Jellyfish Population Time Series Investigation:

Comparison of Statistical and Deep Learning Methods Around the Coast of Ireland and the UK

Thomas Bridgeman

M.Sc. in Computing
in Data Science

2024

Ollscoil Teicneolaíochta an Atlantaigh Dún na nGall
Atlantic Technological University Donegal

Department of Computing, ATU Donegal, Port Road, Letterkenny, Co. Donegal, Ireland.

Jellyfish Population Time Series Investigation:

Comparison of Statistical and Deep Learning Methods Around the Coast of Ireland and the UK

Author: Thomas Bridgeman

Supervised by: Dr William Farrelly

A thesis submitted in partial fulfilment of the requirements for the

Master of Science in Computing in Data Science

Submitted to Atlantic Technological University
Arna chur isteach chuig Ollscoil Teicneolaiochta an Atlantaigh September 2024

Declaration

I hereby certify that the material, which l now submit for assessment on the programmes of study leading to the award of Master of Science in Computing in Data Science, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution. I understand that it is my responsibility to ensure that I have adhered to ATU's rules and regulations.

I hereby certify that the material on which I have relied on for the purpose of my assessment is not deemed as personal data under the GDPR Regulations. Personal data is any data from living people that can be identified. Any personal data used for the purpose of my assessment has been psudonymised and the data set and identifiers are not held by ATU. Alternatively, personal data has been anonymised in line with the Data Protection Commissioners Guidelines on Anonymisation.

I consent that my work will be held for the purposes of education assistance to future students and will be shared on the ATU Donegal (Computing) website (atucomputingdonegal.com) and Research THEA website (https://research.thea.ie/). I understand that documents once uploaded onto the website can be viewed throughout the world and not just in the Ireland. Consent can be withdrawn for the publishing of material online by emailing Jade Lyons; Head of Department at Jade.Lyons@atu.ie to remove items from the ATU Donegal Computing website and by email emailing Denise McCaul; Systems Librarian at denise.mccaul@atu.ie to remove items from the Research THEA website. Material will continue to appear in printed formats once published and as websites are public medium, ATU cannot guarantee that the material has not been saved or downloaded.

Signature of Candidate: Thomas Bridgeman Date: 26/08/2024

Acknowledgements

I want to thank the many lecturers at Atlantic Technological University who helped me get this far, especially Dr William Farrelly, for supporting this dissertation. I would also like to thank my family for their assistance in the original idea of the dissertation.

Abstract

This dissertation investigates the fluctuations in jellyfish populations around the coast of Ireland and the UK from the period 2002 to 2023, with a focus on understanding whether human factors like climate change are a cause. Over the past two decades, jellyfish blooms or large population spikes have garnered increased attention due to their impact on ecosystems and economies. Varying studies attribute these changes to rising sea levels and increased temperatures while others have argued that jellyfish populations are volatile and fluctuate wildly as part of their natural life cycle. This dissertation tackles the problem from both the statistical time series side but also incorporates deep learning techniques to examine the trends and oscillations in jellyfish populations.

The dissertation begins with a focus on the statistical time series approach where tested methods like Auto-Regressive Integrated Moving Average (ARIMA) and its offshoots like Seasonal Auto-Regressive Integrated Moving Average with eXogenus factors (SARIMAX) are explored. Various techniques to improve these models were explored including autocorrelation, differencing and time series split. The results from this varied but suggested an overall increase in the next few years while maintaining an oscillating pattern. Deep learning approaches like Long Short-Term Memory (LSTM) and Transformers were investigated and had the potential to extract greater understanding from the data by capturing longer-term dependencies. These approaches again varied in results but overall, also suggested an increase in the oscillating pattern for the next few years.

The findings as stated were often contradictory and varied mirroring the complexity of jellyfish populations but overall, they agreed that their populations face a natural cycle, but the highs are going to continue to increase likely due to human and climate-related influences.

This dissertation explores the importance of taking multiple approaches to investigate a complex problem as no single model revealed a definitive answer and by extracting parts from multiple sources a solution to the problem becomes clearer. While insights in this dissertation are interesting, they mostly serve as a stepping stone for further research in the future with more complex models and access to better data which could assist in solutions to tackling this complex problem.

Acronyms

Acronym	***Description***
AI	Artificial Intelligence
ARIMA	Auto-Regressive Integrated Moving Average
SARIMAX	Seasonal Auto-Regressive Integrated Moving Average (Exogenous)
VARIMA	Multivariate Auto-Regressive Integrated Moving Average
LSTM	Long-Short Term Memory
NLP	Natural Language Processing
ML	Machine Learning
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
FNN	Feed-Forward Neural Network
RMSE	Root Mean Squared Error
MSE	Mean Squared Error
ADF	Augmented Dickey-Fuller
KPSS	Kwiatkowski-Philips-Schmidt-Shin
AIC	Akaike Information Criterion
BIC	Bayesian Information Criterion
HQIC	Hannan-Quinn Information Criterion

Table of Contents

Table of Figures

Table of Code Listings

Chapter 1: Introduction

Most modern predictions are made using Artificial Intelligence (AI) as Big Data Analysis as AI can examine huge amounts of data and find an underlying pattern or the correlation between the features and the target variables. One of the most common applications of AI predictions is examining empirical data and observing the change over time. This is referred to as time series forecasting or analysis.

This dissertation compares the statistical and deep learning time series around the coast of Ireland and the UK. Climate models and simulations have revealed that rising temperatures accelerate ice sheet melting and sea level rise, solidifying the link between these phenomena and climate change. Projects indicate that Ireland may experience a temperature increase of 1 to 1.6 degrees Celsius by 2050, leading to shifts in growing seasons, altered precipitation patterns, and more extreme weather events.

Jellyfish, predominantly found in oceans and seas are known for their umbrella-shaped bodies and stinging tentacles. While some species are consumed, especially in Asia due to their high protein content, jellyfish are also valued for their bioluminescent proteins used in genetic research. However, jellyfish pose significant challenges such as damaging fishing gear and stinging swimmers. While most stings are mild, some species can cause severe harm or even fatalities.

In recent years, there seems to be an increase in the number of jellyfish and different species in some waters. This can likely be attributed to many things like the rise in ocean temperatures increasing the habitat range that some jellyfish can survive in. Blooms are becoming more prevalent around the coast of Ireland and the UK due to a variety of factors and some locations are even more high-risk (Kennerley *et al.* 2021). Overfishing removes the main predators of jellyfish meaning they are free to overpopulate. Fish farms can also limit the area fish reside giving jellyfish easier access to certain areas jellyfish also seem to swarm and settle around human-made structures like oil rigs in the water giving them more chances to populate (Kamaruddin *et al.* 2023).

Time series analysis of climate change has been examined countless times due to the abundance of data on the topic available, here an attempt will be made to understand the

trends in jellyfish populations due to factors like climate change temperatures and sea levels under different time series applications such as Autoregressive Integrated Moving Average or ARIMA is one of the classical approaches to time series (Kontopoulou *et al.* 2023) or new cutting-edge deep-learning models (Lim and Zohren 2021) like Inverted Transformers which use a variation of the model framework utilised by large language models (Peixeiro 2024).

1.1. Purpose

Time series analysis is a technology that has existed for many years now, but it continues to improve due to new algorithms and more data available. One of the most traditional time series algorithms is the Autoregressive Integrated Moving Average (ARIMA) model, which was built by George Box and Gwilym Jenkins in the 1970s. This algorithm is still used today and has seen improvements to allow for seasonality (SARIMA) or multivariate (VARIMA) data. Modern approaches focus on the deep learning approach to time series and in most cases have been observed to succeed in outperforming the classical approach in terms of capturing complexity and higher accuracies while classic approaches are often simpler and easier to interpret (Sam 2023). These deep learning approaches are often based on neural networks such as Long Short-Term Memory (LSTM) which is a type of recurrent neural network or Convolutional Neural Networks which usually is reserved for working with image data(Lim and Zohren 2021).

One of the most recent approaches in time series is called Inverted Transformers or iTransformer, it is positioning itself as an alternative to other deep learning approaches. Transformers are usually utilised in Natural Language Processing (NLP) and are the backbone of many generative models like ChatGPT. Recent developments have rebuilt a transformer to accommodate time series instead of NLP or computer vision-based applications hence the inverted name. The benefits are meant to include better forecasting accuracy as it can capture more complex patterns and higher interpretability when compared to standard Transformer models (Peixeiro 2024).

This dissertation will examine various time series models, ARIMA, Transformers and LSTM to understand climate change trends and how they may correlate to jellyfish populations. The

dataset from NBN Atlas (Marine Conservation Society 2024) contains jellyfish around the coast of the UK from 2003 to 2022. There are 57 columns of information on the sightings that contain information on location, species, date and abundance. Climate information is taken from (Met Office 2024) which has data from 1884 to 2024 about temperature and rainfall. Two datasets from Kaggle (Geukjian 2024; Koustubhk 2024) are datasets on overfishing and sea level. The aim is to find any correlations and forecast potential jellyfish populations based on changes in the environment.

1.2. Background

Time series has become an extremely common and explainable method used for showing predictions ever since it was first conceptualised. Improvements have been made to the original algorithms to make them more suited to specific problems. Since then, as AI continued to improve through deep learning methods so did time series as it adapted these newer models. In most cases, the deep learning models outperformed the classical autoregressive approaches. Even now, newer more sophisticated solutions to time series are being developed. Transformers are a vital component of natural language processing and large language models, but they are unsuited to time series because any variable of slightly different times gets blurred together and there is no long-term dependency. Researchers created a solution to invert the transformer that changes how variables are created from timestamps and better captures the relationship between variables (Peixeiro 2024).

Previous papers have discussed the effect climate change and overfishing have had on jellyfish populations. A study from 2010 in the Irish Sea concluded that from a time series analysis of 16 years, the population of jellyfish correlated positively with the increase in ocean temperature. They also found that there was likely a link between the ecosystem change and the drop in herring due to overfishing (Lynam *et al.* 2011). Some newer papers also examine a similar problem and come to similar conclusions about how jellyfish benefit from climate change, overfishing, ocean oxygen content and the acidity of water (Liu 2023). They also found that not all species are benefiting from this but for the most part, their increasing dominance in the water is becoming a huge problem.

1.3. Research Aims and Objectives

The aims and objectives of the research paper are:

1. Investigate classical time series approaches such as ARIMA and its various child models.

2. Compare different deep learning approaches like LSTM and Transformers.

3. Perform these time series approaches on climate change-based data – temperature, sea level

4. Find trends in the jellyfish population using these different time series approaches.

1.4. Research Question

This research paper will answer will attempt to answer the following questions:

1. How do classical time series approaches compare to deep learning-based approaches?

2. What trends are present in Jellyfish populations around the coast of the UK due to factors such as increased temperatures and rising sea levels?

1.5. Report Outline

1.5.1. Chapter 2: Literature Review

Chapter 2 is the literature review; it will feature an overview of the topics that are being discussed in this dissertation such as climate change and jellyfish, but the bulk of the chapter will focus on the theory on the different time series algorithms being used. Finally, papers with similarities will be evaluated for their strengths and weaknesses.

1.5.2. Chapter 3: Research Design and Methodologies

Chapter 3 concerns how the project will be developed. It will feature flowcharts to show the intended path of the program.

1.5.3. Chapter 4: Implementation

How the project was implemented will be discussed in chapter 4. It will contain the setting up of the environment and gathering the necessary data. From there it will discuss any exploratory analysis completed, model implementation and the graphing of any results.

1.5.4. Chapter 5: Results and Evaluations

Chapter 5 will feature some of the tests that will have to be completed on the final program for it to be considered a success but most of the chapter will deal with the results and graphs produced from the various time series models and specifically what they mean.

1.5.5. Chapter 6: Conclusion

Chapter 6 will deal with the conclusion, some final thoughts on the project and results and where the research could be improved upon in the future.

Chapter 2: Literature Review

2.1. Introduction

This decade has seen the widespread adoption of artificial intelligence by almost every sector. There are many concerns over the ethics of its widespread use to replace jobs and how accurate the information it is providing is. There have already been clear indications of the 'Dead Internet Theory' on various social media platforms (Zarkadakis 2022). Despite this, there is no denying the good that AI can also produce when correctly used.

Due to the abundance of AI-based applications, the meaning of AI has become an umbrella term that encompasses the entire industry and because of this, it is being used as a buzzword on many products. Traditionally, it has been associated with a computer's ability to learn based on information provided, perceive, and understand its current environment and decide based on both what it has learned and where it currently is (Zhang and Lu 2021).

Machine learning (ML) is a specific aspect of AI that deals with using algorithms to learn information from vast amounts of data and make specific predictions when presented with unseen data. Deep Learning is a further subset of Machine Learning that focuses on artificial neural networks to make decisions in much the same way a human brain makes its decisions (Bengio *et al.* 2016). The basic idea behind a neural network is much like any program in that there are inputs, processes and outputs. These are usually referred to as layers with an input and output layer and the processes are called the hidden layer. Inside the hidden layer, there can be a few or many layers where computation occurs. Much like machine learning, deep learning can be supervised or unsupervised where it either learns the pattern between input and output or where it groups similar unlabelled data points and defines a boundary (Buduma *et al.* 2022).

The basic approach of a deep learning model is the concept of backpropagation. The process is designed to reduce or minimise the error in predictions. The first step in backpropagation is the forward pass when the input data is passed through the hidden layers and an output or prediction is generated. A loss function is used which gives a value between the actual and predicted values. The backward pass is performed which finds the derivative of the error and this value is propagated through the network to the start again. The weights and biases are

then updated using gradient descent and the process is repeated for many epochs until a point or convergence (Wright *et al.* 2022). This is the basic idea of a deep learning model and there are more concepts like activation functions such as Rectified Linear Unit (ReLU) or Sigmoid which can handle more complex problems.

Deep learning is the backbone of most modern AI products. It is utilised in computer vision-based applications like self-driving cars or through facial recognition on a phone and many more. It utilises a convolution layer that breaks down images to be interpreted by a machine. Large Language Models like Chat GPT and other natural language processing like translation. The technology has been or is being developed for many areas including healthcare, finance, gaming and many more (Sharifani and Amini 2023).

Machine learning and deep learning are highly prevalent in the climate sector where it is becoming increasingly important to predict how changes in our lifestyle and industry could snowball the effects of climate change. One of these changes is the jellyfish population, which according to studies (Lynam *et al.* 2011; Fernández-Alías *et al.* 2024) show that changes in climate such as sea level rises and temperature increases have led to an increase in jellyfish blooms. Some papers (Condon *et al.* 2013) reached conclusions that jellyfish populations fluctuate for no apparent reason and are not a result of climate change. All these papers utilised the machine learning-based time series to reach their conclusions, but they failed to integrate any deep learning advancements in their research as the technology did not exist in the format it now does.

2.2. Time Series

Time series comes from the world of mathematics where it refers to data points that are sequenced in order of what time they were captured. From the data points, analysis can be completed which can give insights into a trend. Time series is commonly seen expressed as a run chart or a temporal line chart and is used across various sectors like weather forecasting and economic predictions (Lim and Zohren 2021). Through time series various patterns emerge such as trends which rely on historical data to see overall changes like sea levels rising is only apparent in the long term. Seasonality also plays a role in time series as the time of year can affect tourism in an area or how likely you are to see an animal.

2.2.1. Statistical Time Series

Autoregressive Integrated Moving Average is one of the popular methods of performing time series analysis. When data is nonstationary or data that changes over time is present, ARIMA-based models are commonly found. ARIMA is built upon three parts, AR the Auto Regressive, I the Integrated and MA the Moving Average. AR refers to any previous values in the model and the relationship they exhibit with each other. Integration is the process of differencing the data which is replacing current values with the difference between current and previous values. This makes nonstationary data stationary. MA applies a forecast model to previous observations to produce a regression model based on the error present (Kontopoulou *et al.* 2023).

ARIMA has been used across various industries like finance, retail and science due to its adaptability and accurate short-term findings (Kontopoulou *et al.* 2023). ARIMA can be complex to correctly fit and there needs to be an ample amount of data for it to make accurate predictions. While it will still likely be used, newer deep learning approaches are being explored further every day making ARIMA more obsolete as they can better capture relationships, especially nonlinear relationships (Sam 2023).

2.2.2. Deep Learning Time Series

As the need to solve more complex problems using time series became more apparent, researchers began experimenting with deep learning approaches like Recurrent Neural Networks (RNN). An RNN is a specific type of neural network that uses the output of the previous step as the input for the current step as opposed to inputs and outputs being independent of each other. The RNN's hidden layer is designed to remember the sequence and can be referred to as the memory state (Schmidt 2019).

While an important building block to deep learning time series it suffers from the exploding gradient problem where the loss calculated becomes too large and leads to instability and the network unable to converge to a solution. This becomes more apparent as datasets and sequences grow larger (Sherstinsky 2020).

2.2.2.1 Long Short-Term Memory

As problems exist with using RNNs for time series, it was clear that the technology had a future and one of the first solutions to this was a new RNN, the LSTM which helps address the

vanishing and exploding gradient problem that can plague RNNs. LSTMs consist of three core components and a cell state (Staudemeyer and Morris 2019). The cell state is what allows the LSTM to remember long periods of information. The core components are often referred to as gates; input, forget and output. The input gate controls how much new information is added to the cell state, the forget gate decides what information is relevant and disregards what isn't needed, and the output gate selects the output from the cell state and the input. The forget gate makes use of the sigmoid function which returns a value between 0 and 1 and it is retained if closer to 1 and forgotten if closer to 0. The input gate uses the sigmoid function too to determine which values are updated and the tanh function in the creation of new values. The input and forget gates are combined before the output gate determines the hidden state of the next cell (Sherstinsky 2020).

LSTMs have the advantage of leveraging more information from long-term dependencies and avoiding the exploding and vanishing gradient problems using the gates. They are also more flexible than RNNs and are being used in a wide spectrum of tasks across various industries.

Time-series forecasting with deep learning a survey (Lim and Zohren 2021) talks about some of these advancements in detail and how LSTMs have improved upon RNN architecture. One of their suggestions is to incorporate hybrid models which can outperform statistical approaches and not overfit as much as some deep learning approaches and utilize transformer-based approaches to time series problems.

2.2.2.2. Transformers

Attention Is All You Need (Vaswani *et al.* 2017) is a paper by eight scientists working at Google that changed modern AI by introducing the transformer to deep learning. The architecture of a transformer is what is responsible for large language models, but it has also been used for computer vision and speech processing. The basic idea behind a transformer is a self-attention mechanism and the encoder-decoder architecture. The self-attention mechanism gives weight to words in a sentence based on their importance which allows it to better understand long-term relationships between different words or points which RNNs and LSTMs can struggle with. The encoder-decoder architecture has two key parts the encoder and decoder, the encoder has two components – the multi-head self-attention which enables the transformer to analyse separate areas of a sequence simultaneously and combine their

outputs to give a better perspective on the underlying relationship and position-wise fully connected Feed-Forward Network (FNN). The position is tracked using positional encodings which are added to the input as transformers are not designed in the same sequential way an RNN would be developed. The decoder has the same two components and a multi-head attention of the outputs from the encoder.

These concepts are applied in the architecture of the transformer. Data points are converted into vectors called input embeddings; the positional encoding is also added. The embeddings are passed through the encoder layer which uses multi-head self-attention to assign weights to words and it passes through the feed-forward network and is normalised. The decoder then uses the multi-head self-attention again to predict words one at a time and focuses on the encoder output to analyse specific parts of the sequence before it is passed through the FFN and is then normalised. The output is then passed through a SoftMax activation function to produce a probability (Vaswani *et al.* 2017).

Transformers offer several advantages compared to RNNs as they can process a sequence simultaneously making them much more efficient in training, they scale better and can handle much longer-term dependencies thanks to the self-attention mechanism, and they have been shown to outperform RNNs and LSTMs models in their results. As stated, transformers' main use case has been for Natural Language Processing such as sentiment analysis, large language models and translation. Transformers have been adapted for computer vision, speech processing and reinforcement learning (Zeyer *et al.* 2019).

While transformers were developed for primarily NLP, they have seen success inside time series because of their ability to capture complex relationships and long-term dependencies between temporal data points. Transformers can be used in time series to forecast, and detect anomalies and multivariate time series analysis where dependencies between separate series can be found (Wen *et al.* 2023).

One paper, ITransformer: Inverted Transformers are Effective for Timer Series Forecasting (Peixeiro 2024) take the approach that while transformers are well suited to time series they need to be tweaked in to better suit temporal data. They state that traditional transformers can often blur data together and can still struggle with some long-term dependencies. The architecture was tweaked to make each variable’s history into a token instead of individual

temporal points. Self-attention could then be used over the tokens and in doing so gives higher forecasting accuracy and better understanding of long-term dependencies when compared to other transformer models.

2.3. Jellyfish and Climate Change

2.3.1. Jellyfish

Jellyfish or sea jellies are an aquatic lifeform found in all the world's oceans as well as many freshwater sources (Gershwin 2013). They are defined by their jelly-like umbrella and trailing tentacles that cause varying degrees of stings to humans every year. While most stings are harmless and cause mild discomfort, there are certain species which can cause serious injury and even death (Cegolon *et al.* 2013). Jellyfish can be found in every ocean and some freshwater locations. They can be located inland near the coast, open waters and the depths of the deep sea (Lynam *et al.* 2011).

As the populations of jellyfish have increased, the problems associated with them have only become more apparent. In the field of aquaculture, jellyfish getting caught in nets and causing damage is becoming more frequent which reduces the number of fish caught and more time is expended in sorting out catches of fish (Lynam *et al.* 2006, 2011). Jellyfish are most famous for their stings. Most species cause stings with mild discomfort through itching or swelling but there are species like the box jellyfish which can cause cardiac arrest in a human within a few minutes of being stung. (Cegolon *et al.* 2013).

Climate change, overfishing and pollution have been attributed to jellyfish blooms (Lynam *et al.* 2011; Condon *et al.* 2013). Blooms are a major increase in jellyfish population in an area. These blooms can devastate local ecosystems further by disrupting the food chain further. Jellyfish blooms can also influence algal blooms which deplete oxygen levels and contribute to dead zones (Møller and Riisgård 2007). Some of these blooms are due to invasive species migration using ocean currents and rising temperatures increasing the area species can survive in (Bayha and Graham 2014).

Ireland and the United Kingdom are islands and therefore fully surrounded by water on all sides. They both have the same cool temperate oceanic climate due to their similar latitude,

island status and the North Atlantic Drift helping maintain a constant stream of warm water which prevents coastal areas from freezing in the winter (McCarthy *et al.* 2015). These factors influence the jellyfish species in the area, especially in the warmer summer months where it is common to see large blooms due to temperature, ocean currents and food abundance (Lynam *et al.* 2011). In these regions, you can commonly expect to find moon jellyfish, lion's mane jellyfish, Portuguese Man O' War, which is not technically a jellyfish, instead a siphonophore which comes from the same species as jellyfish, compass jellyfish, barrel jellyfish and blue jellyfish. The Man O' War and Lion's mane are the only dangerous species and seem to have increased as temperatures have risen (Doyle *et al.* 2007).

2.3.2. Climate Change

Climate refers to the average weather condition of a region over a long period and will usually have a massive effect on the flora and fauna of the region as well as the way humans interact with their environment. Climate change is the phenomenon of this average weather change. This can bring many benefits such as better warmer weather and increased growing seasons in some regions, but it will also lead to harsher conditions, less land, invasive species, and refugees from others (Dessler 2021).

Climate change is a natural phenomenon that occurs on Earth. There has been a rough pattern of cold periods called an ice age where there are glaciers on the earth and hot periods called greenhouse periods where there are no glaciers. Based on this current definition, the earth is currently in an ice age called the Quaternary glaciation. During this period of glaciation, sheets of ice covered much of Europe and North America but in the last 10,000 years have been slowly melting with only Antarctica, Greenland and a few mountainous areas still containing any glaciers (Stern and Kaufmann 2014).

In the modern world, however, climate change is considered fast and artificial because, in the 1800s, human beings began using fossil fuels to power a variety of machines resulting in the release of greenhouse gases causing a phenomenon dubbed global warming. (Dessler 2021). In addition to this, humans have also deforested huge areas of woodland and rainforest which means that less of the carbon dioxide released from fossil fuels is being converted back into oxygen through photosynthesis. All this leads to a substantial increase in the earth's temperature much quicker than the earth can adapt or humans will react.

While climate change is affecting the world, different areas are going to experience different symptoms due to their global position and current climate. Ireland and the UK are in a particularly vulnerable position as their experience will be heavily reliant on the Atlantic Ocean (Environmental Protection Agency 2021). Both countries have shown increases in average temperature rising a degree over the last century and highest temperatures reaching new highs of over 40 in some areas. There has been an increase in storms affecting the regions with many lowland areas being affected by flooding and in general a change in the amount of precipitation with more coastal areas experiencing increased rainfall while the southeast of England has seen droughts in 2018 (Hanlon *et al.* 2021). Glacier melting in the poles is causing a rise in sea levels. This speeds up coastal erosion processes along the west coast of Ireland and could affect major cities like Dublin and London which are built around rivers (Tay *et al.* 2022).

Jellyfish are one of the creatures that are benefitting most from changes in the climate, the main reason is that they are endothermic and as ocean temperatures rise, the areas in which they can successfully inhabit are increased which is what is currently happening in coastal waters around Ireland and the UK (Lynam *et al.* 2011). Warmer temperatures have also been attributed to larger blooms due to an increased reproductive rate which also may be happening more frequently. Ocean acidification where the pH of the ocean is slowly decreasing over time because it is absorbing more carbon dioxide, directly affects micro-organisms like plankton. In some cases, this increases their population and in others, it decreases their population, this in turn affects the population of jellyfish either positively or negatively depending on the region (Gattuso and Hansson 2011).

2.3.3. Overfishing

Fish are a sustainable resource, assuming there are enough healthy adult fish to repopulate fish populations the numbers of fish should remain consistent each year. There is historical evidence to suggest that overfishing has been something that has occurred for hundreds of years but at that, it was usually limited to a small area (Shakouri *et al.* 2010). In the 1900s as the number and size of boats increased so did the size of fish hauls. Large nets, winches and technology like sonar and radar allowed for massive catches of fish and over time the demand exceeded the supply. For humans, overfishing can affect many businesses and people's employment. However, the effect on oceans can be devastating to local ecosystems, many

reefs rely on fish populations to survive in a healthy state and lack of fish will also affect larger marine animals that rely on fish as a source of food which could decrease their populations as well.

In addition to these effects, overfishing has allowed jellyfish populations to grow without any of their natural predators like tuna. In areas lacking some of their natural predators, jellyfish populations have been able to thrive. Scientists have also observed that smaller fish would feast upon previous egg and larvae stages of jellyfish further helping to regulate their population in waters (Lynam *et al.* 2011).

2.3.4. Eutrophication

Algal blooms occur when there is an increased amount of nutrients in a body of water that causes an increase in the number of algae and other aquatic plants. This process is known as eutrophication and can be devasting to marine life (van Beusekom 2018). The main culprit of the nutrient increase in agricultural runoff full of fertiliser. The fertiliser increases the amount of phosphorus and nitrogen in the water which increases plant and algae growth. Excess sewage and industrial waste that is not properly treated can also supply nutrients. Dense underground jungles of algae are found in these areas, as algae goes through its life cycle it consumes oxygen in the water making low oxygen zones or hypoxia (El Gamal 2010). The change in oxygen kills certain marine life and certain algae also release toxins that further reduce marine populations. As water quality is affected, industries like drinking water and fisheries suffer the consequences.

Jellyfish are one of few species that have a high tolerance than most to waters with lower oxygen levels which means in areas of eutrophication they become the dominant species to the plankton in the area which also increase their population in areas of algal bloom. Predators of jellyfish struggle to survive in these areas allowing jellyfish populations to bloom (Møller and Riisgård 2007).

2.3.5. Ocean Currents

Ocean currents are the movement of water between locations. They are caused by differences in water density, tide and the wind. Often, they bring cold or warm water to different areas. The Atlantic coast of Europe's climate is heavily influenced by the Gulf Stream and North Atlantic Drift which brings warmer water from the Caribbean. Ireland and other

European countries would likely experience much colder temperatures based on latitude if not for this current. As the temperature of Earth increases more glaciers melt causing sea levels to rise and change the salinity of water. There has been evidence that this slows down or moves ocean currents which could potentially lead to colder temperatures during the winter across Europe (Greene *et al.* 2008). This could shift ecosystems too, with colder or warmer temperatures forcing species to migrate to survive. Jellyfish may be able to thrive if there are no longer any predators in an area and invasive species may choose to settle (Condon *et al.* 2013; Fossette *et al.* 2015).

Recurrent jellyfish blooms are a consequence of global oscillations (Condon *et al.* 2013), this paper claims that any changes in jellyfish populations are not due to changes in the ocean like acidification or climate change but instead follow a natural oscillating cycle of increases and decreases. It examined the longest-term dataset on jellyfish abundance and sightings and used different linear and logistic methods to conclude that there is an approximate 20-year oscillation cycle of heightened jellyfish abundance.

The author claims more research is needed to validate their claim over the next decade and ensure that is not the degradation of the ocean that is causing the increase in jellyfish blooms. It has been a decade since then and the ocean has continued to warm, and ecosystems change and there is evidence to support that this is benefitting jellyfish in most places. Overfishing, acidification and eutrophication which destroy marine ecosystems allow certain species of jellyfish to thrive and many more papers seem to suggest that there continues to be a rise in the number and size of blooms. Their final findings suggest that the next large-scale blooms will be part of this oscillation but will also be augmented by the aforementioned factors.

2.4. Related Work and Research Gaps

Generating High-Resolution Climate Change Projections Using Super-Resolution Convolutional LSTM Neural Networks (Chou *et al.* 2021) examines the climate change problem from the deep learning approach as it struggles less with the non-stationarity of climate data. It also implements CNNs into its model which further enhances predictions. They found that LSTMs outperform statistical methods when it comes to climate change data, and this could also be true for jellyfish data.

Have Jellyfish in the Irish Sea benefitted from climate change and overfishing? (Lynam *et al.* 2011) is the most similar dissertation to this one. The researchers examined 16 years of data from 1994 to 2009 and found that there was a correlation between ocean surface temperatures and the population of jellyfish in the Irish Sea. In terms of climate change, this showcases there is a correlation between the temperature of the sea and populations, likely due to seasonality, but extending this to climate change, an increase in ocean temperatures or decrease if it negates the effects of the North Atlantic Drift may affect populations.

Historical data on the number of herring caught over the years were also examined (Lynam *et al.* 2011) and they found that there was a connection with the jellyfish population as they compete for the same food sources, with a lack of herring due to overfishing, jellyfish populations will have greater access to nutrients to expand their population. Both jellyfish and herring will likely play an important role in the future of the ecosystem. In their discussion they said that there is a clear indication that jellyfish abundance is increasing yet, so few papers exist that investigate the topic. They suggested a reason for the abundance in Ireland was the presence of a cyclonic, near-surface gyre from May to October that led to high numbers of plankton in the area. Recent studies (Jones and Neill 2020) showcase the current state of the gyre containing up to 236 thousand tonnes of microplastic which could potentially affect marine life including jellyfish population dynamics.

Lynam concluded that while overfishing seems to be the most visible effect on jellyfish populations by reducing competition and predators, changes in the climate are likely just as influential because warmer waters affect the habitat and reproductive rate of the jellyfish. As this dissertation is older much of the advancements in deep learning were not available to the authors and it is possible that this could lead to new revelations and more information with more data now available.

A more recent paper, The Unpredictability of Scyphozoan Jellyfish Blooms (Fernández-Alías *et al.* 2024) through their time series analysis, they reached similar conclusions of an increase in the number of species of jellyfish having blooms due to climate change, overfishing and eutrophication. They are unable to conclude with certainty if there are major increases in blooms overall as long-term data suggests that there is variability similar to other research (Condon *et al.* 2013). To conclude, various statistical time series approaches were taken but

utilisation of updated deep learning models like LSTM or transformers was not used on the problem which could provide further insight into the problem, using these deep learning methods could provide better answers to the problem as they may be able to leverage more information from the data than statistical methods to find possible answers in regards to jellyfish population dynamics and climate change.

2.5. Conclusion

Climate change and time series are both areas that have featured extensive study and research continues to this day. New deep learning approaches, Transformers, LSTM, and new evidence come to light often and reveal information that may not have been known before. The connection between jellyfish and climate change has been discussed many times but a lot of the papers are older, especially ones that concentrate around the coast of Ireland and the UK. Combining these three topics will involve looking at papers that discussed the topics in isolation and some papers that combine these ideas to some extent. Most of the papers agree that human interaction seems to have the biggest impact on jellyfish populations whether its directly from pollution or overfishing or indirectly with climate change influencing ocean temperatures and altering marine ecosystems.

This chapter gave an overview of the research completed in this area and while the topics are often researched in isolation likely due to several factors but most likely because deep learning approaches to time series are still relatively new and are constantly being improved upon. There is also a lack of consistency in data sources surrounding jellyfish sightings compared to climate change readings as it is not something that can be measured easily. Through the guidance of prior research, this dissertation will attempt to bring more up-to-date answers to the jellyfish population problem using more sophisticated techniques and data. The next chapter will detail the design and how the project will be implemented from the ground up.

Chapter 3: Research Design and Methodology

3.1. Introduction

Through investigating various papers in the literature review chapter, a gap in the amount of research done on the relationship between climate and jellyfish has been identified especially when considering deep learning time series approaches and has not been attempted for the coast of Ireland and the United Kingdom. This chapter will detail how this will be achieved by examining the acquisition, preprocessing, exploration, modelling and graphing of the data at various points and using different time series models like ARIMA, LSTM and Transformers. The tools used such as software will be explored, and the chapter will feature UML to showcase the flow of the program and structure of the data.

3.2. System Requirements

3.2.1. Software Requirements

The project itself should be contained as it aims to only take data and extract information from it, therefore the only software necessary for the project will be Python, various Python libraries and a Jupyter Notebook code editor. Python is one of the most widely used data science languages due to its simplicity and wealth of libraries for data manipulation, modelling and graphing. NumPy and Pandas will be highly useful in manipulating the data and preparing it for modelling. Libraries like Matplotlib and Seaborn will be important in graphing the various results. Different time series libraries will be implemented like statsmodels for ARIMA modelling and PyTorch will be used in the deep learning approach for LSTMs and Transformers.

3.2.2 Hardware Requirements

In theory, any computer or laptop should be suitable to complete this project. For the early exploration and statistical time series, Google CoLab should be suitable for preparing and exploring the data. Later, in exploring the data it may be preferable to upgrade CoLab to take advantage of more cores and a better GPU or Tensor Processing Unit (TPU). Alternatively, a desktop with the minimum recommended specs could be used for deep learning. This project will make use of a desktop that contains an AMD Ryzen 7 5700X 8-Core processor, 16GB of RAM and an NVIDIA GeForce RTX 4060ti. While this project does not use images or a huge

amount of data, this project can likely be completed quicker, and it should speed up training time and allow for more epochs potentially leading to better results.

3.3. Unified Modelling Language

Unified Modelling Language is standard for outlining the design of a system. Many diagrams exist that can be tailored to suit a project. Large software engineering projects make the best use of UML notation, but a smaller-scale data science project can utilize the diagrams in unique ways even if they aren't specifically catered to them. In this case, a flowchart that focuses on the entire flow of the project rather than just the program and class diagram of the variables in the different datasets that will be used.

The flowchart in Figure 1 gives a breakdown of how the project will flow. At this point the problem has been found, relevant literature has been explored and an appropriate data set has been acquired. The remaining steps of the project are a part of the implementation chapter and are explored in more detail there and sections 3.4 and 3.5.

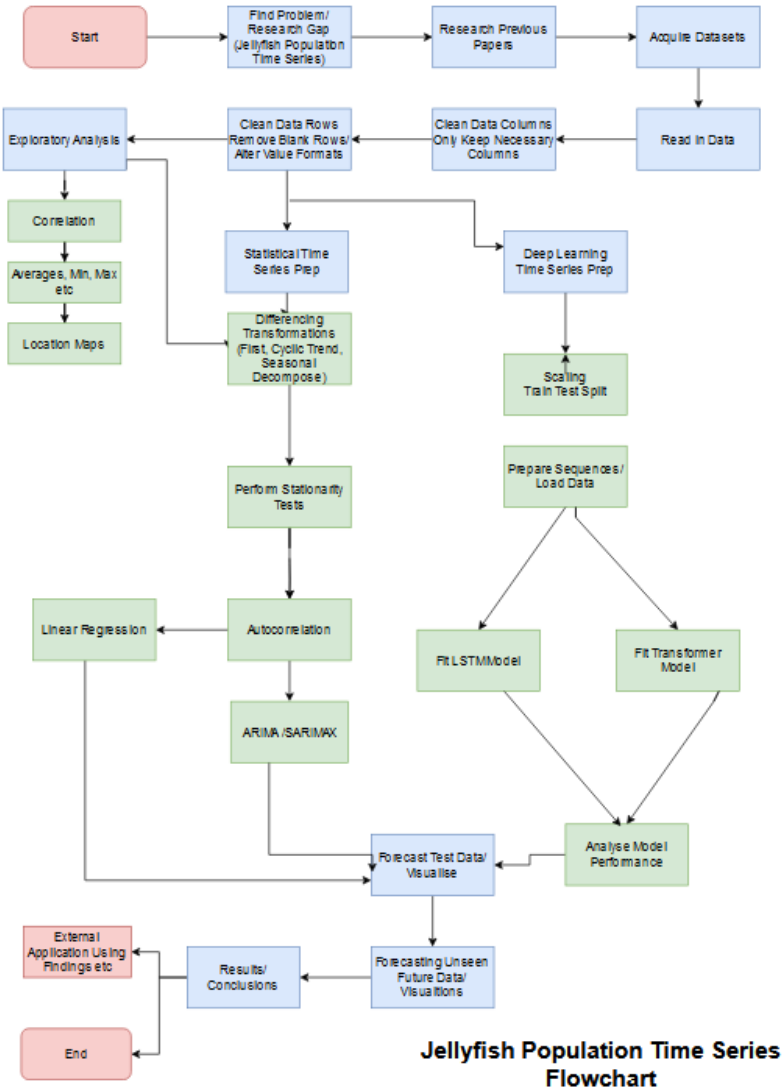


Figure 1: Jellyfish Population Time Series Flowchart

Figure 2 shows a class diagram for the project. The top dataset is the initially cleaned jellyfish dataset that is used for its descriptive statistics that don't reveal much about the actual time series population problem but can be highly useful for finding insights about the data such as what species are found and where. The second dataset is an analysis dataset that will be used for both the statistical and deep learning models. The most important value is the jellyfish population metric, but the different climate metrics are also shown. It is attained by combining the different datasets.

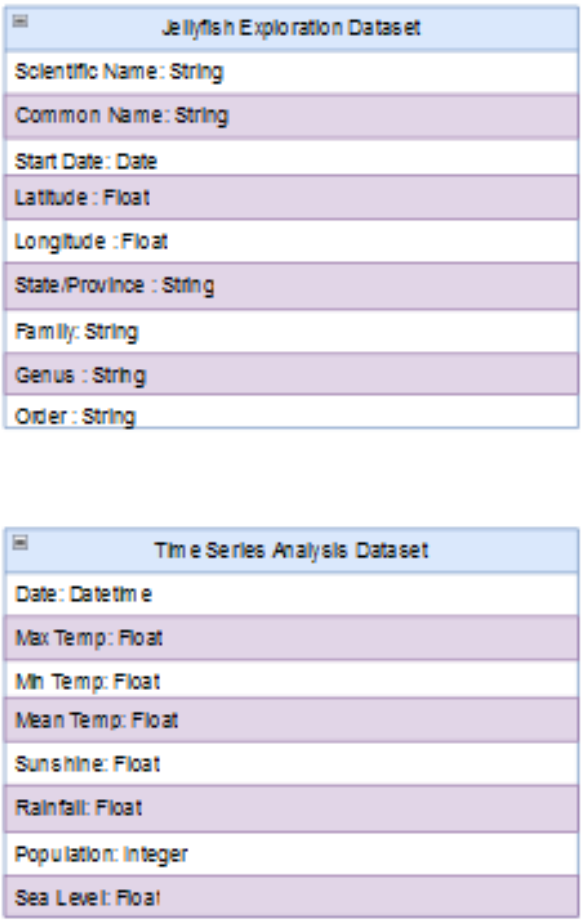


Figure 2: Class Diagram of the Jellyfish Exploration Dataset and the Dataset that will be used for Time Series Analysis

3.4. Dataset

3.4.1. Acquisition

The dataset for this project was acquired from the NBN Atlas (Marine Conservation Society 2024) which contains a detailed description of every sighting of a jellyfish around the coast of the UK from 2002 to 2022. There are 56 columns and over 17,000 rows present which provides a comprehensive breakdown of the sighting containing various measures of location like longitude and latitude. It also contains measures of date already broken down to year, month and day. Each record features information on the specimen age, vitality, depth, life stage and abundance which could provide interesting revelations about the type of jellyfish populations. Finally, there is some supplementary information that contains information about the survey, license, recorder and ID which will not be relevant to the exploratory analysis or time series investigation.

The climate data was acquired from (Met Office 2024) as no licence is required, it has a series of files that can be downloaded and tuned to your specifications. A broad region of the UK can be selected and a parameter for max, min and mean temperature as well as rainfall and sunshine. Each features data recorded from as early as 1884 up until the present day with a value for each month and each season. Another dataset from Kaggle (Koustubhk 2024) provides information from NASA on sea level rise from 1993 to 2021 roughly the same period the jellyfish observations occurred.

3.4.2. Cleaning

Data cleaning is a vital step in any data science project as there is usually irrelevant information or a lack of information present in the dataset. This dataset is no different, examining the dataset through Excel reveals that every single date record in the dataset is accounted for but there are 145 values of location coordinates missing from the data that will be dropped from the dataset because the information is too important to artificially implant. Some of the columns that could have provided some interesting information on the life stage, sex and vitality of the jellyfish have not been recorded so these columns will be dropped. The abundance of the jellyfish is less than half populated. The columns are either blank,1, 2 to 100 or 101 to 1000. As the ranges are broad this information will not be relied on much, but it could provide information on the increase in size of blooms over time.

The climate datasets from the Met Office likely will not require any cleaning as they are already suitable for modelling other than basic formatting and combining. The sea level dataset has a lot of extra information on the sea level concerning Global Isostatic Adjustment (GIA) and Global Mean Sea Level (GMSL) with standard deviations and smoothing attached, some of these columns will be redundant and not needed for the time series. There are no missing rows in the data but there is a disparity in how the years are recorded, with some periods not having as much information as others, so data may have to be scaled proportionally.

3.4.3. Data Exploration

Both the climate data and the jellyfish sightings have the potential to reveal interesting information before any time series takes place. This will include finding measures of central tendency and focusing the information on specific species, locations or by time and a mix of

all three. As the sightings have a location, they can be mapped to a map using a library like Folium, again this information could be filtered to species or period. Some basic line graphs will be constructed to show the trend in climate factors and potential trends in the number of sightings.

The different data points can also be explored from a time series perspective using decomposition which breaks down data into four components, the original, trend, seasonality and residual. The trend shows the long-term direction of data, seasonality refers to the short-term cycles like temperature increasing in the summer months and residual is the remaining information or outliers from the data.

3.5. Modelling

3.5.1. Statistical Time Series

When working with statistical time series there is an assumption that the temporal data is stationary as it makes models easier to implement and forecast from. Stationarity refers to a time series where mean, variance and autocorrelation remain constant for its duration. The mean and variance can be calculated at different times to see if they remain consistent and therefore stationary, but the best way is to perform the Augmented Dickey-Fuller (ADF) Test. Other tests like the Kwiatkowski-Philips-Schmidt-Shin (KPSS) test follow the same principles. Statsmodels has functions in its library that calculate values for ADF and KPSS which can be combined with simple if-else logic to state if data is stationary or non-stationary.

Should the null hypothesis be accepted, and the data be stationary, it will have to be transformed using different methods such as differencing, transforming and seasonal adjustments. Differencing also known as detrending is the subtraction of the previous observation from the current observation. This can be repeated multiple times to remove complex patterns from the data. Usually, the first and second differencing will suffice. Differencing can also be used to remove seasonality from the data by altering the diff brackets to contain the oscillation period such as 12. Time series data can also be transformed using NumPy functions like log and square root which can often make it stationary. The values are simply replaced with their log or square root counterparts. The trend in the data can be removed by utilizing the seasonal_decompose function and taking away the stored trend from the observed values, a detrended series is left behind. Statsmodels also has a function

hpfilter based upon the Hodrick-Prescott filter which separates the trend from the cycle. Not all these methods will ensure stationarity, they will have to be checked using the ADF or KPSS test or by viewing a graph. A stationary graph will have a constant mean, little variance and no trend or seasonality.

Autocorrelation is the linear measure between a current time series value and previous values with a time lag. Partial autocorrelation measures the same without the time lag. It is another method of testing for trend and seasonality but also for the testing of randomness of data. Autocorrelation is measured on the residuals of the time series data and if the model captured all the necessary information there should be no autocorrelation. It can be measured through the Ljung-Box test which states that the null hypothesis is there is no autocorrelation. Statsmodels has the function, acorr_ljungbox which will perform this test. The autocorrelation and partial autocorrelation can be plotted using the plot_acf and plot_pacf functions respectively which can reveal more information about how time series will capture the data.

Using the autocorrelation and partial autocorrelation found previously, the order parameter of ARIMA can be given initial values. Using SKLearn the data can be split into train and test sets and an ARIMA model can be fit and tested upon. The Mean Square Error and Root Mean Square Error are calculated and using matplotlib a line graph showing the training data, the predicted values and the actual values. If this proves to be accurate, it can be used to forecast future values that have not occurred yet. As climate and jellyfish have a seasonal factor, ARIMA may only provide the trend information and fail to consider differences in season, for this reason, there are multiple offshoots of ARIMA such as SARIMAX where S stands for Seasonal and the X refers to exogenous variables.

Like a standard model, different parameters can affect the results by increasing accuracy metrics or decreasing overfitting. Hyperparameters can be tuned to account for this and ARIMA models change the values for AR, I and MA. K-fold cross-validation is the standard method for this with regression and classification tasks, but it does not work for time series so an alternative time series split cross-validation or blocked-cross validation will be used. Time series split will examine the first fifth or fold amount of data and test on the next fifth and then repeat that process but take the first two-fifths and test on the third etc. Blocked

cross-validation instead will select a block for testing and the immediate data right after will be used for validation, the process is repeated always preserving order between train and test. Both can be seen in Figure 3.

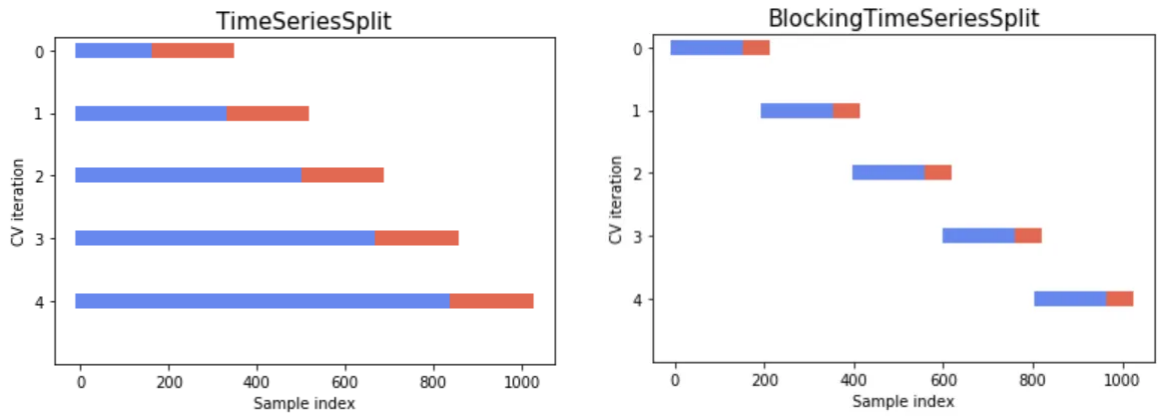


Figure 3: Time Series and Blocking Time Series Split

The best model for each dataset can be graphed using matplotlib. The plot_diagnostics function will give a further breakdown of the best model containing a residual plot, histogram and KDE plot, normal Q-Q plot and a Correlogram. The graph intends to show a good normal distribution of the residuals and check for any remaining autocorrelation if there still is, it means the model failed to capture all the dependencies. After verifying the model, a future forecast can be made with the data and the results can be graphed and compared to the deep learning models.

3.5.2. Deep Learning Time Series

Deep Learning Approaches take less preparation to effectively work than statistical approaches but there are still several steps that must be taken to ensure that the models can be implemented. This project uses both the Keras and PyTorch frameworks. Deep learning approaches require less work on the data before constructing the model, but the data will then have to be scaled, normalised and split into a training and testing set. Then each framework uses tensors or matrices, and the data will have to be transformed here again.

PyTorch can create a class for each of the different models intended to be used, they are explored below but should all follow a similar structure. Once each model has a class they will have to be trained and the same function should work for all with maybe minor tweaks. Inside

this function, the model will be selected, and an optimiser is selected to help reduce the loss function, PyTorch offers several through its optim packages such as Adam or Stochastic Gradient Descent which can all be tried. A loss function is also selected such as MSE and the data is placed in a DataLoader function. The number of epochs is selected, and a loop is defined with that number. Inside that loop, the model calls the train function and another loop based on the DataLoader is called where predictions are made based on the train set, the loss is calculated and a forward pass is made accordingly, the optimiser updates its gradients and a backwards pass is performed on these updated parameters and the optimiser is updated one final time using the step function.

After completing the inner loop, logic can determine how often to perform these next steps, every 50 epochs etc. The model is evaluated using eval, and another loop is called using a function that specifies no_grad so that the weights in the model do not update when testing. Predictions are made on the x_test data and compared to actual values to calculate an RMSE. This is compared to the training set where both should keep decreasing every epoch and converging closer together. Finally, the time series itself can be graphed highlighting train and test areas and could then be adapted to forecast into the future which can be compared against the different deep learning models as well as the ARIMA-based ones. The input data will then be altered to include the entire dataset, and the predictions will be on unseen values where a visualisation will be the only metric.

A basic LSTM would be defined as a class within a PyTorch module. An init method that calls the super method, the LSTM function with parameters suited to the data and its size and finally a fully connected layer to define a single output. A forward pass is defined which calls the LSTM and fully connected layer before returning the output. This model can be passed through the deep learning architecture highlighted above and the results and visualisations from an LSTM can be acquired and compared. The model and the individual architecture of the LSTM can be seen in Figure 4. The bottom diagram is the standard LSTM model featured across various sources and the top is a rough guide to how the overall model is constructed where the data passes through the LSTM and the fully connected layer to find one output (Staudemeyer and Morris 2019).

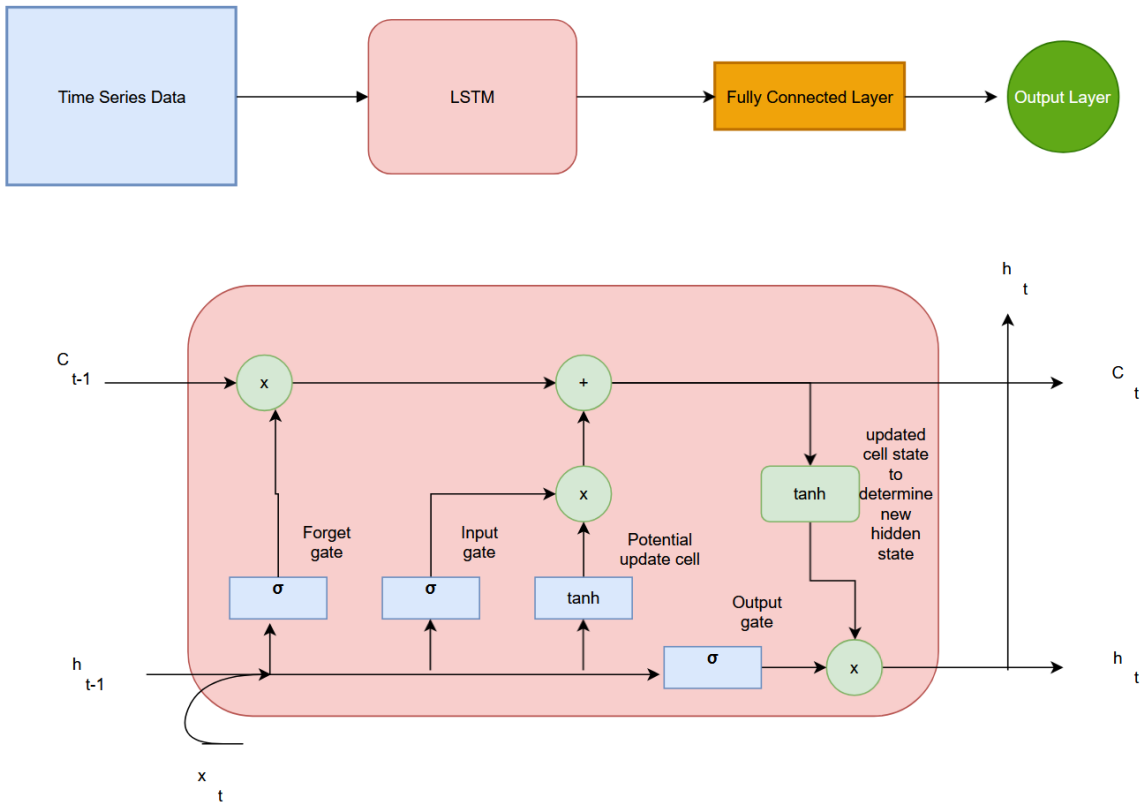


Figure 4: Long Short-Term Memory Model and Architecture

Transformers can also be implemented using PyTorch, the superclass for Transformer is first called. A fully connected layer which takes the input is defined followed by a transformer encoder layer which contains the multi-head self-attention mechanism, a feedforward neural network and layer normalisation. This is followed by a fully connected layer. A forward pass is defined which calls the embedding function, a permute function which prepares the input for the transformer encoder, which also uses the permute function to pass it through the fully connected layer and the last timestep in each sample is returned, see Figure 5 (Wen *et al.* 2023). The previously constructed training function should work here as well where the MSE, RMSE and visualisations can be analysed.

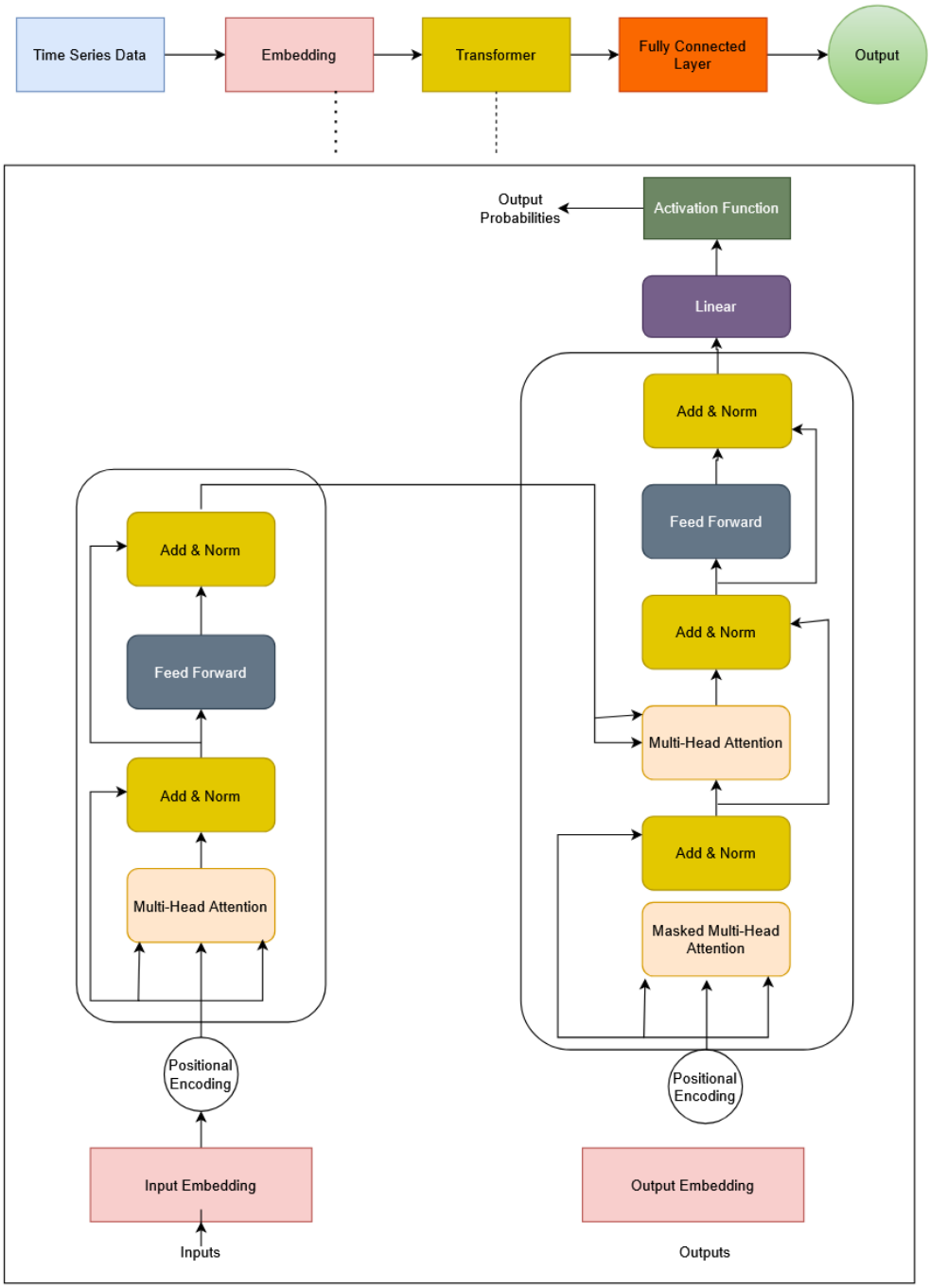


Figure 5: Transformer Model and Architecture

3.6. Conclusion

In Chapter 2, the literature that was reviewed revealed how the problem of jellyfish population modelling using updated deep learning time series has not been adapted based on recent years' advancements in the sector, Chapter 3 gave an insight into the thought process behind some of the previous projects that used statistical methods like ARIMA and

how they could be updated to Transformers and LSTM by following the structure laid out by others. Chapter 3 also highlighted the overall design of the project utilising various diagrams like a flowchart, class diagrams and graphs of the various models to be implemented to give a high-level overview of the potential solutions to the problem.

