Configuration Manual

MSc Research Project
Programme Name

Nithish Christopher
Student ID: X23116099

School of Computing
National College of Ireland

Supervisor: **Anu Sahni**

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nithish Christopher		
Student ID:	X23116099		
Programme:	MSc Data Analytics	**Year:**	2024
Module:	MSc Research Project		
Lecturer:	Anu Sahni		
Submission Due Date:	12.12.2024		
Project Title:	Predicting Ireland House Prices with Deep Learning - A Comparative Study		
Word Count:	1036 **Page Count:** 10		

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:	Nithish Christopher
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

Nithish Christopher
x23116099

1 Introduction

This configuration manual provides a detail guidelines of implementation of the research "Predicting Ireland House Prices with Deep Learning - A Comparative Study". The major research objective is using of advanced deep learning techniques to predict the house prices.
So, in this research used Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Long Short-Term Memory Networks (LSTM) techniques. This configuration manual report contains the detail of hardware and software specifications in detail, used libraries and used to implement the research.

2 Hardware Specification

Operating System	Windows 11 Home
Processor	11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.42 GHz
RAM	16.0 GB (15.8 GB usable)
System Type	64-bit operating system, x64-based processor

3 Software Specification

Programming Language	Python Version: 3.8.20
Tool	Jupyter Notebook

3.1 Python and Jupyter Notebook Setup with libraries

In this research was completely coded in the Python as primary language and Worked on Jupyter notebook environment and version also mentioned in table 2. Used of python on jupter notebook to completed this research.
Further process of research imported necessary libraries based on model implementation, applying preprocessing and deep learning techniques. Python have own libraries to use all visualization in EDA. Figure 1 mentioning about imported libraries list.

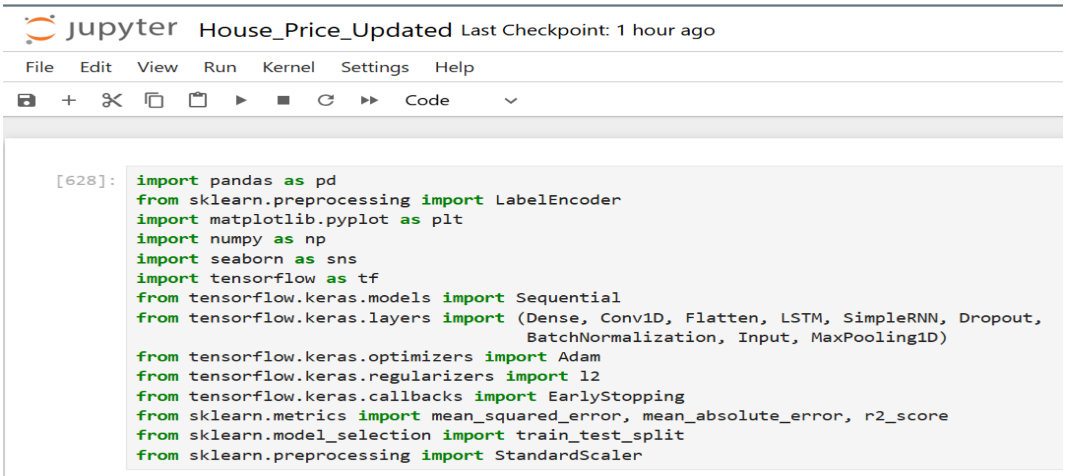

Figure 1: Installed Required Libraries

3.2 Importing data

In Jupyter notebook home page can upload dataset, in this research used data uploaded in this page and Using of pandas to read the CSV house price dataset. Those details mentioned in figure 2.

