CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter includes a research framework describing the research workflow. The framework starts with the problem formulation, which provides a context for the problem identified from current research through literature reviews. Relevant data is collected based on the problem defined from reliable sources. Data pre-processing and exploratory data analysis are proposed for the dataset collected to discover patterns in the dataset and prepare the data for prediction. Two machine learning models are proposed for the prediction of health expenditure. The chapter ends with a summary after discussing result evaluation and hyperparameter tuning.

3.2 Research Framework

The research framework of this study involves 6 phases. The details of the research framework are listed in Figure 3.1.

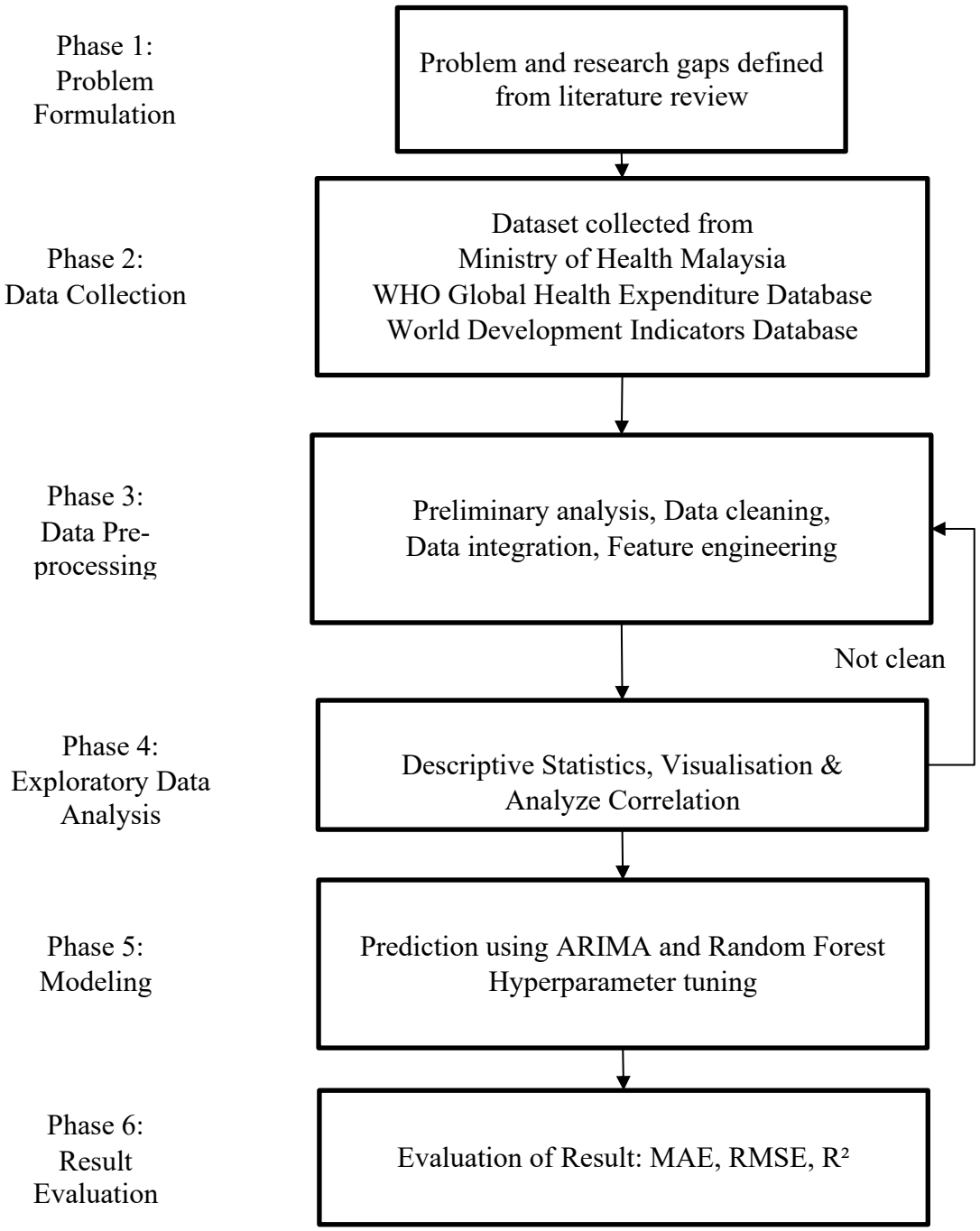

Figure 3.1 Research Framework of Health Expenditure Prediction

3.3 Problem Formulation

This study aims to predict the health expenditure of Malaysia to 2035, to provide insight for policymakers in health financing planning. From the literature review, several research gaps have been identified. Limited academic research has been done on Malaysia's Health expenditure forecasting, and there is no general consensus between studies on which determinants of health expenditure to be used for forecasting.