The following chapter will explore these ideas in detail as they are implemented placing greater emphasis on the parameters for the different models and how different parts of the project influence each other which will be more apparent in Chapter 5 when the results are discussed in detail using the jellyfish and climate datasets.

Chapter 4: Implementation

4.1. Introduction

The implementation chapter will build upon the work carried out in the design chapter by showcasing the steps taken to produce various models for different climate and jellyfish datasets using statistical time series methods like ARIMA, auto-correlation and differencing techniques as well as the deep learning approaches of Transformers and LSTM.

4.2. Environment Set Up

The initial requirements can be set up based on the device's specifications by ensuring that all the correct software is installed and configured to contain a version of Python and Jupyter Notebook IDE and that all the libraries needed are installed through pip. Alternatively, Google CoLab will have the environment pre-set up when creating a file and it is only important to install some of the libraries that have not been used before such as Folium. See Appendix C for GitHub containing all files.

Once an environment has been created the libraries and specialised functions can be imported into the program, as shown in Code Listing 1. It also shows the connection to mount the Google Drive where the necessary dataset files have been stored and can be easily accessed, this can be tweaked to suit the path to the files. Each of the datasets simply the function read_csv from Pandas and stores it to a variable called jellyfishdf for further exploration and analysis. The climate data features five different datasets, to read them in a loop was created that read them each in.

```python
# Fundamental packages for data manipulation and numerical operations
import numpy as np
import pandas as pd
import random

# Plotting and visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Interactive maps
import folium

# Handling warnings
import warnings

# PyTorch - Deep learning framework
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data

# TensorFlow - Deep learning framework
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Flatten, TimeDistributed
from tensorflow.keras.layers import Conv1D, MaxPooling1D

# Bokeh - Interactive visualization library
from bokeh.plotting import figure
from bokeh.io import output_notebook, show
from bokeh.models import CheckboxGroup, CustomJS
from bokeh.layouts import column
output_notebook()

# Scikit-learn - Machine learning library
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Mathematical functions and plotting utilities
from math import sqrt
from pandas.plotting import autocorrelation_plot

# Statsmodels - Statistical modeling and econometrics
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.filters.hp_filter import hpfilter
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.seasonal import seasonal_decompose, STL
```

```
ata Acquistion
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```python
jellyfishdf = pd.read_csv("/content/drive/MyDrive/Dissertation/records-2024-04-12.csv")
jellyfishdf
```

Code Listing 1: Imports and Google CoLab Mount

4.3. Data Cleaning

Different datasets require different data-cleaning approaches, whether it is removing columns, filling in missing values or type manipulation to perform the correct analysis.

4.3.1. Jellyfish Dataset

The jellyfish dataset contained many redundant columns of which, a large portion were blank or were not relevant to the problem such as Survey ID. Some of the columns that remained were the species name, genus, state, latitude and longitude and date. Some blank values were

filled with ‘Unknown’ for visualisation purposes as they will not be relevant for time series only for exploration and any rows with crucial information were dropped from the data frame, shown in Code Listing 2 with the cleaned dataset shown in Figure 6.

```
jellyfishdf = jellyfishdf[['Scientific name', 'Common name', 'Order', 'Family',
                           'Genus', 'State/Province', 'Start date',
                           'Start date month', 'Start date year', 'Latitude (WGS84)',
                           'Longitude (WGS84)', 'Coordinate uncertainty (m)', 'Abundance']]

# Fill blank abundances with 1
jellyfishdf['Abundance'] = jellyfishdf['Abundance'].fillna(1)

# Replace blank states/provinces with 'Unknown'
jellyfishdf['State/Province'] = jellyfishdf['State/Province'].fillna('Unknown')

# Replace blank coordinate uncertainties with 'Unknown'
jellyfishdf['Coordinate uncertainty (m)'] = jellyfishdf['Coordinate uncertainty (m)'].fillna('Unknown')

# Remove rows without latitude or longitude
jellyfishdf = jellyfishdf.dropna(subset=['Latitude (WGS84)', 'Longitude (WGS84)'])
```

Code Listing 2: Jellyfish Data Frame Cleaning Code

	Scientific name	Common name	Order	Family	Genus	State/ Province	Start date	Start date month	Start date year	Latitude (WGS84)	Longitude (WGS84)	Coordinate uncertainty (m)	Abundance
0	Rhizostoma octopus	Barrel Jellyfish	Rhizostomeae	Rhizostomatidae	Rhizostoma	England	2016-02-14	2	2016	51.096600	1.274400	70.7	1
1	Rhizostoma octopus	Barrel Jellyfish	Rhizostomeae	Rhizostomatidae	Rhizostoma	England	2016-02-02	2	2016	51.022600	0.994500	70.7	1
2	Rhizostoma octopus	Barrel Jellyfish	Rhizostomeae	Rhizostomatidae	Rhizostoma	Wales	2017-02-15	2	2017	52.838500	-4.493500	70.7	1
3	Chrysaora hysoscella	Compass jellyfish	Semaeostomeae	Pelagiidae	Chrysaora	England	2022-02-28	2	2022	50.788900	-1.088700	70.7	1
4	Cyanea capillata	Lion's Mane Jellyfish	Semaeostomeae	Cyaneidae	Cyanea	Wales	2020-02-09	2	2020	52.023900	-4.894100	70.7	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
17599	Chrysaora hysoscella	Compass jellyfish	Semaeostomeae	Pelagiidae	Chrysaora	England	2017-07-06	7	2017	50.614600	-3.412800	70.7	1
17600	Aurelia aurita	Moon jelly	Semaeostomeae	Ulmaridae	Aurelia	England	2005-07-28	7	2005	53.860145	-3.052877	70.7	101 to 1000
17601	Chrysaora hysoscella	Compass jellyfish	Semaeostomeae	Pelagiidae	Chrysaora	England	2014-07-29	7	2014	50.723847	-1.119472	70.7	1

Figure 6: Cleaned Jellyfish Data

A dataset for time series analysis was also extracted from the dataset by setting a data index and using the resample function from pandas set to month which counts the occurrences each month. See Code Listing 3.

```python
# Create a copy of the original DataFrame jellyfishdf to avoid modifying the original data.
jellyfishdf1 = jellyfishdf.copy()

# Convert the 'Start date' column to datetime format for accurate time-based operations.
jellyfishdf1['Start date'] = pd.to_datetime(jellyfishdf1['Start date'])

# Set the 'Start date' column as the index of the DataFrame.
# This allows for time-based operations like resampling and sorting.
jellyfishdf1.set_index('Start date', inplace=True)

# Sort the DataFrame by the index (which is now the 'Start date') in ascending order.
# This ensures that the data is ordered by date, which is important for time series analysis.
jellyfishdf1.sort_index(inplace=True)

# Resample the DataFrame to a monthly frequency ('M').
# The .size() function counts the number of occurrences in each month.
jellyfishcountdf = jellyfishdf1.resample('M').size()

# Display the resulting Series with the count of jellyfish occurrences per month.
jellyfishcountdf
```

Start date	0
2003-04-30	2
2003-05-31	16
2003-06-30	18
2003-07-31	49
2003-08-31	120
...	...
2022-04-30	9
2022-05-31	66
2022-06-30	224
2022-07-31	455
2022-08-31	102

233 rows × 1 columns

Code Listing 3: Jellyfish Count Data Frame Creation Code

4.3.2. Climate Dataset

Code Listing 4 shows the steps taken to clean the climate data, first, the pandas melt function is used to separate the measurable variables from the identifier variables. In this instance, it is used to separate years and months. The months are then mapped to their corresponding number i.e. January to 01. A variable called Date is then created which combines 01 with the month number and year, this final variable can then be converted to a datetime variable for any time series analysis. This is performed for each of the climate datasets using the exec function in a loop before they are all merged on their Date column. The final dataset removes anything past 2023 as it is not fully updated and aligns with the jellyfish data better. The resultant dataset is shown in Figure 7.

```python
def climate_data_cleaning(df, name):
    # Use pd.melt to unpivot the DataFrame from a wide format to a long format.
    # 'year' remains as is (id_vars), while the months are transformed into rows (value_vars).
    # 'var_name' specifies the name of the new column that will hold the month names.
    # 'value_name' specifies the name of the new column that will hold the corresponding values.
    df = pd.melt(df, id_vars=['year'], value_vars=['jan', 'feb', 'mar', 'apr', 'may', 'jun',
                                                   'jul', 'aug', 'sep', 'oct', 'nov', 'dec'],
                 var_name='Month', value_name=name)

    # Map the month abbreviations to their numeric equivalent using the provided mapping.
    # Assumes 'month_mapping' is a dictionary that maps month names to month numbers (e.g., 'jan' -> '01').
    df['Month'] = df['Month'].map(month_mapping)

    # Create a new 'Date' column by concatenating '01-' (day), the mapped month number, and the year.
    # This assumes that the day is the first of each month.
    df['Date'] = '01-' + df['Month'] + '-' + df['year'].astype(str)

    # Convert the 'Date' column to a datetime format to enable time-based operations.
    df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')

    # Reorder the DataFrame to have only the 'Date' and the variable of interest (name) columns.
    df = df[['Date', name]]

    # Sort the DataFrame by the 'Date' column to ensure chronological order.
    df = df.sort_values(by='Date')

    # Return the cleaned and transformed DataFrame.
    return df
```

```python
# Loop each variable in the climate list and apply the cleaning function
for i in climate_list:
  exec(f'{i}df = climate_data_cleaning({i}df, "{i}")')
```

```python
# Assign the DataFrame MaxTempdf to climatedf, which will serve as the base DataFrame for merging.
climatedf = MaxTempdf

# Loop through each item in the climate_list, which presumably contains the names of different climate-related DataFrames.
for i in climate_list:

    # If the current item in the list is not 'MaxTemp', proceed with the merging operation.
    if i != 'MaxTemp':

        # Use exec to dynamically merge the current DataFrame (e.g., MinTempdf, Precipdf, etc.) with climatedf.
        # The merge is performed on the 'Date' column, which is assumed to be common across all DataFrames.
        exec(f'climatedf = pd.merge(climatedf, {i}df, on="Date")')

    # If the current item is 'MaxTemp', skip the merging operation (since it's already the base DataFrame).
    else:
        continue

# Filter the combined DataFrame to include only rows where the year in the 'Date' column is less than 2023.
climatedf = climatedf[climatedf['Date'].dt.year < 2023]
```

Code Listing 4: Climate Data Frame Creation, Combining and Cleaning Code

```
climatedf
```

	Date	MaxTemp_x	MaxTemp_y	MinTemp	MeanTemp	Sunshine	Rainfall
0	1910-01-01	5.3	5.3	-0.4	2.5	50.9	109.1
1	1910-02-01	6.8	6.8	0.7	3.8	72.2	122.6
2	1910-03-01	9.0	9.0	1.4	5.2	130.3	49.7
3	1910-04-01	9.8	9.8	2.0	5.9	122.8	93.4
4	1910-05-01	14.3	14.3	5.5	9.9	190.5	70.0
...	...	...	...	...	...	...	...
1351	2022-08-01	21.7	21.7	11.6	16.6	208.4	51.9
1352	2022-09-01	17.4	17.4	9.4	13.4	118.6	105.2
1353	2022-10-01	14.9	14.9	8.3	11.6	104.7	147.5
1354	2022-11-01	10.9	10.9	5.5	8.2	55.0	167.2
1355	2022-12-01	5.9	5.9	-0.1	2.9	49.4	116.0

Figure 7: Climate Data Cleaned

4.3.3. Sea Level Dataset

The sea level dataset only involved removing some of the columns, the cleaned dataset is shown in Code Listing 5.

```
sealeveldf = sealeveldf[['Year', 'GMSL_GIA']]
sealeveldf
```

	Year	GMSL_GIA
0	1993	-38.59
1	1993	-41.97
2	1993	-41.91
3	1993	-42.65
4	1993	-37.83
...	...	...
1043	2021	56.17
1044	2021	57.42
1045	2021	56.57
1046	2021	54.41
1047	2021	57.01

Code Listing 5: Sea Level Cleaning

4.4. Data Exploration

Data Exploration is the process that is initially carried out to understand the characteristics of the data using various statistical measures and visualisation techniques.

4.4.1. Jellyfish Dataset

The jellyfish dataset has the most interesting data exploration to be carried out due to the abundance of information available from the different columns. Most of this is irrelevant to the time series aspect but it can give some interesting insights into the location and species

information that may be interesting to other projects. Some basic checks were performed on the data using the pandas info and describe methods which ensured that variables were present and consolidated different measures of central tendency like mean and standard deviation for columns that contained numerical information.

The categorical data was analysed using the value_counts method which counts occurrences of a string. Different variables like scientific name, state and family were used and variables were displayed. Building upon this, each of the value_counts was output as a pie chart. Code Listing 6 shows the function that was used to create labels and data based on the value_counts label and index. Using matplotlib functions the wedges of the pie were created and placed on the graph, tweaking some parameters for visual improvements. The if statement is used for combining the scientific and common names of a jellyfish species on the graph as they will have the same values. The function could then be called by specifying the data frame, column, title, image name and the flag if it was needed.

```python
def create_pie_chart(df, coloumn, title, file, flag=False):
    # Extract the labels and data for the pie chart.
    # 'labels' are the unique values in the specified column, and 'data' are their respective counts.
    labels, data = df[coloumn].value_counts().index, df[coloumn].value_counts().values

    # Create an 'explode' array to slightly separate the slices of the pie chart.
    # The separation value ranges linearly from 0 to 0.2 based on the number of unique values.
    explode = np.linspace(0, 0.2, data.shape[0])

    # Create a figure and axis object for the plot with a specified size.
    fig, ax = plt.subplots(figsize=(10, 10))

    # If the 'flag' is set to True, modify the labels.
    # This can be useful for adding additional information (like mapped names) to the labels.
    if flag:
        labels = [label + "/" + name_map[label] for label in labels]
        # Adjust the subplots to ensure the modified labels fit within the figure.
        plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)

    # Create the pie chart:
    # - 'autopct' displays the percentage of each slice.
    # - 'startangle' rotates the start of the pie chart by 140 degrees for better visual appeal.
    # - 'pctdistance' adjusts the position of the percentage text.
    # - 'explode' separates the slices of the pie chart slightly.
    wedges, texts, autotexts = ax.pie(data, autopct='%1.1f%%', startangle=140, pctdistance=0.7, explode=explode)

    # Add a legend to the chart:
    # - 'loc' sets the location of the legend.
    # - 'bbox_to_anchor' positions the legend outside the pie chart for better readability.
    ax.legend(labels, loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))

    # Set the title of the pie chart.
    plt.title(title)

    # Ensure the pie chart is a circle by setting the aspect ratio to 'equal'.
    ax.axis('equal')

    # Save the pie chart as an image file with the given file name.
    plt.savefig(file)

    # Display the pie chart.
    plt.show()
```

Code Listing 6: Pie Chart Function Code

Each of the jellyfish data points contains a latitude and location, this can be mapped using a Python library such as Folium to display the map with markers at each recording point. Code Listing 7 shows the map code, a colour map was first created to differentiate each of the species. The latitude and longitude averages were found, and this would be the centre of the map. The map was created with a zoom over the UK and Ireland and a loop was created that placed a marker at each location and coloured according to the species.

```python
# Dictionary mapping scientific names of jellyfish species to specific colors for marker icons on the map.
color_mapping = {
    'Rhizostoma octopus': 'red',
    'Chrysaora hysoscella': 'blue',
    'Cyanea capillata': 'green',
    'Cyanea lamarckii': 'orange',
    'Aurelia aurita': 'purple',
    'Pelagia noctiluca': 'pink',
    'Physalia physalis': 'yellow',
    'Velella velella': 'black'
}

# Calculate the mean latitude and longitude to set the center of the map.
# 'Latitude (WGS84)' and 'Longitude (WGS84)' columns are assumed to contain the coordinates of jellyfish sightings
map_center = [jellyfishdf['Latitude (WGS84)'].mean(), jellyfishdf['Longitude (WGS84)'].mean()]

# Create a Folium map centered on the calculated mean coordinates with a starting zoom level of 5.
m = folium.Map(location=map_center, zoom_start=5)

# Iterate over each row in the jellyfish DataFrame to place markers on the map.
for index, row in jellyfishdf.iterrows():

    # Add a marker to the map at the latitude and longitude specified in the current row.
    # The marker's popup displays the common name of the jellyfish species.
    # The marker's icon color is determined by the species' scientific name using the color_mapping dictionary.
    # If the species is not in the color_mapping dictionary, the icon color defaults to gray.
    folium.Marker(
        [row['Latitude (WGS84)'], row['Longitude (WGS84)']],
        popup=row['Common name'],
        icon=folium.Icon(color=color_mapping.get(row['Scientific name'], 'gray'))
    ).add_to(m)

# Display the interactive map.
m
```

Code Listing 7: Folium Map Code

4.4.2. Climate Dataset

The climate data featured the same info and describe methods to quickly gather some relevant statistics. Code Listing 8 shows the code used to create frequency histograms to gather some information about the frequency of the variables. Sea level featured the same explorations as climate.

```
# Iterate over each column in the DataFrame 'climatedf'.
for col in climatedf.columns:

    # Skip the 'Date' column, as we are not interested in plotting its distribution.
    if col != 'Date':

        # Create a new figure for each column with a specified figure size.
        plt.figure(figsize=(10, 6))

        # Use Seaborn's histplot function to plot the distribution of the column's values.
        # 'kde=True' adds a Kernel Density Estimate (KDE) curve to the histogram, which represents the data's distribution.
        sns.histplot(climatedf[col], kde=True)

        # Set the title of the plot to indicate which column's distribution is being shown.
        plt.title(f'Distribution of {col}')

        # Label the x-axis with the column name.
        plt.xlabel(col)

        # Label the y-axis as 'Frequency', indicating that the y-axis shows the frequency of values in each bin.
        plt.ylabel('Frequency')

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

Code Listing 8: Histogram Distribution Plot Code

4.5. Time Series Exploration

As this is a time series project, analysing the project from a time series aspect will reveal much more interesting and relevant data than a standard exploration. Some of these steps are even necessary to properly perform the statistical methods needed to achieve a good model using ARIMA-based models.

4.5.1. Stationarity

Stationarity in its simplest terms is a series without trend, constant variance, constant autocorrelation and seasonality. If is series is not stationary, it is extremely difficult to detect any trends and without it, any models will not successfully understand any underlying patterns and accurately forecast data. Stationarity can be tested using a hypothesis test on the Augmented Dickey-Fuller Test or ADF test and the Kwiatkowski, Phillips, Schmidt, and Shin or KPSS test. Both tests for stationarity by the null hypothesis are that they are not stationary. The ADF tests checks to see if the p-value is less than 0.5 and the KPSS checks if the p-value is greater than 0.5. In both these scenarios, the null hypothesis is rejected or accepted to determine the stationarity. Statsmodels offers functions for both these tests and combined with if-else logic, stationarity of a series can be found. Code Listing 9 shows this. Each of the different datasets called upon the functions and the stationarity of the dataset is determined, see Code Listing 10.

```python
def stationarity_test(output, test='adf'):
    # Extract values from the output, which is expected to be a tuple or list of results from a stationarity test.
    pval = output[1]  # p-value of the test
    test_score = output[0]  # test statistic score
    lags = output[2]  # number of lags used in the test

    # Initialize the decision as 'Non-Stationary'.
    decision = 'Non-Stationary'

    # Process based on the test type.
    if test == 'adf':
        critical = output[4]  # critical values for the ADF test
        # ADF Test: If p-value < 0.05, reject the null hypothesis of non-stationarity.
        if pval < 0.05:
            decision = 'Stationary'
    elif test == 'kpss':
        critical = output[3]  # critical values for the KPSS test
        # KPSS Test: If p-value >= 0.05, fail to reject the null hypothesis of stationarity.
        if pval >= 0.05:
            decision = 'Stationary'

    # Create a dictionary to store the test results.
    output_dict = {
        'Test Statistic': test_score,
        'p-value': pval,
        'Numbers of lags': lags,
        'decision': decision
    }

    # Add critical values to the output dictionary.
    for key, value in critical.items():
        output_dict["Critical Value (%s)" % key] = value

    # Convert the dictionary to a pandas Series and name it with the test type.
    return pd.Series(output_dict, name=test)
```

Code Listing 9: Stationarity Test Function

```python
adf_output = adfuller(jellyfishcountdf)
kpss_output = kpss(jellyfishcountdf)

pd.concat([stationarity_test(adf_output), stationarity_test(kpss_output, test='kpss')], axis=1)
```

Code Listing 10: Stationarity Test Function Calling

4.5.2. Differencing

If a series dataset is found to be not stationary, some processes transform the data to ensure stationarity. This is known as differencing and aims to reduce the mean which has the benefit of removing trend and seasonality. Code Listing 11 shows a function that performs all the different transformations so they can be graphed. A series of differencing can be done by using the pandas diff and dropna function. The diff function finds the difference between variables in a series and dropna removes any NaN variables left as a result. This process can repeat but this example just uses the first and second differencing. A deseasonalized version can be extrapolated by setting the diff parameter to 12 to work off 12 months. A log and

square root transformation can be applied to the variables as well. The rolling mean can be calculated using the rolling function which can be subtracted from the initial variable to get a stationary series. Using the hp filter, a series can be extracted into its trend and extract. These variables are all returned for graphing and stationarity tests.

```python
def differencing_function(df, col=None):
    """
    Applies various transformations and differencing methods to a time series to help achieve stationarity and extract features.

    Parameters:
    df (pd.DataFrame or pd.Series): Input DataFrame or Series containing time series data.
    col (str, optional): Column name to apply transformations if input is a DataFrame. If None, applies to the entire DataFrame or Series.

    Returns:
    tuple: A tuple containing different transformations and differencing results.
    """
    # If a column name is provided, extract the corresponding Series from the DataFrame
    if col is not None:
        variable = df[col]
    else:
        # If no column is provided, assume the input is a Series and use it directly
        variable = df

    # First order differencing
    first_diff = variable.diff().dropna()

    # Second order differencing
    second_diff = first_diff.diff().dropna()

    # Seasonal differencing (assumes a seasonal period of 12)
    diseasonalized_df = variable.diff(12).dropna()

    # Rolling mean to smooth the series
    rolling_mean = variable.rolling(window=12).mean()

    # Subtract the rolling mean from the original variable to detrend it
    subt_roll_mean = variable - rolling_mean
    subt_roll_mean.dropna(inplace=True)

    # Log transformation to stabilize variance
    log_transform = np.log(variable)
    log_transform.dropna(inplace=True)

    # Square root transformation to stabilize variance
    square_root = np.sqrt(variable)
    square_root.dropna(inplace=True)

    # Decompose the time series into observed, trend, and seasonal components
    decompose_result = seasonal_decompose(variable, model='additive')

    # Extract the deseasonalized data by removing the trend component
    sd_dtrend = (decompose_result.observed - decompose_result.trend)
    sd_dtrend.dropna(inplace=True)

    # Apply Hodrick-Prescott filter to extract cyclical and trend components
    cyclic_extract, trend = hp_filter(variable)
    cyclic_extract.dropna(inplace=True)
    trend.dropna(inplace=True)

    # Return all results as a tuple
    return first_diff, second_diff, diseasonalized_df, subt_roll_mean, log_transform, square_root, sd_dtrend, cyclic_extract, trend
```

Code Listing 11: Differencing Function - Transforms Data into First Difference, Rolling Mean, Cyclic Extract etc

Code Listing 12 shows a simplified stationarity test that removes some information and a function for plotting each of the transformations and checking their stationarity. Each method plots on a 2*X figure with the transformed data and their stationarity. The function is called like in Code Listing 13 for each dataset. Log transform had to be left because you cannot have a zero lag which the data has.

```python
# Function to check the stationarity of a time series using KPSS and ADF tests
def check_stationarity(df):
    # Perform the KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test on the data
    kps = kpss(df)

    # Perform the ADF (Augmented Dickey-Fuller) test on the data
    adf = adfuller(df)

    # Extract p-values from the test results
    kpss_pv = kps[1]  # p-value from the KPSS test
    adf_pv = adf[1]  # p-value from the ADF test

    # Initialize decisions based on the test results
    kpssh, adfh = 'Stationary', 'Non-stationary'

    # KPSS test decision
    # If p-value < 0.05, reject the null hypothesis that the data is stationary
    if adf_pv < 0.05:
        adfh = 'Stationary'  # Data is considered stationary if ADF test p-value is less than 0.05

    # ADF test decision
    # If p-value < 0.05, reject the null hypothesis that the data is non-stationary
    if kpss_pv < 0.05:
        kpssh = 'Non Stationary'  # Data is considered non-stationary if KPSS test p-value is less than 0.05

    # Return the results as a tuple
    return (kpssh, adfh)

# Function to plot and compare different methods applied to a time series
def plot_comparison(methods, col, plot_type='line'):
    # Calculate the number of rows needed for the subplot grid
    n = len(methods) // 2 + 1

    # Create a subplot grid with 'n' rows and 2 columns, sharing x-axis, and set figure size
    fig, ax = plt.subplots(n, 2, sharex=True, figsize=(20, 10))

    # Iterate over each method to plot
    for i, method in enumerate(methods):
        # Remove missing values from the data
        method.dropna(inplace=True)

        # Retrieve the name of the current method (assuming methods are global variables)
        name = [n for n in globals() if globals()[n] is method]

        # Calculate the position of the subplot
        v, r = i // 2, i % 2

        # Check stationarity for the current method
        kpss_s, adf_s = check_stationarity(method)

        # Plot the data for the current method
        method.plot(kind=plot_type,
                    ax=ax[v, r],  # Specify the subplot to plot on
                    legend=False,  # Do not show the legend
                    title=f'{col} {name[0].upper()} KPSS={kpss_s}, ADF={adf_s}')  # Set plot title

        # Set the size of the plot title
        ax[v, r].title.set_size(14)

        # Plot the rolling mean with a window of 12 periods on the same subplot
        method.rolling(12).mean().plot(ax=ax[v, r], legend=False)
```

Code Listing 12: Stationarity Test Simplified and Differencing Graph Code

```python
# Apply the differencing_function to the jellyfish count data
# This will return several transformed versions of the original time series
first_diff, second_diff, diseasonalized_df, subt_roll_mean, log_transform, square_root, sd_dtrend, cyclic_extract, trend = differncing_function(jellyfishcountdf)

# List of transformed time series to be compared
# Note: 'log_transform' is commented out. Uncomment if you wish to include it in the comparison.
methods = [first_diff, second_diff, diseasonalized_df, subt_roll_mean, square_root, sd_dtrend, cyclic_extract] #, log_transform]

# Plot and compare the specified transformations
plot_comparison(methods, 'Jellyfish Count')

# Suppress warnings to avoid clutter in the output
warnings.simplefilter(action='ignore')
```

Code Listing 13: Differencing Function Calling and Plotting Code

4.5.3. Regression

Code Listing 14 shows the code to perform ordinary least squares linear regression on the data. This is done to achieve a baseline model for when the more complex ARIMA models are developed. It can be useful to analyse trend patterns as well. This example uses the ADF test and by using resols.summary metrics on regression can be found mainly in the r-squared value. It was performed for each relevant data point.

```
# Apply the ADF test on the differenced time series and drop any NaN values
adf_result = adfuller(jellyfishcountdf.diff().dropna(), store=True)
# Print the full result of the ADF test
print(f'{adf_result}')
# Print the summary of the ADF test results
print(f'\nJellyfish Count\n{adf_result[-1].resols.summary()}')
```

Code Listing 14: OLS Regression Model Summary Code

4.5.4. Autocorrelation

In Code Listing 15, the code for plotting the autocorrelation and partial autocorrelation is shown. Using matplotlib a figure with the partial autocorrelation below the autocorrelation is created. The data frame is placed in the first differenced state to make stationary and the functions plot_acf and plot_pacf are utilised. The Ljung-Box test is also used to check for autocorrelation in the residuals and the result is returned.

```python
def autocorrplot(df, col):
    """
    Generate and display autocorrelation and partial autocorrelation plots for the given column
    after differencing the data, and perform the Ljung-Box test.

    Parameters:
    df (pd.DataFrame): DataFrame containing the data.
    col (str): The column name in the DataFrame to analyze.

    Returns:
    pd.DataFrame: Results of the Ljung-Box test.
    """
    # Create a figure with two subplots (one for ACF and one for PACF)
    fig, ax = plt.subplots(2, 1, figsize=(15, 10))

    # Compute the first difference of the data to make it stationary
    df_diff = df.diff().dropna()

    # Plot the Autocorrelation Function (ACF) on the first subplot
    plot_acf(df_diff, zero=False, ax=ax[0], title=f'{col} Autocorrelation')

    # Plot the Partial Autocorrelation Function (PACF) on the second subplot
    plot_pacf(df_diff, zero=False, ax=ax[1], title=f'{col} Partial Autocorrelation')

    # Perform the Ljung-Box test to check for autocorrelation in residuals
    # Test is performed on the differenced series with a maximum of 12 lags
    ljung_box_result = acorr_ljungbox(df_diff, lags=12)

    # Return the results of the Ljung-Box test
    return ljung_box_result
```

Code Listing 15: Autocorrelation/Partial Autocorrelation Plot Code

4.6. Statistical Time Series

After exploring both the metrics and time series exploration on the data, statistical time series like ARIMA and SARIMAX can be performed on the data to achieve quantifiable results that can be compared to the deep learning approaches. To successfully do this, the best parameters for a model must be found and using these parameters create an ARIMA-based model, visualise using training and test data and finally make a future prediction based on all the data.

4.6.1. Model Evaluation

Code Listings 16, 17 and 18 show three functions that are used to find the best order for the AR, I and MA aspects of the data. Code Listing 16, best_order_function is the first function and takes a series as its input. Outside the loop values of p, d, and q are decided where P is the AR value etc. Using a for loop each combination is looped and using the TimeSeriesSplit function a train and test set is created with 5 splits. The first split will be a fifth of the data, the second two fifths etc. The function then calls upon the evaluate_arima_model, shown in

Code Listing 17, which takes the current order and train test split fits the model and calculates the error before returning it where it is added to the error list for each order before an average is calculated and added to a results list. The minimum value of the results list is taken as the best order and output on the screen. Finally, in Code Listing 18, the newly found best order is verified by performing the same time series split and finding the average RMSE of all the splits and output the RMSE for each and fold and the final average where the lower the value the better.

```
def evaluate_arima_model(train, test, arima_order):
    """
    Evaluate the ARIMA model by fitting it to the training data, forecasting on the test data,
    and calculating the Mean Squared Error (MSE) between the forecasted values and the actual test data.

    Parameters:
    train (pd.Series): The training data series.
    test (pd.Series): The test data series.
    arima_order (tuple): The order of the ARIMA model (p, d, q).

    Returns:
    float: The Mean Squared Error of the model's predictions.
    """
    # Initialize the ARIMA model with the given order
    model = ARIMA(train, order=arima_order)

    # Fit the model to the training data
    model_fit = model.fit()

    # Forecast the values for the length of the test data
    predictions = model_fit.forecast(steps=len(test))

    # Calculate the Mean Squared Error between the actual and predicted values
    error = mean_squared_error(test, predictions)

    return error

# Define potential values for p, d, and q parameters of the ARIMA model
p_values = [0, 1, 2, 3]
d_values = [0, 1]
q_values = [0, 1, 2]

# Initialize the TimeSeriesSplit object with 5 splits
tscv = TimeSeriesSplit(n_splits=5)
```

Code Listing 16: Evaluate ARIMA Function Returns the Error of a Specified ARIMA Order

```python
def best_order_function(time_series):
    """
    Determine the best ARIMA model order by performing cross-validation and selecting the order with the lowest average CV error.

    Parameters:
    time_series (pd.Series): The time series data for which to find the best ARIMA model order.

    Returns:
    tuple: The best ARIMA model order (p, d, q) based on cross-validation.
    """
    results = []

    # Loop through the different combinations of p, d, and q values
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p, d, q)
                cv_errors = []

                # Perform time series cross-validation
                for train_index, test_index in tscv.split(time_series):
                    train, test = time_series[train_index], time_series[test_index]
                    try:
                        # Evaluate the ARIMA model for the current order and store the error
                        error = evaluate_arima_model(train, test, order)
                        cv_errors.append(error)
                    except:
                        # Handle any exceptions that occur during model evaluation
                        continue

                # Calculate the average cross-validation error for the current order
                avg_error = np.mean(cv_errors)
                results.append((order, avg_error))
                print(f'Order {order} - CV Error: {avg_error}')

    # Find the best order with the lowest average cross-validation error
    best_order, best_error = min(results, key=lambda x: x[1])
    print(f'Best Order: {best_order} with CV Error: {best_error}')

    return best_order
```

Code Listing 17: Best Order Function Finds the Best Order by Looping Possible Patterns

```python
def best_order_rmse(best_order, df):
    """
    Evaluate the Root Mean Squared Error (RMSE) of an ARIMA model with the best order
    using time series cross-validation.

    Parameters:
    best_order (tuple): The order of the ARIMA model (p, d, q).
    df (pd.Series): The time series data.

    Returns:
    None: Prints the RMSE for each fold and the mean RMSE across all folds.
    """
    # Initialize TimeSeriesSplit object with 5 splits
    tscv = TimeSeriesSplit(n_splits=5)

    # List to store RMSE values for each cross-validation fold
    rmse = []

    # Loop through each train-test split
    for train_index, test_index in tscv.split(df):
        # Split the data into training and test sets
        cv_train, cv_test = df.iloc[train_index], df.iloc[test_index]

        # Fit the ARIMA model with the specified order to the training data
        arma = ARIMA(cv_train, order=best_order).fit()

        # Predict the values for the test data period
        predictions = arma.predict(cv_test.index.values[0], cv_test.index.values[-1])

        # Extract the true values from the test data
        true_values = cv_test.values

        # Calculate the RMSE for this fold and append it to the list
        fold_rmse = sqrt(mean_squared_error(true_values, predictions))
        rmse.append(fold_rmse)

    # Print the RMSE for each fold
    print(f'RMSE: {rmse}')

    # Print the mean RMSE across all folds
    print(f'Mean RMSE: {np.mean(rmse)}')
```

Code Listing 18: Function that performs time series cross-validation and finds the average RMSE

After the RMSE of the best order for each model has been achieved, a plot_diagnostics function from statsmodels outputs four graphs, for the residuals, a histogram featuring the Kernel Density Estimate or KDE, a qq plot on normality and finally an autocorrelation plot. This is shown in Code Listing 19. Code Listing 20 shows the code for a separate autocorrelation plot which plots the autocorrelation as a line instead of points at lagged intervals.

```
# Fit the ARIMA model to the jellyfish count data with the specified best order
model = ARIMA(jellyfishcountdf, order=jellyfish_best_order).fit()

# Plot the diagnostic plots of the fitted ARIMA model
# These plots help to check the residuals for model validation
model.plot_diagnostics(figsize=(15, 12))

# Set the title of the diagnostic plots
plt.title(f'Jellyfish Count')

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

Code Listing 19: Plots the residuals, histogram, normality and autocorrelation of the best-order

```
# Plot the autocorrelation function (ACF) of the jellyfish count time series
autocorrelation_plot(jellyfishcountdf)

# Set the title of the ACF plot
plt.title(f'Jellyfish Count')

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

Code Listing 20: Autocorrelation Plot Code

4.6.2. Model Visualisation

The next step in statistical time series is to plot the time series and make predictions based on this information. A series of functions were created to split the data, make predictions, forecast test values, and forecast unknown future values.

Code Listing 21 shows the code for creating a training and test data graph. It uses basic matplotlib plot functions to plot the train data, test data, labels, legend and title before returning the plot. Code Listing 22 displays a prediction function that takes the model and test data to get a series of predictions or y_preds for evaluating a model. The get_forecast function returns these predictions and sets the confidence interval to 0.05 with the conf.int function. The predictions are then placed in a data frame with actual test values before being returned to the forecast function. Code Listing 23 shows this forecast function which takes a data frame, column, best order, and data range set at a default value and a title with a default value. A train and test data frame are created by splitting the data frame at the date variable. The train_test_graph function is then called, an ARIMA model is created for the best_order and the training data is fit, a model summary is displayed, and the predictions are returned from calling the predictions function. The ARIMA predictions are added to the plot with a

green line and the RMSE is also displayed. The previous steps for ARIMA are then repeated for SARIMAX and finally, the legend is added before the final graph is output. These steps were performed for the different datasets under several of the differencing techniques like first differencing or cyclic trend and extract.

```python
# Function to plot training and testing data
def train_test_graph(train, test, col, title):
    # Create a figure with a specific size
    plt.figure(figsize=(15, 6))

    # Plot training data
    plt.plot(train, label='Train')

    # Plot testing data
    plt.plot(test, label='Test')

    # Set the y-axis label
    plt.ylabel(f'{col} Change')

    # Set the x-axis label
    plt.xlabel('Date')

    # Rotate x-axis labels for better readability
    plt.xticks(rotation=45)

    # Set the plot title
    plt.title(f"Train/Test split for {col} : {title}")

    # Return the plot object for further manipulation if needed
    return plt
```

Code Listing 21: Graph the Train and Test Split of a Data Frame Function

```python
# Function to generate predictions from a fitted ARIMA or SARIMAX model
def predictions(model, test):
    # Get forecast and confidence intervals from the model
    y_pred_df = model.get_forecast(steps=len(test.index)).conf_int(alpha=0.05)

    # Add predictions to the DataFrame
    y_pred_df["Predictions"] = model.predict(start=y_pred_df.index[0], end=y_pred_df.index[-1])

    # Align the index of predictions with the test data
    y_pred_df.index = test.index

    # Extract predictions as a Series
    y_pred_out = y_pred_df["Predictions"]

    return y_pred_out
```

Code Listing 22: Function to Find the Predictions for the Test set

```python
# Function to fit models, generate forecasts, and plot results
def forecast(df, col, best_order, date_range="2017-01", title=None):
    # Split the data into training and testing sets
    train = df[df.index < pd.to_datetime(date_range, format='%Y-%m')]
    test = df[df.index >= pd.to_datetime(date_range, format='%Y-%m')]

    # Plot the train/test split
    train_test_graph(train, test, col, title)

    # Fit ARIMA model
    ARIMA_model = ARIMA(train, order=best_order).fit()
    print(f'{ARIMA_model.summary()}')

    # Get ARIMA model predictions
    y_pred_ARIMA = predictions(ARIMA_model, test)
    plt.plot(y_pred_ARIMA, color='green', label='ARIMA Predictions')

    # Calculate and print ARIMA RMSE
    ARIMA_RMSE = sqrt(mean_squared_error(test, y_pred_ARIMA))
    print(f'ARIMA RMSE: {ARIMA_RMSE}')

    # Fit SARIMAX model
    SARIMAX_model = SARIMAX(train, order=best_order, seasonal_order=(0, 0, 0, 12)).fit()
    print(f'{SARIMAX_model.summary()}')

    # Get SARIMAX model predictions
    y_pred_SARIMAX = predictions(SARIMAX_model, test)
    plt.plot(y_pred_SARIMAX, color='red', label='SARIMAX Predictions')

    # Calculate and print SARIMAX RMSE
    SARIMAX_RMSE = sqrt(mean_squared_error(test, y_pred_SARIMAX))
    print(f'SARIMAX RMSE: {SARIMAX_RMSE}')

    # Show the legend for the plot
    plt.legend()

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

Code Listing 23: Function to create a time series model, find predictions, RMSE and plot the results

After forecasting different stationary series, the best results can be forecast into the future by learning from all the data and making predictions for the next 24 months after the last data point which for the data will bring it to the end of 2024 as the data goes till the end of 2022. Code Listing 24 shows the code for future_forecast which takes the data frame, column best order and title. A SARIMAX model is fit for the entire data frame. A range of future dates is created in a separate data frame by taking the last value of the original data frame, offsetting it setting periods of 24 and ensuring the frequency is monthly. The get_forecast method is called again on the model and the confidence intervals are also acquired. The forecasted values are added to a data frame with the future dates. This is plotted using matplotlib and the fill_between function to graph the confidence intervals before adding the title, labels, and

legend and showing the final graph. Again, the future_forecasting function can be used with any stationary data set for several results.

```python
# Function to generate and plot future forecasts
def future_forecast(df, col, best_order, title):
    # Fit SARIMAX model on the full dataset
    SARIMAX_model = SARIMAX(df, order=best_order).fit()

    # Define the number of steps to forecast into the future
    n_steps = 24  # Number of steps (months) to forecast into the future

    # Generate future dates
    future_dates = pd.date_range(start=df.index[-1] + pd.DateOffset(1), periods=n_steps, freq='M')

    # Get forecast and confidence intervals
    forecast = SARIMAX_model.get_forecast(steps=n_steps)
    forecast_ci = forecast.conf_int()

    # Prepare forecast values and confidence intervals for plotting
    forecast_values = pd.DataFrame(forecast.predicted_mean, index=future_dates, columns=['forecast'])
    forecast_ci.index = future_dates

    # Create a plot for the forecast
    plt.figure(figsize=(15, 6))

    # Plot the actual data
    plt.plot(df, label='Actual')

    # Plot the forecasted data
    plt.plot(forecast.predicted_mean, label='Forecast')

    # Fill the area between confidence intervals
    plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='gray', alpha=0.3, label='Confidence Interval')

    # Set the x-axis and y-axis labels
    plt.xlabel('Date')
    plt.ylabel(f'{col} Change')

    # Set the plot title
    plt.title(f'Future Forecast for {col} : {title}')

    # Show the legend for the plot
    plt.legend()

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

Code Listing 24: Function for Forecasting Unknown Values and Visualising Them

4.6.3. Auto ARIMA

Auto ARIMA is a method that automates the process of finding the ARIMA parameters, AR, I and MA as well as applying seasonal parameters if the data requires it. It is like the process carried out in the model evaluation section. The code can be seen in Code Listing 25 by creating a model using the pm.auto_arima function with the dataset and seasonal set to true. A train and test split are created to a desired date and the model is fit to the training data and predictions are then made and compared to the test data. The RMSE is calculated, and training and test data are plotted as well as the predictions shown in Code Listing 26.

```
# Create an ARIMA model with automatic parameter selection for the 'Population' column in 'combineddf'
# The model is set to handle seasonality with a seasonal period of 12 (e.g., monthly data with yearly seasonality)
# The 'trace=True' argument provides a detailed output of the model's fitting process
model = pm.auto_arima(combineddf['Population'], seasonal=True, m=12, trace=True)

# Split the data into a training set and a test set based on the specified date ranges
# The training set includes data from April 2003 to December 2016
train = combineddf[(combineddf.index >= '2003-04-01') & (combineddf.index < '2017-01-01')]

# The test set includes data from January 2017 onward
test = combineddf[(combineddf.index >= '2017-01-01')]

# Fit the ARIMA model to the training data
model.fit(train['Population'])

# Generate forecasts for the same number of periods as the test set
# 'n_periods=len(test)' specifies the number of periods to predict
# 'return_conf_int=True' returns both the forecasted values and the confidence intervals
forecast = model.predict(n_periods=len(test), return_conf_int=True)

# Create a DataFrame from the forecasted values with the test set's index and a column named 'Population'
forecast_df = pd.DataFrame(forecast[0], index=test.index, columns=['Population'])
```

Code Listing 25: Auto ARIMA Model Fitting and Predicting

```
# Calculate the Root Mean Squared Error (RMSE) between the actual test data and the forecasted data
# RMSE is a commonly used metric to measure the accuracy of predictions in a regression model
rmse = sqrt(mean_squared_error(test['Population'], forecast_df['Population']))
print(f'RMSE: {rmse}')  # Print the RMSE value to evaluate model performance

# Plot the jellyfish population data
plt.plot(train['Population'], label='Training Data')  # Plot the training data
plt.plot(test['Population'], label='Actual Test Data')  # Plot the actual test data
plt.plot(forecast_df['Population'], label='Forecasted Data')  # Plot the forecasted data

# Add labels and title to the plot
plt.xlabel('Date')  # Label the x-axis as 'Date'
plt.ylabel('Jellyfish Population')  # Label the y-axis as 'Jellyfish Population'
plt.title('Jellyfish Population Forecast with SARIMAX')  # Set the title of the plot

# Add a legend to differentiate between the training data, actual test data, and forecasted data
plt.legend()

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

Code Listing 26: Auto ARIMA RMSE and Graphing

Code Listing 27 continues by fitting the model to the whole dataset and making unseen predictions. The predictions are based on the next 48 periods or 2 years. A range of dates is created and the prediction for each date is added to a forecast data frame. Code Listing 28 shows the original data and the unseen predictions plot code.

```
# Fit the ARIMA model to the entire dataset ('Population' column in 'combineddf')
model.fit(combineddf['Population'])

# Generate a forecast for the next 48 periods (months in this case)
# 'n_periods=48' specifies the number of future periods to forecast
# 'return_conf_int=True' returns both the forecasted values and the confidence intervals
forecast = model.predict(n_periods=48, return_conf_int=True)

# Create a date range starting from September 2022 with 48 periods, assuming monthly frequency
# 'MS' (Month Start) is used to generate dates at the start of each month
forecast_range = pd.date_range(start='2022-09-01', periods=48, freq='MS')

# Create a DataFrame from the forecasted values, using the generated date range as the index
# The forecasted values are stored in a column named 'Population'
forecast_df = pd.DataFrame(forecast[0], index=forecast_range, columns=['Population'])
```

Code Listing 27: Auto ARIMA Unseen Model Training and Predictions

```
# Plot the historical (training) jellyfish population data from the combined DataFrame
plt.plot(combineddf['Population'], label='Training Data')

# Plot the forecasted jellyfish population data from the forecast DataFrame
plt.plot(forecast_df['Population'], label='Forecasted Data')

# Add labels to the x-axis (Date) and y-axis (Jellyfish Population)
plt.xlabel('Date')
plt.ylabel('Jellyfish Population')

# Set the title of the plot to describe what is being visualized
plt.title('Jellyfish Population Forecast with SARIMAX')

# Add a legend to differentiate between the historical (training) data and the forecasted data
plt.legend()

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

Code Listing 28: Auto ARIMA Unseen Data Forecasting Graph

4.7. Deep Learning Time Series

Statistical time series is a well-tested method and while there are still changes happening with the technology most of the major advancements are coming from the deep learning side of time series. Deep learning time series can take many forms as discussed in previous chapters and require data to be in the correct tensor format only and not checking its stationarity and autocorrelation.

4.7.1. LSTM

The creation of any deep learning model first involves preparing the data, whether using PyTorch or TensorFlow the data must be in the form of a tensor. Code Listings 29 and 30 show

this code for the jellyfish data as well as the train and test split, scaling and the creation of sequences. The scaler function normalises data by using the MinMax scaler to a range of 0 to 1 before returning the scaled data and the scaler itself. The train test split makes use of slicing to place 80 per cent of the data in the training and 20 in the testing arrays. The prepare sequences function is used to create sequences that can be used for time series. Two blank lists are created, and the data is looped adding a sequence to X and a target to y. These values are converted to a NumPy array and returned. A final function create_sequences makes use of the previous function to create X_train, X_test, y_train and y_test variables. The jellyfish data is then first scaled, split into train and test and finally sequences are created.

```python
def scaler_function(data):
    """
    Scales the input data to a range of 0 to 1 using MinMaxScaler.

    Parameters:
    data (pandas.Series or numpy.ndarray): The data to be scaled.

    Returns:
    tuple: (scaled_data, scaler)
        - scaled_data (numpy.ndarray): The scaled version of the input data.
        - scaler (MinMaxScaler): The scaler object used for scaling.
    """
    scaler = MinMaxScaler(feature_range=(0, 1))  # Initialize MinMaxScaler to scale between 0 and 1
    # Reshape data to be a 2D array with one feature, then fit and transform the data
    scaled_data = scaler.fit_transform(data.values.reshape(-1, 1))
    return scaled_data, scaler

def train_test_split(data, train_size=0.8):
    """
    Splits the data into training and testing sets.

    Parameters:
    data (numpy.ndarray or list): The data to be split.
    train_size (float): Proportion of the data to be used for training (between 0 and 1).

    Returns:
    tuple: (train_data, test_data)
        - train_data (numpy.ndarray or list): The training portion of the data.
        - test_data (numpy.ndarray or list): The testing portion of the data.
    """
    train_size = int(len(data) * train_size)  # Determine the split index based on train_size
    train_data = data[:train_size]  # Slice the data to create training data
    test_data = data[train_size:]  # Slice the data to create testing data
    return train_data, test_data

def prepare_sequences(data, seq_length):
    """
    Prepares sequences of data for time series prediction.

    Parameters:
    data (numpy.ndarray): The input data to be split into sequences.
    seq_length (int): The length of each sequence.

    Returns:
    tuple: (X, y)
        - X (numpy.ndarray): Sequences of input data.
        - y (numpy.ndarray): Target values corresponding to each sequence.
    """
    X = []  # Initialize list to hold sequences of features
    y = []  # Initialize list to hold target values
    # Loop through the data to create sequences
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])  # Append the sequence to X
        y.append(data[i+seq_length])    # Append the target value to y
    return np.array(X), np.array(y)  # Convert lists to numpy arrays and return
```

Code Listing 29: LSTM Data Preparation

```
    def create_sequences(train_data, test_data, seq_length=12):
        """
        Creates sequences from training and testing data for time series forecasting.

        Parameters:
        train_data (numpy.ndarray): The training data to be used for creating sequences.
        test_data (numpy.ndarray): The testing data to be used for creating sequences.
        seq_length (int): The length of each sequence.

        Returns:
        tuple: (X_train, y_train, X_test, y_test)
            - X_train (numpy.ndarray): Sequences of training data.
            - y_train (numpy.ndarray): Target values for training data.
            - X_test (numpy.ndarray): Sequences of testing data.
            - y_test (numpy.ndarray): Target values for testing data.
        """
        X_train, y_train = prepare_sequences(train_data, seq_length)  # Create sequences for training data
        X_test, y_test = prepare_sequences(test_data, seq_length)    # Create sequences for testing data
        return X_train, y_train, X_test, y_test  # Return the sequences and target values

[32] jellyfish_scaled, jellyfish_scaler = scaler_function(jellyfishcountdf)
    train_jellyfish, test_jellyfish = train_test_split(jellyfish_scaled)
    X_train_jellyfish, y_train_jellyfish, X_test_jellyfish, y_test_jellyfish = create_sequences(train_jellyfish, test_jellyfish)
```

Code Listing 30: LSTM Data Preparation Continued

After the data had been prepared a series of functions were created for modelling, graphing and predicting. Code Listings 31 and 32 show the model graphing function, it has a flag variable that determines if it is plotting the test data or forecasting unseen data. Labels, titles and the legend are applied before showing the graph. The future_prediction function predicts the next number of periods. The model.predict function takes the last sequence and makes a prediction on the next, this value is added to a list and the last sequence variable is updated before another prediction is made. The predictions are scaled in line with the previous scaling and transformed into a series with corresponding dates and the model_graphing function is called. A fit_model is the final function; it fits the training data to the model and specifies the batch size and epochs. The fitted model is then used to make predictions, these values are used to calculate the MSE and RMSE. The other two functions are called to display graphs for testing and unseen variables.

```python
def model_graphing(actual_values, predictions, title, df, flag=True):
    """
    Plots the actual values and predictions for the model.

    Parameters:
    actual_values (numpy.ndarray or pandas.Series): The actual values to plot.
    predictions (numpy.ndarray or pandas.Series): The predicted values to plot.
    title (str): The title of the plot.
    df (pandas.DataFrame): The original data used to create the plot (for the 'Predicted' series).
    flag (bool): Determines whether to plot 'Actual' vs. 'Predicted' or vice versa.
    """
    plt.figure(figsize=(15, 6))  # Set the figure size for the plot

    if flag:
        # Plot actual values and predictions on the same graph
        plt.plot(actual_values, label='Actual')
        plt.plot(predictions, label='Predicted')
    else:
        # Plot predictions and actual values in reverse order
        plt.plot(df, label='Predicted')
        plt.plot(actual_values, predictions, label='Actual')

    plt.xlabel('Date')  # Label for x-axis
    plt.ylabel(f'{title}')  # Label for y-axis with title
    plt.title(f'RNN: {title} Predictions')  # Title of the plot
    plt.legend()  # Show legend
    plt.show()  # Display the plot

def future_prediction(model, scaler, scaled_data, seq_length, df, title):
    """
    Predicts future values based on the trained model and plots these predictions.

    Parameters:
    model (Keras Model): The trained model used for prediction.
    scaler (MinMaxScaler): The scaler object used to inverse transform predictions.
    scaled_data (numpy.ndarray): The scaled data used for creating predictions.
    seq_length (int): The length of the input sequences.
    df (pandas.DataFrame): The original data, used to create future dates for plotting.
    title (str): The title of the plot.
    """
    future_seq_length = 24  # Number of time steps to predict into the future (24 months)
    last_sequence = scaled_data[-seq_length:]  # Get the last sequence from the dataset

    future_predictions = []
    for _ in range(future_seq_length):
        # Predict the next value using the model
        next_pred = model.predict(last_sequence.reshape(1, seq_length, 1))

        # Append the prediction to the list
        future_predictions.append(next_pred[0, 0])

        # Update the last sequence for the next prediction
        last_sequence = np.roll(last_sequence, -1)  # Roll array elements
        last_sequence[-1] = next_pred[0, 0]  # Update the last value with the prediction

    # Inverse transform predictions to the original scale
    future_predictions = scaler.inverse_transform(np.array(future_predictions).reshape(-1, 1))

    # Create future dates for plotting
    future_dates = pd.date_range(start=df.index[-1] + pd.DateOffset(1), periods=future_seq_length, freq='M')

    # Plot future predictions
    model_graphing(future_dates, future_predictions, title+' Future', df, False)
```

Code Listing 31: LSTM Model Training, Predicting and Visualising Functions

```
def fit_model(model, X_train, y_train, X_test, y_test, title, df, scaler, scaled_data, seq_length, epochs=100, batch_size=32):
    """
    Fits the model to the training data, evaluates it, and plots the results.

    Parameters:
    model (Keras Model): The model to be trained.
    X_train (numpy.ndarray): Training features.
    y_train (numpy.ndarray): Training target values.
    X_test (numpy.ndarray): Testing features.
    y_test (numpy.ndarray): Testing target values.
    title (str): The title for the plot.
    df (pandas.DataFrame): The original data used for plotting.
    scaler (MinMaxScaler): The scaler object used to inverse transform predictions.
    scaled_data (numpy.ndarray): The scaled data used for creating sequences.
    seq_length (int): The length of the input sequences.
    epochs (int): Number of epochs to train the model.
    batch_size (int): Batch size for training.
    """
    # Fit the model on the training data
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test))

    # Predict on the test data
    predictions = model.predict(X_test)
    predictions = scaler.inverse_transform(predictions)  # Convert predictions back to original scale
    actual_values = scaler.inverse_transform(y_test)  # Convert actual values back to original scale

    # Print metrics for performance evaluation
    for i in range(len(predictions)):
        print(f"Actual: {actual_values[i][0]}, Predicted: {predictions[i][0]}")

    print(f'MSE: {mean_squared_error(actual_values, predictions)}')
    print(f'RMSE: {sqrt(mean_squared_error(actual_values, predictions))}')
    print(f'MAE: {mean_absolute_error(actual_values, predictions)}')

    # Plot actual vs. predicted values
    model_graphing(actual_values, predictions, title, df)

    # Predict and plot future values
    future_prediction(model, scaler, scaled_data, seq_length, df, title)
```

Code Listing 32: LSTM Model Training, Predicting and Visualising Function Continued

Using a TensorFlow model a basic LSTM/RNN model was created that used a Sequential layer, LSTM layer and Dense Layer. The Adam optimiser was selected and the MSE loss function, the model summary was output, and it called upon the fit_model function shown in Code Listing 33.

```
def RNNmodel(X_train, y_train, X_test, y_test, title, scaler, scaled_data, df, seq_length=12):
  #Constructs, compiles, and trains an RNN model using LSTM layers, and then fits the model using the provided data.
  # Initialize the Sequential model
    model = Sequential()

    # Add an LSTM layer with 64 units, ReLU activation, and input shape based on training data
    model.add(LSTM(64, activation='relu', input_shape=(X_train.shape[1], 1)))

    # Add a Dense layer to produce the output (one unit for regression)
    model.add(Dense(1))

    # Compile the model with Adam optimizer and Mean Squared Error loss function
    model.compile(optimizer='adam', loss='mse')

    # Print the model summary to show the architecture and parameters
    print(model.summary())

    # Train the model using the provided training and testing data
    fit_model(model, X_train, y_train, X_test, y_test, title, df, scaler, scaled_data, seq_length)

RNNmodel(X_train_jellyfish, y_train_jellyfish, X_test_jellyfish, y_test_jellyfish, 'Jellyfish Count', jellyfish_scaler, jellyfish_scaled, jellyfishcountdf)
```

Code Listing 33: TensorFlow LSTM Model

A second variation of an LSTM model was constructed to work as a multivariate model considering the climate and jellyfish climate dataset to conclude. This LSTM also used the PyTorch framework instead of TensorFlow. The data was transformed in the same way except it had to be converted into PyTorch tensor using the torch.tensor function. Code Listing 34 shows the code used for the LSTM class. First, the training dataset was converted to a tensor dataset and placed inside a data loader. The class was then created, called the LSTM superclass, and then set the hidden layer and number of layers values. An LSTM layer and a fully connected layer were then created. A forward pass function initialised the hidden and cell state to a matrix of zeros and passed the matrices and the data through an LSTM and then through the fully connected layer before returning the output. The model parameters were set, and a model was created.

