National College of Ireland

Configuration Manual

MSc Research Project
MSc in AI for Business

Karan Chenanda Palangappa
Student ID: 23408081

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Karan Chenanda Palangappa
Student ID:	23408081
Programme:	MSc in Artificial Intelligence for Business **Year:** 2025
Module:	Practicum
Lecturer:	Rejwanul Haque
Submission Due Date:	28-01-2026
Project Title:	Decentralized Federated Learning Framework Privacy Preserving and Transport Supply Chain Optimization with Blockchain Anchoring
Word Count:	3408 **Page Count:** 14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Karan Chenanda Palangappa

Date: 28-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Karan Chenanda Palangappa
Student ID: 24138371

1 Introduction (Purpose of this document)

This configuration guide is a detailed, step by step guide to configuring, operating and maintaining the data processing and machine learning pipeline applied in the Supply Chain Analytics project. It describes the manner in which the folders will be configured, the way the datasets will be prepared, the way the preprocessing scripts will be executed, the way the models will be trained (centralized, federated, and Random Forest), the way the outputs will be generated, and the way the results will be exported in a reproducible and user friendly fashion.

2 System Requirements

2.1 Hardware Requirements

- Minimum 8 GB RAM (16 GB recommended for large datasets)
- At least 10 GB of free disk space
- Multi-core CPU recommended for Random Forest training

2.2 Software Requirements

- Operating System: Windows 10/11, macOS, or Linux
- Python 3.10 or above : The main programming language to run and executing the model
- Jupyter Notebook / VSCode / PyCharm
- Required Python Libraries

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, r2_score,classification_report, accuracy_score,confusion_matrix,mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
import json
import hashlib
import datetime
```

✓ 0.0s Python

Figure 1 - Importing necessary libraries

3 Project Libraries

Create a folder by data
project_root/

```
|
├──> data/
|    ├──── > all datasets
|    └──── > processed_clean/ cleaned datasets
|
├──── outputs/
|    ├──── >model_metrics.json
|    ├──── > feature_importance_random_forest.csv
|    ├──── > feature_importance_federated.csv
|    ├──── > sample_predictions_global_model.csv
|    └──── >cleaned_master_sample.csv
```

Note: All code uses relative paths, so the project works on any machine without modification.

```
1
2
3  BASE_DATA_DIR = Path("/Users/KaranCP/Documents/Practicum/CA2/CA2_Final/Datasets")
4  PROCESSED_DIR = BASE_DATA_DIR / "processed_clean"
5  PROCESSED_DIR.mkdir(exist_ok=True)
6
7  print("New cleaned output folder created at:", PROCESSED_DIR)
```

Figure 2 - To import the datasets

4 Dataset Preparation

4.1 Dataset Placement

Place the datasets into the folder below are the links for datasets from Kaggle

1. Food balance sheets data set (Damola Adedoyin) – **supply_chain_data.csv**
   https://www.kaggle.com/datasets/harshsingh2209/supply-chain-analysis
2. Supply Chain Management Dataset – SCM_dataset.xlsx
   https://www.kaggle.com/datasets/lastman0800/supply-chain-management/data
3. Real Data on Logistics (Lakhankumawat) – training_raw_file.csv
   https://www.kaggle.com/datasets/lakhankumawat/real-data-on-logistics
4. Dataset in Supply chain in Blockchain – trust_chain_dataset.csv
   https://www.kaggle.com/datasets/ziya07/blockchain-enabled-dataset-for-supply-chain
5. Dataset High Dimensional Supply Chain Inventory – **supply_chain_dataset1.csv**
   https://www.kaggle.com/datasets/ziya07/high-dimensional-supply-chain-inventory-dataset/data

6. Transportation and Logistics Tracking Dataset – **refined sheet**
   https://www.kaggle.com/datasets/nicolemachado/transportation-and-logistics-tracking-dataset
7. Logistics and Supply Chain Dataset - **dynamic_supply_chain_logistics_dataset.csv**
   https://www.kaggle.com/datasets/datasetengineer/logistics-and-supply-chain-dataset
8. Dataco Smart Supply Chain on Big Data Analysis (Shashwawork) – **DataCoSupplyChainDataset.csv**
   https://www.kaggle.com/datasets/shashwatwork/dataco-smart-supply-chain-for-big-data-analysis

Each dataset may be in CSV or Excel format. Import and change the path directory where you save the downloaded datasets to do the further process as shown in figure 2

4.2 Processed Clean Folder

