Configuration Manual

MSc Research Project
Programme Name

Sadhik Shaik
Student ID: x23213108

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sadhik Shaik ……. …………………………………………………………………………………………………
Student ID:	X23213108 ………………………………………………………………………………………………….……
Programme:	MSCDAD_A_JAN24 ……………………………………………………………… **Year:** 2024 …………………………..
Module:	MSc Research Project ………………………………………………………………………………………………….………
Lecturer:	Vladimir Milosavljevic ………………………………………………………………………………………………….………
Submission Due Date:	12th December 2024 ………………………………………………………………………………………………….………
Project Title:	Assessing Irish Banking Stocks through Time-Series Forecasting and Quantitative Trading Models ………………………………………………………………………………………………….………
Word Count:	600 ……………………………………… **Page Count:** 11 ……………………………….…….………

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:	Sadhik Shaik …………………………………………………………………………………………………
Date:	12/12/2024 …………………………………………………………………………………………………

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

Sadhik Shaik
Student ID: x23213108

1 System Requirements

1.1 Hardware Requirements:

- Processor: Intel Core i5 or higher (or equivalent AMD Ryzen)
- Memory: 8 GB RAM minimum (16 GB recommended)
- Storage: 20 GB free disk space
- GPU: NVIDIA GPU with CUDA support for LSTM training (e.g., NVIDIA GTX 1050 or higher)

1.2 Software Requirements:

- Operating System: Windows 10, macOS, or Linux
- Programming Environment: Python 3.8 or higher
- Required Libraries:
  - NumPy
  - Pandas
  - Matplotlib
  - Seaborn
  - TensorFlow/Keras (for LSTM and Hybrid ARIMA-LSTM)
  - Statsmodels (for ARIMA and SARIMA)
  - Scikit-learn (for Linear Regression)
  - Jupyter Notebook
- Cloud Resources (optional): Google Colab or similar for GPU-based training
  - Import following librares

```python
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import warnings
warnings.filterwarnings('ignore')
```

2 Dataset and Preprocessing

2.1 Dataset Sources:

- Historical stock price data for AIB and BOI banks collected from their official websites.
- Supplementary macroeconomic indicators sourced from publicly available repositories like the Central Statistics Office Ireland.

2.2 Preprocessing Workflow:

1. **Data Cleaning**:

    - Handling missing values (imputation or removal).
    - Removing outliers using statistical thresholds.

```python
def clean_data(df, name):
    """
    Clean the dataframe by handling missing values and outliers
    """
    print(f"\nCleaning data for {name}:")

    # Store original shape
    original_shape = df.shape

    # Check for missing values before cleaning
    print("\nMissing values before cleaning:")
    print(df.isnull().sum())

    # Forward fill missing values in Close, Open, High, Low prices
    price_columns = ['Close', 'Open', 'High', 'Low', 'Adj Close']
    df[price_columns] = df[price_columns].fillna(method='ffill')

    # For Volume, fill NaN with the median value
    df['Volume'] = df['Volume'].fillna(df['Volume'].median())

    # For economic indicators, forward fill
    economic_indicators = ['unemployment_rate', 'exchange_Rate', 'Interest_Rate']
    df[economic_indicators] = df[economic_indicators].fillna(method='ffill')

    # Backward fill any remaining NaN values at the start of the series
    df = df.fillna(method='bfill')

    # Check for any remaining missing values
    print("\nMissing values after cleaning:")
    print(df.isnull().sum())

    # Report on the cleaning results
    final_shape = df.shape
    print(f"\nRows before cleaning: {original_shape[0]}")
    print(f"Rows after cleaning: {final_shape[0]}")

    return df
```

2. **Data Transformation**:

- ❖ Log scaling to stabilize variance.
- ❖ Differencing to achieve stationarity (for ARIMA/SARIMA).

3. **Feature Engineering**:

- ❖ Creating lagged variables, rolling averages, and Bollinger Bands.
- ❖ Normalizing features using Min-Max scaling for LSTM compatibility.

```python
def engineer_features(df):
    """
    Create technical indicators and features for the stock data

    """
    # Create copy to avoid modifying original data
    df_features = df.copy()

    # Technical Indicators
    # 1. Moving Averages
    df_features['MA5'] = df_features['Close'].rolling(window=5).mean()
    df_features['MA20'] = df_features['Close'].rolling(window=20).mean()
    df_features['MA50'] = df_features['Close'].rolling(window=50).mean()

    # 2. Relative Strength Index (RSI)
    delta = df_features['Close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
    rs = gain / loss
    df_features['RSI'] = 100 - (100 / (1 + rs))

    # 3. Bollinger Bands
    df_features['BB_middle'] = df_features['Close'].rolling(window=20).mean()
    df_features['BB_upper'] = df_features['BB_middle'] + 2*df_features['Close'].rolling(window=20).std()
    df_features['BB_lower'] = df_features['BB_middle'] - 2*df_features['Close'].rolling(window=20).std()

    # 4. Price momentum
    df_features['momentum'] = df_features['Close'] - df_features['Close'].shift(4)

    # 5. Volatility
    df_features['volatility'] = df_features['Close'].rolling(window=10).std()

    # 6. Price changes
    df_features['price_change'] = df_features['Close'].pct_change()
    df_features['price_change_5d'] = df_features['Close'].pct_change(periods=5)

    # 7. Volume features
    df_features['volume_ma5'] = df_features['Volume'].rolling(window=5).mean()
    df_features['volume_change'] = df_features['Volume'].pct_change()

    # Economic Indicators - lagged effects
    df_features['unemployment_change'] = df_features['unemployment_rate'].pct_change()
```