```
[ ]  batch_size = 32
     dataset = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
     train_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
     class LSTMModel(nn.Module):
         def __init__(self, input_size, hidden_size, num_layers, output_size):
             super(LSTMModel, self).__init__()
             self.hidden_size = hidden_size
             self.num_layers = num_layers

             # LSTM layer
             self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)

             # Fully connected layer
             self.fc = nn.Linear(hidden_size, output_size)

         def forward(self, x):
             # Initialize hidden state and cell state
             h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
             c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

             # Forward propagate LSTM
             out, _ = self.lstm(x, (h0, c0))

             # Decode the hidden state of the last time step
             out = self.fc(out[:, -1, :])
             return out

     input_size = X_train_tensor.shape[2]
     hidden_size = 50
     num_layers = 2
     output_size = 1

     model = LSTMModel(input_size, hidden_size, num_layers, output_size)
```

Code Listing 34: PyTorch LSTM Model

Code Listing 35 continues the model code and creates a training loop. First, the optimiser and loss function of Adam and MSE are selected, and the number of epochs is set. A loop is run for the number of epochs that trains the model. The model makes a prediction on the output based on the sequence and a forward pass calculates the loss. Based on the loss, the optimiser updates the weights using the backward function and the optimiser step is also updated. Every tenth epoch then displays the current loss. After the model has been trained, it is evaluated using model.eval, then setting the model to not update using the torch.nograd, a series of predictions are made on the test tensor. The test loss is acquired, and the predictions and test values are transformed into the legible original information using the inverse_transformer function. The MSE and RMSE are calculated and a graph of the actual data vs the predicted is output using matplotlib.

```python
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    for i, (seq, labels) in enumerate(train_loader):
        # Forward pass
        outputs = model(seq)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [10/100], Loss: 0.0085
Epoch [20/100], Loss: 0.0170
Epoch [30/100], Loss: 0.0059
Epoch [40/100], Loss: 0.0562
Epoch [50/100], Loss: 0.0290
Epoch [60/100], Loss: 0.0045
Epoch [70/100], Loss: 0.0028
Epoch [80/100], Loss: 0.0042
Epoch [90/100], Loss: 0.0060
Epoch [100/100], Loss: 0.0080
```

```python
model.eval()
with torch.no_grad():
    y_pred = model(X_test_tensor)
    test_loss = criterion(y_pred, y_test_tensor)
    print(f'Test Loss: {test_loss.item():.4f}')

y_pred_inverse = scaler.inverse_transform(y_pred.numpy())
y_test_inverse = scaler.inverse_transform(y_test_tensor.numpy())

#for i in range(len(y_pred_inverse)):
    #print(f"Actual: {y_test_inverse[i][0]}, Predicted: {y_pred_inverse[i][0]}")
```

```
Test Loss: 0.0232
```

```python
print(f'MSE: {mean_squared_error(y_test_inverse, y_pred_inverse)}')
print(f'RMSE: {sqrt(mean_squared_error(y_test_inverse, y_pred_inverse))}')
plt.figure(figsize=(12, 6))
plt.plot(y_test_inverse, label='Actual')
plt.plot(y_pred_inverse, label='Predicted')
plt.legend()
plt.title('Jellyfish Population: Actual vs Predicted')
plt.show()
```

Code Listing 35: Multivariate LSTM Model Training, Testing and Output

Finally, the model is updated to include all the data instead of a train test split. A new model is trained on this data in the same way as before but instead of calculating the test predictions a series of new predictions are made, see Code Listing 36. First, the value of how far to predict is calculated and the last sequence is retrieved from the end. A loop then predicts the next sequence and reshapes the last sequence to the model's input shape. The new prediction is appended to a list and the last sequence is updated. The predictions are transformed so they are no longer scaled and are added to a data frame with their corresponding date. This information is then plotted using matplotlib.

```
# Set the number of future time steps to predict
future_seq_length = 48

# Extract the last sequence of input data that the model was trained on for forecasting
last_sequence = scaled_features[-seq_length:]

# Initialize a list to store the future predictions
future_predictions = []

# Loop to generate predictions for the specified number of future time steps
for _ in range(future_seq_length):
    # Use the model to predict the next value, reshaping the last sequence to match the model's input shape
    next_pred = model(torch.tensor(last_sequence.reshape(1, seq_length, scaled_features.shape[1]), dtype=torch.float32)).detach().numpy()

    # Append the prediction to the list of future predictions
    future_predictions.append(next_pred[0, 0])

    # Update the last sequence by rolling the sequence forward and replacing the last value with the prediction
    last_sequence = np.roll(last_sequence, -1, axis=0)
    last_sequence[-1] = next_pred[0, 0]

# Inverse transform the predicted values to return them to their original scale
future_predictions = scaler.inverse_transform(np.array(future_predictions).reshape(-1, 1))

# Generate a date range for the future predictions, starting from the day after the last date in the historical data
future_dates = pd.date_range(start=combineddf.index[-1] + pd.DateOffset(1), periods=future_seq_length, freq='M')

# Plot the historical and forecasted data
plt.figure(figsize=(12, 6))
plt.plot(combineddf['Population'], label='Historical Data')
plt.plot(future_dates, future_predictions, label='Forecasted')
plt.xlabel('Date')
plt.ylabel('Jellyfish Population')
plt.title('Jellyfish Population Forecast with LSTM')
plt.legend()
plt.show()
```

Code Listing 36: PyTorch LSTM Unseen Future Forecast/ Visualisations

4.7.2. Transformer

Like the LSTM, the setup for the transformer requires many of the same steps in terms of preparing the data mainly creating the sequences. Code Listing 37 shows the model for the transformer itself using PyTorch as the base, the superclass for TransformerModel is first called followed by a linear layer which becomes an embedding layer which maps the input to match with the sequence. A transformer encoder layer is defined next where the number of heads and feed-forward dimensions are set. Finally, a fully connected linear layer is declared. The forward pass then calls the embedding layer, permutes the tensor to fit into the

transformer layer and passes it through the transformer layer and fully connected layer. The output tensor is returned after extracting the last element of its middle dimension and removing the last element if it's a single dimension.

```python
# Define a Transformer model for time series forecasting
class TransformerModel(nn.Module):  # Inherits from nn.Module, which is a base class in PyTorch with useful methods and attributes
    def __init__(self, input_dim, seq_length, num_layers, num_heads, dim_feedforward, output_dim):
        '''
        input_dim = Number of features in the input data (e.g., 1 if only using the closing price)
        seq_length = Length of the input sequence (e.g., 5 if using 5 days to predict the 6th day)
        num_layers = Number of layers in the Transformer encoder
        num_heads = Number of heads in the multi-head attention mechanism
        dim_feedforward = Dimension of the feedforward network in the transformer. Each feedforward network of a transformer layer will transform input data into a
        output_dim = Number of features in the output (e.g., 1 if only predicting the next closing price)
        '''
        super(TransformerModel, self).__init__()  # Calling the constructor (def __init__) of nn.Module

        # Use a linear layer as an embedding layer. Maps input features (input_dim) to the sequence length (seq_length)
        self.embedding = nn.Linear(input_dim, seq_length)
        '''
        In NLP, an embedding layer transforms discrete tokens (like words) into continuous vectors.
        In time series data, an embedding layer transforms features (e.g., dates, categorical data) into a continuous vector representation.
        '''

        # Define a single transformer encoder layer
        transformer_layer = nn.TransformerEncoderLayer(
            d_model=seq_length,  # The size of each input sample (the "model dimension")
            nhead=num_heads,  # Number of heads in the multi-head attention mechanism
            dim_feedforward=dim_feedforward  # Dimension of the feedforward network within the transformer
        )
        '''
        A single transformer layer includes:
        - A multi-head self-attention mechanism
        - A feedforward neural network
        '''

        # Stack multiple transformer layers to create the transformer encoder
        self.transformer = nn.TransformerEncoder(transformer_layer, num_layers=num_layers)
        '''
        Transformer encoder stacks multiple transformer layers to form a deep model.
        '''

        # Define the output layer that transforms the transformer's output to the desired output dimensions
        self.fc_out = nn.Linear(seq_length, output_dim)
        '''
        - The output layer transforms the output of the transformer encoder (i.e., stacked transformer layers) to the final output with the desired dimensions.
        - This layer gives the final output, which is the actual prediction of the model.
        '''

    def forward(self, src):
        '''
        Forward method processes the input (src) and returns the output.

        This function is called implicitly when the model is used in training or prediction.

        Example usage in training:
        - y_pred = model(x_batch) internally calls y_pred = model.forward(x_batch)

        Example usage in prediction:
        - predictions = model(input_data) internally calls predictions = model.forward(input_data)
        '''
        # Pass the input through the embedding layer
        src = self.embedding(src)

        # Reshape input tensor to fit the requirements of the transformer encoder
        src = src.permute(1, 0, 2)  # Transformer expects input in the format (sequence length, batch size, features)

        # Pass the reshaped input through the transformer encoder
        output = self.transformer(src)

        # Reshape the output back to (batch size, sequence length, features)
        output = output.permute(1, 0, 2)

        # Pass the output through the final fully connected layer to get predictions
        output = self.fc_out(output)

        # Return the last time step's prediction (for each sequence in the batch)
        return output[:, -1, :].squeeze(-1)
```

Code Listing 37: PyTorch Transformer Model Code

Code Listing 38 shows a training function and a prediction function. The training function shows the steps taken to train the model; it is mostly like the LSTM. The epochs are looped, and the loss is calculated through the forward pass steps. The loss is then used to update the

weights in the backward pass step. Every tenth epoch's loss is output. The predict function disconnects from the model so no weights are updated, and a series of predictions are made for the input data.

```python
# Function to train the model
def train(model, train_loader, optimizer, criterion, epochs):
    model.train()  # Set the model to training mode
    for epoch in range(epochs):  # Loop over the number of epochs
        for x_batch, y_batch in train_loader:  # Loop over each batch in the DataLoader
            optimizer.zero_grad()  # Clear the gradients of all optimized variables
            y_pred = model(x_batch)  # Forward pass: compute the model's prediction for the batch
            loss = criterion(y_pred, y_batch)  # Calculate the loss between the predictions and actual targets
            loss.backward()  # Backward pass: compute the gradient of the loss with respect to model parameters
            optimizer.step()  # Perform a single optimization step (parameter update)

        # Print the loss at the first and last epoch
        if epoch == 0:
            print(f'Epoch {epoch+1}, Loss: {loss.item()}')  # Print loss for the first epoch
        elif epoch == epochs - 1:
            print(f'Epoch {epoch+1}, Loss: {loss.item()}')  # Print loss for the last epoch

# Function to make predictions with the model
def predict(model, input_data):
    model.eval()  # Set the model to evaluation mode
    with torch.no_grad():  # Disable gradient calculation, since we're only making predictions
        prediction = model(input_data)  # Forward pass: get the model's prediction for the input data
    return prediction  # Return the prediction
```

Code Listing 38: Training and Predicting Function for Transformer Model

The Transformer is interacted with through the loop function which sets several variables such as batch size and input dimensions. The data is transformed into sequences through the create_sequences function and converted into a DataLoader variable. The model is then initialised, and after setting the optimiser and loss function is passed to the training function. The trained model is then evaluated by gathering its predictions and transforming them back to unscaled data using the inverse_transform function. The MSE and RMSE are outputs, and a graph of the trained model is also included. See Code Listing 39 and the cited GitHub Repo and Article where the code Transformer code was adapted from (Chen 2024; Longhui 2024) .

```
# Main function to train and predict with the transformer model
def loop(seq_length=4, batch_size=16, input_dim=1, num_layers=2,
         num_heads=2, dim_feedforward=10, output_dim=1, lr=0.001,
         epochs=50):  # (Note: seq_length should be a multiple of num_heads)

    ################################ Train Model
    set_seed(0)  # Set seed for reproducibility

    # Create sequences for training
    X, y = create_sequences(data_tensor, seq_length)
    train_data = TensorDataset(X, y)  # Create a dataset from the sequences
    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=False)  # Create a DataLoader

    # Initialize the Transformer model
    model = TransformerModel(input_dim, seq_length, num_layers, num_heads, dim_feedforward, output_dim)

    criterion = nn.MSELoss()  # Mean Squared Error loss
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)  # Adam optimizer with specified learning rate

    # Train the model
    train(model, train_loader, optimizer, criterion, epochs=epochs)

    ################################ Predict Training Data
    X, y = train_X, train_y  # Use training data for prediction
    model.eval()  # Set the model to evaluation mode

    all_predictions = []
    all_actuals = []

    with torch.no_grad():  # Disable gradient calculation for prediction
        for i in range(len(X)):  # Loop over all training examples
            single_prediction = predict(model, X[i].unsqueeze(0))  # Predict for each example
            predicted_value = single_prediction.squeeze().numpy()[-1]  # Extract the last predicted value
            all_predictions.append(predicted_value)
            all_actuals.append(y[i].item())  # Store the actual value

    # Convert predictions and actuals to the original scale using the scaler
    all_predictions = scaler.inverse_transform(np.array(all_predictions).reshape(-1, 1))
    all_actuals = scaler.inverse_transform(np.array(all_actuals).reshape(-1, 1))

    # Calculate and print Mean Squared Error (MSE) and Mean Absolute Error (MAE)
    mse = mean_squared_error(all_actuals, all_predictions)
    mae = mean_absolute_error(all_actuals, all_predictions)

    print(f'Mean Squared Error: {mse}')
    print(f'Mean Absolute Error: {mae}')

    # Plot the actual vs predicted values for the training data
    plt.figure(figsize=(12, 6))
    plt.plot(all_actuals, label='Actual')
    plt.plot(all_predictions, label='Predicted')
    plt.title('Jellyfish Population Change (Training Data)')
    plt.xlabel('Time (Months)')
    plt.ylabel('Jellyfish Population')
    plt.legend()
    plt.show()
```

```
60] seq_length = 4
   data = combineddf['Population'].values.reshape(-1, 1)
   scaler = MinMaxScaler(feature_range=(0, 1))
   data_scaled = scaler.fit_transform(data)
   data_tensor = torch.tensor(data_scaled, dtype=torch.float32)
   train_X, train_y = create_sequences(data_tensor, seq_length)
```

```
61] loop()
```

Code Listing 39: Main Program for Transformer Model

4.9. Conclusion

This chapter outlined the completion of the artefact that accompanies this dissertation. Throughout the development, not everything was planned and not every planned feature was implemented either due to data limitations or technical problems but overall, the code in the

Jupyter Notebook present adequately provides the solution to the research questions posed in Chapter 1. The next chapter will detail the results found from the metrics and graphs attained from the work in this chapter and make comparisons for the different models and methods used to predict jellyfish populations outlining the pros and cons of each.

Chapter 5: Results and Evaluations

5.1. Introduction

Results are the foundation of a data science project as they give insight into the problem that is being solved by helping identify trends, outliers and much more. In a time series paper, there are lots of different results to explore and analyse, firstly the preliminary explorations showing counts of jellyfish species and averages are explored, the steps undertaken in time series exploration like differencing and autocorrelation are examined next and finally, the statistical and deep learning models are examined in terms of their forecasting accuracy and model evaluation where they can be used to forecast unknown values. The results discovered will be explored alongside insights gathered from the literature review to come to conclusions about jellyfish and time series analysis.

5.2. Exploration Results

5.2.1. Basic Explorations

The basic explorations performed in this project are standard procedures across many data science projects and are not limited to time series but can still prove useful especially when examining a complex problem such as jellyfish population dynamics.

With the jellyfish dataset, the data was largely categorical as a result, it benefited from making counts of each occurrence and mapping this to a visualisation like a pie chart. Figure 8 shows one such pie chart showing the spread of jellyfish species around the coast of the UK and Ireland. Based on previous observations this aligns with the species that are native to Irish and British waters but there are the Mauve Stinger and the By-the-wind-sailor which are rare to spot around these waters. There are also pie charts about the species, genus and family present in the Jupyter Notebook.

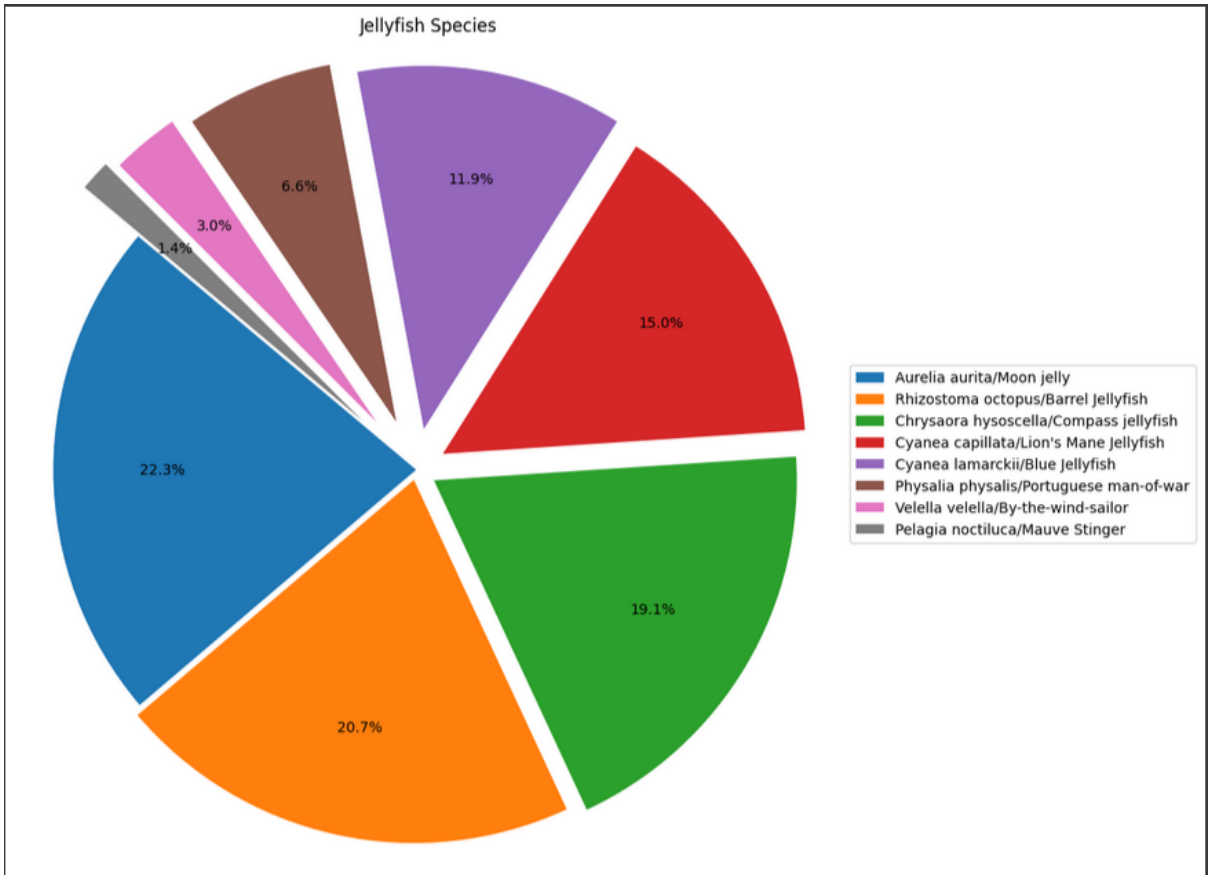


Figure 8: Jellyfish Species Pie Chart

Figure 9 shows a pie chart for the general location or country where the jellyfish sighting occurred. As expected, the spread is mostly focused on England due to its larger coastline compared to the other regions, but it is also more southernly, allowing for warmer waters potentially sustaining better conditions than Scotland.

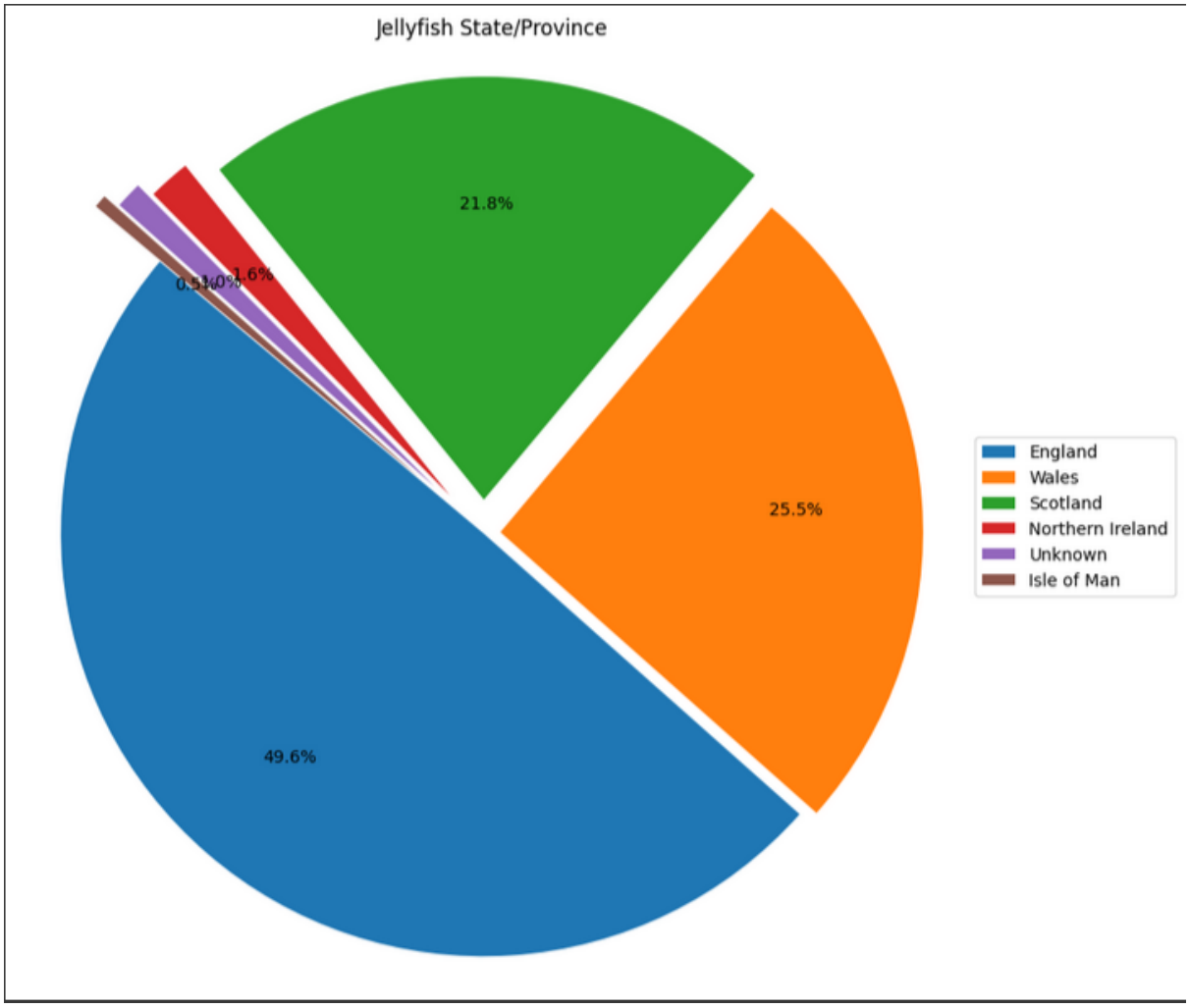


Figure 9: Jellyfish State/Province Pie Chart

Figure 10 shows a pie chart based on the occurrences each month. The summer months of June, July and August result in the most sightings with over three-quarters of sightings occurring in this period. This is likely due to two reasons; the warmer waters allow for more jellyfish blooms meaning there are more jellyfish compared to winter months where they remain almost dormant, but there is also likely the human element. The surveys where these are carried out take place during the summer and during the winter the conditions to check waters are not there resulting in fewer sightings during those months. It is however clearly apparent that the summer months result in more sightings and prove there is a seasonality to the data.

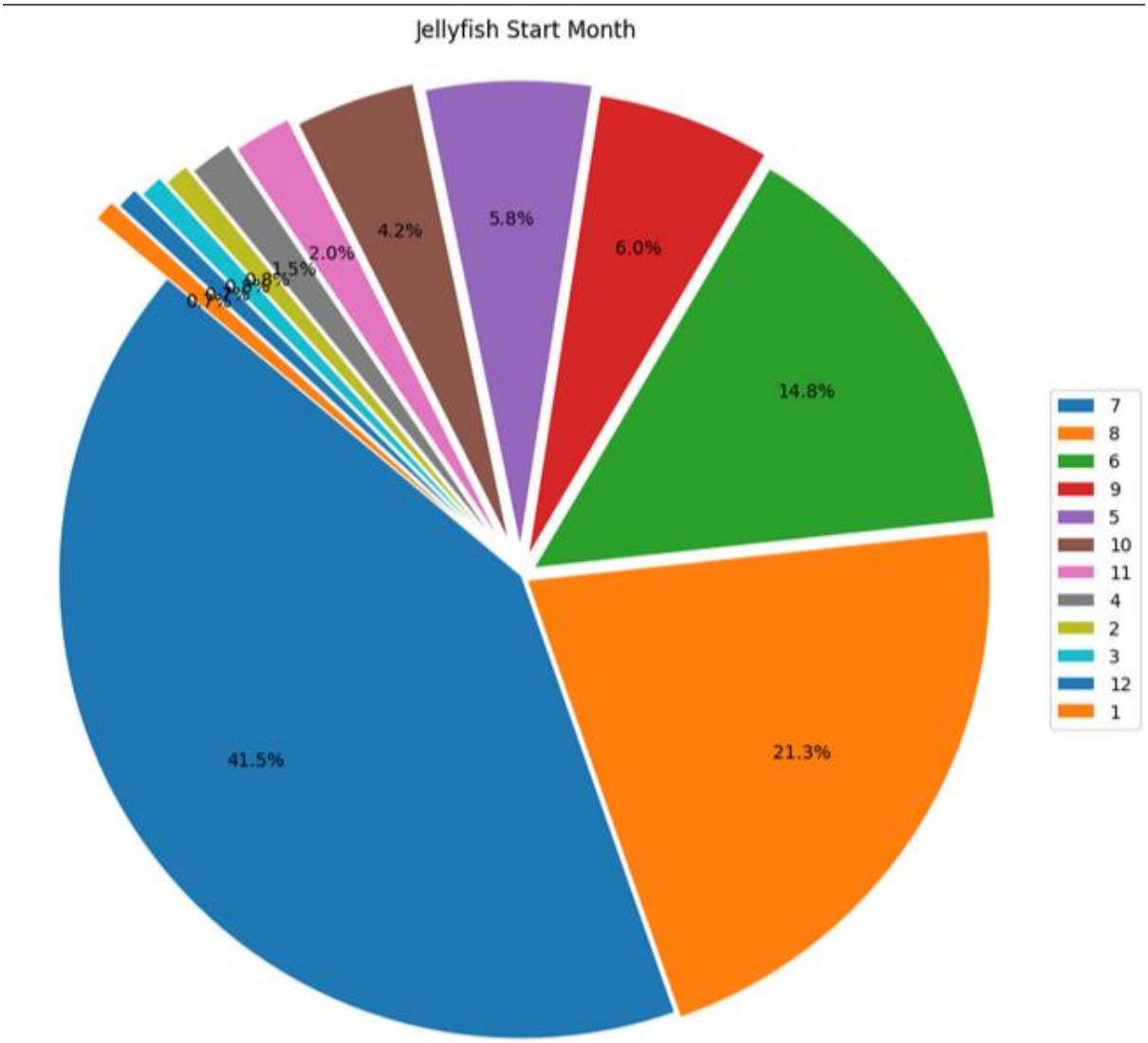


Figure 10: Monthly Occurrences Pie Chart

Figure 11 shows the occurrences based on year. There is not even a split between years suggesting there are fluctuations in populations each year. 2013 to 2017 feature some of the highest occurrences of jellyfish, while 2012 and 2018 feature some of the lowest. It may reinforce the idea that jellyfish populations follow some form of oscillation cycle of having larger populations and smaller populations every few years. But there is not enough data here to conclude proof of that. Some interesting points are that 2021 is quite high but 2020 is lower than expected which could be a result of the COVID-19 pandemic but there is no information to conclude on this either. There is also the human element to consider where 2013 to 2017 may have had more surveys than other years resulting in higher counts but cannot make any conclusion in this regard.

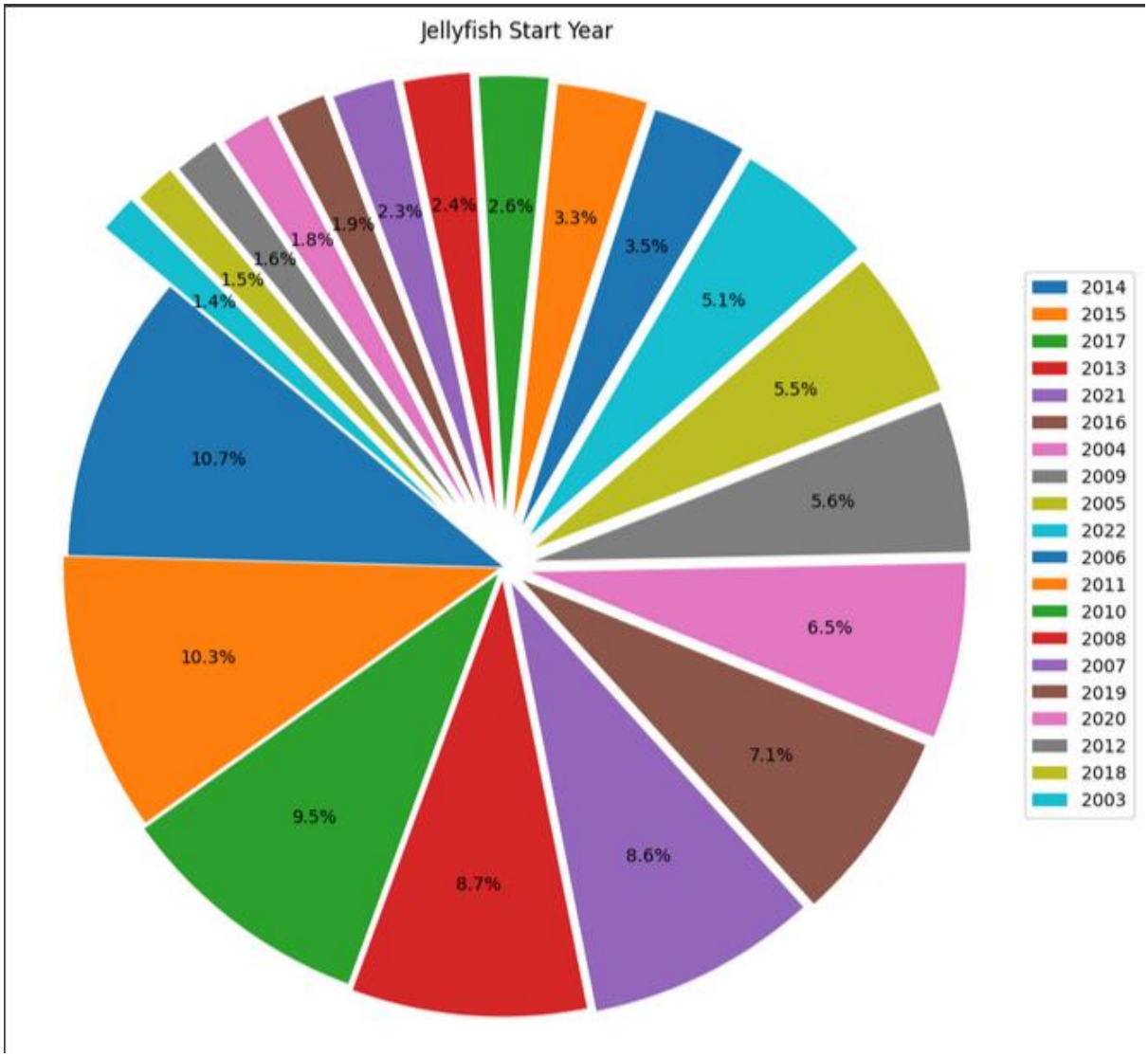


Figure 11: Jellyfish Year Sighting Pie Chart

Figure 12 shows a map created from the Folium library. It has marked the location of each sighting record highlighting the colour based on species. Based on the map there is a possibility that a few of the latitudes were entered incorrectly as the sightings are in Europe and not the UK or Ireland but it does show that there is a good balance of sightings around the islands. Figure 13 shows a zoomed-in version when highlighting a specific point that reveals the species of jellyfish in that location. There are some more graphs of the exploration shown in Appendix B.

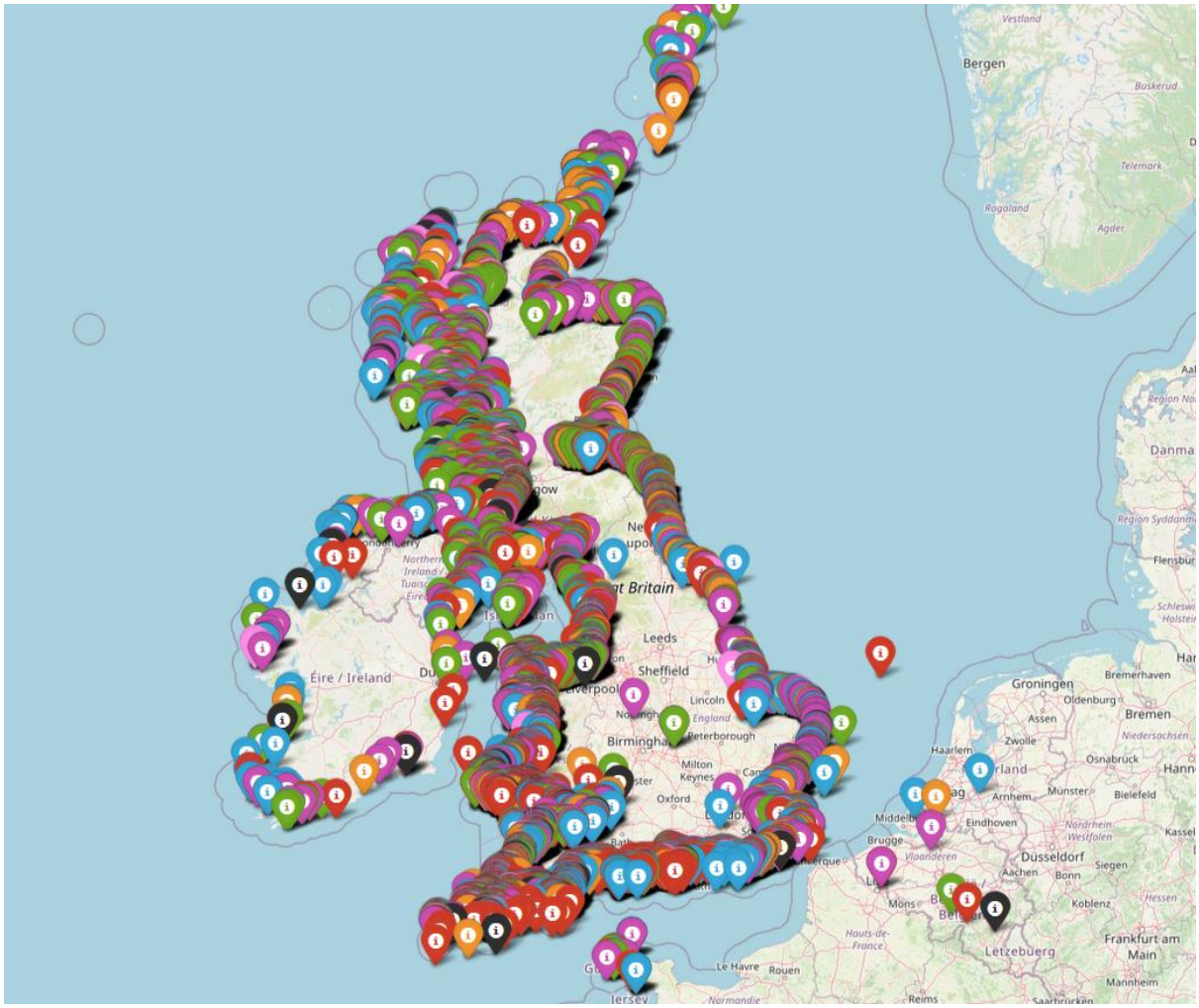


Figure 12: Jellyfish Locations around UK and Ireland

Figure 13: Jellyfish Species on map

5.2.2. Stationarity Results

For the jellyfish data, the stationarity test returned conflicting answers shown in Figure 14. The ADF rejects the null hypothesis that the data is stationary due to the p-value of 0.082453 being greater than 0.05 but according to the KPSS test is stationary due to its p-value of 0.1

where we fail to reject the null hypothesis. It is not uncommon for this to occur in time series because the test analyses different parts of the data to conclude. As a result, it is likely better to differentiate the data to find values where the data agrees on its stationarity. The climate data all agreed it was already stationary except for rainfall where KPSS claimed it was not stationary. Sea levels both ADF and KPSS were not stationary.

	adf	kpss
Test Statistic	-2.653413	0.169995
p-value	0.082452	0.1
Numbers of lags	12	4
decision	Non-Stationary	Stationary
Critical Value (1%)	-3.460428	0.739
Critical Value (5%)	-2.874769	0.463
Critical Value (10%)	-2.573821	0.347
Critical Value (2.5%)	NaN	0.574

Figure 14: Jellyfish dataset Stationarity Test

5.2.3. Differencing Results

As the jellyfish data could not agree on its stationarity, differencing the data is important to perform time series analysis on the data. Figure 15 shows a graph of each of the transformations applied. After applying the first differencing to the data, the stationarity is agreed by both tests and would suggest that the data is now suitable to perform time series on. The other transformations reveal similar information such as removing seasonality, rolling mean, cyclic extract and de-trending the data all produce stationary results. Performing the second difference and square root transformations doesn't make the data stationary according to their tests. These could be used as a comparison later.

The climate data, differencing can be seen in Appendix B, they are now all stationary even though most already were. The sea level data while it could be transformed, the data could not be graphed because the dates are not correctly formatted.

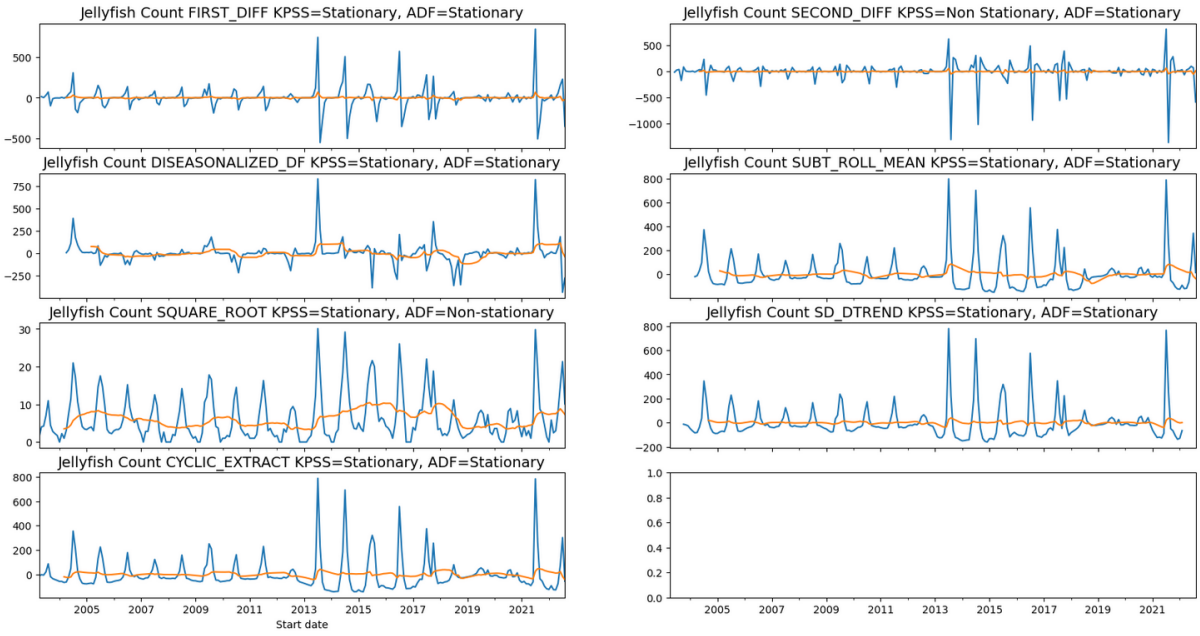


Figure 15: Differencing jellyfish data, stationarity test

5.2.4. Regression Results

Figure 16 shows the Ordinary Least Squares linear regression result for the first difference jellyfish count. The model summary provides a lot of information, but the most important parts are the R squared of 0.77 which would suggest that the model is a good fit. The coefficients are almost all statistically significant as their p-values are less than 0.05. Finally, some of the variables like kurtosis and skew show non-normality, aligning with previous tests meaning that the data may not be ideal for modelling.

Jellyfish Count

OLS Regression Results

Dep. Variable:	y	R-squared:	0.770
Model:	OLS	Adj. R-squared:	0.757
Method:	Least Squares	F-statistic:	57.88
Date:	Wed, 21 Aug 2024	Prob (F-statistic):	2.28e-59
Time:	11:03:24	Log-Likelihood:	-1325.2
No. Observations:	220	AIC:	2676.
Df Residuals:	207	BIC:	2721.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-6.4631	0.748	-8.636	0.000	-7.939	-4.988
x2	4.9418	0.700	7.060	0.000	3.562	6.322
x3	4.3360	0.642	6.749	0.000	3.069	5.603
x4	3.8080	0.581	6.558	0.000	2.663	4.953
x5	3.2509	0.519	6.264	0.000	2.228	4.274
x6	2.7085	0.455	5.953	0.000	1.811	3.605
x7	2.1759	0.390	5.584	0.000	1.408	2.944
x8	1.6548	0.324	5.109	0.000	1.016	2.293
x9	1.1464	0.258	4.437	0.000	0.637	1.656
x10	0.6133	0.194	3.154	0.002	0.230	0.997
x11	0.1104	0.129	0.854	0.394	-0.145	0.366
x12	-0.1790	0.071	-2.534	0.012	-0.318	-0.040
const	1.8140	6.954	0.261	0.794	-11.896	15.525

Omnibus:	268.737	Durbin-Watson:	1.980
Prob(Omnibus):	0.000	Jarque-Bera (JB):	13521.512
Skew:	5.178	Prob(JB):	0.00
Kurtosis:	39.984	Cond. No.	269.

Figure 16: Jellyfish Regression Results

Figure 17 shows the climate data Max Temp regression results as an example. The other results are shown in Appendix B. This model manages to achieve an R squared of 0.79, marginally better than the jellyfish data. Its coefficients are also mostly significant predictors, only x14 and x15 have a higher p-value than 0.05. The values for skew, kurtosis and omnibus are less than jellyfish data while still not ideal suggesting it is better suited to a model.

MaxTemp

OLS Regression Results

Dep. Variable:	y	R-squared:	0.790
Model:	OLS	Adj. R-squared:	0.786
Method:	Least Squares	F-statistic:	195.9
Date:	Wed, 21 Aug 2024	Prob (F-statistic):	0.00
Time:	11:03:24	Log-Likelihood:	-2300.0
No. Observations:	1331	AIC:	4652.
Df Residuals:	1305	BIC:	4787.
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.2721	0.071	-3.808	0.000	-0.412	-0.132
x2	-0.3994	0.074	-5.434	0.000	-0.544	-0.255
x3	-0.3546	0.073	-4.859	0.000	-0.498	-0.211
x4	-0.3332	0.072	-4.635	0.000	-0.474	-0.192
x5	-0.3900	0.070	-5.534	0.000	-0.528	-0.252
x6	-0.4361	0.069	-6.304	0.000	-0.572	-0.300
x7	-0.4446	0.069	-6.471	0.000	-0.579	-0.310
x8	-0.4371	0.068	-6.405	0.000	-0.571	-0.303
x9	-0.4492	0.068	-6.631	0.000	-0.582	-0.316
x10	-0.4189	0.067	-6.214	0.000	-0.551	-0.287
x11	-0.3455	0.067	-5.167	0.000	-0.477	-0.214
x12	-0.2529	0.067	-3.799	0.000	-0.383	-0.122
x13	-0.1807	0.066	-2.735	0.006	-0.310	-0.051
x14	-0.0651	0.065	-1.004	0.316	-0.192	0.062
x15	-0.1125	0.063	-1.785	0.075	-0.236	0.011
x16	-0.2123	0.060	-3.510	0.000	-0.331	-0.094
x17	-0.2044	0.057	-3.561	0.000	-0.317	-0.092
x18	-0.2239	0.054	-4.158	0.000	-0.330	-0.118
x19	-0.2307	0.050	-4.585	0.000	-0.329	-0.132
x20	-0.2250	0.046	-4.847	0.000	-0.316	-0.134
x21	-0.3247	0.042	-7.668	0.000	-0.408	-0.242
x22	-0.3208	0.039	-8.212	0.000	-0.397	-0.244
x23	-0.2830	0.036	-7.809	0.000	-0.354	-0.212
x24	-0.2227	0.033	-6.812	0.000	-0.287	-0.159
x25	-0.1055	0.027	-3.840	0.000	-0.159	-0.052
const	3.3160	0.869	3.816	0.000	1.611	5.021

Omnibus:	44.891	Durbin-Watson:	1.999
Prob(Omnibus):	0.000	Jarque-Bera (JB):	76.920
Skew:	-0.269	Prob(JB):	1.98e-17
Kurtosis:	4.048	Cond. No.	325.

Figure 17: Max Temp Regression Results

5.2.5. Autocorrelation Results

Figure 18 shows the autocorrelation results for the base jellyfish dataset. In the ACF plot, it shows a significant spike at lags 1 and 2 which would suggest a correlation between the current count and the previous two periods. Still, as the lags progress this dampens showing a decrease in correlation with past values. In the PACF plot, there is a spike at the first lag

suggesting a correlation between the current and previous value. Still, after that, the values are within the confidence bands, explaining the correlation sufficiently. This suggests the values for AR and MA are 2 and 1 respectively.

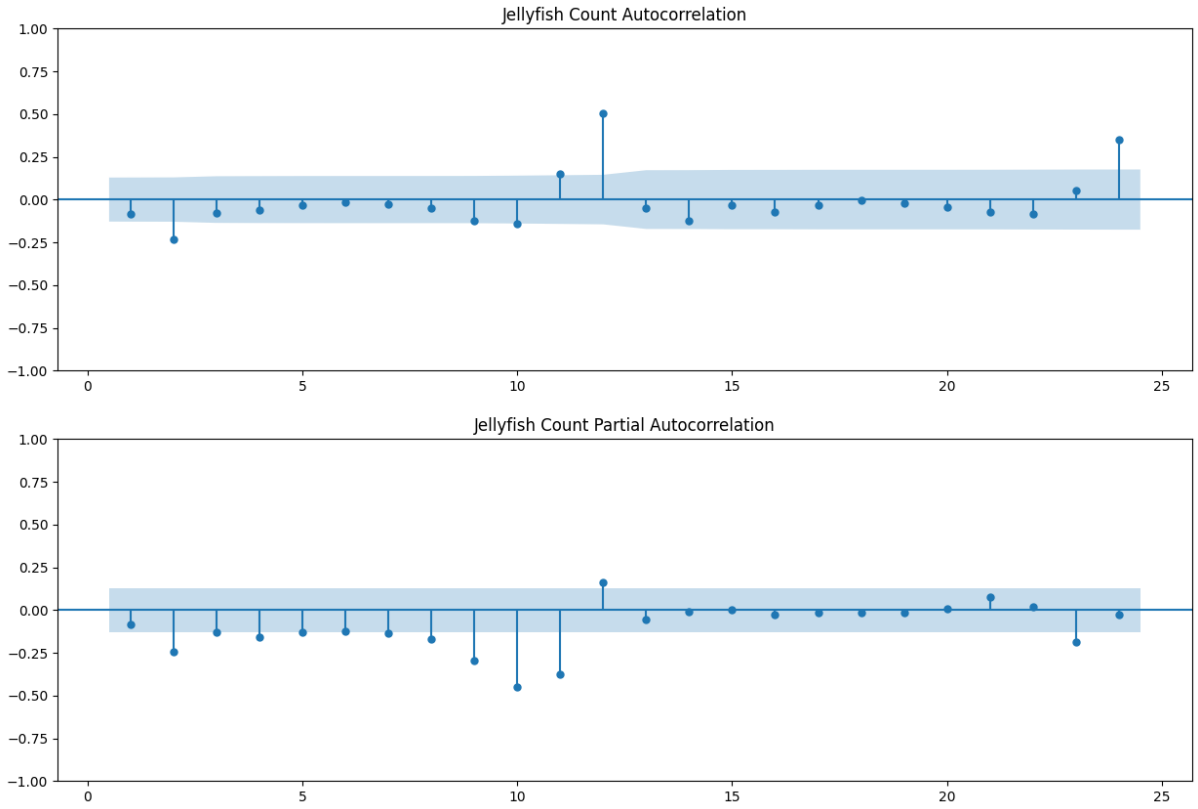


Figure 18: Jellyfish Base ACF/PACF plot

Figure 19 shows the first difference for the jellyfish data ACF and PACF plots. The ACF has less significant spikes outside the confidence interval implying it has removed an aspect of autocorrelation from the data. The PACF still suggests a relationship with the current and previous value, but this eventually cuts off. Suggests the values for AR and MA are 1 and 0 respectively. Appendix B features the ACF and PACF for the climate data. These graphs vary but usually have significant spikes that dampen or cut off to explain a decrease in correlation or sufficiently capture the correlation relationship.

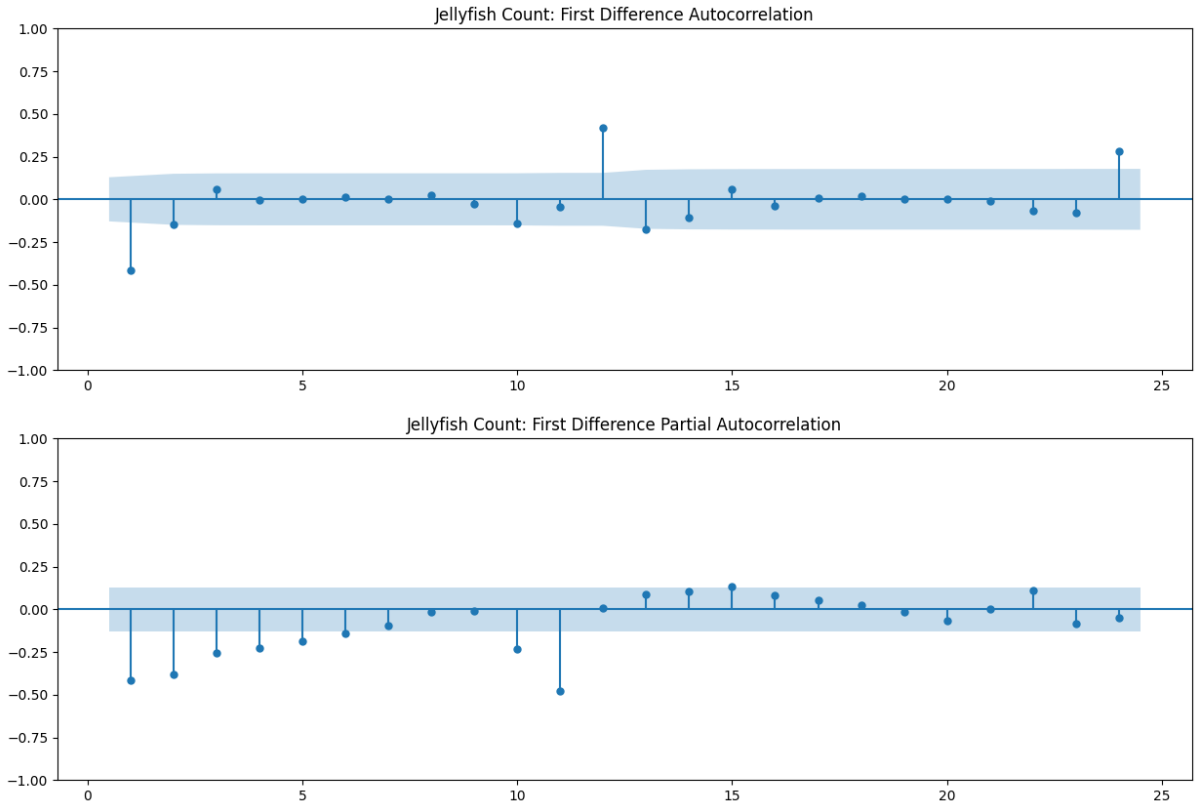


Figure 19: Jellyfish First Difference ACF/PACF

5.3. Statistical Time Series Results

5.3.1. Model Evaluations

Using the functions created, the best order for an ARIMA-based model was found for each dataset, in Figure 20, the first differenced jellyfish dataset results are shown for the combinations of 0 to 3, 0 to 1 and 0 to 2. Looping the orders returns the best order for the dataset to be 3, 0, 2 which has a very high cross-validation error of 21,709. The lower this value is the better and this is the best value that could be acquired for the data using these values. There are several reasons such as overfitting, underfitting or incorrect splitting but, likely, there is just not enough data available and it's too irregular for it to capture a massive pattern. As this is the best value, its RMSE is tested, and it returns an average RMSE of 137.7 which again is poor as it should ideally be a value less than 1 but also expected due to the nature of the data.

