National College of Ireland

Configuration Manual

MSc Research Practicum
MSc Data Analytics

Sidhesh Bhambid
Student ID: x23270110

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Sidhesh Bhambid

Student ID: x23270110

Programme: MSCDAD_A **Year:** 2024-25

Module: MSc Research Practicum

Supervisor: Vladimir Milosavljevic

Submission Due Date: 11-08-2025

Project Title: Optimising Supply Chain Logistics with Predictive Analytics

Word Count: 1055 **Page Count**: 6

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: SIDHESH BHAMBID

Date: 10-08-2025

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

Sidhesh Bhambid
x23270110

1 Introduction

This document serves as a complete configuration and operational manual for the "Optimising Supply Chain Logistics with Predictive Analytics" project. The primary purpose of this manual is to provide system administrators, developers, and data scientists with the necessary instructions to set up the software environment, execute the model training pipeline, and utilize the trained model for making real-time risk predictions.
The project's core output is a highly accurate Support Vector Machine (SVM) model capable of classifying the operational risk of a logistics shipment into one of three categories: 'Low Risk', 'Moderate Risk', or 'High Risk'. This manual details the hardware and software prerequisites, the steps for installing required libraries, the project's file structure, and provides clear, executable code snippets for both training the model from scratch and using the pre-trained artifacts for inference on new data. By following these instructions, users will be able to replicate the research outcomes and integrate the predictive model into a practical workflow for proactive supply chain management.

2 System Requirements

To successfully run the project code, the following hardware and software configurations are required.

2.1 Hardware Requirements

- **Processor:** 64-bit processor (e.g., Intel Core i5/i7, AMD Ryzen 5/7 or equivalent)
- **RAM:** Minimum 8 GB, 16 GB recommended for smoother performance with large datasets.
- **Storage:** Minimum 5 GB of free disk space to accommodate the dataset, Python environment, and saved model artifacts.

2.2 Software Requirements

- **OS Requirements:** Windows 10/11, macOS (11.0 and later) or a contemporary Linux distribution (like Ubuntu 20.04 or later).
- **Python Requirements:** Python 3.8, 3.9, or 3.10. The code has been developed and tested with these versions.
- **Python Libraries:** The following libraries are necessary. Section 3 has a requirements.txt for easy installation.
    - pandas
    - numpy
    - scikit-learn
    - matplotlib
    - seaborn
    - plotly
    - jupyterlab or notebook (for running the original notebook)

- joblib
- tabulate (for formatted output in the testing script)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import itertools
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_score, roc_auc_score, precision_score, recall_score
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, StratifiedKFold
import time
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA

from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize

from sklearn.model_selection import GridSearchCV
```

Figure 1: Libraries

3 Environment Setup

Follow these steps to create an isolated Python environment and install all necessary dependencies. This ensures that the project's dependencies do not conflict with other Python projects on your system.

Step 1: Create a Project Directory

Create a new folder for the project and navigate into it using your terminal or command prompt.
mkdir x23270110_code artifacts
cd x23270110_code artifacts

Step 2: Create a Virtual Environment

It is highly recommended to use a virtual environment.

For Windows
python -m venv thesis_venv

For macOS/Linux
python3 -m venv thesis_venv

Step 3: Activate the Virtual Environment
For Windows
.\thesis_venv\Scripts\activate

For macOS/Linux
source thesis_venv/bin/activate

Your terminal prompt should now be prefixed with (thesis_venv).

Step 4: Install Required Libraries
Create a file named requirements.txt in your project directory and paste the following content into it:

requirements.txt:
pandas==2.1.4
numpy==1.26.2
scikit-learn==1.3.2
matplotlib==3.8.2
seaborn==0.13.0
plotly==5.18.0
jupyterlab==4.0.9
joblib==1.3.2
tabulate==0.9.0
Now, install all the libraries at once using pip:
pip install -r requirements.txt
Your environment is now fully configured and ready for use.

4 Project Structure and Key Files

Place the provided project files into your main project directory. The expected structure is as follows:
x23270110_code artifacts/

