Exploring Timeseries Atmospheric Data from the Met Office

Michael Kunov
eg21865@bristol.ac.uk

George Atkinson
rp21039@bristol.ac.uk

Dan Granger
dan.granger.2021@bristol.ac.uk

Ethan Cole
ethan.cole.2021@bristol.ac.uk

Keegan Hill
qr21863@bristol.ac.uk

***Abstract*—The abstract goes here.**

1. Introduction

The dataset involved in our project consists of a partial sample from the ERA5 Global Atmospheric Reanalysis Dataset [1]. ERA5 is produced by the Copernicus Climate Change Service (C3S) at ECMWF and provides hourly measurements of various atmospheric, land and oceanic climate variables. As ERA5 is a reanalysis dataset, the data values are not direct sensor readings but rather learned estimates made by using measured data and filling the gaps with a numerical weather prediction (NWP) model.

An NWP model uses a set of equations to model the movements of atmospheric fluids which when paired with an array of additional governing equations including the Ideal gas law and Newton's laws as well as parameterizations of other atmospheric variables and numerical methods can produce what is widely accepted as being an accurate proxy for weather variables across the locations it captures.

The sample we have been given to work with is significantly smaller than the original set; it's made up of 12 individual CSV files containing sensor data from 12 different locations across the globe, as shown by Figure 1. Additionally, see Table 1 for location names, as well as their corresponding latitude and longitude.

City	Country	Coordinates
Cape Town	South Africa	-33.9N 18.5E
Sydney	Australia	-33.5N 151E
Hyderabad	India	17.36N 78.5E
Kingston	Jamaica	18N -76.8E
New York	USA	40.75N -73.99E
Rome	Italy	41.9N 12.46E
Marseille	France	43.28N 5.39E
Toronto	Canada	43.64N -79.37E
Churchill	Canada	58.76N -94.17E
Calgary	Canada	51.03N -114.06E
London	UK	51.5N -0.1E
Oslo	Norway	59.92N 10.75E

TABLE 1: Dataset Locations

Figure 1: The Locations from which the Sensor Data Originates.[2]

Each CSV file contains a small subset of hourly sensor readings over 38 years between 01-01-1980 and 31-12-2018; including columns for the date and hour the reading was recorded, precipitation, u-wind, v-wind and temperature.

During this project, we look to utilise statistical and machine learning techniques; specifically Recurring Neural Networks (RNNs) and Multi-layer Perceptron Regressors (MLP Regressors), to explore the relationships between different variables and to solve two main tasks. For a given time, we want to predict one of the atmospheric variables, when given all of the other variables using an MLP Regressor, and given past values of a variable we want to predict future values in a similar manner to a weather forecast.

Multiple previous methods have been made to predict weather with machine learning techniques. Salman et all [3] are currently looking to predict current rainfall using real time measurments from the El Nino Southern Oscillation dataset. They look to find links with rainfall and multiple factors including windspeed, wind direction and surface sea temperature, concluding that an RNN can adequately predict rainfall with ambitions to use a CNN to improve upon this finding.

An essential part of this process is fully exploring trends, relationships and correlations across the data. We will look to identify these and examine them using data visualisation methods such as wind roses, dynamic heat-mapping and time series plots.

2. Data Preparation

The dataset consists of hourly time-series data with 4 weather features. Firstly, there is temperature, measured in Kelvin, which is simply the hourly temperature reading. Secondly there is precipitation, measured in metres, which is the hourly rainfall at the location. Finally, there is u-wind and v-wind, both measured in metres per second, each being the wind speed split into horizontal and vertical components, respectively. Positive u-wind means that the wind is travelling from west to east, and positive v-wind means the wind is travelling from south to north, and vice-versa. If the wind speed had been provided as a magnitude and direction, the u-wind and v-wind calculations would be as follows:

$$\text{u-wind} = w_{\text{speed}} \cdot \cos(w_{\text{direction}}) \quad (1)$$

$$\text{u-wind} = w_{\text{speed}} \cdot \sin(w_{\text{direction}}) \quad (2)$$

It is common for extreme data values to be removed before modelling data due to its potential to skew the models predictions and cause overfitting. The dataset was examined for outliers by using box plots for each location, which can be seen in Figure 2. We found a large number of outliers in our data, especially with rainfall, due to when there has been extreme weather events. Also, in locations with more tropical weather conditions, such as Hyderabad, we observed more outliers due to the more extreme weather conditions. Ultimately, we kept these outliers in the data, because the model still needs to be able to predict extreme values. Since there are so many outliers, removing them would not leave us with enough useful data for modelling. Furthermore, it is important that the model can see these extreme weather events, so that it could be able to warn people of future ones, allowing them to take necessary precautions.
Another consideration was the units of the features. The temperature column, which was provided to us in Kelvin, was determined to be a variable of particular significance. In the interest of being able to present the visualizations in an interpretable way, the temperature was converted to degrees Celsius simply by subtracting 273.15 from all values. In doing so, the temperature is also now more centered, with more frequent temperatures being closer to zero, such as 10 degrees Celsius instead of 283.15 Kelvin. This aspect could potentially allow model training to be enhanced by more rationally attributing importance to the temperature.

To determine the geographical locations from which the data in each CSV file originated we used regex to parse filenames for their latitude and longitude. Then we used

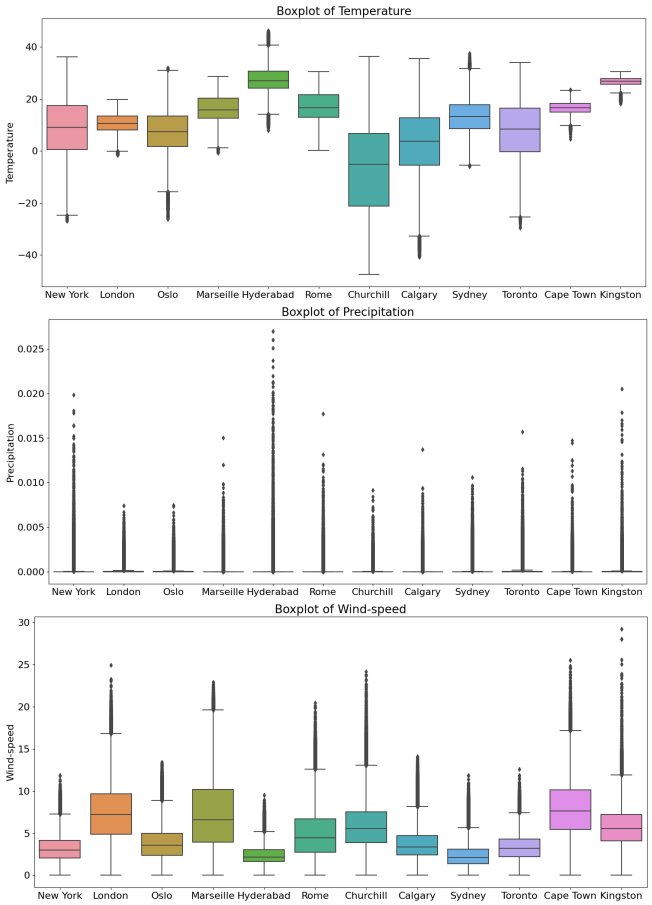


Figure 2: Boxplots showing the distribution across the 12 locations of temperature, precipitation, and wind-speed, top, bottom, and middle, respectively. As can be seen, precipitation has an extremely large number of outliers, which is due to the hourly values mostly being zero. Because of this, the upper and lower quartiles are extremely low, causing almost any rain at all to be an outlier.

online maps to determine the exact geographic location of each dataset. The locations that our data comes from are shown by Table 1.