```
jellyfish_best_order = best_order = best_order_function(jellyfishcountdf.diff().dropna().values)

Order (0, 0, 0) - CV Error: 22146.65866174703
Order (0, 0, 1) - CV Error: 22147.61541996657
Order (0, 0, 2) - CV Error: 22163.37944758425
Order (0, 1, 0) - CV Error: 22424.22631578947
Order (0, 1, 1) - CV Error: 22146.66259484316
Order (0, 1, 2) - CV Error: 22147.688074551832
Order (1, 0, 0) - CV Error: 22147.0670430088
Order (1, 0, 1) - CV Error: 22097.210774527743
Order (1, 0, 2) - CV Error: 22076.747498620207
Order (1, 1, 0) - CV Error: 22390.309714457697
Order (1, 1, 1) - CV Error: 22147.070130045784
Order (1, 1, 2) - CV Error: 22149.66966263674
Order (2, 0, 0) - CV Error: 22145.867439592454
Order (2, 0, 1) - CV Error: 22046.407043911662
Order (2, 0, 2) - CV Error: 21772.37578419944
Order (2, 1, 0) - CV Error: 22760.067060581223
Order (2, 1, 1) - CV Error: 22145.834815844508
Order (2, 1, 2) - CV Error: 22103.646928701255
Order (3, 0, 0) - CV Error: 22141.477035198266
Order (3, 0, 1) - CV Error: 22017.478333241346
Order (3, 0, 2) - CV Error: 21709.811331107263
Order (3, 1, 0) - CV Error: 22771.77026982261
Order (3, 1, 1) - CV Error: 22140.751691863952
Order (3, 1, 2) - CV Error: 21798.198419287586
Best Order: (3, 0, 2) with CV Error: 21709.811331107263

best_order_rmse(jellyfish_best_order, jellyfishcountdf)

RMSE: [81.65931620244085, 63.83917493799558, 231.25545376824806, 143.1216096763351, 168.69840663081166]
Mean RMSE: 137.71479224316624
```

Figure 20: Best order for Jellyfish data and its RMSE

While the value may not be ideal, a graphical representation of the model can still be achieved as it may lead to satisfactory results or reveal information as to why it didn't work as expected.

Figure 21 shows the first graphical representation of this, A plot_diagnostics plot, graphs four graphs four plots. A standardised residual plot in the top left of Figure X shows a healthy level of fluctuation which shows it does not have a bias but there are spikes especially two large spikes around 2014 and 2021. 2014 the data shows a massive boost in jellyfish populations which would have been hard to predict based on previous data available and 2021 could easily struggle with less data collected during the COVID-19 pandemic skewing results. The second graph is the histogram with Kernel Estimated Density (KDE) in the top right, this graph shows that the residuals of the data are located around the zero good mark, but it skewed a lot more in favour of the left side, the KDE plot line suggests the same as it almost normally distributed but there is an imbalance to the left suggesting potential outliers in the data.

The normal Q-Q plot in the bottom left suggests there is a deviation from the normality of the data as an ideal graph would be a straight line, and based on the tails of the data, it verifies

that there are outliers in the data present. The final plot in the bottom right, is another autocorrelation plot, it reveals that most of the data falls inside the confidence interval except the first value at lag 1 which shows there is an autocorrelation in the data. Overall, these graphs showcase that the model may be successful but there are still problems with spikes, autocorrelation and normal distribution.

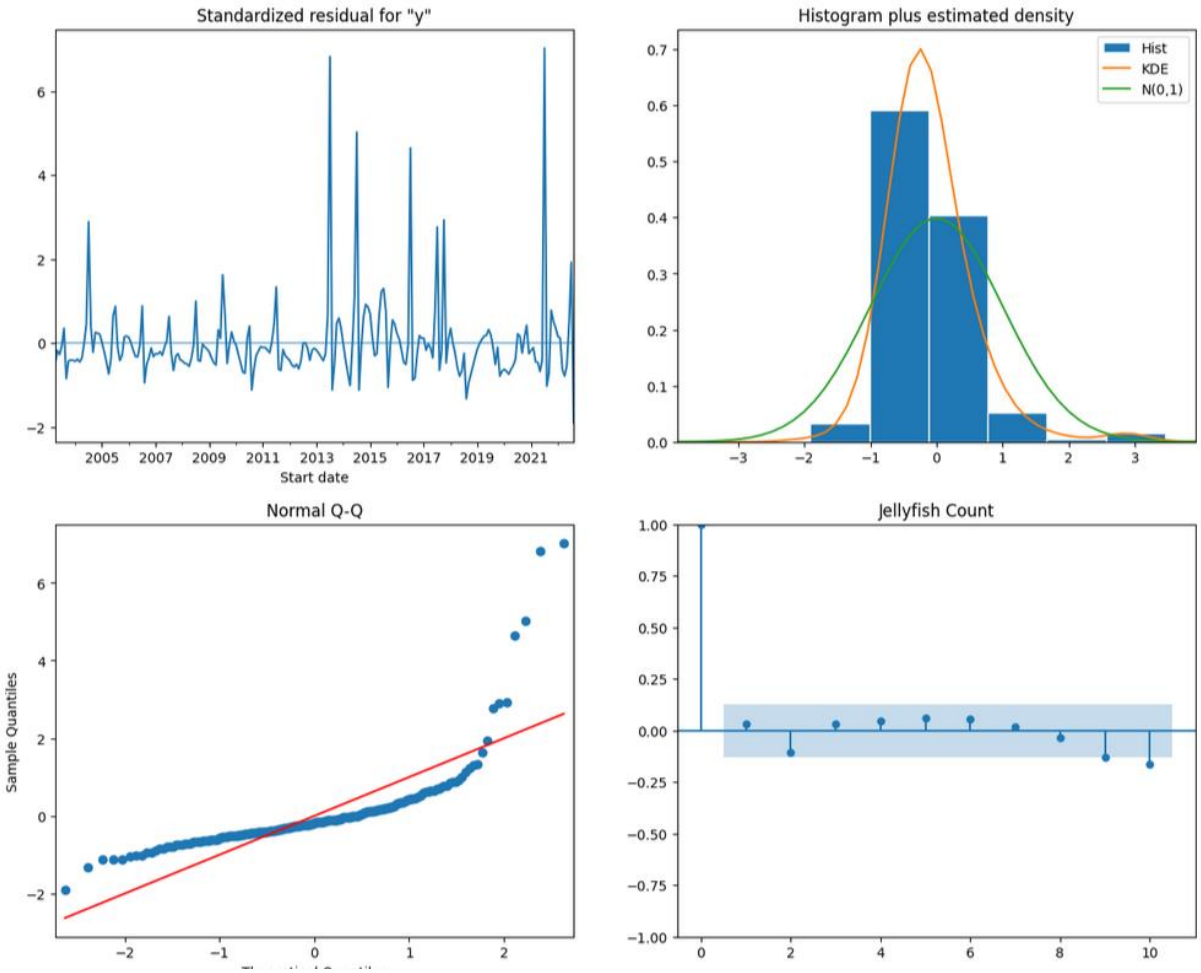


Figure 21: Residuals, Histogram KDE, QQ and autocorrelation plot for Jellyfish first differenced data

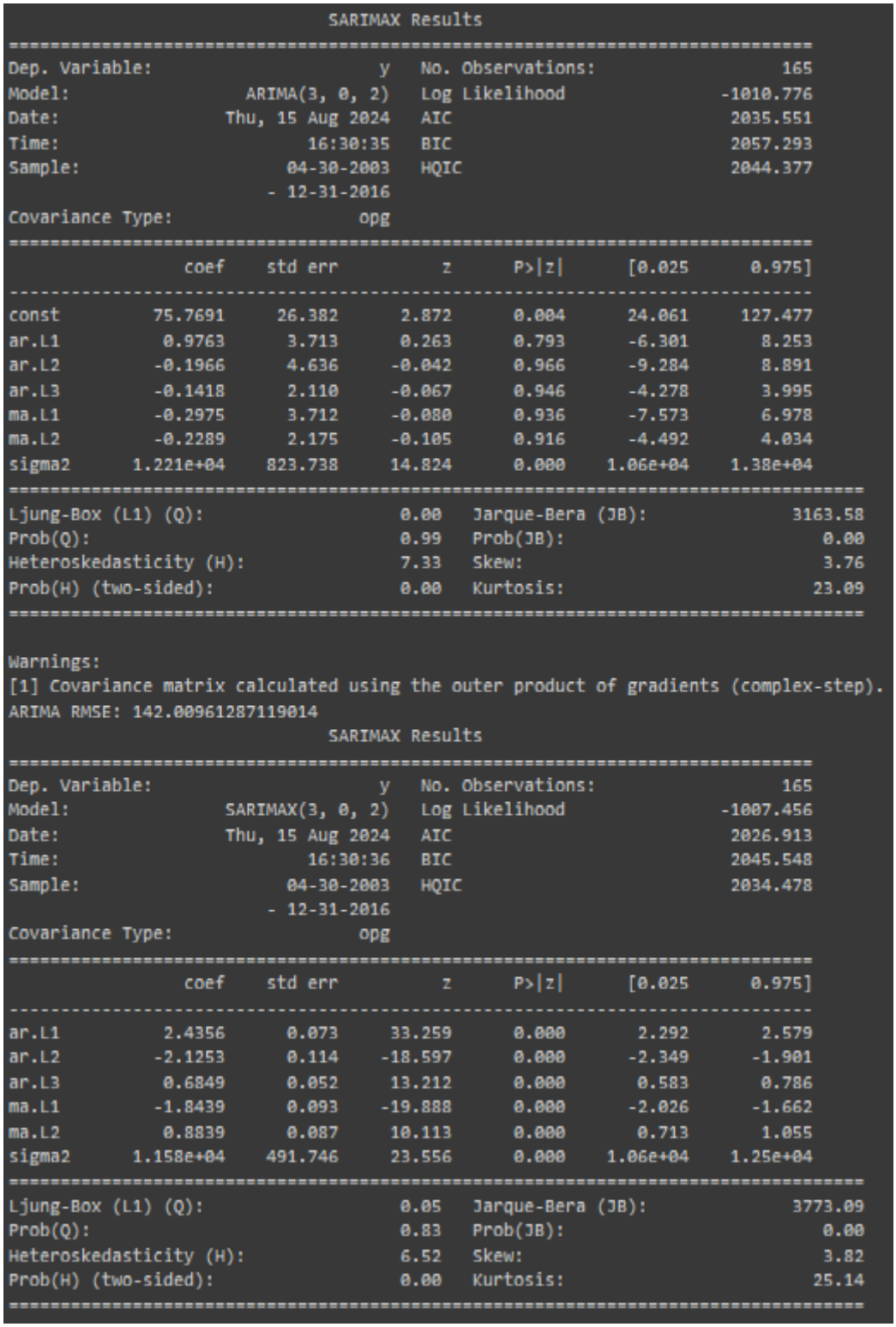
The best order and diagnostics plot was also found for the climate data and sea level data which can be seen in the Appendix. Overall, these datasets are more regular, and they resulted in better results with MaxTemp having a value of a CV error of 4.66 and an RMSE of 1.47. The plot_diagnostics function showed a good fluctuation of residuals, a very slightly skewed normal distribution, a solid Q-Q plot with a slight tail suggesting outliers and a bit of

autocorrelation, especially in the first lag as well. The other columns followed a similar pattern.

5.3.2. Model Visualisation

Exploring the potential models in more detail showed that while it has problems, it is likely better for forecasting than the RMSE would suggest. A series of functions shown in Chapter 4 were created for creating a model and visualising both ARIMA and SARIMAX, the results are shown below in Figure 22 for the base Jellyfish data. Using the summary function on the model produces the below output for ARIMA on top and SARIMAX on the bottom. Between the two, the SARIMAX model outperforms the ARIMA model in multiple categories. The log-likelihood of the SARIMAX is less negative, and the Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), and Hannan-Quinn-Information Criterion (HQIC) are all lower for SARIMAX which is preferable. The SARIMAX model shows more compatibility with AR, I and MA values as the p-value is 0 showing they are significant, unlike the ARIMA counterpart. Both however have high p values for Ljung-Box, Jarque-Bera and Heteroskedasticity but this is explained by using the base data instead of differenced data.

The resultant graph produced a result of this data is shown in Figure 23. It is obvious that both ARIMA and SARIMAX models failed to capture any underlying pattern in the data, but this can be explained by the base data being used which many previous steps showcased to not be the case.



```
                            SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  165
Model:                 ARIMA(3, 0, 2)   Log Likelihood               -1010.776
Date:                Thu, 15 Aug 2024   AIC                           2035.551
Time:                        16:30:35   BIC                           2057.293
Sample:                    04-30-2003   HQIC                          2044.377
                         - 12-31-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         75.7691     26.382      2.872      0.004      24.061     127.477
ar.L1          0.9763      3.713      0.263      0.793      -6.301       8.253
ar.L2         -0.1966      4.636     -0.042      0.966      -9.284       8.891
ar.L3         -0.1418      2.110     -0.067      0.946      -4.278       3.995
ma.L1         -0.2975      3.712     -0.080      0.936      -7.573       6.978
ma.L2         -0.2289      2.175     -0.105      0.916      -4.492       4.034
sigma2      1.221e+04    823.738     14.824      0.000    1.06e+04    1.38e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              3163.58
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               7.33   Skew:                             3.76
Prob(H) (two-sided):                  0.00   Kurtosis:                        23.09
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
ARIMA RMSE: 142.00961287119014
                            SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  165
Model:               SARIMAX(3, 0, 2)   Log Likelihood               -1007.456
Date:                Thu, 15 Aug 2024   AIC                           2026.913
Time:                        16:30:36   BIC                           2045.548
Sample:                    04-30-2003   HQIC                          2034.478
                         - 12-31-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          2.4356      0.073     33.259      0.000       2.292       2.579
ar.L2         -2.1253      0.114    -18.597      0.000      -2.349      -1.901
ar.L3          0.6849      0.052     13.212      0.000       0.583       0.786
ma.L1         -1.8439      0.093    -19.888      0.000      -2.026      -1.662
ma.L2          0.8839      0.087     10.113      0.000       0.713       1.055
sigma2      1.158e+04    491.746     23.556      0.000    1.06e+04    1.25e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):              3773.09
Prob(Q):                              0.83   Prob(JB):                         0.00
Heteroskedasticity (H):               6.52   Skew:                             3.82
Prob(H) (two-sided):                  0.00   Kurtosis:                        25.14
===================================================================================
```

Figure 22: ARIMA and SARIMAX model summary for Base Jellyfish Dataset

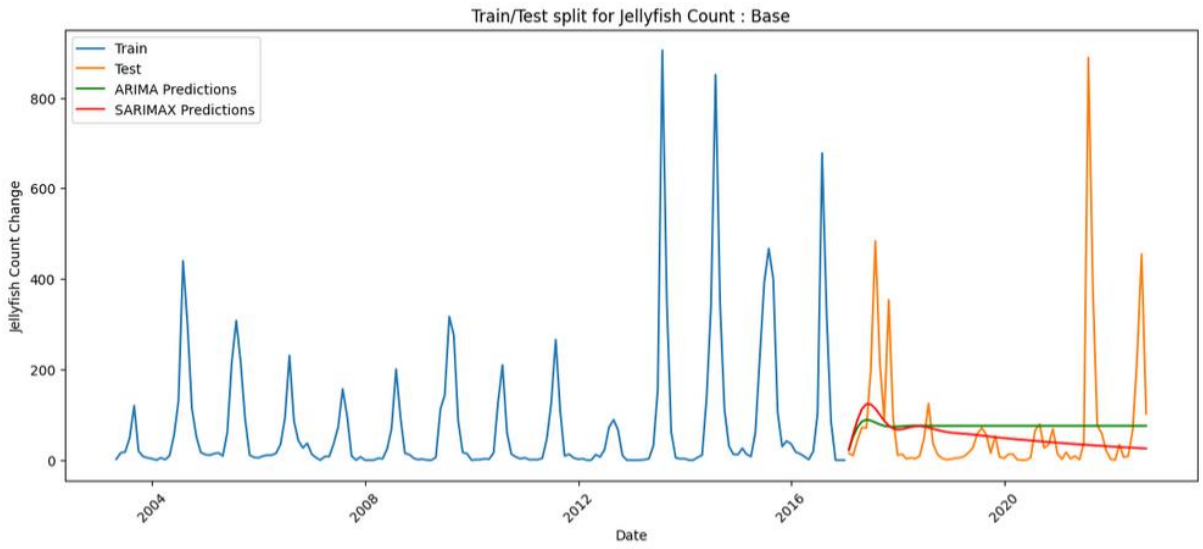


Figure 23: Base Jellyfish Time Series ARIMA and SARIMAX visualisation

Changing the representation from base to an order of differencing produces alternate results. In Figure 24, the first differencing model is shown. The ARIMA and SARIMAX values capture the same information in this instance. The values for log-likelihood, BIC, AIC, and HQIC are approximately the same as the base. Some of the values perform worse, such as the Ljung-Box test and heteroskedasticity, but only minimally. It becomes evident why when examining the graphical representation of the model shown in Figure 25, the first differencing only captures a pattern of seasonality from the data but fails to extract any information on the outliers. Again, likely a consequence of the data itself and its unpredictability with its outliers. The RMSE is higher than the base too at 155 but the different models capture different things so is up to higher interpretation.

```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                  164
Model:                 ARIMA(3, 0, 2)   Log Likelihood               -1006.265
Date:                Thu, 15 Aug 2024   AIC                           2026.530
Time:                        16:30:37   BIC                           2048.229
Sample:                    05-31-2003   HQIC                          2035.339
                         - 12-31-2016
Covariance Type:                  opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const          0.5170      0.580      0.891      0.373      -0.620       1.654
ar.L1         -0.2618      0.068     -3.840      0.000      -0.395      -0.128
ar.L2          0.4367      0.109      4.001      0.000       0.223       0.651
ar.L3         -0.3010      0.139     -2.173      0.030      -0.573      -0.029
ma.L1         -0.0039      0.239     -0.016      0.987      -0.472       0.464
ma.L2         -0.9961      0.074    -13.475      0.000      -1.141      -0.851
sigma2      1.208e+04   2.27e-05   5.33e+08      0.000    1.21e+04    1.21e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):              2800.66
Prob(Q):                              0.79   Prob(JB):                         0.00
Heteroskedasticity (H):               7.10   Skew:                             3.54
Prob(H) (two-sided):                  0.00   Kurtosis:                        21.96
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 5.25e+25. Standard errors may be unstabl
ARIMA RMSE: 155.167554640327
                               SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                  164
Model:               SARIMAX(3, 0, 2)   Log Likelihood               -1007.172
Date:                Thu, 15 Aug 2024   AIC                           2026.344
Time:                        16:30:38   BIC                           2044.943
Sample:                    05-31-2003   HQIC                          2033.894
                         - 12-31-2016
Covariance Type:                  opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
ar.L1         -0.2534      0.080     -3.184      0.001      -0.409      -0.097
ar.L2          0.4393      0.118      3.735      0.000       0.209       0.670
ar.L3         -0.3021      0.133     -2.271      0.023      -0.563      -0.041
ma.L1          0.0061      0.092      0.066      0.947      -0.174       0.187
ma.L2         -0.9643      0.090    -10.744      0.000      -1.140      -0.788
sigma2      1.239e+04    963.496     12.858      0.000    1.05e+04    1.43e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.09   Jarque-Bera (JB):              2873.71
Prob(Q):                              0.76   Prob(JB):                         0.00
Heteroskedasticity (H):               7.30   Skew:                             3.56
Prob(H) (two-sided):                  0.00   Kurtosis:                        22.23
===================================================================================
```

Figure 24: Jellyfish First Difference ARIMA/SARIMAX model summary

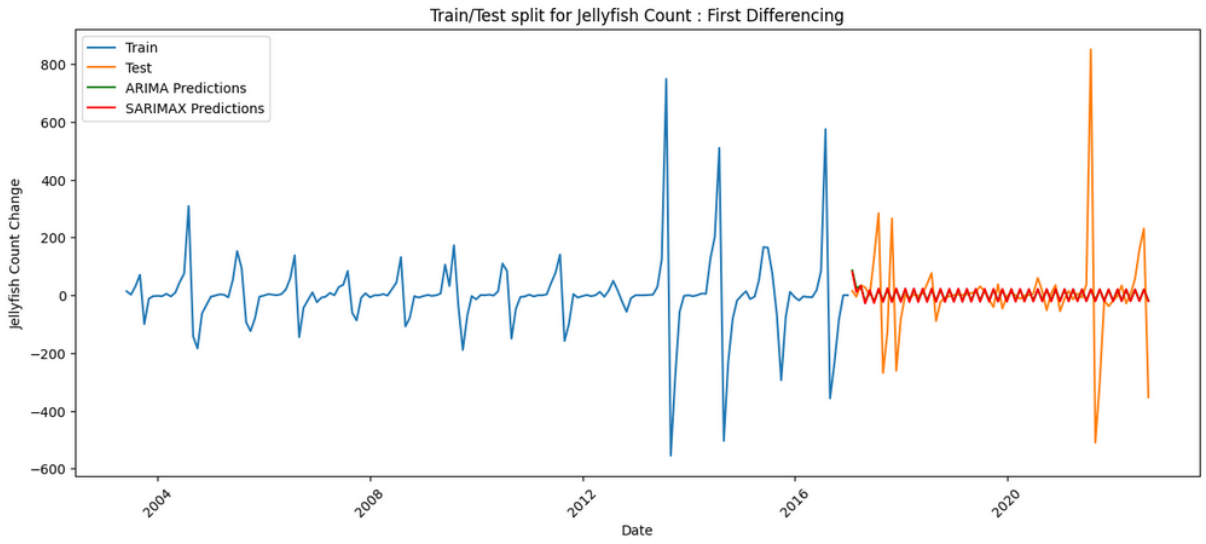


Figure 25: Jellyfish First Differencing ARIMA/SARIMAX plot

The second difference and seasonal differencing exhibit very similar information to the first differencing but while the second differencing features the same seasonality, the seasonal differencing model fails to capture any pattern in the model and is a flat line along zero, as seen in the Appendix B. Seasonal Decompose is the best-performing model so far, both its ARIMA and SARIMAX are almost identical and feature an RMSE of 128 and the lowest BIC, AIC, HQIC and log-likelihood shown in Figure 26. In Figure 27 the seasonal decompose graph is shown, while it eventually settles into the same seasonal pattern as the first and second difference, it starts with a distinct spike at the start which lines up with the data. It suggests that with more data, this method may be able to predict more but again as the data has unpredictable outliers this cannot be guaranteed.

```
                           SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  159
Model:                 ARIMA(3, 0, 2)   Log Likelihood                -945.738
Date:                Thu, 15 Aug 2024   AIC                           1905.477
Time:                        16:30:41   BIC                           1926.959
Sample:                  10-31-2003   HQIC                          1914.201
                       - 12-31-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0882      0.570      0.155      0.877      -1.028       1.205
ar.L1          0.3466      0.050      6.885      0.000       0.248       0.445
ar.L2          0.6427      0.051     12.526      0.000       0.542       0.743
ar.L3         -0.7036      0.045    -15.660      0.000      -0.792      -0.616
ma.L1         -0.0026      0.335     -0.008      0.994      -0.660       0.655
ma.L2         -0.9974      0.086    -11.535      0.000      -1.167      -0.828
sigma2      8231.7679   4.37e-05   1.88e+08      0.000    8231.768    8231.768
===================================================================================
Ljung-Box (L1) (Q):                   0.29   Jarque-Bera (JB):              1594.35
Prob(Q):                              0.59   Prob(JB):                         0.00
Heteroskedasticity (H):               7.86   Skew:                             2.79
Prob(H) (two-sided):                  0.00   Kurtosis:                        17.48
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 7.11e+23. Standard errors may be unstable.
ARIMA RMSE: 128.90831642236904
                           SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  159
Model:               SARIMAX(3, 0, 2)   Log Likelihood                -945.759
Date:                Thu, 15 Aug 2024   AIC                           1903.517
Time:                        16:30:42   BIC                           1921.931
Sample:                  10-31-2003   HQIC                          1910.995
                       - 12-31-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3470      0.051      6.789      0.000       0.247       0.447
ar.L2          0.6428      0.046     13.902      0.000       0.552       0.733
ar.L3         -0.7035      0.039    -17.827      0.000      -0.781      -0.626
ma.L1         -0.0035      0.260     -0.014      0.989      -0.514       0.507
ma.L2         -0.9964      0.086    -11.554      0.000      -1.165      -0.827
sigma2      8239.6983   3.55e-05   2.32e+08      0.000    8239.698    8239.698
===================================================================================
Ljung-Box (L1) (Q):                   0.29   Jarque-Bera (JB):              1614.80
Prob(Q):                              0.59   Prob(JB):                         0.00
Heteroskedasticity (H):               7.85   Skew:                             2.81
Prob(H) (two-sided):                  0.00   Kurtosis:                        17.57
===================================================================================
```

Figure 26: Jellyfish Seasonal Decompose ARIMA/SARIMAX model summary

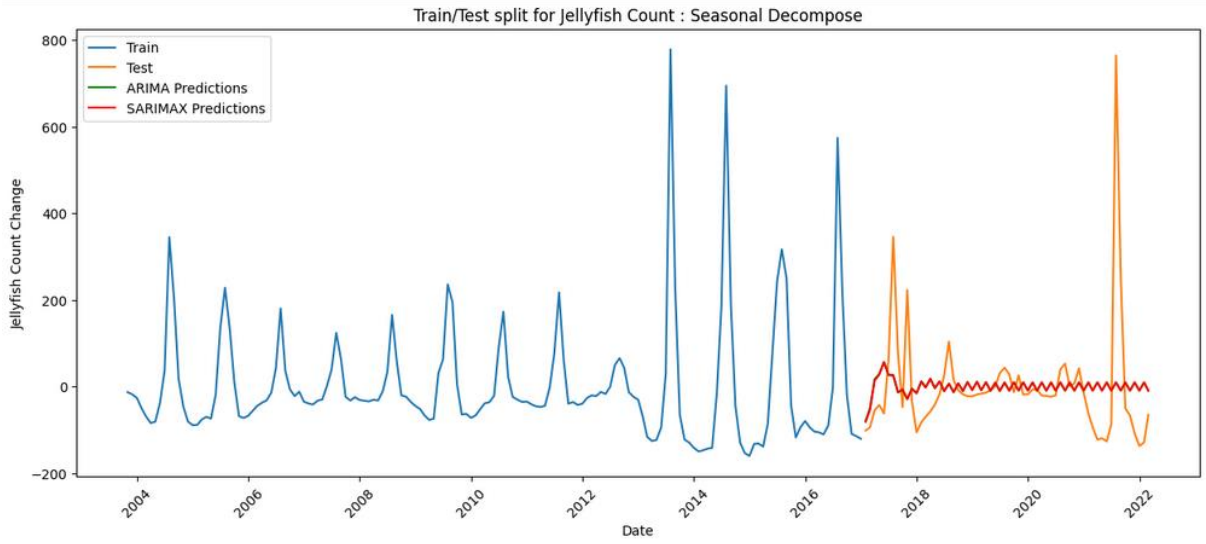
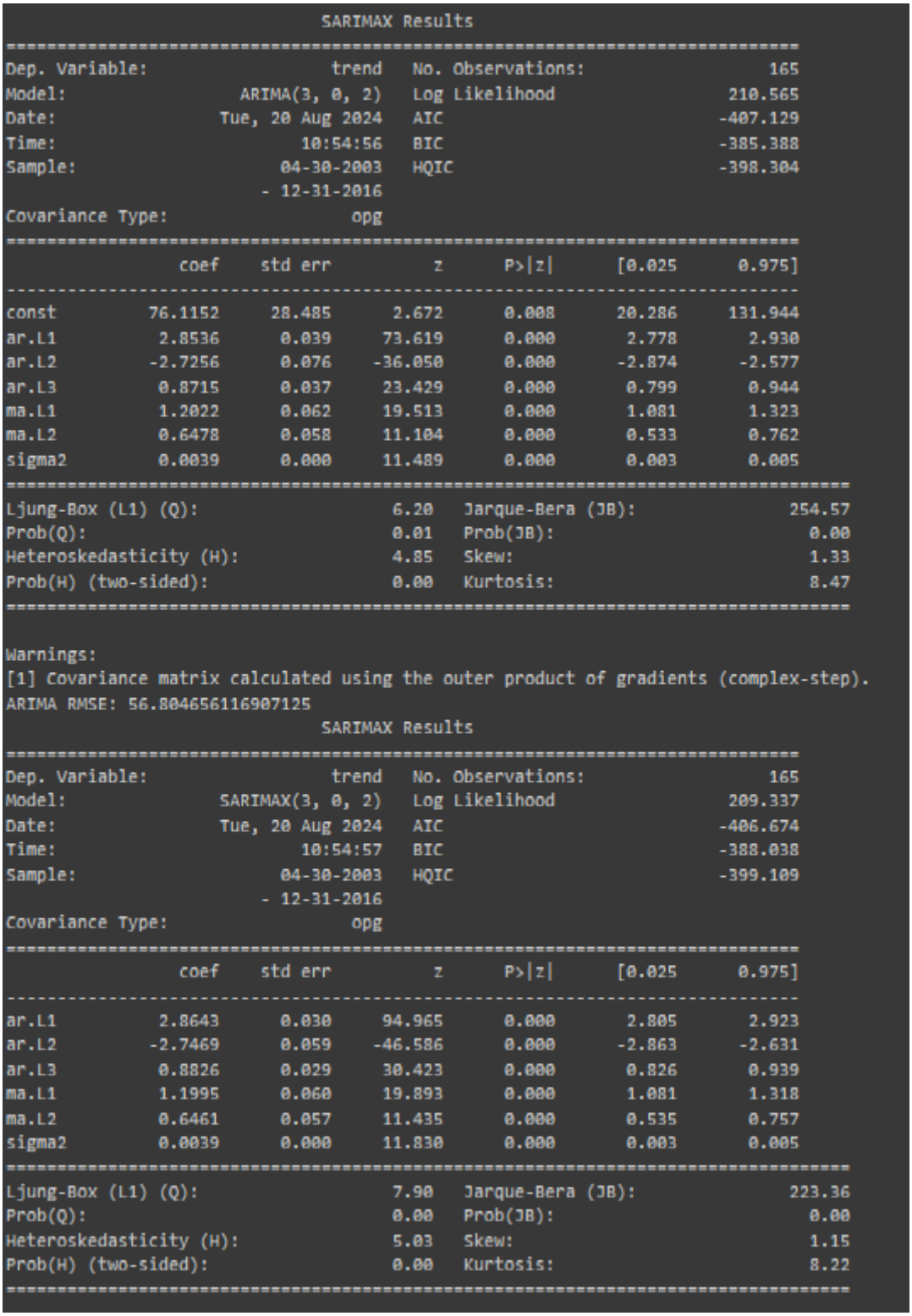


Figure 27: Jellyfish Seasonal Decompose ARIMA/SARIMAX model graph

Cyclic trend results and graph shown in Figures 28 and 29, show more interesting results. The RMSE for ARIMA is 56.8 and SARIMAX is 66.3 by far the lowest result yet. Examining the graph, it has showcased the trend instead of the data which starts low and fluctuates between high and low points. This could align with (Condon *et al.* 2013) where they claim that jellyfish blooms follow an increase and decrease in blooms every few years. But the graph also shows that the blooms could be increasing in scale based on the values around 2014 to 2016 and the increase after 2021. Again, this graph could be affected by the pandemic influencing the amount of data gathered especially as it drops so slowly and quickly increases. The models themselves manage to capture some of the data patterns and could likely be a result of where the values for 2019 to 2021 should in theory have been but again as this data is unpredictable and has outliers this cannot be concluded.



```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  trend   No. Observations:                  165
Model:                 ARIMA(3, 0, 2)   Log Likelihood                 210.565
Date:                Tue, 20 Aug 2024   AIC                           -407.129
Time:                        10:54:56   BIC                           -385.388
Sample:                    04-30-2003   HQIC                          -398.304
                         - 12-31-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         76.1152     28.485      2.672      0.008      20.286     131.944
ar.L1          2.8536      0.039     73.619      0.000       2.778       2.930
ar.L2         -2.7256      0.076    -36.050      0.000      -2.874      -2.577
ar.L3          0.8715      0.037     23.429      0.000       0.799       0.944
ma.L1          1.2022      0.062     19.513      0.000       1.081       1.323
ma.L2          0.6478      0.058     11.104      0.000       0.533       0.762
sigma2         0.0039      0.000     11.489      0.000       0.003       0.005
===================================================================================
Ljung-Box (L1) (Q):                   6.20   Jarque-Bera (JB):               254.57
Prob(Q):                              0.01   Prob(JB):                         0.00
Heteroskedasticity (H):               4.85   Skew:                             1.33
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.47
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
ARIMA RMSE: 56.804656116907125
                               SARIMAX Results
==============================================================================
Dep. Variable:                  trend   No. Observations:                  165
Model:               SARIMAX(3, 0, 2)   Log Likelihood                 209.337
Date:                Tue, 20 Aug 2024   AIC                           -406.674
Time:                        10:54:57   BIC                           -388.038
Sample:                    04-30-2003   HQIC                          -399.109
                         - 12-31-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          2.8643      0.030     94.965      0.000       2.805       2.923
ar.L2         -2.7469      0.059    -46.586      0.000      -2.863      -2.631
ar.L3          0.8826      0.029     30.423      0.000       0.826       0.939
ma.L1          1.1995      0.060     19.893      0.000       1.081       1.318
ma.L2          0.6461      0.057     11.435      0.000       0.535       0.757
sigma2         0.0039      0.000     11.830      0.000       0.003       0.005
===================================================================================
Ljung-Box (L1) (Q):                   7.90   Jarque-Bera (JB):               223.36
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               5.03   Skew:                             1.15
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.22
===================================================================================
```

Figure 28: Jellyfish Cyclic Trend ARIMA/SARIMAX model summary

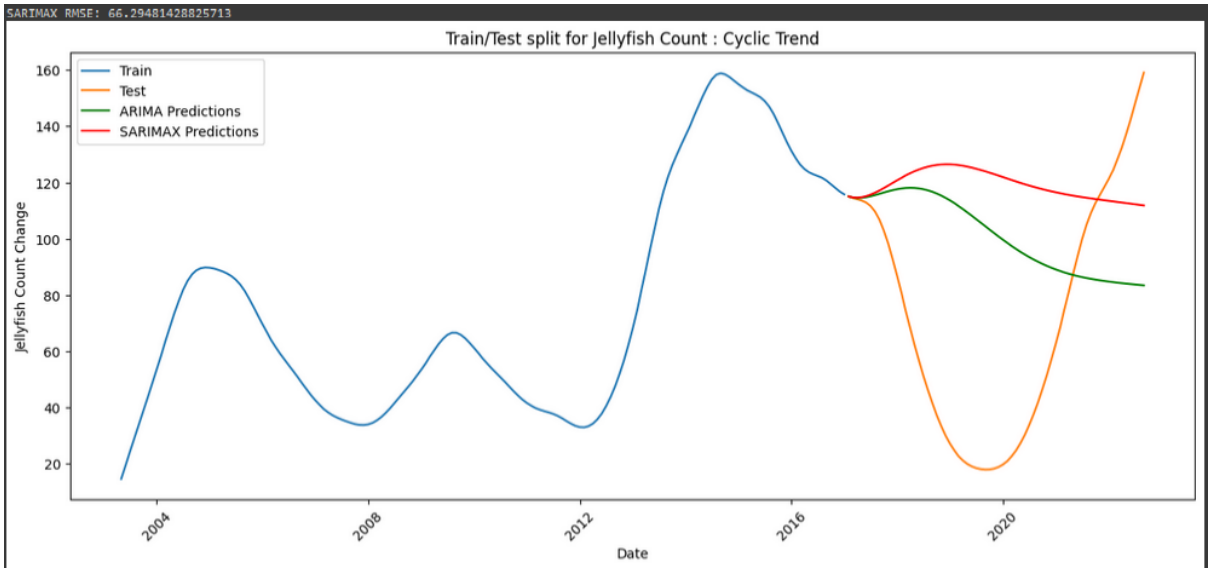


Figure 29: Jellyfish Cyclic Trend ARIMA/SARIMAX model graph

In Appendix B, there are several other graphs for rolling mean, square root, de-trended seasonal decompose and cyclic extract, these graphs and results sometimes show minor variations, and reveal no insights not gathered from the models. Building on this, each model was graphed to adapt to unseen data for the next 24 months which would encompass till the end of 2024. Each graph shows the data, its predicted trend line and the confidence interval in grey. Figure 30 shows the base data, and it fails to capture any pattern in the data. Figure 31 shows the first difference which manages to capture some of the data's seasonality but also fails to extract the outliers in the data.

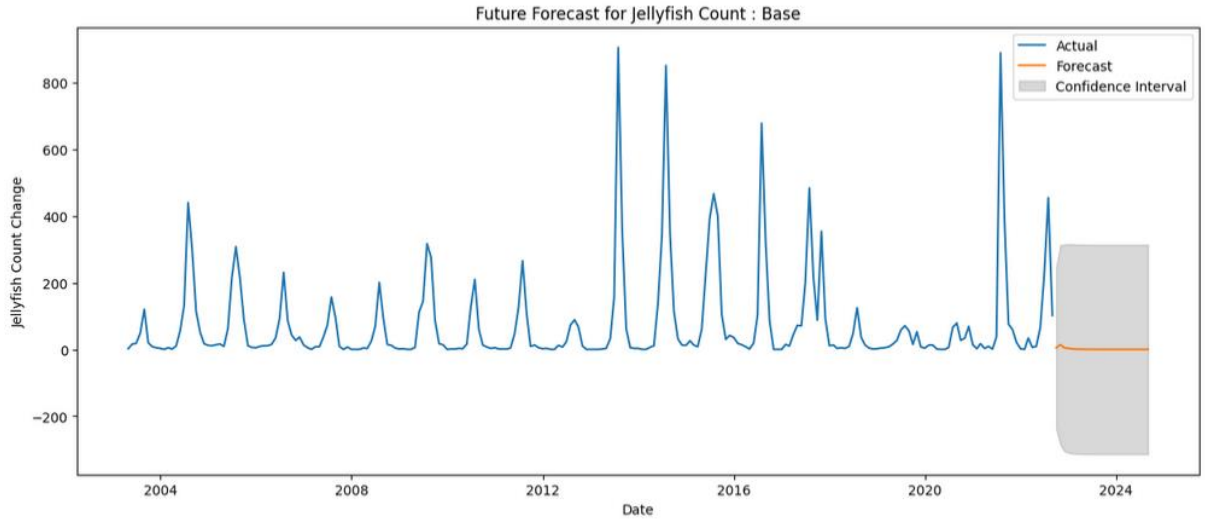


Figure 30: Jellyfish Base Future Forecast

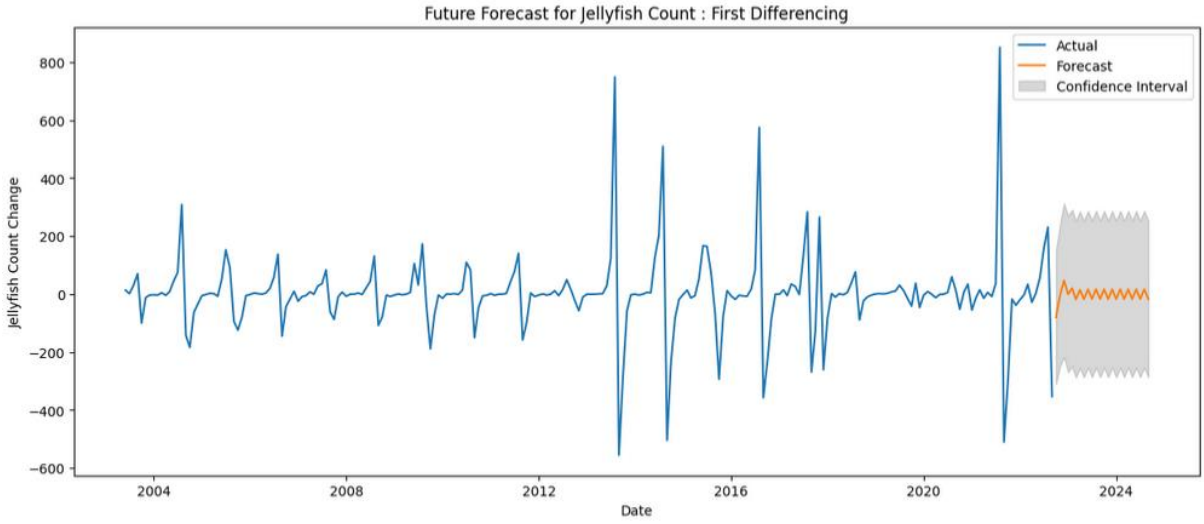


Figure 31: Jellyfish First Difference Future Forecast

Figure 32 shows the cyclic trend, which had the best values in the test stage, and it predicts that the highs of 2024 will be higher than the highs of 2015 before beginning to decrease again. This suggests again about the theory of oscillation (Condon *et al.* 2013). But it could also suggest that while it is oscillating, the highs are increasing which could be devasting to local ecosystems. The cyclic extract is the only graph with deviation from the first difference or base, the others listed before are all shown in Appendix B as they are either straight lines or small fluctuations.

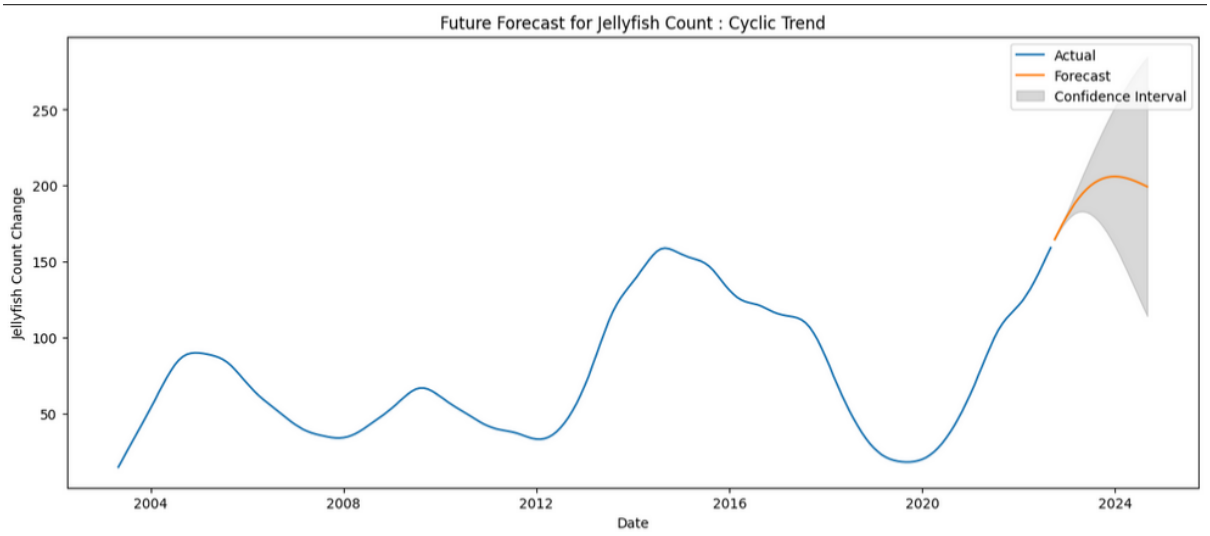


Figure 32: Jellyfish Cyclic Trend Future Forecast

The same methods were performed on the climate data to varying degrees of success. For example, Figures 33 and 34 show the model summary and graph for max temperature in its base state. It manages to achieve an RMSE of 1.9 and 3 for ARIMA and SARIMAX respectively. The graph itself shows that it has managed to capture the pattern present in the data with a minor decrease in temperature

overall compared to the actual temperature which is slowly increasing. The rest are shown in Appendix B to varying degrees of success with some managing to get highly accurate graphs and extremely good RMSE scores.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                MaxTemp   No. Observations:                 1200
Model:                 ARIMA(3, 0, 2)   Log Likelihood               -2094.720
Date:                Tue, 20 Aug 2024   AIC                           4203.441
Time:                        10:55:13   BIC                           4239.071
Sample:                    01-01-1910   HQIC                          4216.862
                         - 12-01-2009
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.0929      0.055    220.243      0.000      11.985      12.201
ar.L1          1.9777      0.033     59.215      0.000       1.912       2.043
ar.L2         -1.4265      0.058    -24.760      0.000      -1.539      -1.314
ar.L3          0.2479      0.033      7.499      0.000       0.183       0.313
ma.L1         -1.6481      0.019    -84.607      0.000      -1.686      -1.610
ma.L2          0.9150      0.018     49.866      0.000       0.879       0.951
sigma2         1.9613      0.084     23.357      0.000       1.797       2.126
===================================================================================
Ljung-Box (L1) (Q):                   0.83   Jarque-Bera (JB):                 9.11
Prob(Q):                              0.36   Prob(JB):                         0.01
Heteroskedasticity (H):               0.98   Skew:                            -0.08
Prob(H) (two-sided):                  0.82   Kurtosis:                         3.39
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
ARIMA RMSE: 1.924012417089566
                               SARIMAX Results
==============================================================================
Dep. Variable:                MaxTemp   No. Observations:                 1200
Model:               SARIMAX(3, 0, 2)   Log Likelihood               -2326.344
Date:                Tue, 20 Aug 2024   AIC                           4664.688
Time:                        10:55:15   BIC                           4695.228
Sample:                    01-01-1910   HQIC                          4676.192
                         - 12-01-2009
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          2.7315      0.001   5343.610      0.000       2.730       2.732
ar.L2         -2.7310      0.001  -3100.045      0.000      -2.733      -2.729
ar.L3          0.9995      0.000   2062.886      0.000       0.999       1.000
ma.L1         -1.7223      0.007   -251.720      0.000      -1.736      -1.709
ma.L2          0.9970      0.007    137.840      0.000       0.983       1.011
sigma2         2.7782      0.115     24.243      0.000       2.554       3.003
===================================================================================
Ljung-Box (L1) (Q):                 142.55   Jarque-Bera (JB):                 1.09
Prob(Q):                              0.00   Prob(JB):                         0.58
Heteroskedasticity (H):               0.92   Skew:                             0.05
Prob(H) (two-sided):                  0.43   Kurtosis:                         3.11
===================================================================================
```

Figure 33: Max Temperature Base ARIMA\SARIMAX model summary

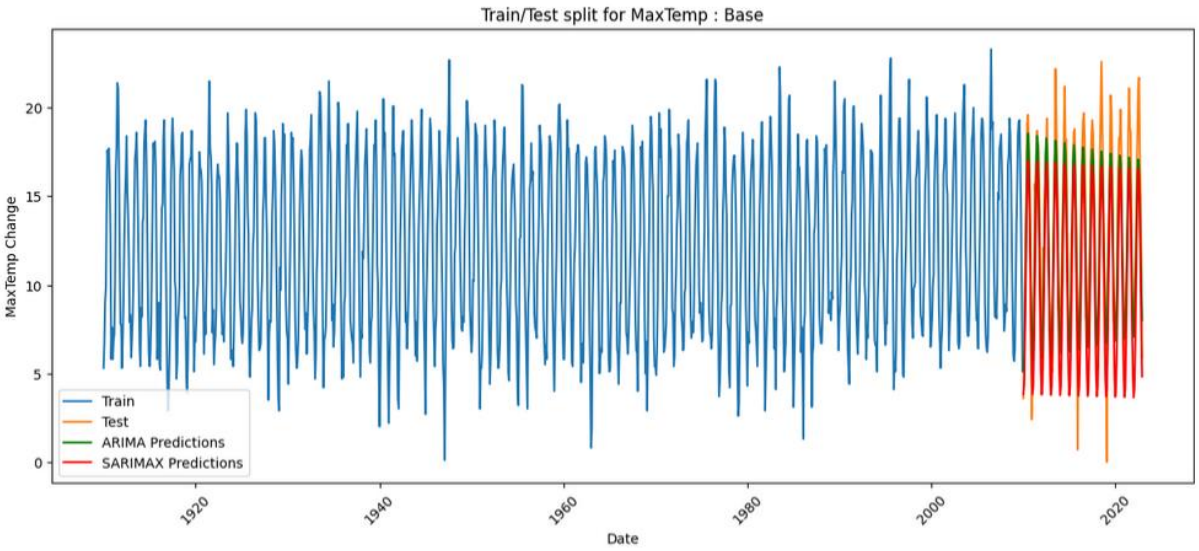


Figure 34: Max Temperature Base ARIMA\SARIMAX model graph

In Figure 35, the future forecast for the base of max temperature is shown. The orange line showcases the prediction, and the grey is the confidence interval. The confidence interval for this data is quite broad potentially showing the potential range of temperature change to vary. In the next two years, it does not predict any significant increase, but its confidence interval implies it could be much higher than any previous temperature which aligns with a lot of current predictions about temperature trends. Though again due to outliers it is not something that can be accurately predicted. Appendix B shows the remaining climate predictions, this data while interesting often failed to capture the pattern of the data with several values much lower or higher than expected, meaning that the previous models could be overfitting a lot.

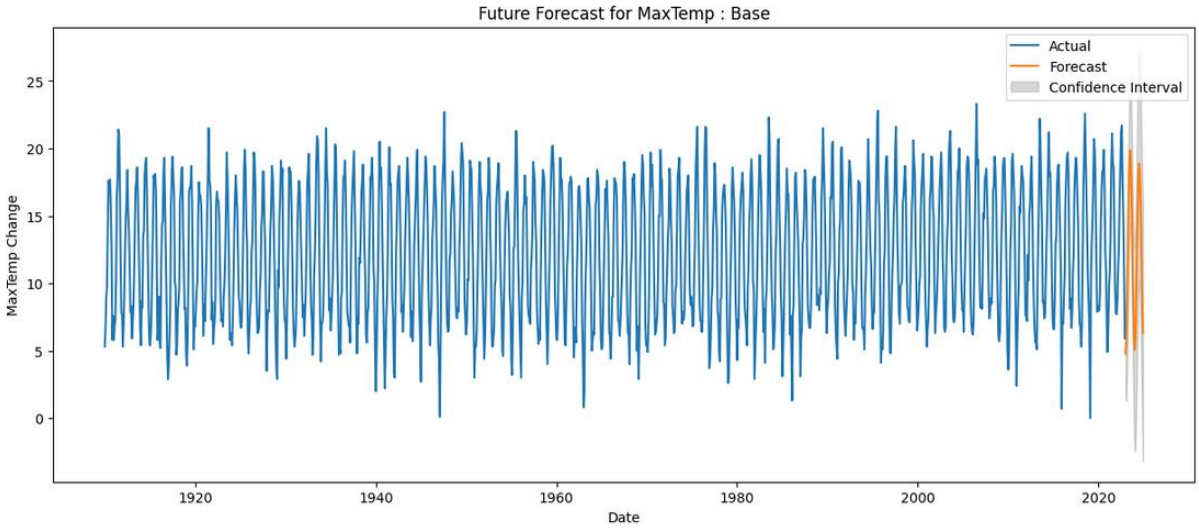


Figure 35: Mean Temperature Future Forecast

5.3.3. Auto ARIMA

Using Auto ARIMA to find the best order incurred a different answer than the best_order function created that utilised time series split, and it worked quicker. This answer was (1, 0, 0) as opposed to (3, 0, 2). The (1, 0, 0) also aligned better with the results from the ACF and PACF plots. It has the benefit of detecting seasonality as well and this resulted in a model with an RMSE of 128.5. While still high examining the graph in Figure 36 shows it to be satisfactory as it can gather the expected trends and spikes of data especially due to seasonality. It shows a slow decrease over time of overall jellyfish populations suggesting it gathered a pattern of oscillations in their population outside of seasonality as suggested by (Condon *et al.* 2013). The predictions seem to align with the data if there were no outliers in 2020 and a massive spike in 2021 in comparison.

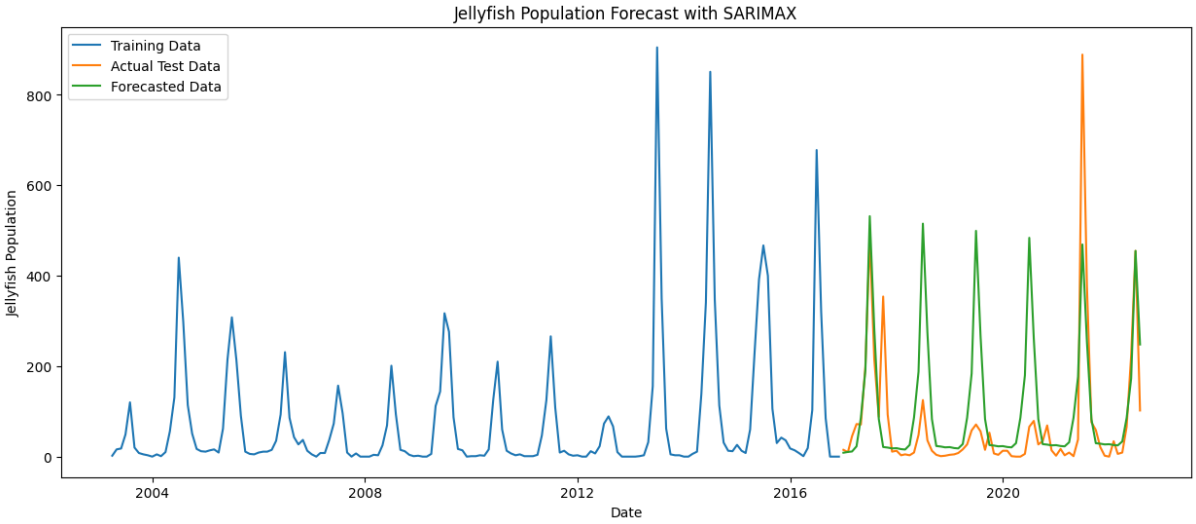


Figure 36: Jellyfish Auto ARIMA Graph

Expanding this model to account for unseen future values is shown in Figure 37. Based on the previous testing it picks up the same pattern as before of spikes during the summer months and dips during the winter months. The pattern doesn't feature any increase or decrease in the overall trend suggesting jellyfish populations will remain consistent from 2022 to 2024, which based on the data makes sense as data is often dependent on the previous year barring a major spike after 2013 and a decrease in 2019.

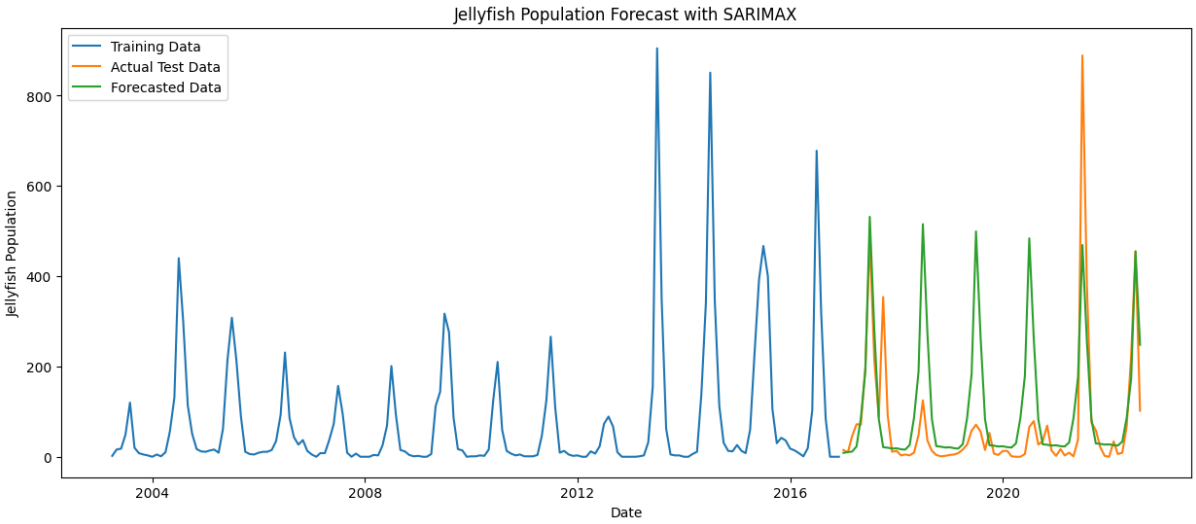


Figure 37: Jellyfish Auto ARIMA Future Predictions Graph

5.4 Deep Learning Time Series Results

5.4.1. LSTM Results

The TensorFlow LSTM used achieved an RMSE of 153.2 which is quite high and in line with the values that were achieved for the statistical methods. However, examining the graph, it has managed to extract the pattern from the data and only fails in the exact values and extreme outliers, see Figure 38. It suggests that this LSTM is a good model for predicting the overall trend and pattern of the jellyfish even if it still fails to calculate the massive outlier spikes.

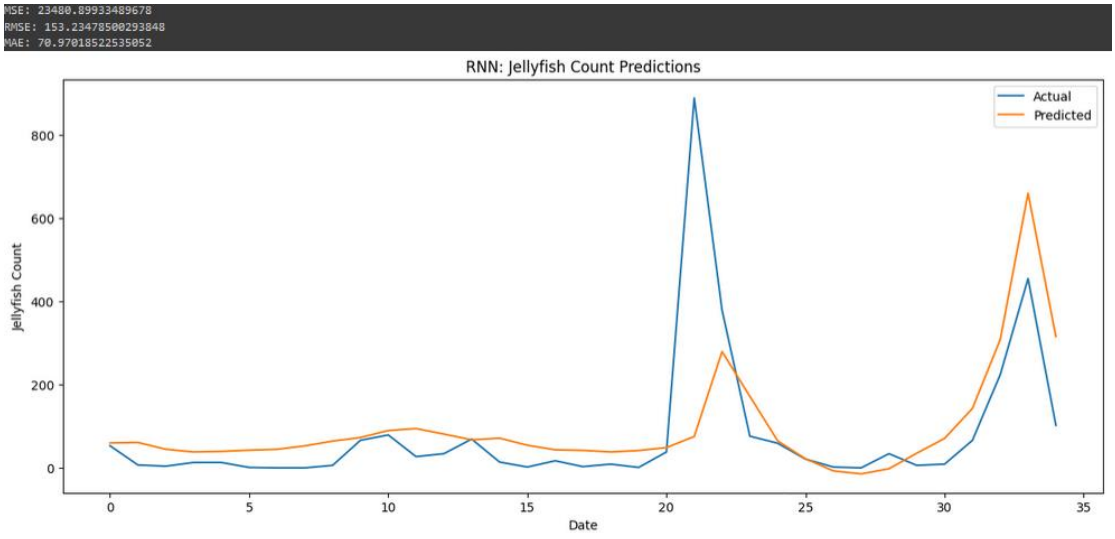


Figure 38: TensorFlow LSTM Model Graph

Furthering this model further for unseen data is shown in Figure 39, this model again does not capture the outliers to an extreme extent but manages to accurately capture the pattern of the data. It suggests that the summers of 2023 and 2024 will have lower counts than 2021

and 2022 which could be the case as the series suggests after a peak population slowly decreases again. However, the predictions don't show an overall increase in the data compared to previous years which most of the statistical time series did, especially when examining the overall trends. So, the LSTM either failed to learn this or thinks differently to the ARIMA models.

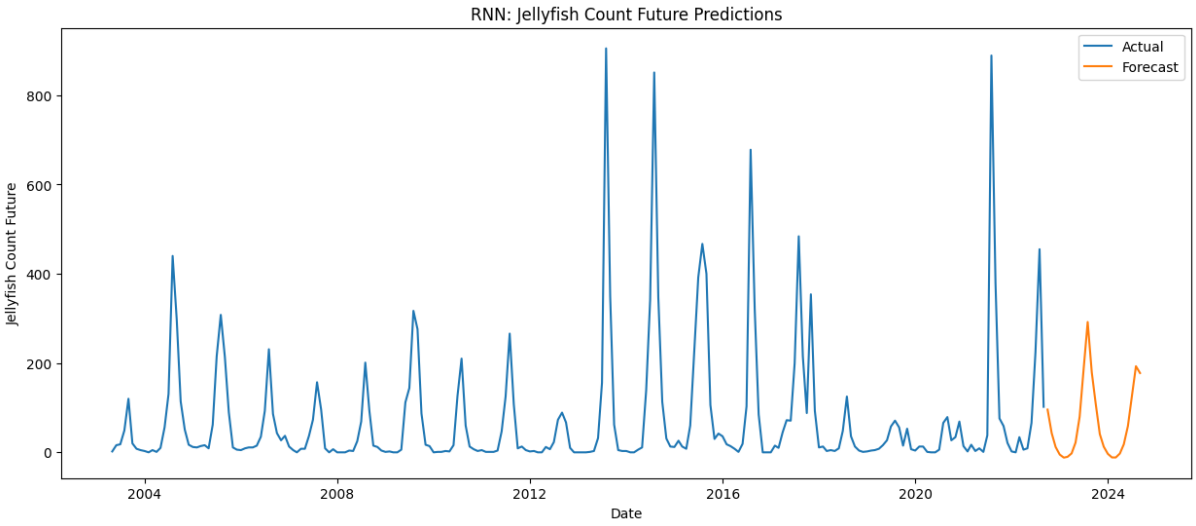


Figure 39: TensorFlow LSTM Future Forecast Graph

5.4.2. Multivariate LSTM Results

Combining the climate data with the jellyfish data and converting the whole information to a tensor for prediction to see if the factors will have a bearing on the data resulted in the graph shown in Figure 40. This model is an LSTM created using PyTorch and it produced an RMSE of 137.8 which is slightly better than the previous LSTM. The graph itself is interesting again it captured the oscillating pattern of the data between different years but its values at the start are distinct spikes when compared to the data. This could potentially be a more accurate reading of jellyfish recordings during the pandemic when record gathering was affected. It failed to capture the total range of the 2021 spike, but this is likely due to outliers in the data.

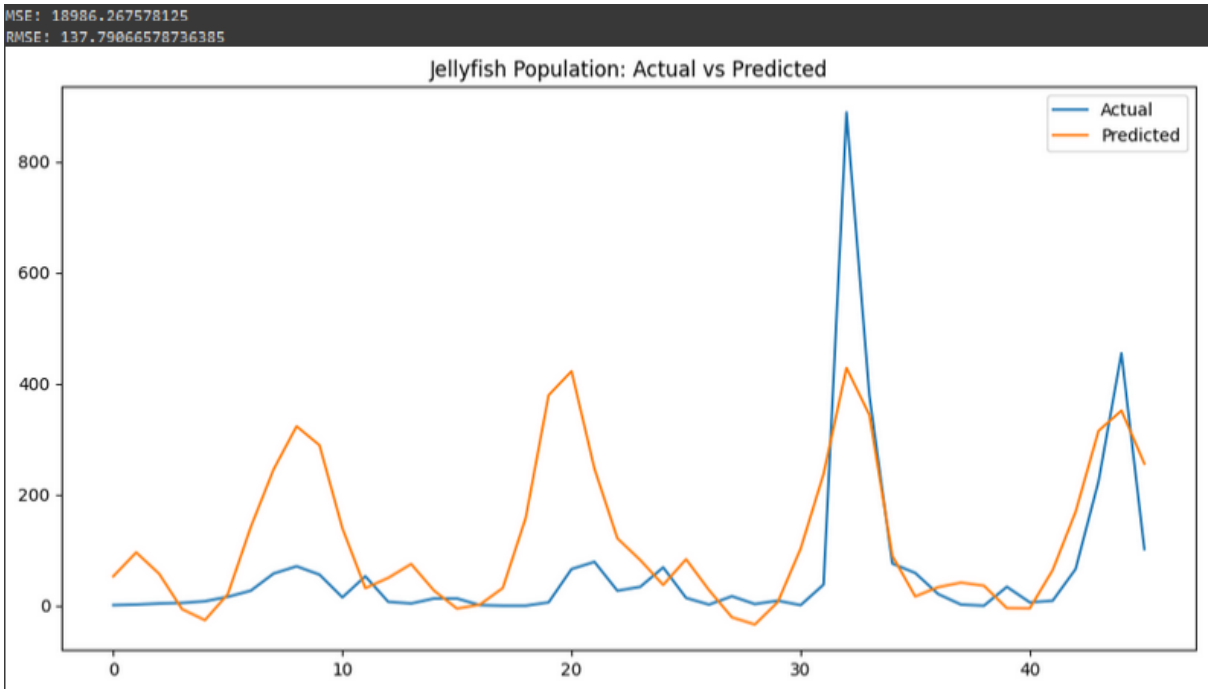


Figure 40: PyTorch Multivariate LSTM Actual vs Predicted Plot

In Figure 41, the future forecast of this model, the multivariate LSTM showcases it predicts massive spikes in the next two years much higher than any other model before it. This could be accurate based on the data as it may have seen a larger increase in the climate factors than in previous years resulting in a direct correlation to the jellyfish population. Also, a large spike after a decrease is not unheard of in the data as it occurred between 2012 and 2013. More inclined to believe that the model is predicting a spike in these years, but the model has blown the scale of it of proportion, but it is still interesting, nonetheless.