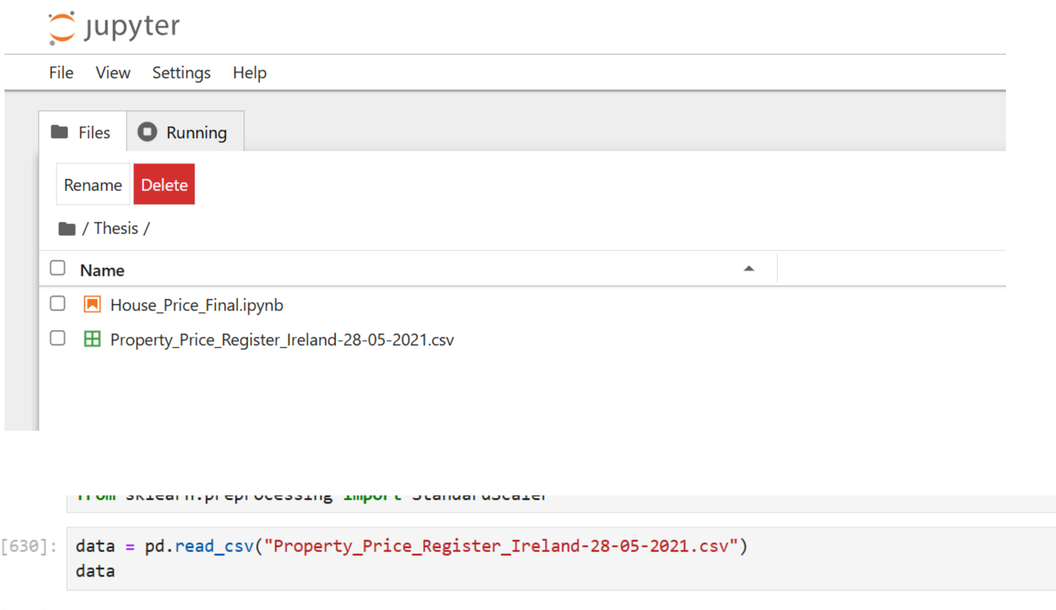

Figure 2: Dataset loading

3.3 Data preprocessing and Data cleaning

In this research gone through very detailed manner in preprocessing and data cleaning. So as mentioned figure 3 and figure 4 worked on basic cleaning and for balancing the data to clear more outliers as well.

```
[634]: data.isna().sum()

[634]: SALE DATE                    0

656]: data.duplicated().sum()

656]: 42204

658]:
      data = data.drop_duplicates()

36]: data.drop(columns=['POSTAL_CODE','PROPERTY_SIZE_DESC','ADDRESS'],inplace=True)
     data

36]:      SALE DATE    COUNTY   SALE PRICE   IF MARKET PRICE   IF VAT EXCLUDED        PROPERTY DESC
```

Figure 3: Basic of data cleaning

```
print("Before filtering outliers:")
print(f"Number of rows: {len(data)}")
Q1 = data['SALE_PRICE'].quantile(0.25)
Q3 = data['SALE_PRICE'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

data = data[(data['SALE_PRICE'] >= lower_bound) &
            (data['SALE_PRICE'] <= upper_bound)]

# After filtering
print("\nAfter filtering outliers:")
print(f"Number of rows: {len(data)}")

    Number of outliers in SALE_PRICE: 5340

[708]: total_rows = len(data)
       data_no_outliers = data[(data['SALE_PRICE'] >= lower_bound) &
                               (data['SALE_PRICE'] <= upper_bound)]
       rows_after_removal = len(data_no_outliers)
       print(f"Total rows: {total_rows}")
       print(f"Rows after removing outliers: {rows_after_removal}")
       print(f"Number of outliers removed: {total_rows - rows_after_removal}")

       Total rows: 409969
```