Current problems to be solved:

1. Data collection needs to be done for health expenditure-related data in Malaysia from year 2000-2022. Issues with data need to be resolved through data pre-processing, which includes data cleaning and data integration.

2. The correlation between variables is required to be analysed through exploratory data analysis. This will support the feature selections for the prediction model.

3. ARIMA and Random Forest will be used for the prediction of health expenditure. Evaluation and comparison of models need to be done to select the best model.

3.4 Data Collection

Data were collected from Ministry of Health Malaysia website, WHO Global Health Expenditure Database and World Development Indicators Database. Table 3.1 contains dataset details and the sources.

MHNA_2022.csv and MHNA_2017.csv datasets were scraped from the table in MHNA 2022.pdf and MHNA 2017.pdf to provide accurate data on total health

expenditure. The table scraping was done by using importing tabula library in Python. The scraped table was saved in the form of a CSV file.

For the dataset NHA indicators.xlsx, the data explorer of the WHO Global Health Expenditure Database was assessed on 26 May 2025. The respective indicators (as shown in Table 3.2) were selected, with the country chosen as Malaysia with the period set as 2000 to 2022. Currency is set as in million current National Currency Units (NCU). Excel file was generated and downloaded.

The last dataset is downloaded from World Development Indicators Database. The indicators were selected accordingly, with the year set as 2000 to 2022 and the country set as Malaysia. A Zip file was generated and downloaded, with 2 Excel files containing data and metadata of the dataset. Only the dataset P_Data_Extract_From_World_Development_Indicators.csv was used for the subsequent steps.

According to literature reviews, these datasets provide data required for predictive modelling. Table 3.2 describes the variables present in the datasets. The dataset contained time series data of health expenditures from 2000 to 2022, including Total Health Expenditure, Current Health Expenditure, Domestic General Government Health Expenditure and Out-of-pocket as a percentage of Current Health Expenditures.

The determinants of health proposed were included for further analysis, which include 8 distinct features, which are population size, GDP, number of physicians, number of hospital beds, population aged 65 years old and above, infant mortality rate, annual population growth and life expectancy at birth. These features were analysed in the exploratory analysis sections before being selected as predictors of health expenditures.

Table 3.1 Dataset details and sources

No	Dataset	Year	File size	Columns	Rows	Sources
1	MHNA_2022.csv	2011 - 2022	1KB	3	13	(MOH, 2024)
2	MHNA_2017.csv	1997 - 2017	1KB	3	22	(MOH, 2019)
3	NHA indicators.xlsx	2000-2022	8KB	26	8	(World Health Organization, 2025b)
4	P_Data_Extract_From_World_Development_Indicators.csv	2000-2022	2KB	27	12	(The World Bank, World Development Indicators, 2025)

Table 3.2 Dataset and variables

Dataset	Variables
MHNA_2017.csv, MHNA_2022.csv	Total Health Expenditure **(TEH)** in million (MYR)
NHA indicators.xlsx	Current Health Expenditure **(CHE)** in million (MYR)
	Domestic General Government Health Expenditure (**GGHE-D**), in million (MYR)
	Domestic Private Health Expenditure (**PVT-D**), in million (MYR)
	Out-of-pocket (**OOPS**) as % of Current Health Expenditure (CHE)
	Population Size (in thousands)
P_Data_Extract_From_World_ Development_Indicators.csv	Gross Domestic Product (GDP)
	Number of Physicians (per 1000 people)
	Number of Hospital Beds (per 1000 people)
	Population aged 65 years old and above, total
	Infant Mortality Rate (per 1000 live birth)
	Population Growth (annual %)
	Life Expectancy at birth, total (years)

3.5 Data Pre-processing

The data pre-processing for this project can be divided into the following subsections: preliminary analysis, data cleaning, data integration and feature engineering. Each step is crucial in ensuring the quality of the data for analysis and modelling. This step was done iteratively with exploratory data analysis to prepare a cleaned dataset before modelling.