```python
# importing the logistics dataset
file_path = BASE_DATA_DIR / "dynamic_supply_chain_logistics_dataset.csv"
df = pd.read_csv(file_path)
# Cleaning the column names
df.columns = [c.strip().lower().replace(" ", "_") for c in df.columns]
# Renaming the columns to match master dataset
df = df.rename(columns={
    "timestamp": "timestamp",
    "warehouse_inventory_level": "quantity",
    "fuel_consumption_rate": "shipping_cost",
    "supplier_reliability_score": "from_location",
    "delivery_time_deviation": "delivery_status"
})
# Add dataset label
df["source_dataset"] = "logistics_kpi"
# Save cleaned dataset
output_path = PROCESSED_DIR / "logistics_kpi_clean.csv"
df.to_csv(output_path, index=False)

print(" Saved cleaned logistics KPI dataset →", output_path)
df.head()
✓ 2.0s
```

Figure 3- Cleaning the each datasets to normalize them before merging in cleaner format

The above code helps to make the clean data a "project_root/data/processed_clean/" which store each files name clean version for further processing of the data. There are multiple codes repeated in same manner which to run each datasets to make cleaner datasets. Below are the file names which will store in the processed clean data.

Files include names like:
blockchain_supply_clean.csv
highdim_inventory_clean.csv
supply_chain_analysis_clean.csv
logistics_kpi_clean.csv
dataco_smart_ali_clean.csv
real_logistics_clean.csv
supply_chain_analysis_clean.csv

transport_tracking_clean.csv

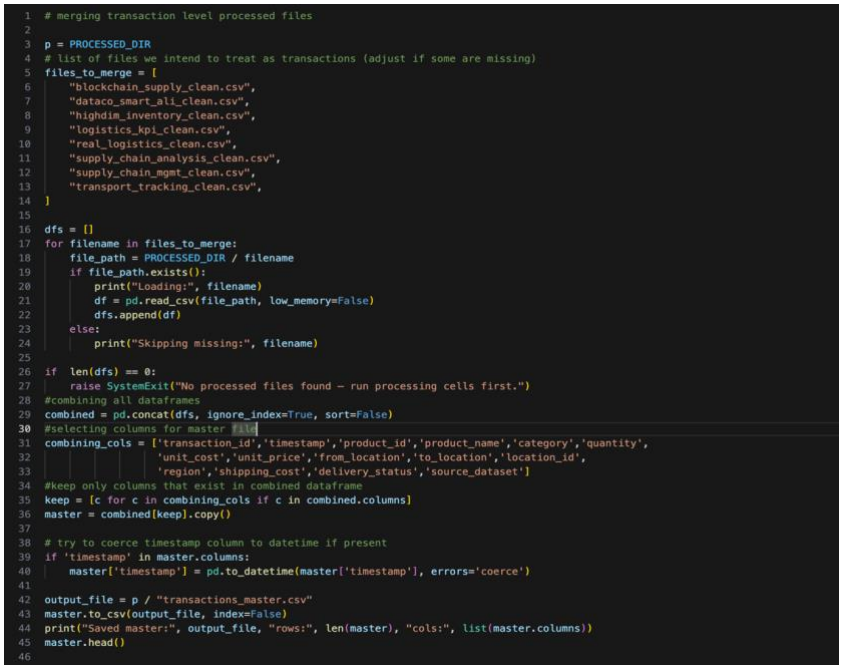
The code runs as cleaning and make column standardization which we normalise into lower case and underscore and renaming into cleaner understandable for further process. And creating the source of dataset tag where the data is sourced from and also performing any missing values in each datasets as well (figure 3)

4.3 Master Transaction File Creation

Once the cleaning procedure was done in all the datasets, we combined all the valid records in one unified transactional dataset. The merge script is depicted in Figure 4, and it loads each cleaned dataset and concatenates them into a single master file. The datasets successfully cleaned and available were merged, as shown below. These files are joined to give the final transactions-master.csv that contains the entire consolidated transaction-level data to do additional analysis and modelling. The merge code combines only valid datasets:

selected_files = [
  "blockchain_supply_clean.csv","dataco_smart_ali_clean.csv",
   "highdim_inventory_clean.csv","logistics_kpi_clean.csv",
   "real_logistics_clean.csv","supply_chain_analysis_clean.csv",
   "supply_chain_mgmt_clean.csv","transport_tracking_clean.csv",
]

Resulting file saved as: data/processed_clean/transactions_master.csv