Figure 4: Outliers removal

Balancing the data to implemented important techniques from feature engineering to create some columns for more evaluation and detailed preprocessing. Figure 5 explaining for create differencing of prices in past year to created new column using of label coding concept. At the same time data monthly based separated as season. In more detail to balancing data into created county, property description is into some frequencies concepts to balancing the data.

```
[674]: data = data.copy()
```

```
[676]: data['PRICE_CATEGORY'] = pd.cut(data['SALE_PRICE'], bins=bins, labels=labels)
```

```
[678]: bins = [0, 100000, 500000, 1000000, 5000000, np.inf]
labels = ['Low', 'Moderate', 'High', 'Very High', 'Luxury']
data['PRICE_CATEGORY'] = pd.cut(data['SALE_PRICE'], bins=bins, labels=labels)
```

```
[680]:
data['SEASON'] = data['MONTH'].apply(lambda x: 'Winter' if x in [12, 1, 2]
                                    else 'Spring' if x in [3, 4, 5]
                                    else 'Summer' if x in [6, 7, 8]
                                    else 'Fall')

# Convert days to weekends
data['IS_WEEKEND'] = data['DATE'].apply(lambda x: 1 if x >= 5 else 0)
```

```
38]: county_freq = data['COUNTY'].value_counts()
data['COUNTY_FREQ'] = data['COUNTY'].map(county_freq)
```

```
90]: county_mean_price = data.groupby('COUNTY')['SALE_PRICE'].mean()
data['COUNTY_MEAN_PRICE'] = data['COUNTY'].map(county_mean_price)
```

```
92]: data = pd.get_dummies(data, columns=['PROPERTY_DESC'], drop_first=True)
```

```
94]: property_mean_price = data.groupby('PROPERTY_DESC_Second-Hand Dwelling apartment')['SALE_PRICE'].mean()
data['PROPERTY_DESC_MEAN_PRICE'] = data['PROPERTY_DESC_Second-Hand Dwelling apartment'].map(property_mean_price)
```

```
96]:
data['PRICE_MARKET_RATIO'] = data['SALE_PRICE'] / (data['IF_MARKET_PRICE'] + 1)
data['VAT_EFFECT_ON_PRICE'] = data['SALE_PRICE'] * data['IF_VAT_EXCLUDED']
```

```
98]:
data['PRICE_MARKET_RATIO'] = data['SALE_PRICE'] / (data['IF_MARKET_PRICE'] + 1)
data['VAT_EFFECT_ON_PRICE'] = data['SALE_PRICE'] * data['IF_VAT_EXCLUDED']
```

Figure 5: Feature engineering

3.4 Exploratory Data Analysis

One of the Important parts of this research is EDA, using of impelled libraries to performed detailed overview of data through visuals. As mentioned in figure 6 those visuals are created for more understanding of pre-processed data. These visuals are explaining of trends of past year in price, seasonal trends, relationship between variables, those analysis are more helpful for this detailed research.

```
[710]: sns.boxplot(data, x='SALE_PRICE', color='purple')
plt.title('Boxplot of SALE_PRICE')
plt.show()
```

Boxplot of SALE_PRICE

Line Plot of Sale Price Over Year(2010 - 2021)

```
[714]: yearly_data = data.groupby('YEAR')['SALE_PRICE'].mean()
plt.figure(figsize=(12, 6))
plt.plot(yearly_data.index, yearly_data.values, marker='o', label='Average Sale Price')
plt.title('Sale Price Trends (2010-2021)')
plt.xlabel('Year')
plt.ylabel('Average Sale Price')
plt.xticks(ticks=yearly_data.index)
plt.legend()
plt.grid()
plt.show()
```

Seasonal Trends

```
[720]: monthly_avg = data.groupby(['YEAR', 'MONTH'])['SALE_PRICE'].mean().unstack()
monthly_avg.plot(figsize=(12, 6), cmap='viridis')
plt.title('Monthly Average Sale Prices (2010-2021)')
plt.xlabel('Month')
plt.ylabel('Average Sale Price')
plt.legend(title='Year')
plt.show()
```

Correlation Analysis

```
[722]: numerical_cols = data.select_dtypes(include=['float64', 'int64', 'int32'])
correlation_matrix = numerical_cols.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

Property Description Analysis

```
[726]: property_avg = data.groupby('PRICE_CATEGORY')['SALE_PRICE'].mean().sort_values()
plt.figure(figsize=(12, 6))
property_avg.plot(kind='barh', color='coral')
plt.title('Average Sale Price by Property Description')
plt.xlabel('Average Sale Price')
plt.ylabel('Property Description')
plt.show()
```

Figure 6: Visuals of detailed EDA

3.5 Modelling Training and Evaluations

Implementing of advanced deep learning to begin with splitting of data into test and train. Figure 7 explaining based on target variable to splitting the data and validation. And also for normalize the data to standard scalar method applied.

Split the data into Test and Train

```
[518]: ### Here spliting tartget variable of sale price column and also checked with scaler transform as well.