3.5.1 Preliminary analysis

Preliminary analysis is the process of inspection of raw data to understand the dataset. It is done in the first part of data-preprocessing because it determines what subsequent steps should be done to the dataset to achieve the goals.

First, several Python libraries were imported into Jupyter notebook, which runs on a web browser. The libraries imported include numpy, pandas, and matplotlib. Next, the datasets were imported into the notebook. The dataset's information was accessed through pandas method df.info() and df.head() to check for the first few rows of data, how many rows and columns are present, data types of each column. Missing values, or null values, and duplicated values in the datasets are checked. The issues with the dataset were identified and noted to be resolved in the next step.

3.5.2 Data Cleaning

All the collected dataset needs to be cleaned before applying machine learning models to ensure accurate prediction is obtained and prevent errors from occurring due to missing values or inappropriate formats. The data cleaning process was done on each dataset individually. The data cleaning of the dataset starts by dropping unused columns and rows after exploring the dataset. The structure of the dataset was fixed by making sure the features are aligned across the columns. This step involves transposing the DataFrame.

```
#fix structure of wdi_data by dropping unused column
wdi_data = wdi_data.drop(columns= ['Country Name','Country Code','Series Code'])

#drop unused rows
wdi_data = wdi_data.drop(wdi_data.index[6:])

wdi_data
```

Figure 3.2 Dropping Unused Columns and Rows

```
#inverse the row and column after setting first column index
wdi_data= wdi_data.set_index(wdi_data.columns[0])
wdi_data = wdi_data.T
```

Figure 3.3 Fixing the Structure by Transposing the Dataframe

The datasets contain time series data. If the data changes gradually, missing values are filled by linear interpolation, which is estimated from the data points close to the missing values. However, if the missing values cannot be interpolated, for example, located at the start or end, backfill and forward fill were applied accordingly. The data points were imputed instead of being dropped because dropping data might disrupt the continuity of the time series data. Duplicated rows are identified and dropped. As the period used is between 2000 to 2022, the data outside the intended time frame is dropped as well.

```
# fill in missing value for physician data
# fill first missing value, interpolating from the back and forward value
wdi_data.loc[1, 'Physicians (per 1,000 people)'] = (wdi_data.loc[0, 'Physicians (per 1,000 people)'] + wdi_data.loc[2, 'Physicians (per 1,000 people)'])

# for missing value in 2022, fill using forward fill
wdi_data.loc[22, 'Physicians (per 1,000 people)'] = wdi_data.loc[21, 'Physicians (per 1,000 people)']

# fill in missing value for hospital beds data, using forward fill
wdi_data.loc[22, 'Hospital beds (per 1,000 people)'] = wdi_data.loc[21, 'Hospital beds (per 1,000 people)']

wdi_data
```

Figure 3.4 Filling Missing Values

The format of each dataset is reorganised by ensuring the year is the first column, in ascending order. If the dataset has year as its columns, it is transposed to make sure that all datasets are aligned the same. The index will need to be converted into columns if it contains important information, like the year. The index will be reset when necessary. This will prepare for the process of merging the dataset. The data types are changed into suitable data types for the columns, for example, text and integer. If there are commas between the integers or unintended brackets or parentheses, they will need to be removed. The columns are also renamed to reflect their features. It will make the dataset cleaner and easier for interpretation.

```
#change datatype of whole dataset except year to float
columns_to_convert = wdi_data.columns.difference(['Year'])
wdi_data[columns_to_convert]= wdi_data[columns_to_convert].astype(float)
```

Figure 3.5 Correcting Data Types of the Columns

```
# setting year as index
df['Year'] = pd.to_datetime(df['Year'], format='%Y')
df.set_index('Year', inplace=True)
```

Figure 3.6 Correcting Data Types of the 'Year' Column and Set as Index

3.5.3 Data Integration

Data integration refers to the compilation of datasets from various sources into a unified dataset. This allows data to be compared easily and is ready for the machine learning algorithm. 4 datasets are being used in this project. The datasets are merged into a single dataset after the cleaning process through concatenation and merging. Pandas' merge method will be used for this function, with an inner join chosen and merge on the 'Year' column. This will align variables across the dataset by using year as the key to consolidate a comprehensive dataset for analysis.

```
#combine 2 table into one according to year using concat
df_combined = pd.concat([df1_2000_2010, df2_extracted])
print(df_combined)
```