```
# merging transaction level processed files

p = PROCESSED_DIR
# list of files we intend to treat as transactions (adjust if some are missing)
files_to_merge = [
    "blockchain_supply_clean.csv",
    "dataco_smart_ali_clean.csv",
    "highdim_inventory_clean.csv",
    "logistics_kpi_clean.csv",
    "real_logistics_clean.csv",
    "supply_chain_analysis_clean.csv",
    "supply_chain_mgmt_clean.csv",
    "transport_tracking_clean.csv",
]

dfs = []
for filename in files_to_merge:
    file_path = PROCESSED_DIR / filename
    if file_path.exists():
        print("Loading:", filename)
        df = pd.read_csv(file_path, low_memory=False)
        dfs.append(df)
    else:
        print("Skipping missing:", filename)

if len(dfs) == 0:
    raise SystemExit("No processed files found — run processing cells first.")
#combining all dataframes
combined = pd.concat(dfs, ignore_index=True, sort=False)
#selecting columns for master file
combining_cols = ['transaction_id','timestamp','product_id','product_name','category','quantity',
                  'unit_cost','unit_price','from_location','to_location','location_id',
                  'region','shipping_cost','delivery_status','source_dataset']
#keep only columns that exist in combined dataframe
keep = [c for c in combining_cols if c in combined.columns]
master = combined[keep].copy()

# try to coerce timestamp column to datetime if present
if 'timestamp' in master.columns:
    master['timestamp'] = pd.to_datetime(master['timestamp'], errors='coerce')

output_file = p / "transactions_master.csv"
master.to_csv(output_file, index=False)
print("Saved master:", output_file, "rows:", len(master), "cols:", list(master.columns))
master.head()
```

Figure 4 - Merging the cleaned datasets into one transaction master file

4.4 Exploratory Analysis

Once the master file was prepared, I needed to prepare it to familiarize with exploratory data analysis (EDA). Figure 5 presents the preprocessing script that is used on the merged data. First, the timestamp column is changed to the correct format of date time to permit effective trend and time series examination. All numerical types, including quantity, unit_cost, unit_price, and shipping_cost are converted to the numeric types with invalid values safely converted to NaN. In order to be consistent and not to create holes in downstream analysis, categorical values (e.g., product category, region, and delivery status) that are missing are filled with default labels, e.g. Unknown or Pending. After these transformations have been made, a summary check is carried out to ensure that the data has the right shape, and that the problem of missing values has been addressed. All the visualisations, feature engineering attributes and machine learning models that will be applied later in the study are based on this processed dataset..

```
# Loading master
df = pd.read_csv("/Users/KaranCP/Documents/Practicum/CA2/CA2_Final/Datasets/processed_clean_data/transactions_master.csv")

# Converting timestamp to datetime
df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce")

# Converting numeric columns
for col in ["quantity", "unit_cost", "unit_price", "shipping_cost"]:
    df[col] = pd.to_numeric(df[col], errors="coerce")

# Fill some missing values
df["category"].fillna("Unknown", inplace=True)
df["region"].fillna("Unknown", inplace=True)
df["delivery_status"].fillna("Pending", inplace=True)

# Check summary again
print("Cleaned shape:", df.shape)
print("\nMissing values after cleaning:\n")
print(df.isna().sum().sort_values(ascending=False).head(10))
✓ 1.2s
```

Figure 5 - EDA performing part 1

The second step involved the preparation of the generated consolidated master dataset to be subjected to exploratory data analysis (EDA). Figure 6 presents the logic of the preprocessing. The script will start by transforming the timestamp column into a reasonable date format, which will allow the correct trend based and time analysis. Then, the numerical fields (quantity, unit_cost, unit price, and shipping cost) are forced to the numbers and the invalid values are converted to NaN, so further downstream processing is not affected by the invalid values.

There are also missing values in the key categorical attributes, which are standardised. The fields category, region and delivery status are populated with the defaults of Unknown and Pending to prevent the null related inconsistencies in the process of grouping and visualisation. After performing these transformations, the script will give the cleaned data format and list of any remaining missing values, which will ensure that the data is now prepared to be examined through EDA and model building. This processed data is the basis of all the further visualisation, feature engineering and modelling procedures in the study.

