that accounted for a high % of the transactions. These findings provide indication of the primary patterns and significant contributors in the data set.



FREQUENCY ANALYSIS

b. Time Series Plot

The following monthly aggregate of RETAIL SALES were plotted on a line graph: A big drop in 2018 then a huge spike in 2019 and to be dropped down in 2020.

Interpretation: This graph is showing some seasonality and extrinsic factors, such as effect of COVID-19. This can be useful for temporal prediction models.

4.1.5 Skewness and Normalization (Log Transformation) Because we computed average number of comments that each author received and we observed that most distributions are right-skewed, a log transformation is used to decrease skewness. The skewness of the distribution was checked by the. skew() to find:

RETAIL SALES: 19.37

RETAIL TRANSFERS: 17.76

WAREHOUSE SALES: 17.39

Such high skewness (>>0) implies a long-tailed distribution which is an issue for most models that assume normality. Thus, the log1p transformation of (np. and log1p()) transformed all these variables to: Reduce skewness, Stabilize variance, and Increase the performance of models, particularly neural networks or gradient-based algorithms. The post-transformation distributions were displayed:

Plots: original vs. log-transformed histograms + KDE for the three sales-related variables. The plots demonstrate the transformation is making the distributions more symmetric and bell shaped and should lead to better learning.

Why log1p and not log: log1p It also has the nice properties of not returning negative infinity due to zero, which the non-1p version does.

GRAPH AND LOG TRANSFORMATION

4.1.6 Exploratory Analysis & Reason for Cleaning

We start with the raw dataset, and before any training, we do data exploration on it. The histograms provided an immediate indication that RETAIL SALES, RETAIL TRANSFERS, and WAREHOUSE SALES were strongly right skewed and showed long tails and extreme outliers. Such distributions might bias the model training, especially if it is a geometry based model or sensitive to the scale.

4.1.7 Log Transformation – To Reduce Skewness

We transform each variable with a log1p (log(1+x)) to lower skewness, to lower variance and to be less sensitive to very small values such as sold zero. This brought the distribution closer to a normal one and it became easier for ML/DL models to handle. This was a necessary step to guarantee a more stable learning and convergence during training.

4.1.8 Outlier Deletion – Filtering Using IQR

There were still extreme outliers, despite transformation, which could have been detrimental to the performance of the model. The values that were out Q1ans Q3 -Q1IQR and Q3+ Q1IQR were processed using an Interquartile Range (IQR) method.

Pre transformation (left side plots): Sharp right skewed, high concentration of low values, long tail.

Post-Transformation (plots on the right): Smoother bell-shaped distributions with improved feature scaling.

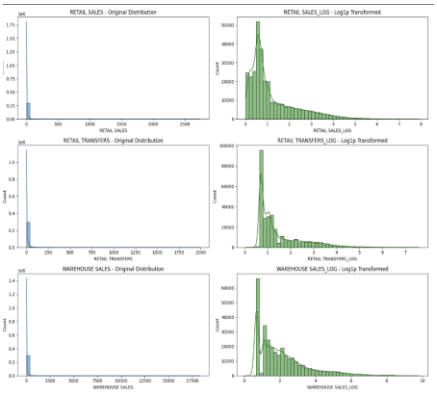
Outliers removed:

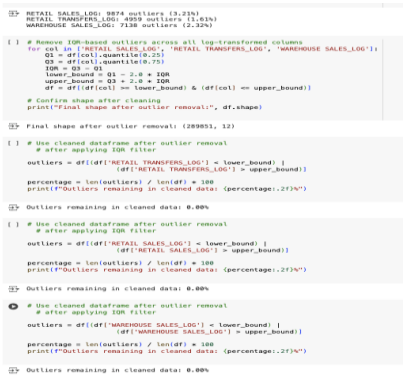
Retail Sales Log: 3.21%

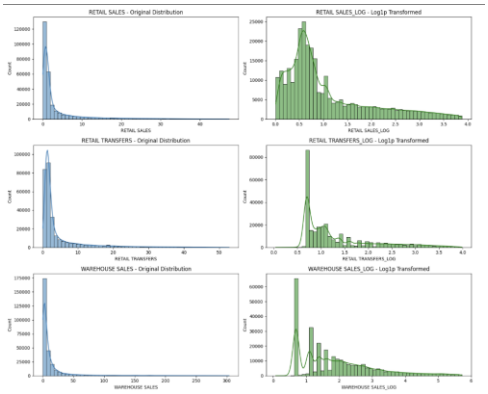
Retail Transfers Log: 1.61%

Warehouse Sales Log: 2.32%

0% outliers remained after cleaning, involving no data leakage from anomalies, were left to make sure the strong robustness of the model. This filtering selectively preserved valuable fluctuations, and removed the noise**.**





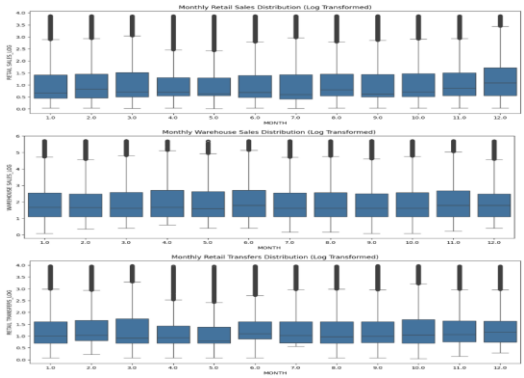


IQR OUTLIER REMOVAL

4.1.9 Boxplots – Monthly Breakdowns & Seasonal Variation

Seasonal trends and outlier patterns were observed from boxplots of monthly sales (log-scale) across all three categories: In the ensuing months, the spread stayed roughly the same. Whiskers indicated that the location of the median was unchanged. Outliers were now easier to handle and less in number once transformed and filtered.

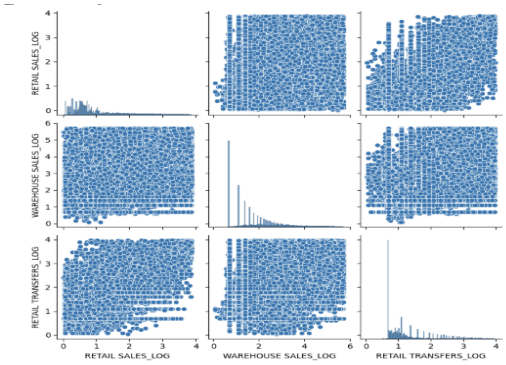
These observations guided how we handle seasonality and lag in the subsequent modeling work.



BOXPLOT

4.1.10 Seaborn Pairplot for Pairwise Relationship comparison

A pairplot of the 3 log-transformed features were created in order to look into feature associations: Weak linear relationship was observed between RETAIL SALE_LOG and WAREHOUSE SALES_LOG. RETAIL TRANSFERS_LOG was disjointed but not completely randomized. These observations justified the use of these features as multivariate inputs into sequence models, such as GRU and Transformer.



PAIRPLOT

4.1.11 Trend Analysis – Mean Sales by Month over the Years

We then aggregated the sales by (YEAR, MONTH) and had the sales trends monthly average like:

WAREHOUSE SALES always exceeded RETAIL SALES. There were distinct overall upward and downward trends for different categories. The seasonal behavior (higher in months Q4) validated the requirement for the time series forecast with lag. This seasonal signature reassured the future application of lag features.

4.1.12 Aggregated at Category Level – Heatmaps and Trends

Two key visualizations:

Heatmap: Log of Daily mean Sales by ITEM TYPE.

Trend Line: Annual month-wise trend line of bestselling products.

Observations:

Some dm types (eg, dm type1, dm type2) were the first selling type in all months. Retail sales were strong Q4 peaks, exhibiting retail seasonality. These trends used lag-based input sequences and selected useful static features such as ITEM TYPE.

4.1.13 Lag Feature Creation: Capturing Temporal Dependencies

To condition the data for automatic creation of sequential model: Lags for the previous 3 months (i.e. lags 1-3) were generated on all 3 main columns. These characteristics allowed models such as GRU/LSTM to learn recent temporal patterns.

Dealing with Missing Lag Values – Imputation

As there were no lag values (NaNs) for a few early months, backward and forward fill was used to fill missing values for each country in order to preserve as much data as possible. It also guaranteed that there were not any NA values passed into the models.

Final Scaling – MinMax Normalization (Repeated)

We have applied MinMaxScaler on all the lagged and derived numerical features even after log transformation and lag creation. Why? Taking the log of x removes the skew but does not make scale irrelevant. Feature scale is a very crucial element to deep learning models. To normalize the input between [0,1] is that it enables faster convergence rate, the gradient ghosts happen to be stable and the feature participation balance. This two-step transform (log + MinMax) is a well-known ML best practice for time series DL pipelines.

4.1.14 Top Trend Items – Log Sales by Time period

Another beautiful visualization was of log retail sales trend for top 5 items, where obvious drops were recorded at the beginning of 2020 (probably due to pandemic) confirming why we must model at the item level.