Figure 3.7 Concatenation

```
#merging data
df= df_combined.merge(who_nha_ind, on='Year', how= 'outer').merge(wdi_data, on='Year', how= 'outer')
df
```

Figure 3.8 Merging the Datasets

3.5.4 Feature Engineering

Feature refers to a variable in the dataset. In this step, the new feature was calculated from the existing features. For example, the Out-of-pocket health expenditure is represented as a percentage of Current Health Expenditure (CHE). Calculation of actual out-of-pocket health expenditure was conducted by multiplying the columns of CHE by the column with the percentage.

```
# transformation of OOPS into actual number instead of percentages
who_nha_ind['Out-of-pocket Health Expenditure (OOP)'] = (
    who_nha_ind['Out-of-pocket (OOPS) as % of Current Health Expenditure (CHE)'] / 100 *
    who_nha_ind['Current Health Expenditure (CHE)']
)
```

Figure 3.9 Feature Engineering

Feature selection was carried out at the end of the data pre-processing step. This step is important in improving model performance and reducing overfitting. The feature selection step selects the appropriate variables that are being used for the predictive modelling. This is based on the analysis done in the exploratory data analysis.

3.6 Exploratory Data Analysis

Exploratory Data Analysis was carried out to understand the dataset and identify hidden patterns in the dataset. This process can provide understanding of the data distribution, trends, relationship between features, and detect outliers and

anomalies. EDA and data pre-processing are iterative processes where the anomalies that are detected can be cleaned by repeating the data cleaning process. Descriptive statistics of the dataset will be computed and presented to provide an overview of the dataset. This will be done by df.describe() method.

Python libraries imported for this step include Matplotlib and Seaborn. Time series analysis was done by plotting a line graph for the health expenditures to analyse the data. Visualization helps to determine trend and seasonality in the time series data. It will determine whether the data is stationary and provide insight into the need for differencing in the next step.

```
# import necessary libraries for visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Create a line plot showing the total health expenditure over time
sns.set(style="whitegrid")
sns.lineplot(data= df, x= 'Year', y='Total Health Expenditure (TEH)',marker='o')
plt.title('Total Health Expenditure (TEH) in Malaysia From 2000 to 2022', fontsize=12, fontweight='bold')
plt.xlabel('Year', fontsize=10)
plt.ylabel('Health Expenditure (in million RM)', fontsize=10)
plt.savefig("TEH.png", dpi=1000)
plt.show()
```

Figure 3.10 Plotting Line Graph for Total Health Expenditure

Detailed correlation analysis was done by calculating the correlation between the features and target variable. Correlation between the variables were visualised by a Correlation heatmap. Renaming columns is done for easier view in visualization This step supports the feature selection step in data pre-processing.

```
#rearrange column
df = df[['Total Health Expenditure (TEH)','Domestic General Government Health Expenditure (GGHE-D)','Out-of-pocket Health Expenditure(OOP)',
         'Gross Domestic Product (GDP)', 'Population (in thousands)', 'Physicians (per 1,000 people)',
         'Hospital beds (per 1,000 people)', 'Population ages 65 and above, total',
         'Population growth (annual %)', 'Life expectancy at birth, total (years)',
         'Mortality rate, infant (per 1,000 live births)']]

#rename the column using shortform for easier views
short_name= {'Total Health Expenditure (TEH)':'TEH','Domestic General Government Health Expenditure (GGHE-D)':'GGHE',
             'Gross Domestic Product (GDP)' :'GDP', 'Population (in thousands)' :'Pop', 'Out-of-pocket Health Expenditure(OOP)':'OOP',
             'Physicians (per 1,000 people)' :'Phys No.', 'Hospital beds (per 1,000 people)':'HospBed No.',
             'Population ages 65 and above, total':'Pop65', 'Mortality rate, infant (per 1,000 live births)':'Infant Mort',
             'Population growth (annual %)' :'Pop growth', 'Life expectancy at birth, total (years)' :'Life Exp'}

df_acronym= df.rename(columns= short_name)

# view correlation between variables
corr= df_acronym.corr()
corr
```