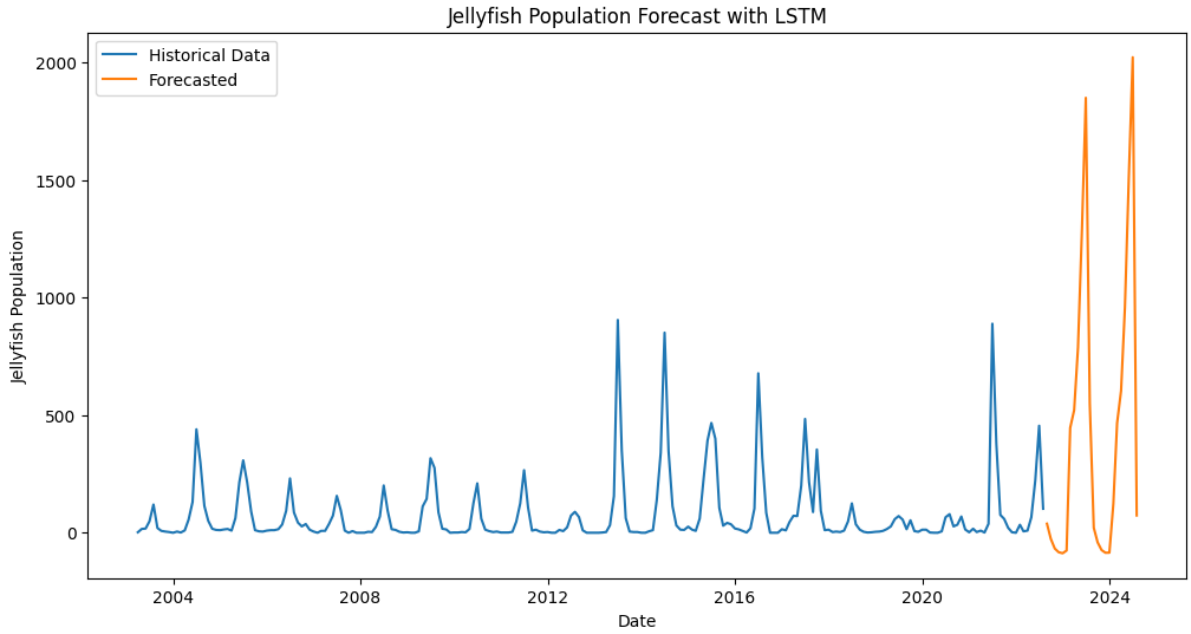


Figure 41: Multivariate LSTM Future Forecast

5.4.3 Transformer Results

The transformer model achieved an RMSE of 127 which is close to a lot of the other models that achieved RMSE around that range, but it is in the lower range. Examining the graph in Figure 42 verifies this information as it has extracted the underlying pattern of the data where it has managed to predict that the summer months will experience spikes but fails to capture the highs and lows of each. It seems to have extrapolated a base value that spikes during the summer months. While on paper this model seems to have worked well, examining in detail it fails a lot of criteria and there is a strong possibility of overfitting in this data which is more evident in Figure 43 where it is subjected to unseen values. Here the graph has an initial spike but trails off to nothing suggesting that this model has learned nothing of note.

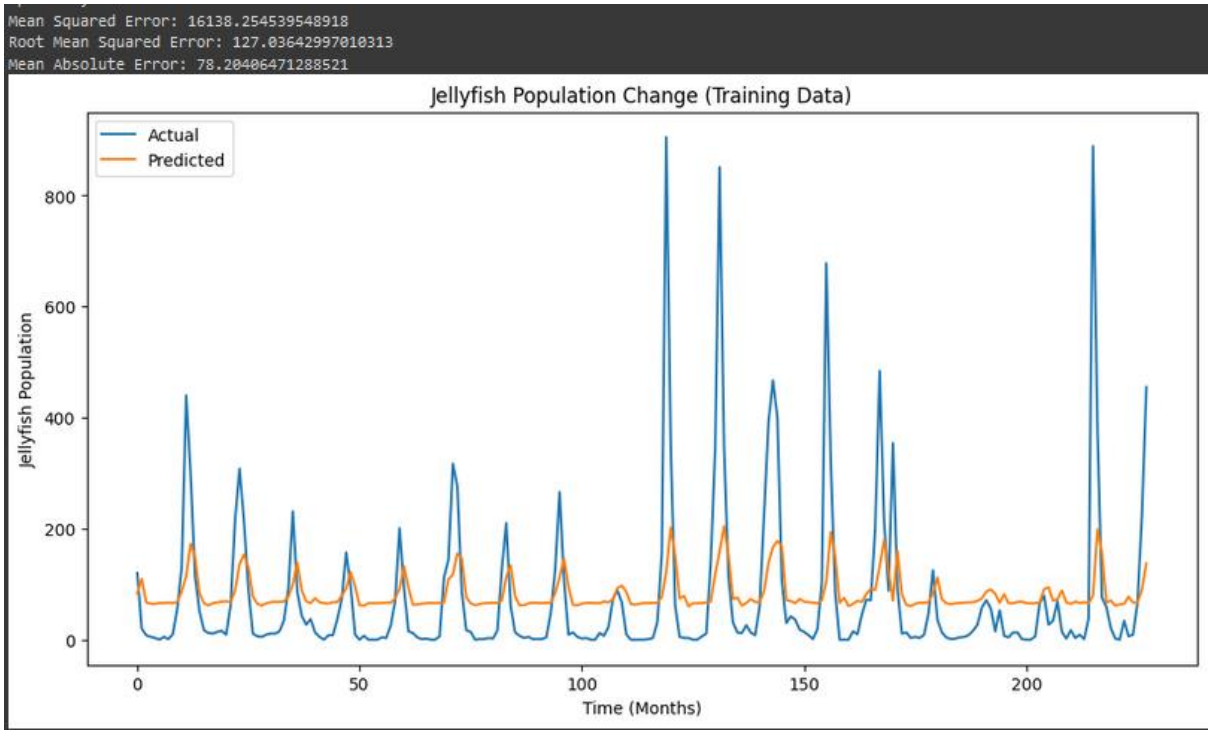


Figure 42: Transformer Model Jellyfish Data Graph

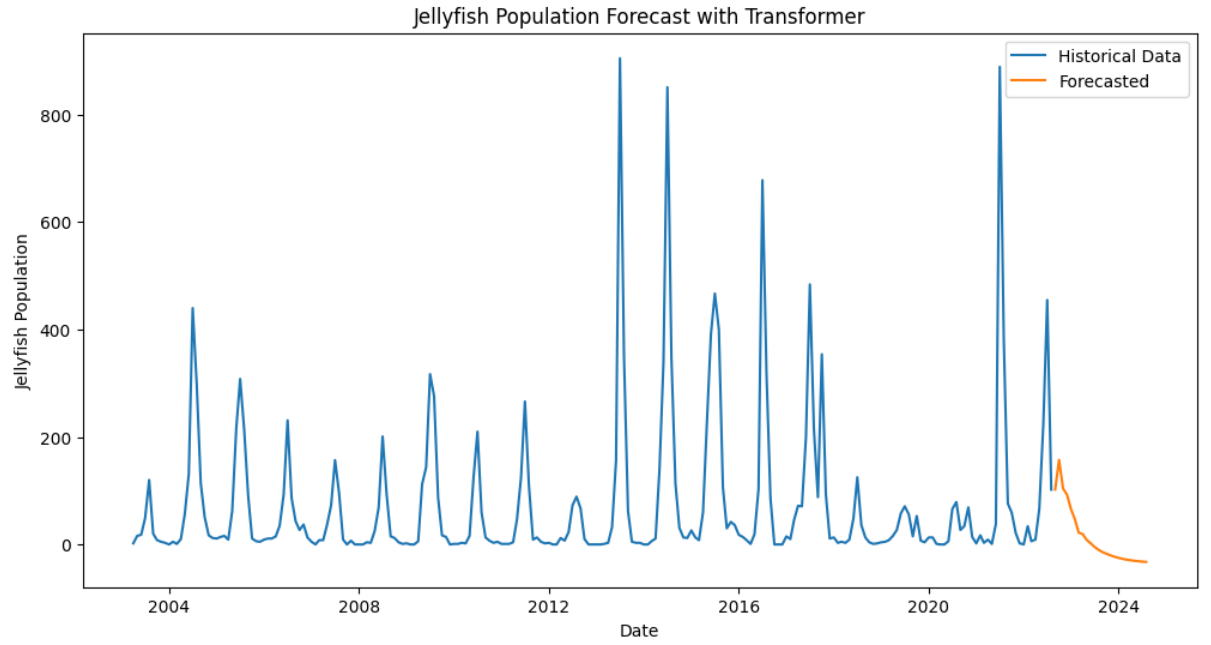


Figure 43: Transformer Model Jellyfish Data Unseen Values

5.5. Conclusion

Individually a lot of the metrics and graphs by themselves show relatively poor results with high RMSE scores all around and a lot of graphs, some with overfitting and some that failed to capture any pattern at all, but few results provide interesting results about jellyfish

population dynamics. The SARIMAX and ARIMA models had varying success levels on the jellyfish dataset but it's clear that on a lot of different orders, it extracted the pattern of seasonality and some like the cyclic trend achieved a relatively low RMSE score and a graph that predicted the trend. While the trend did not necessarily line up with the actual data it helped in confirmation that the data may have been affected by the covid-19 pandemic. Further forecasting this into the future suggested that the trend of jellyfish should increase. The Auto ARIMA library outperformed most of the manual work done in this dissertation in extracting parameters and the model attained was much better than most statistical methods. This graph managed to extract the pattern and, in the future, suggests that jellyfish populations will continue their seasonality but remain relatively stable year after year. This could be true, or it may not be able to predict anything with the outliers that plague this data.

The deep learning time series managed to successfully extract the pattern from the dataset much more easily than any of the statistical methods. The basic LSTM while not achieving a strong RMSE managed to capture the pattern well just not the scale to any detail, again due to the presence of outliers in the data and when presented with unseen variables it seemed to agree with the Auto ARIMA function of a relatively stable population but an overall decrease. The multivariate LSTM manages to capture the outliers of the dataset much better than any before likely because it has far more variables to work with and it incorrectly guesses the populations for the pandemic years but accurately the rest which would give further credibility that there is a constraint on this data as a result. Looking at its future predictions, its scale is off, but the pattern is correct suggesting that potentially in contrast to others there will be an increase in population in the future. The transformer model while it captured the data well, failed to transfer this information to unseen values suggesting it is overfitting.

Overall, there are disagreements between the models about how jellyfish populations will change over the next few years but it's possible that by taking different information from each the research questions asked can be successfully answered.

Chapter 6: Conclusion

6.1. Discussion

This dissertation investigated the population dynamics of jellyfish around the coast of Ireland and the UK by using both statistical and deep learning methods and comparing this to aspects of climate change. This project accomplished all the aims as it investigated the statistical and deep learning time series approaches fulfilling aims 1 and 2. These methods were performed on the climate data and jellyfish data where different trends were discovered, accomplishing aims 3 and 4.

There were two research questions posed:

1. How do classical time series approaches differ from deep learning-based approaches?

2. What trends are present in Jellyfish populations around the coast of the UK due to factors such as increased temperatures and rising sea levels?

The project easily answers the first question as both the statistical and deep learning approaches performed similarly in terms of their results. Both had successful and unsuccessful models. The successful models found various contradicting conclusions but overall showed the potential for deep learning models to outperform statistical approaches especially when presented with complex multivariate problems with high degrees of outliers where the LSTM shone.

The second question is harder to answer as stated there are some contradicting results between the successful models and the problems with the data but there are a few conclusions that can be drawn. First one of the most successful models used multivariate jellyfish population and climate and ended up being one of the most accurate. This model suggests the oscillation pattern and an overall increase in jellyfish due to increased temperatures and rainfall. Some of the other models also suggested this oscillating pattern with an overall increase in population at the high ends.

The literature review section focused on the different aspects of time series, jellyfish and climate change to give a solid background to all three. There was then a focus on specific similar papers like (Condon *et al.* 2013) which concluded that jellyfish blooms increase and

decrease in number in cycles or oscillations but they found that the oscillations spikes may be increasing due to climate change. Another dissertation (Lynam *et al.* 2011) focused on the same areas as this dissertation but focused on catches rather than sightings and they found that climate change and overfishing had a significant impact on jellyfish populations and had the potential to destroy ecosystems. These papers reached these conclusions without the use of deep learning algorithms, unlike this dissertation which made use of both an LSTM model and a Transformer model.

The design chapter and implementation chapter detailed the concepts utilised and steps taken to complete the project, but the results chapter presented various answers through the explorations, statistical and deep learning models. Early explorations gave insights that would define the project going forward mainly from the visualisations of date, which showcased the difference in data between the winter and summer months as well as the disparity between different years, especially the drop in the series in 2020 which has a high chance of being attributed to the pandemic and demonstrates the human aspect of the data set which may have a bearing on the overall results.

Analysing the jellyfish dataset further revealed information on its stationarity and autocorrelation where it had to be transformed before applying an ARIMA model to the data. While not all the ARIMA models ended up being successful, some like the cyclic trend seem to agree with (Condon *et al.* 2013) where the trend is forecasting an increasing oscillation cycle which could be attributed to climate change. The Auto ARIMA model was the strongest statistical model created and it managed to capture the underlying pattern of the data well and potentially showcase where 2020 would have been without limitations. The future forecast of this model oscillated high and low but remained consistent with these highs and lows either suggesting it expects the population to level off or this is the equilibrium of the population.

Unfortunately, the transformer model failed to capture much meaningful data from the model as it overfits the data and its outliers, but the LSTM was quite successful in its predictions. The TensorFlow LSTM managed to accurately capture the pattern and future forecast a decrease for the next two years in population. The PyTorch model utilised the multivariate climate data and this model also captured the pattern well. It like the Auto

ARIMA model suggested that the 2020 values should be different and managed to more accurately capture the outlier spikes. Its future forecast is likely a bit too high due to predictions made from the climate data as well.

6.2. Limitations

Several limitations directly or indirectly influenced the results of this project. As this is a data science dissertation, the biggest limiting factor is the data itself. The jellyfish data would have benefitted from having a lot more information and consistency. The original dataset had values on the abundance of jellyfish but very few rows had values here and the range was too big. If this information was consistent and more descriptive it would have revealed greater information about jellyfish blooms themselves and could change the results. The dataset itself could have more records in general which would have allowed for specific species or families to be examined and still gain acceptable results. Another dataset for climate and sea level could have been preferable if it corresponded better to the jellyfish data in terms of location and date which again could further prove the connection between them.

The models themselves are also a limiting factor, there is only so much a model can successfully learn especially the statistical models which struggled a lot with the outliers in the data. It is also difficult to find the best possible parameters for each model when there are so many so there is a chance that some of the models could still be improved further. The deep learning models also suffered from these issues where they are quite basic models with little parameter tuning compared to a basic model. A lot of these models are still in their infancy compared to the ARIMA models and addressing these issues could have led to better results.

6.3. Future Work

Some of the future work has already been hinted at by achieving some better data could lead to better results and improving the model parameters further may fix issues of over and underfitting. The literature review chapter detailed a few advanced models like the iTransformer (Peixeiro 2024) and combining a Convolutional Neural Network with an LSTM (Chou *et al.* 2021) has sometimes outperformed the base models but at this stage, they are still being tested and can be difficult to successfully implement. There are also improvements

to be made in the consistency of graphs and information returned from a deep learning model did not align with the statistical methods making them slightly more difficult to compare.

The data itself should it be improved, could result in better results and filterable results based on region or species which would give further insights other than just jellyfish around Ireland and the UK. If this was achieved, the range could be expanded to more areas around the world and encompass more species of jellyfish as well. This could culminate together in an application or website that creates a model based on parameters specifying climate parameters, species and an area or ocean.

6.4. Conclusion

In conclusion, while there are problems with the data and some of the models the most probable conclusion is similar to Condon's, where jellyfish blooms are not reliant on climate change and there seem to be other factors at play that are contributing to this cycle of increasing and decreasing populations, but the upper end of this cycle is increasing at a potentially alarming rate that can be attributed to the effects of climate change such as higher ocean temperatures and an increase in sea level.

As our understanding of climate change improves further thanks to more complex AI models and specific research the links between jellyfish population dynamics and oceanographic conditions will likely become clearer and lead to concrete discoveries in this field. But there is no doubt that jellyfish will continue to play a vital role in Irish and British waters for the foreseeable future. Further research with more complex data will likely improve our grasp of jellyfish ecology in the face of ongoing environmental changes and potentially offer solutions to this emerging issue.

Appendices

Appendix A: References

Bayha, K.M. and Graham, W.M. (2014) 'Nonindigenous Marine Jellyfish: Invasiveness, Invasibility, and Impacts', in Pitt, K.A. and Lucas, C.H., eds., *Jellyfish Blooms*, Dordrecht: Springer Netherlands, 45–77, available: https://doi.org/10.1007/978-94-007-7015-7_3.

Bengio, Y., Goodfellow, I., and Courville, A. (2016) *Deep Learning* [online], 1st ed., Boston: MIT Press, available: https://www.deeplearningbook.org/ [accessed 12 Nov 2022].

van Beusekom, J.E.E. (2018) 'Eutrophication', in Salomon, M. and Markus, T., eds., *Handbook on Marine Environment Protection : Science, Impacts and Sustainable Management*, Cham: Springer International Publishing, 429–445, available: https://doi.org/10.1007/978-3-319-60156-4_22.

Buduma, N., Buduma, N., and Papa, J. (2022) *Fundamentals of Deep Learning*, O'Reilly Media, Inc.

Cegolon, L., Heymann, W.C., Lange, J.H., and Mastrangelo, G. (2013) 'Jellyfish Stings and Their Management: A Review', *Marine Drugs*, 11(2), 523–550, available: https://doi.org/10.3390/md11020523.

Chen, A. (2024) Quantitative_Trading_and_Research/Transformer, Time Series, Tsfresh, Mann Whitney Test, Benjamini Yekutieli Procedure.Ipynb at Master · Adam5644/Quantitative_Trading_and_Research [online], *GitHub*, available: https://github.com/adam5644/Quantitative_Trading_and_Research/blob/master/Transformer%2C%20time%20series%2C%20tsfresh%2C%20Mann%20Whitney%20test%2C%20Benjamini%20Yekutieli%20procedure.ipynb [accessed 26 Aug 2024].

Chiodi, A., Gargiulo, M., Rogan, F., Deane, J.P., Lavigne, D., Rout, U.K., and Ó Gallachóir, B.P. (2013) 'Modelling the impacts of challenging 2050 European climate mitigation targets on Ireland's energy system', *Energy Policy*, 53, 169–189, available: https://doi.org/10.1016/j.enpol.2012.10.045.

Chou, C., Park, J., and Chou, E. (2021) 'Generating High-Resolution Climate Change Projections Using Super-Resolution Convolutional LSTM Neural Networks', in *2021 13th International Conference on Advanced Computational Intelligence (ICACI)*, Presented at the 2021 13th International Conference on Advanced Computational Intelligence (ICACI), 293–298, available: https://doi.org/10.1109/ICACI52617.2021.9435890.

Condon, R.H., Duarte, C.M., Pitt, K.A., Robinson, K.L., Lucas, C.H., Sutherland, K.R., Mianzan, H.W., Bogeberg, M., Purcell, J.E., Decker, M.B., Uye, S., Madin, L.P., Brodeur, R.D., Haddock, S.H.D., Malej, A., Parry, G.D., Eriksen, E., Quiñones, J., Acha, M., Harvey, M., Arthur, J.M., and Graham, W.M. (2013) 'Recurrent jellyfish blooms are a consequence of global oscillations', *Proceedings of the National Academy of Sciences*, 110(3), 1000–1005, available: https://doi.org/10.1073/pnas.1210920110.

Dessler, A.E. (2021) *Introduction to Modern Climate Change*, 2nd Edition., New York, NY, USA: Cambridge University Press.

Doyle, T.K., Hays, G.C., Harrod, C., and Houghton, J.D.R. (2014) 'Ecological and Societal Benefits of Jellyfish', in Pitt, K.A. and Lucas, C.H., eds., *Jellyfish Blooms*, Dordrecht: Springer Netherlands, 105–127, available: https://doi.org/10.1007/978-94-007-7015-7_5.

Doyle, T.K., Houghton, J.D.R., Buckley, S.M., Hays, G.C., and Davenport, J. (2007) 'The broad-scale distribution of five jellyfish species across a temperate coastal environment', *Hydrobiologia*, 579(1), 29–39, available: https://doi.org/10.1007/s10750-006-0362-2.

El Gamal, A.A. (2010) 'Biological importance of marine algae', *Saudi Pharmaceutical Journal*, 18(1), 1–25, available: https://doi.org/10.1016/j.jsps.2009.12.001.

Environmental Protection Agency (2021) *Climate Status Report for Ireland 2020*, 286, EPA, available: https://www.epa.ie/publications/research/epa-research-2030-reports/research-386.php [accessed 10 Nov 2021].

Fernández-Alías, A., Marcos, C., and Pérez-Ruzafa, A. (2024) 'The unpredictability of scyphozoan jellyfish blooms', *Frontiers in Marine Science*, 11, available: https://doi.org/10.3389/fmars.2024.1349956.

Fossette, S., Gleiss, A.C., Chalumeau, J., Bastian, T., Armstrong, C.D., Vandenabeele, S., Karpytchev, M., and Hays, G.C. (2015) 'Current-Oriented Swimming by Jellyfish and Its Role in Bloom Maintenance', *Current Biology*, 25(3), 342–347, available: https://doi.org/10.1016/j.cub.2014.11.050.

Gattuso, J.-P. and Hansson, L. (2011) *Ocean Acidification*, OUP Oxford.

Gershwin, L. (2013) *Stung!: On Jellyfish Blooms and the Future of the Ocean*, University of Chicago Press.

Geukjian, S. (2024) 'Fish and Overfishing', available: https://www.kaggle.com/datasets/kkhandekar/global-sea-level-1993-2021 [accessed 22 Jul 2024].

Gibbons, M.J., Boero, F., and Brotz, L. (2016) 'We should not assume that fishing jellyfish will solve our jellyfish problem', *ICES Journal of Marine Science*, 73(4), 1012–1018, available: https://doi.org/10.1093/icesjms/fsv255.

Greene, C.H., Pershing, A.J., Cronin, T.M., and Ceci, N. (2008) 'Arctic Climate Change and Its Impacts on the Ecology of the North Atlantic', *Ecology*, 89(sp11), S24–S38, available: https://doi.org/10.1890/07-0550.1.

Hanlon, H.M., Bernie, D., Carigi, G., and Lowe, J.A. (2021) 'Future changes to high impact weather in the UK', *Climatic Change*, 166(3), 50, available: https://doi.org/10.1007/s10584-021-03100-5.

Jones, N. and Neill, S.P. (2020) 'Assessment of Microplastic in the Western Irish Sea Gyre: From Surface to Sediment', 2020, H180-04.

Kamaruddin, S., Buyong, F., Abdullah, A., Tajam, J., Azis, T., Anscelly, A., Che Ismail, C.-Z., Hashim, A., and Nazir, E. (2023) 'Jellyfish Blooming: Are We Responsible?', Presented at the ICAN International

Virtual Conference, available: https://www.researchgate.net/publication/368392135_Jellyfish_Blooming_Are_We_Responsible.

Kennerley, A., Lorenzoni, I., Luisetti, T., Wood, L.E., and Taylor, N.G.H. (2021) 'Mapping habitats for the suitability of jellyfish blooms around the UK and Ireland', *Hydrobiologia*, 848(7), 1535–1552, available: https://doi.org/10.1007/s10750-021-04539-4.

Kontopoulou, V.I., Panagopoulos, A.D., Kakkos, I., and Matsopoulos, G.K. (2023) 'A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks', *Future Internet*, 15(8), 255, available: https://doi.org/10.3390/fi15080255.

Koustubhk (2024) 'Global Sea Level | 1993 - 2021', available: https://www.kaggle.com/datasets/kkhandekar/global-sea-level-1993-2021 [accessed 22 Jul 2024].

Lim, B., Arik, S.O., Loeff, N., and Pfister, T. (2020) 'Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting', available: https://doi.org/10.48550/arXiv.1912.09363.

Lim, B. and Zohren, S. (2021) 'Time-series forecasting with deep learning: a survey', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194), 20200209, available: https://doi.org/10.1098/rsta.2020.0209.

Liu, Y. (2023) 'Impact, Responses and Future Prediction of Climate Change on the Phenology of Jellyfish', *Theoretical and Natural Science*, 3, 658–665, available: https://doi.org/10.54254/2753-8818/3/20220413.

Longhui, A.C. (2024) 'Transformer Model in Time-Series Analysis with tsfresh, Mann-Whitney Test and Benjamini-Yekutieli…', *Medium*, available: https://medium.com/@adamchen564/transformer-model-in-time-series-analysis-with-tsfresh-mann-whitney-test-and-benjamini-yekutieli-a1812a9648dc [accessed 26 Aug 2024].

Lynam, C.P., Gibbons, M.J., Axelsen, B.E., Sparks, C.A.J., Coetzee, J., Heywood, B.G., and Brierley, A.S. (2006) 'Jellyfish overtake fish in a heavily fished ecosystem', *Current Biology*, 16(13), R492–R493, available: https://doi.org/10.1016/j.cub.2006.06.018.

Lynam, C.P., Lilley, M.K.S., Bastian, T., Doyle, T.K., Beggs, S.E., and Hays, G.C. (2011) 'Have jellyfish in the Irish Sea benefited from climate change and overfishing?', *Global Change Biology*, 17(2), 767–782, available: https://doi.org/10.1111/j.1365-2486.2010.02352.x.

Marine Conservation Society (2024) 'UK Jellyfish Sightings from 2003 to 2022 | Occurrence records | NBN Atlas', available: https://records.nbnatlas.org/occurrences/search?q=data_resource_uid%3Adr1138&nbn_loading=true&fq=-occurrence_status%3A%22absent%22 [accessed 30 Jun 2024].

McCarthy, G.D., Gleeson, E., and Walsh, S. (2015) 'The influence of ocean variations on the climate of Ireland', *Weather*, 70(8), 242–245, available: https://doi.org/10.1002/wea.2543.

Met Office (2024) 'UK and regional series', available: https://www.metoffice.gov.uk/research/climate/maps-and-data/uk-and-regional-series [accessed 22 Jul 2024].

Miles, I. (1979) ‘The Development of Forecasting: Towards a History of the Future’, in Whiston, T., ed., *The Uses and Abuses of Forecasting*, London: Palgrave Macmillan UK, 5–41, available: https://doi.org/10.1007/978-1-349-04486-3_2.

Møller, L.F. and Riisgård, H.U. (2007) ‘Impact of jellyfish and mussels on algal blooms caused by seasonal oxygen depletion and nutrient release from the sediment in a Danish fjord’, *Journal of Experimental Marine Biology and Ecology*, 351(1), 92–105, available: https://doi.org/10.1016/j.jembe.2007.06.026.

Nilsson, N.J. (1996) *Introduction to Machine Learning. An Early Draft of a Proposed Textbook* [online], Stanford University, available: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=PJguPTEAAAAJ&citation_for_view=PJguPTEAAAAJ:35N4QoGY0k4C [accessed 21 Nov 2021].

O’Sullivan, K. (2020) ‘Climate predictions for Ireland by 2050 come with unprecedented detail’, *The Irish Times*, 18 Sep, available: https://www.irishtimes.com/news/environment/climate-predictions-for-ireland-by-2050-come-with-unprecedented-detail-1.4357727 [accessed 30 Jun 2024].

Paroha, A.D. and Chotrani, A. (2023) ‘A Comparative Analysis of TimeGPT and Time-LLM in Predicting ESP Maintenance Needs in the Oil and Gas Sector’, *International Journal of Computer Applications*, 186.

Peixeiro, M. (2024) iTransformer: The Latest Breakthrough in Time Series Forecasting [online], *Medium*, available: https://towardsdatascience.com/itransformer-the-latest-breakthrough-in-time-series-forecasting-d538ddc6c5d1 [accessed 28 May 2024].

Rodhe, H., Charlson, R., and Crawford, E. (1997) ‘Svante Arrhenius and the Greenhouse Effect’, *Ambio*, 26(1), 2–5.

Sam, E. (2023) ‘Choosing the Right Forecasting Model: Time Series vs. Machine Learning vs. Deep Learning’, *Medium*, available: https://medium.com/@samaelizabeth/choosing-the-right-forecasting-model-time-series-vs-machine-learning-vs-deep-learning-6ab319ed4fcd [accessed 30 Jun 2024].

Schmidt, R.M. (2019) ‘Recurrent Neural Networks (RNNs): A gentle Introduction and Overview’, available: https://doi.org/10.48550/arXiv.1912.05911.

Shakouri, B., Yazdi, S. khoshnevis, and Fashandi, A. (2010) ‘Overfishing’, in *2010 2nd International Conference on Chemical, Biological and Environmental Engineering*, Presented at the 2010 2nd International Conference on Chemical, Biological and Environmental Engineering, 229–234, available: https://doi.org/10.1109/ICBEE.2010.5649533.

Sharifani, K. and Amini, M. (2023) ‘Machine Learning and Deep Learning: A Review of Methods and Applications’, available: https://papers.ssrn.com/abstract=4458723 [accessed 1 Jul 2024].

Sherstinsky, A. (2020) ‘Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network’, *Physica D: Nonlinear Phenomena*, 404, 132306, available: https://doi.org/10.1016/j.physd.2019.132306.

Staudemeyer, R.C. and Morris, E.R. (2019) 'Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks', available: https://doi.org/10.48550/arXiv.1909.09586.

Stern, D.I. and Kaufmann, R.K. (2014) 'Anthropogenic and natural causes of climate change', *Climatic Change*, 122(1), 257–269, available: https://doi.org/10.1007/s10584-013-1007-x.

Tay, C., Lindsey, E.O., Chin, S.T., McCaughey, J.W., Bekaert, D., Nguyen, M., Hua, H., Manipon, G., Karim, M., Horton, B.P., Li, T., and Hill, E.M. (2022) 'Sea-level rise from land subsidence in major coastal cities', *Nature Sustainability*, 5(12), 1049–1057, available: https://doi.org/10.1038/s41893-022-00947-z.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. ukasz, and Polosukhin, I. (2017) 'Attention is All you Need', in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html [accessed 12 Jul 2024].

Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., and Sun, L. (2023) 'Transformers in Time Series: A Survey', available: https://doi.org/10.48550/arXiv.2202.07125.

Wright, L.G., Onodera, T., Stein, M.M., Wang, T., Schachter, D.T., Hu, Z., and McMahon, P.L. (2022) 'Deep physical neural networks trained with backpropagation', *Nature*, 601(7894), 549–555, available: https://doi.org/10.1038/s41586-021-04223-6.

Zarkadakis, G. (2022) 'The Internet Is Dead: Long Live the Internet', in Werthner, H., Prem, E., Lee, E.A. and Ghezzi, C., eds., *Perspectives on Digital Humanism*, Cham: Springer International Publishing, 47–52, available: https://doi.org/10.1007/978-3-030-86144-5_7.

Zeyer, A., Bahar, P., Irie, K., Schlüter, R., and Ney, H. (2019) 'A Comparison of Transformer and LSTM Encoder Decoder Models for ASR', in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Presented at the 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), 8–15, available: https://doi.org/10.1109/ASRU46091.2019.9004025.

Zhang, C. and Lu, Y. (2021) 'Study on artificial intelligence: The state of the art and future prospects', *Journal of Industrial Information Integration*, 23, 100224, available: https://doi.org/10.1016/j.jii.2021.100224.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., and Zhang, W. (2021) 'Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting', available: https://doi.org/10.48550/arXiv.2012.07436.

Zimmer, M. (2009) 'GFP: from jellyfish to the Nobel prize and beyond', *Chemical Society Reviews*, 38(10), 2823–2832, available: https://doi.org/10.1039/B904023D.

Appendix B: Excess Results

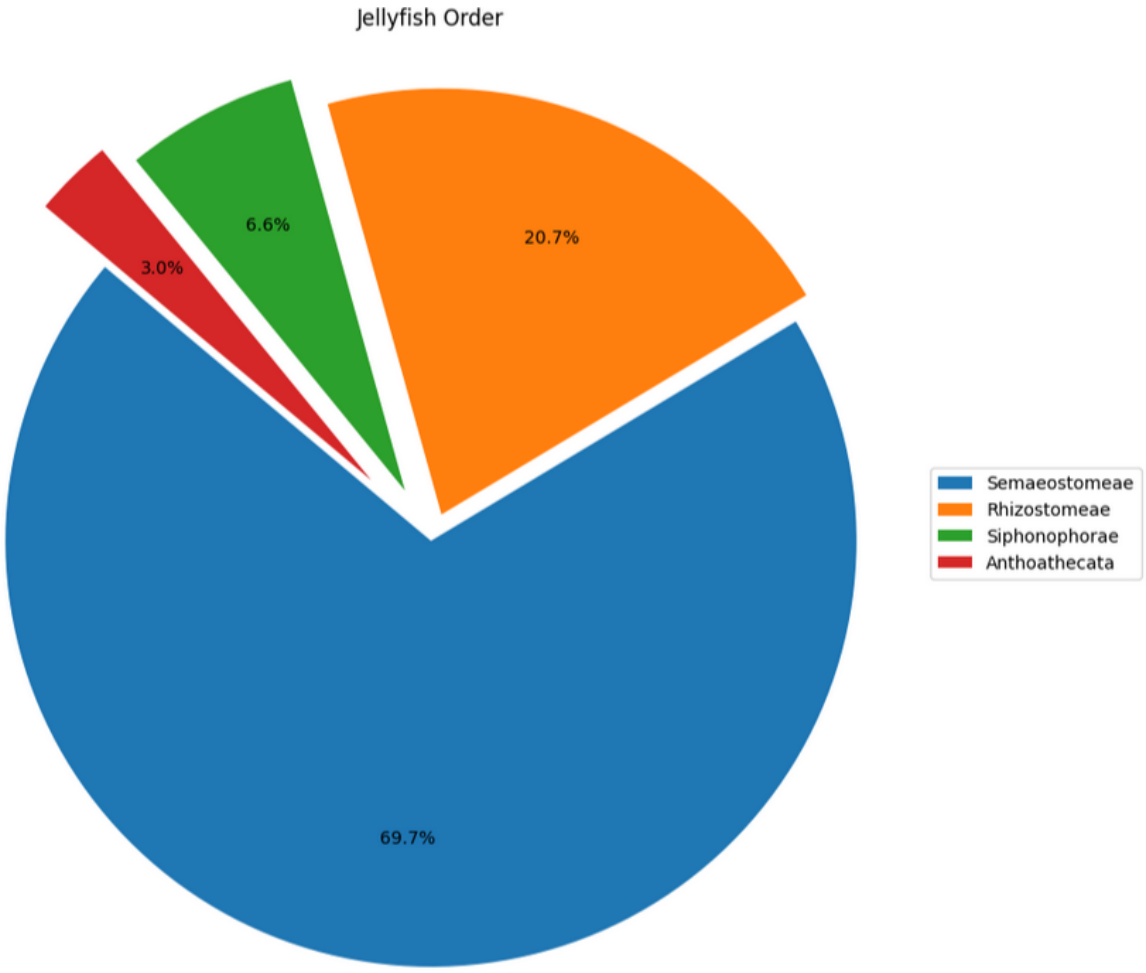


Figure 44: Jellyfish Species Order Pie Chart

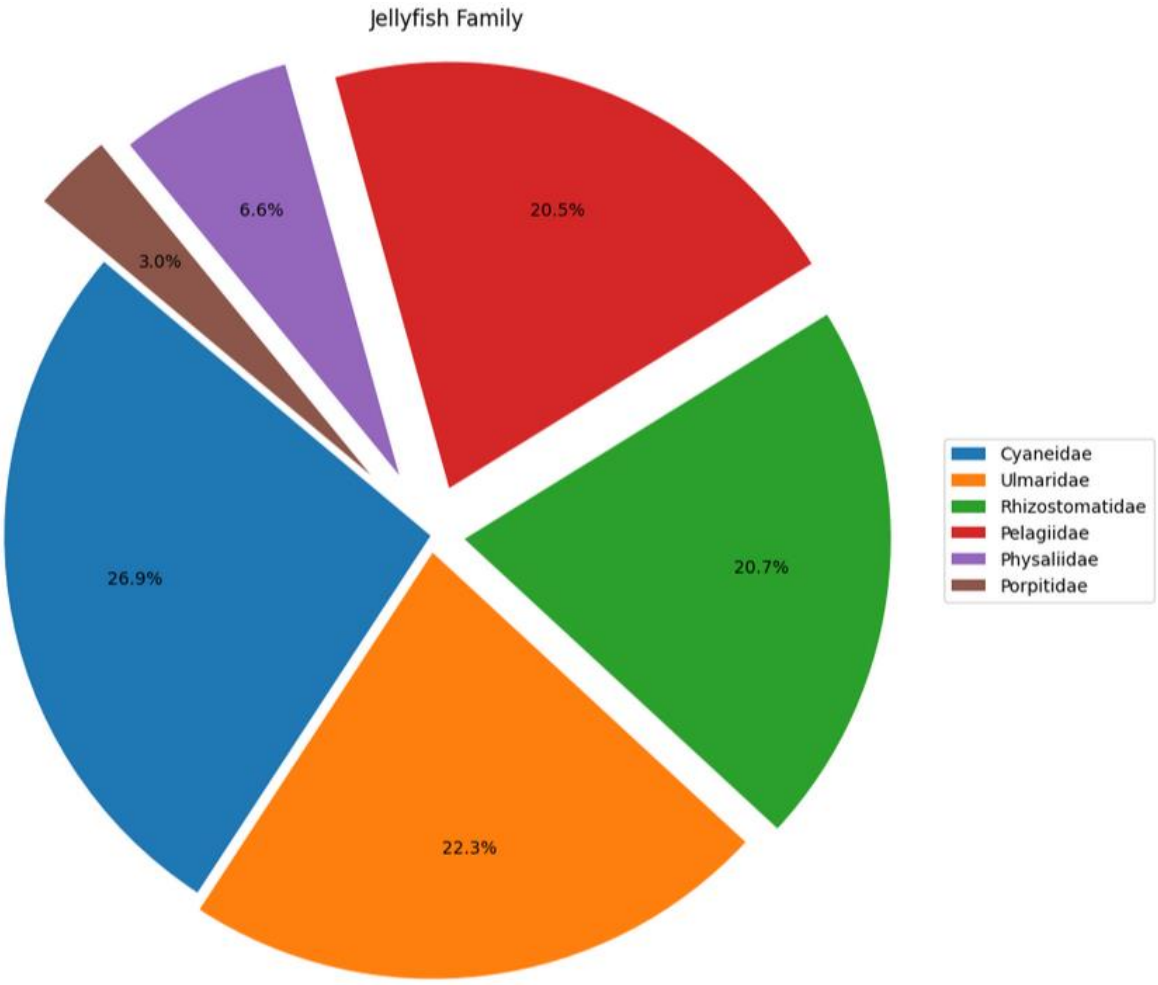


Figure 45: Jellyfish Family Pie Chart

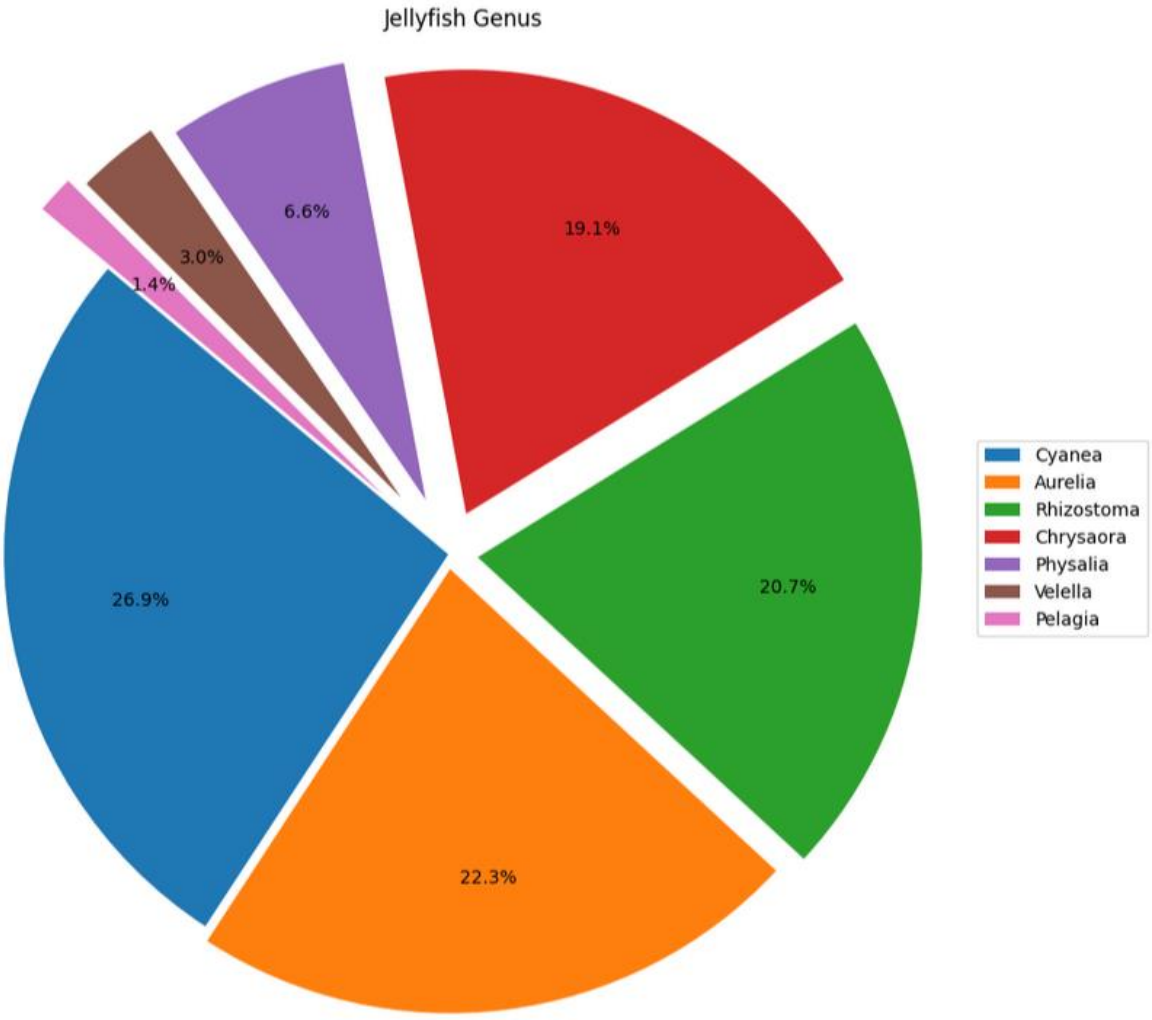


Figure 46: Jellyfish Genus Pie Chart

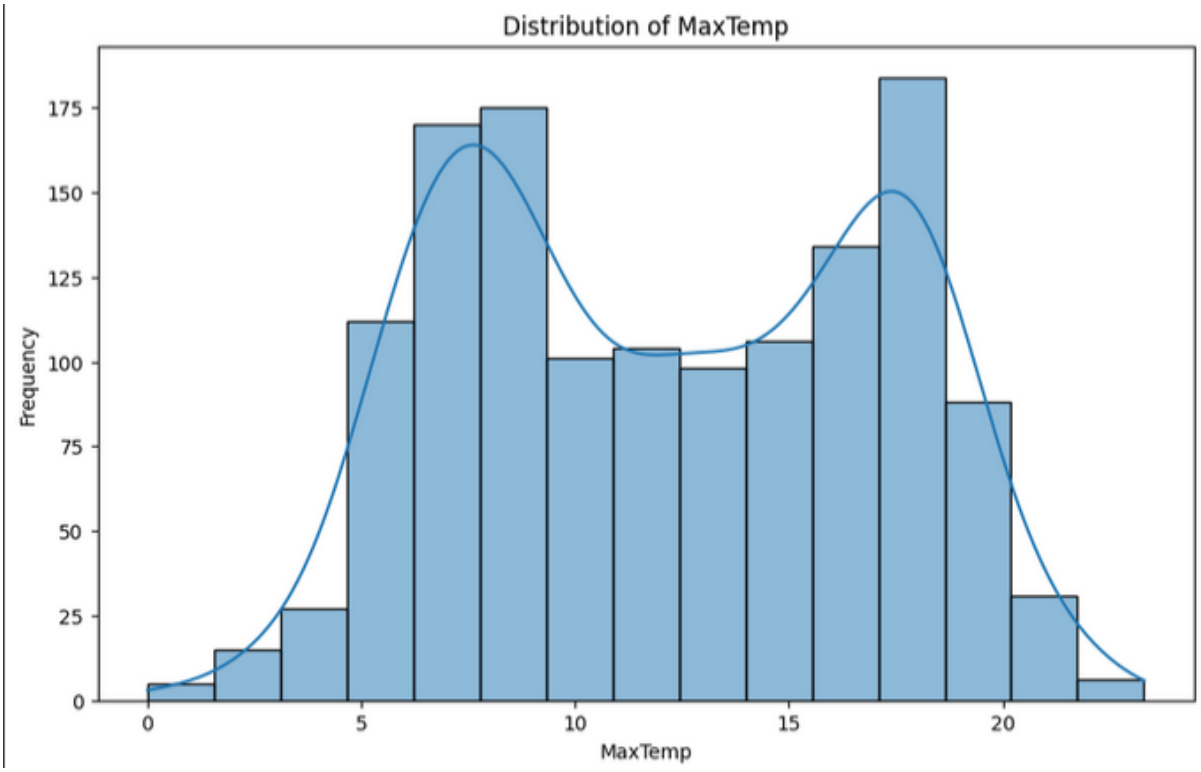


Figure 47: Max Temperature Histogram Distribution

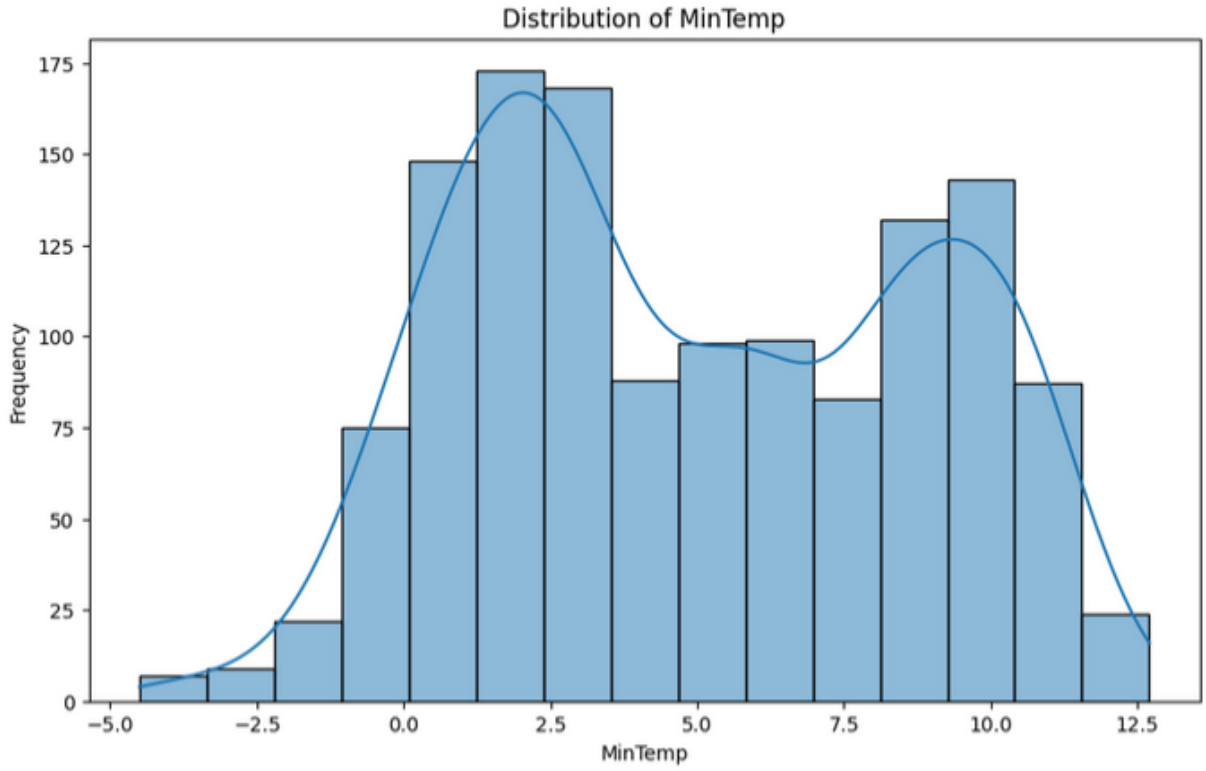


Figure 48: Min Temperature Histogram Distribution

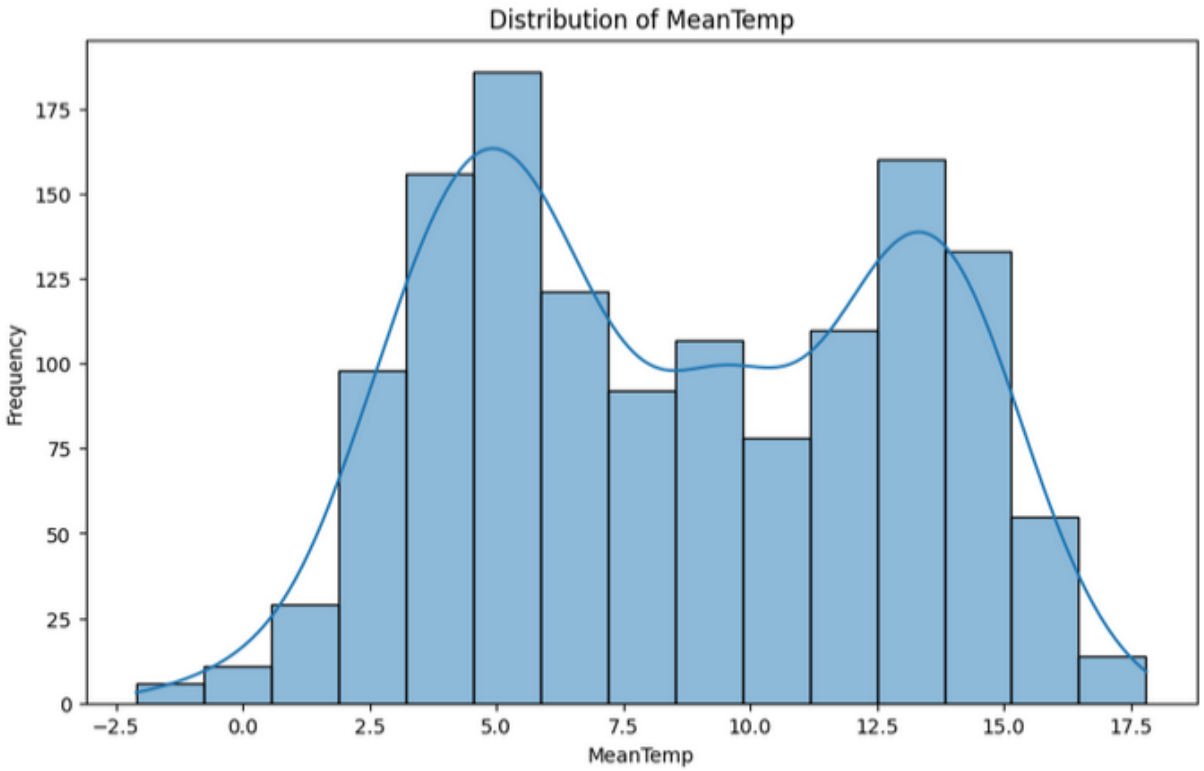


Figure 49: Mean Temperature Histogram Distribution

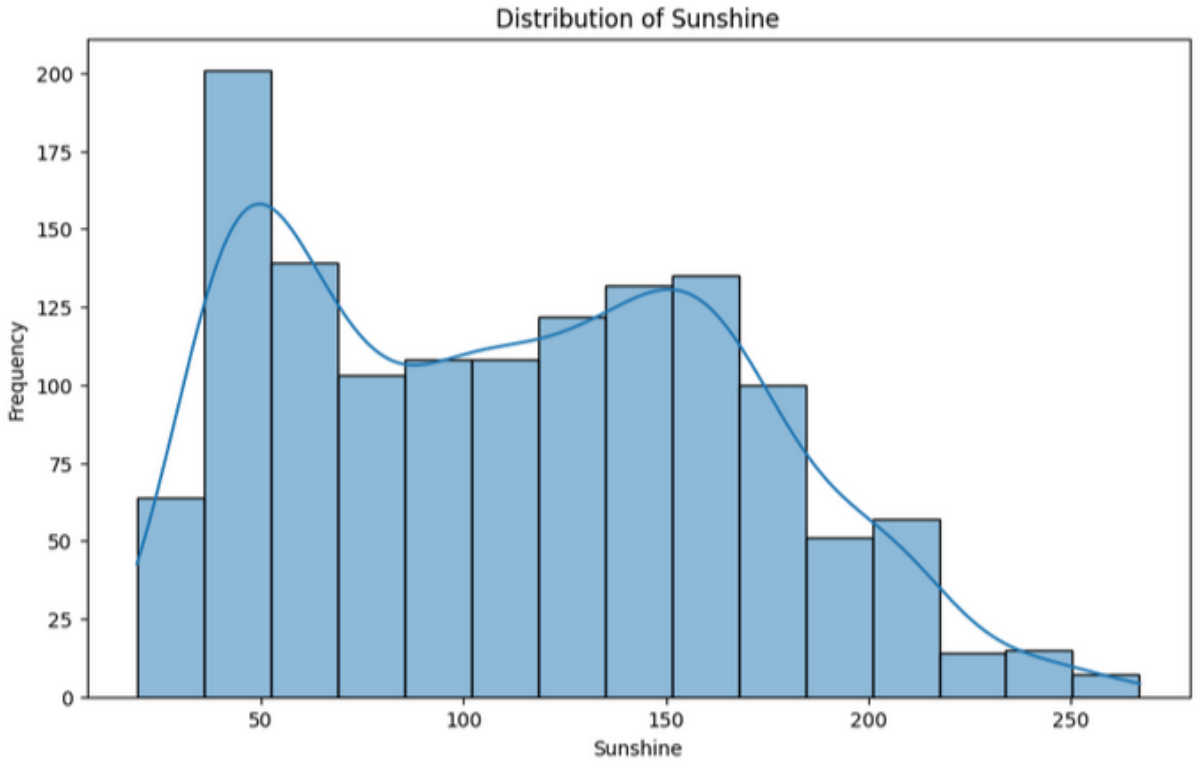


Figure 50: Sunshine Histogram Distribution

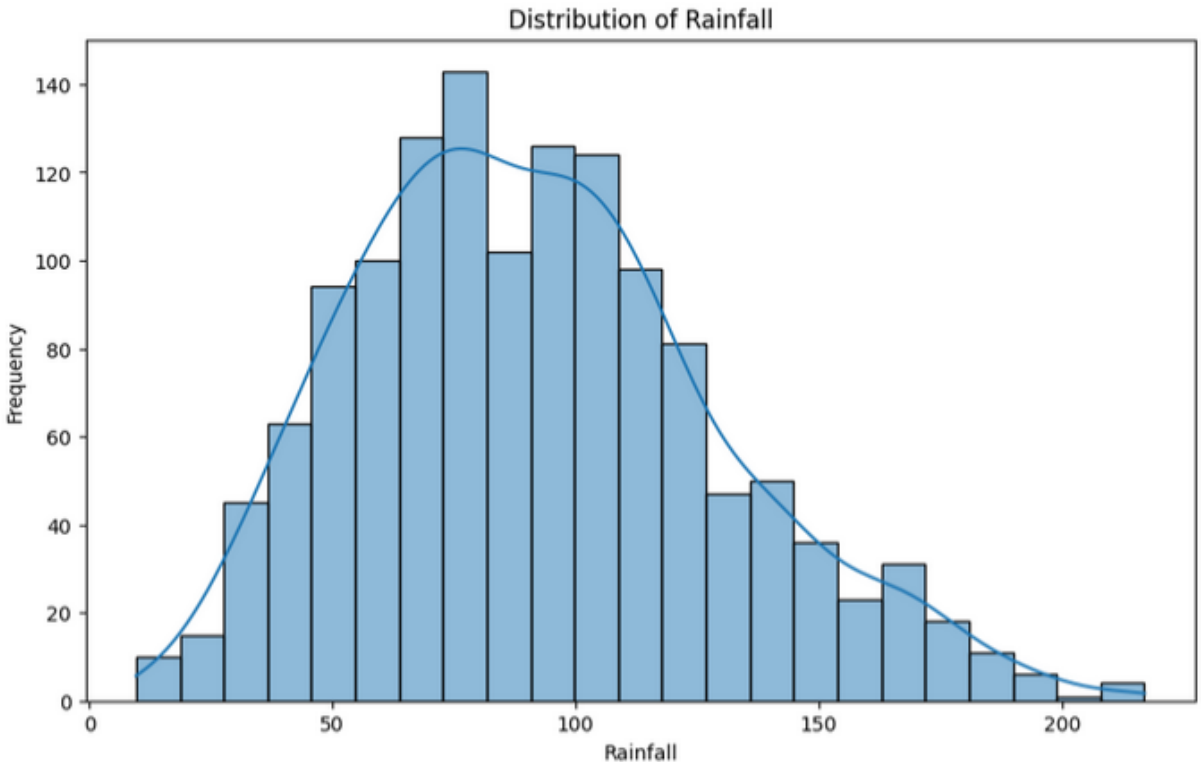


Figure 51: Rainfall Histogram Distribution

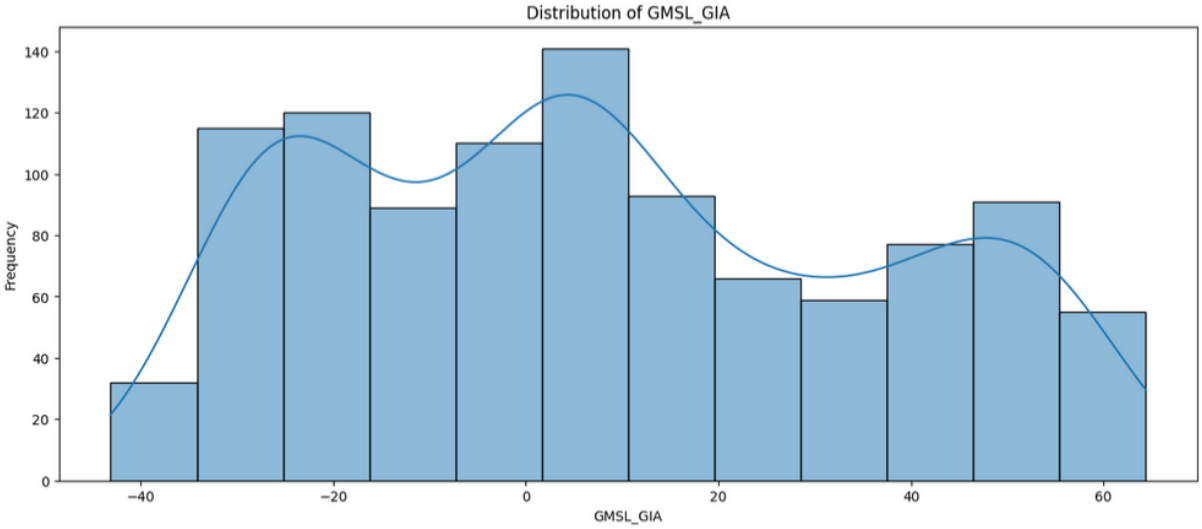


Figure 52: Sea Level Histogram Distribution

Stationarity of MaxTemp

	adf	kpss
Test Statistic	-3.807549	0.339651
p-value	0.002831	0.1
Numbers of lags	24	1
decision	Stationary	Stationary
Critical Value (1%)	-3.435273	0.739
Critical Value (5%)	-2.863714	0.463
Critical Value (10%)	-2.567927	0.347
Critical Value (2.5%)	NaN	0.574