```
├── thesis_venv/                # The virtual environment folder
│
├── saved_models/               # Directory for all trained model artifacts
│   ├── best_svm_model.pkl      # The trained SVM model
│   ├── scaler.pkl              # The fitted StandardScaler object
│   ├── pca_transformer.pkl     # The fitted PCA object
│   ├── label_encoders.pkl      # The fitted LabelEncoder objects
│   ├── column_info.pkl         # Information about feature columns
│   └── allowed_categories.pkl. # Allowed categories for dropdowns
│
├── log_sup_dataset.csv         # The raw dataset for training
│
├── x23270110_code_file.ipynb   # The original Jupyter Notebook with all analysis
│
└── requirements.txt            # The list of Python dependencies
```

5 Model Training Execution

You can train the model by running the Jupyter Notebook. The final retraining block is the most important as it generates the artifacts in the saved_models directory.
Instructions:

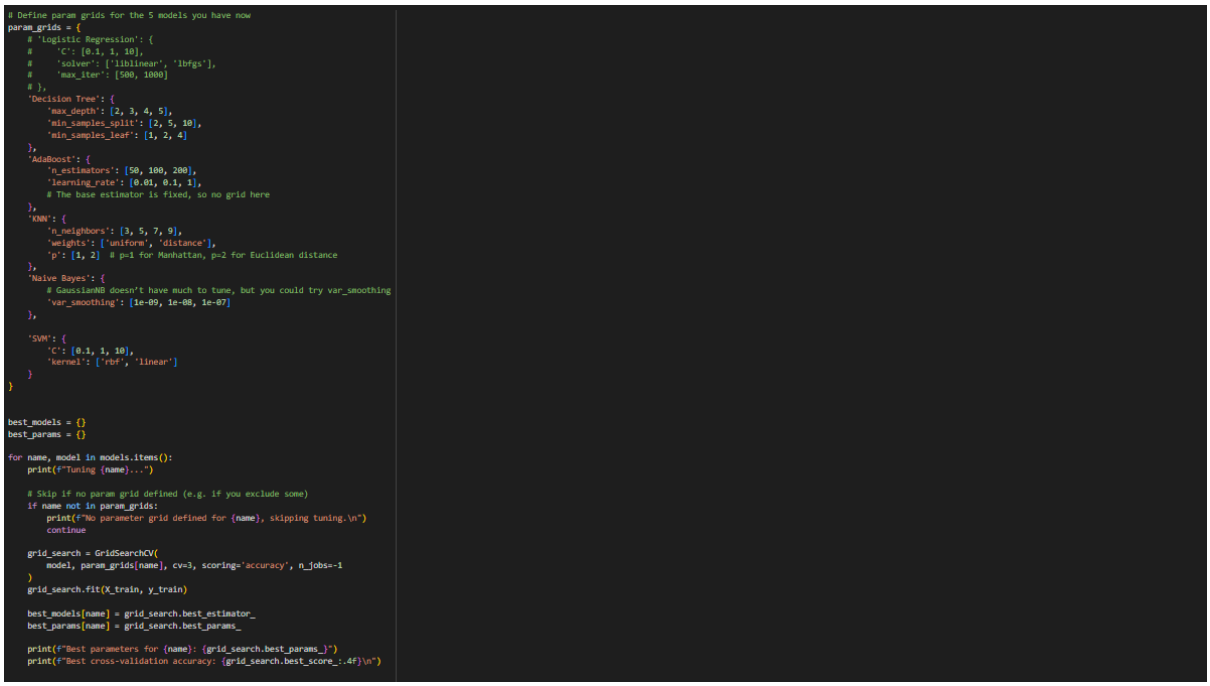
1. Ensure your virtual environment is activated.
2. Launch Jupyter Lab from your terminal:

jupyter lab

3. A new tab will open in your web browser with the Jupyter Lab interface.
4. Navigate to and open notebook.ipynb.
5. To train the model and generate the final artifacts, you can either:

- Run all cells sequentially by clicking Run > Run All Cells.
- Scroll down to the final cell block titled "**Retraiining the model**" and run this block and the subsequent cells. This specific block is designed to produce the final, production-ready model and save all necessary artifacts to the saved_models/ directory.

Upon successful execution, the saved_models directory will be populated with the latest versions of the .pkl files.