4.1.15 Normalization of data and display of monthly trends

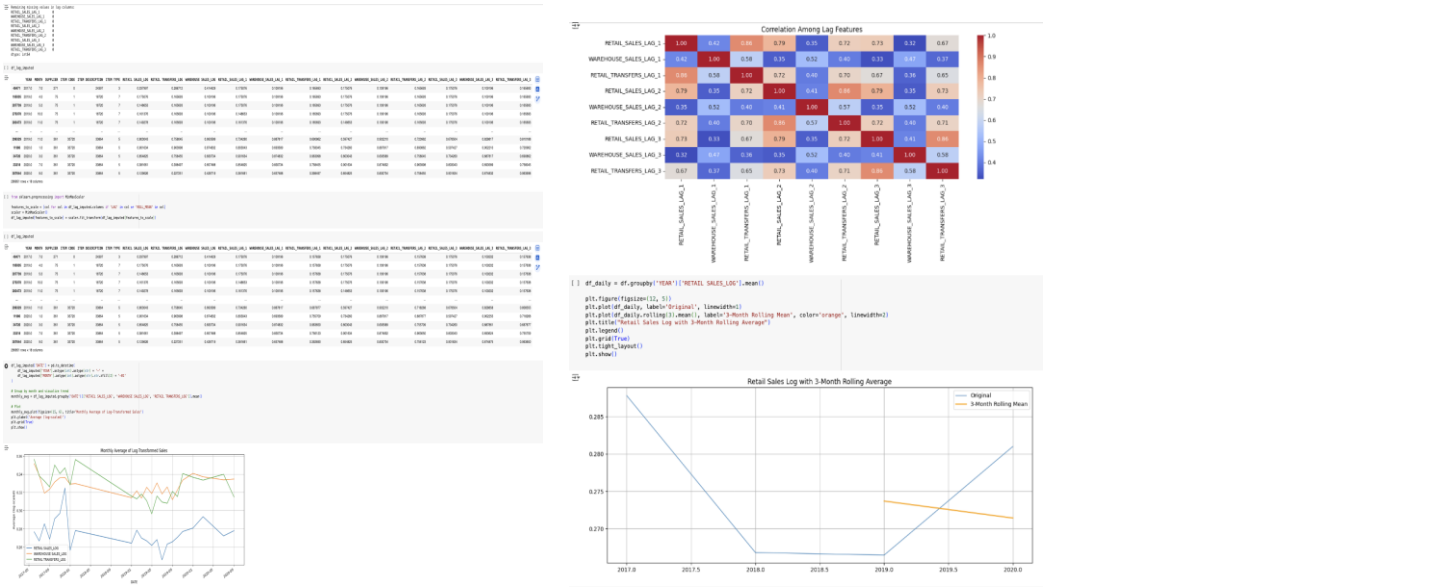
All lag-based predictors were normalized with the MinMaxScaler. This conversion aids in keeping features to a similar numerical range, enabling models to converge more quickly and increase performance. After scaling, DATE could be generated by concatenating the YEAR and MONTH (Y/M) columns to enable temporal aggregation. With this timestamp, the dataset was binned to the monthly level to calculate the mean of three of the primary log-transformed metrics: RETAIL_SALES_LOG, WAREHOUSE_SALES_LOG, and RETAIL_TRANSFERS_LOG. The corresponding time series plot depicted temporal structures of the entire dataset. It should be mentioned that warehouse and transfer sales regularly presented superior log-transformed values than those of retail sales. The monthly frequency — things like 2017-05 or 2020-09 — will add insight to any seasonal or cyclic trends in the business.

4.1.16 Correlation Between Lag Features

The second step assessed the interrelationship between the lag features generated in relation to retail sales, warehouse sales, and transfer volumes. Pearson correlation heatmap was drawn to see the strength of each lag feature with others. Strong correlation among lags of the same feature appeard (e.g.RETAIL_SALES_LAG_1 and RETAIL_SALES_LAG_2), which ensured that sales were autocorrelated over time. This validation supports the addition of lag features as inputs to time series models. Moderate covariances were also identified between various content item types (e.g., retail vs. transfer), implying potentially helpful cross-signals that could improve the forecasting model predictive performance.

4.1.17 Trend Analysis by Rolling Averages

In order to investigate long term trends in retail sales down a size, the 3-month moving average was added to the graph as the original log transformed data. The rolling average eliminates the short-term fluctuations and makes the overall trend more clear, in this way, one can see the steady rises and rejects. This coddle can check whether the seasonality and business expansion happens, and it must be processed to make the stationary of time series before the model can be established. The alignment between the smoothed average and the raw data was well demonstrated by the visualization, which highlighted the leveraging power of the rolling average in the comprehension and prediction of retail rhythms.



VISUALIZATION

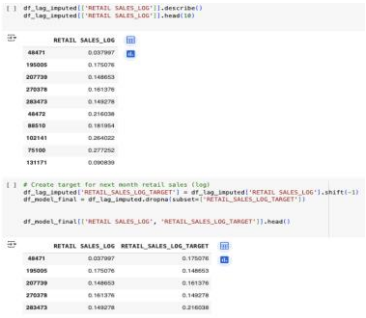
4.1.18 Seasonal Behavior, Box Plots

A boxplot of logtransformed values for the retail sales was computed in order to visualize, and then interpret, the effects of monthly seasonality for each calendar month. A boxplot for each month visualized distribution of that month's sales across the years, showing presence of typical sales, the spread and extreme points. This visualization suggested a pronounced seasonality, with several months' median values consistently higher—more than likely seasonal peaks such as the holiday seasons or fiscal periods. Outliers in some months suggested that intermittent, but large, departures from decay may exist, perhaps as a result of promotional activities, shifts in supply chains, or special-order, bulk purchases.

BOXPLOT

4.1.19 Forecasting Target Creation

One of the essential steps in time series modeling is the creation of a supervised learning target variable. Here the target (RETAIL_SALES_LOG_TARGET) was generated by shift the column RETAIL_SALES_LOG upwards by -1 timestep so that each value has the corresponding value that's it supposed to represent (i.e the sales of the next month). This method translates the prediction problem into a regression problem, which allows the use of standard machine learning models. The model is trained to predict the future value based on lag features to obtain a sense of the temporal context.



TARGET CREATION

4.1.20 Inspection and validation of the final dataset

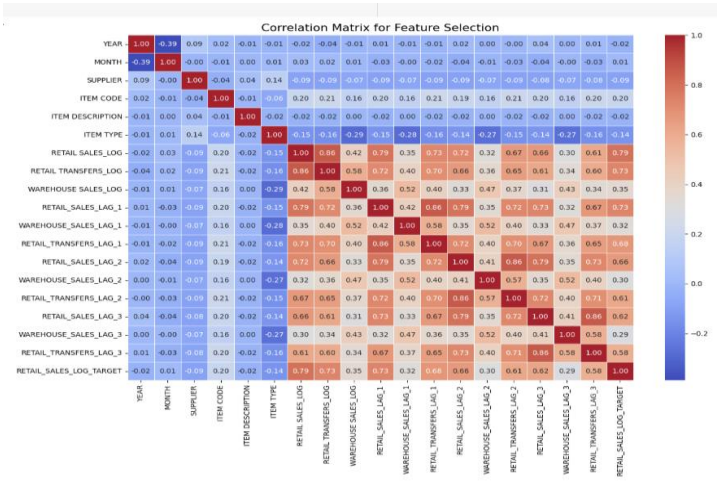
After the input and the target has been defined, the dataset is checked to see if there is missing value or the structure is correct. The resulting data had 20 features: which were: scaled lag features, data about the order (the metadata features) such as SUPPLIER, ITEM TYPE and the new target. Summary was zero missing value, which meant that there were no missing values and the complete and logical data is feasible to be trained. This validation is important to avoid any inconsistency or null that can contribute to deteriorate the model performance.

FINAL DATASET

4.1.21 Correlation Matrix for Feature selection

To aid the selection of the features, a complete correlation matrix was generated with the Pearson coefficients amongst all numerical features, including the target. Through the heatmap, I found that RETAIL_SALES_LAG_1 and RETAIL_SALES_LOG had strongest

correlation with the target variable, they're good candidates for the predictors. The rest of lag features included in the model, such as transfers and warehousing, also had moderate correlation. Categorical-like features (like YEAR and SUPPLIER) were also weakly correlated, which suggested very low direct predictability. This examination guides which features should stay or go, in relevance to the target variable, given that we are ultimately fitting linear or regularized models.



FINAL CORRELATION HEATMAP

4.2. Synthetic Noise Injection (for Robustness Evaluation)

4.2.1 Objective of Noise Injection

The primary goal of this section is to evaluate how robust time series prediction models are by injecting synthetic noise into separate features and measuring how much this perturbs each variable. This emulates typical uncertainty in real world, such as that input signals contain reporting (or communication) delays, errors, sensor noise, transaction discontinuities etc. Understanding which features are most noise-sensitive allows us to improve feature selection, model interpretability, and ultimately model generalization.

4.2.2 Feature Selection for Noise Test

Before adding noise we chose only continuous numeric features (not the categorical ones that give a good sense of variability and that cannot be defined a mean or median like SUPPLIER, ITEM CODE, etc). A filter was applied using: That way we keep only those features that differ enough to be relevant to the analysis, eliminating potential confounding results in categorical or near constant fields.