Stationarity of MinTemp

	adf	kpss
Test Statistic	-4.049892	0.388691
p-value	0.001171	0.08203
Numbers of lags	24	2
decision	Stationary	Stationary
Critical Value (1%)	-3.435273	0.739
Critical Value (5%)	-2.863714	0.463
Critical Value (10%)	-2.567927	0.347
Critical Value (2.5%)	NaN	0.574

Stationarity of MeanTemp

	adf	kpss
Test Statistic	-3.473476	0.427171
p-value	0.008695	0.065444
Numbers of lags	24	1
decision	Stationary	Stationary
Critical Value (1%)	-3.435273	0.739
Critical Value (5%)	-2.863714	0.463
Critical Value (10%)	-2.567927	0.347
Critical Value (2.5%)	NaN	0.574

Stationarity of Sunshine

	adf	kpss
Test Statistic	-5.984221	0.212095
p-value	0.0	0.1
Numbers of lags	23	1
decision	Stationary	Stationary
Critical Value (1%)	-3.435269	0.739
Critical Value (5%)	-2.863712	0.463
Critical Value (10%)	-2.567927	0.347
Critical Value (2.5%)	NaN	0.574

Stationarity of Rainfall

	adf	kpss
Test Statistic	-6.195948	0.575446
p-value	0.0	0.024869
Numbers of lags	24	3
decision	Stationary	Non-Stationary
Critical Value (1%)	-3.435273	0.739
Critical Value (5%)	-2.863714	0.463
Critical Value (10%)	-2.567927	0.347
Critical Value (2.5%)	NaN	0.574

Figure 53: Climate Stationarity Tests

	adf	kpss
Test Statistic	-0.812664	5.207346
p-value	0.815374	0.01
Numbers of lags	14	19
decision	Non-Stationary	Non-Stationary
Critical Value (1%)	-3.436696	0.739
Critical Value (5%)	-2.864342	0.463
Critical Value (10%)	-2.568262	0.347
Critical Value (2.5%)	NaN	0.574

Figure 54: Sea Level Stationarity Test

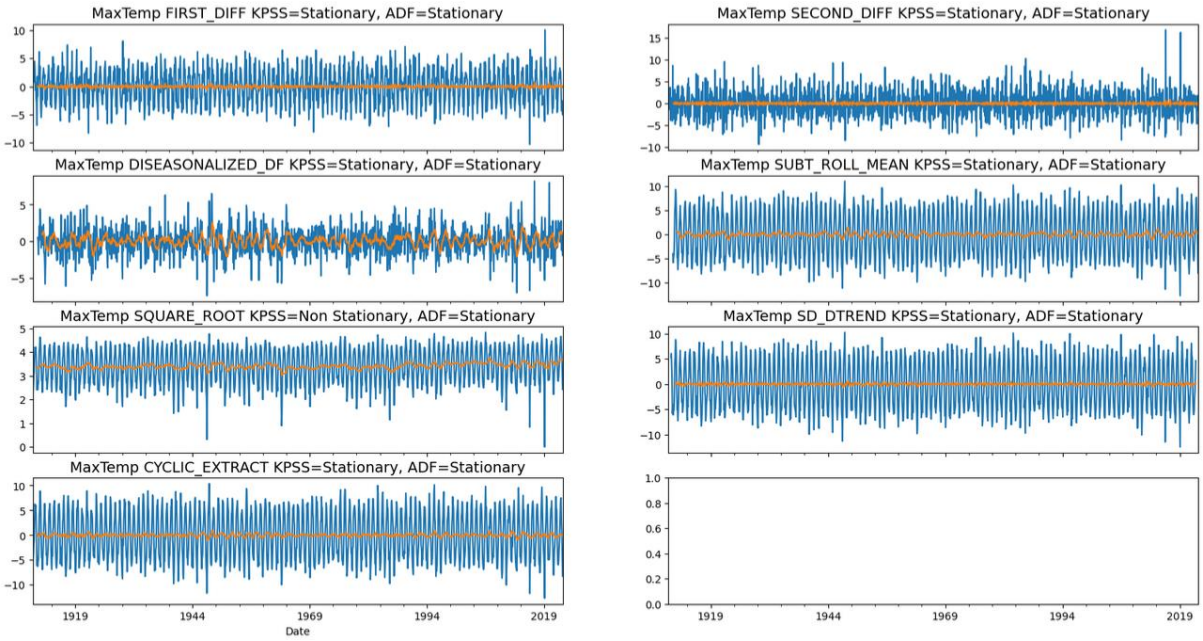


Figure 55: Max Temperature Differencing Transformations and Stationarity Tests

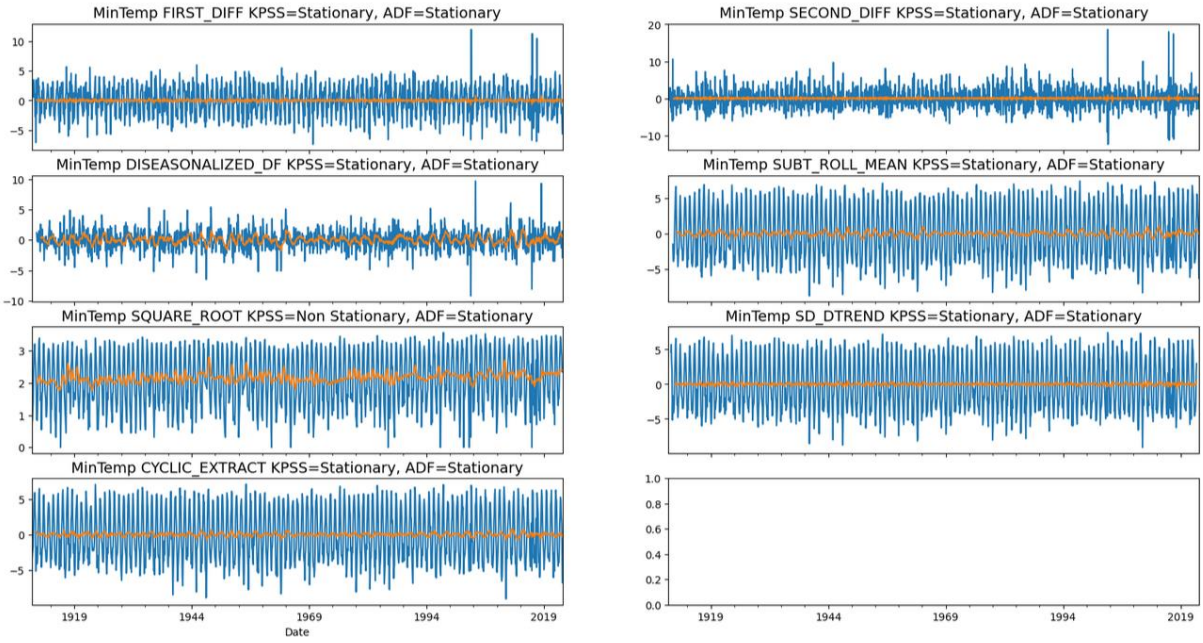


Figure 56: Min Temperature Differencing Transformations and Stationarity Tests

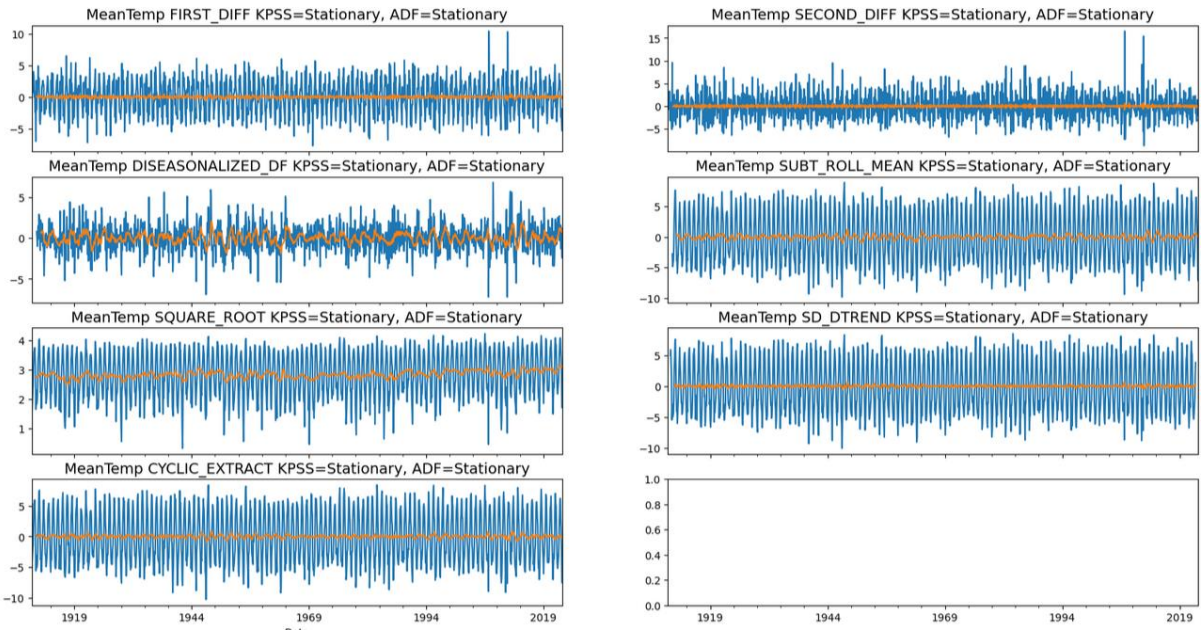


Figure 57: Mean Temperature Differencing Transformations and Stationarity Tests

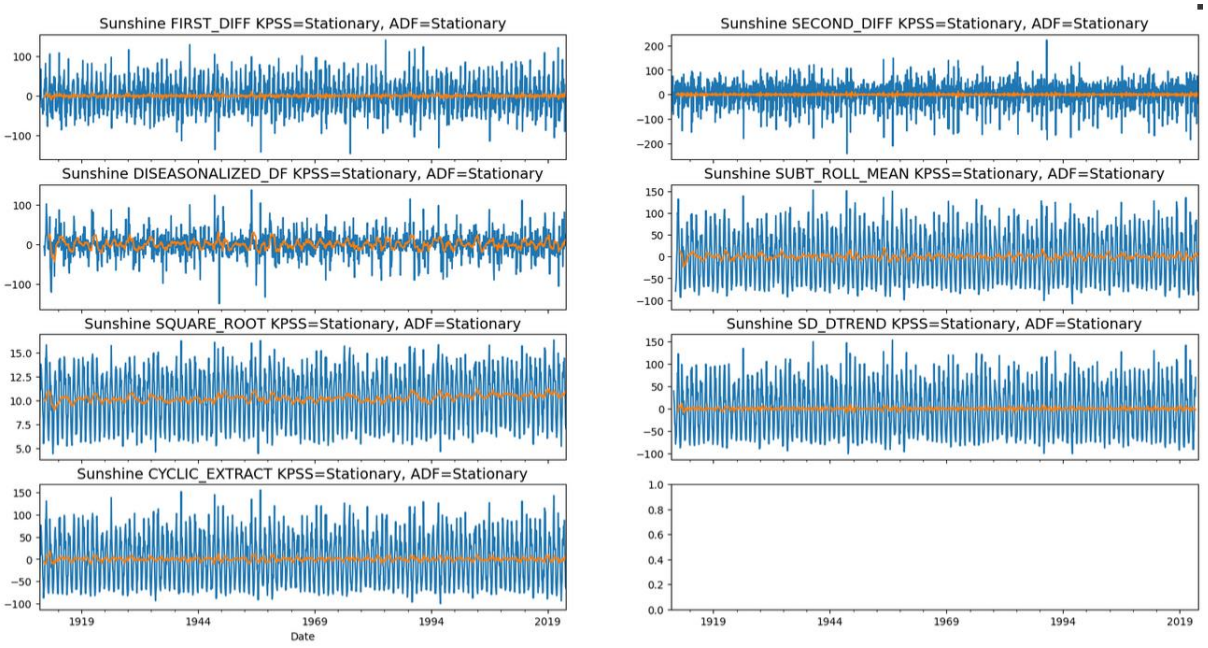


Figure 58: Sunshine Differencing Transformations and Stationarity Tests

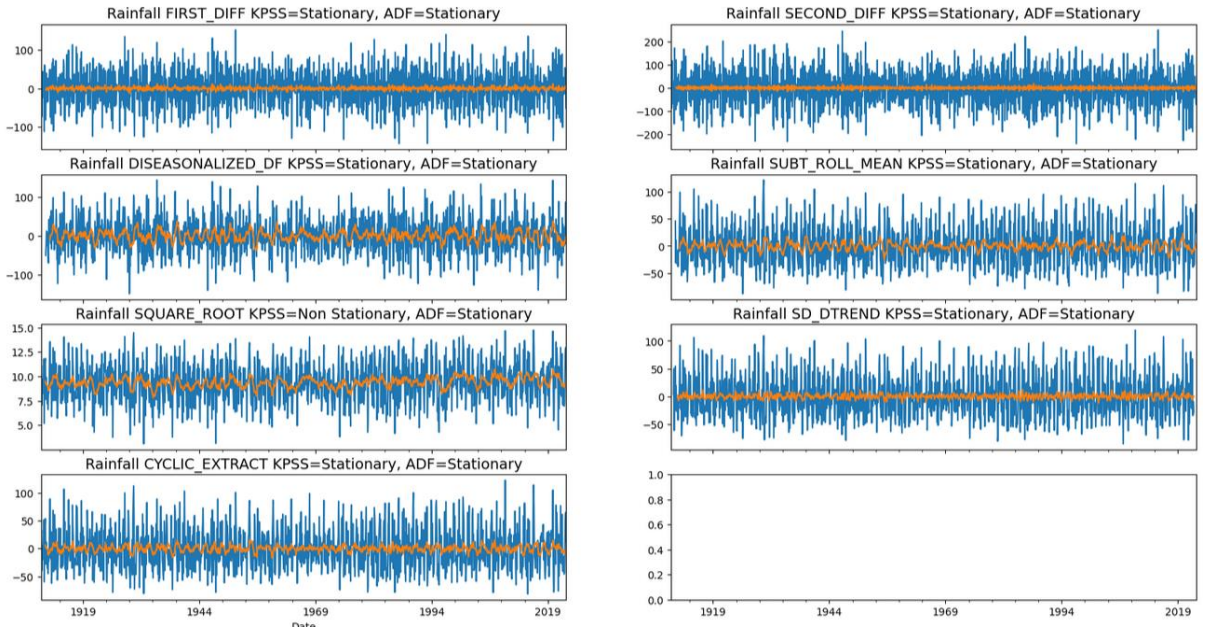


Figure 59: Rainfall Differencing Transformations and Stationarity Tests

MinTemp

OLS Regression Results

Dep. Variable:	y	R-squared:	0.738
Model:	OLS	Adj. R-squared:	0.733
Method:	Least Squares	F-statistic:	146.9
Date:	Wed, 21 Aug 2024	Prob (F-statistic):	0.00
Time:	11:03:24	Log-Likelihood:	-2198.0
No. Observations:	1331	AIC:	4448.
Df Residuals:	1305	BIC:	4583.
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.3254	0.080	-4.050	0.000	-0.483	-0.168
x2	-0.4101	0.082	-5.012	0.000	-0.571	-0.250
x3	-0.3745	0.081	-4.613	0.000	-0.534	-0.215
x4	-0.3795	0.080	-4.748	0.000	-0.536	-0.223
x5	-0.4207	0.078	-5.367	0.000	-0.575	-0.267
x6	-0.4482	0.077	-5.811	0.000	-0.599	-0.297
x7	-0.4694	0.077	-6.129	0.000	-0.620	-0.319
x8	-0.4527	0.076	-5.948	0.000	-0.602	-0.303
x9	-0.4708	0.076	-6.234	0.000	-0.619	-0.323
x10	-0.4551	0.075	-6.070	0.000	-0.602	-0.308
x11	-0.4245	0.075	-5.694	0.000	-0.571	-0.278
x12	-0.3222	0.074	-4.329	0.000	-0.468	-0.176
x13	-0.1816	0.074	-2.455	0.014	-0.327	-0.037
x14	-0.0823	0.073	-1.133	0.258	-0.225	0.060
x15	-0.0985	0.070	-1.404	0.161	-0.236	0.039
x16	-0.1912	0.067	-2.860	0.004	-0.322	-0.060
x17	-0.2228	0.063	-3.526	0.000	-0.347	-0.099
x18	-0.2215	0.059	-3.738	0.000	-0.338	-0.105
x19	-0.2254	0.055	-4.074	0.000	-0.334	-0.117
x20	-0.2218	0.051	-4.358	0.000	-0.322	-0.122
x21	-0.3207	0.046	-6.925	0.000	-0.412	-0.230
x22	-0.3427	0.042	-8.123	0.000	-0.425	-0.260
x23	-0.2935	0.038	-7.650	0.000	-0.369	-0.218
x24	-0.2162	0.034	-6.354	0.000	-0.283	-0.149
x25	-0.0766	0.028	-2.769	0.006	-0.131	-0.022
const	1.6164	0.398	4.059	0.000	0.835	2.398

Omnibus:	236.014	Durbin-Watson:	1.999
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1091.309
Skew:	-0.758	Prob(JB):	1.06e-237
Kurtosis:	7.169	Cond. No.	108.

Figure 60: Min Temperature OLS Regression Summary

MeanTemp

OLS Regression Results

Dep. Variable:	y	R-squared:	0.779
Model:	OLS	Adj. R-squared:	0.775
Method:	Least Squares	F-statistic:	184.4
Date:	Wed, 21 Aug 2024	Prob (F-statistic):	0.00
Time:	11:03:24	Log-Likelihood:	-2177.6
No. Observations:	1331	AIC:	4407.
Df Residuals:	1305	BIC:	4542.
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.2463	0.071	-3.473	0.001	-0.385	-0.107
x2	-0.4275	0.073	-5.827	0.000	-0.571	-0.284
x3	-0.4059	0.073	-5.552	0.000	-0.549	-0.262
x4	-0.3995	0.072	-5.522	0.000	-0.541	-0.258
x5	-0.4369	0.071	-6.134	0.000	-0.577	-0.297
x6	-0.4847	0.070	-6.902	0.000	-0.622	-0.347
x7	-0.4972	0.070	-7.097	0.000	-0.635	-0.360
x8	-0.5021	0.070	-7.191	0.000	-0.639	-0.365
x9	-0.5060	0.070	-7.276	0.000	-0.642	-0.370
x10	-0.5122	0.069	-7.396	0.000	-0.648	-0.376
x11	-0.4482	0.069	-6.489	0.000	-0.584	-0.313
x12	-0.3562	0.069	-5.155	0.000	-0.492	-0.221
x13	-0.2536	0.069	-3.687	0.000	-0.389	-0.119
x14	-0.1419	0.068	-2.100	0.036	-0.274	-0.009
x15	-0.1641	0.065	-2.514	0.012	-0.292	-0.036
x16	-0.2673	0.062	-4.292	0.000	-0.390	-0.145
x17	-0.2754	0.059	-4.685	0.000	-0.391	-0.160
x18	-0.2753	0.055	-4.987	0.000	-0.384	-0.167
x19	-0.2686	0.052	-5.215	0.000	-0.370	-0.168
x20	-0.2481	0.048	-5.221	0.000	-0.341	-0.155
x21	-0.3432	0.043	-7.922	0.000	-0.428	-0.258
x22	-0.3412	0.040	-8.549	0.000	-0.420	-0.263
x23	-0.2915	0.037	-7.940	0.000	-0.364	-0.219
x24	-0.2152	0.033	-6.541	0.000	-0.280	-0.151
x25	-0.0894	0.028	-3.245	0.001	-0.143	-0.035
const	2.1106	0.606	3.485	0.001	0.923	3.299

Omnibus:	154.809	Durbin-Watson:	1.996
Prob(Omnibus):	0.000	Jarque-Bera (JB):	465.958
Skew:	-0.589	Prob(JB):	6.58e-102
Kurtosis:	5.648	Cond. No.	196.

Figure 61: Mean Temp OLS Regression Summary

Sunshine

OLS Regression Results

Dep. Variable:	y	R-squared:	0.678
Model:	OLS	Adj. R-squared:	0.672
Method:	Least Squares	F-statistic:	114.5
Date:	Wed, 21 Aug 2024	Prob (F-statistic):	1.80e-300
Time:	11:03:24	Log-Likelihood:	-6046.0
No. Observations:	1332	AIC:	1.214e+04
Df Residuals:	1307	BIC:	1.227e+04
Df Model:	24		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.6371	0.106	-5.984	0.000	-0.846	-0.428
x2	-0.1945	0.105	-1.850	0.064	-0.401	0.012
x3	-0.1242	0.103	-1.211	0.226	-0.325	0.077
x4	-0.1584	0.099	-1.594	0.111	-0.353	0.037
x5	-0.1758	0.096	-1.825	0.068	-0.365	0.013
x6	-0.1887	0.094	-2.011	0.045	-0.373	-0.005
x7	-0.2271	0.091	-2.485	0.013	-0.406	-0.048
x8	-0.2812	0.089	-3.149	0.002	-0.456	-0.106
x9	-0.2780	0.088	-3.174	0.002	-0.450	-0.106
x10	-0.2667	0.086	-3.092	0.002	-0.436	-0.098
x11	-0.2823	0.085	-3.332	0.001	-0.448	-0.116
x12	-0.1924	0.083	-2.306	0.021	-0.356	-0.029
x13	-0.0243	0.081	-0.298	0.766	-0.184	0.136
x14	0.0480	0.078	0.614	0.539	-0.105	0.201
x15	0.0100	0.074	0.135	0.893	-0.135	0.155
x16	0.0292	0.070	0.420	0.674	-0.107	0.166
x17	-0.0326	0.065	-0.502	0.616	-0.160	0.095
x18	-0.0870	0.060	-1.456	0.146	-0.204	0.030
x19	-0.1376	0.055	-2.507	0.012	-0.245	-0.030
x20	-0.1620	0.050	-3.236	0.001	-0.260	-0.064
x21	-0.2456	0.045	-5.439	0.000	-0.334	-0.157
x22	-0.2753	0.040	-6.855	0.000	-0.354	-0.197
x23	-0.2148	0.035	-6.087	0.000	-0.284	-0.146
x24	-0.0656	0.028	-2.374	0.018	-0.120	-0.011
const	71.6825	11.992	5.977	0.000	48.156	95.209

Omnibus:	120.428	Durbin-Watson:	2.003
Prob(Omnibus):	0.000	Jarque-Bera (JB):	318.423
Skew:	0.491	Prob(JB):	7.17e-70
Kurtosis:	5.184	Cond. No.	2.58e+03

Figure 62: Sunshine OLS Regression Model Summary

Rainfall

OLS Regression Results

Dep. Variable:	y	R-squared:	0.484
Model:	OLS	Adj. R-squared:	0.474
Method:	Least Squares	F-statistic:	48.93
Date:	Wed, 21 Aug 2024	Prob (F-statistic):	1.82e-167
Time:	11:03:25	Log-Likelihood:	-6566.4
No. Observations:	1331	AIC:	1.318e+04
Df Residuals:	1305	BIC:	1.332e+04
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.8519	0.137	-6.196	0.000	-1.122	-0.582
x2	-0.0561	0.135	-0.415	0.678	-0.321	0.209
x3	-0.0430	0.132	-0.326	0.744	-0.302	0.216
x4	-0.1154	0.129	-0.896	0.370	-0.368	0.137
x5	-0.1100	0.125	-0.879	0.380	-0.356	0.136
x6	-0.1605	0.121	-1.323	0.186	-0.399	0.077
x7	-0.2284	0.118	-1.941	0.053	-0.459	0.002
x8	-0.2356	0.114	-2.061	0.040	-0.460	-0.011
x9	-0.2605	0.111	-2.341	0.019	-0.479	-0.042
x10	-0.2469	0.109	-2.273	0.023	-0.460	-0.034
x11	-0.2532	0.106	-2.385	0.017	-0.462	-0.045
x12	-0.2271	0.103	-2.194	0.028	-0.430	-0.024
x13	-0.0924	0.101	-0.919	0.358	-0.290	0.105
x14	-0.0659	0.097	-0.679	0.497	-0.256	0.125
x15	-0.0430	0.092	-0.464	0.642	-0.224	0.138
x16	-0.0080	0.087	-0.092	0.927	-0.179	0.163
x17	-0.0596	0.082	-0.728	0.467	-0.220	0.101
x18	-0.1251	0.076	-1.649	0.099	-0.274	0.024
x19	-0.1780	0.070	-2.550	0.011	-0.315	-0.041
x20	-0.2279	0.063	-3.595	0.000	-0.352	-0.104
x21	-0.2710	0.057	-4.743	0.000	-0.383	-0.159
x22	-0.2452	0.051	-4.790	0.000	-0.346	-0.145
x23	-0.1875	0.044	-4.241	0.000	-0.274	-0.101
x24	-0.1657	0.037	-4.482	0.000	-0.238	-0.093
x25	-0.1012	0.028	-3.669	0.000	-0.155	-0.047
const	78.3082	12.669	6.181	0.000	53.455	103.161

Omnibus:	15.126	Durbin-Watson:	2.004
Prob(Omnibus):	0.001	Jarque-Bera (JB):	15.426
Skew:	0.264	Prob(JB):	0.000447
Kurtosis:	3.016	Cond. No.	1.37e+03

Figure 63: Rainfall OSL Regression Model Summary

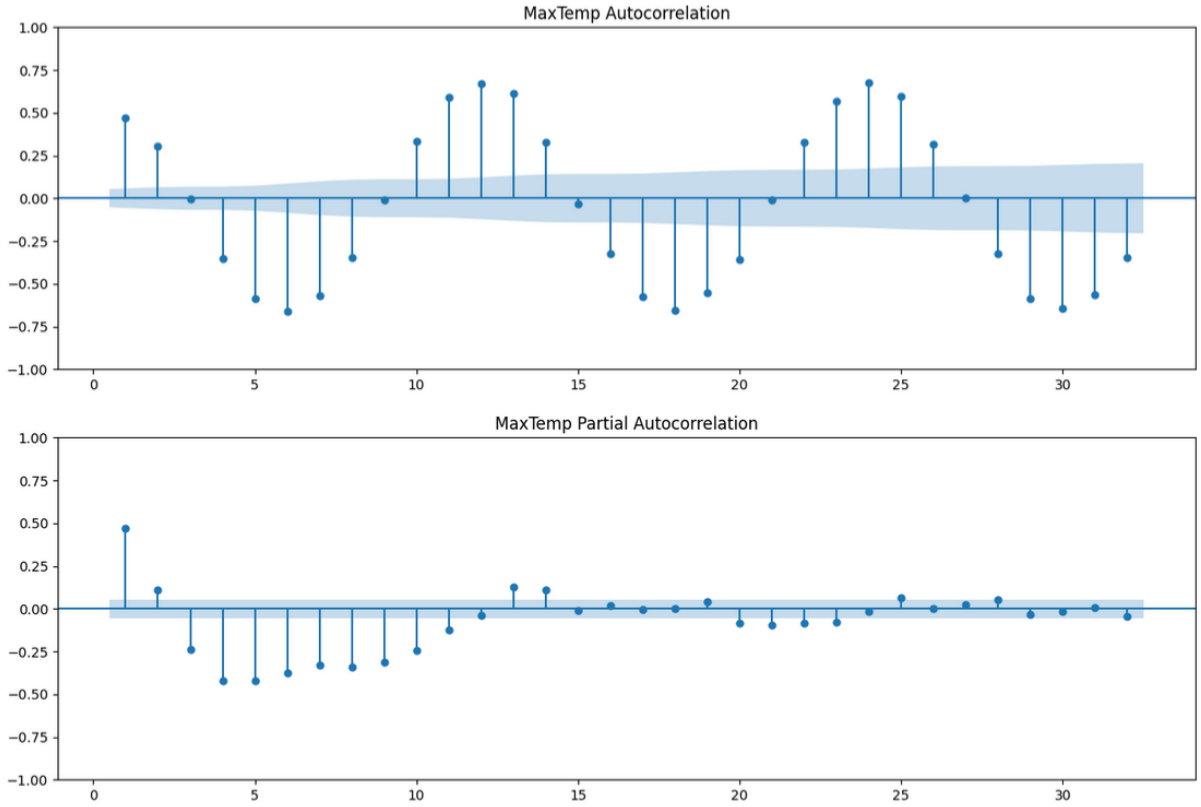


Figure 64: Max Temperature Autocorrelation and Partial Autocorrelation Plot

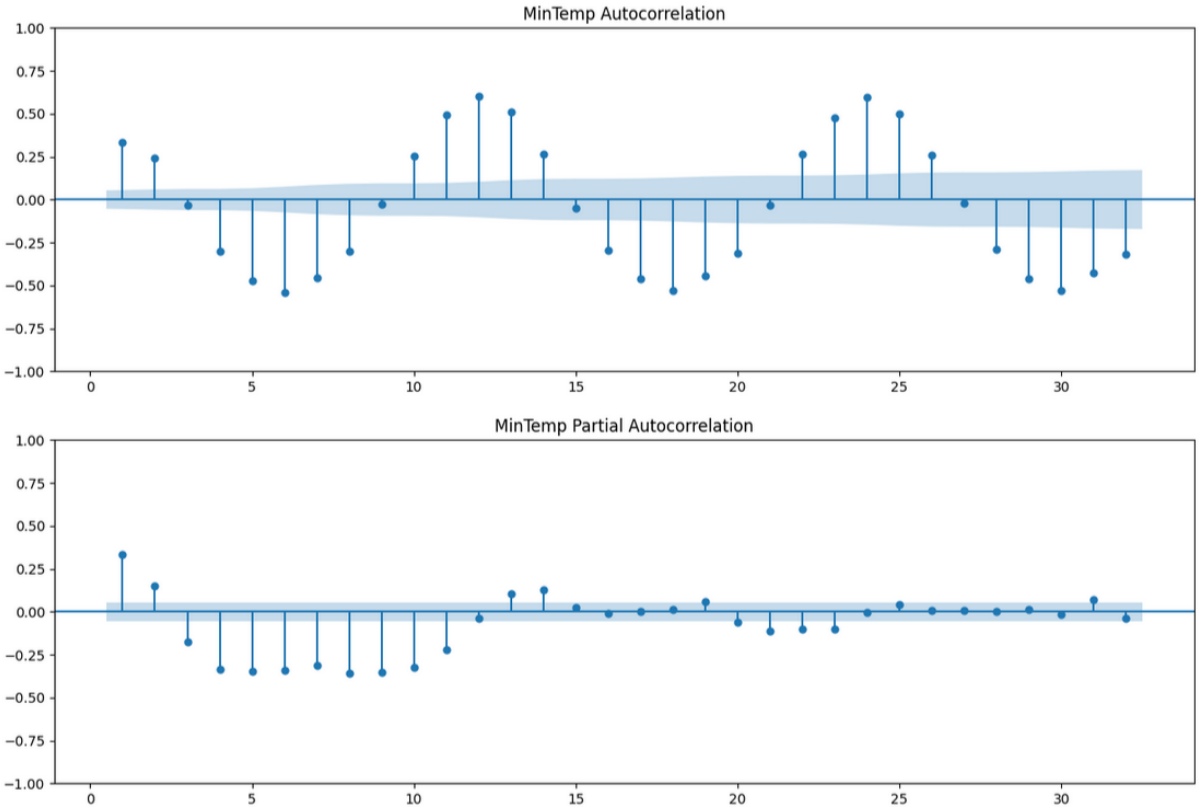


Figure 65: Min Temperature Autocorrelation and Partial Autocorrelation Plot

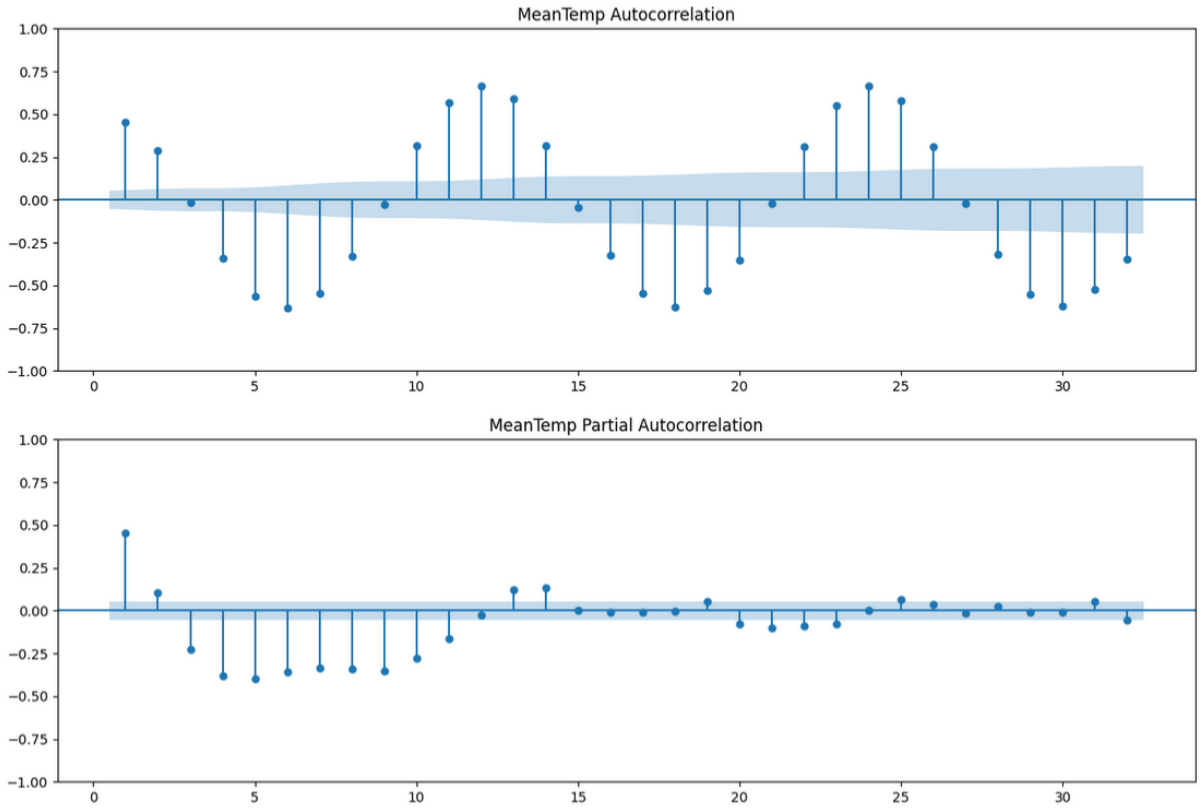


Figure 66: Mean Temperature Autocorrelation and Partial Autocorrelation Plot

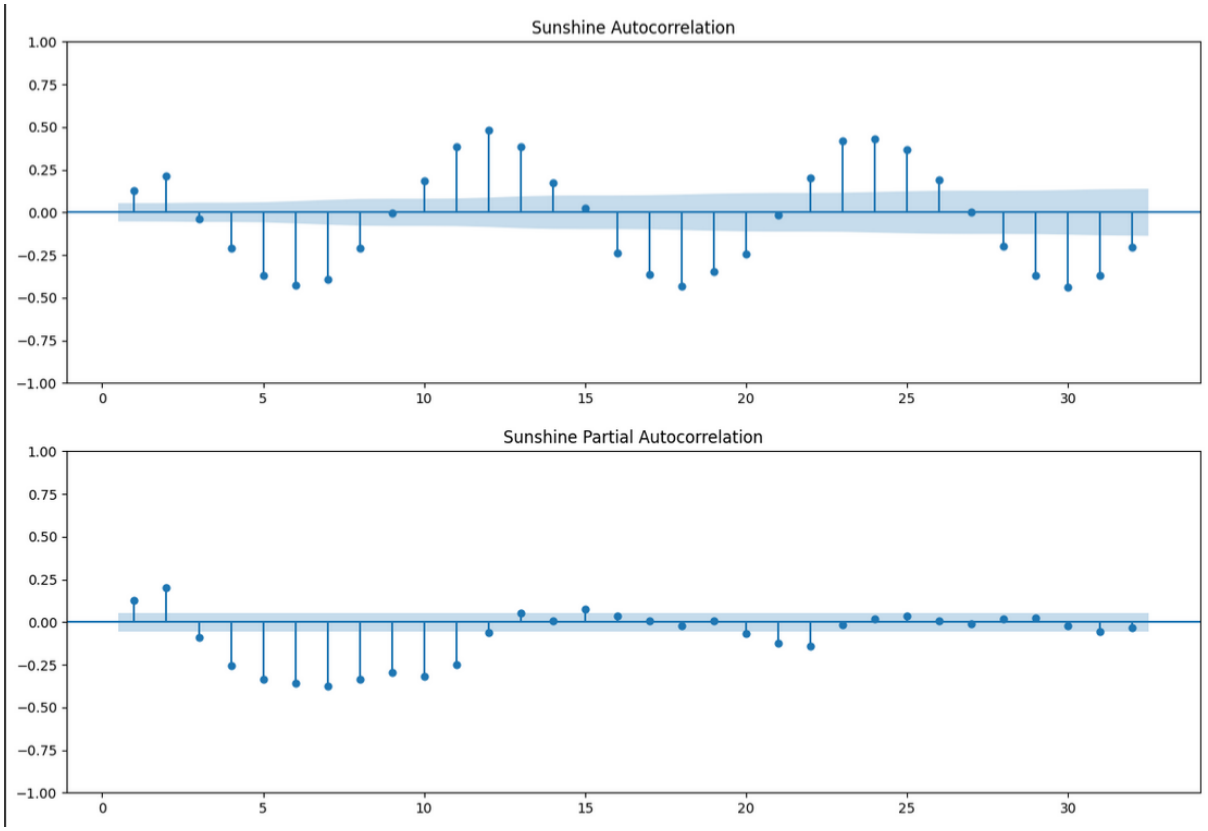


Figure 67: Sunshine Autocorrelation and Partial Autocorrelation Plot

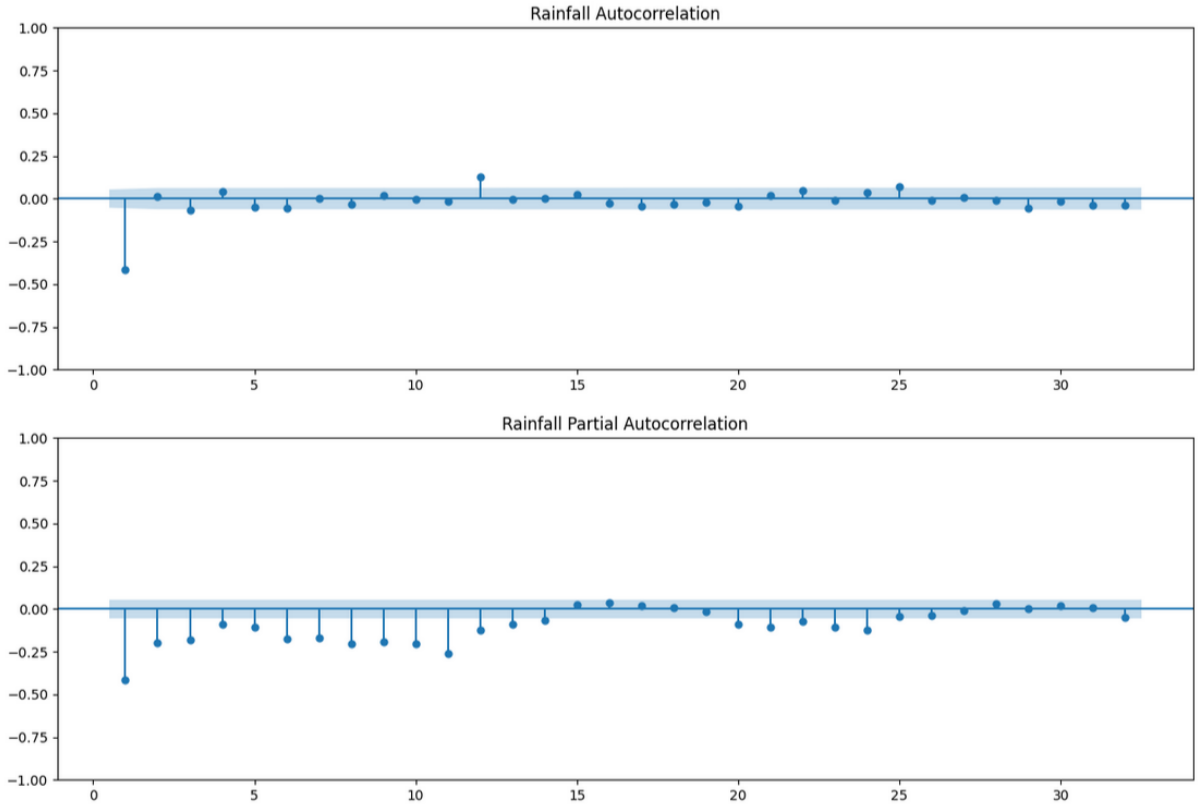


Figure 68: Rainfall Autocorrelation and Partial Autocorrelation Plot

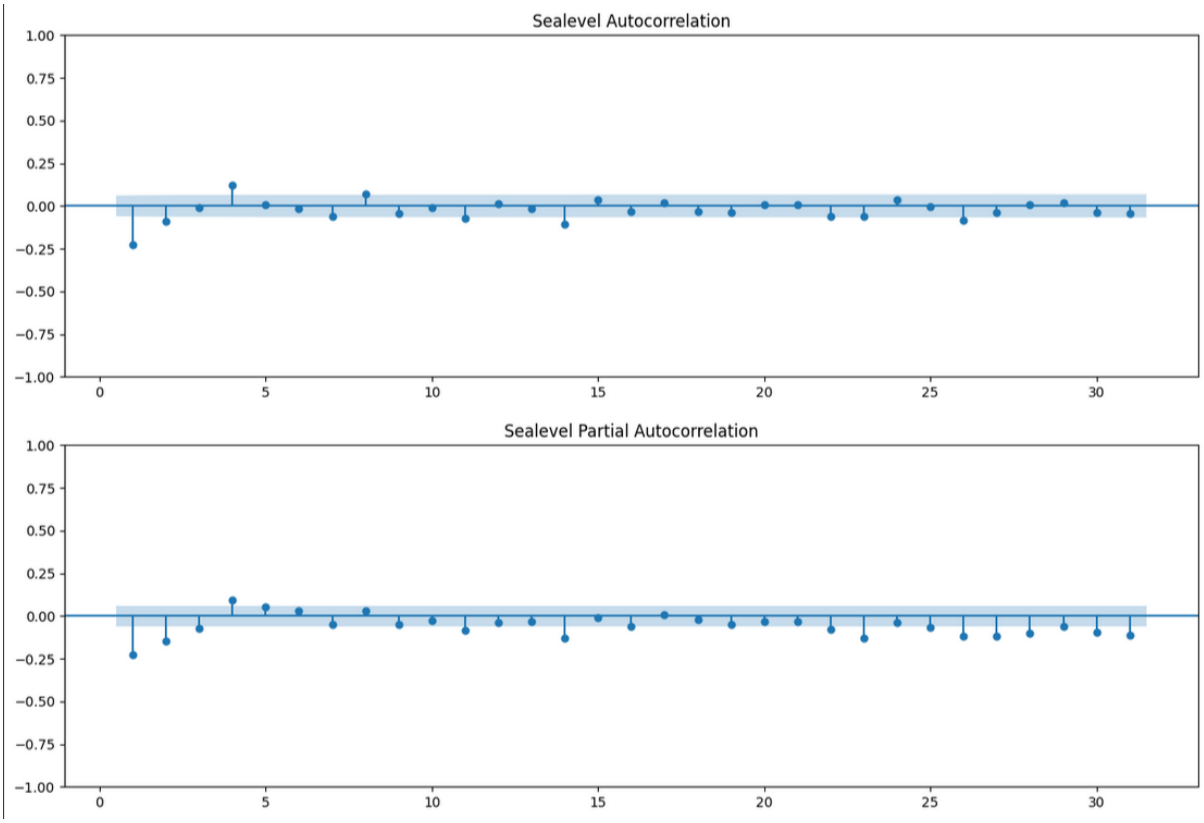


Figure 69: Sea Level Autocorrelation and Partial Autocorrelation Plot

```
Order (0, 0, 0) - CV Error: 8.859790112495398
Order (0, 0, 1) - CV Error: 8.866640381906779
Order (0, 0, 2) - CV Error: 8.85262826228978
Order (0, 1, 0) - CV Error: 24.609422222222218
Order (0, 1, 1) - CV Error: 17.288603100526906
Order (0, 1, 2) - CV Error: 21.373197002486297
Order (1, 0, 0) - CV Error: 8.856314797967844
Order (1, 0, 1) - CV Error: 8.857354373913898
Order (1, 0, 2) - CV Error: 8.843077713692079
Order (1, 1, 0) - CV Error: 19.19341713086946
Order (1, 1, 1) - CV Error: 18.565115428717043
Order (1, 1, 2) - CV Error: 23.92502378782749
Order (2, 0, 0) - CV Error: 8.858614725701253
Order (2, 0, 1) - CV Error: 8.856546957266435
Order (2, 0, 2) - CV Error: 5.155237459367346
Order (2, 1, 0) - CV Error: 18.38632882237642
Order (2, 1, 1) - CV Error: 18.596096025599515
Order (2, 1, 2) - CV Error: 22.309236833777653
Order (3, 0, 0) - CV Error: 8.83353587339134
Order (3, 0, 1) - CV Error: 8.532485651108903
Order (3, 0, 2) - CV Error: 4.6637818457685585
Order (3, 1, 0) - CV Error: 20.712029120008687
Order (3, 1, 1) - CV Error: 20.506339490590605
Order (3, 1, 2) - CV Error: 11.147538522691935
Best Order: (3, 0, 2) with CV Error: 4.6637818457685585
Order (0, 0, 0) - CV Error: 6.19424476857621
Order (0, 0, 1) - CV Error: 6.189550412255208
Order (0, 0, 2) - CV Error: 6.186479729036075
Order (0, 1, 0) - CV Error: 8.419173333333333
Order (0, 1, 1) - CV Error: 7.655209577536937
Order (0, 1, 2) - CV Error: 7.506715561445044
Order (1, 0, 0) - CV Error: 6.185801188986145
Order (1, 0, 1) - CV Error: 6.186026937039176
Order (1, 0, 2) - CV Error: 6.172753017974889
Order (1, 1, 0) - CV Error: 7.949846498045183
Order (1, 1, 1) - CV Error: 7.502471302268601
Order (1, 1, 2) - CV Error: 7.496987689652554
Order (2, 0, 0) - CV Error: 6.186550834881437
Order (2, 0, 1) - CV Error: 6.186883783748209
Order (2, 0, 2) - CV Error: 5.393849373371047
Order (2, 1, 0) - CV Error: 7.852605352650857
Order (2, 1, 1) - CV Error: 7.235091483865536
Order (2, 1, 2) - CV Error: 6.915152137785654
Order (3, 0, 0) - CV Error: 6.181780547154602
Order (3, 0, 1) - CV Error: 6.024307360975786
Order (3, 0, 2) - CV Error: 4.829002922678258
Order (3, 1, 0) - CV Error: 7.882997326523464
Order (3, 1, 1) - CV Error: 6.937076902261121
Order (3, 1, 2) - CV Error: 6.50398178184228
Best Order: (3, 0, 2) with CV Error: 4.829002922678258
Order (0, 0, 0) - CV Error: 7.066918504339905
Order (0, 0, 1) - CV Error: 7.063900247522961
Order (0, 0, 2) - CV Error: 7.056632775352187
Order (0, 1, 0) - CV Error: 14.342977777777781
Order (0, 1, 1) - CV Error: 10.990210036588792
Order (0, 1, 2) - CV Error: 12.176026033404888
Order (1, 0, 0) - CV Error: 7.055728766735885
Order (1, 0, 1) - CV Error: 7.056195889291297
```

Figure 70: Climate Dataset Best Orders

```
RMSE: [1.4813539153343813, 1.3471989141802718, 1.3081786014600818, 1.5385964980094604, 1.6843503201096488]
Mean RMSE: 1.471935649818769
RMSE: [1.4498178800755228, 1.2338709833115704, 1.2516061615491914, 1.4925844167354665, 1.6578747913135012]
Mean RMSE: 1.4171508465970506
RMSE: [1.3219622814609826, 1.2710019074653345, 1.1960503339348352, 1.4150440866806342, 1.5970946483441633]
Mean RMSE: 1.36023065157719
RMSE: [22.929243601621348, 22.92062095289428, 22.418445719930567, 23.11735169249497, 25.005511744262954]
Mean RMSE: 23.278234742240823
RMSE: [38.427040053137134, 34.901617809497324, 36.10941147881898, 40.10107654054152, 39.57793123141614]
Mean RMSE: 37.82341542268222
```

