Predicting the effects of Climate Change on Irish Agriculture

Rodrigo Matsumoto
CCT College Dublin

Sarah Kuprian Carrinho
CCT College Dublin

Climate Change in Ireland: Agriculture, Business & Machine Learning

Rodrigo Matsumoto

Sarah Kuprian Carrinho

A Report Submitted in Partial Fulfilment

of the requirements for the

Degree of

BSc in Computing in IT (4th year)

May 2023

Supervisor: Dr. Muhammad Iqbal

Table of Contents

Abstract

The impact of climate change on agriculture is a growing concern worldwide, and Ireland is no exception. The purpose of this project is to use machine learning techniques to predict the effects of climate change on Irish agriculture and identify strategies for adaptation and mitigation. The project uses the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology to guide the data analysis process, MoSCoW prioritization to identify the most critical needs, and SWOT analysis to evaluate the strengths, weaknesses, opportunities, and threats our project may encounter.

Historical temperature data for Ireland and Dublin will be used as our data sources. The project will use machine learning algorithms to predict the potential effects of climate change on agriculture and make recommendations for policymakers, farmers, and researchers to mitigate the potential effects of climate change on Irish agriculture.

Using CRISP-DM as our framework, the project began with a thorough business understanding phase, where we identified the key stakeholders and their information needs, as well as the challenges and opportunities associated with climate change and agriculture in Ireland (FAO, 2016). This helped us to define our project objectives more clearly and to develop a comprehensive plan for data collection, analysis, and modelling.

Introduction

Climate change is one of the most significant challenges facing the world today, and its impact on agriculture has become increasingly significant. Agriculture is very sensitive to weather and climate, and relies on natural resources that are directly affected by the climate. Changes in climate may impact agricultural productivity, natural resources (such as water and soil) and the health of agricultural workers and livestock (United States EPA, 2022). In recent years, the impact of climate change on agriculture has become increasingly significant, as rising temperatures, changing precipitation patterns, and extreme weather events have disrupted agricultural production and threatened global food security (IPCC, 2019; Ray et al., 2019).

In Ireland, agriculture is a critical industry, contributing to the economy and providing a significant portion of the nation's food supply (Teagasc, 2021). However, the agricultural sector in Ireland is highly vulnerable to the impacts of climate change, with rising temperatures, changing rainfall patterns, and extreme weather events projected to have significant effects on agricultural productivity, natural resources, and the health of agricultural workers and livestock (EPA, 2021).

The aim of this project is to develop a machine learning model to predict the impacts of climate change on Irish agriculture and identify strategies for adaptation and mitigation.

Our project will involve analyzing datasets related to temperature changes in Ireland and Dublin. We will use statistical models and machine learning algorithms to identify patterns and trends in these datasets, and to predict the impact of climate change on crop yields, water availability, and other key variables.

Overall, we aim to develop a predictive model that can help stakeholders in the Irish agriculture sector to adapt to the challenges of climate change, and help businesses to gain a competitive advantage by leveraging the insights provided by the model.

We will be using the **Cross Industry Standard Process for Data Mining (CRISP-DM)** framework to guide and structure our project.

Technologies Used for This Project

The technologies that will be used to complete the technical phases of our project, as outlined in the CRISP-DM framework, are described below.

Programming Language

Our programming language of choice is **Python:**

- High-level general purpose programming language (Kuhlman, D., 2012)
- Extensive availability of libraries with tools for ***manipulating***, ***visualizing***, and ***training*** machine learning models (e.g. pandas, Matplotlib, NumPy) (Tuama, D., 2022)
- Ideal for computationally-intensive applications and general purpose systems (McKinney, W., 2013).
- Open source.

Environment for development

Jupyter Notebook

We will be using Jupyter Notebook as our web-based environment for the development and presentation of our project.

- **Open source** web application, part of Project Jupyter.
- Supports the creation and sharing of documents that contain **live code**, equations, **visualizations**, and text.
- Free for download (on its own, or through the **Anaconda repository**).
- The core programming languages it supports are: Julia, **Python** and R.

(Driscoll, M.)

Project Goals and Objectives

Using the MoSCoW prioritization technique, we are able to identify the most critical needs of our project in order to successfully organize its development and adaptation strategies.

The MoSCoW prioritization resulted in the following list of objectives:

- **Must-Haves:**
    - Analysis and result evaluation of historical yearly temperature data for Ireland from 1961 to 2021.
    - Analysis and result evaluation of historical monthly temperature data for Dublin from November 1941 to March 2023.
    - Predictive machine learning model for temperature trend prediction of the next decades.

- **Should-Haves:**
    - Visualizations of the results acquired through data mining.
    - Visualizations of climate predictions.
    - Visualizations comparing the different Machine Learning regression models used.

- **Could-Haves:**
    - User interface for the analyzed historical data.
    - User interface for the machine learning model.

- **Won't-Haves:**

- Website to market our project's goals and allow users to access the results of our analysis and explore the climate predictions of the ML model.

Roles and Responsibilities

Identifying an area of interest and idea

Rodrigo	Sarah	Supervisor
Finding and selecting and area of interest	Finding and selecting and area of interest	Support students to identify areas of interest relevant to project requirements

Developing a proposal and Strategic Analysis / Business Case

Rodrigo	Sarah	Supervisor
Responsible for research, developing and writing the proposal	Responsible for research, developing and writing the proposal	Feedback on whether project meets QQI Level 8 standards
		Feedback on feasibility and sustainability of the project and technologies chosen.

Obtaining the data

Rodrigo	Sarah	Advisor
Responsible for carrying out the selection and "collection" of the datasets	Supported the selection of the datasets	Advice on legal and ethical issues related to the data sourcing

Data Understanding and Initial Analysis

Rodrigo	Sarah	Advisor
Responsible for carrying out the initial analysis and understanding of the data	Responsible for supporting and reviewing the initial analysis and understanding of the data	Available to support students on code development, structure, and quality of the data

Data Preparation

Rodrigo	Sarah	Advisor
Responsible for carrying out the preparation of the data	Responsible for supporting and reviewing the data preparation process	Available to support students on code development, structure, and quality of the data

Modelling

Rodrigo	Sarah	Advisor
Responsible for creating the ML model, performing cross-validation, fitting and evaluating the model	Responsible for supporting the development of the ML model	Available to support students on code development and machine learning algorithms to be used
Responsible for carrying out improvements on the model	Responsible for supporting and advising improvement steps	

Evaluation

Rodrigo	Sarah	Advisor
Responsible for performing the evaluation of the models		Available to support students on code development and evaluation of the machine learning models
Responsible for interpreting, understanding and developing a conclusion from the models' evaluation	Responsible for interpreting, understanding and developing a conclusion from the models' evaluation	

Deployment

Rodrigo	Sarah	Advisor

Responsible for drawing conclusions and performing a project review of results.	Responsible for drawing conclusions and performing a project review of results.	Available to support students on project results.

Developing the Project Report

Rodrigo	Sarah	Advisor
Responsible for images and graphs of the report	Responsible for research and writing of the report	Available to support and provide feedback to students on the report's progress, writing and structuring
Responsible for supporting and advising on the report's development	Responsible for review and proofreading of the report	
Responsible for review and proofreading of the report	Responsible for analysis and deriving conclusions from the results obtained	
Responsible for analysis and deriving conclusions from the results obtained		

Developing the Poster Presentation

Rodrigo	Sarah	Advisor
Responsible for developing the poster presentation	Responsible for supporting and reviewing the poster development	Available to support and provide feedback to students on the poster presentation
Responsible for review and proofreading of the poster	Responsible for review and proofreading of the poster	Provided template for the poster

Developing the slides for Pre-Recorded Presentation

Rodrigo	Sarah	Advisor
Responsible for creating the slides, preparing and participating in the pre-recorded presentation	Responsible for creating the slides, preparing and participating in the pre-recorded presentation	

CRISP-DM Overview

The CRISP-DM framework for the data mining process consists of six iterative phases, described below:

1. **Business Understanding:** Concrete goals for data mining and requirements are defined; outline of project plan and business goals.
2. **Data Understanding**: Initial data is collected; an initial analysis of the data and its quality is carried out; it is identified whether the available data meets the requirements defined in the previous phase.
3. **Data Preparation:** Here, relevant data is selected, cleaned and prepared in order to be used in the next modelling phase.
4. **Modelling:** A modelling technique is chosen; the model is created and assessed.
5. **Evaluation:** The results produced are evaluated; the overall process is reviewed; next steps are determined.
6. **Deployment:** The deployment, maintenance and monitoring are planned; the final report is concluded; the project is reviewed.

2.1. Data Understanding - Surface Temperature Change

For this project we will be working with two different datasets. Initially, we will be dealing with them separately. The first dataset '**Surface_Temperature_Change.csv**' deals with the surface temperature change globally, by country and by year.

Surface Temperature Change dataset

Import the libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import geopandas as gpd
pd.set_option('display.max_columns', None)
import warnings
warnings.filterwarnings('ignore')
```

Import the data

```
df = pd.read_csv('../Project/Data/Surface_Temperature_Change.csv')
df
```

FAO.[FAOSTAT]. License: CC BY-NC-SA 3.0 IGO. Extracted from: [https://www.fao.org/faostat/en/#data/ET]. Date of Access: [20-03-2023].

This dataset provides information on changes in global surface temperature across all countries from 1961 to 2021 using temperatures between 1951 and 1980 as a baseline (Climate Change International Monetary Fund). The temperature is measured in degrees celsius.
The dataset was obtained and used for this project sourced by the Food and Agriculture Organization of the United Nations (FAO) in compliance to its Statistical Database Terms of Use, which can be found here: https://www.fao.org/contact-us/terms/db-terms-of-use/en/

Preview of the dataset

	ObjectId	Country	ISO2	ISO3	Indicator	Unit	Source	CTS_Code	CTS_Name	CTS_Full_Descriptor	F1961	F1962	F1963	F1964	F1965	F1
0	1	Afghanistan, Islamic Rep. of	AF	AFG	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	ECCS	Surface Temperature Change	Environment, Climate Change, Climate Indicator...	-0.105	-0.157	0.852	-0.743	-0.211	0
1	2	Albania	AL	ALB	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	ECCS	Surface Temperature Change	Environment, Climate Change, Climate Indicator...	0.627	0.330	0.068	-0.172	-0.393	0
2	3	Algeria	DZ	DZA	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	ECCS	Surface Temperature Change	Environment, Climate Change, Climate Indicator...	0.162	0.131	0.110	0.284	-0.081	0
3	4	American Samoa	AS	ASM	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	ECCS	Surface Temperature Change	Environment, Climate Change, Climate Indicator...	0.066	-0.055	0.160	-0.150	-0.582	0
4	5	Andorra, Principality of	AD	AND	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	ECCS	Surface Temperature Change	Environment, Climate Change, Climate Indicator...	0.744	0.102	-0.762	0.300	-0.492	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

The columns 'ISO2' and 'ISO3' are two and three-letter country codes defined by the International Organization for Standardization (ISO).
The 'CTS' columns (8, 9 and 10) describe what the data contained in the dataset is referring to: the surface temperature change.
The 'Indicator' column tells us the temperature values contained in the dataset were measured as the "temperature change with respect to a baseline climatology, corresponding to the period between 1951 and 1980".
The data was sourced by the Food and Agriculture Organization of the United States as indicated in the 'Source' column.
The later columns correspond to the years, from 1961 to 2021.

Remove unnecessary columns

```
df.drop(['ISO2', 'ISO3', 'CTS_Code', 'CTS_Name', 'CTS_Full_Descriptor'], axis=1, inplace=True)
df.head()
```

Dataset after removal of ‘ISO2’, ‘ISO3’, ‘CTS_Code’, ‘CTS_Name’, ‘CTS_Full_Descriptor” columns

	ObjectId	Country	Indicator	Unit	Source	F1961	F1962	F1963	F1964	F1965	F1966	F1967	F1968	F1969	F1970	F1971	F1972	F1973	F19
0	1	Afghanistan, Islamic Rep. of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	-0.105	-0.157	0.852	-0.743	-0.211	0.156	-0.389	-0.384	-0.539	0.898	0.652	-1.089	0.262	-0.4
1	2	Albania	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.627	0.330	0.068	-0.172	-0.393	0.553	-0.080	0.073	-0.023	-0.119	-0.200	-0.077	-0.299	-0.1
2	3	Algeria	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.162	0.131	0.110	0.284	-0.081	0.436	0.006	-0.027	0.278	0.114	-0.380	-0.342	-0.028	-0.5
3	4	American Samoa	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.066	-0.055	0.160	-0.150	-0.582	0.166	-0.364	-0.174	0.142	-0.036	-0.473	-0.070	0.322	-0.3
4	5	Andorra, Principality of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.744	0.102	-0.762	0.300	-0.492	0.407	0.621	-0.013	-0.176	0.081	-0.355	-0.526	-0.010	-0.4

Dataset information

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 66 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ObjectId   227 non-null    int64
 1   Country    227 non-null    object
 2   Indicator  227 non-null    object
 3   Unit       227 non-null    object
 4   Source     227 non-null    object
 5   F1961      191 non-null    float64
 6   F1962      192 non-null    float64
 7   F1963      191 non-null    float64
 8   F1964      190 non-null    float64
 9   F1965      190 non-null    float64
 10  F1966      194 non-null    float64
 11  F1967      192 non-null    float64
 12  F1968      191 non-null    float64
 13  F1969      191 non-null    float64
 14  F1970      190 non-null    float64
 15  F1971      193 non-null    float64
 16  F1972      195 non-null    float64
 17  F1973      195 non-null    float64
 18  F1974      193 non-null    float64
 19  F1975      190 non-null    float64
 20  F1976      192 non-null    float64
 21  F1977      191 non-null    float64
 22  F1978      192 non-null    float64
 23  F1979      191 non-null    float64
 24  F1980      193 non-null    float64
 25  F1981      192 non-null    float64
 26  F1982      192 non-null    float64
```

```
 57  F2013      216 non-null    float64
 58  F2014      216 non-null    float64
 59  F2015      216 non-null    float64
 60  F2016      215 non-null    float64
 61  F2017      214 non-null    float64
 62  F2018      216 non-null    float64
 63  F2019      215 non-null    float64
 64  F2020      214 non-null    float64
 65  F2021      214 non-null    float64
dtypes: float64(61), int64(1), object(4)
memory usage: 117.2+ KB
```

With the ‘info()’ command we are able to gather the following information about the ‘Surface_Temperature_Change’ dataset:

- It contains a total of 227 entries and 66 columns
- The year columns contain float data types, while the rest are made up of Object data types and Integer values (ObjectID);
- All year columns contain null values.

Displaying the number of rows and columns with ‘shape’ command

```
df.shape
```

```
(227, 66)
```

Displaying the number of unique values per column with ‘nunique()’ command

```
df.nunique()
```

```
ObjectId     227
Country      227
Indicator      1
Unit           1
Source         1
            ...
F2017        196
F2018        194
F2019        197
F2020        202
F2021        200
Length: 66, dtype: int64
```

Displaying the summary of statistics with ‘describe()’ command

```
df.describe()
```

	ObjectId	F1961	F1962	F1963	F1964	F1965	F1966	F1967	F1968	F1969	F1970	F1971	F
count	227.000000	191.000000	192.000000	191.000000	190.000000	190.000000	194.000000	192.000000	191.000000	191.000000	190.000000	193.000000	195.00
mean	114.013216	0.157152	-0.018589	-0.009361	-0.084511	-0.250305	0.110010	-0.118010	-0.197230	0.154440	0.097689	-0.190124	-0.07
std	65.696573	0.405438	0.345941	0.381624	0.308949	0.265949	0.386485	0.346238	0.276508	0.302022	0.354175	0.232647	0.38
min	1.000000	-0.745000	-0.910000	-1.273000	-0.876000	-1.060000	-1.793000	-1.002000	-1.624000	-0.939000	-1.284000	-0.879000	-1.79
25%	57.500000	-0.112000	-0.184500	-0.202500	-0.253500	-0.408750	-0.042500	-0.286000	-0.323000	-0.016000	-0.038750	-0.307000	-0.19
50%	114.000000	0.052000	-0.084000	-0.011000	-0.094500	-0.235000	0.078000	-0.156000	-0.186000	0.197000	0.133000	-0.206000	-0.02
75%	170.500000	0.325000	0.111250	0.186000	0.109000	-0.100250	0.286000	0.018750	-0.063000	0.362000	0.293750	-0.069000	0.11
max	230.000000	1.906000	1.057000	1.204000	1.100000	0.856000	1.421000	1.135000	0.478000	0.808000	0.978000	0.683000	0.94

3.1. Data Preparation

Check for missing values

```
# Check for missing data
missing_df = df.isnull().any()

# Print out columns with missing data, if any
if missing_df.any():
    print("The following columns have missing data:")
    print(missing_df[missing_df].index.tolist())
else:
    print("No columns have missing data.")
```

```
The following columns have missing data:
['F1961', 'F1962', 'F1963', 'F1964', 'F1965', 'F1966', 'F1967', 'F1968', 'F1969', 'F1970', 'F1971', 'F1972', 'F1973', 'F1974',
'F1975', 'F1976', 'F1977', 'F1978', 'F1979', 'F1980', 'F1981', 'F1982', 'F1983', 'F1984', 'F1985', 'F1986', 'F1987', 'F1988',
'F1989', 'F1990', 'F1991', 'F1992', 'F1993', 'F1994', 'F1995', 'F1996', 'F1997', 'F1998', 'F1999', 'F2000', 'F2001', 'F2002',
'F2003', 'F2004', 'F2005', 'F2006', 'F2007', 'F2008', 'F2009', 'F2010', 'F2011', 'F2012', 'F2013', 'F2014', 'F2015', 'F2016',
'F2017', 'F2018', 'F2019', 'F2020', 'F2021']
```

‘Missing_df’ is an array of all columns that contain missing data.

The ‘isna().sum()’ functions display a sum of all the null values in each column.

```
# check for missing data
missing_df = df.isna().sum()

print(missing_df)
```

```
ObjectId      0
Country       0
Indicator     0
Unit          0
Source        0
             ..
F2017        13
F2018        11
F2019        12
F2020        13
F2021        13
Length: 66, dtype: int64
```

As shown previously with the ‘info()’ command, all year columns contain NA values.