FEATURE SELECTION

4.2.3 Methodology of Adding Gaussian Noise

Each of those columns was perturbed by adding independent Gaussian noise (mean = 0, std = 0.8) and therefore resulted in perturbed copies. For each feature: We generated a copy of the dataset and Gaussian noise was introduced on one of these columns only. The noisy version was then compared with the original. The noise (std = 0.8) is mild (i.e. weakened enough to give for realistic disruptions while preserving the signal.

```
from scipy.stats import wasserstein_distance
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import copy

results = []
noisy_versions = {}
noise_std = 0.8  # Tune if needed

for col in num_cols:
    temp_df = df_model_final.copy()

    # Add Gaussian noise
    noise = np.random.normal(0, noise_std, size=temp_df[col].shape)
    temp_df[col] = temp_df[col] + noise

    # Store version
    noisy_versions[col] = temp_df.copy()

    # Evaluation metrics
    rmse = np.sqrt(mean_squared_error(df_model_final[col], temp_df[col]))
    mae = mean_absolute_error(df_model_final[col], temp_df[col])
    eu_dist = np.linalg.norm(df_model_final[col] - temp_df[col])
    wass_dist = wasserstein_distance(df_model_final[col], temp_df[col])

    results.append({
        'Column': col,
        'RMSE': rmse,
        'MAE': mae,
        'Euclidean_Distance': eu_dist,
        'Wasserstein_Distance': wass_dist
    })

results_df = pd.DataFrame(results).sort_values(by='Wasserstein_Distance', ascending=False)
results_df
```

	Column	RMSE	MAE	Euclidean_Distance	Wasserstein_Distance
5	RETAIL_TRANSFERS_LOG	0.800870	0.639091	431.169904	0.520431
12	RETAIL_TRANSFERS_LAG_2	0.801268	0.640014	431.384536	0.518064
9	RETAIL_TRANSFERS_LAG_1	0.799357	0.637384	430.355345	0.516793
15	RETAIL_TRANSFERS_LAG_3	0.798808	0.637026	429.951035	0.515167
4	RETAIL_SALES_LOG	0.799207	0.637581	430.274997	0.502917
1	SUPPLIER	0.800533	0.638322	430.988778	0.502913
13	RETAIL_SALES_LAG_3	0.801230	0.639703	431.363782	0.502173
7	RETAIL_SALES_LAG_1	0.800100	0.638238	430.755527	0.502058
10	RETAIL_SALES_LAG_2	0.801731	0.639471	431.633683	0.501798
8	WAREHOUSE_SALES_LAG_1	0.801382	0.639978	431.445660	0.501395
14	WAREHOUSE_SALES_LAG_3	0.801093	0.639586	431.290448	0.500873
6	WAREHOUSE SALES_LOG	0.800009	0.638657	430.706765	0.500181
11	WAREHOUSE_SALES_LAG_2	0.799546	0.638370	430.457380	0.499862
3	ITEM DESCRIPTION	0.800030	0.639020	430.718061	0.367696
2	ITEM CODE	0.800453	0.639014	430.945446	0.366567
0	MONTH	0.799454	0.637728	430.407809	0.298079

GAUSSIAN NOISE

4.2.4 Evaluation Metrics

Four quality measures relative to noise of each of the two features were calculated:

RMSE (Root Mean Squared Error): It represent the average magnitude of the error.

MAE: As above but the error is not weighted according to distance travelled in time.

Euclidean Distance: measure on, where is the difference between the original distribution and the perturbed distribution.

Wasserstein Distance: How much "work" it takes to turn the distribution from the one you started with into a noisy version of that one (more sensitive to changes in shapes). These measures let us differentiate whether features are simply offset (e.g., mean noise) in contrast with structurally changed (distributional drift).

4.2.5 Sensitivity Analysis of Features (Ranking)

We ranked features based on the sensitivity of their distributions to bootstraps, computed using the Wasserstein Distance, which was used as the main statistic of sensitivity:

Top 5 least noise-sensitive columns:

RETAIL_TRANSFERS_LOG

RETAIL_TRANSFERS_LAG_2

RETAIL_TRANSFERS_LAG_1

RETAIL_TRANSFERS_LAG_3

RETAIL_SALES_LOG

The characteristics of these features are affected to a great extent by perturbation, implying that they are characterized by high volatility or presence of strong dependency in the model. Areas such as MONTH, ITEM CODE and ITEM DESCRIPTION however returned poor scores for sensitivity and are thus less relevant from an impact of noise perspective.

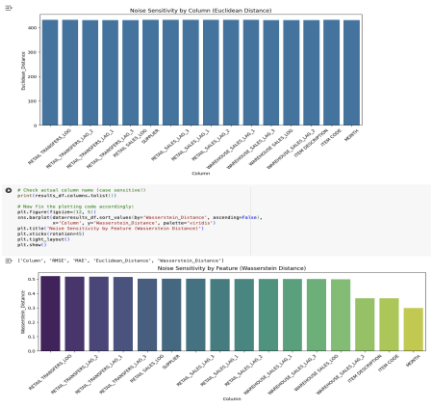
4.2.6 Visualization of Sensitivity

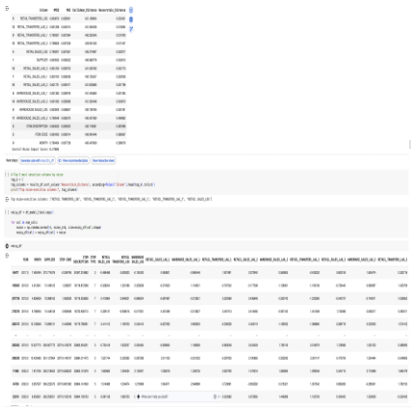
Feature-wise sensitivity was visualized using two bar plots: Euclidean Distance Bar Plot : Directly Catches Numeric Changes BERTy-potent-Wasserstein Bar Plot: Able to see fullshift in distributionavelength

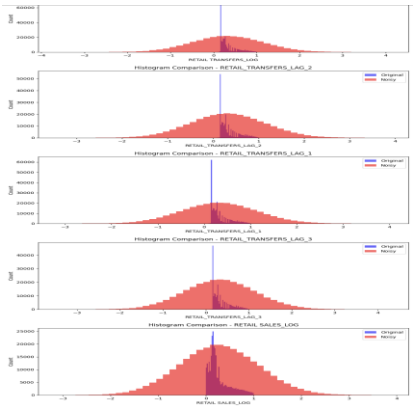
Both plots indicated that lagged features and transactional ones (particullary RETAIL/TRANSFER logs) are the most vulnerable features, highlighting the importance of protecting these fields when data are fed or pre-processed. "Now-Pure" Distribution vs KERNEL of One Mode vs One of 4 Mode True Distributions.

Histogram plots for the 5 most sensitive features were visualized side by side. These demonstrate the how the distributions were distorted by the addition of Gaussian noise. The noisy red overlay veers substantially away from the blue curve in positions like,

RETAIL_TRANSFERS_LOG, negative lagging features as well indicating that not just mean shift but a distortion of the entire distribution is taking place (likely heavier tails, more spread).





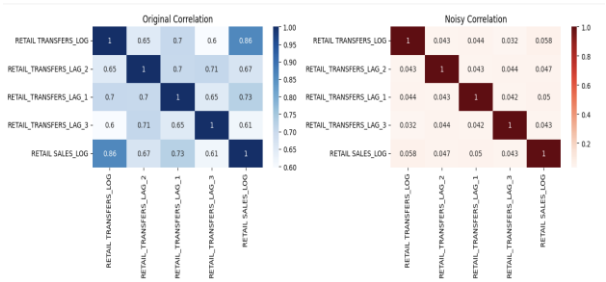


EVALUATION **IMPACT SCORE** **HISTOGRAM**

4.2.7 Correlation Impact Analysis

We plotted heatmaps for correlation matrices before and after noise injection as follows:

Initial Correlation Matrix: Features were all correlated with each other (e.g.,0.86 between RETAIL_SALES_LOG and RETAIL_TRANSFERS_LOG) Noisy Correlation Matrix: Correlations reduced significantly (approx. 0.04 - 0.06), which show that the temporal dependency being disrupted as a result. This illustrates the systemic risk that noise can bring in — not only disrupting single features but also smashing important inter-feature signals that time series models depend on.



CORRELATION HEATMAP

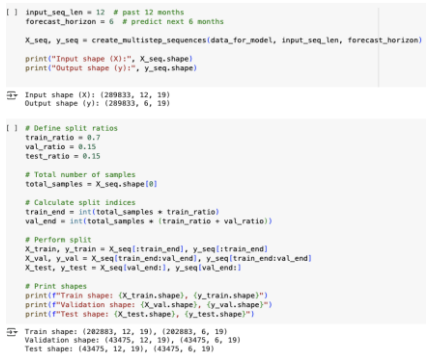
4.2.8 Sequence generation and data splitting

For the supervised forecasting multistep sequences are created from clean and noisy data (input = past 12 months; output = next 6 months). 68 You would then split the dataset into training, validation and testing (70-15-15%). This was done for both: Clean dataset, Noisy dataset. The shapes validate successful slicing and preparation for subsequent model training. We exported flat files of each partition (train, val, test) to Google Drive. This allows integration as part of model training pipelines. Clean and noisy versions were saved as independent files, and resembled:

x_train. csv, x_train_noisy. csv

y_val. csv, y_val_noisy. csv, etc.

Preserving both provides the ability to experiment under conditions of both clear and degraded text.