Figure 3.11 Renaming Columns and Compute Correlation

```
# generate heatmap
plt.figure(figsize=(12,10))
sns.heatmap(data=corr, cmap= 'coolwarm', vmin= -1, vmax= 1,annot= True, annot_kws={"size": 12})
plt.savefig("correlation heatmap.png", dpi=1000)
```

Figure 3.12 Generate Heatmap from the Correlation Computed

3.7 Modeling

Two different approaches were used for data modelling. The models were trained on training data, evaluated with performance metrics according to test data, and then used to predict health expenditure to 2035. ARIMA and Random Forest were applied to the pre-processed dataset to predict the future health expenditure in Malaysia. For ARIMA, input data will consist of past data of total health expenditure, GHE and OOP as a time series analysis to predict to future value of health expenditure. For Random Forest, as the machine learning model can handle more variables than the ARIMA model, multiple features were selected for data modelling.

3.7.1 ARIMA

ARIMA model is a time series predicting model that can be broken down into three parts: autoregressive (AR), integrated (I), and moving average (MA). It integrated autoregressive modelling and moving average modelling, which are distinct approaches for predicting time series data. ARIMA is represented as ARIMA (p, d, q) model, where p is the order of the autoregressive component, d is the degree of differencing involved, and q is the order of the moving average part, which corresponds to the three components above.

The Autoregressive (AR) part of the ARIMA model represents a combination of past data points to forecast future values. It is represented by the equation

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t \quad (3.1)$$

where y_t is the dependent variable, and the lagged observations of y_t are used as predictors. c is the constant term, ε_t is the white noise or error term, while ϕ are the autoregressive coefficients, and p is the number of lagged components. The changes in ϕ varies the patterns while ε_t varies the scale of the time series (Hyndman & Athanasopoulos, 2025).

The moving average (MA) part focuses on the relationship between observations and the residual errors. It predicts using past forecast errors in a regression. MA model can capture meaningful short-term changes and remove random noise from the time series. MA can be presented in the equation

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \quad (3.2)$$

where y_t is the dependent variable, ε_t is the random noise, ε_{t-1} is the previous noise, θ is the moving average coefficient, c is the constant or long-term mean of the process, and q is the number of lagged error components. It is combined with AR to improve attention for recent incidents than the pure AR process (Siegel, 2016).

The integrated (I) part aims to turn the time series stationary by performing differencing to eliminate trend and seasonality. Trend is the long-term increase or decrease in the time series, for example, rising prices in GDP. Seasonality is the regular patterns that occur. Differencing can stabilise the mean of the time series by eliminating the changes in the level of a time series, thus removing or reducing trend and seasonality (Hyndman & Athanasopoulos, 2025). To determine the requirement for differencing, a unit root test needs to be done, and the Augmented Dickey-Fuller (ADF) test is chosen in this project

The integrated part is done so that non-stationary time series can be used for ARIMA process because both AR and MA assume stationarity. By combining all three components, the ARIMA model smooths out the changes while maintaining a general pattern of the time series (Siegel, 2016). Forecasting with ARIMA is useful for the prediction of health expenditure, where the values do not return to the long-term mean value.

Python is used to model ARIMA in this project. The required packages are imported from the statsmodels library as described in Figure 3.13.

```
# Import packages required
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
```

Figure 3.13 Packages imported for ARIMA model

ADFuller was imported, which conducts Augmented Dickey-Fuller test to confirm stationarity of the data. Differencing has to be done if the data is not stationary. The order of differencing required is the minimum number of differences needed to produce a stationary series.

```
result = adfuller(df['Total Health Expenditure (TEH)'])

print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

Figure 3.14 ADFuller

Before running the ARIMA (*p, d, q*) model, *p, d, q* terms need to be determined. ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) are two statistical tools used to determine the terms. The number of lagged values that ACF cuts off is set as q, and the same applies for p using PACF. A suitable value of d is

selected by ensuring that after the differencing all trends and seasonality have been eliminated.