Handling missing values

Dropping the ‘ObjectID’ column2

```
df= df.drop(['ObjectId'], axis=1)
df
```

	Country	Indicator	Unit	Source	F1961	F1962	F1963	F1964	F1965	F1966	F1967	F1968	F1969	F1970	F1971	F1972	F1973	F1974	F19
0	Afghanistan, Islamic Rep. of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	-0.105	-0.157	0.852	-0.743	-0.211	0.156	-0.389	-0.384	-0.539	0.898	0.652	-1.089	0.262	-0.470	-0.4
1	Albania	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.627	0.330	0.068	-0.172	-0.393	0.553	-0.080	0.073	-0.023	-0.119	-0.200	-0.077	-0.299	-0.134	-0.2
2	Algeria	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.162	0.131	0.110	0.284	-0.081	0.436	0.006	-0.027	0.278	0.114	-0.380	-0.342	-0.028	-0.502	-0.5
4	Andorra, Principality of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.744	0.102	-0.762	0.300	-0.492	0.407	0.621	-0.013	-0.176	0.081	-0.355	-0.526	-0.010	-0.412	0.2
5	Angola	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.051	-0.145	-0.186	-0.228	-0.207	0.162	-0.088	-0.194	0.191	0.249	-0.092	-0.029	0.477	-0.152	-0.0

Dropping all rows that contain NA values

```
df = df.dropna()
df.isna().sum()
```

```
ObjectId     0
Country      0
Indicator    0
Unit         0
Source       0
            ..
F2017        0
F2018        0
F2019        0
F2020        0
F2021        0
Length: 66, dtype: int64
```

Further preparation and reshaping of the dataset

Renaming the ‘Country’ column using ‘rename’ function

```
df= df.rename(columns={'Country':'Countries'})
df
```

	Countries	Indicator	Unit	Source	F1961	F1962	F1963	F1964	F1965	F1966	F1967	F1968	F1969	F1970	F1971	F1972	F1973	F1974	F19
0	Afghanistan, Islamic Rep. of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	-0.105	-0.157	0.852	-0.743	-0.211	0.156	-0.389	-0.384	-0.539	0.898	0.652	-1.089	0.262	-0.470	-0.4
1	Albania	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.627	0.330	0.068	-0.172	-0.393	0.553	-0.080	0.073	-0.023	-0.119	-0.200	-0.077	-0.299	-0.134	-0.2
2	Algeria	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.162	0.131	0.110	0.284	-0.081	0.436	0.006	-0.027	0.278	0.114	-0.380	-0.342	-0.028	-0.502	-0.5
4	Andorra, Principality of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.744	0.102	-0.762	0.300	-0.492	0.407	0.621	-0.013	-0.176	0.081	-0.355	-0.526	-0.010	-0.412	0.2
5	Angola	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the	0.051	-0.145	-0.186	-0.228	-0.207	0.162	-0.088	-0.194	0.191	0.249	-0.092	-0.029	0.477	-0.152	-0.0

Renaming the year columns

```
# rename columns
new_names = {col: col[1:] for col in df.columns if col.startswith('F') and col[1:].isdigit()}
df = df.rename(columns=new_names)
df.head()
```

	Countries	Indicator	Unit	Source	1961	1962	1963	1964	1965	1966	1967	1968	1969	1970	1971	1972	1973	1974	1975
0	Afghanistan, Islamic Rep. of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	-0.105	-0.157	0.852	-0.743	-0.211	0.156	-0.389	-0.384	-0.539	0.898	0.652	-1.089	0.262	-0.470	-0.468
1	Albania	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.627	0.330	0.068	-0.172	-0.393	0.553	-0.080	0.073	-0.023	-0.119	-0.200	-0.077	-0.299	-0.134	-0.203
2	Algeria	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.162	0.131	0.110	0.284	-0.081	0.436	0.006	-0.027	0.278	0.114	-0.380	-0.342	-0.028	-0.502	-0.554
4	Andorra, Principality of	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.744	0.102	-0.762	0.300	-0.492	0.407	0.621	-0.013	-0.176	0.081	-0.355	-0.526	-0.010	-0.412	0.207
5	Angola	Temperature change with respect to a baseline ...	Degree Celsius	Food and Agriculture Organization of the Unite...	0.051	-0.145	-0.186	-0.228	-0.207	0.162	-0.088	-0.194	0.191	0.249	-0.092	-0.029	0.477	-0.152	-0.018

Here we have removed the letter ‘F’ which was present in all the year columns, that now only contain numeric characters. We do this by using a conditional statement.

Applying the 'melt()' method and creating a new 'df2' dataframe.

The Pandas.melt function is used to reshape the data from a wide format (multiple columns) to a long format (more rows) (GeeksforGeeks, undated). Now, each year is a row in the 'Year' column, while the temperatures are presented in the 'Total_Temperature' column.

```
years = [str(x) for x in range(1961,2022)]

df2 = df.melt(id_vars = ['Countries'],
              value_vars=years,
              var_name='Year',
              value_name='Total_Temperature'
             )
df2
```

	Countries	Year	Total_Temperature
0	Afghanistan, Islamic Rep. of	1961	-0.105
1	Albania	1961	0.627
2	Algeria	1961	0.162
3	Andorra, Principality of	1961	0.744
4	Angola	1961	0.051
...	...	...	...
9694	West Bank and Gaza	2021	1.887
9695	Western Sahara	2021	1.557
9696	World	2021	1.442
9697	Zambia	2021	1.002
9698	Zimbabwe	2021	-0.101

9699 rows × 3 columns

Grouping all the countries and performing the sum of their total temperatures over the years with 'groupby()' and 'sum()' functions.

```
total_df2= df2.groupby('Countries')['Total_Temperature'].sum()
total_df2
```

```
Countries
Afghanistan, Islamic Rep. of    31.209
Albania                         30.534
Algeria                         42.886
Andorra, Principality of        43.452
Angola                          33.662
                                 ...
West Bank and Gaza              20.920
Western Sahara                  44.767
World                           33.914
Zambia                          31.536
Zimbabwe                        14.044
Name: Total_Temperature, Length: 159, dtype: float64
```

Creating a new dataframe 'new_df' with the values of 'total_df2'

```
new_df = pd.DataFrame({'country': total_df2.index, 'total': total_df2.values})
new_df
```

	country	total
0	Afghanistan, Islamic Rep. of	31.209
1	Albania	30.534
2	Algeria	42.886
3	Andorra, Principality of	43.452
4	Angola	33.662
...	...	...
154	West Bank and Gaza	20.920
155	Western Sahara	44.767
156	World	33.914
157	Zambia	31.536
158	Zimbabwe	14.044

159 rows × 2 columns

This new dataframe was created with the values obtained with 'total_df2'. The 'country' column is made up of the indexes of 'total_df2' while the 'total' column is the corresponding total temperature values of each row.

Grouping the top 10 countries with the highest total temperature sum.

```
top_10_countries = total_df2.sort_values(ascending=False).head(10)
top_10_countries
```

```
Countries
Mongolia                        53.822
Gambia, The                     50.402
Mauritania, Islamic Rep. of     49.708
Austria                         47.985
Morocco                         47.739
Senegal                         47.568
Finland                         47.076
Guinea-Bissau                   46.777
Liechtenstein                   46.310
Tunisia                         45.175
Name: Total_Temperature, dtype: float64
```

Creating new ‘df3’ dataset that includes only Ireland

```
df3 = df2[df2['Countries'] == 'Ireland']
df3
```

	Countries	Year	Total_Temperature
71	Ireland	1961	0.262
230	Ireland	1962	-0.690
389	Ireland	1963	-0.776
548	Ireland	1964	0.154
707	Ireland	1965	-0.558
...	...	...	...
8975	Ireland	2017	1.318
9134	Ireland	2018	0.834
9293	Ireland	2019	1.248
9452	Ireland	2020	1.117
9611	Ireland	2021	1.057

61 rows × 3 columns

If the row value for the column ‘Country’ is ‘Ireland’, then include all rows in ‘df3’.

Measuring maximum and minimum temperatures found for Ireland using the max() and min() functions.

```
max_temp = df3['Total_Temperature'].max()
min_temp = df3['Total_Temperature'].min()
print(f"Max temperature: {max_temp}, Min temperature: {min_temp}")
```

```
Max temperature: 1.424, Min temperature: -0.776
```

Measuring the average temperature in Ireland using the mean() method.

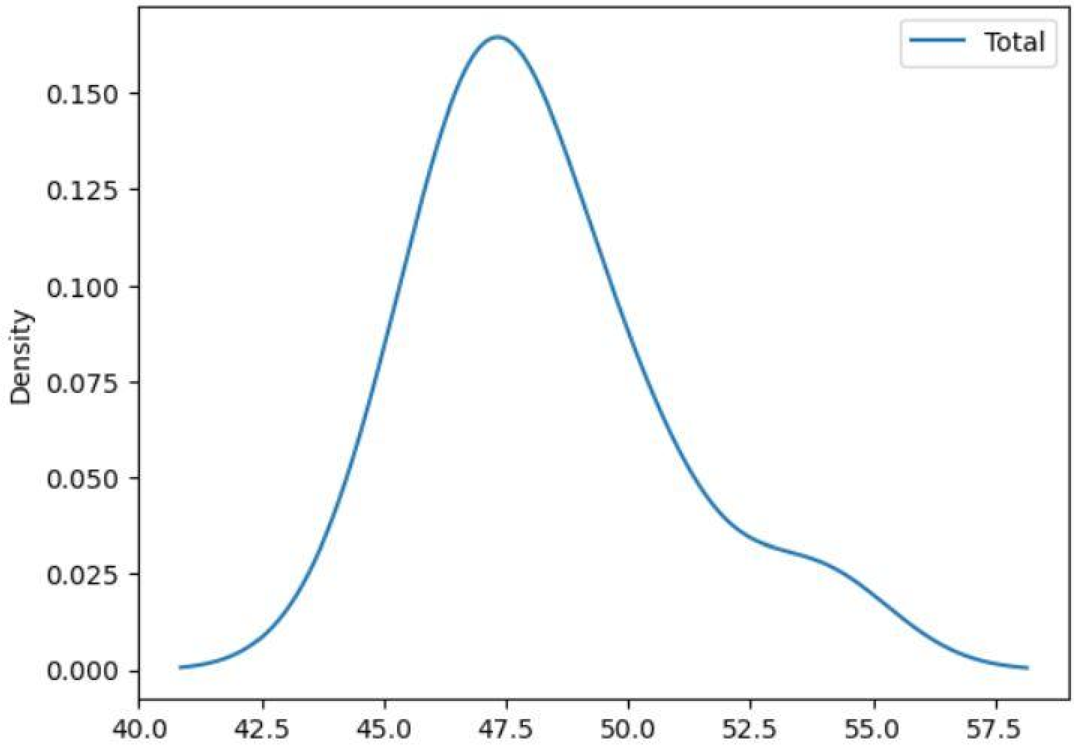
```
avg_temp = df3['Total_Temperature'].mean()
print(f"Average temperature: {avg_temp}")
```

```
Average temperature: 0.41242622950819674
```

3.1. Graphical Analysis after initial data preparation

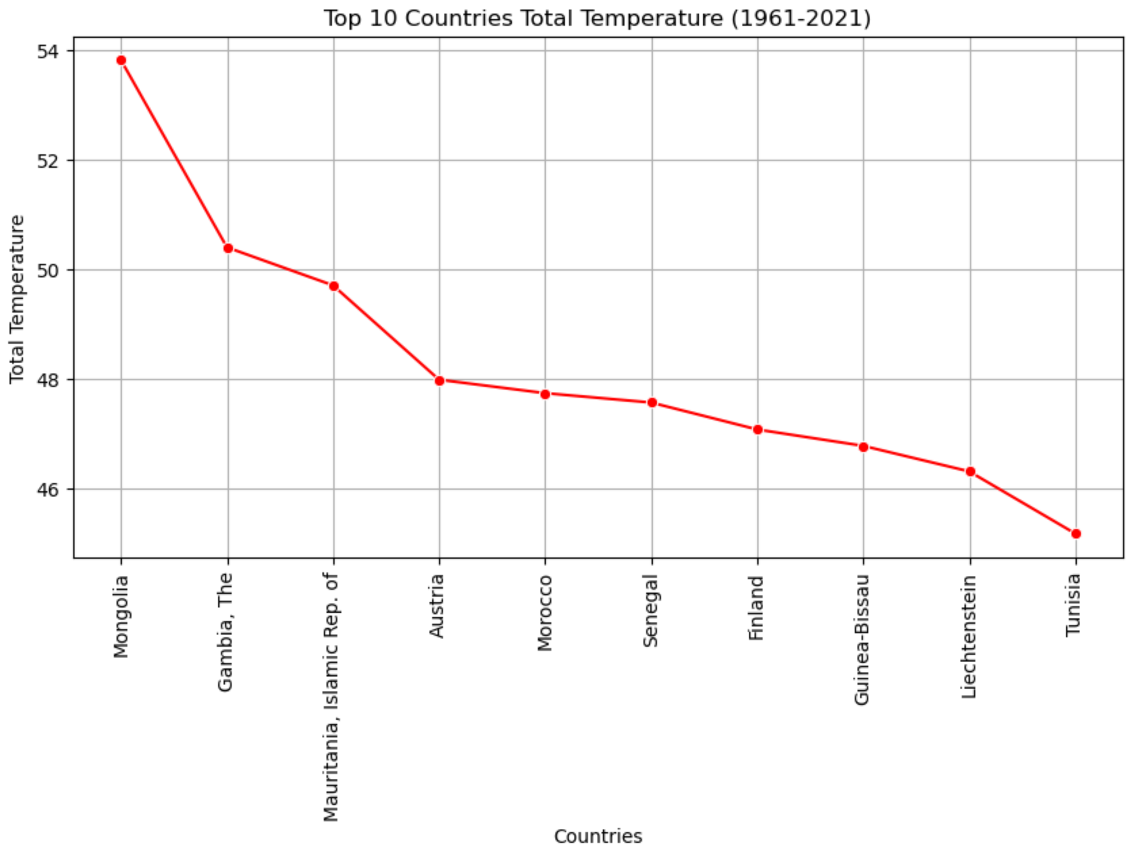
Visualizing the distribution of the 'Total' values from the 'top_10_countries' using Seaborn's Kernel Density Estimate (KDE) plot

```
df_top_countries.plot(kind='kde')
plt.show()
```



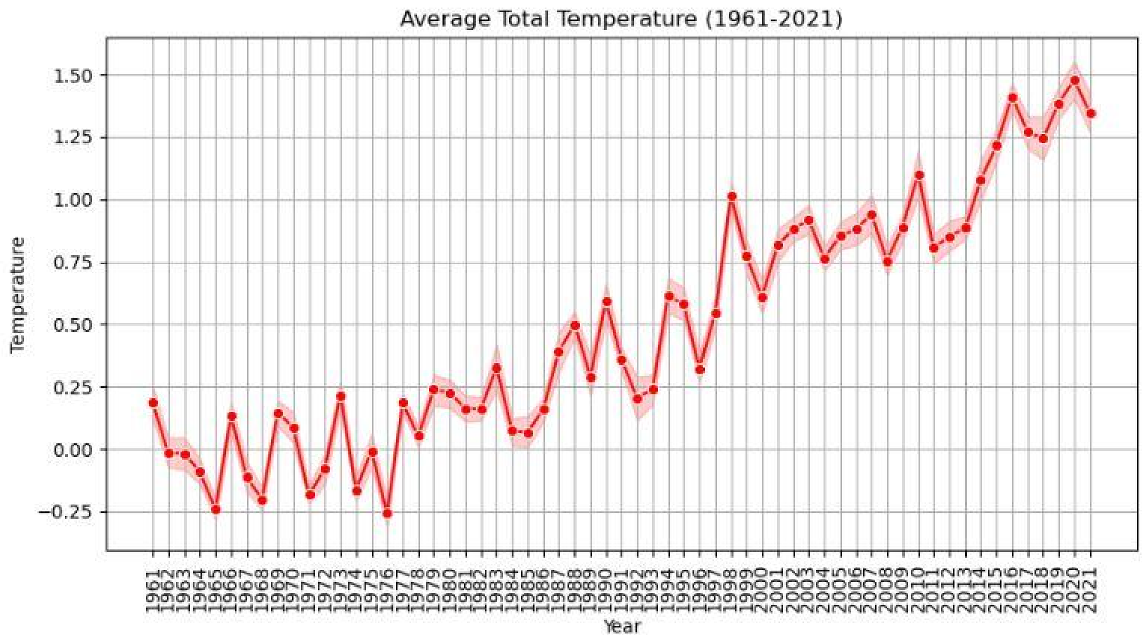
Visualizing the Total Temperatures of the Top 10 Countries with the highest temperature sum using Seaborn's line plot.

```
plt.figure(figsize=(10,5))
x=sns.lineplot(data=df_top_countries, x='Country', y='Total', color='red', marker='o')
plt.xticks(rotation=90)
x.yaxis.grid()
x.xaxis.grid()
plt.title('Top 10 Countries Total Temperature (1961-2021)')
plt.xlabel('Countries')
plt.ylabel('Total Temperature')
plt.show()
```


Top 10 Countries Total Temperature (1961-2021)
Total Temperature
54
52
50
48
46
Mongolia
Gambia, The
Mauritania, Islamic Rep. of
Austria
Morocco
Senegal
Finland
Guinea-Bissau
Liechtenstein
Tunisia
Countries

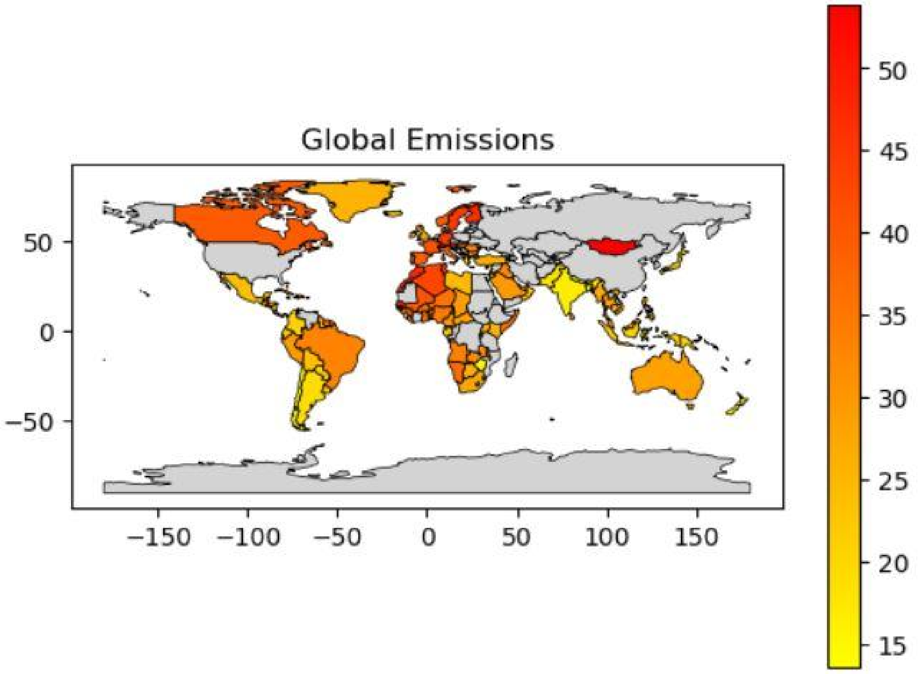
Visualizing the average total temperature change from 1961 to 2021 using Seaborn's line plot

```
plt.figure(figsize=(10,5))
x=sns.lineplot(data=df2, x='Year', y='Total_Temperature', color='red', marker='o
plt.xticks(rotation=90)
x.yaxis.grid()
x.xaxis.grid()
plt.title('Average Total Temperature (1961-2021)')
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.show()
```



World heat map for Global Emissions merged with our existing ‘new_df’ dataset.

```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world_map = world.merge(new_df, left_on='name', right_on='country', how='left')
x = world_map.plot(column='total',
            cmap='autumn_r',
            edgecolor='Black',
            linewidth=0.5,
            missing_kwds={"color": "lightgray"},
            legend=True)
x.set_title('Global Emissions')

plt.show()
```