4. **Data Splitting**:

- Training data: Historical data up to May 2024.
- Testing data: Data from June 2024 onward.

```
def split_data(df):
    """
    Split data into training and testing sets
    """
    # Convert the index to datetime if it's not already
    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)

    train = df[:'2024-05-31']
    test = df['2024-06-01':]
    return train, test

aib_train, aib_test = split_data(aib_df)
boi_train, boi_test = split_data(boi_df)
```

```
[ ] # Engineered Data
    aib_train_eng, aib_test_eng = split_data(aib_features)
    boi_train_eng, boi_test_eng = split_data(boi_features)
```

3 Model Configuration

3.1 Linear Regression Configuration:

- Parameters: fit_intercept=True, positive=True
- Evaluation Metrics: RMSE, R^2, MAE

```python
def build_linear_regression(df_train, df_test):
    """
    Build and evaluate linear regression model with R-squared and RMSE metrics
    """
    # Prepare features
    features = ['unemployment_rate', 'exchange_Rate', 'Interest_Rate']
    X_train = df_train[features]
    y_train = df_train['Close']
    X_test = df_test[features]
    y_test = df_test['Close']

    best_params = {
    'copy_X': True,
    'fit_intercept': True,
    'n_jobs': -1,
    'positive': True
}
    # Train model
    model = LinearRegression(**best_params)
    model.fit(X_train, y_train)

    # Make predictions
    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    # Calculate metrics
    train_rmse = np.sqrt(mean_squared_error(y_train, train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, test_pred))

    # Calculate R-squared scores
    train_r2 = r2_score(y_train, train_pred)
    test_r2 = r2_score(y_test, test_pred)

    print("Linear Regression Results:")
    print(f"Training RMSE: {train_rmse:.2f}")
    print(f"Testing RMSE: {test_rmse:.2f}")
    print(f"Training R²: {train_r2:.4f}")
    print(f"Testing R²: {test_r2:.4f}")
    print("\nFeature Coefficients:")
    for feature, coef in zip(features, model.coef_):
        print(f"{feature}: {coef:.4f}")
```

ARIMA/SARIMA Configuration:

- Best Order for ARIMA (AIB): (3, 1, 5)
- Seasonal Component for SARIMA (if applicable): (p, d, q, m)
- Tuning Criteria: AIC, BIC

```python
def tune_time_series_models(train_data):
    """
    Tune ARIMA and SARIMA models
    """
    from pmdarima import auto_arima

    # Tune ARIMA
    print("\nTuning ARIMA parameters...")
    arima_model = auto_arima(
        train_data,
        start_p=0, start_q=0, max_p=3, max_q=3, m=1,
        start_P=0, seasonal=False, d=1, D=1, trace=True,
        error_action='ignore', suppress_warnings=True, stepwise=True
    )

    print("\nBest ARIMA parameters:", arima_model.order)

    # Tune SARIMA
    print("\nTuning SARIMA parameters...")
    sarima_model = auto_arima(
        train_data,
        start_p=0, start_q=0, max_p=2, max_q=2, m=12,
        start_P=0, seasonal=True, d=1, D=1, trace=True,
        error_action='ignore', suppress_warnings=True, stepwise=True
    )

    print("\nBest SARIMA parameters:", sarima_model.order)

    return arima_model.order, sarima_model.order
```

LSTM Configuration:

- Architecture: 2 LSTM layers with 128 units each, followed by a dense output layer.
- Optimizer: Adam with a learning rate of 0.001
- Epochs: 50
- Batch Size: 32
- Dropout: 0.2

```
# Build LSTM model with correct input shape
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(60, 2)),
    LSTM(50),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')

# Train model
history = model.fit(X_train, y_train,
                   epochs=10,
                   batch_size=32,
                   validation_split=0.1,
                   verbose=1)

# Generate predictions
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)

# Inverse transform predictions
# Create dummy array with same shape as input data for inverse transform
train_dummy = np.zeros((len(train_pred), 2))
train_dummy[:, 0] = train_pred.flatten()
train_pred_transformed = scaler.inverse_transform(train_dummy)[:, 0]

test_dummy = np.zeros((len(test_pred), 2))
test_dummy[:, 0] = test_pred.flatten()
test_pred_transformed = scaler.inverse_transform(test_dummy)[:, 0]

return train_pred_transformed, test_pred_transformed, history
```

Hybrid ARIMA-LSTM Configuration:

- Workflow:
    1. ARIMA model trained to predict linear components.
    2. Residuals passed to LSTM for modeling non-linear dependencies.
- Combined predictions integrated into a final output.

```python
# Build hybrid LSTM models for each banks
def run_hybrid_lstm_analysis(bank_name, train, test, arima_train_pred, arima_test_pred):
    """
    Run hybrid LSTM analysis for each bank
    """
    print(f"\nHybrid LSTM Results for {bank_name}:")

    # Build and train the hybrid model
    train_pred, test_pred, history = build_hybrid_lstm(
        train, test, arima_train_pred, arima_test_pred)

    # Calculate metrics
    train_rmse = np.sqrt(mean_squared_error(
        train['Close'].iloc[60:], train_pred))
    test_rmse = np.sqrt(mean_squared_error(
        test['Close'].iloc[60:], test_pred))

    print(f"Training RMSE: {train_rmse:.2f}")
    print(f"Testing RMSE: {test_rmse:.2f}")

    # Plot training history
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{bank_name} Hybrid LSTM Training History')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

    return train_pred, test_pred, history, train_rmse, test_rmse
```