Configuration Manual

MSc Research Project
Data Analytics

Pradeep Kumar Reddy Bathula
Student ID: X22165801

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Pradeep Kumar Reddy Bathula
Student ID:	X22165801
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Aaloka Anant
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	535
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:	
Date:	31st January 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

Pradeep Kumar Reddy Bathula
X22165801

1 Introduction

The system setup, data preparation, modeling, assessment, and Explainable AI methodologies used in the analysis are described in this paper

2 System Configuration

This Section on System Configuration provides full details on all the hardware and software utilized in this project.

2.1 Hardware Specifications

The device utilized for the project has the following hardware configuration.

Figure 1: Design Specification

2.2 Software Environment

Jupyter Notebook was utilized as the program for this project. Anaconda Navigator was used to launch this software. This web-interactive, open-source coding tool is simple to use. The Anaconda navigator is seen in Figure 2.

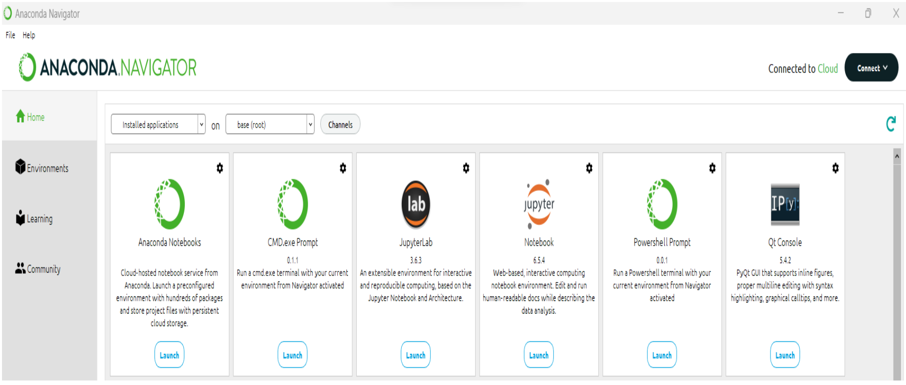

Figure 2: Anaconda Launcher

3 Data Preparation

3.1 Data Selection

Four separate datasets in CSV format from the Federal Reserve Bank of St. Louis used in this study(FRED; 2022).

3.2 Importing Libraries

importing necessary libraries that are needed for preprocessing and data import as shown in Figure 3.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Figure 3: Importing Libraries

3.3 Importing Data

Required Datasets are downlanded and imported as shown in fig 4.

3.4 Data Preprocessing

3.4.1 Merging Dataset

once the datasets have been imported. In order to compute in the future, those datasets must be combined as shown in Figure 5.

```
df1 = pd.read_csv('PCE.csv')
df2 = pd.read_csv('PPIACO (1).csv')
df3 = pd.read_csv('CPIAUCSL.csv')
df4 = pd.read_csv('USALORSGPNOSTSAM.csv')
```

Figure 4: Importing Libraries

```
merged_df = pd.merge(df1, df2, on='DATE')
merged_df = pd.merge(merged_df, df3, on='DATE')
merged_df = pd.merge(merged_df, df4, on='DATE')


merged_df.to_csv('merged_data.csv', index=False)

df = pd.read_csv('merged_data.csv')
```

Figure 5: Merging Data

3.4.2 Null Values

Examining the dataset for null values.

```
nan_count = df['PCE'].isna().sum()
print("Number of NaN values in 'PCE' column:", nan_count)
nan_count = df['PPIACO'].isna().sum()
print("Number of NaN values in 'PPI' column:", nan_count)
nan_count = df['CPIAUCSL'].isna().sum()
print("Number of NaN values in 'CPI' column:", nan_count)
nan_count = df['USALORSGPNOSTSAM'].isna().sum()
print("Number of NaN values in 'GDP' column:", nan_count)
```

Figure 6: Checking for Null value

4 Data Modeling

4.1 LSTM

4.1.1 Model Architecture

A sequential neural network model with two LSTM layers, a dropout layer, and a dense layer is defined by the code, as seen in Fig.7.

```python
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(look_back, df.shape[1])))
model.add(LSTM(50))
model.add(Dropout(0.2))
model.add(Dense(df.shape[1]))
```

Figure 7: LSTM Model Architecture

4.1.2 Model Fitting

Figure 8's code creates a neural network model, configures early stopping and model checkpoint callbacks, and uses validation data to train the model.

```python
model.compile(optimizer='adam', loss='mean_squared_error')

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

# Training the model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.1,
    callbacks=[early_stopping, model_checkpoint],
    verbose=1
)
```

Figure 8: LSTM Model Fitting

4.2 Transformers

4.2.1 Model Architecture

A Transformer-based neural network model with adjustable parameters and multi-layer perceptron (MLP) components for sequence data is defined using this code, as seen in Figure 9

```python
def TransformerModel(input_shape, head_size, num_heads, ff_dim, num_transformer_blocks, mlp_units, dropout=0, mlp_dropout=0):
    inputs = tf.keras.Input(shape=input_shape)
    x = inputs
    for _ in range(num_transformer_blocks):
        x1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(x)
        x2 = tf.keras.layers.MultiHeadAttention(key_dim=head_size, num_heads=num_heads, dropout=dropout)(x1, x1)
        x = tf.keras.layers.Add()([x, x2])
        x1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(x)
        x2 = tf.keras.layers.Dense(ff_dim, activation="relu")(x1)
        x2 = tf.keras.layers.Dropout(mlp_dropout)(x2)
        x2 = tf.keras.layers.Dense(input_shape[-1])(x2)
        x = tf.keras.layers.Add()([x, x2])
    x = tf.keras.layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = tf.keras.layers.Dense(dim, activation="relu")(x)
        x = tf.keras.layers.Dropout(mlp_dropout)(x)
    outputs = tf.keras.layers.Dense(y_train.shape[1])(x)  # Adjusted for four variables
    return tf.keras.Model(inputs, outputs)
```