```python
# Fix mixed data type issues (force IDs to strings)
df["transaction_id"] = df["transaction_id"].astype(str)
df["product_id"] = df["product_id"].astype(str)
df["from_location"] = df["from_location"].astype(str)
df["to_location"] = df["to_location"].astype(str)

# Remove rows where critical fields are all missing
df = df.dropna(subset=["timestamp", "product_id", "quantity"], how="all")

# Fill missing numeric values using median
num_cols = ["quantity", "unit_cost", "unit_price", "shipping_cost"]
for col in num_cols:
    df[col] = pd.to_numeric(df[col], errors="coerce")
    df[col] = df[col].fillna(df[col].median())

# Fill missing text values
df["product_name"] = df["product_name"].fillna("Unknown Product")
df["location_id"] = df["location_id"].fillna("Unknown")

print("Cleaned and typed shape:", df.shape)
df.info()

✓ 0.7s
```

Figure 6 - EDA Performing part 2

Once all cleaning and preprocessing is done, final master dataset is stored to be subjected to downstream analysis and modelling. Figure 7 displays the code that was used to save the completely cleaned data as transactionsmasterfinal.csv. It is the last and consolidated file with all aggregated supply chain transactions, which is not in any way failed to undergo all transformations, typic corrections, and imputations. After the dataset has been saved, a summary statistics command is run to confirm the design and distribution of values in the final dataset. This gives a brief summary of numeric and categorical fields, which makes sure that the dataset is prepared to continue with further exploratory analysis, model building, and experiments of federated learning that are going to come.

```python
# Save final cleaned master dataset
df.to_csv("/Users/KaranCP/Documents/Practicum/CA2/CA2_Final/Datasets/processed_clean_data/transactions_master_final.csv", index=False)
print("Final master dataset saved successfully!")
```

[37] ✓ 3.6s

```
Final master dataset saved successfully!
```

```python
# Summary statistics of final cleaned master dataset
df.describe(include="all").transpose()
```

[38] ✓ 0.5s

Figure 7 - Saving Cleaned Dataset

4.5 Visualization

This piece of code creates a horizontal bar graph of the Top 10 Product Categories by the number of transactions. It has the same plotting element applied to all other visualisations in the EDA section. The code loads the cleaned dataframe, calculates the most commonly used categories with value counts and presents them with the help of bar plot in Matplotlib. Axis labels, title, and light grid have been used to make the visual readable. The remaining visualisations in the project are built in the same principle that is, they only alter the choice of column(s) and type of plot but they still utilize the same plotting pipeline as in Figure 8.

```
#Visulizations
# Top 10 categories by transaction count
plt.figure(figsize=(8,5))
df["category"].value_counts().head(10).plot(
    kind="barh",
    color="skyblue"
)
plt.title("Top 10 Product Categories")
plt.xlabel("Count")
plt.ylabel("Category")
plt.grid(axis="x", linestyle="--", alpha=0.4)
plt.show()
✓ 0.4s
```

Figure 8 - Visualization code of bar plot

The remaining code follows same flow of steps are the mentioned visualisation which has in code in table 1

Visualization Name	Purpose of the Visualization
Top 10 Product Categories	Shows the most common product categories by the volume of transaction. Helps determine categories of high demand.
Delivery Status Distribution	The percentage of various delivery statuses (e.g. completed, pending, late) is shown and is a summary of delivery performance.
Quantity vs Shipping Cost	Scatterplot shows the association between the quantity ordered and the resulting shipping cost. helpful in identifying patterns of costs.
Average Shipping Cost by Region	Comparison of the shipping costs in different regions to determine the high-cost areas and inefficiency in logistics.
Average Unit Price vs Unit Cost per Category	Displays the pricing and cost differences by category in order to determine the margins.
Daily Transaction Volume	Shows how transactions change with time, used to identify periods of maximum activity and trends of operations.
Distribution of Numeric Fields	Shows the histograms of the key numeric fields in order to learn about the spread of values, skewness, and possible outliers.
Top 10 Shipping Routes	Determines the most common routes in the shipping routes utilized in the data..
Total Revenue by Product Category	Displays the highest revenue sources, which help the analysis of profitability.
Delivery Status by Source Dataset	Compares statuses of delivery between datasets to identify anomalies or discrepancies in the quality of data.
Datasets with Highest Late/Pending Deliveries	Displays the sources or datasets that have the most significant contributions to the late/pending deliveries.

Table 1 - Visualization plot lists

5 Feature Engineering

This step gives the cleaned master dataset a feature engineering step before it goes to the modelling stage. The script loads the processed master file and generates a number of new derived fields, one of them being a binary delay flag (is_delayed), profit margin, total order value and time related attributes such as delivery month and delivery year. There is also simplification of the region names. Once these new features have been created the improved

dataset is stored as a new CSV file to be used in downstream modelling. The code that was used to achieve this transformation is displayed in Figure 9.