Figure 71: Climate Best Order MSE and RMSE Results

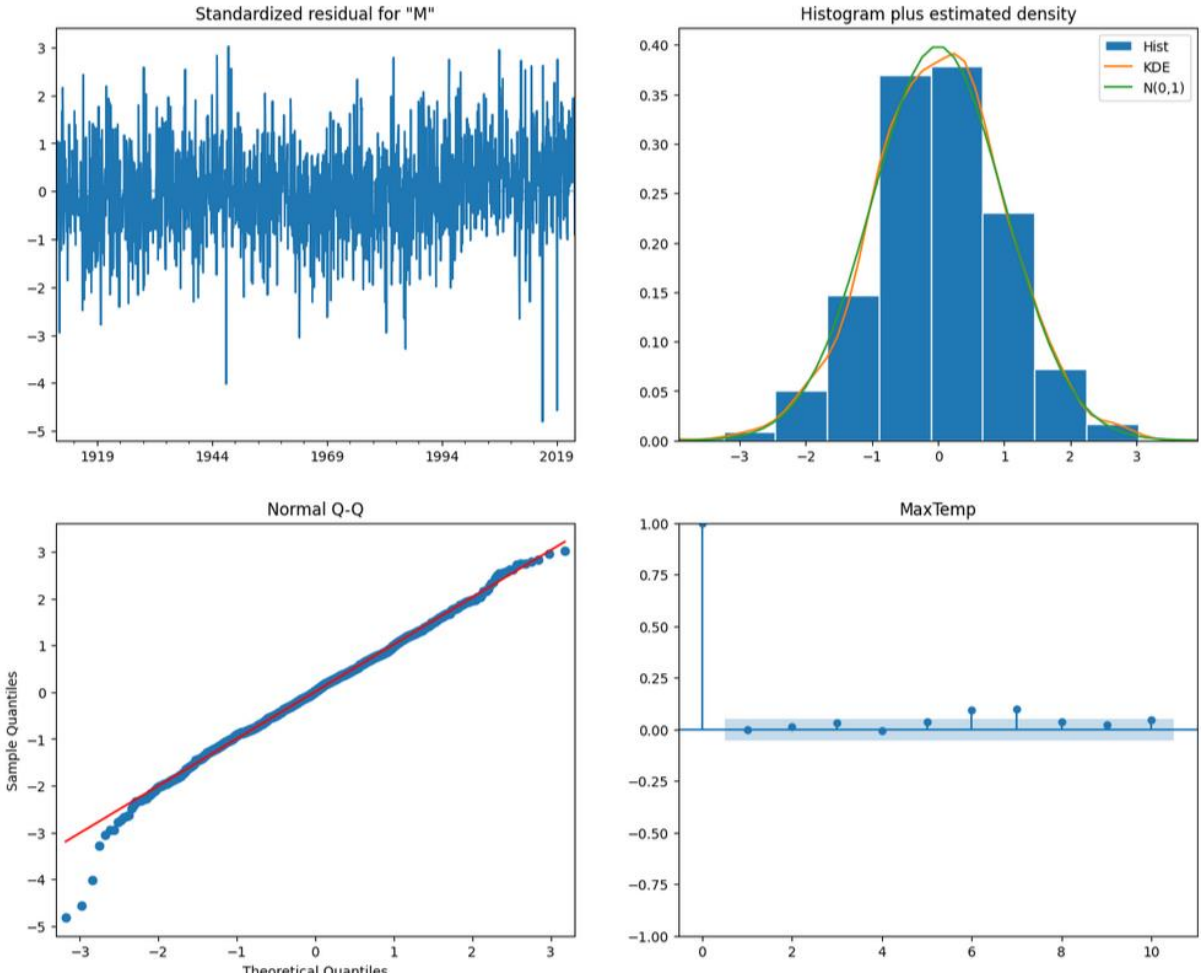


Figure 72: Max Temperature Diagnostics Plot

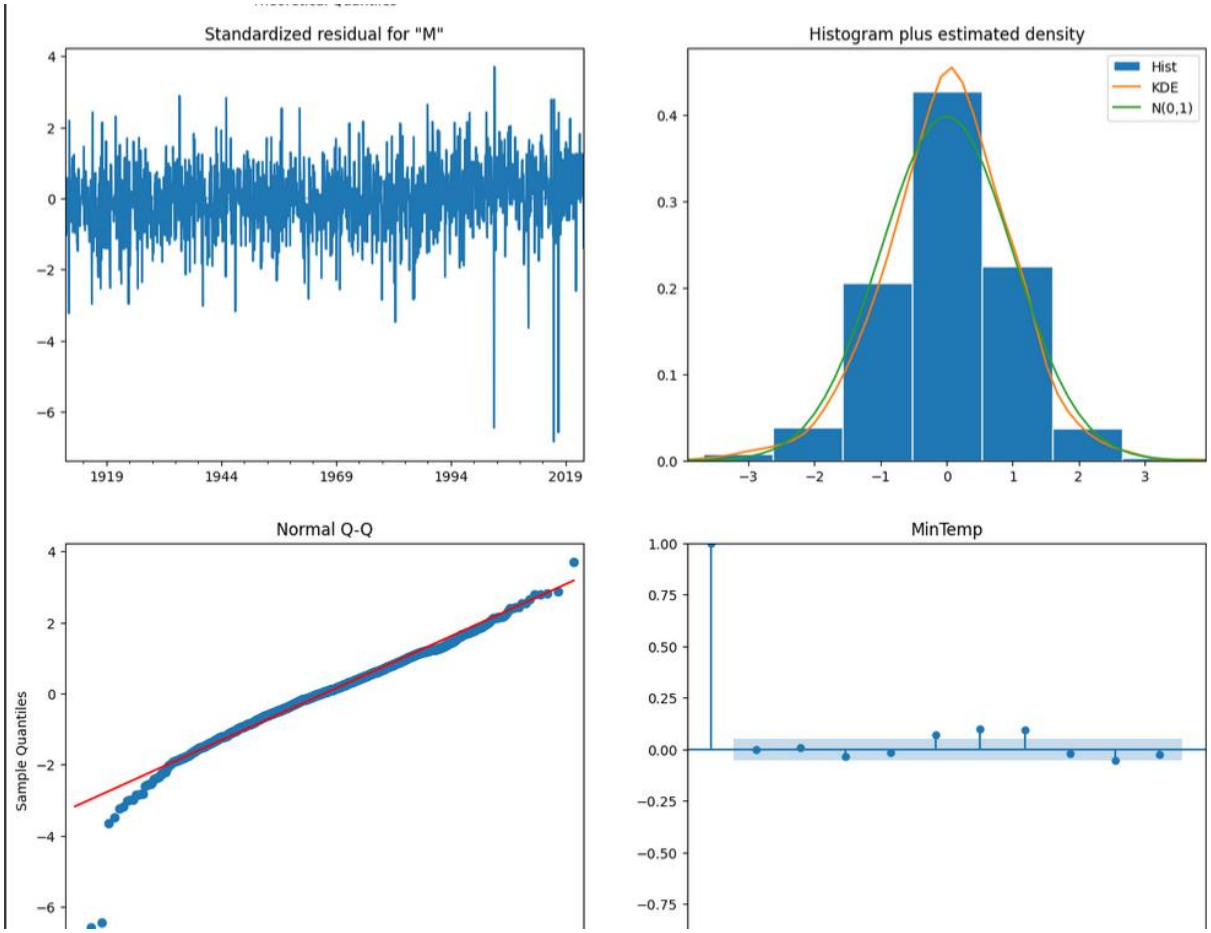


Figure 73: Min Temperature Diagnostics Plot

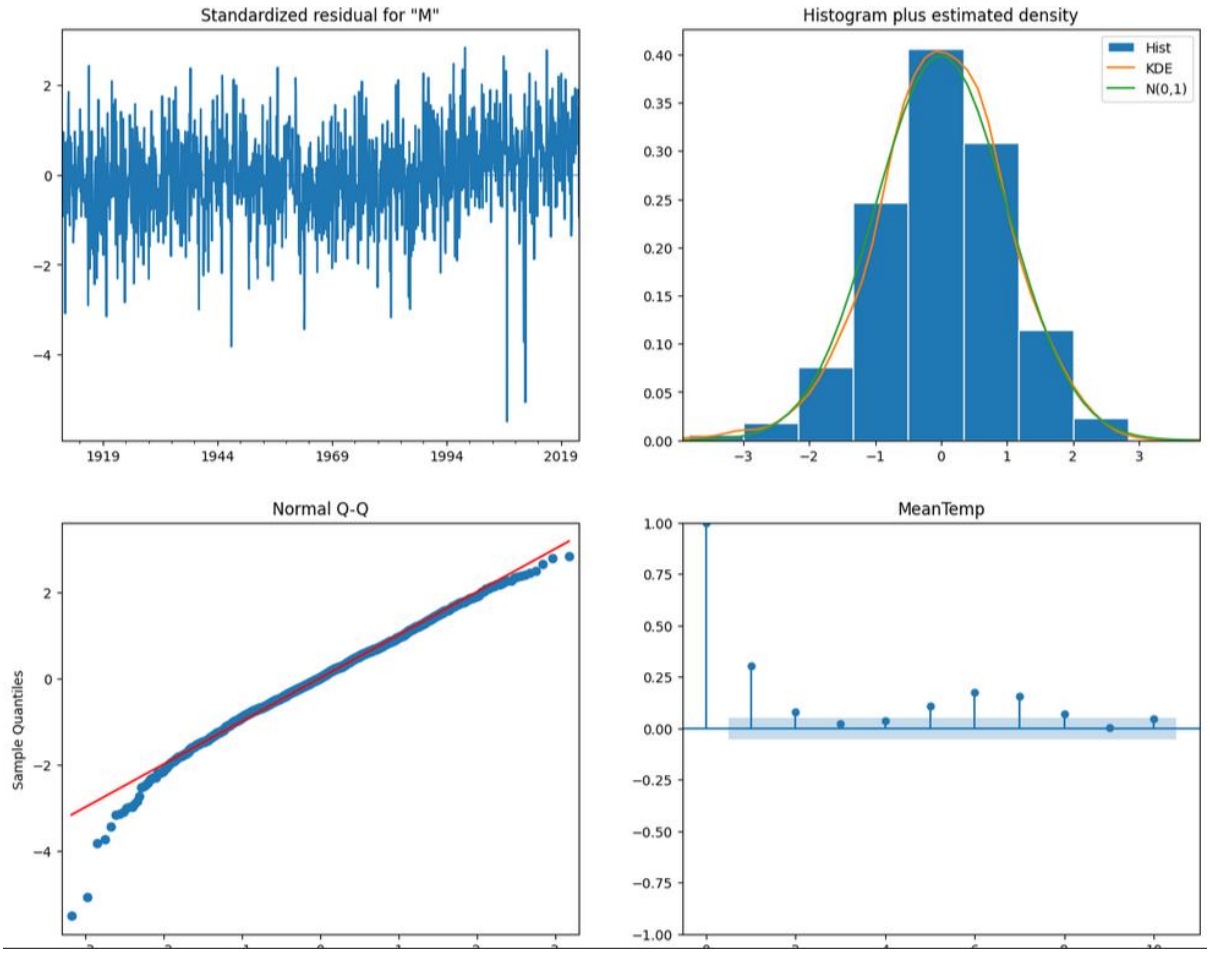


Figure 74: Mean Temperature Diagnostics Plot

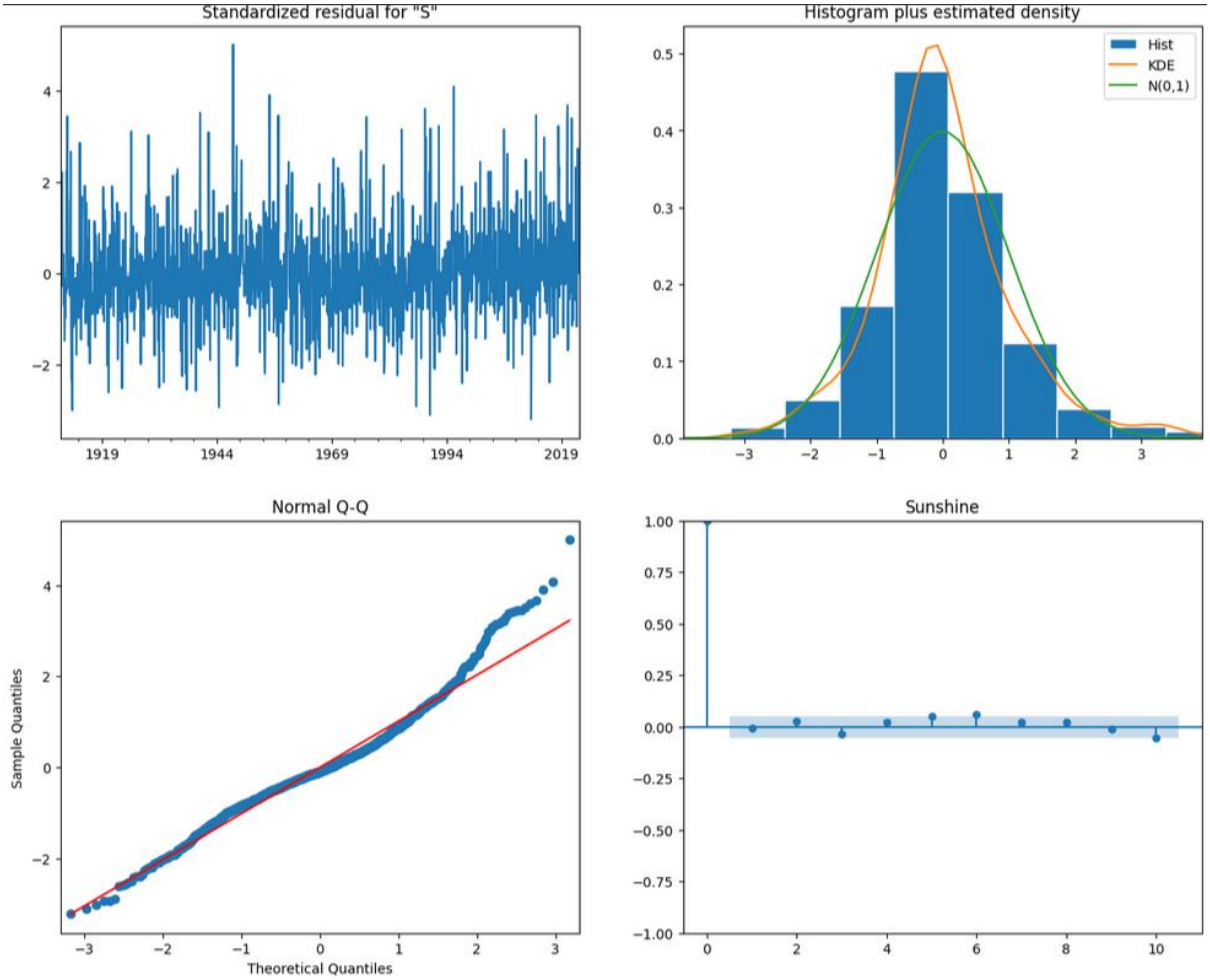


Figure 75: Sunshine Diagnostics Plot

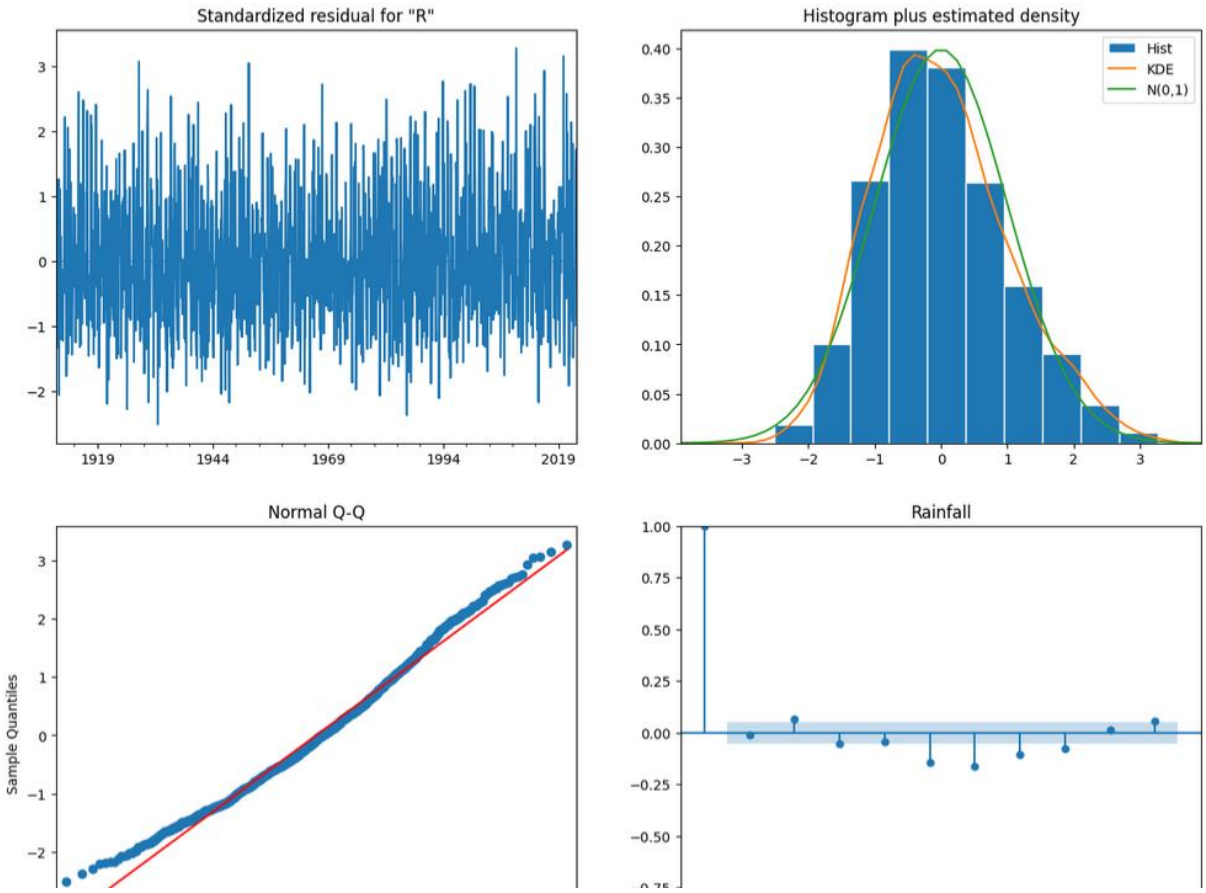


Figure 76: Rainfall Plot Diagnostics

```
sealevel_best_order = best_order_function(sealeveldf.diff().dropna().values)

Order (0, 0, 0) - CV Error: 5.849769598918871
Order (0, 0, 1) - CV Error: 5.8559481843204235
Order (0, 0, 2) - CV Error: 5.857430775233796
Order (0, 1, 0) - CV Error: 10.081476206896554
Order (0, 1, 1) - CV Error: 5.849830306953615
Order (0, 1, 2) - CV Error: 5.8558254150071205
Order (1, 0, 0) - CV Error: 5.8518891154242105
Order (1, 0, 1) - CV Error: 5.857156868699745
Order (1, 0, 2) - CV Error: 5.85743585172512
Order (1, 1, 0) - CV Error: 8.354450327743645
Order (1, 1, 1) - CV Error: 5.8518848932472185
Order (1, 1, 2) - CV Error: 5.856989580513064
Order (2, 0, 0) - CV Error: 5.85291165100993
Order (2, 0, 1) - CV Error: 5.8583082090140035
Order (2, 0, 2) - CV Error: 5.857146939697531
Order (2, 1, 0) - CV Error: 7.247660654920895
Order (2, 1, 1) - CV Error: 5.852951203831779
Order (2, 1, 2) - CV Error: 5.8581017864143305
Order (3, 0, 0) - CV Error: 5.863852454155283
Order (3, 0, 1) - CV Error: 5.864114374103115
Order (3, 0, 2) - CV Error: 5.854002872422891
Order (3, 1, 0) - CV Error: 6.609381754099111
Order (3, 1, 1) - CV Error: 5.863526036787336
Order (3, 1, 2) - CV Error: 5.855997380843032
Best Order: (0, 0, 0) with CV Error: 5.849769598918871
```