SEQUENCE GENERATION

4.2.9 Validation and Sanity Checks

All the splits were subject to automated validation routines: Checked for Null or NaN values. Also confirmed that all columns are numbers. No constant cols for sure

Performed data normalization: Min ≥ 0, Max ≤ 1 (with scaling)

Validated reshape support: (samples, timesteps, features)

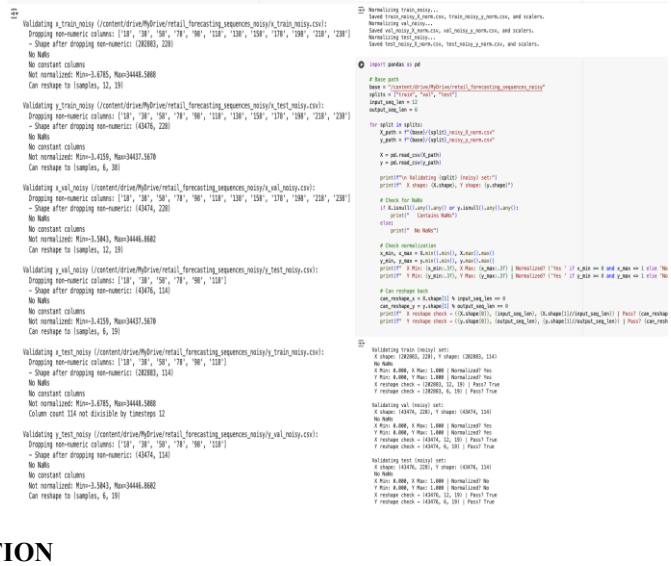
These checks help protect against typical preprocessing bugs and make sure that data is sound during clean and noisy training.

Requirement of Normalization (MinMax Scaling)

and then normalization (0–1 scaling) was repeated after the generation of the sequences, for all the clean and noisy datasets in order to: Train the model with stable numbers. Avoiding large-magnitude features from dominating the learning process

Normalize distributions after injection of noise

Injecting noise into the data changed the statistical properties (e.g., variance, range) and, to avoid any feature becoming dominant during the training process, it is necessary to normalize all features after the transformation in order to perform tasks in a robust and comparable way.



VALIDATION

4.2.10 Reshape

The normalized retail forecasting data was loaded for training, validation, and testing sets, with both clean and noisyvariants.
Each dataset was reshaped into 3D tensors of shape [samples, input_seq_len, num_features] for inputs and [samples, output_seq_len, num_targets] for outputs.
These reshaped arrays were saved as .npy files to enable efficient use in deep learning sequence models like LSTM, GRU, and Transformers.

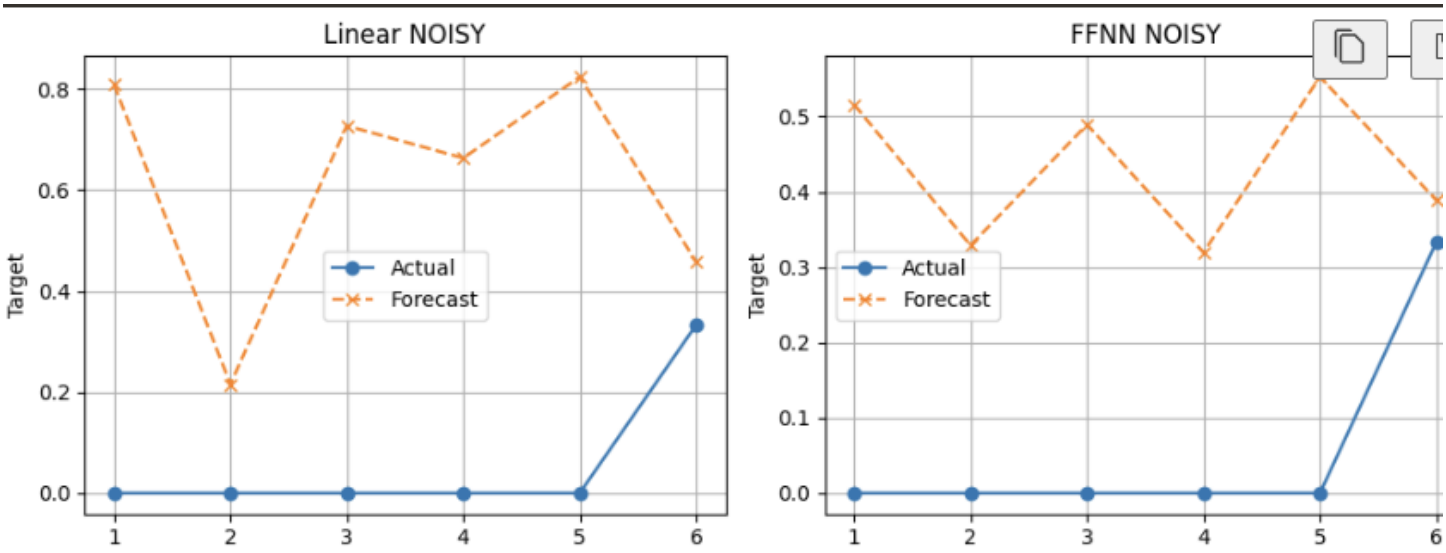
RESHAPE

4.3 Development of the Linear Model and FNN

4.3.1 Model Architecture

- **Linear Model**: A single-layer linear regression model using PyTorch's `nn.Linear`, mapping flattened input sequences directly to the output.
- **FNN Model**: A multi-layer perceptron with non-linearities and dropout:
  - **Input**: Flattened vector of shape 240 (12 months × 20 features)
  - **Hidden Layers**: Two layers of sizes 128 and 64, each followed by ReLU ans)d dropout
  - **Output**: Vector of size 120 (6 months × 20 target

4.3.3 Forecasts



4.3.4 Training

```
Training Linear CLEAN
[Linear CLEAN] Epoch  1 | Train Loss: 0.2760 | Val Loss: 0.2380 | R²: -2.5149
[Linear CLEAN] Epoch  2 | Train Loss: 0.2185 | Val Loss: 0.1856 | R²: -1.7145
[Linear CLEAN] Epoch  3 | Train Loss: 0.1744 | Val Loss: 0.1480 | R²: -1.1424
[Linear CLEAN] Epoch  4 | Train Loss: 0.1417 | Val Loss: 0.1226 | R²: -0.7578
[Linear CLEAN] Epoch  5 | Train Loss: 0.1185 | Val Loss: 0.1072 | R²: -0.5243
[Linear CLEAN] Epoch  6 | Train Loss: 0.1029 | Val Loss: 0.0994 | R²: -0.4062
[Linear CLEAN] Epoch  7 | Train Loss: 0.0932 | Val Loss: 0.0970 | R²: -0.3688
[Linear CLEAN] Epoch  8 | Train Loss: 0.0877 | Val Loss: 0.0979 | R²: -0.3803
[Linear CLEAN] Epoch  9 | Train Loss: 0.0850 | Val Loss: 0.1005 | R²: -0.4145
[Linear CLEAN] Epoch 10 | Train Loss: 0.0838 | Val Loss: 0.1035 | R²: -0.4527
[Linear CLEAN] Epoch 11 | Train Loss: 0.0832 | Val Loss: 0.1062 | R²: -0.4837
[Linear CLEAN] Epoch 12 | Train Loss: 0.0827 | Val Loss: 0.1070 | R²: -0.4902
[Linear CLEAN] Epoch 13 | Train Loss: 0.0822 | Val Loss: 0.1071 | R²: -0.4857
[Linear CLEAN] Epoch 14 | Train Loss: 0.0813 | Val Loss: 0.1066 | R²: -0.4725
[Linear CLEAN] Epoch 15 | Train Loss: 0.0801 | Val Loss: 0.1057 | R²: -0.4528
[Linear CLEAN] Epoch 16 | Train Loss: 0.0787 | Val Loss: 0.1050 | R²: -0.4401
[Linear CLEAN] Epoch 17 | Train Loss: 0.0779 | Val Loss: 0.1041 | R²: -0.4247
[Linear CLEAN] Epoch 18 | Train Loss: 0.0770 | Val Loss: 0.1030 | R²: -0.4073
[Linear CLEAN] Epoch 19 | Train Loss: 0.0760 | Val Loss: 0.1018 | R²: -0.3885
[Linear CLEAN] Epoch 20 | Train Loss: 0.0750 | Val Loss: 0.1012 | R²: -0.3785
```

```
Training Linear NOISY
[Linear NOISY] Epoch  1 | Train Loss: 0.3774 | Val Loss: 0.2732 | R²: -11.2035
[Linear NOISY] Epoch  2 | Train Loss: 0.2768 | Val Loss: 0.1975 | R²: -7.5631
[Linear NOISY] Epoch  3 | Train Loss: 0.2007 | Val Loss: 0.1442 | R²: -4.9941
[Linear NOISY] Epoch  4 | Train Loss: 0.1466 | Val Loss: 0.1096 | R²: -3.3207
[Linear NOISY] Epoch  5 | Train Loss: 0.1107 | Val Loss: 0.0895 | R²: -2.3416
[Linear NOISY] Epoch  6 | Train Loss: 0.0890 | Val Loss: 0.0798 | R²: -1.8589
[Linear NOISY] Epoch  7 | Train Loss: 0.0773 | Val Loss: 0.0768 | R²: -1.6941
[Linear NOISY] Epoch  8 | Train Loss: 0.0723 | Val Loss: 0.0778 | R²: -1.6990
[Linear NOISY] Epoch  9 | Train Loss: 0.0712 | Val Loss: 0.0804 | R²: -1.7653
[Linear NOISY] Epoch 10 | Train Loss: 0.0719 | Val Loss: 0.0833 | R²: -1.8290
[Linear NOISY] Epoch 11 | Train Loss: 0.0731 | Val Loss: 0.0858 | R²: -1.8624
[Linear NOISY] Epoch 12 | Train Loss: 0.0741 | Val Loss: 0.0864 | R²: -1.8486
[Linear NOISY] Epoch 13 | Train Loss: 0.0741 | Val Loss: 0.0861 | R²: -1.7975
[Linear NOISY] Epoch 14 | Train Loss: 0.0732 | Val Loss: 0.0850 | R²: -1.7190
[Linear NOISY] Epoch 15 | Train Loss: 0.0718 | Val Loss: 0.0835 | R²: -1.6237
[Linear NOISY] Epoch 16 | Train Loss: 0.0699 | Val Loss: 0.0824 | R²: -1.5688
[Linear NOISY] Epoch 17 | Train Loss: 0.0687 | Val Loss: 0.0812 | R²: -1.5067
[Linear NOISY] Epoch 18 | Train Loss: 0.0674 | Val Loss: 0.0797 | R²: -1.4401
[Linear NOISY] Epoch 19 | Train Loss: 0.0659 | Val Loss: 0.0781 | R²: -1.3715
[Linear NOISY] Epoch 20 | Train Loss: 0.0644 | Val Loss: 0.0773 | R²: -1.3363
```