4.1. Modelling

Importing the libraries

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split
```

We will be using the ***scikit-learn*** machine learning library for our model development. More specifically the **sklearn** module and the following functions:

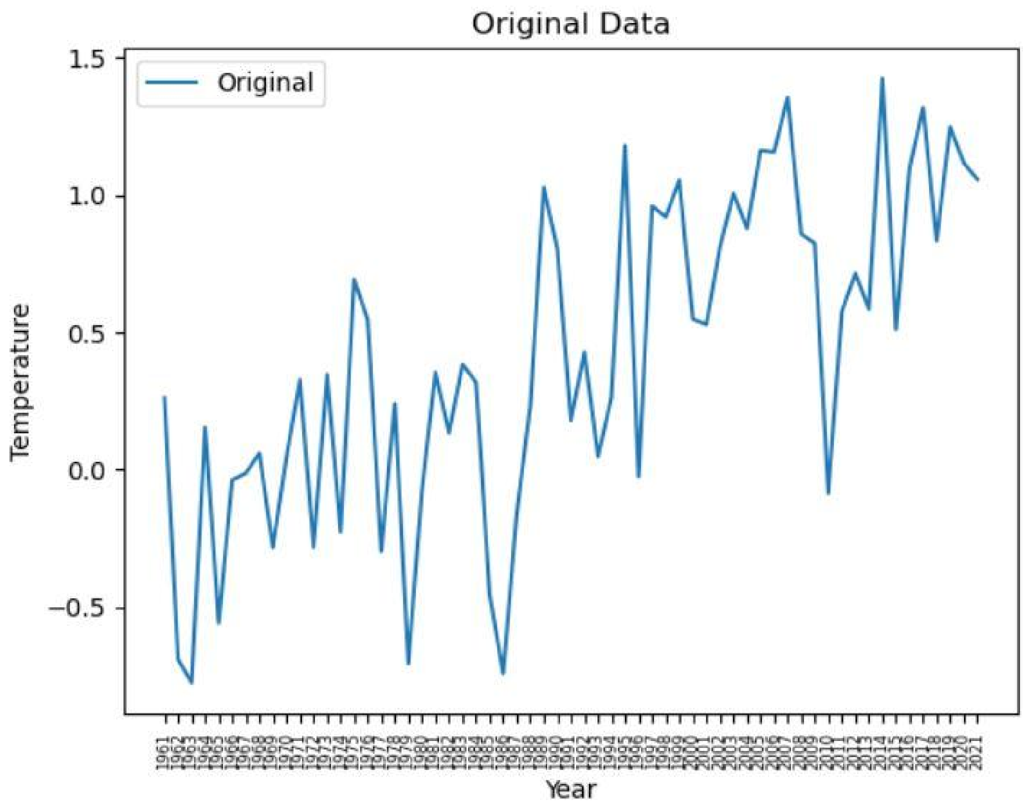
- **LinearRegression** for our Linear Regression model
- **PolynomialFeatures** for our Polynomial Regression model
- **RandomForestRegressor** for our Random Forest Regression model
- **mean_squared_error** to measure the error index of our models
- **TimesSeriesSplit** to provide train/test indices to split our time series data (scikit-learn documentation)
- **train_test_split** to randomly split our data into train and test subsets.

The data used in the modelling will be from the dataframe ‘df3’

```
data = df3
data2 = df3
```

Plotting the original data using Matplotlib.pyplot

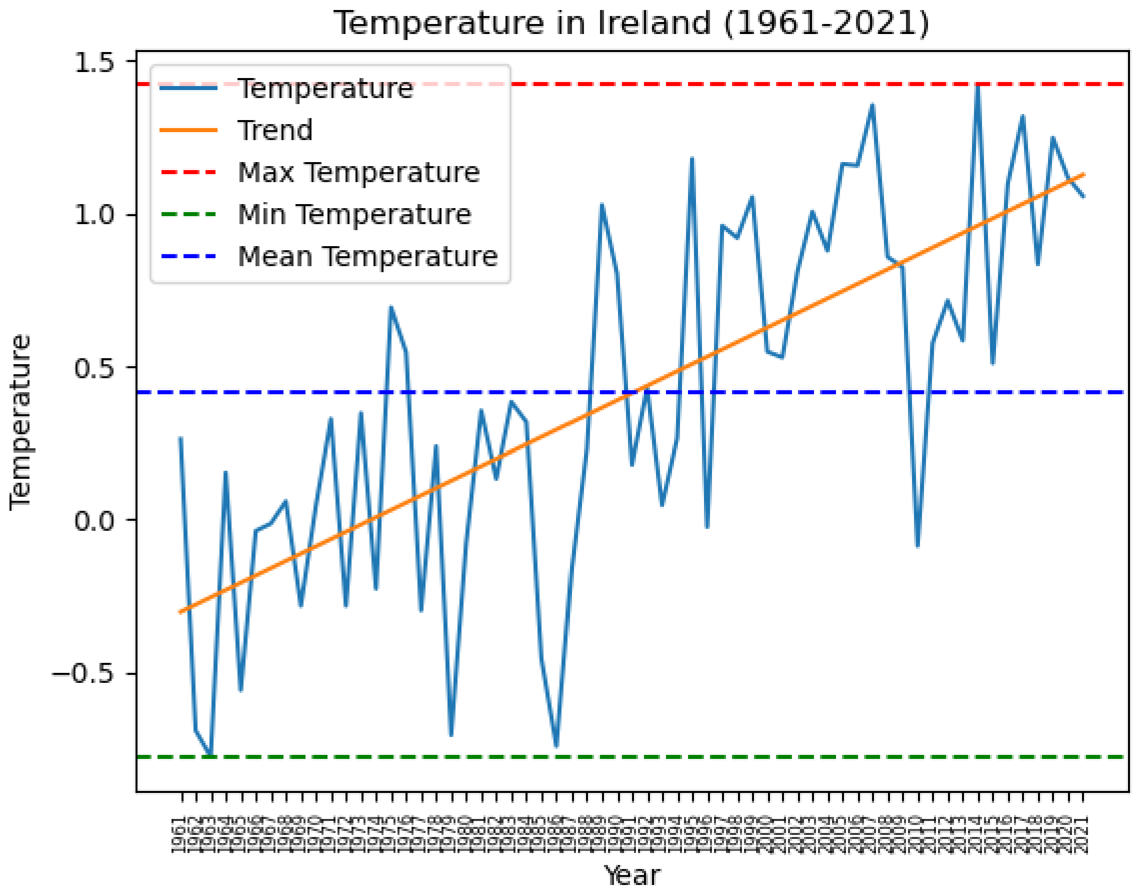
```
plt.plot(data.Year, data['Total_Temperature'], label='Original')
plt.xlabel('Year')
plt.xticks(fontsize=6, rotation=90)
plt.ylabel('Temperature')
plt.title('Original Data')
plt.legend()
plt.show()
```



Plotting the original data using Matplotlib.pyplot with Minimum, Maximum and Mean temperatures + trend

```
xx = years.values.reshape(-1, 1)
yx = temps.values.reshape(-1, 1)

modelx = LinearRegression()
modelx.fit(xx, yx)

trendx = modelx.predict(xx)
```

```
plt.plot(years, temps, label='Temperature')
plt.plot(years, trendx, label='Trend')

plt.axhline(y=max_temp, color='r', linestyle='--', label='Max Temperature')
plt.axhline(y=min_temp, color='g', linestyle='--', label='Min Temperature')
plt.axhline(y=mean_temp, color='b', linestyle='--', label='Mean Temperature')

plt.title('Temperature in Ireland (1961-2021)')
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.legend()
plt.xticks(fontsize=6, rotation=90)
plt.show()
```



Train Test Split

Using the 'reshape()' method

```
X = df3['Year'].values.reshape(-1, 1)
y = df3['Total_Temperature'].values.reshape(-1, 1)
```

Train Test Split using sklearn train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

ML Model Development

We will be creating and performing the same code for each of our models, step by step, and later conclude which model performed better predictions overall.

Linear Regression

We begin with the simplest Machine Learning algorithm: Linear Regression.
With Linear Regression we analyse the relationship between two variables. Here, the variables 'Total_Temperature' and 'Year'. This relationship, if drawn in a two-dimensional space, will result in a straight line. This line is made up of points in a graph that best fits our data points for our dependent variable 'Total_Temperature' and our independent variable 'Year', and it is the one that results in the least error (N. S., Chauhan, 2019).
Our model is trained as the algorithm runs multiple times, until it has found all constants. We can then start using it to perform our predictions.

Creating the Linear Regression 'model_lr' Model with scikit-learn 'LinearRegression' and fitting the model with 'fit()' method using X_train and y_train values.

```
# Train the model
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)
```

```
LinearRegression()
```

Creating the predicted values 'y_pred_lr' on the test data

```
# Predict on the test data
y_pred_lr = model_lr.predict(X_test)
```

Calculating the Mean Squared Error (MSE) for our Linear Regression model using ‘mean_squared_error’ function from scikit-learn.metrics module

```
mse_lr = mean_squared_error(y_test, y_pred_lr)
print(f"Linear Regression MSE: {mse_lr}")
#rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
#print('Linear Regression RMSE:', rmse_lr)
```

```
Linear Regression MSE: 0.20042577262075825
```

Polynomial Regression

This special case of linear regression may produce better results since our variables may not present a simple linear relationship.

The polynomial equation produces a curvilinear relationship between our target and independent variable, where the target variable changes in a non-uniform manner (A. Sharma, 2022)

Transforming the features into polynomial features using scikit-learn ‘PolynomialFeatures’ and ‘poly.fit_transform’ methods.

```
# Transform the features to polynomial features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.fit_transform(X_test)
```

Creating the Polynomial Regression ‘model_poly’ Model with scikit-learn ‘LinearRegression’ and fitting the model with X_train_poly and y_train_poly values

```
model_poly = LinearRegression()
model_poly.fit(X_train_poly, y_train)
```

Creating the predicted values ‘y_pred_poly’ on the test data

```
# Predict on the test data
y_pred_poly = model_poly.predict(X_test_poly)
```

Calculating the Mean Squared Error (MSE) for our Polynomial Regression Model using ‘mean_squared_error’ function from scikit-learn.metrics module

```
# Calculate the mean squared error
mse_poly = mean_squared_error(y_test, y_pred_poly)
print(f"Polynomial Regression MSE: {mse_poly}")
```

```
Polynomial Regression MSE: 0.6303938043309947
```

Random Forest Regression

With the Random Forest model we create a number of random decision trees base models and combine them into a single “ensemble model”.
At each decision split the algorithm makes, a random sample of our attributes (our decision trees’ average results) is drawn and whichever gives the “highest information gain” is chosen, and the algorithm continues to traverse down the decision trees, repeating this process (Russell, S. J. & Norvig, P., 2022).
Random Forest is well known for producing good predictions and efficiently handling large datasets (Mbaabu, O., 2020).

Creating the Random Forest Regression ‘model_rf’ Model with scikit-learn ‘RandomForestRegressor’ and fitting the model with ‘fit()’ method using X_train and y_train values.

```
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
model_rf.fit(X_train, y_train.ravel())
```

Here we also set the number of trees in the forest ‘n_estimators’ to 100 and the ‘random_state’ to 42. We also use the ‘numpy.ravel()’ function to return ‘y_train’ as a contiguous flattened array since the ‘fit()’ function expects a flat array.

Creating the predicted values ‘y_pred_rf’ on the test data

```
# Predict on the test data
y_pred_rf = model_rf.predict(X_test)
```

Calculating the Mean Squared Error (MSE) for our Random Forest Regression Model using ‘mean_squared_error’ function from scikit-learn.metrics module

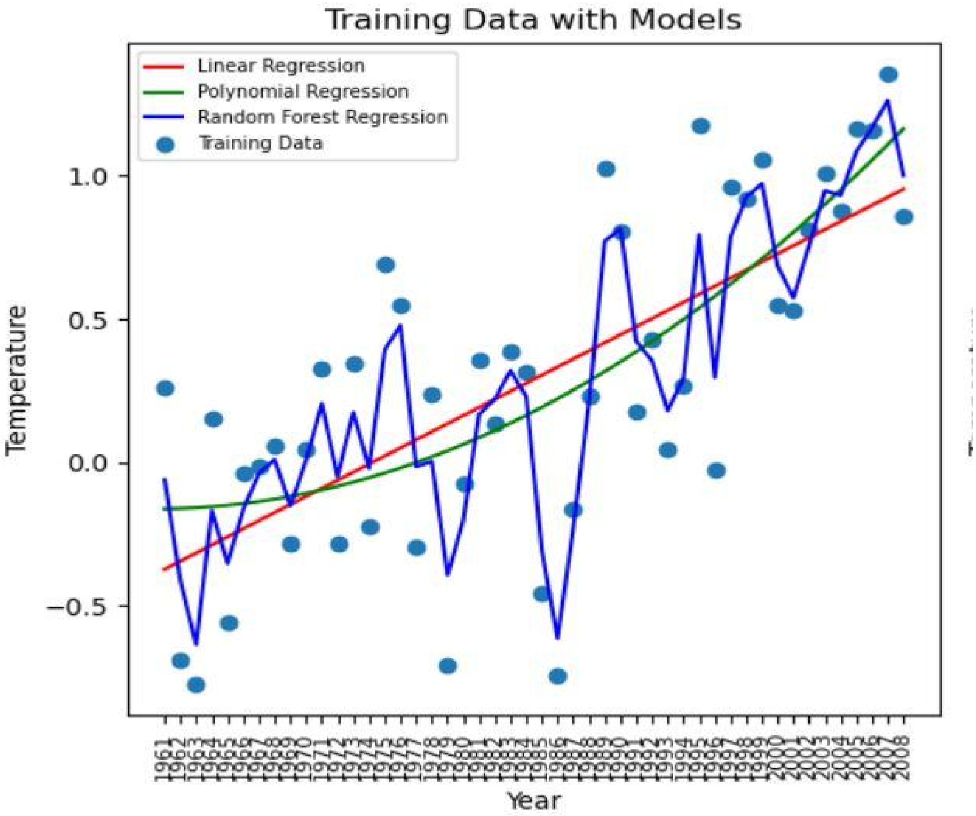
```
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Regression MSE: {mse_rf}")
```

```
Random Forest Regression MSE: 0.1758744031769233
```

We can see that the Random Forest Regression model performed with better accuracy if compared to the other two models, with an MSE value of 0.175.
We can also see that the Linear Regression model performed well, with a close MSE score of 0.2. However in the following sections we will be able to visualize the predictions in plots and better understand the predictions made by each, and conclude which model best fits our needs.

Training Data vs. Model Predictions using matplotlib.pyplot subplot()

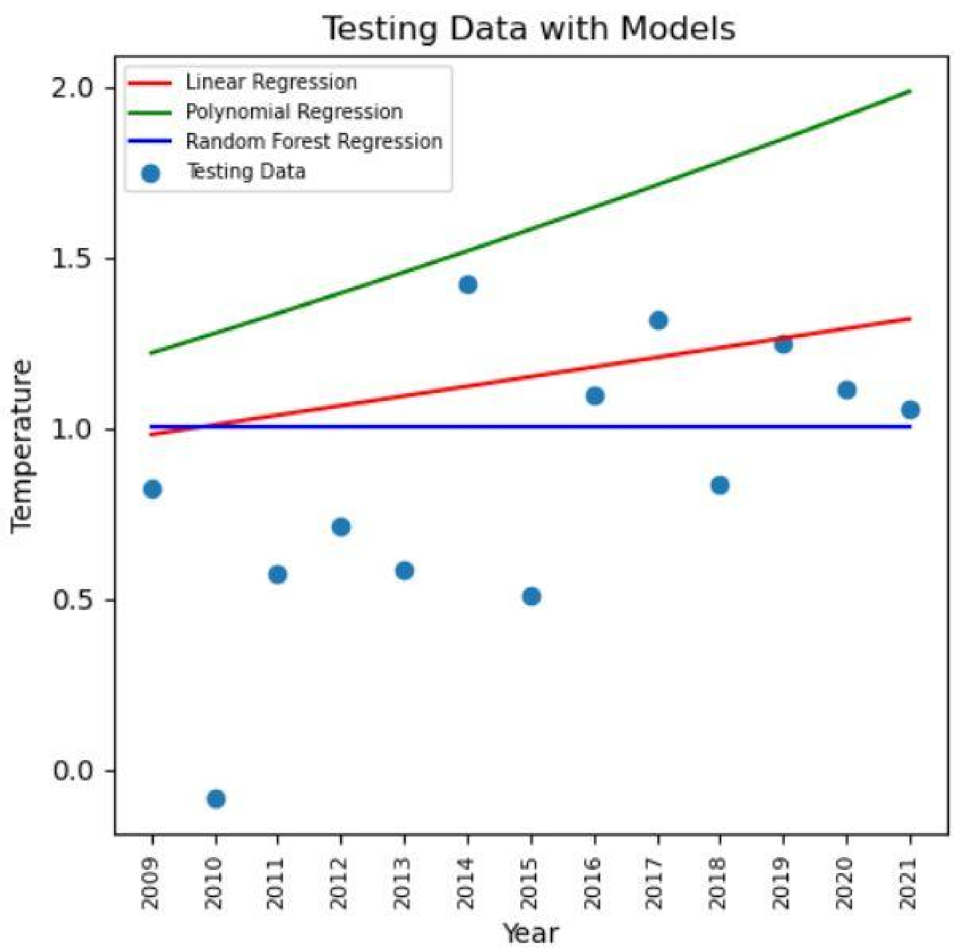
```
plt.subplot(1, 2, 1)
plt.plot(X_train.flatten(), model_lr.predict(X_train), color='red', label='Linear Regression')
plt.plot(X_train.flatten(), model_poly.predict(X_train_poly), color='green', label='Polynomial Regression')
plt.plot(X_train.flatten(), model_rf.predict(X_train), color='blue', label='Random Forest Regression')
plt.scatter(X_train.flatten(), y_train, label='Training Data')
plt.xlabel('Year')
plt.xticks(fontsize=8, rotation=90)
plt.ylabel('Temperature')
plt.title('Training Data with Models')
plt.legend(fontsize=7)
```



Through this plot we are able to see that the regressor that most accurately captures the information of the data is the Random Forest Regressor. The Linear Regression and Polynomial Regression models better show the tendency for increase in the temperatures. However they do not capture temperature drops seen in the historical data and which can occur in the future.

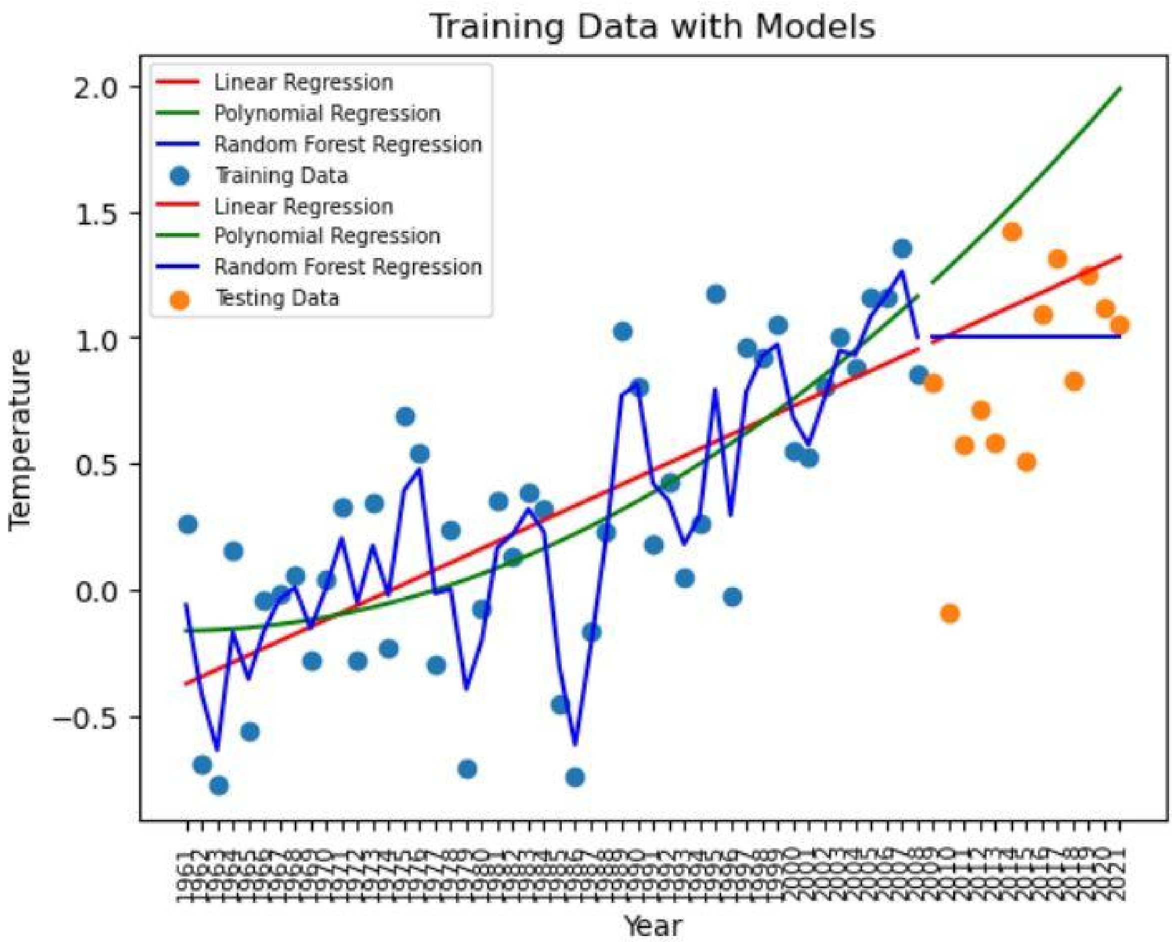
Plotting the Test Data vs. Model Predictions using matplotlib.pyplot subplot()