Figure 77: Sea Level Best Order

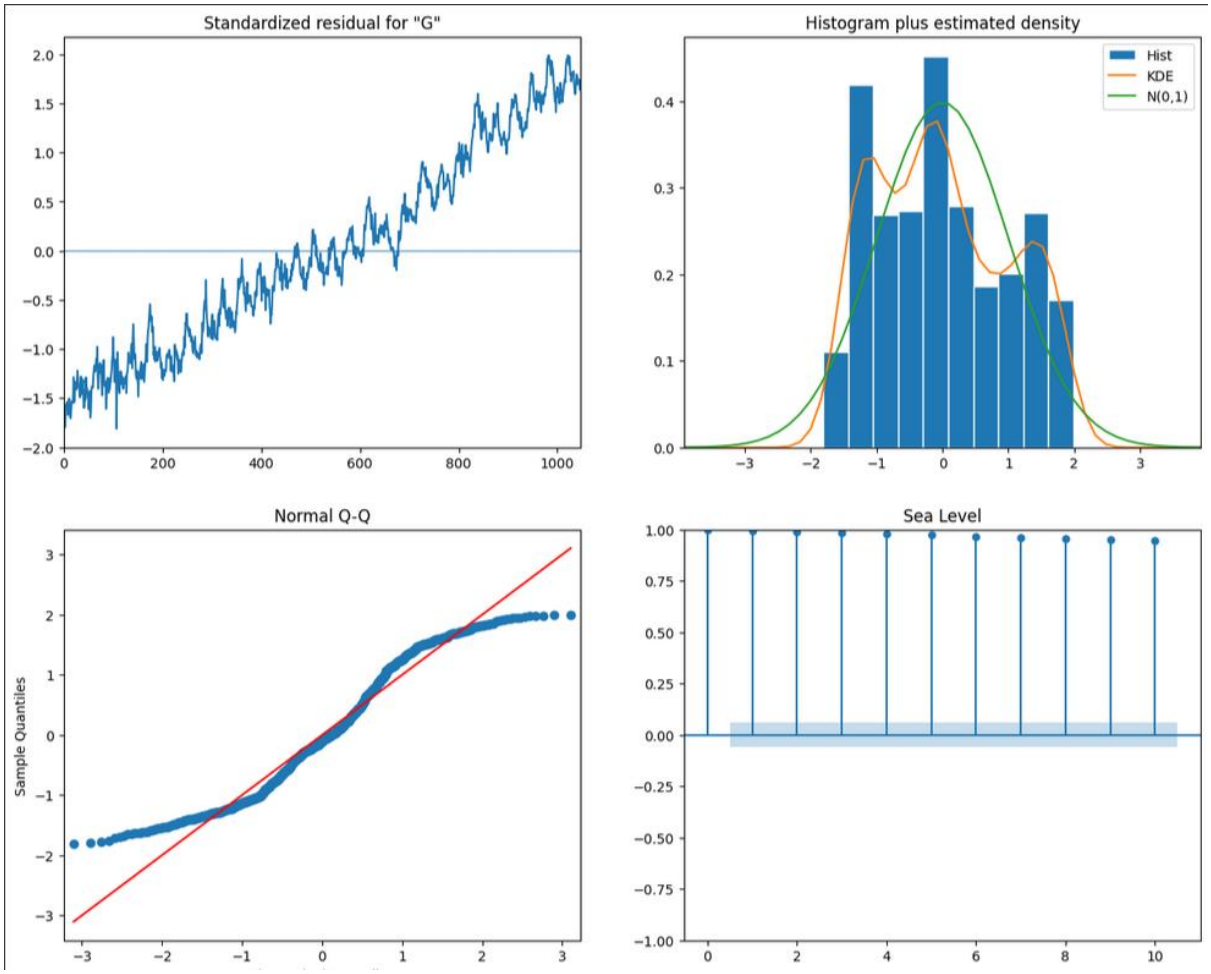


Figure 78: Sea Level Diagnostics Plot

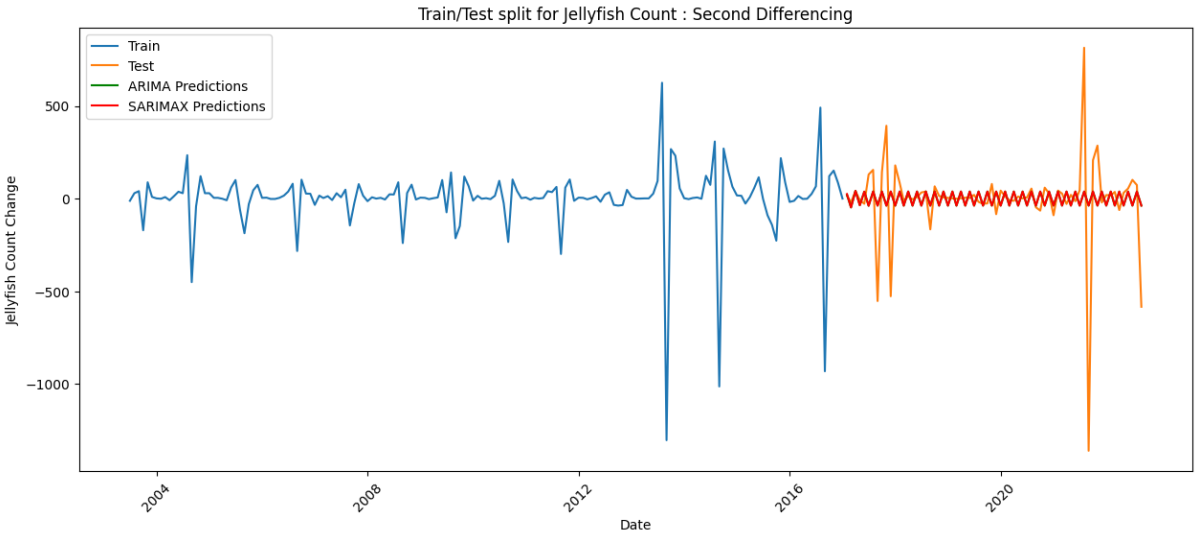


Figure 79: Jellyfish Count Second Differencing ARIMA/SARIMAX plot

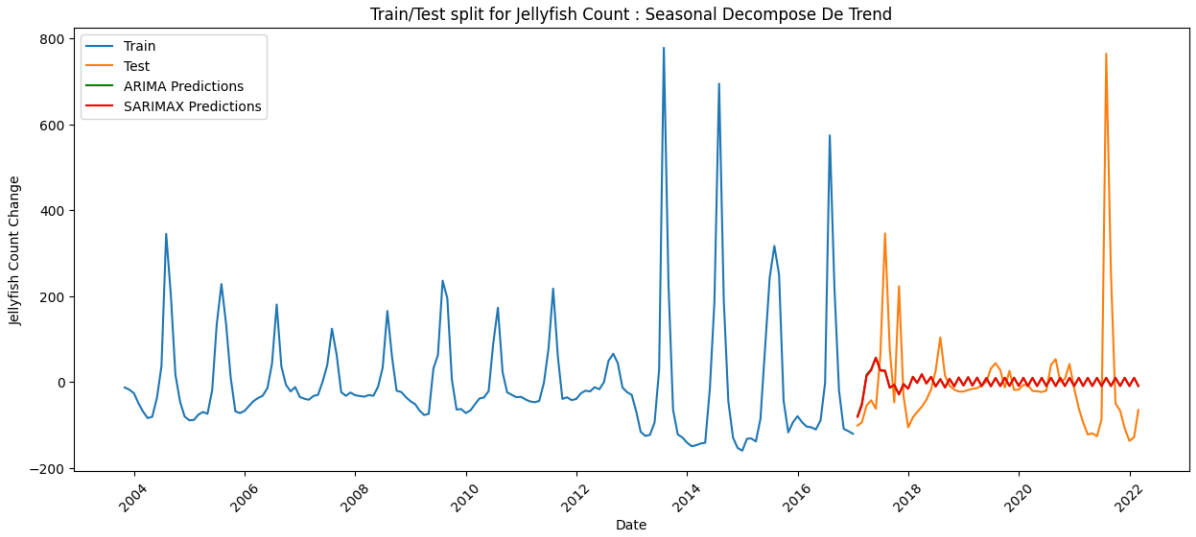


Figure 80: Jellyfish Count Seasonal Decompose De Trended ARIMA/SARIMAX Plot

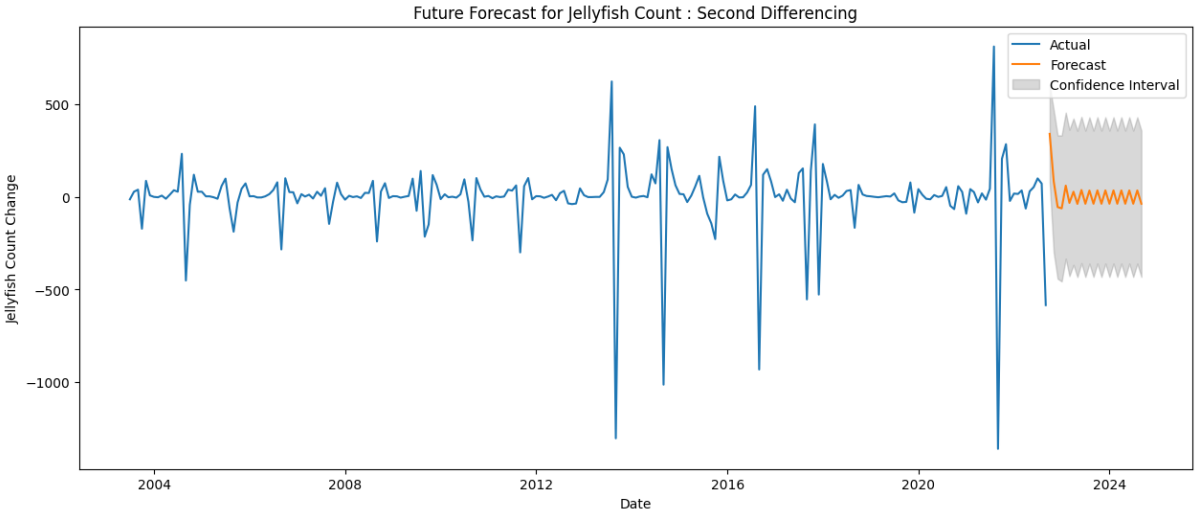


Figure 81: Jellyfish Count Second Differencing Future Forecast

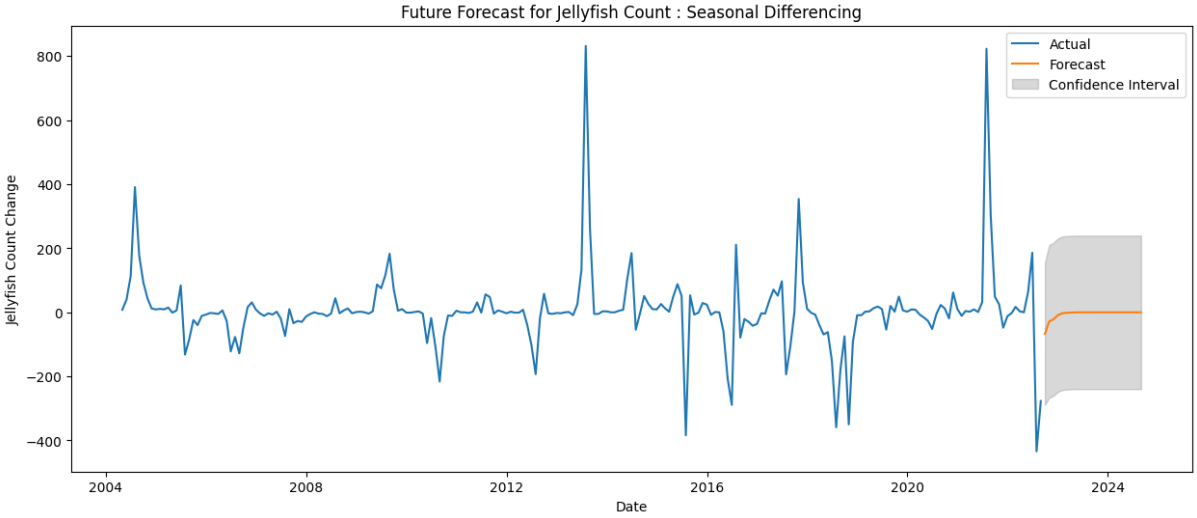


Figure 82: Jellyfish Count Seasonal Differencing Future Forecast

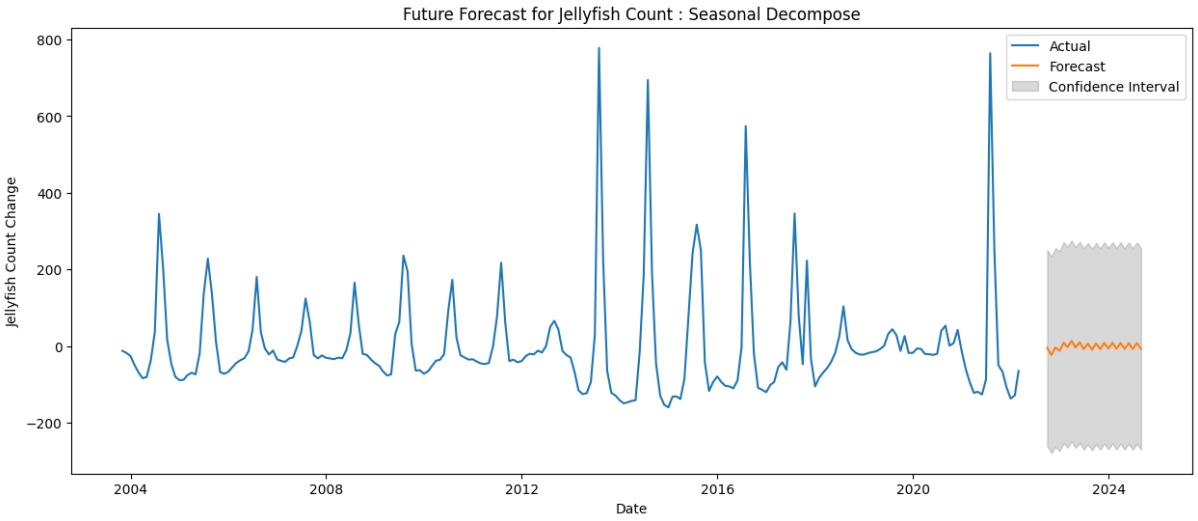


Figure 83: Jellyfish Count Seasonal Decompose Future Forecasting

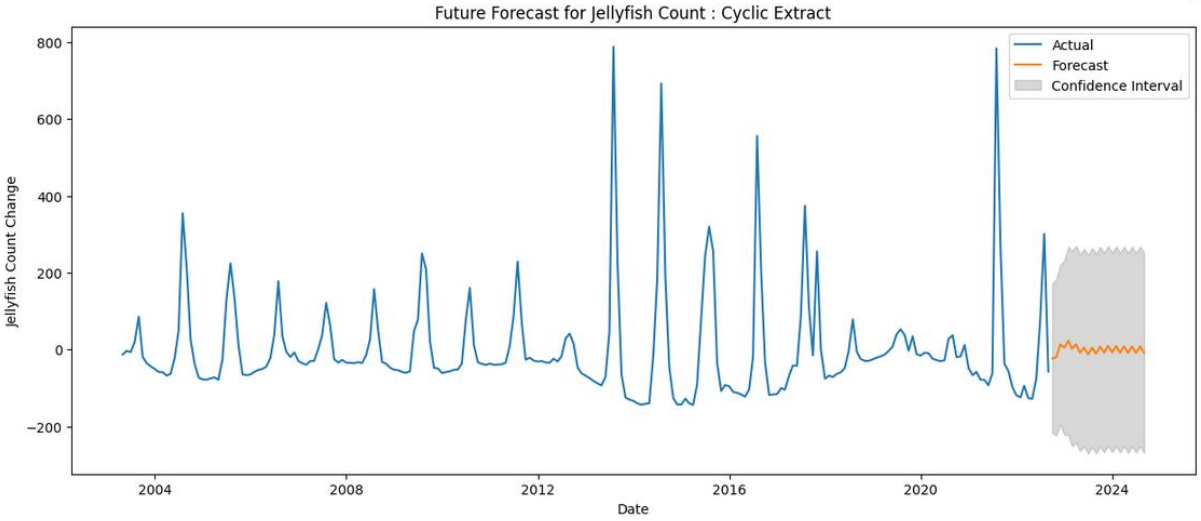


Figure 84: Jellyfish Count Cyclic Extract Future Forecasting

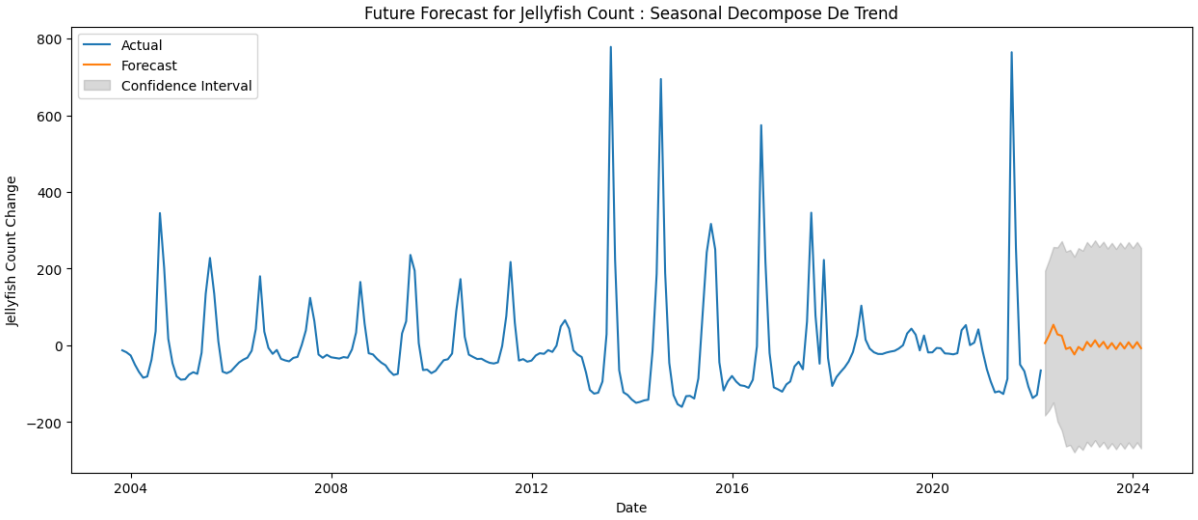


Figure 85: Jellyfish Count Seasonal Decompose De Trend Future Forecast

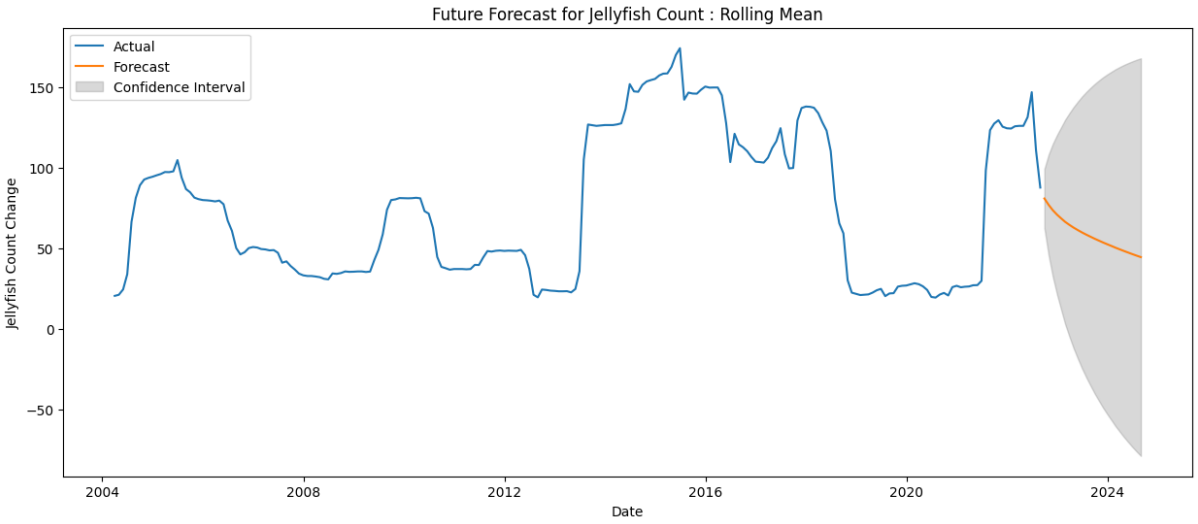


Figure 86: Jellyfish Count Rolling Mean Future Forecast

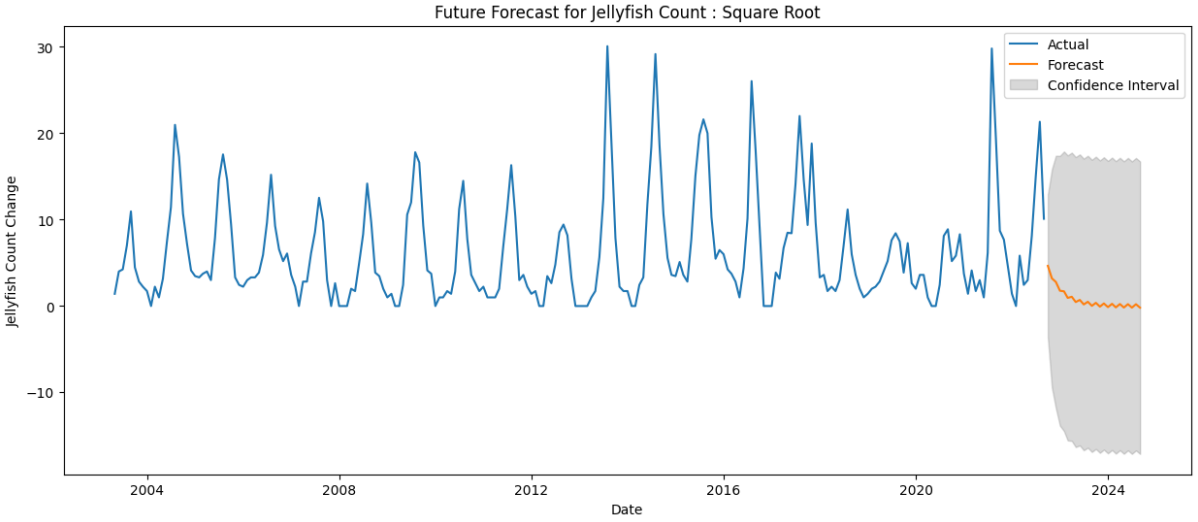


Figure 87: Jellyfish Count Square Root Future Forecast

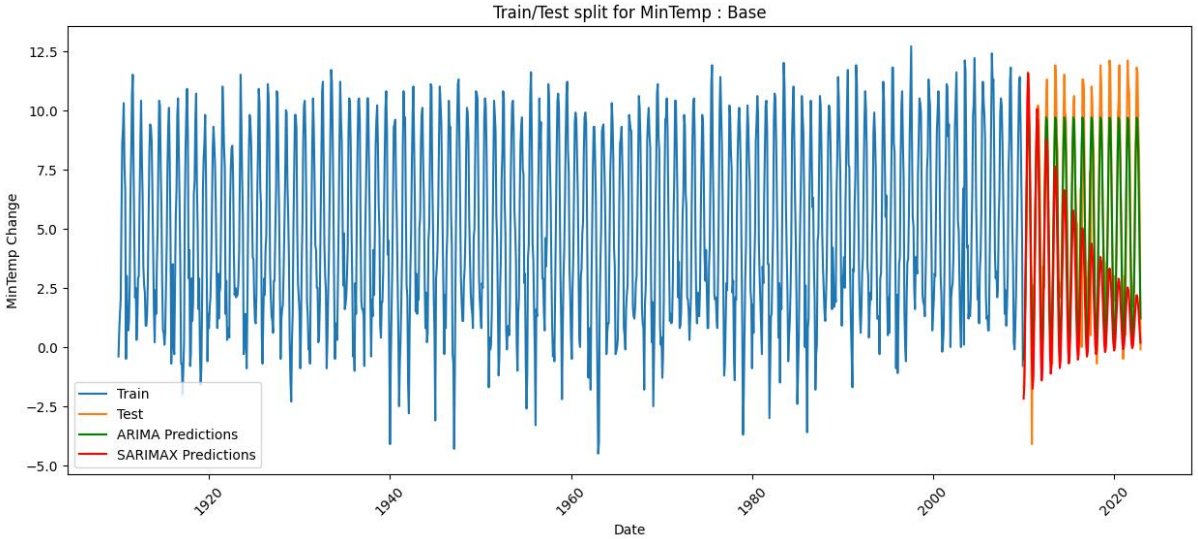


Figure 88: Min Temperature ARIMA/SARIMAX model

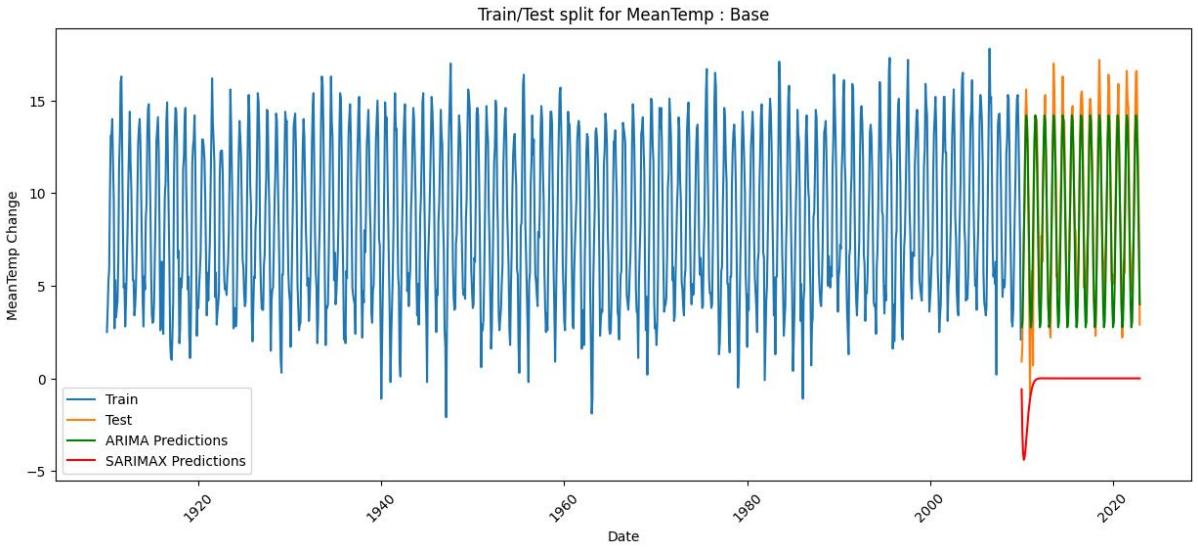


Figure 89: Mean Temperature ARIMA/SARIMAX Model

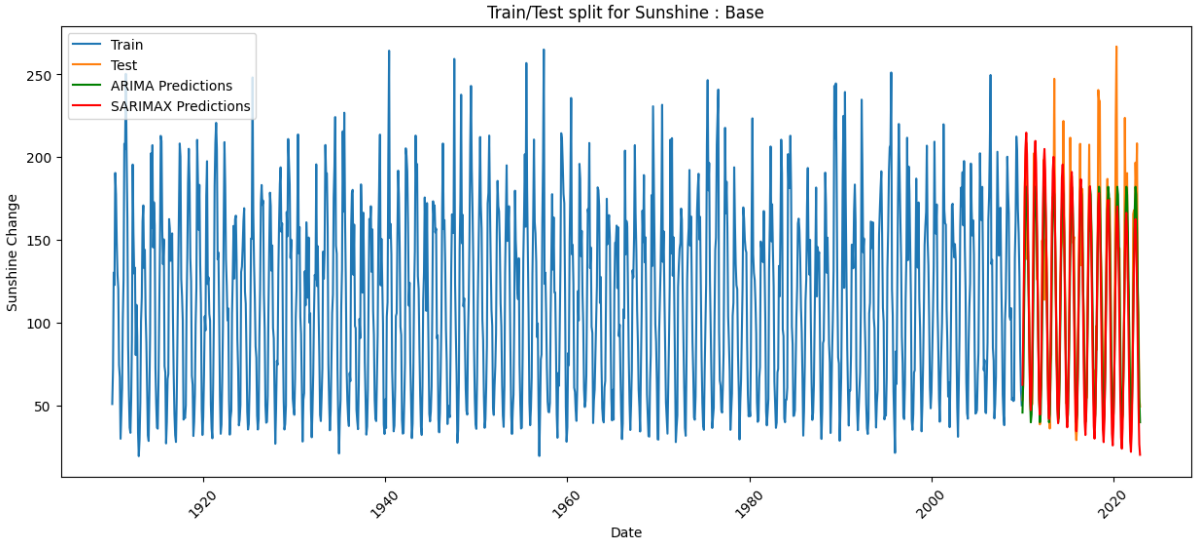


Figure 90: Sunshine ARIMA/SARIMAX Model

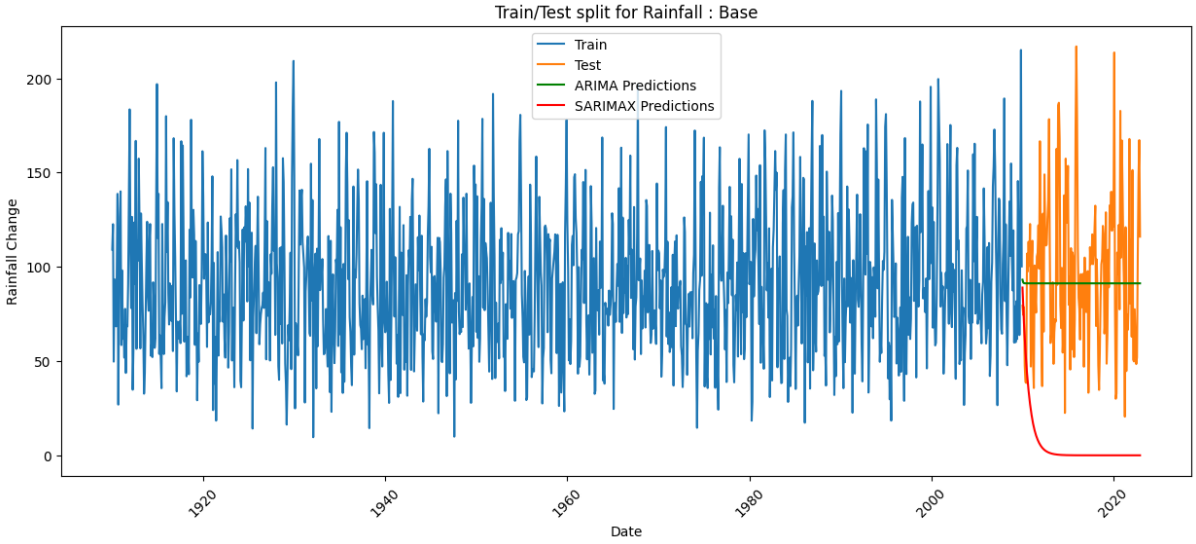


Figure 91: Rainfall ARIMA/SARIMAX Model

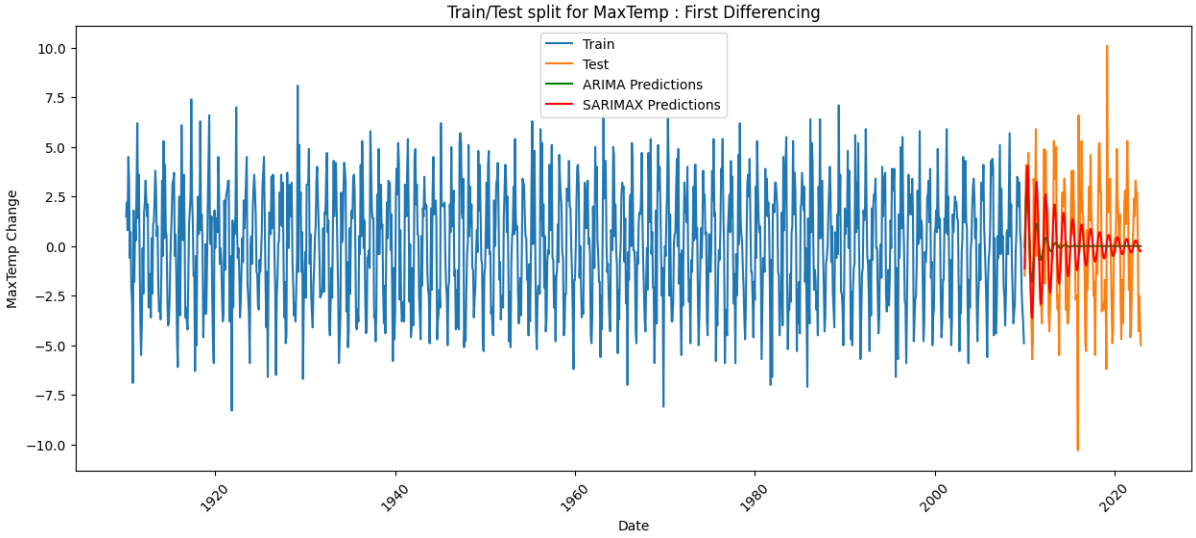


Figure 92: Max Temperature First Differencing ARIMA/SARIMAX Model

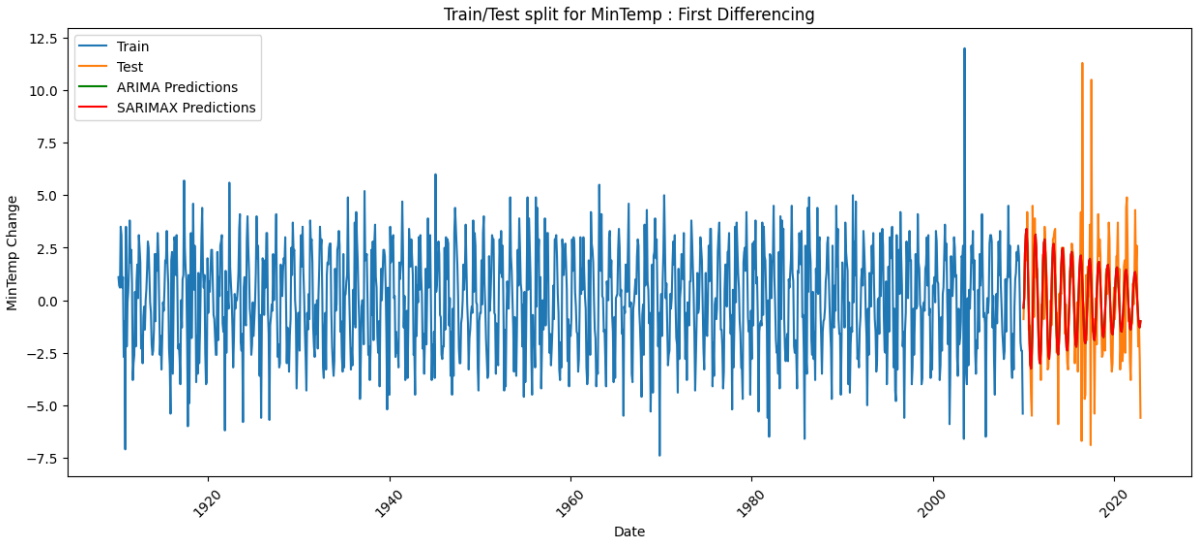


Figure 93: Min Temperature First Differencing ARIMA/SARIMAX Model

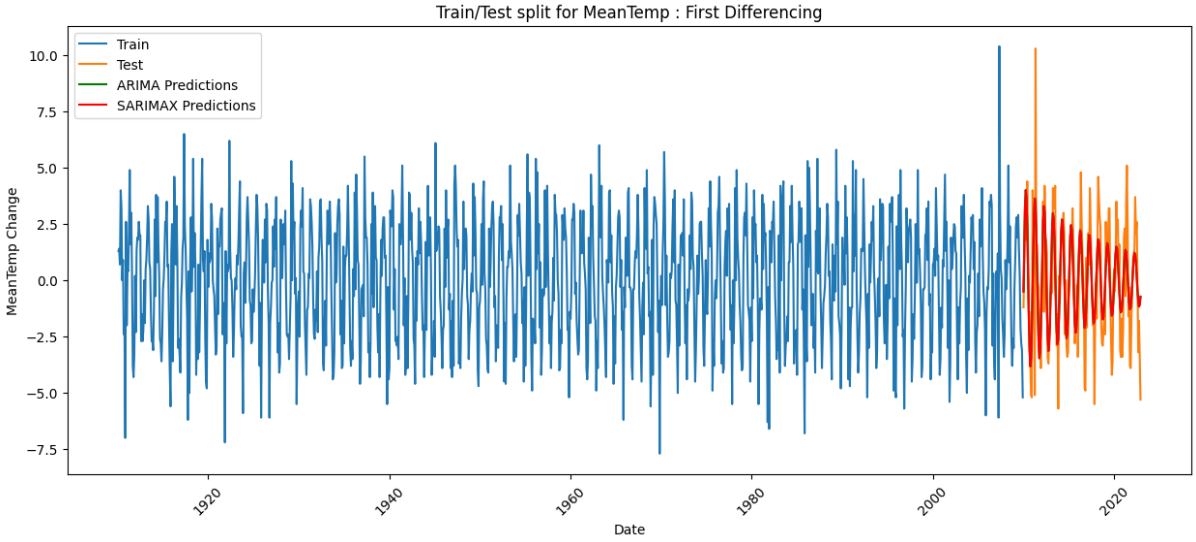


Figure 94: Mean Temperature First Differencing ARIMA/SARIMAX Model

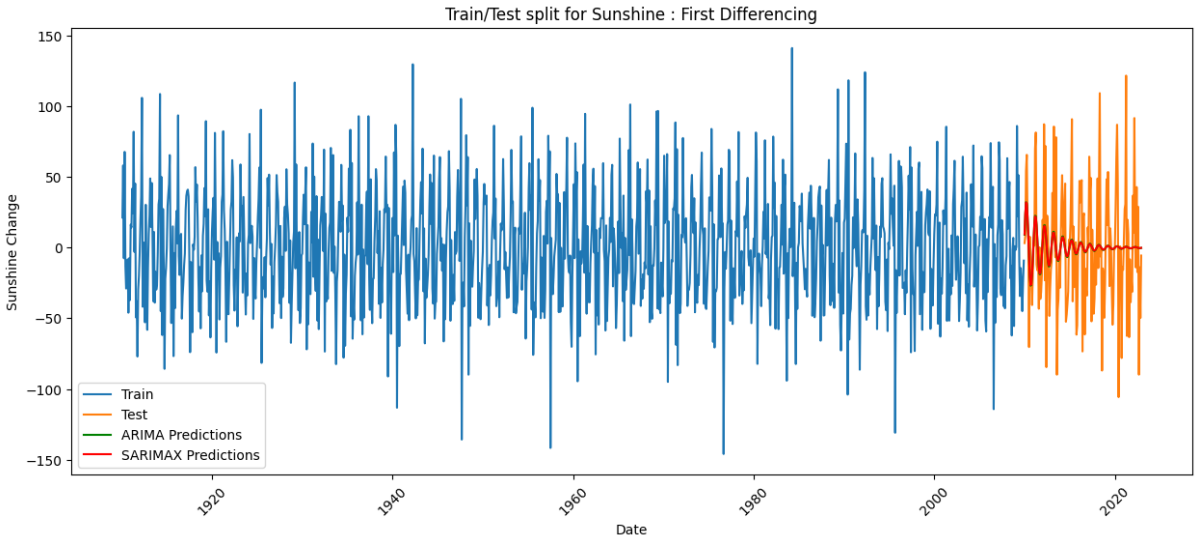


Figure 95: Sunshine First Difference ARIMA/SARIMAX Model

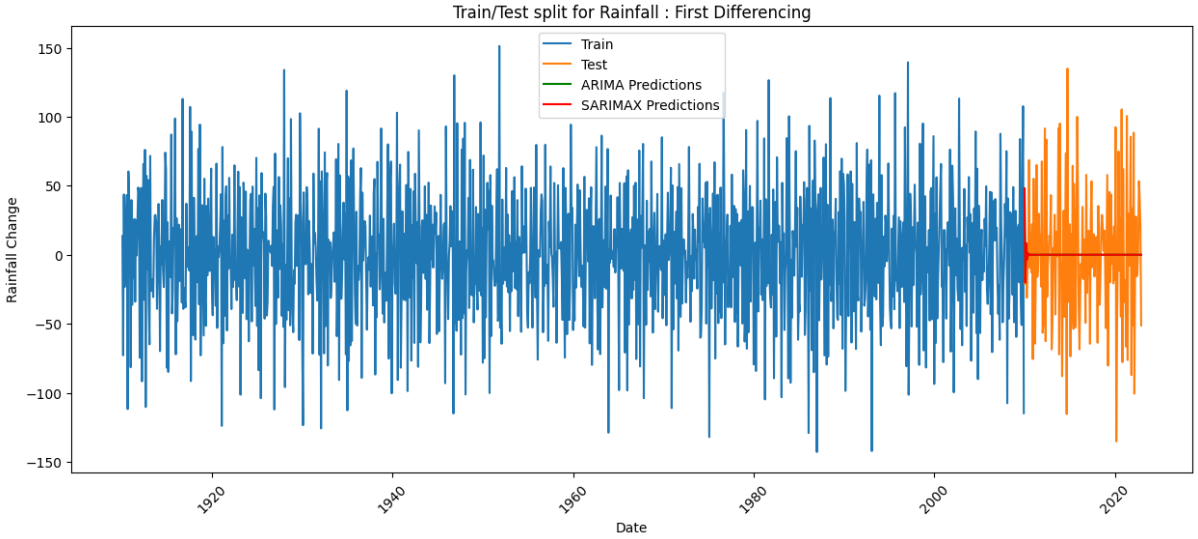


Figure 96: Rainfall First Difference ARIMA/SARIMAX Model

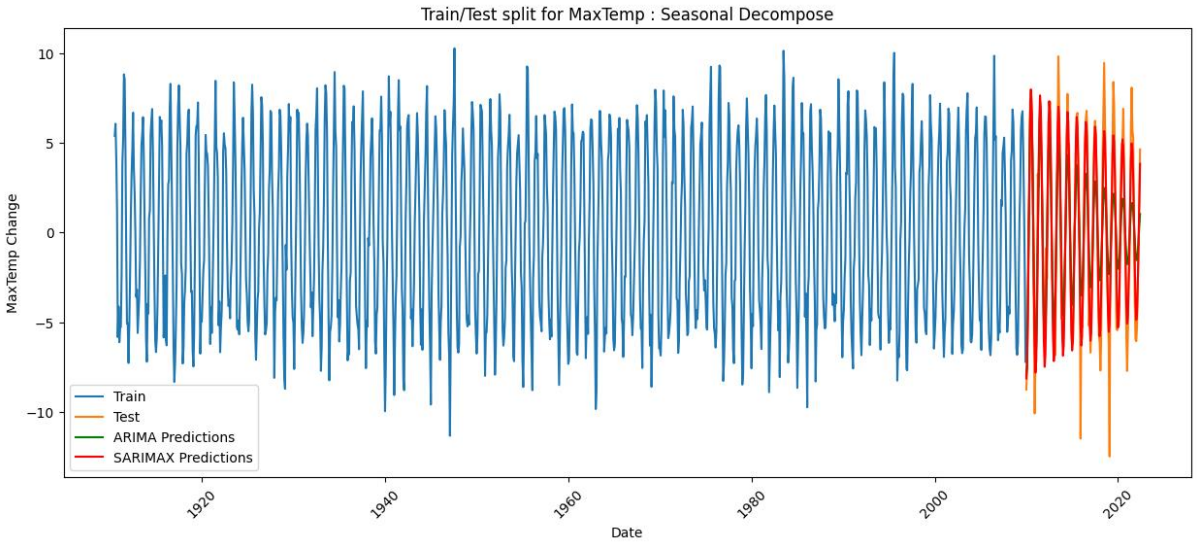


Figure 97: Max Temperature Seasonal Decompose ARIMA/SARIMAX Model

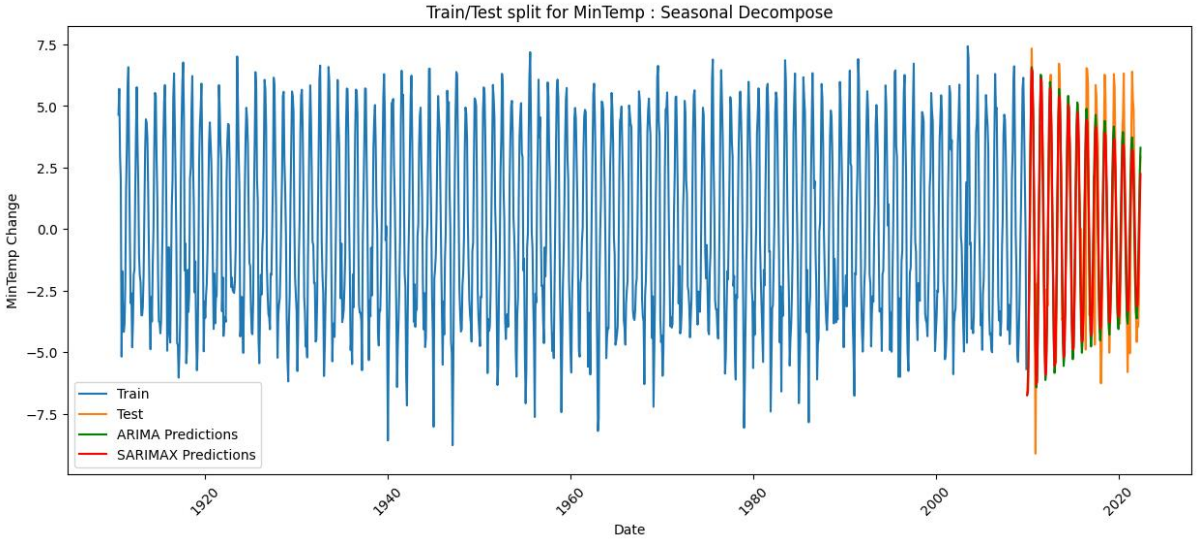


Figure 98: Min Temperature Seasonal Decompose ARIMA/SARIMAX Model

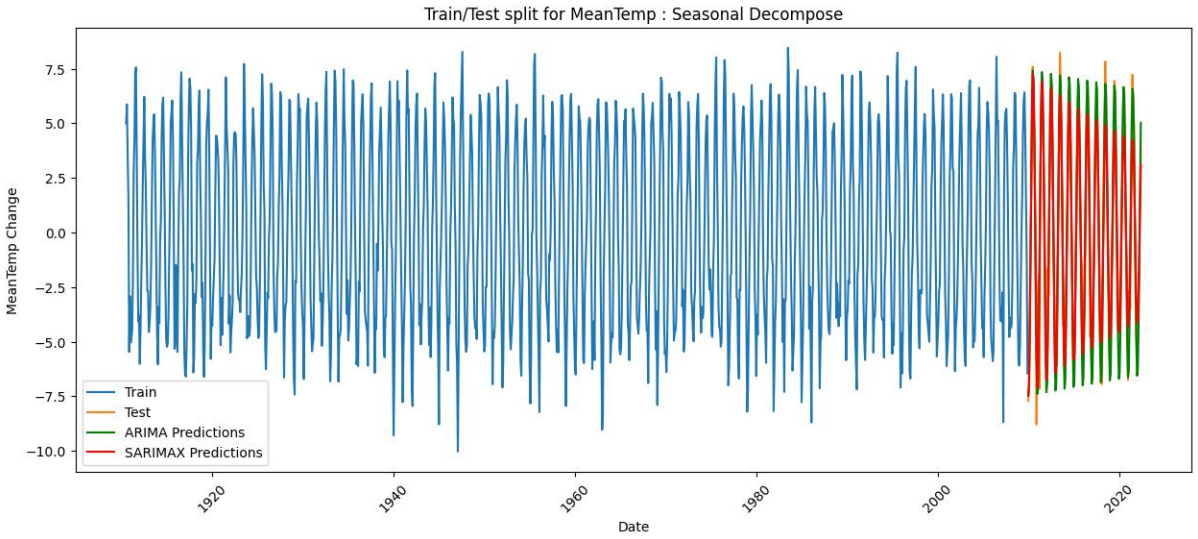


Figure 99: Mean Temperature Seasonal Decompose ARIMA/SARIMAX Model

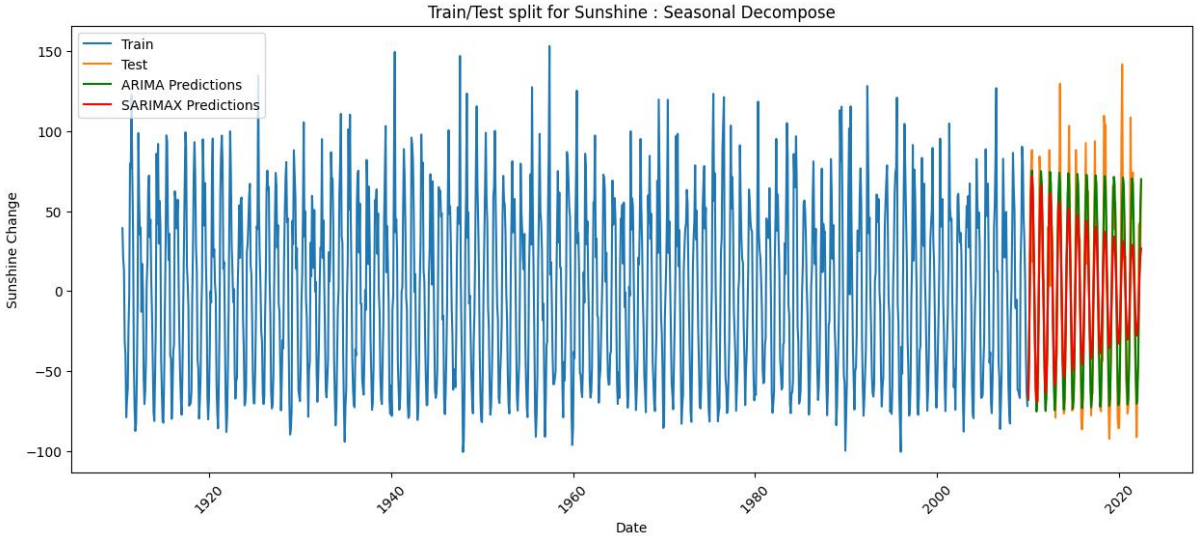


Figure 100: Sunshine Seasonal Decompose ARIMA/SARIMAX Model

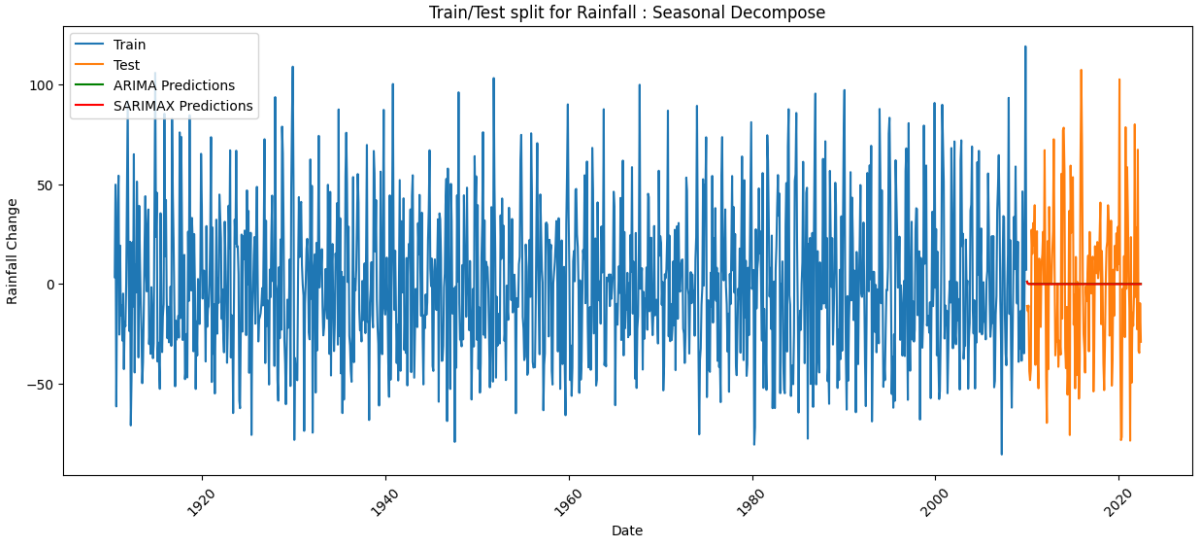


Figure 101: Rainfall Seasonal Decompose ARIMA/SARIMAX Model

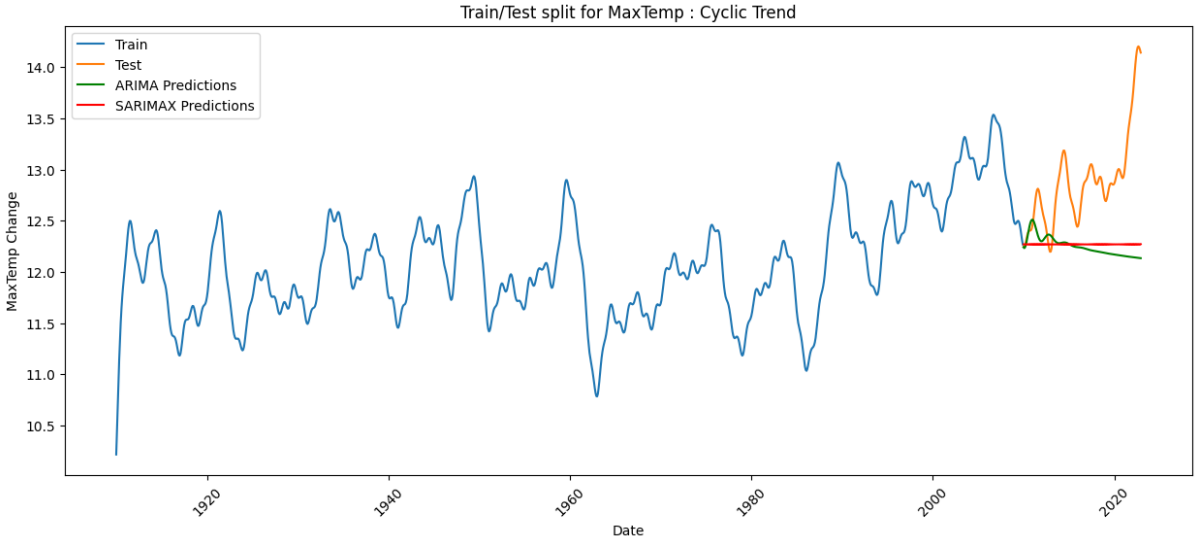


Figure 102: Max Temperature Cyclic Trend ARIMA/SARIMAX Model

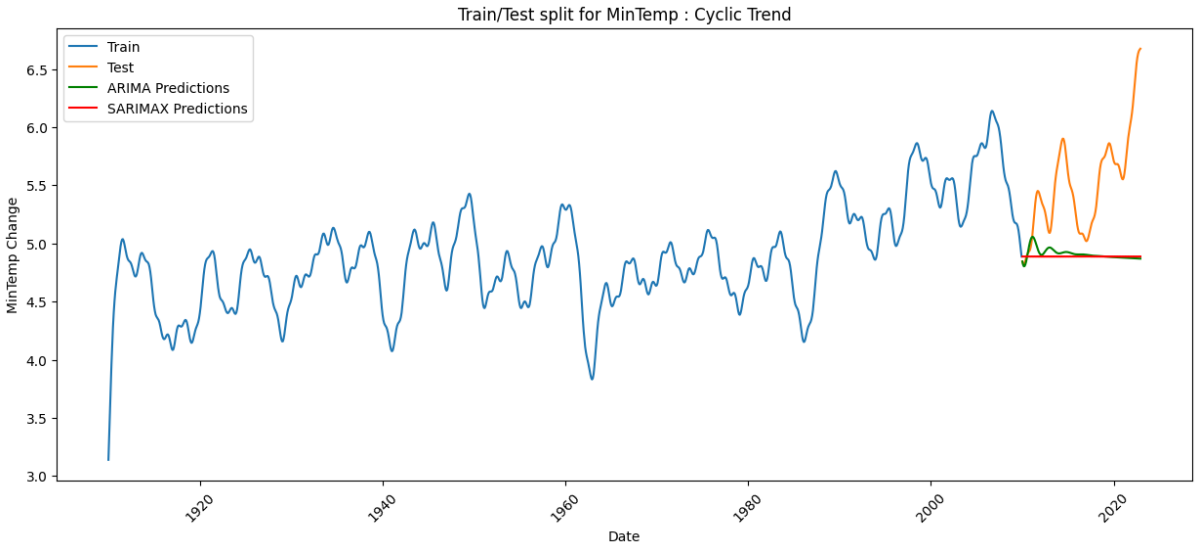


Figure 103: Min Temperature Cyclic Trend ARIMA/SARIMAX Model

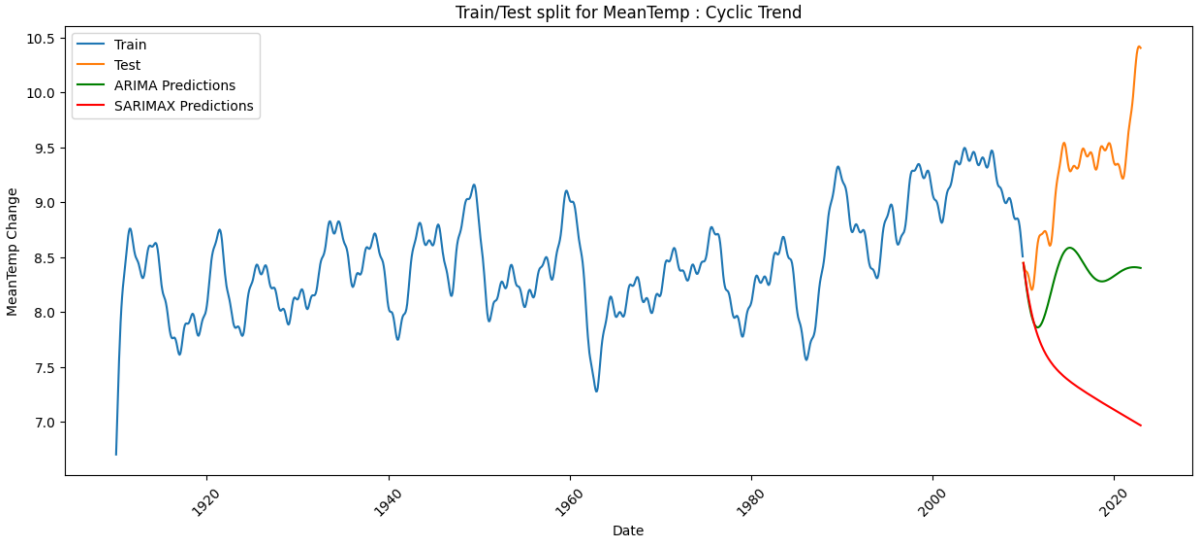


Figure 104: Mean Temperature Cyclic Trend ARIMA/SARIMAX Model

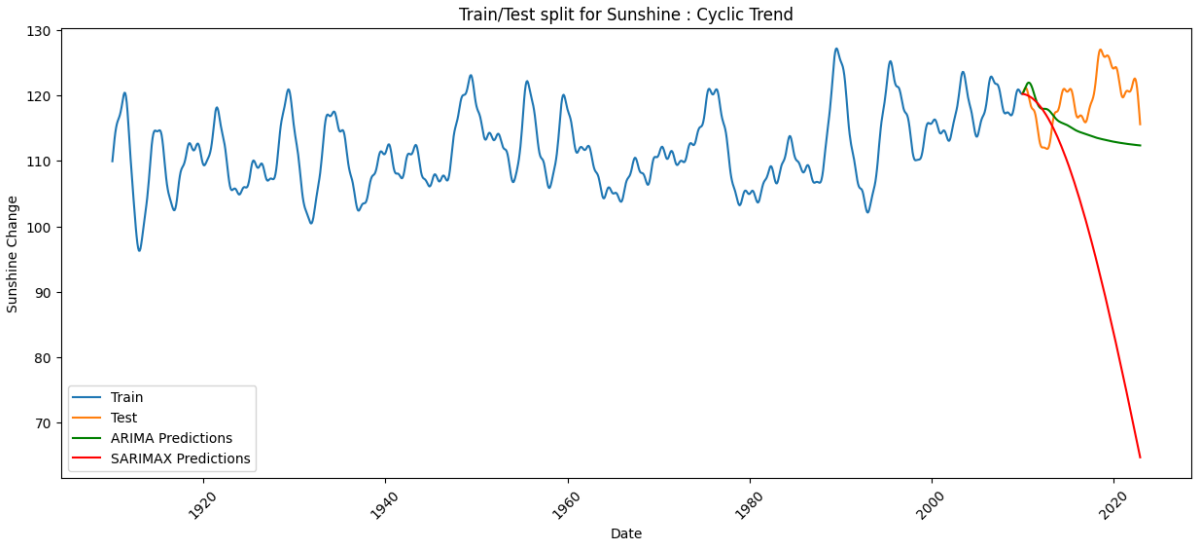


Figure 105: Sunshine Cyclic Trend ARIMA/SARIMAX Model

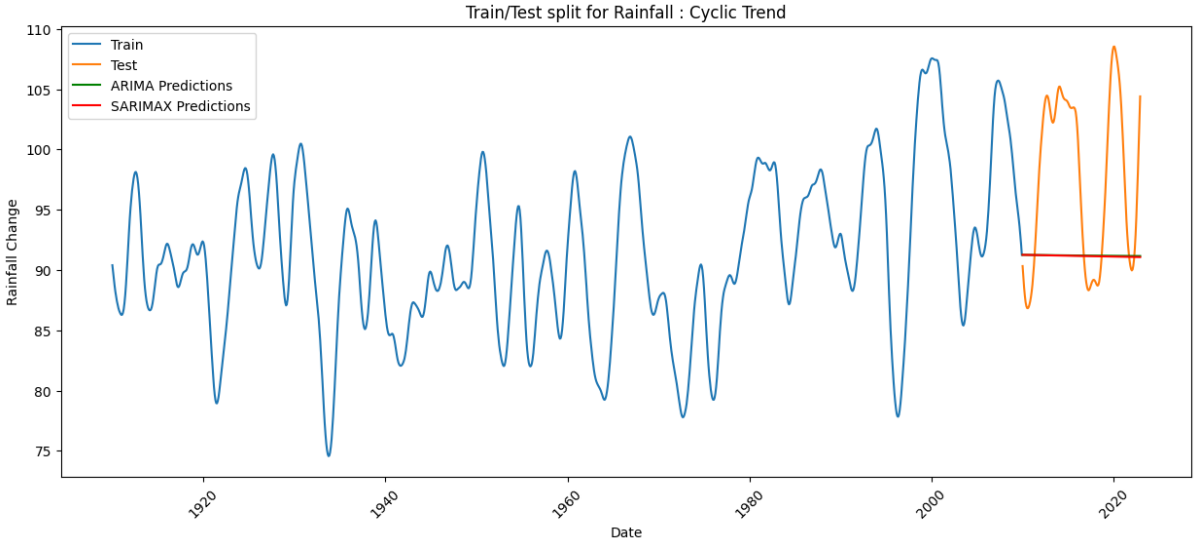


Figure 106: Rainfall Cyclic Trend ARIMA/SARIMAX Model

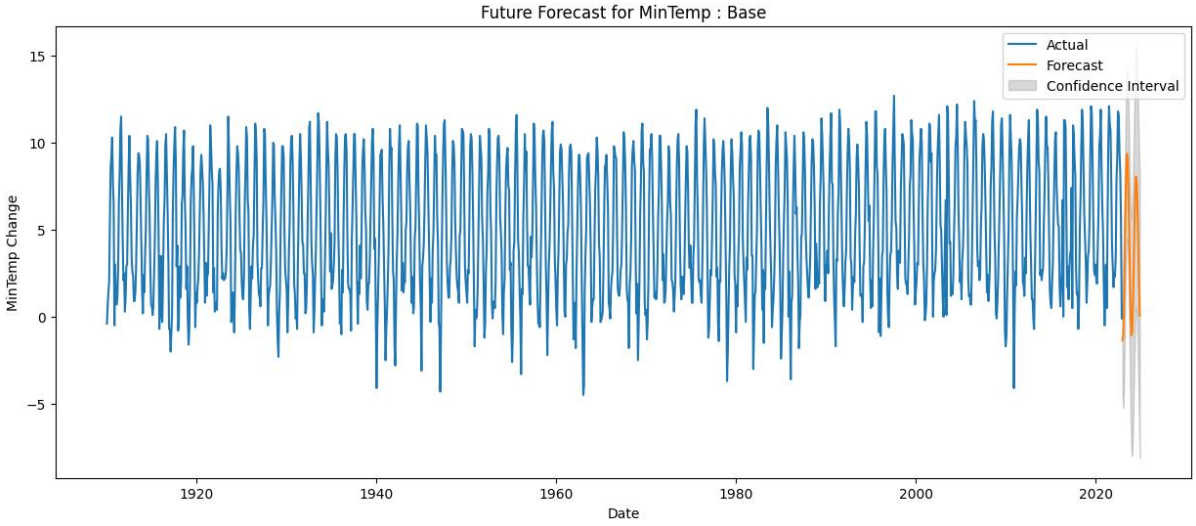


Figure 107: Min Temperature Future Forecast

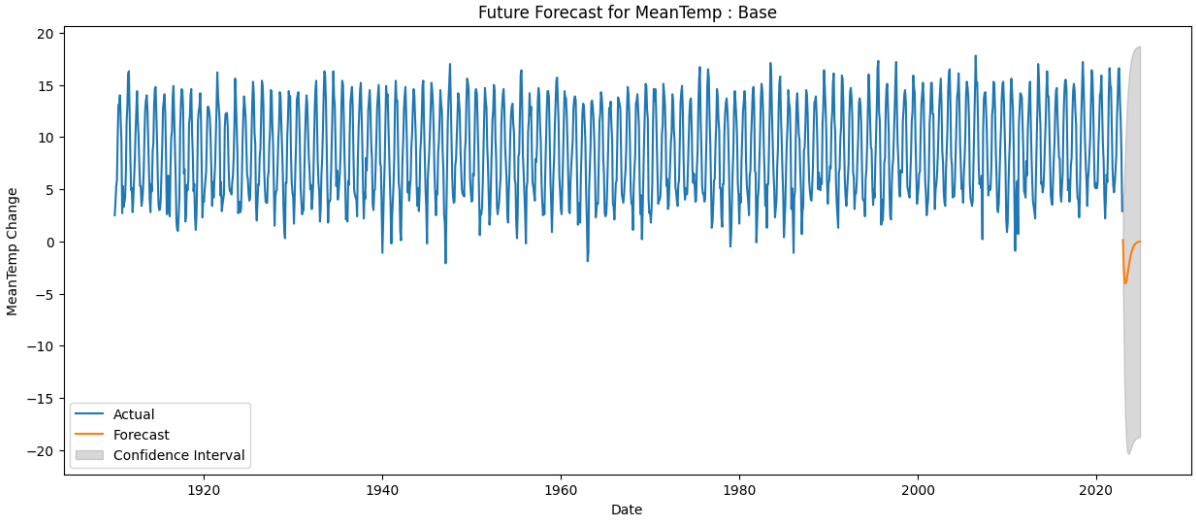


Figure 108: Mean Temperature Future Forecast

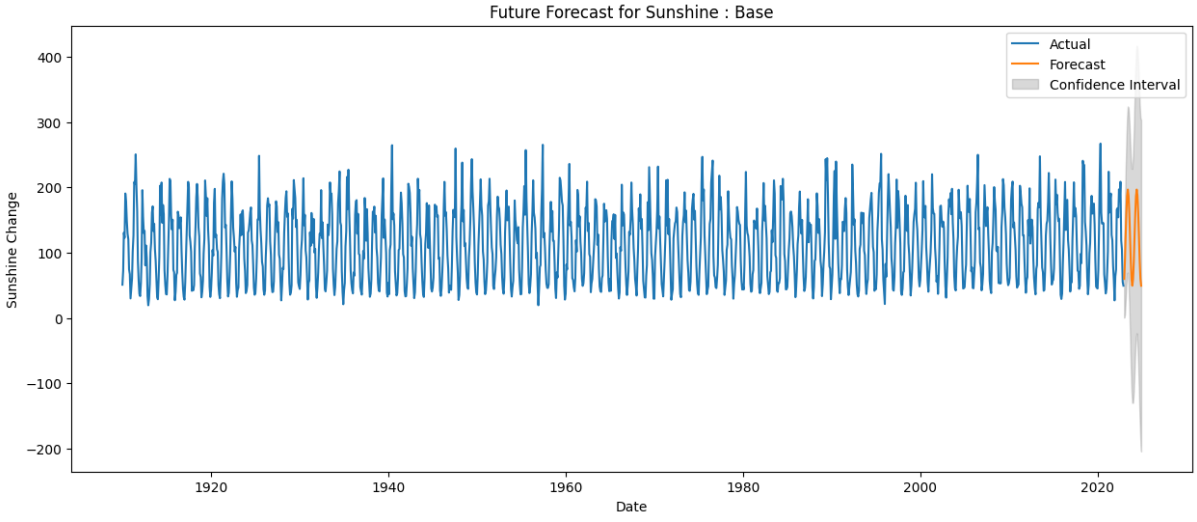


Figure 109: Sunshine Future Forecast

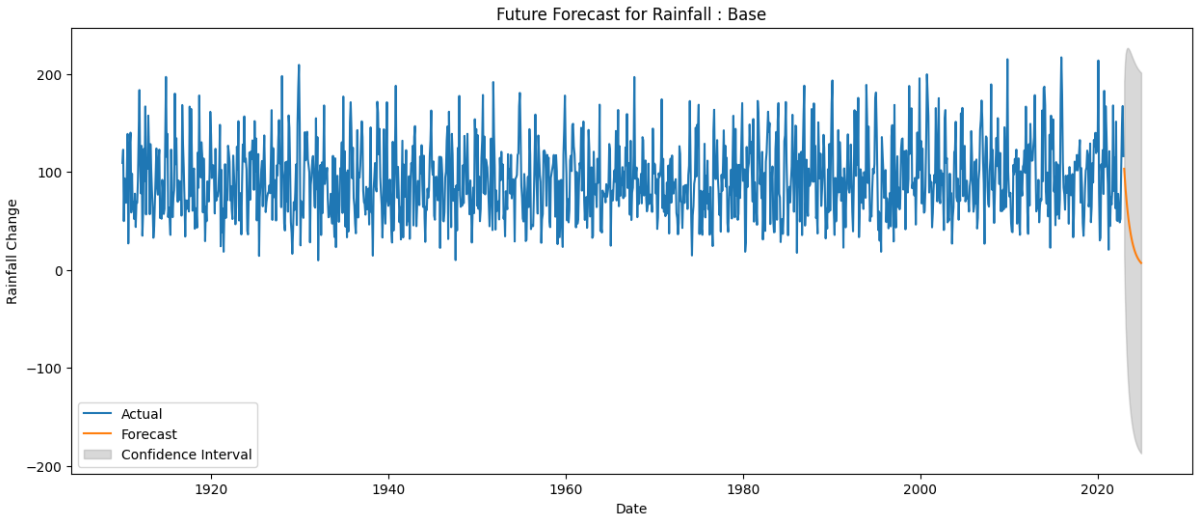


Figure 110: Rainfall Future Forecast

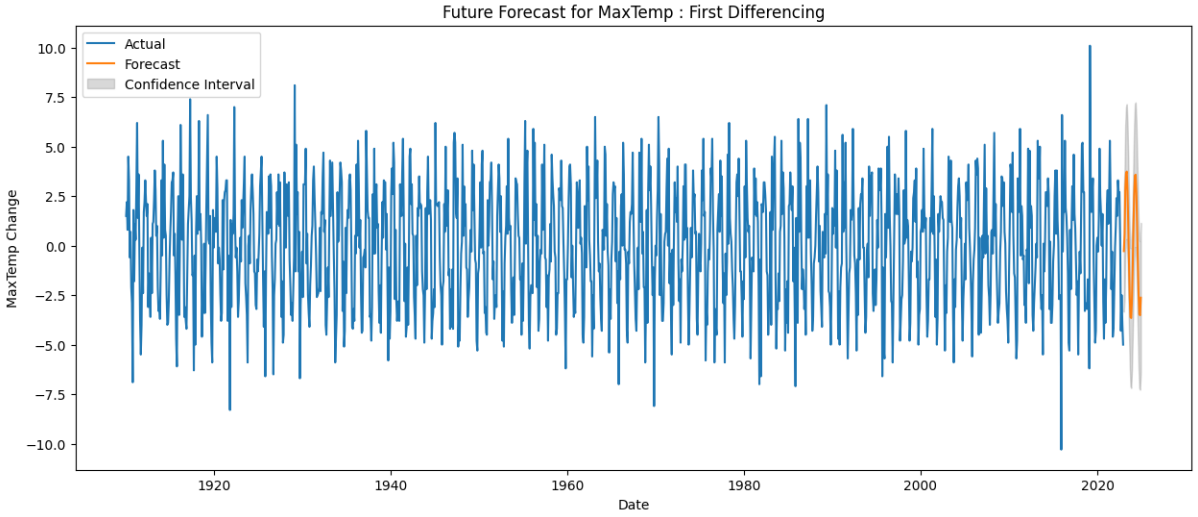


Figure 111: Max Temperature First Difference Future Forecast

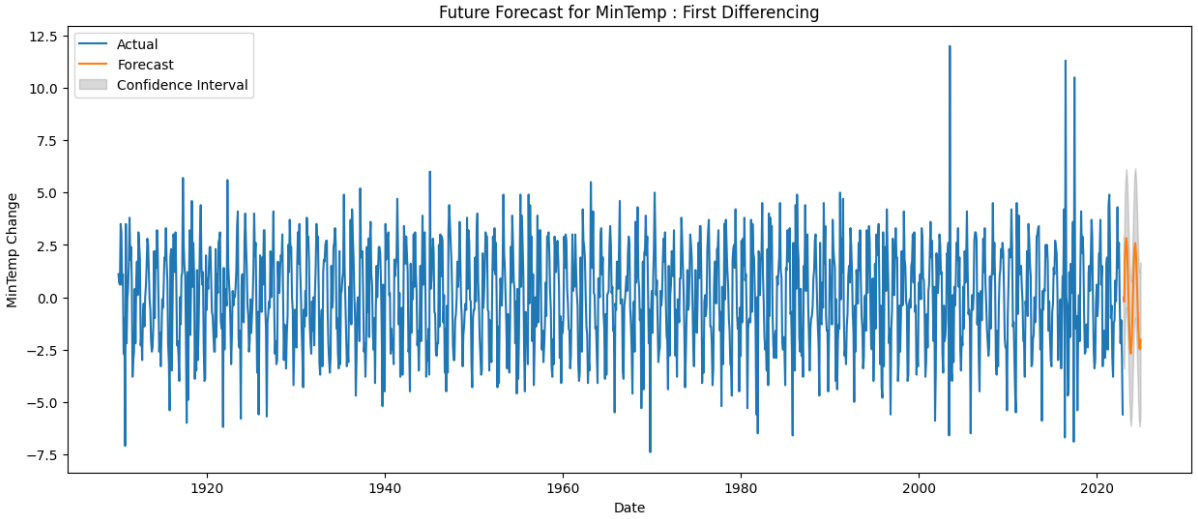


Figure 112: Min Temperature First Difference Future Forecast

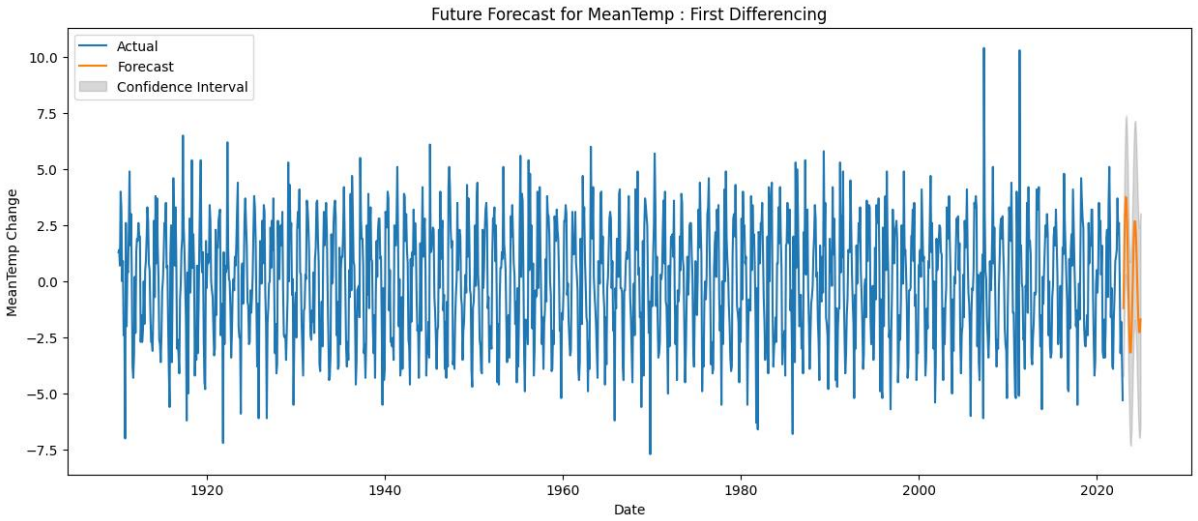


Figure 113: Mean Temperature First Difference Future Forecast

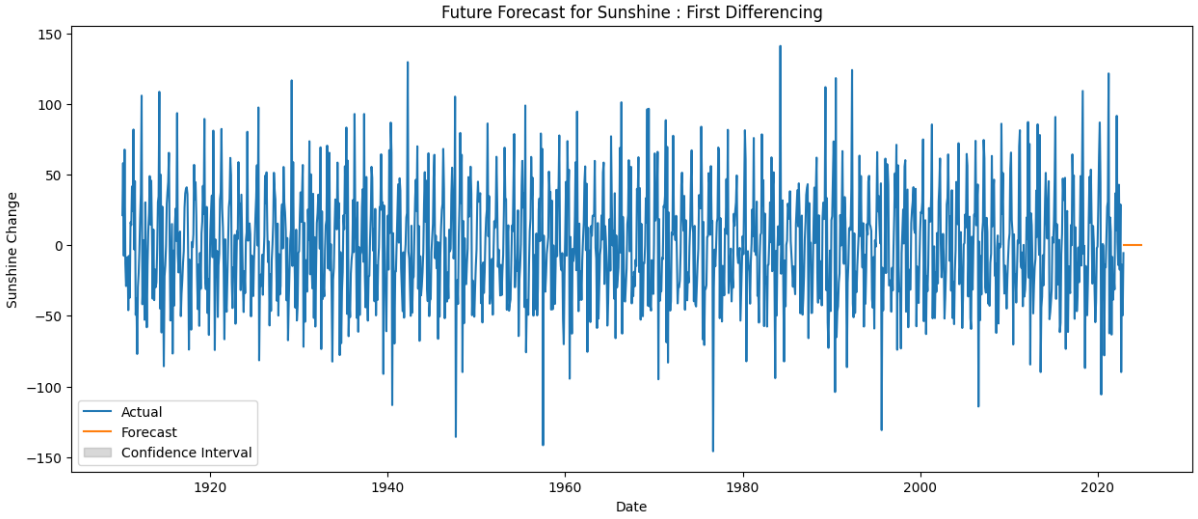


Figure 114: Sunshine First Difference Future Forecast

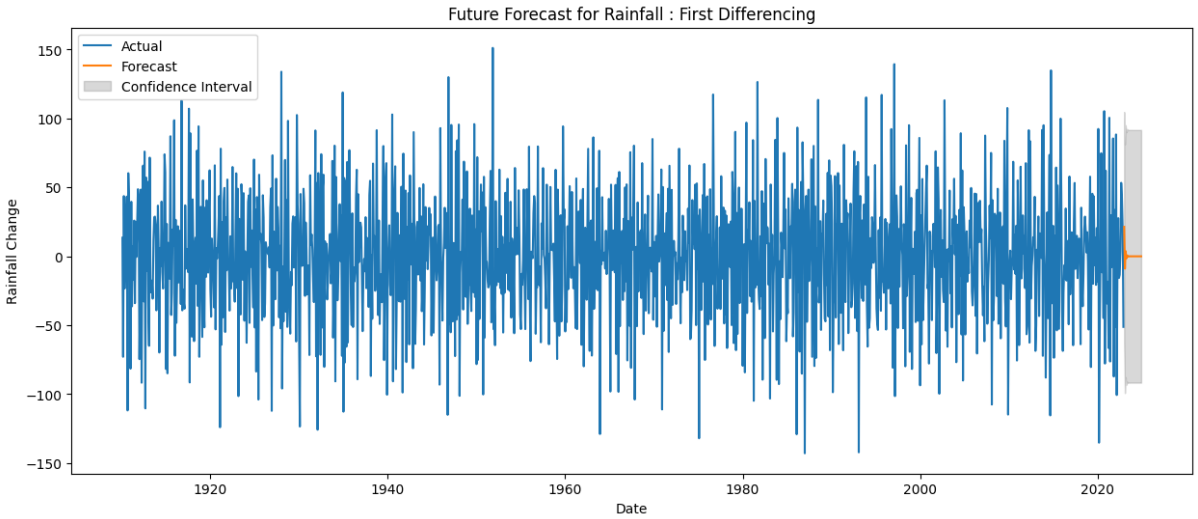


Figure 115: Rainfall First Difference Future Forecast

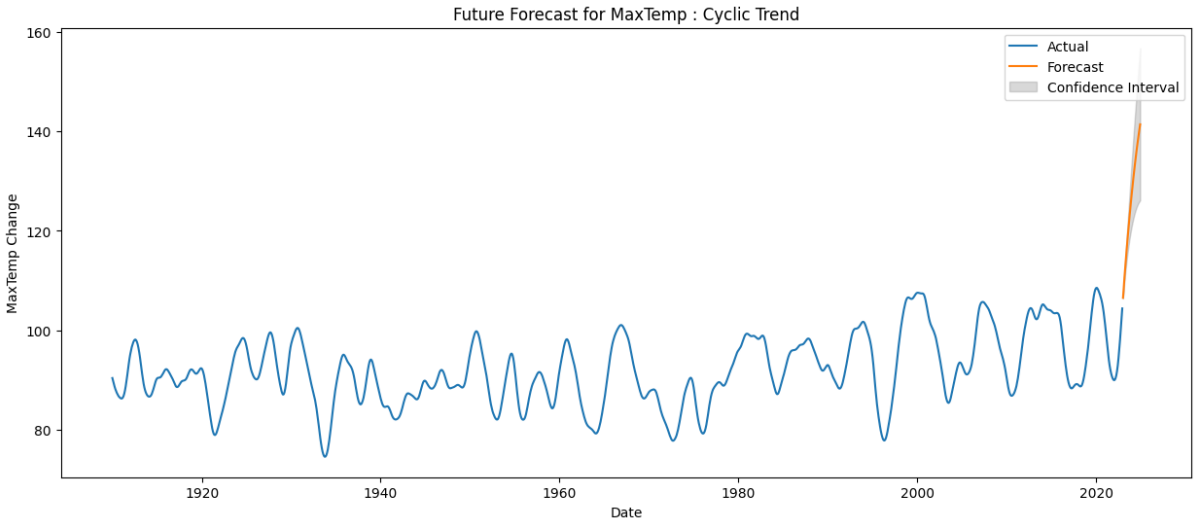


Figure 116: Max Temperature Cyclic Trend Future Forecast

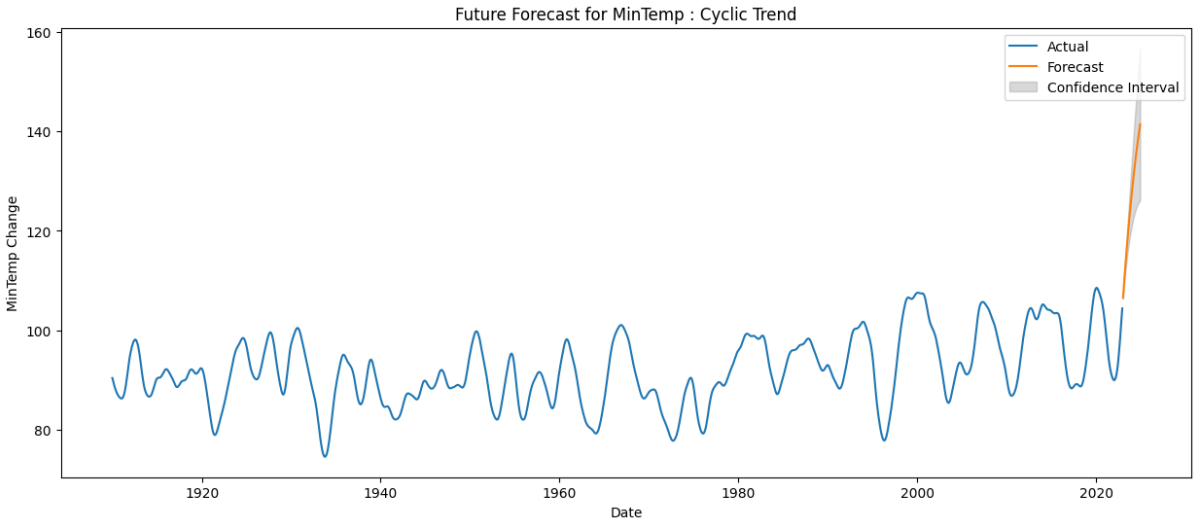


Figure 117: Min Temperature Cyclic Trend Future Forecast

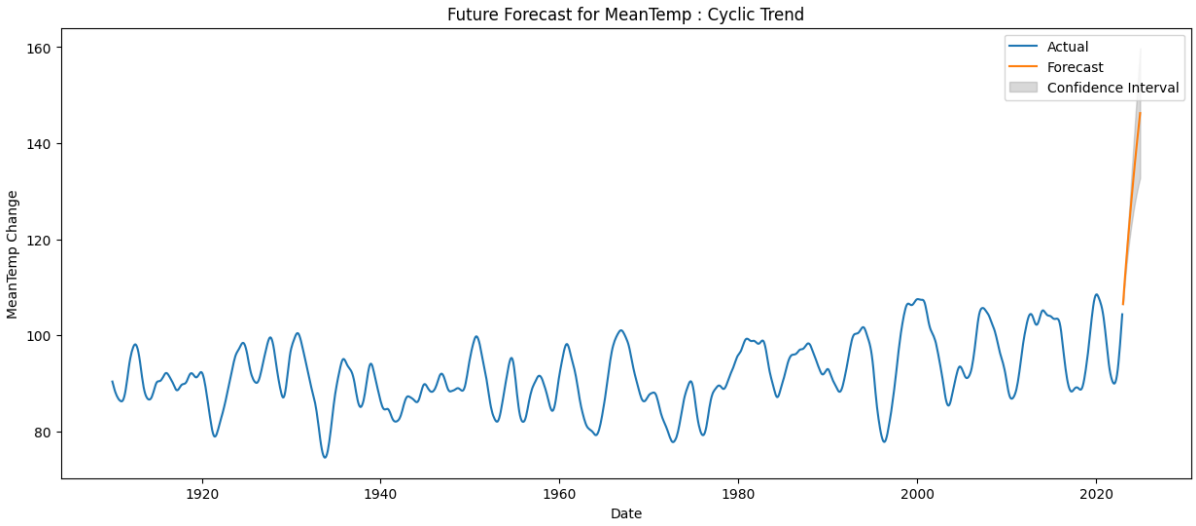


Figure 118: Mean Temperature Cyclic Trend Future Forecast

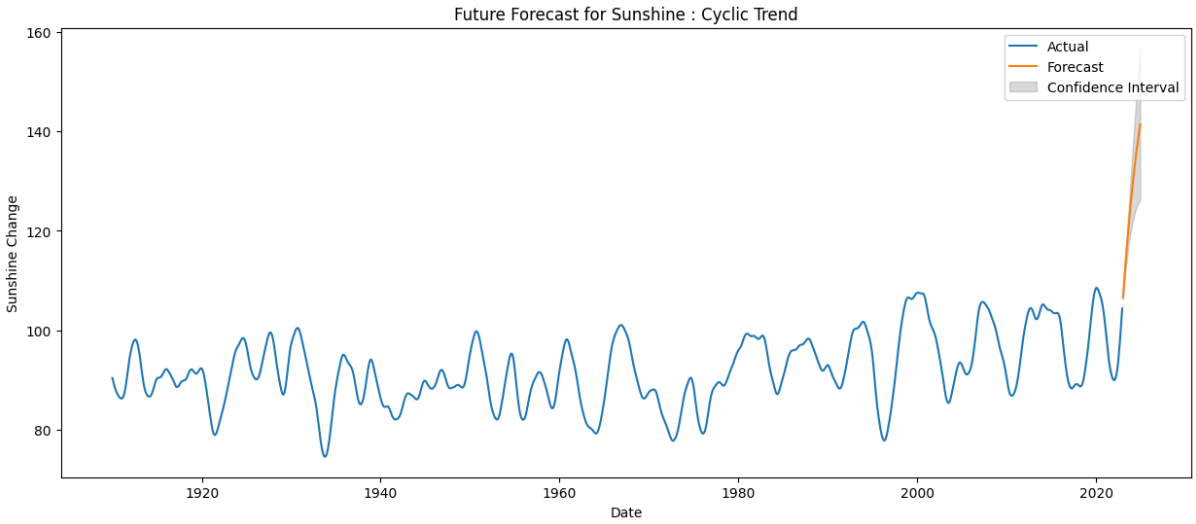


Figure 119: Sunshine Cyclic Trend Future Forecast

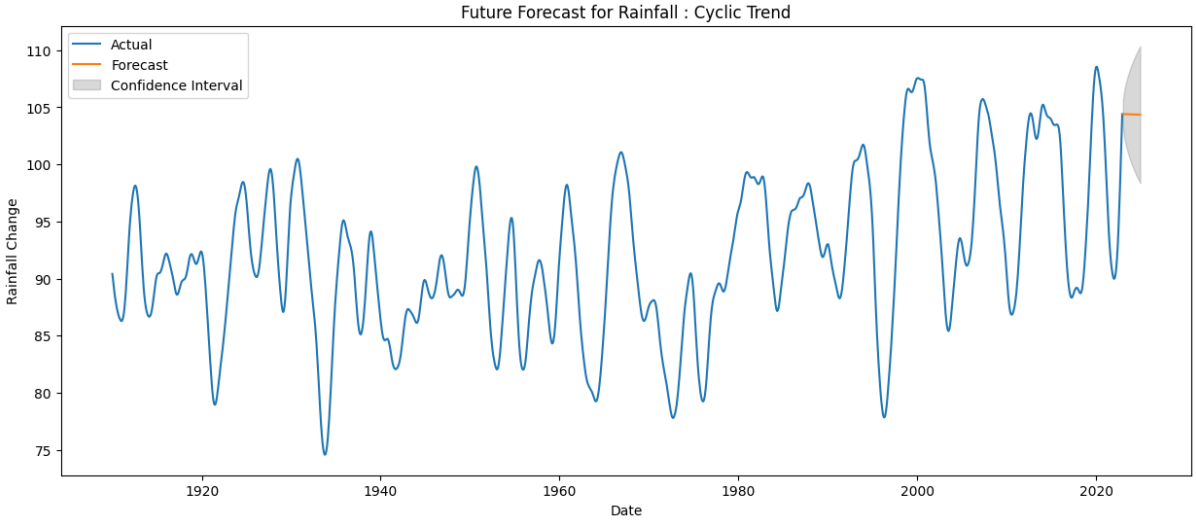


Figure 120: Rainfall Cyclic Trend Future Forecast

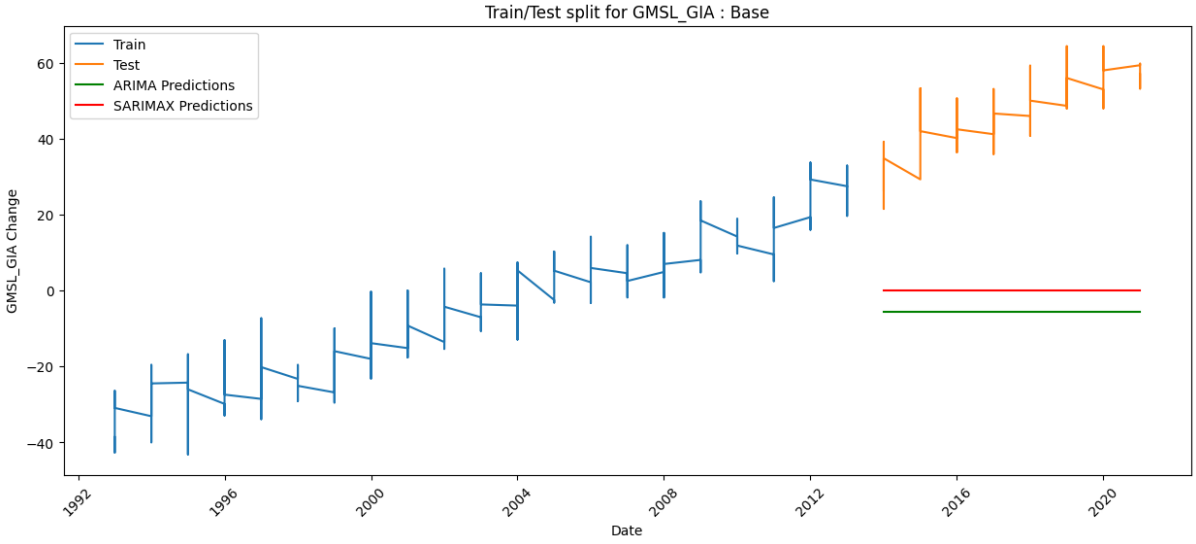


Figure 121: Sea Level Base ARIMA/SARIMAX Model

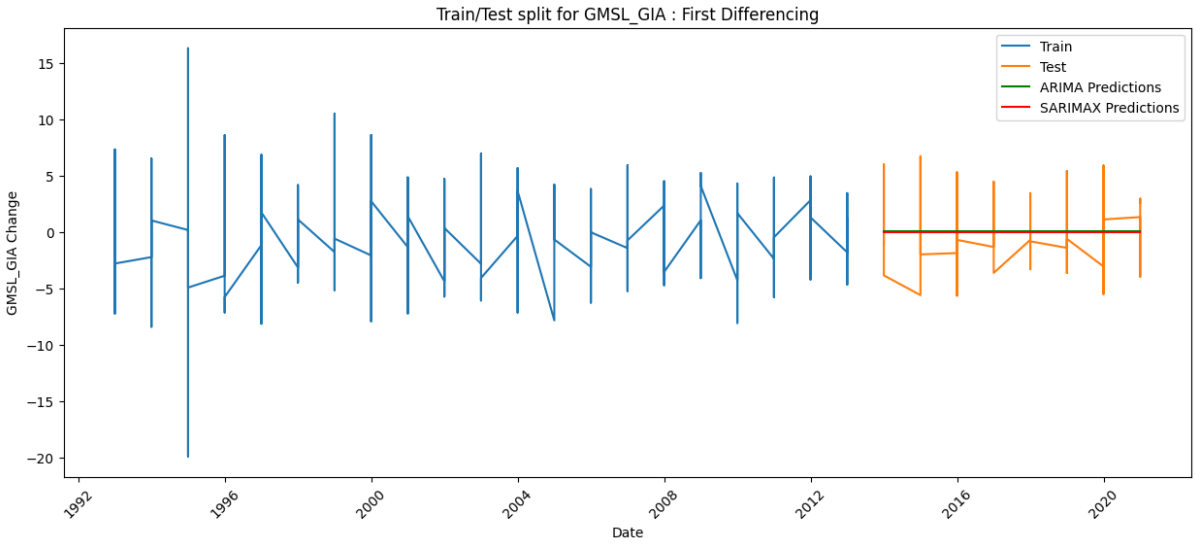


Figure 122: Sea Level First Differencing ARIMA/SARIMAX Model

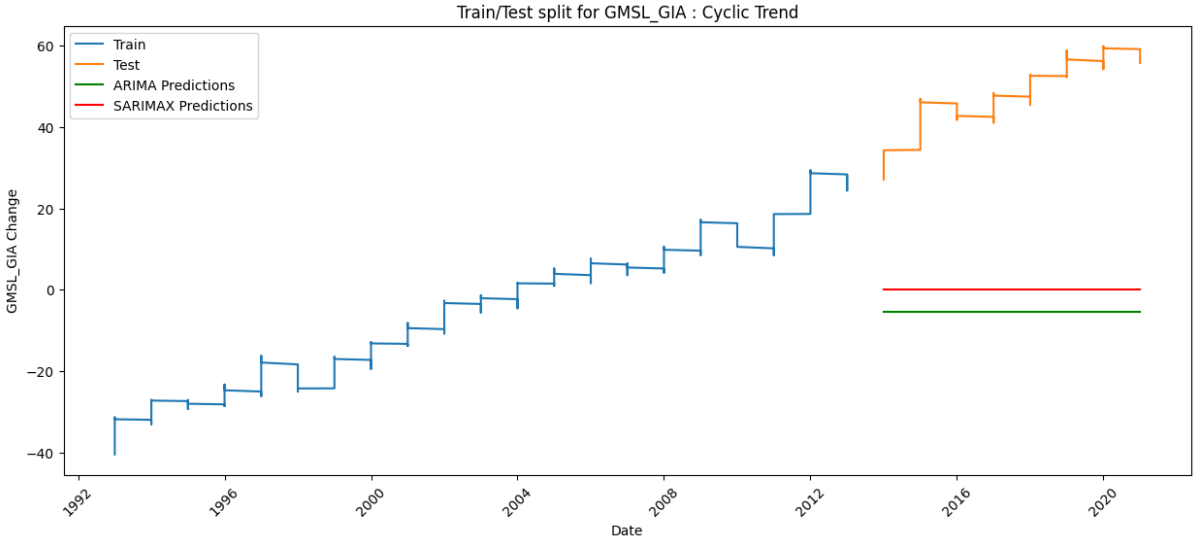
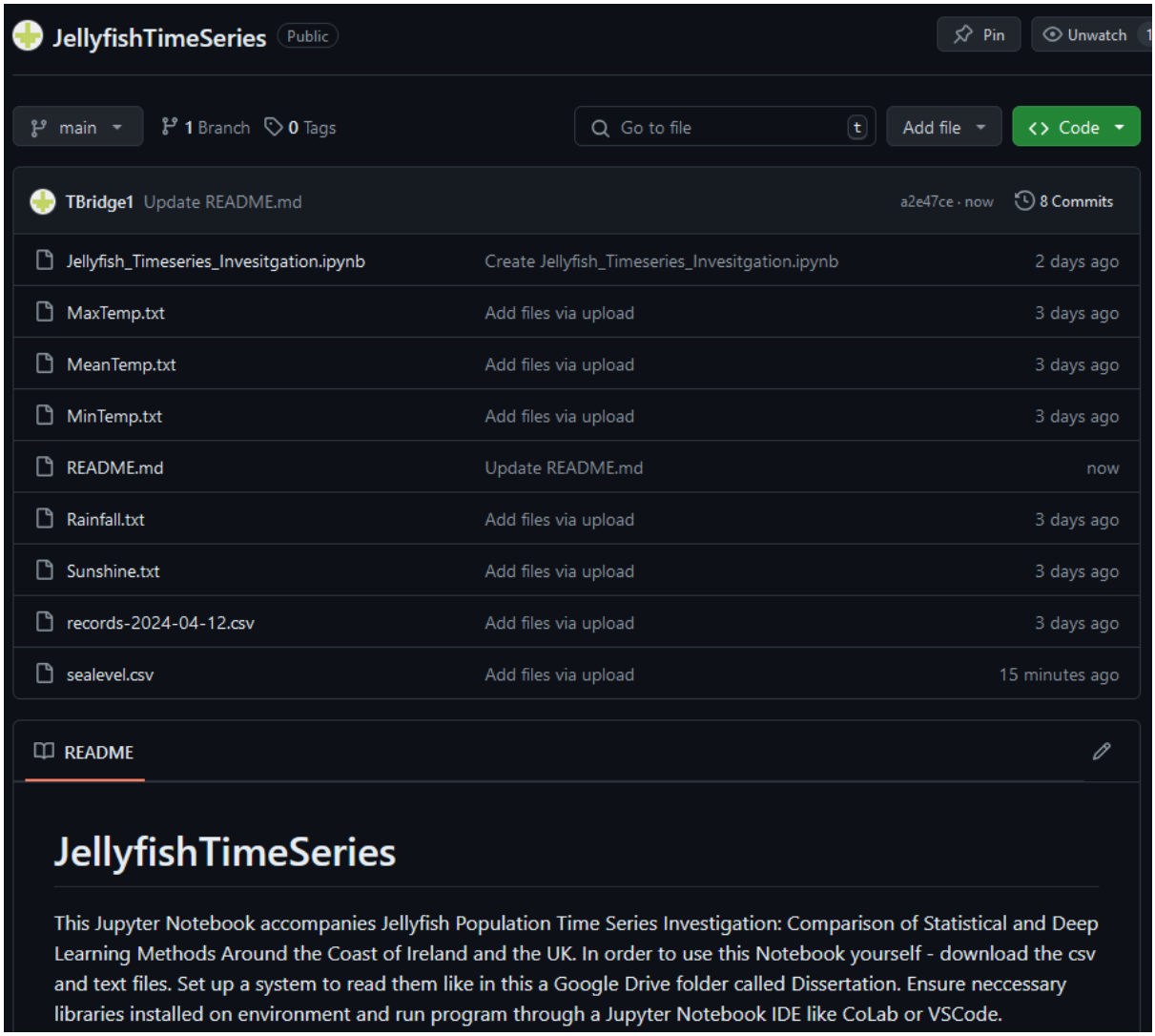


Figure 123: Sea Level Cyclic Trend ARIMA/SARIMAX Model

Appendix C: GitHub Repository

https://github.com/TBridge1/JellyfishTimeSeries


JellyfishTimeSeries
Public
Pin
Unwatch
main
1 Branch
0 Tags
Go to file
t
Add file
Code
TBridge1 Update README.md
a2e47ce · now
8 Commits
Jellyfish_Timeseries_Invesitgation.ipynb
Create Jellyfish_Timeseries_Invesitgation.ipynb
2 days ago
MaxTemp.txt
Add files via upload
3 days ago
MeanTemp.txt
Add files via upload
3 days ago
MinTemp.txt
Add files via upload
3 days ago
README.md
Update README.md
now
Rainfall.txt
Add files via upload
3 days ago
Sunshine.txt
Add files via upload
3 days ago
records-2024-04-12.csv
Add files via upload
3 days ago
sealevel.csv
Add files via upload
15 minutes ago
README
JellyfishTimeSeries
This Jupyter Notebook accompanies Jellyfish Population Time Series Investigation: Comparison of Statistical and Deep Learning Methods Around the Coast of Ireland and the UK. In order to use this Notebook yourself - download the csv and text files. Set up a system to read them like in this a Google Drive folder called Dissertation. Ensure neccessary libraries installed on environment and run program through a Jupyter Notebook IDE like CoLab or VSCode.