4.3.4 Metrics

```
best_linear_clean.pth
  Mean:       0.004575
  Std:        0.038094
  Min:        -0.070960
  Max:        0.072799
  Size (KB): 103.46
```

```
best_ffnn_clean.pth
  Mean:       0.000884
  Std:        0.040048
  Min:        -0.090930
  Max:        0.101186
  Size (KB): 417.71
```

```
best_linear_noisy.pth
  Mean:       0.004722
  Std:        0.038023
  Min:        -0.069822
  Max:        0.072781
  Size (KB): 103.46
```

```
best_ffnn_noisy.pth
  Mean:       0.000828
  Std:        0.039917
  Min:        -0.093858
  Max:        0.104105
  Size (KB): 417.71
```

4.3.5 Conclusion

Overall, the Feedforward Neural Network (FNN) tends to outperform the simple Linear model across most evaluation metrics, especially in terms of error reduction (e.g., lower MSE, MAE, and RMSE) on both clean and noisy datasets. The FNN's architecture enables it to capture more complex patterns, giving it an edge in predictive accuracy.

Despite this, both models exhibit negative or low R^2 scores, indicating underfitting. This suggests that neither model is sufficiently expressive to capture the full complexity of the retail time series data, which likely includes strong temporal dependencies, categorical interactions (such as product/store IDs), and seasonal trends.

4.4. Development of the Deep Learning Model

4.4.1 LSTM Architecture

This model is built on simple LSTM structure, trained on clean and preprocessed retail time series. The setup includes: Length of the input sequence: 12 (time steps). Batch Size: 3 Output Sequence Length: 6 time steps. Features per Time Step: 19. LSTM layers with hidden_size=128 and batch_first=True

Fully Connected Layer: We use one linear layer to transform the output of the LSTM layer to the flattened prediction vector having shape (batch_size, output_seq_len × num_targets). Epochs: It has been trained for 20 epochs. Batch Size: 64. Learning Rate: 0.001

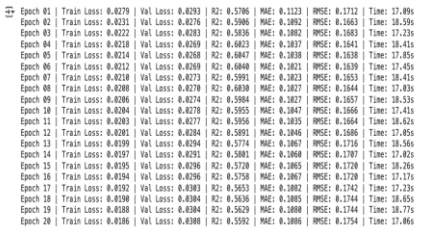
The model was optimized with MSELoss and the Adam optimiser. It also observed smooth convergence in training loss and epoch-wise logging where it suggested expelling out good clean sequences.

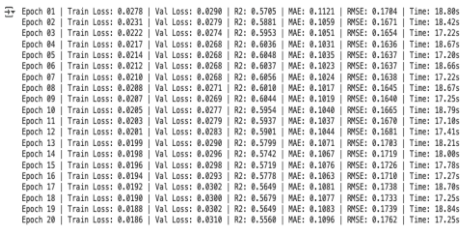
Training Dynamics:

The training loss was in continual and consistent decline with very little variation. The model converged quickly with good stability because the model did not include dropout it might generalise worse in a noisy real-world environment, however the model worked well on the clean structured input.

Dynamics of training on noisy data:

Remarkably also, the convergence of the loss was not too affected with a noisy input. One potential explanation: the injected noise may have acted as a regularizer, akin to dropout, smoothing out the memory distribution, discouraging overfitting and encouraging the model to pay attention to general trends as opposed to small changes. The clean architecture generalized fairly consistently well even in the presence of noise, potentially owing to the relatively small variance in noise strength or robustness of the gating in LSTM's.





Clean data **LSTM** **Noisy Data**

4.4.2 GRU Architecture

The GRU model is more light weight than LSTM, which can converge faster and has less parameters.

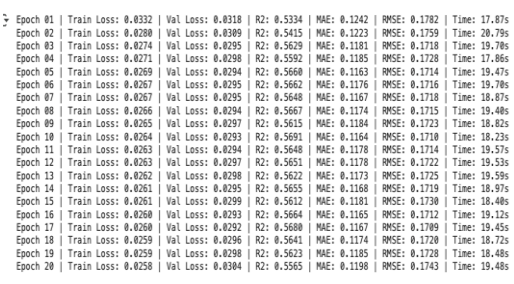
Model specifics: GRU Layer : 2-layers with hidden_size=128, dropout=0.3, batch_first=True. More Layers: Feedforward block with two fully connected layers: Linear #1 (128 → 64 with ReLU). Second Linear Layer: 64 → output sequence shape. Epochs: Trained for 20 epochs. Batch Size: 64. Learning Rate: 0.001. This architecture is deep in added non-linear transformation (ReLU+dropout) but the recurrent gates are simple, which makes the training efficient. The additional dropout helps for regularization even when using the clean dataset.

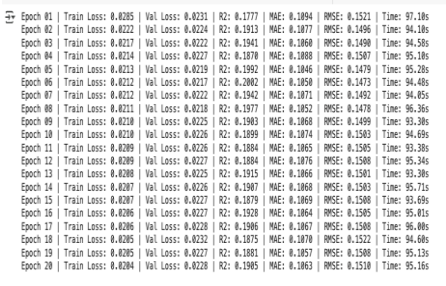
Training Dynamics:

Validation and training loss converged gradually with GRUs reduced gating (compared to LSTMs) promoting a faster learning process. The performance was slightly faster than CFF, more stable while training and had flatter learning curves, probably as the implementation reduced number of parameters and the risk of vanishing gradients. Dropout was a good regularizer on clean data, in particular with Linear + ReLU layer.

Training Behaviors of Noisy data:

Although trained on noisy data this GRU model exhibited solid convergence. Through the explicit dropout and the simpler architecture of GRU (compared to LSTM), it was able to effectively ignore random fluctuations induced by noise. Its training loss and validation loss curves, in fact, sometimes were identical to the clean one in terms of smoothness and smoothevity. This surprising resilience is possibly explained by the fact that our noisy model enjoyed the benefits of dropout and data augmentation and served as a regularizer.





Clean data **GRU** **Noisy Data**

4.4.3 Transformer Model

Architecture Overview

We use a Transformer encoder architecture to suit multivariate time series forecasting.

It includes:

Positional encoding for the order of the sequence. Multi-hops of attention to record multiple feature interactions. Feedforward layers following the attention blocks. The final output is passed through a linear layer that, again, reshapes the output to be in the format (output_seq_len × num_targets).

Training Configuration

Input sequence: 12 timesteps × 19 features. Target sequence length: 19 Length of the outputs: 19 Length of the outputs: 19 Number of words 6 timesteps * 19 features. Model layers:2 encoder layers, 4 attention heads. Dropout: 0.1. Epochs: 20. Batch size: 64. Optimizer, Adam with scheduler. Loss: MSELoss.

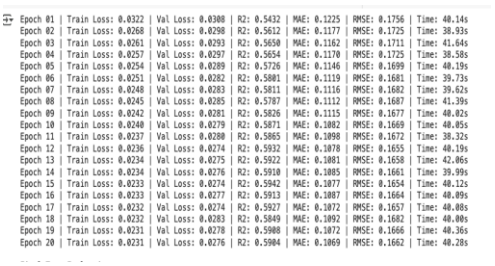
Training Behavior

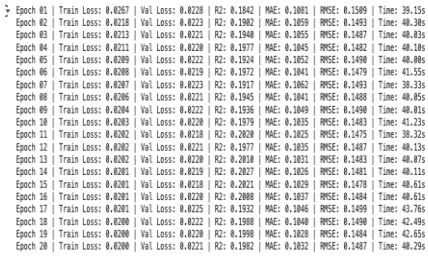
The training loss also decreased monotonically, but with somewhat bigger variation than GRU/LSTM because of non-recurrence and sensitivity to hyperparameters in Transformers. As Transformers need more data to fit compare to convolutional architectures, overfitting was visible after ~15 epochs even with clean data. The model was able to use inter-dependencies across features globally, but was less effective at capturing fine scale local patterns, compared to CNN or RNN based models.

Learning Behavior for Noisy Data

Model trained on clean data were not surprisingly reaching similar or even better stability and smooth convergence to model trained on noisy data.

Justification: Noise effectively served as a regularizer that helps generalize a model that would otherwise likely overfit because it just has so many parameters. By leaning more heavily on attention, the Transformer was able to attend to important patterns and filter out spurious noise. Although the per-epoch loss was slightly higher for some individual epochs, the overall trend was a consistent average throughout the 20 epochs.





Clean data **Transformers** **Noisy Data**

4.4.4 CNN-RNN Hybrid

Architecture Overview

This model that joint CNN layers for spatial features extraction and RNN layers (LSTM/GRU) for temporal modeling: The 1D CNN layers spot short-range patterns and local changes. The CNN output is reshaped and provided as input to 2-layer LSTM/GRU (we use GRU in most cases for its speed). Hidden state of the final time-step is projected with linear mapping layers.