```python
# Feature Engineering on Master Dataset

# Load your cleaned master dataset
df = pd.read_csv("/Users/KaranCP/Documents/Practicum/CA2/CA2_Final/Datasets/processed_clean_data/transactions_master.csv")

#  Delivery flag (binary: 1 = delivered on time, 0 = delayed/pending)
df["is_delayed"] = df["delivery_status"].apply(
    lambda x: 1 if str(x).lower() in ["late delivery", "pending", "overdue"] else 0
)

# Profit margin (difference between selling and cost price)
df["profit_margin"] = df["unit_price"] - df["unit_cost"]

# Total order value
df["total_value"] = df["quantity"] * df["unit_price"]

# Estimate delivery time buckets
df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce")
df["delivery_month"] = df["timestamp"].dt.month
df["delivery_year"] = df["timestamp"].dt.year

# Region simplification
df["region_clean"] = df["region"].str.extract(r"([A-Za-z]+)").fillna("Unknown")

# Save feature-engineered dataset
df.to_csv("/Users/KaranCP/Documents/Practicum/CA2/CA2_Final/Datasets/processed_clean_data/transactions_featured.csv", index=False)

print("✅ Feature engineered dataset saved successfully.")
df.head()
```

Figure 9 - Feature Engineering Pipeline on Master Dataset

This step visualises the relation of the engineered features with each other. The script takes the key numeric features generated in the course of feature engineering, including profitmargin, totalvalue, and the binary delay feature isdelayed and calculates the correlation matrix of them with the help of .corr(). Seaborn is then used to create a heatmap that demonstrates these correlations. This assists in confirming that new features designed are acting as expected by the engineer before they are applied in modelling. Figure 10 demonstrates the specific code that was used in calculating and plotting the correlation matrix.This step illustrates the relationship between the engineered features with each other. The script picks the key numerical features generated in the feature engineering process (i.e. profitmargin, totalvalue, and the binary delay flag isdelayed) and calculates their correlation matrix with the help of the .corr() method. Seaborn is then used to create a heatmap to clearly display these correlations. This aids in making sure that expected behaviour of newly engineered features is as expected before they are utilized in modelling. The precise code that was used in computing and plotting of the correlation matrix is illustrated in Figure 10

```python
# Correlation Between Engineered Features

corr = df[["quantity", "unit_cost", "unit_price", "shipping_cost", "profit_margin", "total_value", "is_delayed"]].corr()