```
# ACF plot
plot_acf(df['TEH_diff1'].dropna())
# PACF plot
plot_pacf(df['TEH_diff1'].dropna())

plt.show()
```

Figure 3.15 Plotting ACF and PACF plot

The forecast can be done after the *p, d, q* terms are determined. Chronological splitting is used, where data is split in 80% training and 20% testing. This means data from 2000 to 2018 is used as training data, while 2019 to 2022 is used as testing data. The training data was fit to the ARIMA model by setting the parameters accordingly. The forecast result and actual result were visualised by using line plot after reversing the differencing done to the forecasted result. The model performance will also be evaluated by using AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) model statistics to decide the best combination of terms. The lower the value, the better the model fits. Other statistical metrics, as mentioned in Section 3.8, are computed and compared with other models. After the model is tuned, estimation of health expenditure to 2035 will be done.

```
# train test split
train = df.iloc[:-int(len(df) * 0.2)]
test = df.iloc[-int(len(df) * 0.2):]


#fitting the time series data to the ARIMA model
model = ARIMA(train['TEH_diff1'], order=(0, 1, 0)).fit()
print(model.summary())

# forecast result
forecasts = model.forecast(len(test))
forecasts
```

Figure 3.16 Train Test Split and Fitting Data to ARIMA model

3.7.2 Random Forest

Random Forest is a supervised machine learning method that is chosen to model the dataset. Tree-based methods can be applied to regression problems, therefore, suitable for the prediction of healthcare expenditures. The individual decision tree is easy to interpret; however, it is not as accurate as other supervised learning approaches. Random forest ensembles multiple decision trees, each having moderate predicting capability, to achieve higher forecasting accuracy at the cost of some interpretability.

In bagging or bootstrap aggregation, several decision trees are created based on bootstrapped training samples. This is to overcome the main weakness of decision trees: high variance, which means each decision tree can produce very different results from the training data if we split the data at random and feed it to the decision trees. Bootstrapping refers to the process of resampling the training dataset with replacement to create many simulated samples. This allows multiple decision trees to be trained on the bootstrapped training dataset, and the results from each decision tree are combined and averaged to reduce variance.

Random Forest improves the bagging procedure by decorrelating the trees. This algorithm limits only several features, randomly selected from the total number of features in the training set, to be considered during each split in the tree. Commonly, m predictors are used at each split, calculated by the equation $m = \sqrt{p}$, where m, the number of predictors (features or independent variables), equals to square root of p, the full set of predictors. The algorithm aims to resolve the issue with bagging where the strong predictor in the training data will always be used at the top split in the tree, which results in the similarity of decision tree results when other less determining predictors are used at the split downwards (James et al., 2023).

Python is applied for this research and scikit learn library is imported for the purpose of training random forest on the cleaned data. There are several classes that are imported from different modules as shown in Figure 3.17.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, KFold
```

Figure 3.17 Classes Imported for the Random Forest Model

Next, the feature columns (X) and the target column (y) to predict will be set. The feature columns include all the features selected in the feature engineering step, while the target column to predict is the total health expenditure. Then, the data is split into a training and a testing set. As the data is a time series, chronological splitting is used. Data from 2000 to 2018 is used as training data, while 2019 to 2022 is used as testing data. This will result in 4 sets, which are X_train and X_ test for the independent variable, y_train, y_test for the dependent variable.

```
# Split train and test
train = df.iloc[:-int(len(df) * 0.2)]
test = df.iloc[-int(len(df) * 0.2):]

# dropping health expenditure for X and use total health expenditure for y
X_train = train.drop(['Total Health Expenditure (TEH)','Domestic General Government Health Expenditure (GGHE-D)',
                      'Out-of-pocket Health Expenditure(OOP)'],axis =1)
y_train = train['Total Health Expenditure (TEH)']

# dropping health expenditure for X and use total health expenditure for y
X_test = test.drop(['Total Health Expenditure (TEH)','Domestic General Government Health Expenditure (GGHE-D)',
                    'Out-of-pocket Health Expenditure(OOP)'],axis =1)
y_test = test['Total Health Expenditure (TEH)']
```

Figure 3.18 Train Test Split and Feature Selection

The random forest regressor is initiated. This model builds several decision trees and combines the predictions. Random state is set at 20 to ensure the result. There are multiple parameters in this model, including where number of decision trees, the maximum depth of the tree, the minimum samples to split an internal node and a leaf node, maximum features and the bootstrap option.

```
#instanstiate the model and fit to train set
model = RandomForestRegressor(random_state=20)
model.fit(X_train, y_train)

# predict the result
y_pred = model.predict(X_test)
```