```
# Testing Data with Models
plt.subplot(1, 2, 2)
plt.plot(X_test.flatten(), y_pred_lr, color='red', label='Linear Regression')
plt.plot(X_test.flatten(), y_pred_poly, color='green', label='Polynomial Regression')
plt.plot(X_test.flatten(), y_pred_rf, color='blue', label='Random Forest Regression')
plt.scatter(X_test.flatten(), y_test, label='Testing Data')
plt.xlabel('Year')
plt.xticks(fontsize=8, rotation=90)
plt.ylabel('Temperature')
plt.title('Testing Data with Models')
plt.legend(fontsize=7)
```



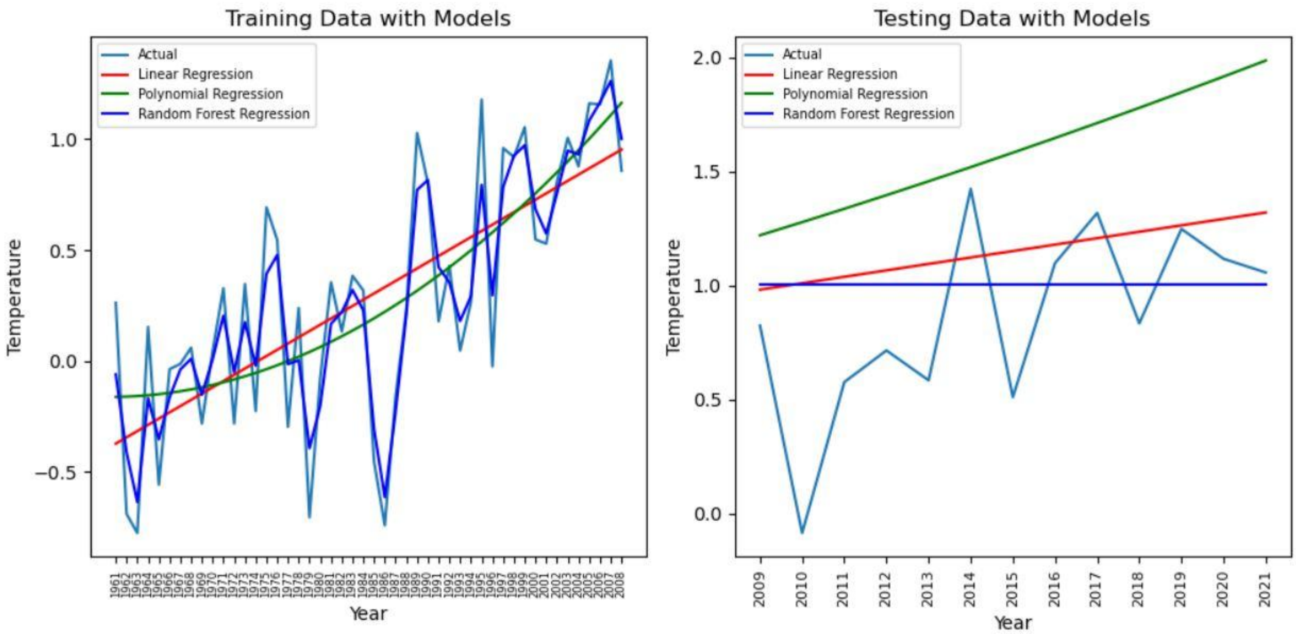
Combining both plots into a single subplot using Matplotlib.pyplot

```
plt.plot(X_train.flatten(), model_lr.predict(X_train), color='red', label='Linear Regression')
plt.plot(X_train.flatten(), model_poly.predict(X_train_poly), color='green', label='Polynomial Regression')
plt.plot(X_train.flatten(), model_rf.predict(X_train), color='blue', label='Random Forest Regression')
plt.scatter(X_train.flatten(), y_train, label='Training Data')
plt.plot(X_test.flatten(), y_pred_lr, color='red', label='Linear Regression')
plt.plot(X_test.flatten(), y_pred_poly, color='green', label='Polynomial Regression')
plt.plot(X_test.flatten(), y_pred_rf, color='blue', label='Random Forest Regression')
plt.scatter(X_test.flatten(), y_test, label='Testing Data')
plt.xlabel('Year')
plt.xticks(fontsize=8, rotation=90)
plt.ylabel('Temperature')
plt.title('Training Data with Models')
plt.legend(fontsize=7)
plt.show()
```



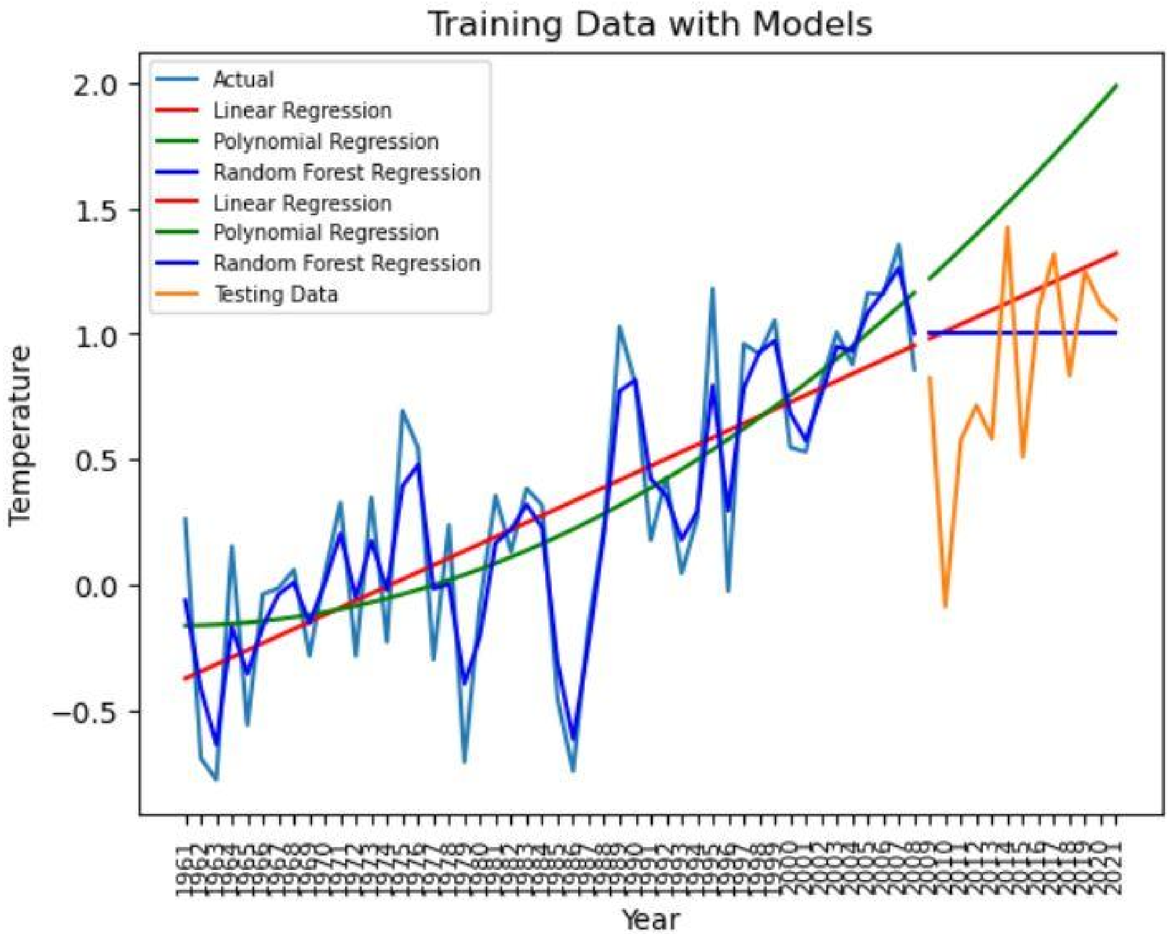
Training and Test Data vs. Predicted Data of all three models

```
plt.figure(figsize=(10, 5))

# Training Data with Models
plt.subplot(1, 2, 1)
#plt.plot(X_train.flatten(), y_train.flatten(), color='#F0F0F0', label='Actual')
plt.plot(X_train.flatten(), y_train.flatten(), label='Actual')
plt.plot(X_train.flatten(), model_lr.predict(X_train), color='red', label='Linear Regression')
plt.plot(X_train.flatten(), model_poly.predict(X_train_poly), color='green', label='Polynomial Regression')
plt.plot(X_train.flatten(), model_rf.predict(X_train), color='blue', label='Random Forest Regression')
#plt.scatter(X_train.flatten(), y_train, label='Training Data')
plt.xlabel('Year')
plt.xticks(fontsize=6, rotation=90)
plt.ylabel('Temperature')
plt.title('Training Data with Models')
plt.legend(fontsize=7)

# Testing Data with Models
plt.subplot(1, 2, 2)
#plt.plot(X_test.flatten(), y_test.flatten(), color='#F0F0F0', label='Actual')
plt.plot(X_test.flatten(), y_test.flatten(), label='Actual')
plt.plot(X_test.flatten(), y_pred_lr, color='red', label='Linear Regression')
plt.plot(X_test.flatten(), y_pred_poly, color='green', label='Polynomial Regression')
plt.plot(X_test.flatten(), y_pred_rf, color='blue', label='Random Forest Regression')
#plt.scatter(X_test.flatten(), y_test, label='Testing Data')
plt.xlabel('Year')
plt.xticks(fontsize=8, rotation=90)
plt.ylabel('Temperature')
plt.title('Testing Data with Models')
plt.legend(fontsize=7)

plt.tight_layout()
plt.show()
```



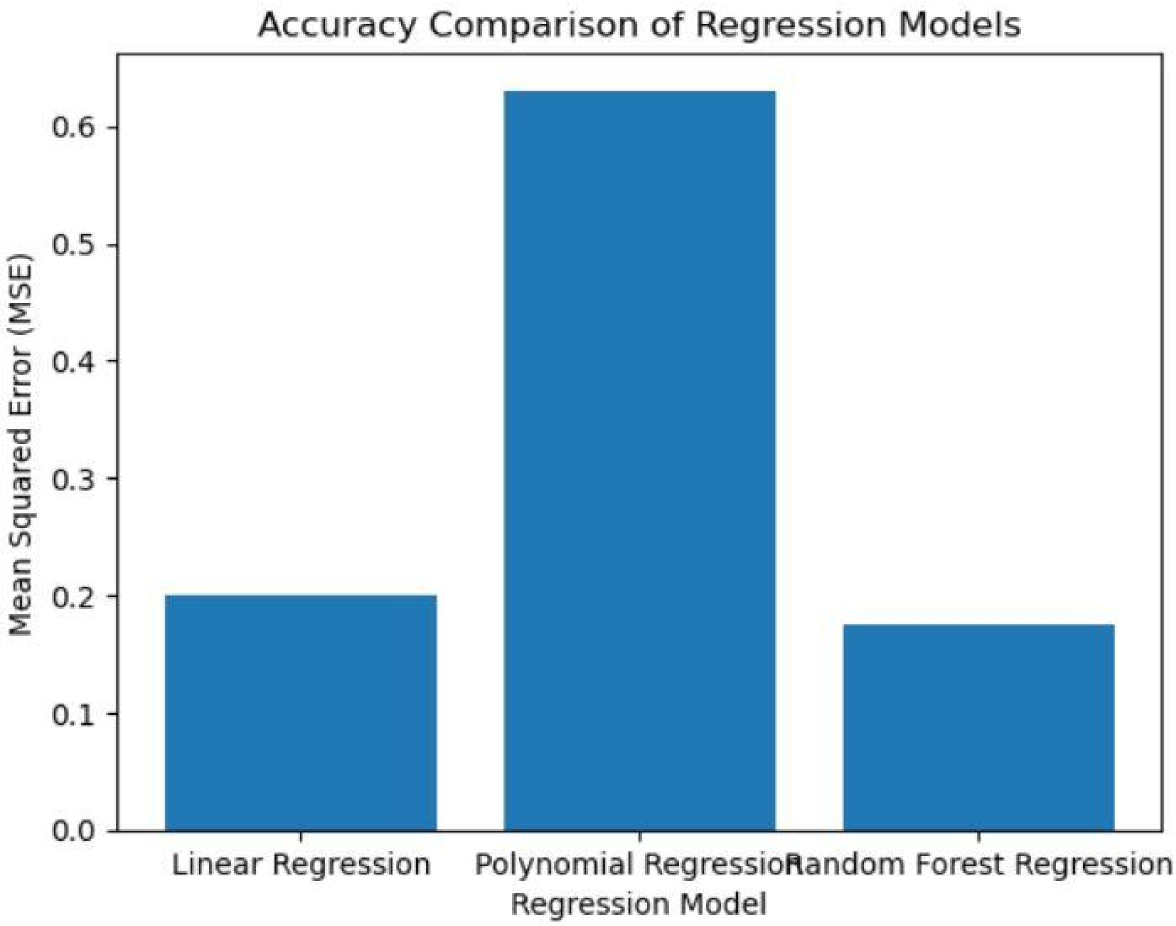
Combining both plots into a single subplot() using Matplotlib.pyplot

```
plt.plot(X_train.flatten(), y_train.flatten(), label='Actual')
plt.plot(X_train.flatten(), model_lr.predict(X_train), color='red', label='Linear Regression')
plt.plot(X_train.flatten(), model_poly.predict(X_train_poly), color='green', label='Polynomial Regression')
plt.plot(X_train.flatten(), model_rf.predict(X_train), color='blue', label='Random Forest Regression')

plt.plot(X_test.flatten(), y_pred_lr, color='red', label='Linear Regression')
plt.plot(X_test.flatten(), y_pred_poly, color='green', label='Polynomial Regression')
plt.plot(X_test.flatten(), y_pred_rf, color='blue', label='Random Forest Regression')

plt.plot(X_test.flatten(), y_test, label='Testing Data')
plt.xlabel('Year')
plt.xticks(fontsize=8, rotation=90)
plt.ylabel('Temperature')
plt.title('Training Data with Models')
plt.legend(fontsize=7)
plt.show()
```



Using Matplotlib’s Bar plot to compare the model accuracies

```
models = ['Linear Regression', 'Polynomial Regression', 'Random Forest Regression']
mse_scores = [mse_lr, mse_poly, mse_rf]

plt.bar(models, mse_scores)
plt.xlabel('Regression Model')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Accuracy Comparison of Regression Models')
plt.show()
```




```
Linear regression RMSE: 0.4476893706810094
Polynomial regression RMSE: 0.7939734279753918
Random forest regression RMSE: 0.41937382271301016
```

ML Model Development Using SARIMAX

SARIMAX stands for ***Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors*** and is one of multiple Time Series Forecasting models available from Python's *statsmodels* module.

Time Series Forecasting refers to "the task of predicting future values based on historical data" (Pierre, S., updated 2022), and it has been used across industries for weather, sales numbers and stock prices forecasting.

Treating our data as time series data allowed us to better interpret it as a sequence of variations that occurred over the years observed in our dataset. The variations depend on time, so as the time increases these variations will take place, whether by an increase, decrease or neutral change in the temperatures observed (Verma, Y., 2021).

We decided to implement the SARIMAX model to our data to perform a temperature change forecast and to compare its performance to the previously used machine learning models where we did not read the data as time series data.

Importing the library from Python's statsmodels module

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Transforming 'Year' column values to integer data type

```
df3['Year'] = df3['Year'].astype(int)
```

Train Test Split

```
train_data = df3[df3['Year'] < 1992]['Total_Temperature']
test_data = df3[df3['Year'] >= 1992]['Total_Temperature']
```

Our training and test sets were split by years. The training data will be composed of the data referring to the years before 1992, while the test data will be composed of the data referring to the year of and after 1992.

Training our SARIMAX model

```
model = SARIMAX(train_data, order=(1, 1, 1), seasonal_order=(0, 1, 1, 12))
results = model.fit()
```

Predicting future values (30 years ahead) using the trained model

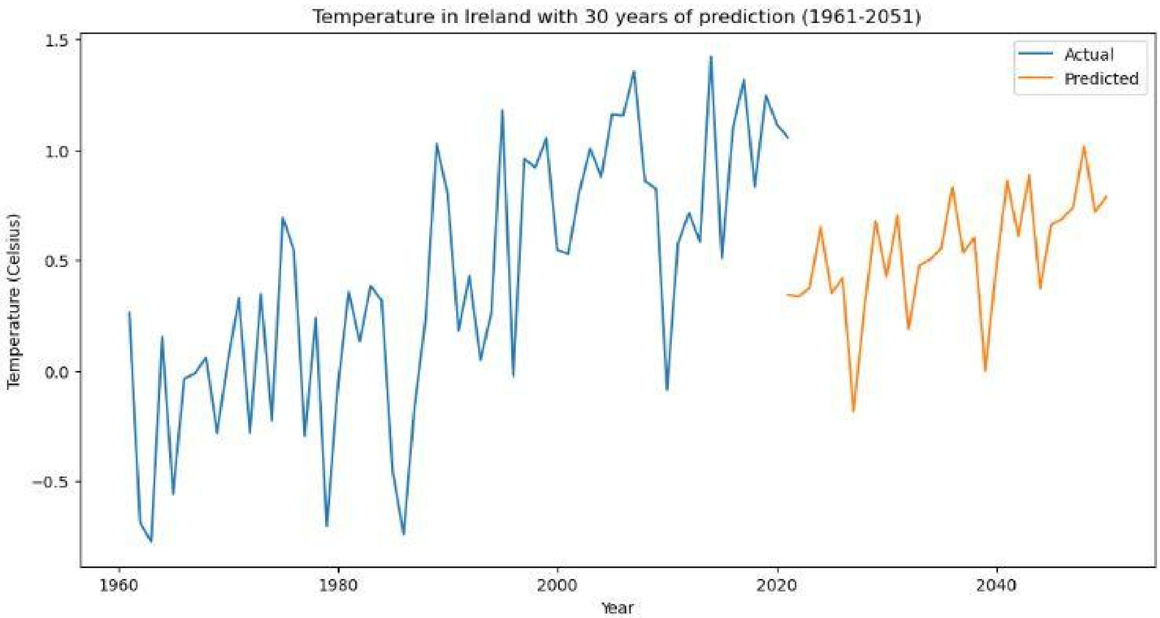
```
start = len(train_data)
end = len(train_data) + 29 # 30 years ahead
predictions = results.predict(start=start, end=end, dynamic=False)
```

Plotting the actual vs. predicted future values of the model

```
plt.figure(figsize=(12,6))
plt.plot(df3['Year'], df3['Total_Temperature'], label='Actual')
plt.plot(range(2021, 2051), predictions, label='Predicted')
plt.title('Temperature in Ireland with 30 years of prediction (1961-2051)')
plt.xlabel('Year')
plt.ylabel('Temperature (Celsius)')
plt.legend()
plt.show()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            4     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.38332D-01    |proj g|=  8.46990D-02