plt.figure(figsize=(10,6))
sns.heatmap(corr, annot=True, cmap="magma", fmt=".2f")
plt.title("Correlation Between Engineered Features")
plt.show()
```

✓ 0.5s

Figure 10 - Coorelation Analysis of Engineered Features

In the below code snippet, the key numerical fields (shippingcost, quantity and totalvalue) are clipped using 1st and 99 th percentile values. The lower and upper percentile thresholds are calculated and the script clips off any values less than these values. This is to guarantee that the dataset is stabilised through the reduction of extreme outliers prior to modelling. The code actually used to compute and apply clipping on the selected columns is presented in figure 11

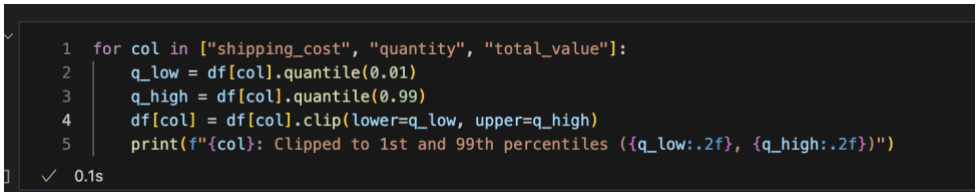


Figure 11 - Percentile Based Outlier Code

This fragment creates the visual analogy between outlier treatment. Each chosen column has two boxplots generated by the code, one of the raw distribution, and the other of the cleaned distribution. This gives an opportunity to verify in a short time that the extreme values were properly mitigated. The plotting routine employed in creating the before and after boxplots is shown in Figure 12.

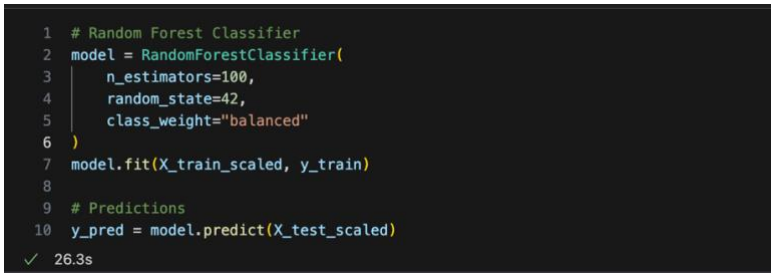


Figure 12 - Before and After Outlier Clipping

6 Model Training

6.1 Random Forest classifier

This sample prepares the labeled data needed in the training of the model. Because all the prevalent imports have been done previously, as well as the transactionmasterfeatured.csv file loaded, the only attention in this step is to choose the target variable (is_delayed) and the engineered numerical features required to model. Any missing values (in case they exist) within the feature matrix are replaced with a zero as a constant. The step prepares the entire X (features) and y (target) input in the further train test split and scaling phases (Figure 13).

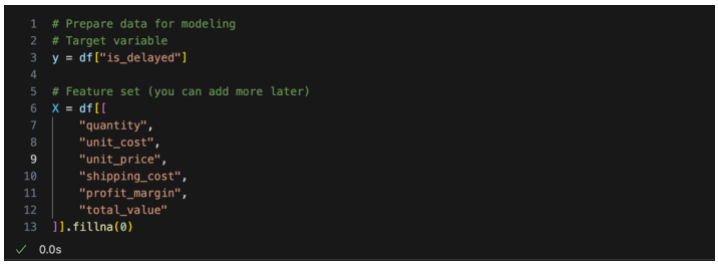


Figure 13 - Preparing Feature Matrix and Target Variable for Model Training

In this step, the training and testing subsets are put together and the prepared feature matrix (X) and the target variable (y) are separated. The stratified split is applied to make sure that

the ratio of delayed to non delays deliveries is the same in both sets so that there is no imbalance of classes as the models are being trained. The data is split into 80% training and 20% testing with a fixed random seed and is deterministic. Before scaling and model fitting, the resulting shapes are printed in order to ensure that the data has been partitioned properly and to verify the shapes (Figure 14).

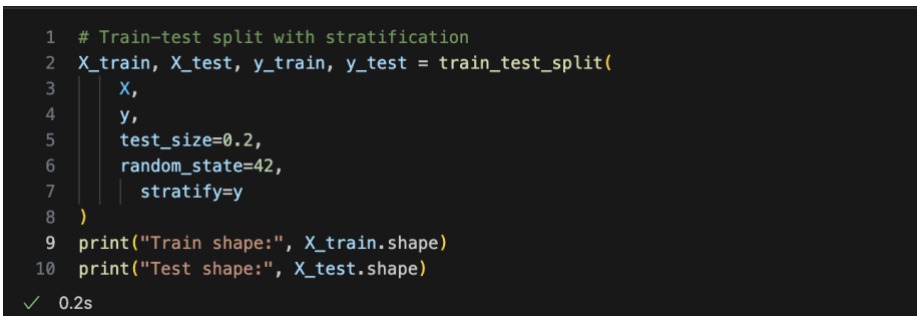


Figure 14 - Train Test Split with Stratification

This is used to train the Random Forest Classifier on the scaled training features. There is a balanced weight on the classes, delayed and non delayed delivery, to ensure that the two classes are given equal weight in the model. The classifier is set to use 100 trees (n_estimators=100) and a fixed random seed so that it can be reproduced. The model will produce predictions in the scaled test data after training and this will be assessed later by measuring accuracy, confusion matrix, and classification measures. Figure 15.