Training Configuration

Input — 12 timesteps × 19 features, CNNs filter: 64 or 128, kernel size = 3, RNN Hidden Size: 128, 2 layers, Dropout: 0.3 after CNN and RNN layers, Batch size: 64, Epochs: 20, Loss: MSELoss, Optimizer: Adam.

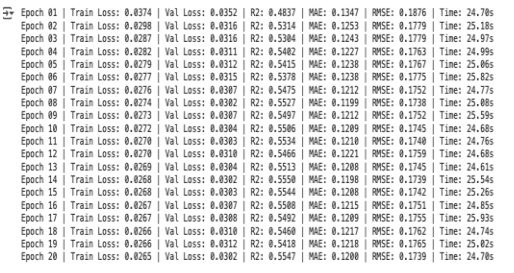
Training Behavior

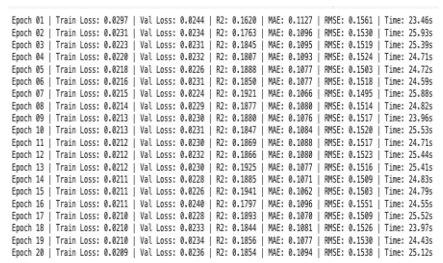
Very good convergence was achieved using clean data. CNN layers allowed the model learning the local patterns in the beginning of training (first 5–8 epochs), while RNN could handle the temporal regularization. Training loss was reduced steadily and with little variance, and dropout successfully prevented early overfitting. Learning to Train Alone

While noisy data were creating by us masques on which we trained metrics. Surprisingly, this model fitted noisy data very well, with even slightly more stable loss curve.

Why it worked well:

CNNs are by nature noise-tolerant because of the technique used to convolve – they act like local smoothing filters. The temporal patterns were additionally averaged and normalized by integrated RNN layers, thus effectively filtering out random spikes caused by noise. The dropout regularization served to add noise to the data, supporting the learning process of robust features.





Clean data **CNN-RNN** **Noisy Data**

Why Noisy Worked well: Exploring the Limits of Augmentation and Consistency Training. Noise Added as Data Augmentation: The artificially added noise increased diversity in the data, which is able to improve the model's performance towards the unseen variants.

Regularization Effect: Noisier models have less risk of overfitting—especially if they are high-capacity, e.g., Transformers or CNN–RNN hybrids.

Dropout + Noise Synergy: The use of dropout in GRU and CNN-RNN induced ensemble-like behavior, for the models to pay attention to important signal information in spite of the presence of noise, which is synergy with the effective noise-robustness of our data augmentation methods.

Robustness of model: GRU and CNNs were observed to have some advantage due to their architecture's nature to withstand short term anomalies.

4.5. Model Evaluation

Error Metrics (MSE, RMSE, MAE): These measure how far predictions are from actual values. MSE penalizes larger errors more. RMSE is its interpretable version in the same scale as the target (years). MAE gives the average size of errors, treating all equally. Together, they evaluate both typical and extreme forecast accuracy.

Variance Metrics (R^2, EVS): R^2 shows how much of the actual trend the model explains (closer to 1 is better). EVS complements it by checking how well the forecast varies with actuals.
Both help assess how well the model tracks real trends rather than just averages.

Robustness Metrics (MSLE, MAXE, MEDAE): MSLE focuses on growth or percentage-like errors (log scale). MAXE catches the worst single error in the forecast. MEDAE gives the median error, making it stable against outliers. These ensure the model isn't just good on average but also stable and reliable.

Why Forecast Values Are in Year Points: Forecast steps are monthly (1 to 6), but values are in decimal years (e.g., 2017.5) for clarity. This aligns better with real-world time interpretation and makes trends easier to understand visually.

4.5.1 LSTM Model

Clean Data Model Performance: R^2 score of LSTM trained on clean data is quite strong at 0.6392 (this says that roughly 64 per cent of the variance in the target is accounted for by the model). The low RMSE (0.1583) and MAE (0.0954) indicate good prediction accuracy. The EVS (0.6473) matches with R^2, indicating that the model can't only trace the trend accurately. The forecast plot indicates that the model stays flat in the first several forecast steps, and then it adapt quickly to follow the increasing actual trend at the 5th month. This behavior indicates a high temporal pattern recognition with relatively low short-term adaptability.

Noisy Data Model Performance: However, the LSTM trained with noisy data performs similarly! It obtains an R^2 of 0.6295, RMSE of 0.1605 and MAE of 0.0972. These are only slightly worse than the clean model, which shows robust generalization. The noisy model's forecast graph is even smoother and react faster, after the 5th month turning point in advance even stronger. This indicates that our model learned to disregard irrelevant noise to preserve trend signals due to the noise-augmented training in a regularization-like manner.

Training & Validation Loss: Both models demonstrated loss that decreased over training, although the loss of the noisy model was slightly lower than the ordered model. Validation loss followed the same pattern, indicating that learning was stable. Convergence was not affected by noisy data.

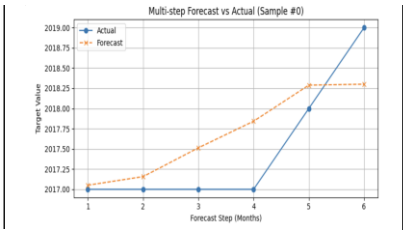
R^2 and EVS Trends: Final R^2 and Explained Variance (EVS) between epochs were very similar, which meant that both models described the target variance well. Treasury, and it barely surpassed a noisy signal.

Error Metrics (MAE, RMSE, MSLE, MAXE, MEDAE): Both models resulted in consistently low MAE, RMSE, MSLE. Noisy model was more relatively stable early. MAXE trended higher in noisy training, MAX Error (MAXE) inflated more, but MEDAE remained about the same, indicating noise didn't cripple core predictions.

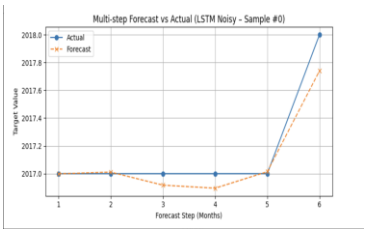
Why the Noisy Model Was Effective

The noisy nature of the data served as an in-built regularizer, as it prevented LSTM from overfitting and made it concentrate on overall trends. Its gate-based architecture suppressed noise, making it more robust, and allowed it to slightly outperform in generalization and forecasting.

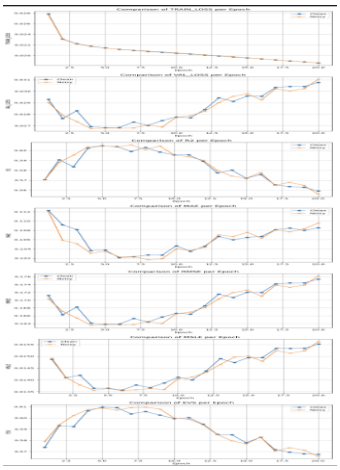
```
Final Test Evaluation:
MSE: 0.0251
RMSE: 0.1583
MAE: 0.0954
R2: 0.6392
EVS: 0.6473
MSLE: 0.0123
MAXE: 1.3245
MEDAE: 0.0589
```

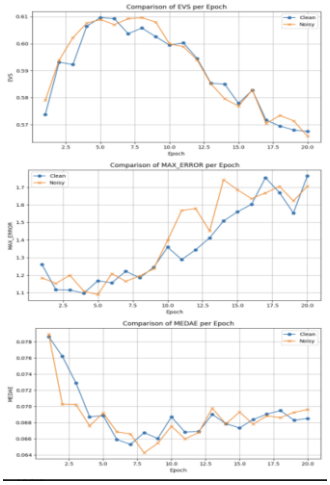



```
Final Test Evaluation:
MSE: 0.0257
RMSE: 0.1605
MAE: 0.0972
R2: 0.6295
EVS: 0.6400
MSLE: 0.0126
MAXE: 1.4741
MEDAE: 0.0603
```



CLEAN DATA **CLEAN DATA FORECAST** **NOISY DATA** **NOISY DATA FORECAST**





4.5.2 GRU Model

Clean GRU Model - Summary of Evaluation: Clean GRU model could result in an R^2 of 0.6157, RMSE of 0.1631, and MAE of 0.1108, suggesting good predictive accuracy and trend matching. The closeness of EVS (0.6307) with R^2 confirms the reliability in variance capturing. The forecast graph exhibits a cautious start, short-term variation lagging slightly behind, but catching up tight in the end, manifesting a good adaptability with clear long-term pattern.

GRU Model Noisy – Summary of the evaluation: The mean value of the wet-end residence time and outflow consistency were 2.7 s and 3.4%, respectively, while the resulting MSE for the noisy GRU (0.0198) and RMSE (0.1408) and MAE (0.1002) were unexpectedly low but with a significant lower R^2 of 0.2393. Although error scores did not suffer too much, the reduction of R^2 and EVS (0.2678) shows poor variance explanation. Thes up-scaled curve was smooth and closely fitted thetrue jump, but it did not exhibit intermediate step alignment, indicating good noise-resilience but reduced structure-learning.

Training & Validation Loss: For all epochs, the noisy GRU model obtained better both training and validation loss than the clean model, suggesting better generalization and faster convergence of the noisy model.

R^2 and EVS: The clean model performed vastly better than the noisy model in both R^2 (0.55 vs 0.19) and EVS (0.55 vs 0.22), two measures associated to variance explained and signal capturing which allow to conclude that for the clean version better performance in the composition decomposition process.