At iterate    5    f=  6.02087D-01    |proj g|=  2.42958D-01

At iterate   10    f=  5.74481D-01    |proj g|=  1.20834D-02

At iterate   15    f=  5.71351D-01    |proj g|=  4.31977D-03

At iterate   20    f=  5.71125D-01    |proj g|=  3.70585D-03

At iterate   25    f=  5.71095D-01    |proj g|=  4.88259D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    4     25     39      1     0     0   4.883D-05   5.711D-01
  F =   0.57109513759617758
```

Output



Actual vs. Predicted Future Data (30 Years)

Calculating the Mean Squared Error (MSE) of our SARIMAX Model

```
mse_sarimax30 = mean_squared_error(test_data, predictions)
print(f"Sarimax 30 years MSE: {mse_sarimax30}")
```

```
Sarimax 30 years MSE: 0.19826890124815194
```

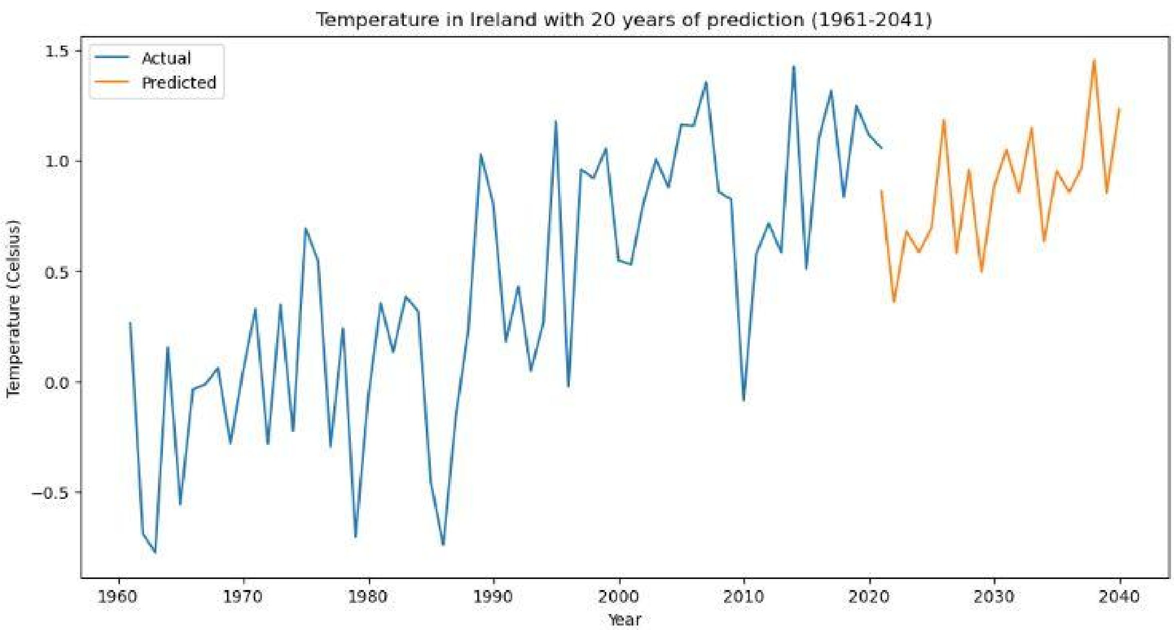
Our SARIMAX model performed with a Mean Squared Error score of 0.198

Using different training and test sets to perform predictions of 20 years in the future

```
train_data = df3[df3['Year'] < 2002]['Total_Temperature']
test_data = df3[df3['Year'] >= 2002]['Total_Temperature']

# Train the SARIMA model
model = SARIMAX(train_data, order=(1, 1, 1), seasonal_order=(0, 1, 1, 12))
results = model.fit()

# Predict future values using the trained model
start = len(train_data)
end = len(train_data) + 19 # 20 years ahead
predictions = results.predict(start=start, end=end, dynamic=False)

# Plot the predicted values against the actual values
plt.figure(figsize=(12,6))
plt.plot(df3['Year'], df3['Total_Temperature'], label='Actual')
plt.plot(range(2021, 2041), predictions, label='Predicted')
plt.title('Temperature in Ireland with 20 years of prediction (1961-2041)')
plt.xlabel('Year')
plt.ylabel('Temperature (Celsius)')
plt.legend()
plt.show()
```

Here we split our training set up to the year of 2002 and the test set from the year 2002 and further. Additionally, we will perform predictions for 20 years in the future instead of 30, and see if our predictions are better.

Plot of the actual and predicted future data (20 years)



Actual vs. Predicted Future Data (20 Years)

Calculating the Mean Squared Error (MSE) of our SARIMAX Model with predictions of 20 years

```
mse_sarimax20 = mean_squared_error(test_data, predictions)
print(f"Sarimax 20 years MSE: {mse_sarimax20}")
```

```
Sarimax 20 years MSE: 0.11186351786036959
```

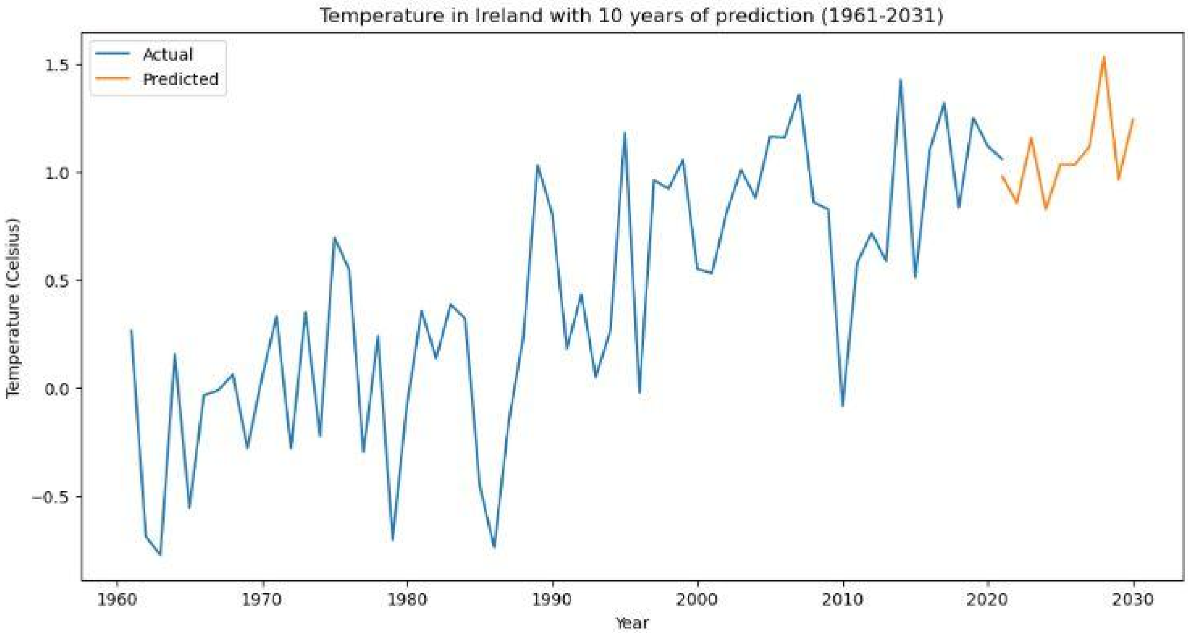
Our SARIMAX model performed with a Mean Squared Error score of 0.112 when using a different train test split and performing a prediction of 20 years.

Using different training and test sets to perform predictions of 10 years in the future

```
train_data = df3[df3['Year'] < 2012]['Total_Temperature']
test_data = df3[df3['Year'] >= 2012]['Total_Temperature']

# Train the SARIMA model
model = SARIMAX(train_data, order=(1, 1, 1), seasonal_order=(0, 1, 1, 12))
results = model.fit()

# Predict future values using the trained model
start = len(train_data)
end = len(train_data) + 9 # 10 years ahead
predictions = results.predict(start=start, end=end, dynamic=False)

# Plot the predicted values against the actual values
plt.figure(figsize=(12,6))
plt.plot(df3['Year'], df3['Total_Temperature'], label='Actual')
plt.plot(range(2021, 2031), predictions, label='Predicted')
plt.title('Temperature in Ireland with 10 years of prediction (1961-2031)')
plt.xlabel('Year')
plt.ylabel('Temperature (Celsius)')
plt.legend()
plt.show()
```

Here we split our training set up to the year of 2012 and the test set from the year 2012 and further. Additionally, we will perform predictions for 10 years in the future, and see if our predictions are better.

Plot of the actual and predicted future data (10 years)



Actual vs. Predicted Future Data (10 Years)

Calculating the Mean Squared Error (MSE) of our SARIMAX Model with predictions of 10 years

```
mse_sarimax10 = mean_squared_error(test_data, predictions)
print(f"Sarimax 10 years MSE: {mse_sarimax10}")
```

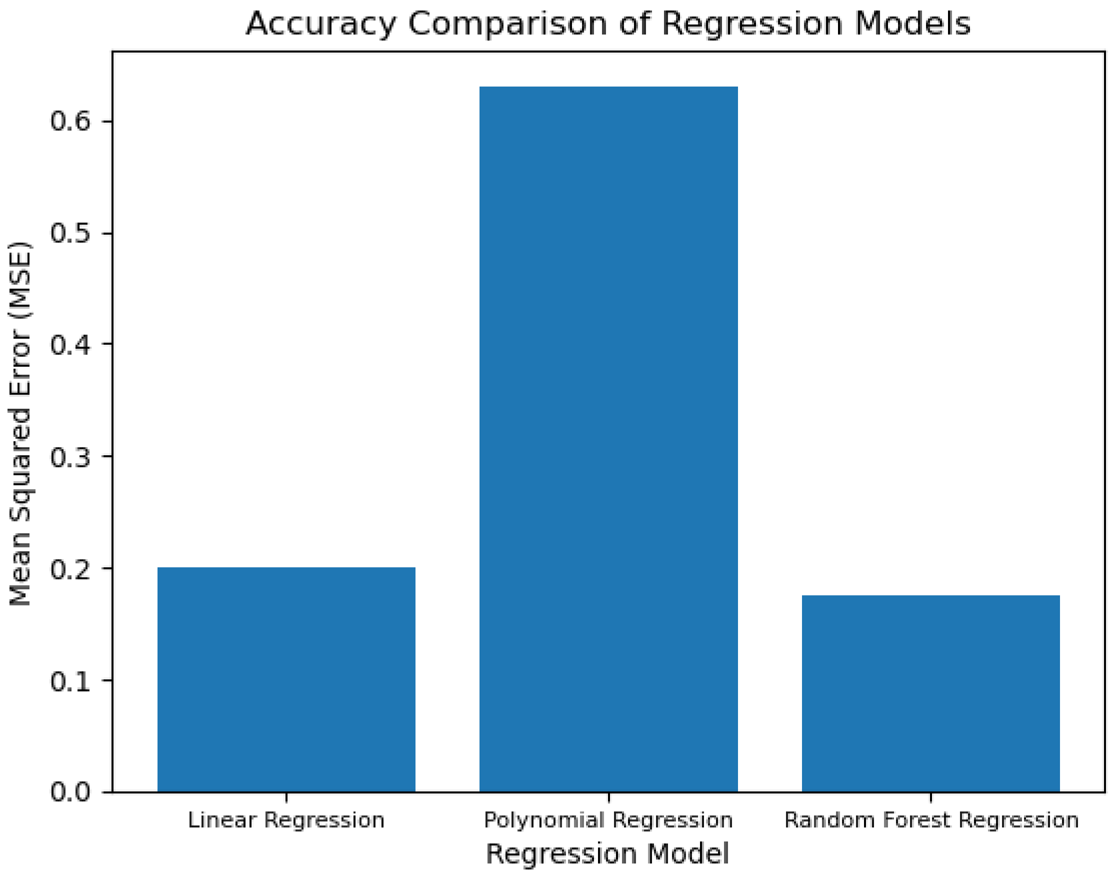
```
Sarimax 10 years MSE: 0.06169775253320307
```

Our SARIMAX model performed with a Mean Squared Error score of 0.061 when using a different train test split and performing a prediction of 10 years.

5. Evaluation

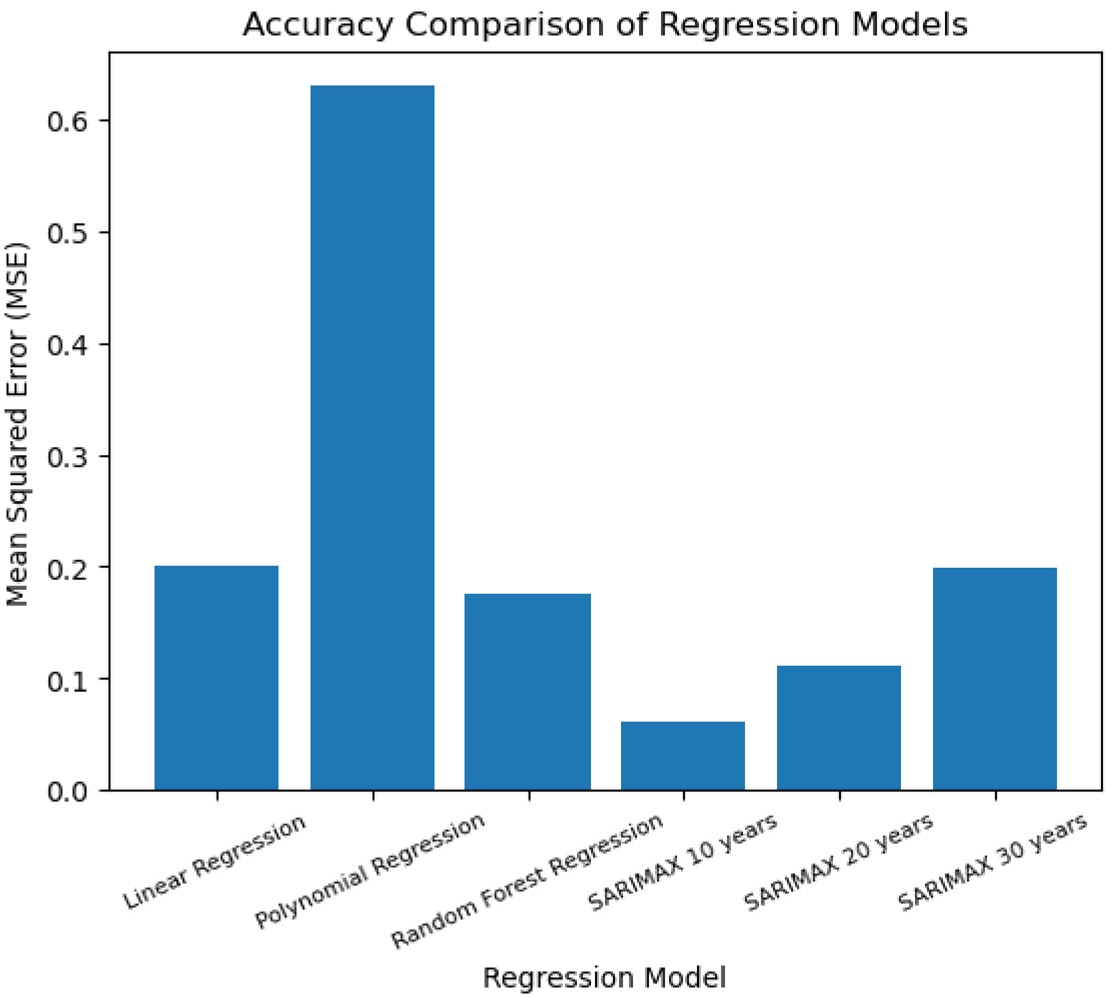
Comparing the accuracy of our Regression models using the Mean Squared Error (MSE)

```
models = ['Linear Regression', 'Polynomial Regression', 'Random Forest Regression']
mse_scores = [mse_lr, mse_poly, mse_rf]

plt.bar(models, mse_scores)
plt.xlabel('Regression Model')
plt.xticks(fontsize=8)
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Accuracy Comparison of Regression Models')
plt.show()
```



Comparing the accuracy of all of our models using the Mean Squared Error (MSE) and plotting their MSE scores using Matplotlib's Bar plot

```
models = ['Linear Regression', 'Polynomial Regression', 'Random Forest Regression', 'SARIMAX 10 years', 'SARIMAX 20 years', 'SARI
mse_scores = [mse_lr, mse_poly, mse_rf, mse_sarimax10, mse_sarimax20, mse_sarimax30] #mse_random_forest???

plt.bar(models, mse_scores)
plt.xlabel('Regression Model')
plt.xticks(fontsize=8, rotation=25)
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Accuracy Comparison of Regression Models')
plt.show()
```


Accuracy Comparison of Regression Models
Mean Squared Error (MSE)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Linear Regression
Polynomial Regression
Random Forest Regression
SARIMAX 10 years
SARIMAX 20 years
SARIMAX 30 years
Regression Model

2.2. Data Understanding - Historical Climate Data for Dublin

Historical Climate Data Dublin

Import the libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import geopandas as gpd
pd.set_option('display.max_columns', None)
import warnings
warnings.filterwarnings('ignore')
```

Import the data

```
dfnew = pd.read_excel('../Project/Data/mly532-1.xlsx')
```

Preview of the dataset

```
dfnew
```

	year	month	meant	maxtp	mintp	mnmax	mnmin	rain	gmin	wdsp	maxgt	sun
0	1941	11	6.9	14.0	-3.1	9.9	3.9	67.2	-5.7	12.0		56.1
1	1941	12	6.5	12.7	-3.6	9.1	3.9	41.7	-7.6	12.5		46.1
2	1942	1	4.3	11.9	-3.1	6.9	1.7	91.9	-9.5	13.1		72.8
3	1942	2	2.9	11.6	-4.3	5.8	0.0	25.8	-10.7	9.0		51.4
4	1942	3	6.3	16.2	-6.1	9.4	3.2	76.4	-8.3	10.7		73.9
...	...	...	...	...	...	...	...	...	...	...	...	...
972	2022	11	8.8	16.5	-0.8	11.7	6.0	46.1	-3.5	9.9	51	84.5
973	2022	12	4.4	14.7	-4.2	7.3	1.6	74.0	-6.2	8.6	39	76.2
974	2023	1	6.0	13.6	-4.8	8.8	3.3	41.2	-9.2	10.0	47	92.1
975	2023	2	7.2	14.2	-4.0	10.3	4.1	16.2	-8.3	9.3	41	67.7
976	2023	3	7.0	16.2	-4.3	10.4	3.5	119.0	-8.3	9.8	40	96.1

977 rows × 12 columns

Source: https://www.met.ie/climate/available-data/historical-data

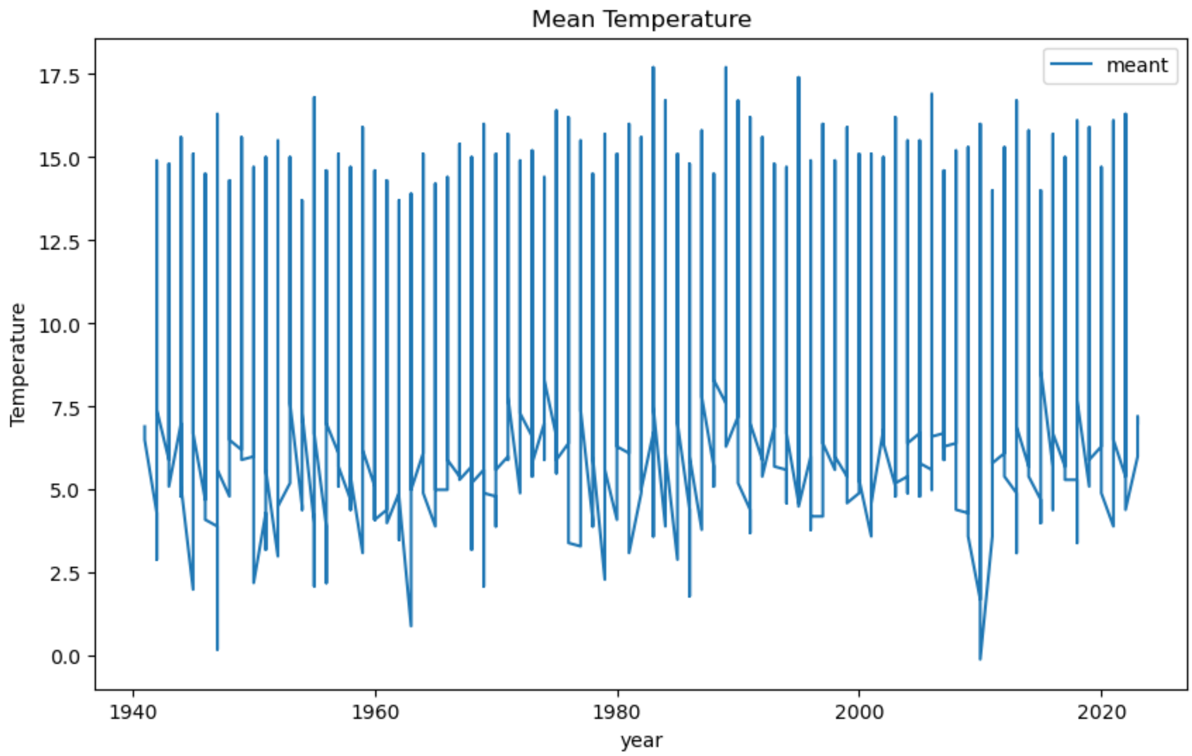
Our second dataset ‘**mly532-1.xlsx**’ will deal with monthly historical climate data in Ireland, showing the minimum, maximum, and mean of temperatures, as well as other variables such as Sunshine duration (hours) and Mean Wind Speed. This dataset was sourced from Met Éireann, with data gathered from the Dublin Airport station.