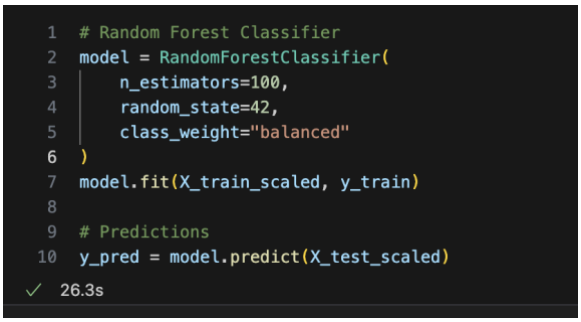


Figure 15 - Training the Random Forest Classifier and Generating Predictions

This step is used to test the performance of the Random Forest Classification model that has been trained during the previous steps. As we are predicting the delay of a delivery (is_delayed), we obtain the accuracy score, and the classification report (precision, recall, F1-score) to know how the model functions on the test data. Ytest and y-pred are then used to create the confusion matrix and Seaborn is used to visualize it using heatmap. This is a visual that can be used to determine the number of deliveries that were placed into the right category and patterns of misclassification. The plotting format has been done in the same way as was done in previous figures, with the Matplotlib plot and labeled axes and description title. Figure 16.

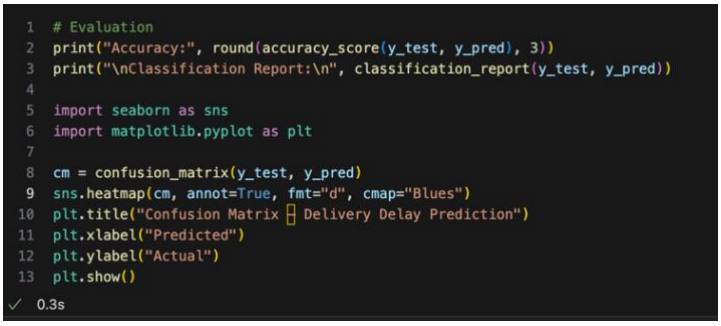


Figure 16 - Model Evaluation Using Confusion Matrix

6.2 Random Forest Regressor

This action appraises the Random Forest Regressor with usual regression measurements. The trained model is used to predict the shipping costs of the test set and performance is evaluated by MAE, RMSE and R2. These measures can be used to measure the accuracy of prediction without employing the confusion matrices, as it is a regression problem, not a classification one. Figure 17.

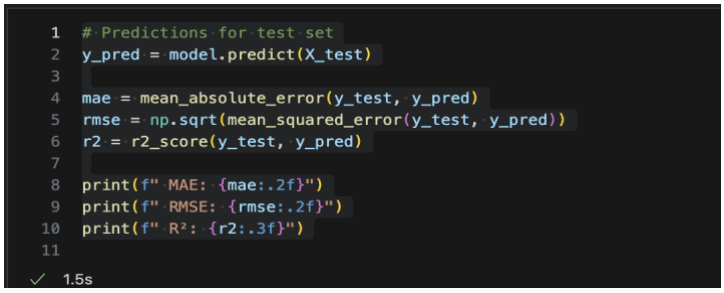


Figure 17 -Random Forest Regressor Training (Shipping Cost Prediction)

This line creates a scatter plot between predicted shipping costs and the real ones in the test data of the model. The plot gives a visual evaluation of the effectiveness of the Random Forest Regressor to determine the underlying pattern of the data. The points that are more near to the diagonal line are associated with a higher accuracy in the predictions based on real values. This visual analysis is used to supplement the previous numerical results (MAE, RMSE, R2) and allow the confirmation of the model results. Figure 18

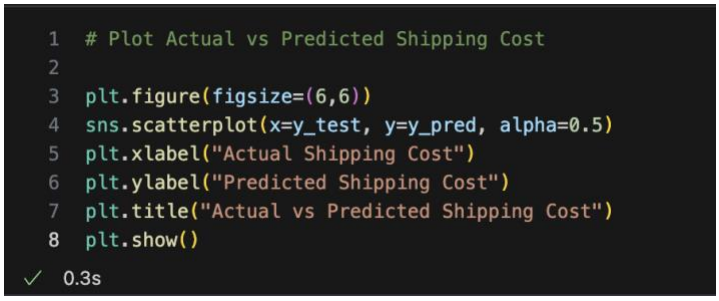


Figure 18 - Actual vs Predicted Shipping Cost (Regression Model)

6.3 Federated Avg

The provided snippet executes the fundamental algorithm of a Federated Averaging simulation based on the distribution of the dataset among several clients, local linear regression training on each part, and the average of the resulting coefficients and intercepts to create a global federated model. It allows completing decentralized model training without accessing raw data as evidence of how a simple federated learning pipeline can be implemented on the featured transaction dataset. The provided snippet executes the fundamental algorithm of a Federated Averaging simulation based on the distribution of the dataset among several clients, local linear regression training on each part, and the average of the resulting coefficients and intercepts to create a global federated model. It allows completing decentralized model training without accessing raw data as evidence of how a simple federated learning pipeline can be implemented on the featured transaction dataset. Figure 19