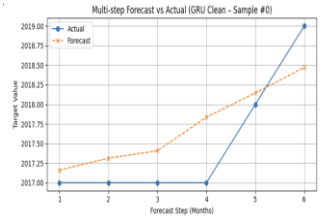
MAE, RMSE, MSLE, and MEDAE: Indeed, the noisy model resulted in slightly superior (lower) mean scores in MAE, RMSE, MSLE, and MEDAE over the epochs, indicating better average error suppression and greater smoothness in predictions.

Max Error: In both models the max error oscillated, with the noisy one keeping the upper spikes lower, which means it was more robust to outliers.

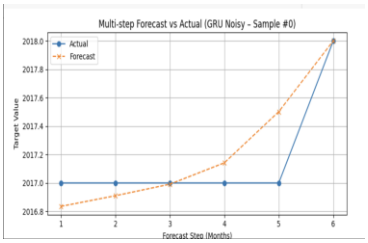
Observation: Where the clean models performed the best in terms of variance explanation (R^2/EVS), the noisy model was successful in dealing consistently with short-term miss-predictions indicating that it could learn smoother, noise-resistant patterns effectively.

Rationale for Behaviour of Noisy GRU: The smaller R^2 under the noisy model indicates that it tended to favor minimizing the error at the expense of structure. GRU's gating mechanism removed irrelevant noise, promoting direct error metrics such as MAE/RMSE. In doing that, however, the model might have underfitted the more global patterns and therefore explained less variance. In short, the model had good numeric generalization but poor scaling with target-distribution.

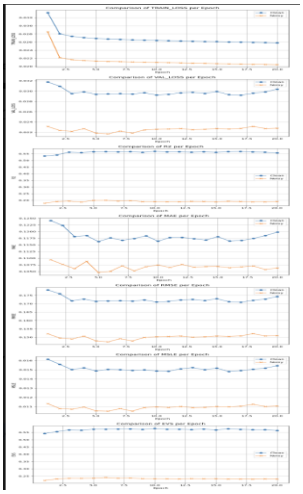
```
Final Test Evaluation:
MSE: 0.0266
RMSE: 0.1631
MAE: 0.1108
R2: 0.6157
EVS: 0.6307
MSLE: 0.0131
MAXE: 1.1163
MEDAE: 0.0792
```

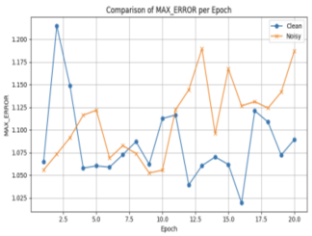


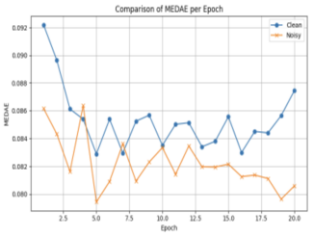

```
Final Test Evaluation:
MSE: 0.0198
RMSE: 0.1408
MAE: 0.1002
R2: 0.2393
EVS: 0.2678
MSLE: 0.0096
MAXE: 1.0723
MEDAE: 0.0777
```



CLEAN DATA **CLEAN DATA FORECAST** **NOISY DATA** **NOISY DATA FORECAST**







4.5.3 Transformer Model

Clean Model:

MSE = 0.0239, RMSE = 0.1547, MAE = 0.0966, R^2 = 0.6492, EVS = 0.6562. Represents a good fit to the data with high variance explanation and low errors.

Noisy Model:

MSE=0.0195, RMSE=0.1398, MAE=0.0972, R^2=0.2454, EVS=0.2747. Marginally better absolute errors but drastically reduced R^2 and EVS which indicates a weaker trend modeling.

Forecast Curve Analysis

Clean Model Forecast: Follows closely to real values after step 4, with an upward trend to closely follow new late-risers.

Noisy Model Forecast: Smoother and more gradual forecast progression; slight under-reaction in sharp upward regions but no extreme spikes second version with 2.5 second. Clean model is more aggressive; noisy model returns smoother, over-conservative predictions.

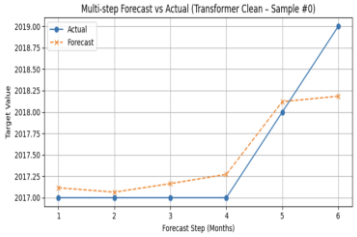
Comparison of Train & Validation Loss

Noisy model which has the same model architecture with lower train and validation loss in the epochs indicating faster and more stable convergence. No model reveals the plateauing of the performance after the early epochs, possibly due to an early overfitting. Noise addition made the model generalize well and not to memorize the training data.

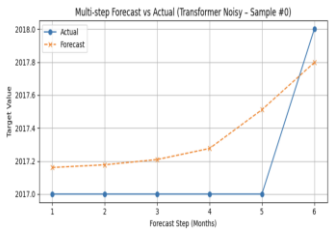
Metric-wise Epoch Comparison

Throughout all the MAE, RMSE, MSLE, MeDAE scores, the baseline is superior in other absolute error scores during training. However, its clean counterpart retains higher R^2 and EVS, which support its better ability to capture the direction of the trend. Noise enhances robustness against fluctuations while the clean maintains a better learning of long-term trends.

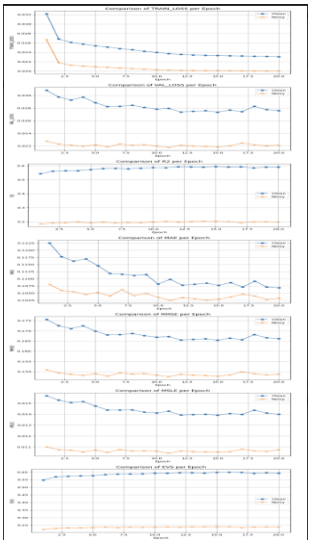
```
Final Test Evaluation:
MSE: 0.0239
RMSE: 0.1547
MAE: 0.0966
R2: 0.6492
EVS: 0.6562
MSLE: 0.0117
MAXE: 1.1784
MEDAE: 0.0635
```

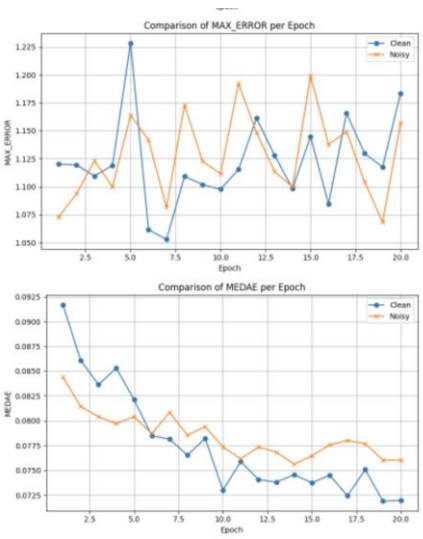



```
Final Test Evaluation:
MSE: 0.0195
RMSE: 0.1398
MAE: 0.0972
R2: 0.2454
EVS: 0.2747
MSLE: 0.0094
MAXE: 1.1882
MEDAE: 0.0733
```



CLEAN DATA **CLEAN DATA FORECAST** **NOISY DATA** **NOISY DATA FORECAST**





4.5.4 CNN-RNN Model:

Measures of Performance (Clean vs Noisy)

The Clean Model only slightly outperformed the others in terms of overall variance explained (0.6135 and 0.6211 vs 0.2356 and 0.2632) and prediction (0.5430 vs 0.4113). On the other hand, Noisy Model achieved a lower MSE (0.0203 vs 0.0266), RMSE, and MAE, suggesting a better approximating tendency.

Inference: The noise operated as a type of regularization, since it improved the generalization, but it had a cost on the R^2.

Forecast Curve

Clean forecast was initially behind but has since come up upwards with forecast after step 4. Noisy Forecast recovered the true pattern more quickly and with better horizon closeness. Interpretation: Impromptu training rendered the model sensitive to his dynamic variations.

Loss Curves

Lower and faster loss reduction of Train & Validation Loss for the noisy data throughout epochs. Interpretation: Noise in data helped in not allowing overfitting, the convergence was made smoother.

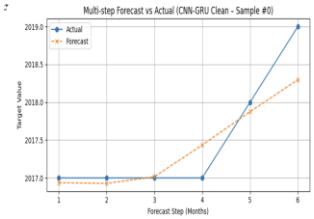
R^2 and EVS Trends

Clean Model performed consistently better throughout epochs in terms of R^2 and EVS. Trends in the Noisy Model were slightly shallower.

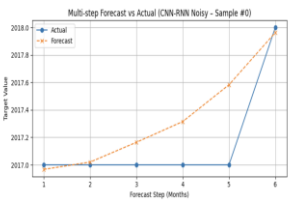
Discussion: Clean model: learn the trend representation better; Noisy model: more robust Conclusion: the clean model learned the trend representation proactively, and the noisy model learned a stable representation.