Variables List:

- year: Year
- month: Month
- rain: Precipitation Amount (mm)
- maxtp: Maximum Air Temperature (°C)
- mintp: Minimum Air Temperature (°C)
- mnmax: Mean Maximum Temperature (°C)
- mnmin: Mean Minimum Temperature (°C)
- gmin: Grass Minimum Temperature (°C)
- wdsp: Mean Wind Speed (knot)
- maxgt: Highest Gust (knot)
- sun: Sunshine duration (hours)

Plotting the mean temperatures of Dublin over the years

```
plt.figure(figsize=(10, 6))
plt.plot(dfnew['year'], dfnew['meant'], label='meant')
plt.xlabel('year')
plt.ylabel('Temperature')
plt.title('Mean Temperature')
plt.legend()
plt.show()
```



3.2. Data Preparation

Check for missing data

```
missing_dfnew = dfnew.isna().sum()

print(missing_dfnew)
```

```
year      0
month     0
meant     0
maxtp     0
mintp     0
mnmax     0
mnmin     0
rain      0
gmin      0
wdsp      0
maxgt     0
sun       0
dtype: int64
```

None of the columns or rows present missing data for this dataset.

Filtering the dataset to exclude the years of 1941 and 2023

```
dff = dfnew[(dfnew['year'] != 1941) & (dfnew['year'] != 2023)]
dff
```

	year	month	meant	maxtp	mintp	mnmax	mnmin	rain	gmin	wdsp	maxgt	sun
2	1942	1	4.3	11.9	-3.1	6.9	1.7	91.9	-9.5	13.1		72.8
3	1942	2	2.9	11.6	-4.3	5.8	0.0	25.8	-10.7	9.0		51.4
4	1942	3	6.3	16.2	-6.1	9.4	3.2	76.4	-8.3	10.7		73.9
5	1942	4	8.4	16.2	0.8	11.9	4.9	36.9	-0.4	15.1		185.4
6	1942	5	10.4	20.9	1.8	14.4	6.3	108.2	-0.7	12.0		195.9
...	...	...	...	...	...	...	...	...	...	...	...	...
969	2022	8	15.9	26.3	6.0	21.3	10.5	34.6	0.9	7.3	30	222.0
970	2022	9	13.0	20.1	3.1	17.2	8.8	127.9	-0.3	8.7	39	112.9
971	2022	10	12.0	18.0	1.3	15.6	8.4	106.8	-3.3	9.6	41	115.7
972	2022	11	8.8	16.5	-0.8	11.7	6.0	46.1	-3.5	9.9	51	84.5
973	2022	12	4.4	14.7	-4.2	7.3	1.6	74.0	-6.2	8.6	39	76.2

972 rows × 12 columns

This is done in order to improve the quality of the data visualizations later in this report.

Checking the maximum and minimum temperatures

```
max_tempf = dff['maxtp'].max()
min_tempf = dff['mintp'].min()
print(f"Max temperature: {max_tempf}, Min temperature: {min_tempf}")
```

```
Max temperature: 29.1, Min temperature: -12.2
```

Checking the maximum and minimum mean temperatures

```
maxm_tempf = dff['meant'].max()
minm_tempf = dff['meant'].min()
print(f"Max mean temperature: {maxm_tempf}, Min mean temperature: {minm_tempf}")
```

```
Max mean temperature: 17.7, Min mean temperature: -0.1
```

Filtering the rows that contain the maximum and minimum temperature values

```
max_temp_rows = dff[dff['maxtp'] == max_tempf]
min_temp_rows = dff[dff['mintp'] == min_tempf]
```

Extracting the dates in which occurred the maximum and minimum temperature values, then print the results

```
max_temp_dates = [(row['year'], row['month']) for i, row in max_temp_rows.iterrows()]
min_temp_dates = [(row['year'], row['month']) for i, row in min_temp_rows.iterrows()]

print(f"Max temperature of {max_tempf} was recorded in:")
for date in max_temp_dates:
    print(f"{date[0]}-{date[1]}")

print(f"Min temperature of {min_tempf} was recorded in:")
for date in min_temp_dates:
    print(f"{date[0]}-{date[1]}")
```

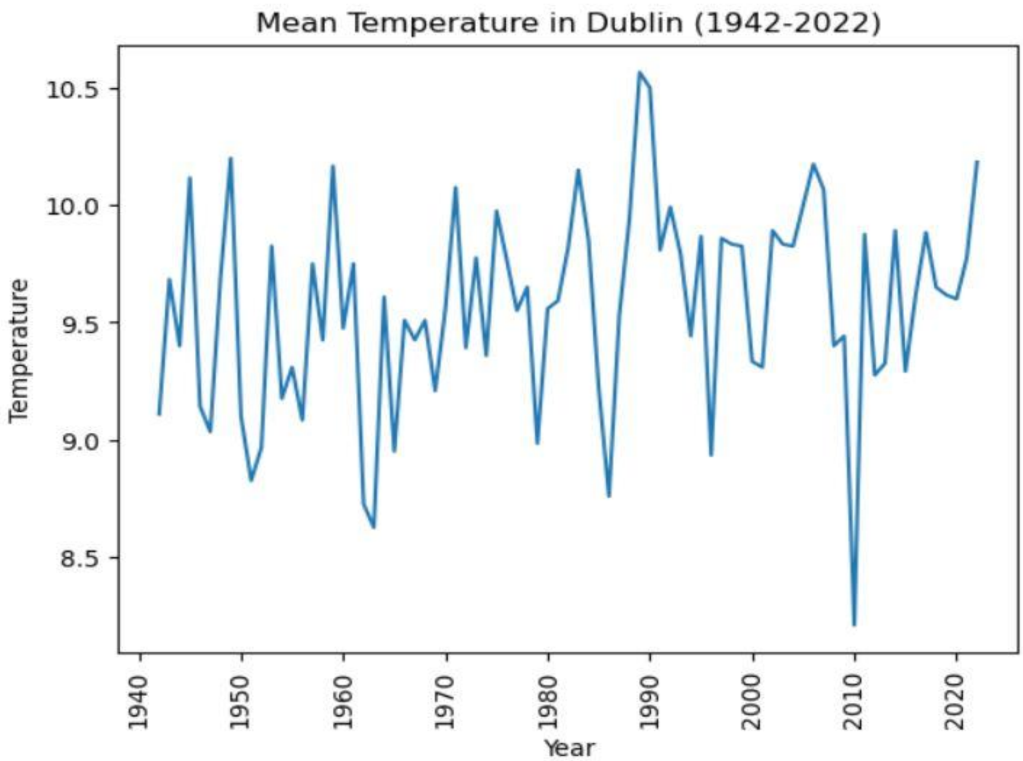
```
Max temperature of 29.1 was recorded in:
2022-7
Min temperature of -12.2 was recorded in:
2010-12
```

Calculating the mean temperature for all years

```
mean_t = dff.groupby('year')['meant'].mean()
```

Plotting the mean temperature in Dublin

```
plt.plot(mean_t.index, mean_t.values)
plt.title('Mean Temperature in Dublin (1942-2022)')
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.xticks(rotation=90)
plt.show()
```



Renaming the columns

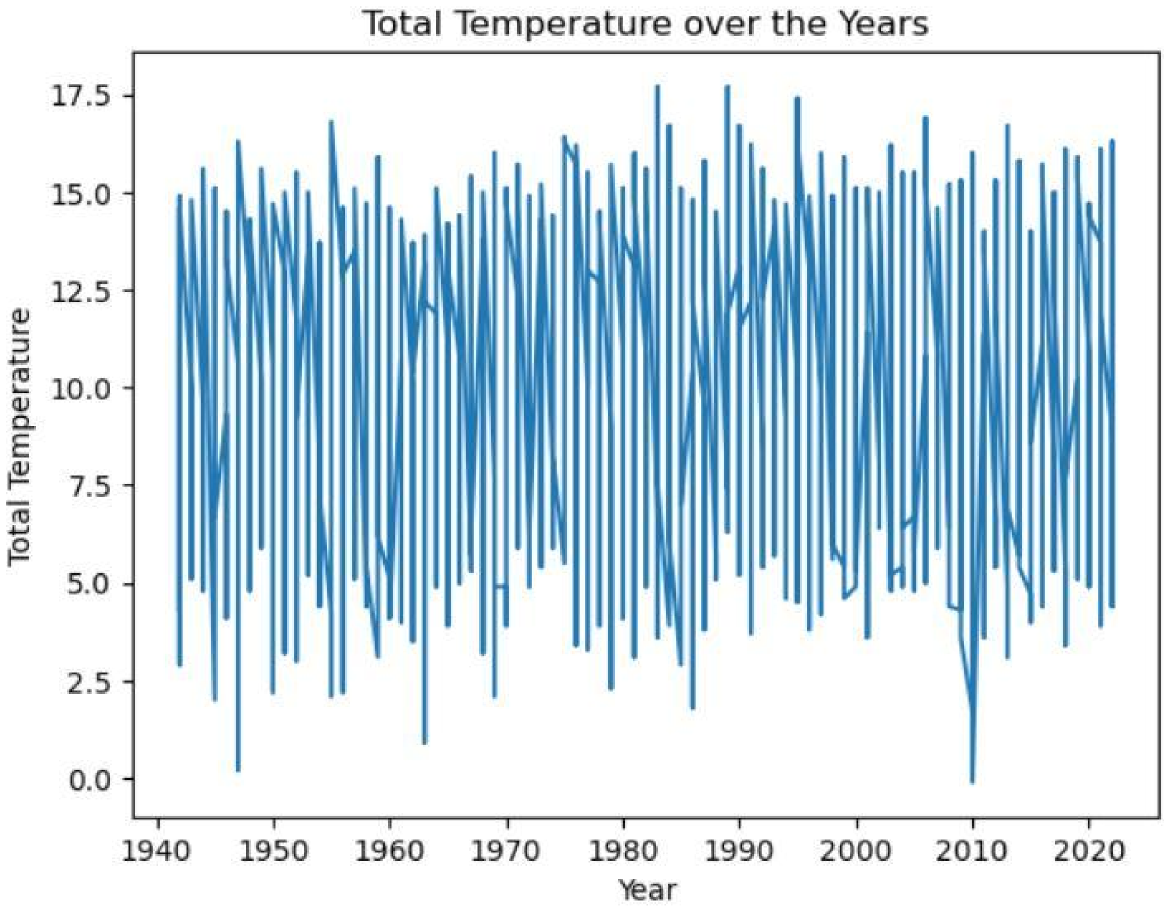
```
dff = dff.rename(columns={"year": "Year", "meant": "Total_Temperature"})
dff = dff[["Year", "Total_Temperature"]]
dff["Countries"] = "Ireland"
dff = dff[["Countries", "Year", "Total_Temperature"]]
dff = dff.sort_values("Year")
dff
```

	Countries	Year	Total_Temperature
2	Ireland	1942	4.3
7	Ireland	1942	13.1
3	Ireland	1942	2.9
4	Ireland	1942	6.3
6	Ireland	1942	10.4
...	...	...	...
970	Ireland	2022	13.0
971	Ireland	2022	12.0
966	Ireland	2022	11.9
967	Ireland	2022	13.6
973	Ireland	2022	4.4

972 rows × 3 columns

Plotting the new Total Temperature over the years

```
plt.plot(dff["Year"], dff["Total_Temperature"])

# Adding labels and title to the plot
plt.xlabel("Year")
plt.ylabel("Total Temperature")
plt.title("Total Temperature over the Years")

# Display the plot
plt.show()
```



Calculating the Minimum, Maximum and Average Temperatures

```
max_temp2 = df4['Total_Temperature'].max()
min_temp2 = df4['Total_Temperature'].min()
print(f"Max temperature: {max_temp}, Min temperature: {min_temp}")
```

```
Max temperature: 1.424, Min temperature: -0.776
```

```
avg_temp2 = df4['Total_Temperature'].mean()
print(f"Average temperature: {avg_temp}")
```

```
Average temperature: 0.41242622950819674
```

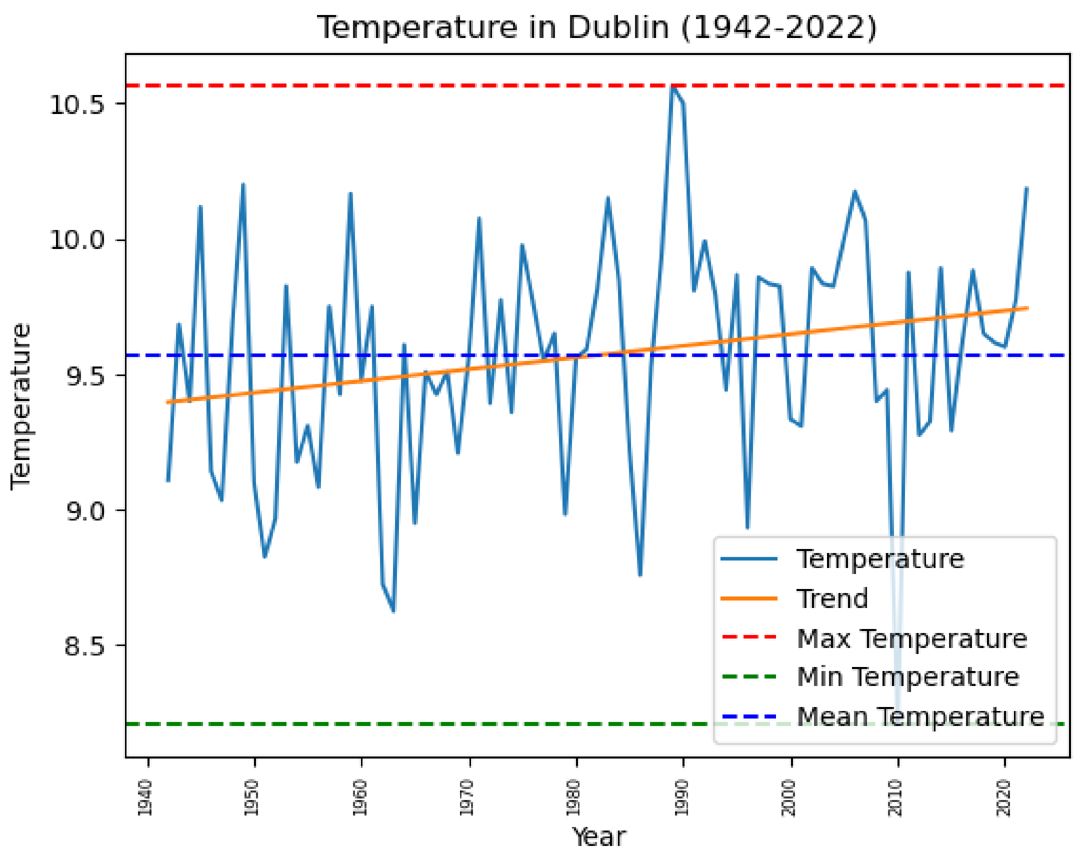
4.2. Modelling

Importing the libraries

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split
```

Plotting the original data with Maximum, Minimum and Mean Temperatures + Trend for Dublin using Matplotlib

```
plt.plot(data2['Year'], data2['Total_Temperature'], label='Temperature')
plt.plot(data2['Year'], trendx2, label='Trend')

plt.axhline(y=max_temp2, color='r', linestyle='--', label='Max Temperature')
plt.axhline(y=min_temp2, color='g', linestyle='--', label='Min Temperature')
plt.axhline(y=avg_temp2, color='b', linestyle='--', label='Mean Temperature')

plt.title('Temperature in Dublin (1942-2022)')
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.legend()
plt.xticks(fontsize=6, rotation=90)
plt.show()
```



Train Test Split

Using the ‘reshape()’ method

```
X2 = df4['Year'].values.reshape(-1, 1)
y2 = df4['Total_Temperature'].values.reshape(-1, 1)
```

Train Test split using scikit-learn’s ‘train_test_split’ method

```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.2, shuffle=False)
```

ML Model Development

Linear Regression

Creating the Linear Regression 'model_lr2' model with scikit-learn 'LinearRegression' and fitting the model with 'fit()' method using X_train2 and y_train2 values.