[730]: X = data.drop(['SALE_PRICE'], axis=1)
y = data['SALE_PRICE']
X = pd.get_dummies(X, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train_rnn = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_rnn = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Figure 7 : Splitting of data

3.6 Implementation of Deep learning Techniques

In this complete research focus on implementing models. So, based on hyperparameter concepts to implemented all the models. Here ANN, RNN, LSTM, CNN models are performed and those are performances are good as well. Used of hyperparameter to increased performance of training models in all four models.

```
[732]: model_ann = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],), kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(32, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dense(1)
])

model_ann.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

history_ann = model_ann.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=150,
    batch_size=64,
    callbacks=[early_stopping],
    verbose=1
```

```
[734]: y_pred_ann = model_ann.predict(X_test)
r2_ann = r2_score(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
print(f"MSE: {mse_ann}")
print(f"RMSE: {rmse_ann}")
print(f"MAE: {mae_ann}")
print(f"R²: {r2_ann}")

3844/3844 [==============================] - 13s 3ms/step
MSE: 6906881.083665769
RMSE: 2628.094572816163
MAE: 1847.469747542503
```

Figure 8 : Implementing and evaluation of ANN

Figure 8,9,10,11 showing of implementing all models in detailed manner and performing advanced techniques like using of layer configuration set up, dropouts, epochs range and early stopping. And also showing evaluation metrics of each model. Tthese all implemented models are discussed based on MSE, RMSE, MAE metrics.

RNN MODEL

```
738]: model_rnn = Sequential([
    SimpleRNN(128, activation='relu', input_shape=(X_train_rnn.shape[1], 1)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)
])

model_rnn.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
history_rnn = model_rnn.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=150,
    batch_size=64,
    callbacks=[early_stopping],
    verbose=1
)

Epoch 1/150
```

```
•[420]: y_pred_rnn = model_rnn.predict(X_test)
r2_rnn = r2_score(y_test, y_pred_rnn)
mse_rnn = mean_squared_error(y_test, y_pred_rnn)
rmse_rnn = np.sqrt(mse_rnn)
mae_rnn = mean_absolute_error(y_test, y_pred_rnn)
print(f"MSE: {mse_rnn}")
print(f"RMSE: {rmse_rnn}")
print(f"MAE: {mae_rnn}")
#print(f"R²: {r2_rnn}")

3844/3844 [==============================] - 19s 5ms/step
MSE: 135709162.78083733
RMSE: 11649.4275730972
MAE: 9656.251108295968
```

Figure 9 : Implementing and evaluation of RNN

LSTM MODEL

```
[423]: model_lstm = Sequential([
    LSTM(128, activation='relu', input_shape=(X_train_rnn.shape[1], 1), kernel_regularizer=l2(0.01)),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=l2(0.01)),
    BatchNormalization(),
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.3),
    Dense(1)
])

model_lstm.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history_lstm = model_lstm.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=150,
    batch_size=64,
    callbacks=[early_stopping],
    verbose=1
)
```

```
•[424]: y_pred_lstm = model_lstm.predict(X_test)
r2_lstm = r2_score(y_test, y_pred_lstm)
mse_lstm = mean_squared_error(y_test, y_pred_lstm)
rmse_lstm = np.sqrt(mse_lstm)
mae_lstm = mean_absolute_error(y_test, y_pred_lstm)
print(f"MSE: {mse_lstm}")
print(f"RMSE: {rmse_lstm}")
print(f"MAE: {mae_lstm}")
#print(f"R²: {r2_lstm}")

3844/3844 [==============================] - 58s 15ms/step
MSE: 665935536.4883336
RMSE: 25805.72681573479
MAE: 21464.420460879763
```