```
# Define param grids for the 5 models you have now
param_grids = {
    # 'Logistic Regression': {
    #     'C': [0.1, 1, 10],
    #     'solver': ['liblinear', 'lbfgs'],
    #     'max_iter': [500, 1000]
    # },
    'Decision Tree': {
        'max_depth': [2, 3, 4, 5],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    },
    'AdaBoost': {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 1],
        # The base estimator is fixed, so no grid here
    },
    'KNN': {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance'],
        'p': [1, 2]  # p=1 for Manhattan, p=2 for Euclidean distance
    },
    'Naive Bayes': {
        # GaussianNB doesn't have much to tune, but you could try var_smoothing
        'var_smoothing': [1e-09, 1e-08, 1e-07]
    },

    'SVM': {
        'C': [0.1, 1, 10],
        'kernel': ['rbf', 'linear']
    }
}


best_models = {}
best_params = {}

for name, model in models.items():
    print(f"Tuning {name}...")

    # Skip if no param grid defined (e.g. if you exclude some)
    if name not in param_grids:
        print(f"No parameter grid defined for {name}, skipping tuning.\n")
        continue

    grid_search = GridSearchCV(
        model, param_grids[name], cv=3, scoring='accuracy', n_jobs=-1
    )
    grid_search.fit(X_train, y_train)

    best_models[name] = grid_search.best_estimator_
    best_params[name] = grid_search.best_params_

    print(f"Best parameters for {name}: {grid_search.best_params_}")
    print(f"Best cross-validation accuracy: {grid_search.best_score_:.4f}\n")
```

Figure 2: Models Training

6 Using the Trained Model for Prediction (Inference)

Once the model and preprocessing artifacts are saved, you can use them to make predictions on new, unseen data. Prediction can be checked through below code also without web app.

```
sample_input = X_test.iloc[:5]
sample_scaled = scaler.transform(sample_input)
sample_pca = pca.transform(sample_scaled)

sample_preds = model.predict(sample_pca)
sample_probs = model.predict_proba(sample_pca)


# Define colors
COLOR_RED = "\033[91m"
COLOR_YELLOW = "\033[93m"
COLOR_GREEN = "\033[92m"
COLOR_RESET = "\033[0m"

risk_color_map = {
   "High Risk": COLOR_RED,
```

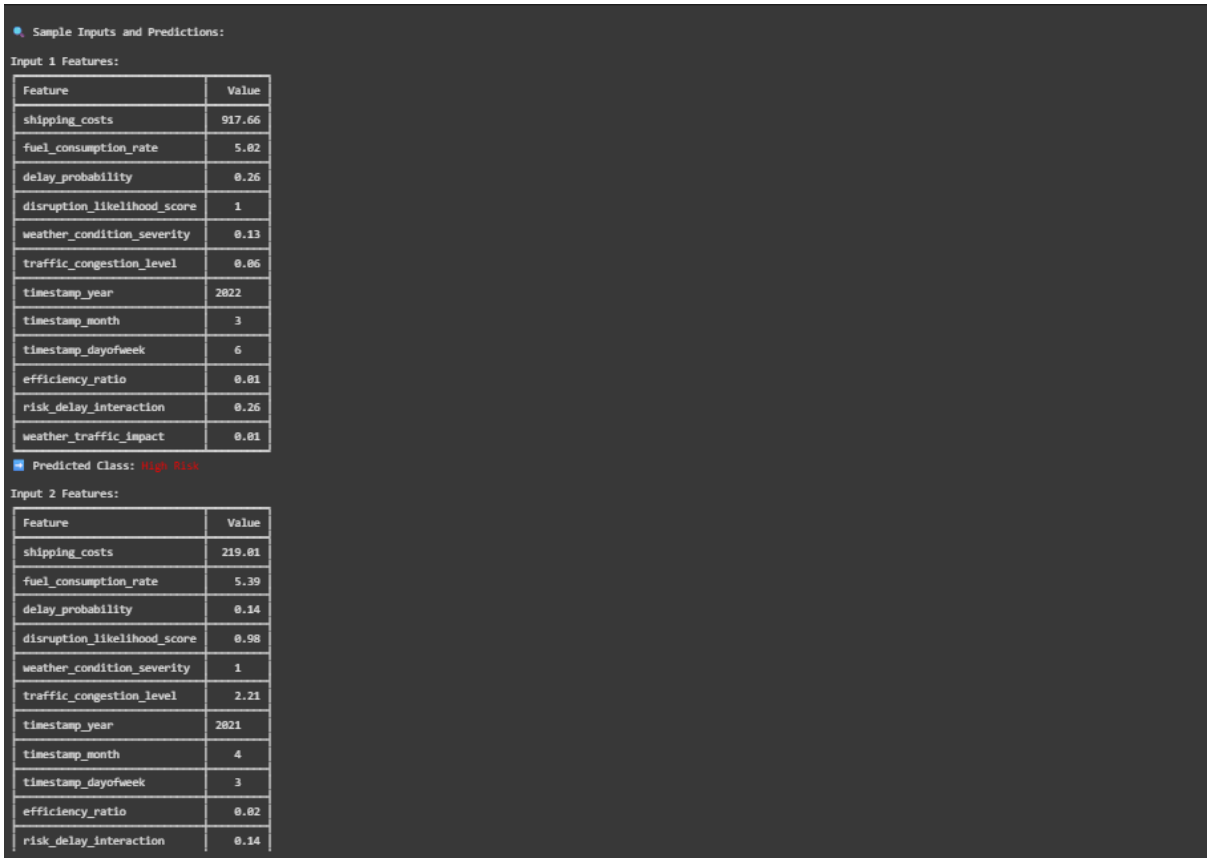
```
    "Moderate Risk": COLOR_YELLOW,
    "Low Risk": COLOR_GREEN
}