```
model_lr2 = LinearRegression()
model_lr2.fit(X_train2, y_train2)
```

```
LinearRegression()
```

Creating the predicted values 'y_pred_lr' on the test data

```
y_pred_lr2 = model_lr2.predict(X_test2)
```

Calculating the Mean Squared Error (MSE) for our Linear Regression model using 'mean_squared_error' function from scikit-learn.metrics module

```
mse_lr2 = mean_squared_error(y_test2, y_pred_lr2)
print(f"Linear Regression MSE: {mse_lr2}")
```

```
Linear Regression MSE: 0.263363596489173
```

Polynomial Regression

Transforming the features into polynomial features using scikit-learn 'PolynomialFeatures' and 'poly.fit_transform' methods.

Creating the Polynomial Regression 'model_poly2' Model with scikit-learn 'LinearRegression' and fitting the model with X_train_poly2 and y_train_poly2 values

Creating the predicted values 'y_pred_poly2' on the test data

Calculating the Mean Squared Error (MSE) for our Polynomial Regression Model using 'mean_squared_error' function from scikit-learn.metrics module

```python
poly2 = PolynomialFeatures(degree=2)
X_poly2 = poly2.fit_transform(X2)
X_train_poly2 = poly2.fit_transform(X_train2)
X_test_poly2 = poly2.fit_transform(X_test2)

# Train the model
model_poly2 = LinearRegression()
model_poly2.fit(X_train_poly2, y_train2)

# Predict on the test data
y_pred_poly2 = model_poly2.predict(X_test_poly2)

# Calculate the mean squared error
mse_poly2 = mean_squared_error(y_test2, y_pred_poly2)
print(f"Polynomial Regression MSE: {mse_poly2}")
```

```
Polynomial Regression MSE: 0.31387182622314486
```

Random Forest Regression

Creating the Random Forest Regression ‘model_rf2’ Model with scikit-learn ‘RandomForestRegressor’ and fitting the model with ‘fit()’ method using X_train2 and y_train2 values.

Creating the predicted values ‘y_pred_rf’ on the test data

Calculating the Mean Squared Error (MSE) for our Random Forest Regression Model using ‘mean_squared_error’ function from scikit-learn.metrics module

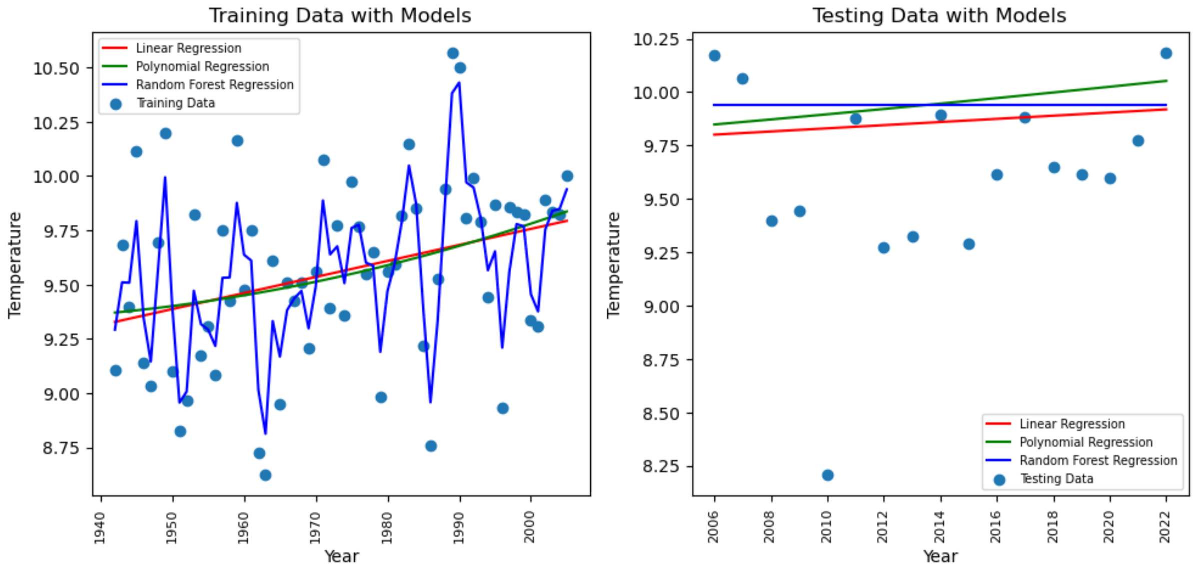
```python
model_rf2 = RandomForestRegressor(n_estimators=100, random_state=42)
model_rf2.fit(X_train2, y_train2.ravel())

# Predict on the test data
y_pred_rf2 = model_rf2.predict(X_test2)

# Calculate the mean squared error
mse_rf2 = mean_squared_error(y_test2, y_pred_rf2)
print(f"Random Forest Regression MSE: {mse_rf2}")
```

```
Random Forest Regression MSE: 0.31414560130719277
```

We can see that the Linear Regression model performed better with lower error margin if compared to the other two models, with an MSE value of 0.26. However, similar to our Ireland dataset, we will be able to visualize the predictions in plots and better understand the predictions made by each, and conclude which model best fits our needs.

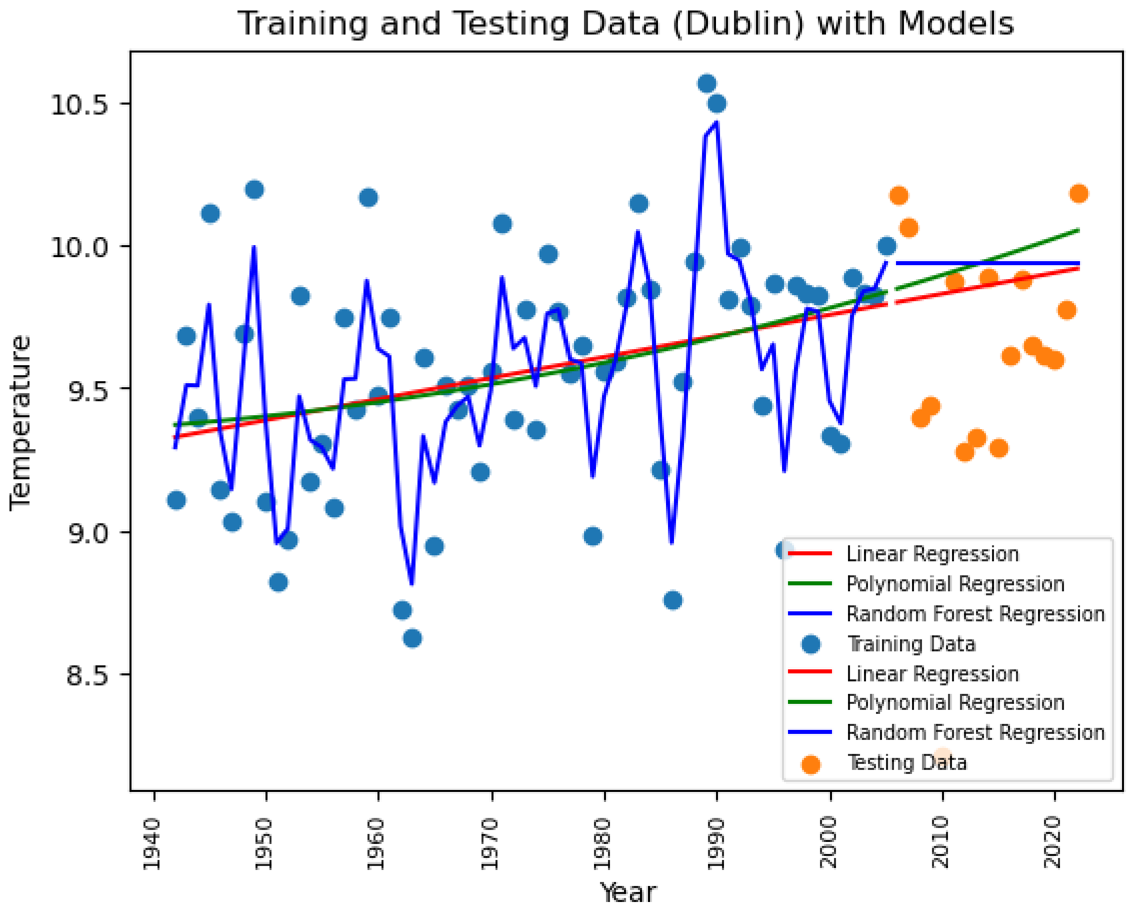
Training Data vs. Model Predictions using matplotlib.pyplot subplot()

```
plt.figure(figsize=(10, 5))

# Training Data with Models
plt.subplot(1, 2, 1)
#plt.plot(X_train.reshape(-1), y_train.reshape(-1), color='#F0F0F0', label='Actual')
plt.plot(X_train2.flatten(), model_lr2.predict(X_train2), color='red', label='Linear Regression')
plt.plot(X_train2.flatten(), model_poly2.predict(X_train_poly2), color='green', label='Polynomial Regression')
plt.plot(X_train2.flatten(), model_rf2.predict(X_train2), color='blue', label='Random Forest Regression')
plt.scatter(X_train2.flatten(), y_train2, label='Training Data')
plt.xlabel('Year')
plt.xticks(fontsize=8, rotation=90)
plt.ylabel('Temperature')
plt.title('Training Data with Models')
plt.legend(fontsize=7)

# Testing Data with Models
plt.subplot(1, 2, 2)
#plt.plot(X_test.reshape(-1), y_test.reshape(-1), color='#F0F0F0', label='Actual')
plt.plot(X_test2.flatten(), y_pred_lr2, color='red', label='Linear Regression')
plt.plot(X_test2.flatten(), y_pred_poly2, color='green', label='Polynomial Regression')
plt.plot(X_test2.flatten(), y_pred_rf2, color='blue', label='Random Forest Regression')
plt.scatter(X_test2.flatten(), y_test2, label='Testing Data')
plt.xlabel('Year')
plt.xticks(fontsize=8, rotation=90)
plt.ylabel('Temperature')
plt.title('Testing Data with Models')
plt.legend(fontsize=7)

plt.tight_layout()
plt.show()
```

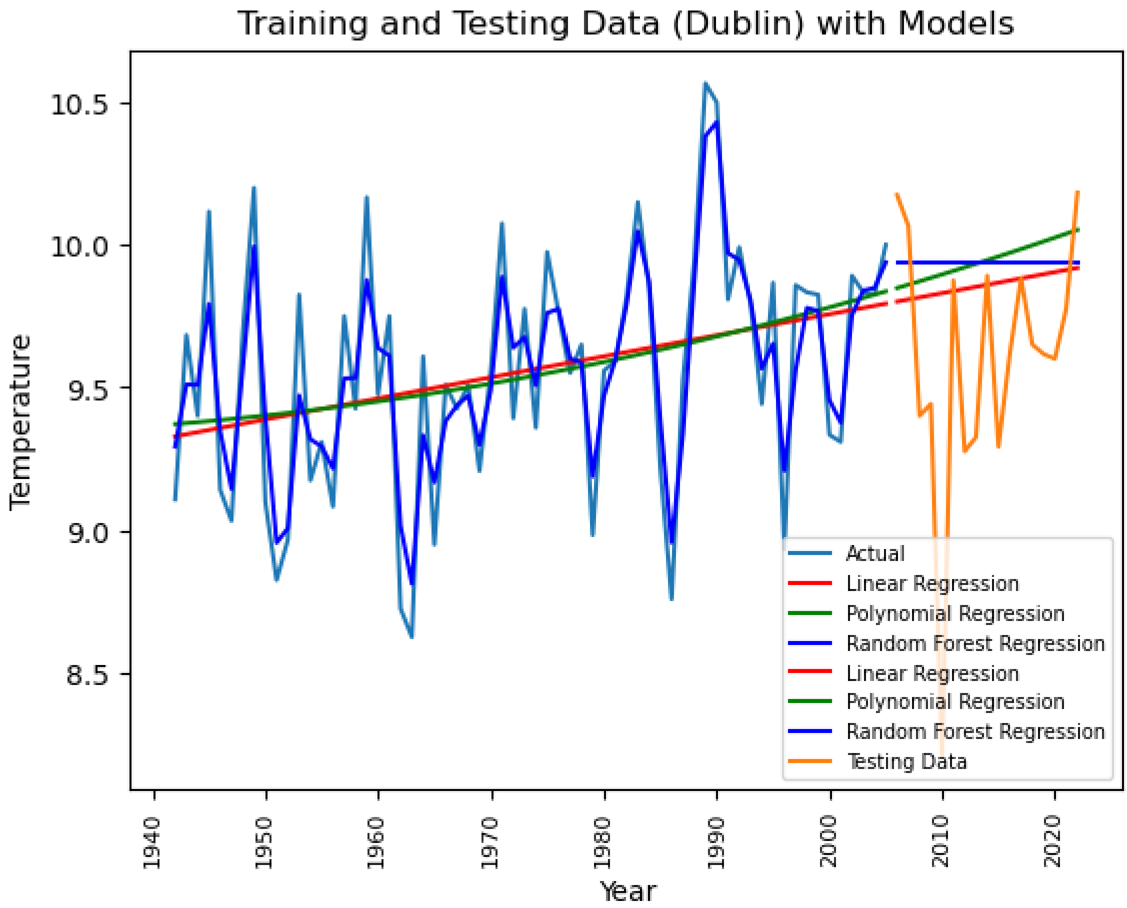


Through this plot we are able to see that the regressor that most accurately captures the information of the data is the Random Forest Regressor. The Linear Regression and Polynomial Regression models better show the tendency for increase in the temperatures. However they do not capture temperature drops seen in the historical data and which can occur in the future.

Plotting the Test Data vs. Model Predictions using matplotlib.pyplot subplot()

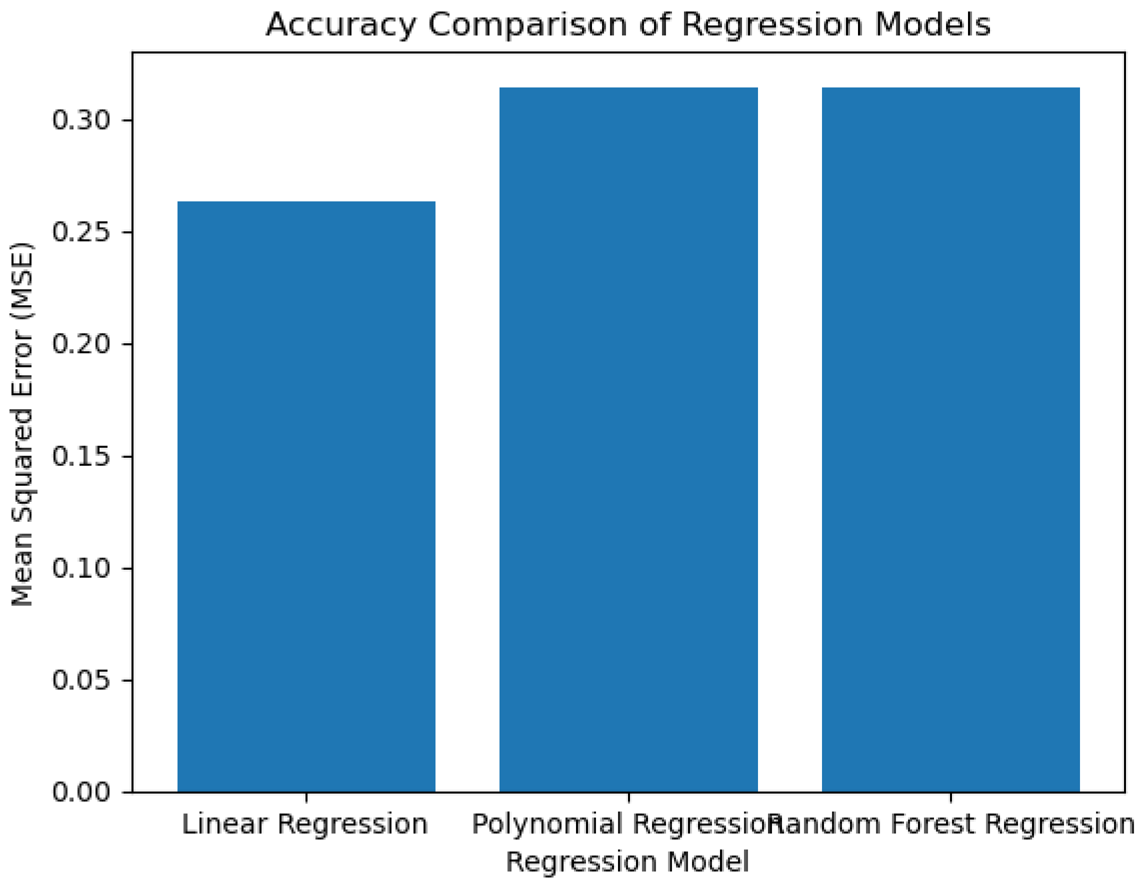


Combining both plots into a single subplot using Matplotlib.pyplot



Using Matplotlib's Bar plot to compare the model accuracies

```
models2 = ['Linear Regression', 'Polynomial Regression', 'Random Forest Regression']
mse_scores2 = [mse_lr2, mse_poly2, mse_rf2]

plt.bar(models2, mse_scores2)
plt.xlabel('Regression Model')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Accuracy Comparison of Regression Models')
plt.show()
```


Accuracy Comparison of Regression Models
Mean Squared Error (MSE)
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Linear Regression
Polynomial Regression
Random Forest Regression
Regression Model

Model Development Using SARIMAX

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

df4['Year'] = df4['Year'].astype(int)

# Split the data into training and testing sets
train_data2 = df4[df4['Year'] < 1992]['Total_Temperature']
test_data2 = df4[df4['Year'] >= 1992]['Total_Temperature']

# Train the SARIMA model
model2 = SARIMAX(train_data2, order=(1, 1, 1), seasonal_order=(0, 1, 1, 12))
results2 = model2.fit()

# Predict future values using the trained model
start2 = len(train_data2)
end2 = len(train_data2) + 30 # 30 years ahead
predictions2 = results2.predict(start=start2, end=end2, dynamic=False)

# Plot the predicted values against the actual values
plt.figure(figsize=(12,6))
plt.plot(df4['Year'], df4['Total_Temperature'], label='Actual')
plt.plot(range(2022, 2053), predictions2, label='Predicted')
plt.title('Temperature in Dublin with 30 years of prediction (1942-2053)')
plt.xlabel('Year')
plt.ylabel('Temperature (Celsius)')
plt.legend()
plt.show()
```

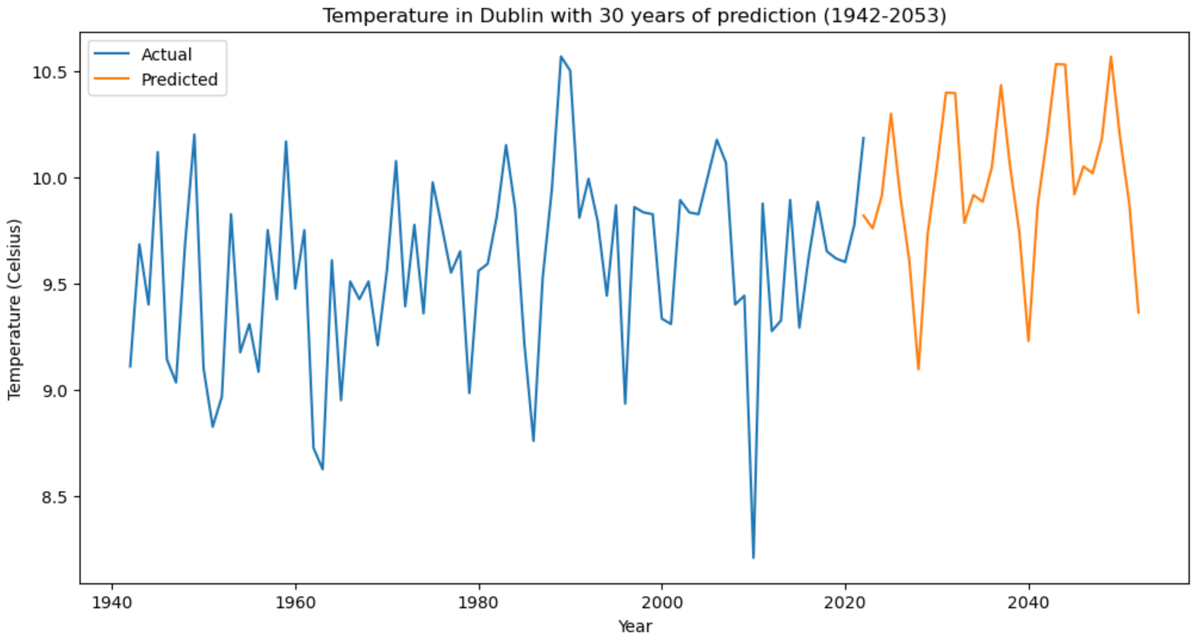
Transforming ‘Year’ column values to integer data type

Train Test Split

Training our SARIMAX model

Predicting future values (30 years ahead) using the trained model

Plotting the actual vs. predicted future values of the model



Calculating the Mean Squared Error (MSE) of our SARIMAX Model

```
mse_sarimax302 = mean_squared_error(test_data2, predictions2)
print(f"Sarimax 30 years MSE: {mse_sarimax302}")
```

```
Sarimax 30 years MSE: 0.36008314375526723
```

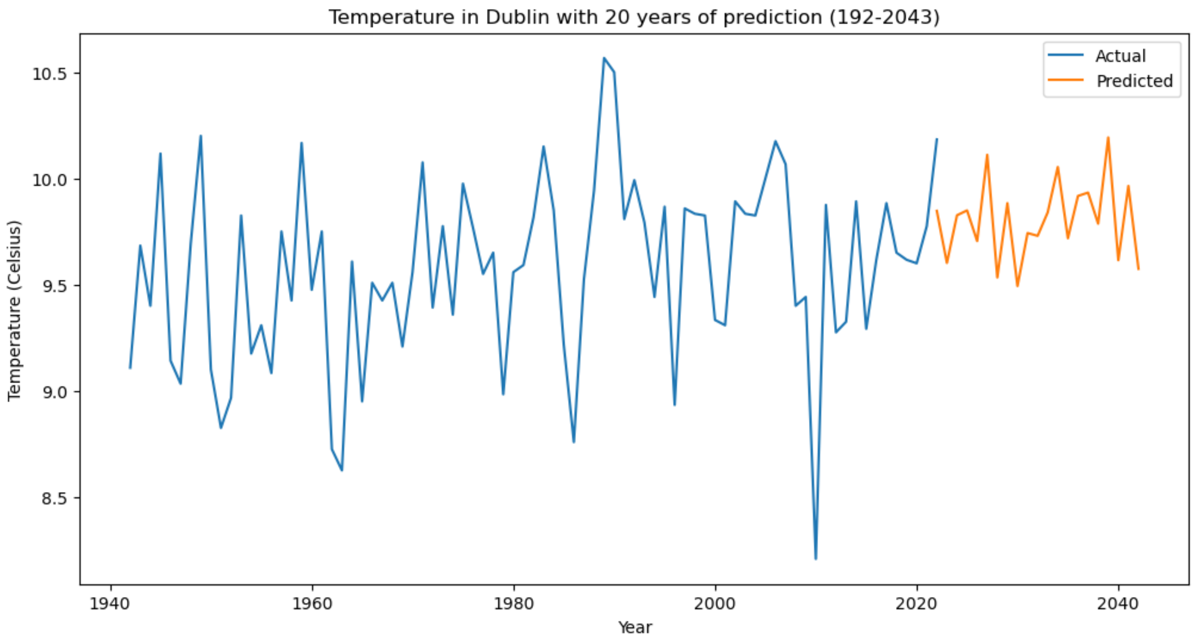
Our model performed with a Mean Squared Error score of 0.36 when performing a prediction for 30 years in the future.