Figure 10 : Implementing and evaluation of LSTM

```
[*]: model_cnn = Sequential([
    Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1), kernel_regularizer=l2(0.01)),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),
    Conv1D(128, kernel_size=3, activation='relu', kernel_regularizer=l2(0.01)),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),
    Flatten(),
    Dense(64, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.3),
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(1)
])

model_cnn.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
history_cnn = model_cnn.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=150,
    batch_size=64,
    callbacks=[early_stopping],
    verbose=1
)
```

```
[428]: y_pred_cnn = model_cnn.predict(X_test)
r2_cnn = r2_score(y_test, y_pred_cnn)
mse_cnn = mean_squared_error(y_test, y_pred_cnn)
rmse_cnn = np.sqrt(mse_cnn)
mae_cnn = mean_absolute_error(y_test, y_pred_cnn)
print(f"MSE: {mse_cnn}")
print(f"RMSE: {rmse_cnn}")
print(f"MAE: {mae_cnn}")
print(f"R²: {r2_cnn}")

3844/3844 [==============================] - 15s 4ms/step
MSE: 149549109.88382056
RMSE: 12229.027348232588
MAE: 7865.718515490652
```

Figure 11: Implementing and evaluation of CNN

3.7 Comparing evaluation metrics

In this section, Figure 12 clearly showing the comparison of all implemented model performance based on RMSE, MAE, MSE evaluation metrics. In finally based on this comparison ANN is performed well because ANN achieved less error. So these metrics are detailed over here. Figure 13 explains the identification and comparison of predicted prices and actual prices.

```
[431]:
metrics = {
    "Model": ["ANN", "RNN", "LSTM", "CNN"],
    #"R²": [r2_ann, r2_rnn, r2_lstm, r2_cnn],
    "MSE": [mse_ann, mse_rnn, mse_lstm, mse_cnn],
    "RMSE": [rmse_ann, rmse_rnn, rmse_lstm, rmse_cnn],
    "MAE": [mae_ann, mae_rnn, mae_lstm, mae_cnn]
}
metrics_df = pd.DataFrame(metrics)
metrics_df
```

Figure 12: Comparing result

```
y_test_array = y_test.to_numpy().ravel() if isinstance(y_test, pd.Series) else y_test.ravel()
y_pred_ann_array = y_pred_ann.to_numpy().ravel() if isinstance(y_pred_ann, pd.Series) else y_pred_ann.ravel()
comparison_df = pd.DataFrame({
    'Actual': y_test_array,
    'Predicted': y_pred_ann_array
})
print("Comparison of Actual and Predicted Values (ANN):")
print(comparison_df.head(5))
plt.figure(figsize=(12, 6))
plt.plot(y_test_array, label='Actual Values', color='blue', linestyle='-', marker='o', markersize=4)
plt.plot(y_pred_ann_array, label='Predicted Values (ANN)', color='orange', linestyle='--', marker='x', markersize=4)
plt.title('Comparison of Actual and Predicted Values (ANN)', fontsize=16)
plt.xlabel('Sample Index', fontsize=14)
plt.ylabel('Value', fontsize=14)
plt.legend(fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
plt.figure(figsize=(8, 8))
plt.scatter(y_test_array, y_pred_ann_array, alpha=0.7, color='brown')
plt.title('Scatter Plot of Actual vs Predicted Values (ANN)', fontsize=16)
plt.xlabel('Actual Values', fontsize=14)
plt.ylabel('Predicted Values', fontsize=14)
plt.grid(True)
min_val = min(min(y_test_array), min(y_pred_ann_array))
max_val = max(max(y_test_array), max(y_pred_ann_array))
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--', label='Ideal Prediction')
```

Figure 13: Predicting and actual values visual

4 Results

After executed of these above all steps, result achieved based on the research objective. In these four implemented advanced deep learning techniques are performed well and values.

Models	**Root Mean Square Error (RMSE)**	**Mean Absolute Error (MAE)**	**Mean Squared Error (MSE)**
ANN	**2510.298456**	**1682.450483**	**6301598.3393**
RNN	**11380.7728**	**9099.2021**	**129521990.2486**
CNN	**13297.9670**	**8534.6090**	**176835927.6497**
LSTM	**6181.7690**	**4060.1167**	**38214268.5791**