Figure 3.19 Instantiate the Random Forest and Predict the Result

The regressor is fit to X_train and y_train, respectively, to train the model. After the model is trained, predictions can be made using the X_test as input and predict() method on the model. The prediction can be visualised by using a residual plot and a line plot to compare with the y_test result. The performance of the model was then evaluated with the metrics as described in the next section. All parameters in this model will be tuned as described in Section 3.9, Hyperparameter tuning, to achieve the best performance. The parameter setting with the best performance will be selected for final comparison with ARIMA model. After the model is fine-tuned, the prediction of health expenditure for 2035 will be done.

3.8 Evaluation

The models will be evaluated for their performance. This step is crucial as it not only validates the performance of the model, but also enables comparison of performance between different models. There are 3 key evaluation metrics applicable to regression models that will be used in this project, which will be discussed in detail in the following subsection.

```
: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

  # print evaluation metrics
  print("MAE:", mean_absolute_error(y_test, y_pred))
  print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
  print("R²:", r2_score(y_test, y_pred))
```

Figure 3.20 Evaluation metrics imported from Sklearn.metrics

3.8.1 Mean Absolute Error (MAE)

Mean absolute error calculates the mean of the errors by their absolute value. The absolute difference between the actual value and predicted value is calculated to avoid the effects of negative values cancelling out the positive value during summation. The absolute values of error are summed up and divided by the number of observations to get the average error. The equation of MAE is

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.3}$$

where n is the number of observations, y_i is the actual value and $\hat{y}_i$ is the predicted value. The metric is included because it is a simple metric for interpretation.

3.8.2 Root Mean Squared Error (RMSE)

Root Mean Squared Error is calculated by summing up squares of all the errors, calculating the mean, then square-rooting the result. The metric is square-rooted, so it is easier to compare to other metrics, like with mean absolute error. The equation of RMSE is

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \tag{3.4}$$

where n is the number of observations, y_i is the actual value and $\hat{y}_i$ is the predicted value. This metric measures the average magnitude of prediction error and is suitable for this project, where larger prediction errors need to be penalised more heavily to ensure the reliability of expenditure forecasting.

3.8.3 Coefficient of Determination (R^2 score)

The coefficient of determination, or R^2 is indicates the goodness of fit of a model. It reflects the model's performance in forecasting the outcomes. R^2 score equals to 1 indicates all the actual value lies perfectly on the prediction model, while $R^2 = 0$ indicates the model does not fit any actual value. The equation of R^2 is

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \overline{y_i})^2} \tag{3.5}$$

where n is number of observations, y_i is the actual value, $\hat{y}_i$ is the predicted value and $\overline{y_i}$ is the mean of actual value. Higher R^2 indicates better fit to the model.

3.9 Hyperparameter tuning

For the Random Forest model, hyperparameter tuning will be conducted using GridSearch Cross Validation to optimise the key parameters. Grid Search is chosen over the randomised search method because the dataset is small, and the Grid Search CV will result in better performance.

6 main parameters in this model are aimed to be tuned:

i) The number of trees in the forest.

ii) The maximum depth of the tree.

iii) The minimum samples to split an internal node.

iv) The minimum number of samples required to be at the leaf node.

v) The maximum features when finding the best split.

vi) Bootstrap. This will be set between True and False to determine whether the samples are bootstrapped when building the tree.

The parameters are set in a parameter grid, which is a dictionary mapping each parameter to the range of values that are intended for grid search. GridSearchCV is imported from sklearn.model_selection. The number of cross-validation (cv) is set to 5-fold. The RandomForestRegression, parameter grid, and cv are set as the parameters for GridSearchCV, then fit to the training data. The best parameter and best estimator are printed out to show the best combination of hyperparameters from the search. Lastly, the model is updated by using the parameters from the result above.

3.10 Summary

This chapter discusses the research methodology, from problem formulation, data collection, data cleaning, data pre-processing, exploratory data analysis, modelling and result evaluation. The ARIMA model and Random Forest model are discussed in depth, from the introduction, equations, strengths and weaknesses, parameters, statistical tools for setting up the model, hyperparameter tuning and the proposed application of the model by using Python. Three statistical metrics and proposed for the evaluation of the results between models. Initial results of the project will be presented in the next chapter.