Using different training and test sets to perform predictions of 20 years in the future

```
# Split the data into training and testing sets
train_data2 = df4[df4['Year'] < 2002]['Total_Temperature']
test_data2 = df4[df4['Year'] >= 2002]['Total_Temperature']

# Train the SARIMA model
model2 = SARIMAX(train_data2, order=(1, 1, 1), seasonal_order=(0, 1, 1, 12))
results2 = model2.fit()

# Predict future values using the trained model
start2 = len(train_data2)
end2 = len(train_data2) + 20 # 20 years ahead
predictions2 = results2.predict(start=start2, end=end2, dynamic=False)

# Plot the predicted values against the actual values
plt.figure(figsize=(12,6))
plt.plot(df4['Year'], df4['Total_Temperature'], label='Actual')
plt.plot(range(2022, 2043), predictions2, label='Predicted')
plt.title('Temperature in Dublin with 20 years of prediction (192-2043)')
plt.xlabel('Year')
plt.ylabel('Temperature (Celsius)')
plt.legend()
plt.show()
```

Here we split our training set up to the year of 2002 and the test set from the year 2002 and further. Additionally, we will perform predictions for 20 years in the future instead of 30, and see if our predictions are better.

Plot of the actual and predicted future data (20 years)



Calculating the Mean Squared Error (MSE) of our SARIMAX Model with predictions of 20 years

```
mse_sarimax202 = mean_squared_error(test_data2, predictions2)
print(f"Sarimax 20 years MSE: {mse_sarimax202}")
```

```
Sarimax 20 years MSE: 0.17675543156786794
```

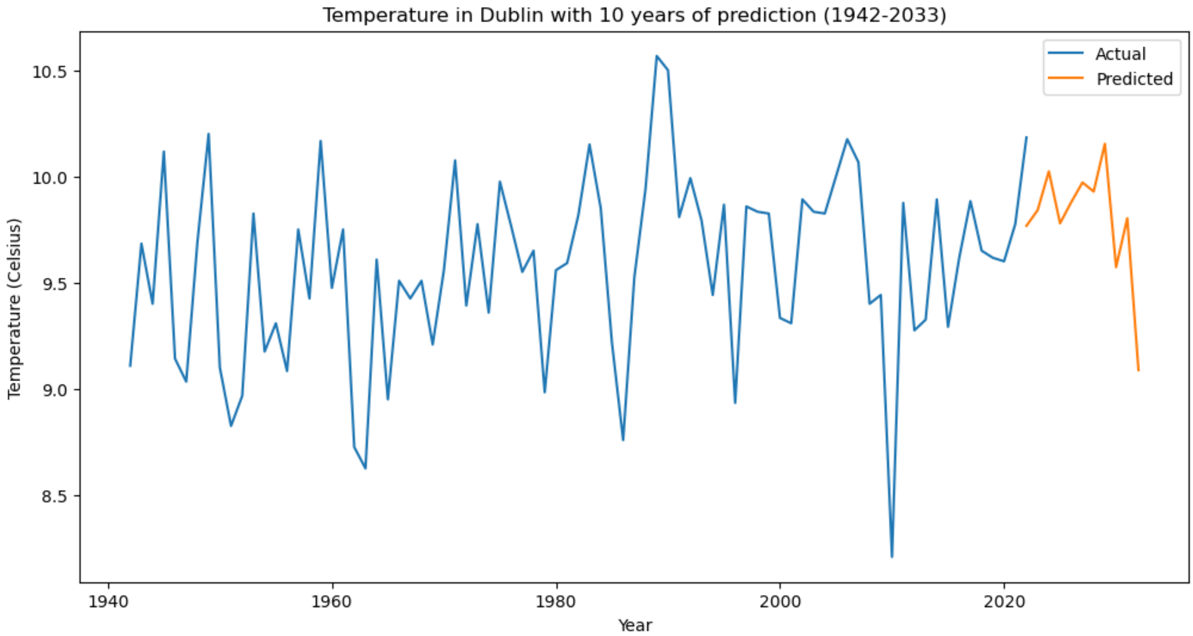
Our SARIMAX model performed with a Mean Squared Error score of 0.176 when using a different train test split and performing a prediction of 20 years.

Using different training and test sets to perform predictions of 10 years in the future

```
# Split the data into training and testing sets
train_data2 = df4[df4['Year'] < 2012]['Total_Temperature']
test_data2 = df4[df4['Year'] >= 2012]['Total_Temperature']

# Train the SARIMA model
model2 = SARIMAX(train_data2, order=(1, 1, 1), seasonal_order=(0, 1, 1, 12))
results2 = model2.fit()

# Predict future values using the trained model
start2 = len(train_data2)
end2 = len(train_data2) + 10 # 10 years ahead
predictions2 = results2.predict(start=start2, end=end2, dynamic=False)

# Plot the predicted values against the actual values
plt.figure(figsize=(12,6))
plt.plot(df4['Year'], df4['Total_Temperature'], label='Actual')
plt.plot(range(2022, 2033), predictions2, label='Predicted')
plt.title('Temperature in Dublin with 10 years of prediction (1942-2033)')
plt.xlabel('Year')
plt.ylabel('Temperature (Celsius)')
plt.legend()
plt.show()
```

Here we split our training set up to the year of 2012 and the test set from the year 2012 and further. Additionally, we will perform predictions for 10 years in the future, and see if our predictions are better.

Plot of the actual and predicted future data (10 years)



Calculating the Mean Squared Error (MSE) of our SARIMAX Model with predictions of 10 years

```
mse_sarimax102 = mean_squared_error(test_data2, predictions2)
print(f"Sarimax 10 years MSE: {mse_sarimax102}")
```

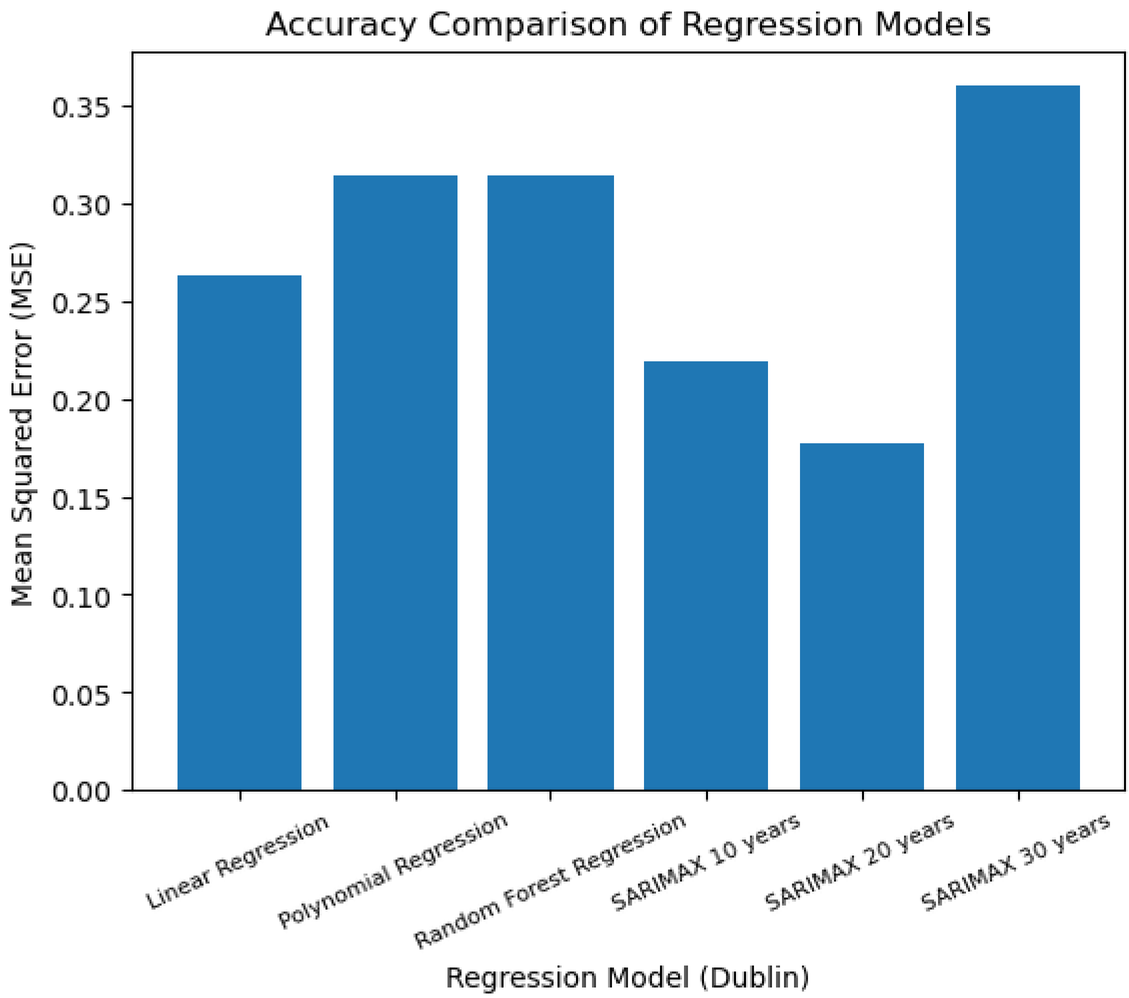
```
Sarimax 10 years MSE: 0.21872342716255522
```

Our SARIMAX model performed with a Mean Squared Error score of 0.218 when using a different train test split and performing a prediction of 10 years.

5.2. Evaluation

Comparing the accuracy of our models using the Mean Squared Error (MSE) and plotting their MSE scores using Matplotlib's Bar plot

```
models2 = ['Linear Regression', 'Polynomial Regression', 'Random Forest Regression', 'SARIMAX 10 years', 'SARIMAX 20 years', 'SA
mse_scores2 = [mse_lr2, mse_poly2, mse_rf2, mse_sarimax102, mse_sarimax202, mse_sarimax302]

plt.bar(models2, mse_scores2)
plt.xlabel('Regression Model (Dublin)')
plt.xticks(fontsize=8, rotation=25)
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Accuracy Comparison of Regression Models')
plt.show()
```



6. Deployment

In our final CRISP-DM phase we deal with the deployment of our Machine Learning model.

For the Ireland dataset we concluded that the Random Forest Regressor showed a better performance when compared to the other models, both based on its MSE score of 0.175 and the plotted predicted values we saw previously. This regression model better captures the variability in the data and provides relatively accurate predictions.

As for the Dublin dataset, we concluded that the SARIMAX model with a 20 year prediction outperformed the other regression models. The predicted future data as shown in the plot below, shows a similar pattern and behaviour to the actual data.

Conclusion

Our study shows that machine learning algorithms can, although never 100%, accurately predict changes in the climate and other variables through the study of historical data for the development of these models.

With our prediction system, farmers can obtain useful insights and begin developing strategies on how to adapt their practices for sustainable food production in the face of climate change.

We hope that our findings will encourage further research and development of machine learning-based prediction systems to help mitigate the impact of climate change on agriculture and natural resources.

Link for pre-recorded presentation:

https://drive.google.com/file/d/14NJjeWvsQ8L35jVETv4b8gkhqAu9EJlv/view?usp=sharing

References

Chahuan, N., S. (2019) *A Beginner's Guide to Linear Regression in Python with Scikit-Learn.* Available at: https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html (accessed 14 May 2023)

Sharma, A. (last updated 2022) *Introduction to Polynomial Regression (with Python Implementation)*. Available at: https://www.analyticsvidhya.com/blog/2020/03/polynomial-regression-python/ (accessed 14 May 2023)

Russel, S., J., & Norvig, P. (2022) *Artificial Intelligence: A Modern Approach*, 4th Edition, Global Edition. Harlow: Pearson Education Limited.

Mbaabu, O. (2020) Introduction to Random Forest in Machine Learning. Available at: https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/ (accessed 14 May 2023)

Pierre, S. (updated 2022) *A Guide to Time Series Forecasting in Python.* Available at: https://builtin.com/data-science/time-series-forecasting-python (updated by Urwin, M. on 4 November 2022, reviewed by Sawtell-Rickson, J.) (accessed 19 May 2023)

Verma, Y. (2021) *General Overview of Time Series Data Analysis.* Available at https://analyticsindiamag.com/general-overview-of-time-series-data-analysis/ (accessed 19 May 2023)

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). *CRISP-DM 1.0: Step-by-step data mining guide*. SPSS Inc.

DSDM Consortium. (2008). *DSDM Atern: The handbook*. DSDM Consortium.

Houben, G., Lenie, K., & Vanhoof, K. (1999). *A knowledge-based SWOT-analysis system as an instrument for strategic planning in small and medium sized enterprises*. Decision support systems, 26(2), 125-135.

IPCC. (2019). *Climate change and land. Intergovernmental Panel on Climate Change*. Retrieved from https://www.ipcc.ch/srccl/

Witten, I. H., & Frank, E. (2005). *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann.

FAO. (2021). *Climate change and agriculture. Food and Agriculture Organization of the United Nations*. Retrieved from http://www.fao.org/climate-change/en/

EPA. (2021). *Climate change impacts and adaptation for agriculture in Ireland. Environmental Protection Agency*. Retrieved from https://www.epa.ie/pubs/reports/research/climatechange/ccraagricultureinireland/

Ray, D. K., West, P. C., Clark, M., Gerber, J. S., Prishchepov, A. V., & Chatterjee, S. (2019). *Climate change has likely already affected global food production*. PLOS ONE, 14(5), e0217148. doi: 10.1371/journal.pone.0217148

Teagasc. (2021). *Agriculture in the Irish economy*. Teagasc. Retrieved from https://www.teagasc.ie/rural-economy/agriculture/agriculture-in-the-irish-economy/

Central Statistics Office (CSO) Ireland. (2020). *Agriculture Statistics*. https://www.cso.ie/en/statistics/agriculture/

Climate Ireland. (2021). *Climate Change Scenarios for Ireland*. https://www.climateireland.ie/scenarios

Irish Farmers' Association (IFA). (2020). *Climate Change and Agriculture*. https://www.ifa.ie/sectors/climate-change-and-sustainability/climate-change-and-agriculture/

European Environment Agency. (2020). *Climate change, impacts and vulnerability in Europe 2020: An indicator-based report.* Retrieved from https://www.eea.europa.eu/publications/climate-change-impacts-and-vulnerability-2020/

Ireland's Climate Change Advisory Council. (2019). *Climate change risks and opportunities - Adaptation in the agriculture and forestry sectors*. Retrieved from https://climatecouncil.ie/assets/uploads/2019/11/CCAC-Sectoral-Report-Agriculture-and-Forestry.pdf

USDA. (2017). *Climate change and agriculture in the United States: Effects and adaptation*. Retrieved from https://www.usda.gov/sites/default/files/documents/USDA_Climate_Change_Adaptation_Plan_Oct2017.pdf

Sustainable Food Systems Ireland. *Agriculture & Food in Ireland* https://www.sfsi.ie/agriculture-food-ireland/

Department of Agriculture, Food and the Marine (2021) *Annual Review and Outlook for Agriculture, Food and the Marine 2020 - Ch.1 Agri-food Sector and the Economy*., Government of Ireland, p.16.

https://www.gov.ie/pdf/?file=https://assets.gov.ie/118138/afc83651-331b-4edc-bec2-1e870ee10d4d.pdf#page=null

Walsh, M.K., et al. (2020). *Climate indicators for agriculture*. USDA Technical Bulletin 1953. Washington, DC, p. 1. Retrieved from https://www.usda.gov/sites/default/files/documents/climate_indicators_for_agriculture.pdf

United States Environmental Protection Agency (EPA) (last updated December 13 2022) *Climate Changes Impacts on Agriculture and Food Supply* (accessed 9 March 2023) Retrieved from https://www.epa.gov/climateimpacts/climate-change-impacts-agriculture-and-food-supply

R. Baruchi (2022) *The Impact of Predictive Analytics on the Global Food System* https://www.dataversity.net/the-impact-of-predictive-analytics-on-the-global-food-system/ (accessed 11 March 2023)

S. Shekhar and et al., (2017) *Agriculture Big Data (AgBD) Challenges and Opportunities From Farm To Table: A Midwest Big Data Hub Community† Whitepaper.* Retrieved from https://midwestbigdatahub.org/wp-content/uploads/2021/08/AgBD_Whitepaper2017.pdf

Global Open Data for Agriculture & Nutrition https://www.godan.info/

Nural Research (2021) *The Future of Food: Predicting Crop Yields with Machine Learning* https://www.nural.cc/climate-change-crop-yield-machine-learning/ (accessed 12 March 2023)

Driscoll, M. (undated) *Jupyter Notebook: An Introduction* https://realpython.com/jupyter-notebook-introduction (accessed 16 March 2023) Real Python.

Kuhlman, D. (2012) *A Python Book: Beginning Python, Advanced Python, and Python Exercises* *https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html* (accessed 16 March 2023) Wayback Machine.

McKinney, W., (2013) *Python for Data Analysis*. Sebastopol: O'Reilly Media, Inc. pp. 1-3.

Tuama, O. D. (2022) *What are Libraries in Python?* https://codeinstitute.net/ie/blog/what-are-libraries-in-python/ (accessed 17 March 2023) Code Institute.

IBM Cloud Team (2021) *Python vs R: What's the Difference?* https://www.ibm.com/cloud/blog/python-vs-r (accessed 18 March 2023) IBM.

Simpao, C. M. (2022) *10 Best Python IDE & Code Editor* https://hackr.io/blog/best-python-ide (accessed 18 March 2023) Hackr

Hall, P., and Ouederni, A. (2021) *Seven Legal Questions for Data Scientists* https://www.oreilly.com/radar/seven-legal-questions-for-data-scientists/ (accessed 19 March 2023) O'Reilly.