Figure 9: Transformers Model Architecture

4.2.2 Model Fitting

Figure 10 illustrates how this method uses the mean squared error loss and Adam optimizer to build and train a Transformer-based neural network model for sequence data with given parameters.

```
model = TransformerModel(input_shape, head_size, num_heads, ff_dim, num_transformer_blocks, mlp_units, dropout, mlp_dropout)

model.compile(loss="mean_squared_error", optimizer=tf.keras.optimizers.Adam(learning_rate=0.001))

num_epochs = 50
batch_size = 32
history = model.fit(X_train, y_train, epochs=num_epochs, batch_size=batch_size, validation_split=0.2)
```

Figure 10: Transformers Model Fitting

4.3 NBEATS

4.3.1 Model Architecture

This code, which is displayed in Figure 11, describes an N-BEATS block neural network design for time series forecasting applications that has adjustable parameters.

```
def nbeats_block(input_shape, theta_size, horizon, n_neurons=512, n_layers=4, block_number=0):
    input_layer = Input(shape=input_shape)
    x = Flatten()(input_layer)

    for _ in range(n_layers):
        x = Dense(n_neurons, activation='relu')(x)

    theta_b = Dense(theta_size, activation='linear')(x)
    theta_f = Dense(horizon, activation='linear')(x)

    backcast = Dense(np.prod(input_shape), activation='linear')(theta_b)
    backcast_reshape = Reshape(input_shape)(backcast)

    forecast = Dense(horizon, activation='linear')(theta_f)

    block_name = f'nbeats_block_{block_number}'
    block = Model(inputs=input_layer, outputs=[backcast_reshape, forecast], name=block_name)

    return block
```

Figure 11: NBEATS Model Architecture

4.3.2 Model Fitting

Using the mean squared error loss and Adam optimizer, this algorithm builds an N-BEATS neural network model by stacking several N-BEATS blocks. It is then trained for time series forecasting, as seen in Figure 12.

5 Evaluation

As evaluation metrics, this code, as shown in fig. 13, computes and outputs the RMSE, MAE, and MAPE scores for each feature in the dataset.

```
input_layer = Input(shape=input_shape)
cumulative_backcast = input_layer
cumulative_forecast = None

for i in range(n_blocks):
    backcast, forecast = nbeats_block(input_shape, theta_size, horizon, n_neurons, n_layers, i)(cumulative_backcast)
    cumulative_backcast = Subtract()([cumulative_backcast, backcast])
    if cumulative_forecast is None:
        cumulative_forecast = forecast
    else:
        cumulative_forecast = Add()([cumulative_forecast, forecast])

model = Model(inputs=input_layer, outputs=cumulative_forecast)
model.compile(optimizer=Adam(), loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1, verbose=1)
```

Figure 12: NBEATS Model Fitting

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
import matplotlib.pyplot as plt
rmse_scores = {feature: np.sqrt(mean_squared_error(y_test_rescaled[:, i], y_pred_rescaled[:, i]))
               for i, feature in enumerate(df.columns)}
mae_scores = {feature: mean_absolute_error(y_test_rescaled[:, i], y_pred_rescaled[:, i])
              for i, feature in enumerate(df.columns)}
mape_scores = {feature: np.mean(np.abs((y_test_rescaled[:, i] - y_pred_rescaled[:, i]) / y_test_rescaled[:, i])) * 100
               for i, feature in enumerate(df.columns)}
print("Evaluation Metrics:")
for feature in df.columns:
    print(f"Feature: {feature}")
    print(f"RMSE: {rmse_scores[feature]}")
    print(f"MAE: {mae_scores[feature]}")
    print(f"MAPE: {mape_scores[feature]}%")
    print()
```

Figure 13: Evaluation

6 Explainable AI

As seen in Figure 14, this code restructures the input data, generates a SHAP explanation, calculates SHAP values, and outputs the mean SHAP values for every feature.

```
import shap

def model_predict(data_2d):
    data_3d = data_2d.reshape(-1, X_train.shape[1], X_train.shape[2])
    return model.predict(data_3d)

X_train_reshaped = X_train[:100].reshape(-1, X_train.shape[2] * X_train.shape[1])
X_test_reshaped = X_test[:100].reshape(-1, X_test.shape[2] * X_test.shape[1])


explainer = shap.KernelExplainer(model_predict, X_train_reshaped)
shap_values = explainer.shap_values(X_test_reshaped)

shap_sum = np.abs(shap_values).mean(axis=0)
feature_names = ['PCE', 'PPI', 'CPI', 'GDP']
for i in range(len(feature_names)):
    feature_shap_values = shap_sum[i::len(feature_names)]
    print(f"Mean SHAP value for {feature_names[i]}: {np.mean(feature_shap_values)}")
```

Figure 14: Explainable AI

References

FRED, S. L. (2022). Welcome to fred, your trusted source for economic data since 1991. **URL:** *https://fred.stlouisfed.org/*