columns_to_hide = ['order_fulfillment_status', 'cargo_condition_status']
int_columns = ['timestamp_year', 'timestamp_month', 'timestamp_dayofweek']

print("\n🔍 Sample Inputs and Predictions:")
for i in range(len(sample_input)):
    row = sample_input.iloc[i].drop(columns_to_hide).copy()
    row_dict = {}
    for col in row.index:
        if col in int_columns:
            row_dict[col] = int(round(row[col]))
        else:
            row_dict[col] = round(row[col], 2)

    table = [[k, v] for k, v in row_dict.items()]
    print(f"\nInput {i+1} Features:")
    print(tabulate(table, headers=["Feature", "Value"], tablefmt="fancy_grid"))

    color = risk_color_map.get(sample_preds[i], COLOR_RESET)
    print(f"→ Predicted Class: {color}{sample_preds[i]}{COLOR_RESET}")
```



🔍 Sample Inputs and Predictions:

Input 1 Features:

Feature	Value
shipping_costs	917.66
fuel_consumption_rate	5.02
delay_probability	0.26
disruption_likelihood_score	1
weather_condition_severity	0.13
traffic_congestion_level	0.06
timestamp_year	2022
timestamp_month	3
timestamp_dayofweek	6
efficiency_ratio	0.01
risk_delay_interaction	0.26
weather_traffic_impact	0.01

→ Predicted Class: High Risk

Input 2 Features:

Feature	Value
shipping_costs	219.01
fuel_consumption_rate	5.39
delay_probability	0.14
disruption_likelihood_score	0.98
weather_condition_severity	1
traffic_congestion_level	2.21
timestamp_year	2021
timestamp_month	4
timestamp_dayofweek	3
efficiency_ratio	0.02
risk_delay_interaction	0.14

Figure 3: Output from Above Command

7 Conclusion

This configuration manual has provided a comprehensive, step-by-step guide to setting up and operating the predictive logistics risk model. By following the outlined procedures, a user can successfully create the necessary software environment, install all dependencies, and execute the model training pipeline to generate the required predictive artifacts.
Furthermore, the manual provides a ready-to-use inference script (predict.py) that demonstrates how to leverage the trained model to assess the risk of new, unseen shipments. This serves as a template for integrating the model into larger applications, such as a real-time monitoring dashboard or an automated alert system. The successful completion of these steps enables the practical application of this research, empowering organisations to transition towards a more proactive, data-driven, and resilient supply chain strategy. For any issues, first ensure all library versions match those in requirements.txt and that all file paths are correct relative to the script being executed.