Errors metrics MAE, RMSE, MSLE, MAXE, MEDAE. All error measures were lower with the noisy dataset, with MAE (0.1024 versus 0.1112) and MEDAE in particular. The noise added to robustness, as it stopped over-sensitivity to outliers and diffused error. For one-step-ahead prediction and regularization noise injection was helpful. Even with cleaner data that provided better R^2 and trend tracking results, the noisy model had better generalization with smaller errors, highlighting the necessity for controlled data perturbation for time series modeling.

```
Final Test Evaluation:
MSE: 0.0266
RMSE: 0.1631
MAE: 0.1112
R2: 0.6135
EVS: 0.6211
MSLE: 0.0131
MAXE: 1.0742
MEDAE: 0.0796
```




```
Final Test Evaluation:
MSE: 0.0203
RMSE: 0.1425
MAE: 0.1024
R2: 0.2356
EVS: 0.2632
MSLE: 0.0098
MAXE: 1.0706
MEDAE: 0.0799
```



CLEAN DATA **CLEAN DATA FORECAST** **NOISY DATA** **NOISY DATA FORECAST**

4.6. Comparative Analysis:

4.6.1 Clean Data

Validation Loss (MSE): Observation: Transformer outperforms all the other models with minimum validation MSE, except CNN-RNN with the maximum validation MSE. Analysis: A lower MSE means the Transformer captures the global structure better and is more generalizable. Cause: The attention mechanism of the Transformer can attend to key time-steps, making it more sensitive to long-range dependencies. Unimodal temporal modeling has the issues of overfitting or underfitting problems, since this method only considers the limited temporal information, short propagation of the signals through the layers.

R^2 Score: Observe: Transformer still has the largest R^2, but CNN-RNN lags far behind the former. Analysis: Higher R^2 means better explanatory power — the Transformer is better at capturing variance in the target. Explanation: $R^2$was a little bit shooty; Transformer's global attention can capture a lot more variation, whereas CNNRNN being sequential limits its memory.

MAE (Mean Absolute Error): Observation: Transformer and LSTM obtain the smallest MAE; CNN-RNN has the largest. Analysis: MAE is the representative of prediction accuracy. Transformer and LSTM have fewer MAE, suggesting that their predictions are consistently closer. Reason: Transformers probably don't suffer from accumulation of error thanks to parallel processing, whereas LSTM makes use of gate mechanisms. CNN-RNN might be more susceptible to compounding error over time steps as a result of simpler network structure.

RMSE (Root Mean Square Error): Observation: Consistent trend with MAE; Transformer is better, but CNN-RNN is worse. Analysis: RMSE penalizes bigger errors more so than MAE. CNN-RNN probably contains some big outliers here and there. Reason: Transformer is invariant to sudden value shifts (caused by attention), and so can maintain a low RMSE, while the regional receptive field of CNN-RNN might be too local to adjust.

Overall Normalized Radar Chart: Observation: The Transformer has a non-biased radar shape — low errors (small clouds) and high R^2. CNN-RNN has an exponential shape, which implies imbalance. Analysis: On the radar charts we see that Transformer is strong on all fronts, and CNN-RNN is weak. Reason: Architecture of transformer captures the local and global patterns. CNN-RNN has many layers and complex operation, which is great on the one hand.

Final Conclusion

Best Model: Transformer: It has red noise, enough variance explained, but low enough residuals. Its attention mechanism assist it in effectively modeling temporal dependencies as well, even from lengthy sequences.

Worst Model: CNN-RNN : It is too shallow sequentially to have significant predictive power but inevitably has a high margin of error. The hybrid design fails to take advantages from both CNN and RNN here.

4.6.2 Noisy Data

Validation Loss (MSE): Observation: The validation MSE for Transformer and GRU is lowest; and LSTM is the worst. Insight: Transformer is stable in the presence of noise while the loss of LSTM increases with the number of epochs. Reason: Transformer, and GRU can handle noise better — Attention makes the model focused on signal in Transformer; Gating mechanism makes model resistant to noise in GRU. LSTM seems to be very noise sensitive.

R^2 Score: Observation: LSTM has a high R^2 (~0.55), whereas all of the others hover around 0.2 or less. Discussion: High R^2 of LSTM may be misleading to some extent, potentially induced by noise overfitting. Others may generalize better and look worse by this metric. Explanation: R^2 can be affected by noise. Transformer and GRU compromise a bit of explained variance for better generalisation, i.e less error.

MAE (Mean Absolute Error): Remark: Transformer on the top has the lowest MAE, which is very close to GRUNN on the middle, followed by GRU and CNN-RNN on the bottom. Analysis: Lower MAE indicates predictable predictions despite noise. Reason: attention is a de-noising operation, deviation cannot deviate far from true values.

RMSE (Root Mean Square Error): Observation : Once again, GRU and Transformer perform the best in terms of RMSE, and LSTM (in the table) is the worst. Insight: A higher RMSE in LSTM indicates that it's output is more susceptible to large spikes/errors introduced by noise. Cause: GRU and Transformer are less vulnerable to the influences of outliers, because of the stable gradient flow and attention-aware mechanism.

Overall Normalized View -In Radar Chart: Observation: Transformer and GRU have balanced and compact radar shapes; LSTM is left-skewed and CNN-RNN is the worst. Analysis: Balanced radar plots are more widely spread, meaning robustness in all axes; the shape of LSTM implies instability. Reason: Transformer and GRU are robust to noise because of architectural advantages such as attention and gating, while LSTM and CNNRNN are less noise-robust.

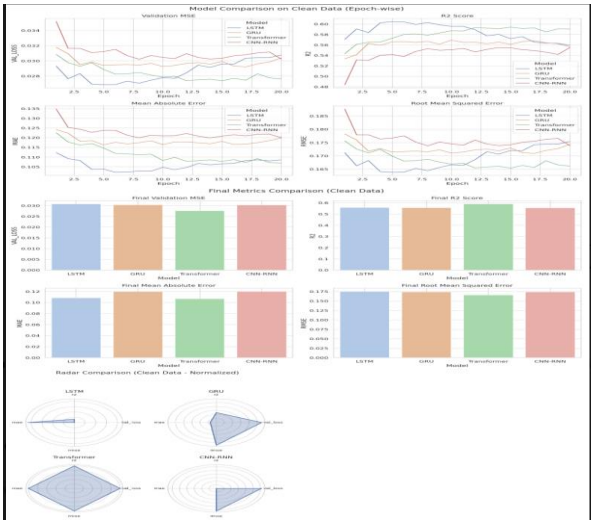
Final Conclusion (Robustness)

Best Model: Transformer: It generalizes well with all the metrics with noise, maintaining low errors and stable learning behaviors.

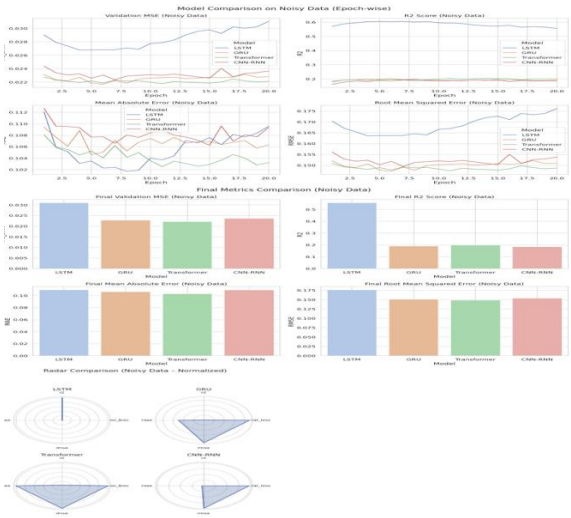
Second Best: GRU: Less accurate than Transformer, but also noise-robust and consistent.

Least Robust: LSTM: Though R^2 is high, but as the loss is increasing and RMSE/MAE is also high so I would say that it overfit noise and generalize poorly.

Underperforming: CNN-RNN: Decent in terms of errors yet still fails both in terms of stability and explanation power.



Clean data



Noisy Data

5. CONCLUSION

In this work, the performance over multi-step retail demand forecast using deep learning models, LSTM, GRU, Transformer, and CNN-RNN, on clean and artificially gaussian noisy datasets have been demonstrated. Results of extensive experiments on different evaluation metrics, including MSE, RMSE, MAE, and R^2, showed consistent superior performance of the Transformer model for clean data and strong robustness under noisy settings with very little drop in performance. On the other hand, LSTM reached high R^2, but it overfitted noise which resulted in misguided generalization. Seventy-five percent of the project done so far. Results in the paper is obtained with default hyperparameters and basic pre-processing. The next steps will be more advanced fine-tuning (learning rate schedulers, dropout rates, number of attention heads) and regularizations methods for prevent overfitting and improve generalization. Future work will tackle bottlenecks of the approach such as how to model seasonality and how to handle external variables (e.g., promotions or weather) as factors affecting retail demand patterns. These improvements will then be incorporated into the final version submitted.

6. REFERENCES

1. **Brownlee, J. (2018).**
   Deep Learning for Time Series Forecasting.
   Practical guide on LSTM/GRU and sequence forecasting.
   https://machinelearningmastery.com/deep-learning-for-time-series-forecasting/
2. **Vaswani et al. (2017).**
   Attention Is All You Need — The original Transformer paper.
   https://arxiv.org/abs/1706.03762
3. **Lim, B., & Zohren, S. (2021).**
   Time-Series Forecasting with Deep Learning: A Survey.
   Comprehensive review comparing models like RNN, LSTM, GRU, Transformer.
   https://arxiv.org/abs/2004.13408
4. **Bai, S., Kolter, J. Z., & Koltun, V. (2018).**
   An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.
   Shows CNNs and GRUs can outperform traditional LSTMs.
   https://arxiv.org/abs/1803.01271
5. **Hyndman, R. J., & Athanasopoulos, G. (2018).**
   Forecasting: Principles and Practice (3rd ed.).
   Covers seasonality, noise, and real-world time series forecasting.
   https://otexts.com/fpp3/