```
# SIMPLE FEDERATED LINEAR MODEL (FedAvg)
clients = np.array_split(df, 5)

coefs = []
intercepts = []

for cdf in clients:
    Xc = cdf[features]
    yc = cdf[target]

    m = LinearRegression()
    m.fit(Xc, yc)

    coefs.append(m.coef_)
    intercepts.append(m.intercept_)

avg_coef = np.mean(coefs, axis=0)
avg_intercept = np.mean(intercepts)

y_pred_global = np.dot(X, avg_coef) + avg_intercept
mae_global = mean_absolute_error(y, y_pred_global)
r2_global = r2_score(y, y_pred_global)
```

Figure 19 - Federated Linear Regression (FedAvg) Implementation

This snippet compares the prediction performance of three models (Federated Linear Regression (FedAvg), centralized linear regression and random forest regression) based on their Mean Absolute Error (MAE) scores. The visual is very explicit on the performance of each model on the same dataset, thus one can easily interpret the accuracy of models. On top of each bar, numerical values of MAE are presented in order to compare them easily.

```
# Model performance table
model_names = ["Federated", "Centralized", "Random Forest"]
mae_scores = [mae_global, mae_cent, rf_mae]

plt.figure(figsize=(7,5))
bars = plt.bar(model_names, mae_scores)

# Add values on bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.1, f"{yval:.2f}",
             ha='center', fontsize=12)

plt.title("Model Comparison – MAE")
plt.ylabel("Mean Absolute ErrorMAE")
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.show()
```

Figure 20 - Model Comparison (MAE) Code Snippet

Figure 21 represents the coefficient obtained in the Federated Linear Regression model following the FedAvg aggregation process. This step provides the visualisation of the distribution of the globally averaged model weights across each feature, which proves that the federated training loop has been executed properly. The snippet merely loads the aggregated coefficients (coef_df) and plots them in a horizontal bar chart and its output indicates the relative contribution of each attribute in the input. This visual check will mean that the federated model was properly constructed and that the values of the coefficients are duly calculated and reported following client level training.

```
# Model performance table
model_names = ["Federated", "Centralized", "Random Forest"]
mae_scores = [mae_global, mae_cent, rf_mae]

plt.figure(figsize=(7,5))
bars = plt.bar(model_names, mae_scores)

# Add values on bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.1, f"{yval:.2f}",
             ha='center', fontsize=12)

plt.title("Model Comparison – MAE")
plt.ylabel("Mean Absolute ErrorMAE")
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.show()
```

Figure 21 - Federated model coefficients

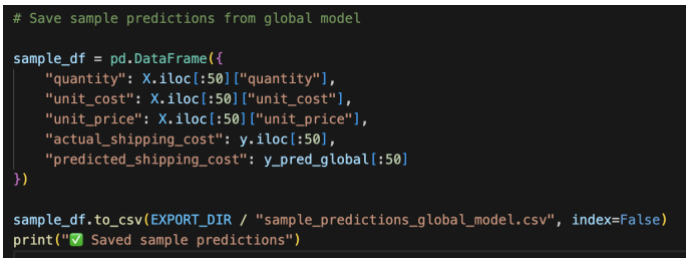
7 Exporting Outputs

This piece of code sets up a special Final_Exports folder where all output files generated by the project process should be put. The directory is automatically generated in case it is not there and its complete resolved path is shown to confirm its presence.

```
1  from pathlib import Path
2
3  EXPORT_DIR = Path("Final_Exports")
4  EXPORT_DIR.mkdir(exist_ok=True)
5
6  print("Export directory created at:", EXPORT_DIR.resolve())
✓ 0.0s
```

Figure 22 - Creating Export Directory

This code takes the 50 initial records of the feature set and creates a sample prediction dataset in global (FedAvg) model. The output consists of the quantity and the cost values, the actual shipping cost, projected shipping cost. Sampling is then verified and analyzed by exporting the output sample in the form of a CSV file to the Final_Exports directory.



```
# Save sample predictions from global model

sample_df = pd.DataFrame({
    "quantity": X.iloc[:50]["quantity"],
    "unit_cost": X.iloc[:50]["unit_cost"],
    "unit_price": X.iloc[:50]["unit_price"],
    "actual_shipping_cost": y.iloc[:50],
    "predicted_shipping_cost": y_pred_global[:50]
})

sample_df.to_csv(EXPORT_DIR / "sample_predictions_global_model.csv", index=False)
print("✅ Saved sample predictions")
```

Figure 23 - Saving Sample Predictions from the Global Model

This below code will export the initial 20 rows of the cleaned master dataset to a CSV file within the FinalExports folder. This does give an instant picture of the processed data and it is used in validation and documentation as well as downstream verification. The downloaded file (cleanedmaster_sample.csv) gives the user an opportunity to see the format and structure of the final clean data without necessarily loading the entire file

```
# Save cleaned master dataset sample (first 20 rows)
df.head(20).to_csv(EXPORT_DIR / "cleaned_master_sample.csv", index=False)
print("✅ Saved cleaned master sample (20 rows)")
```

Figure 24 - Saving a Cleaned Master Dataset Sample