3. Data Exploration

[KH: violin plot and heatmap need to be swapped around]Data exploration is a critical first step when developing machine learning models. Through the use of different visualisation techniques it is possible to reveal underlying patterns and correlations in the dataset.

Figure 3 shows an example of one the initial visualisations. This instance is the temperature, wind, and precipitation for New York City in 1980. This was completed across all locations for multiple years to draw conclusions about the random nature of the climate from year to year. The purpose of this was to identify any localised patterns or trend. Looking at the localised climate allows patterns

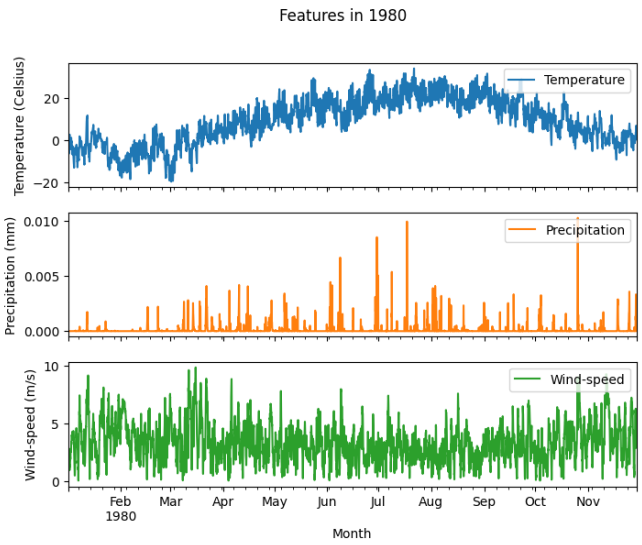


Figure 3: Three line graphs, showing the temperature, precipitation, and wind-speed magnitude in New York City. The top graph shows this for the yer 1980, whilst the bottom graph shows this for the full 38 years.

to be observed such as seasonal temperate or precipitation changes. After this it was possible to delve into some more advanced visualisation techniques to better understand the dataset.

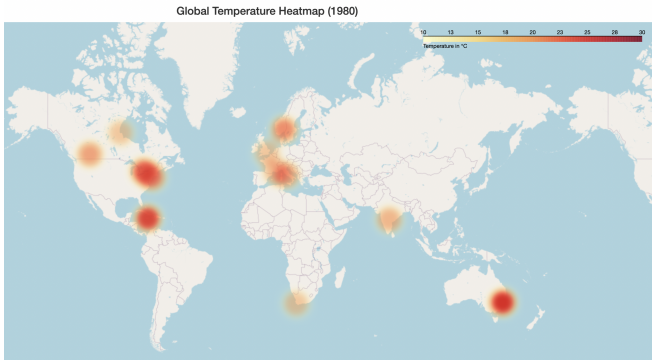


Figure 4: Heatmap produced by a dynamic heatmap for the year 1980 that shows the average yearly temperature for all 12 locations.

Figure 4 shows one of the heatmap for the year 1980. This is one of many produced by a dynamic heatmap that allows the visualisation of temperate over time. This is a nice tool that allows the visualisation of global weather patterns and identification of natural phenomena that might influence localised weather systems. A dynamic heatmap can illustrate how temperatures change across different seasons or years. This aligns with some of the observation made from the violins plots but allows us the delve deeper into the patters short term. Looking at monthly data we notice very similar seasonal trends where the temperature increase in the summer and decrease over the winter. However it can be observed that a city such as Calgary has a much sharper temperature gradient that a tropical location such as Hyderabad which indicates not only that the temperature has greater variation but also changes faster. Furthermore it appears that coastal location tend to warm more slowly than locations that are surrounded by land. Specifically Sydney, Australia saw slower overall warming that Calgary, Canada despite maintaining a higher average and peak temperature all year round and This is important when considering machine learning models as applying the same model to cities with substantially different climates may yield unrepresentative results.

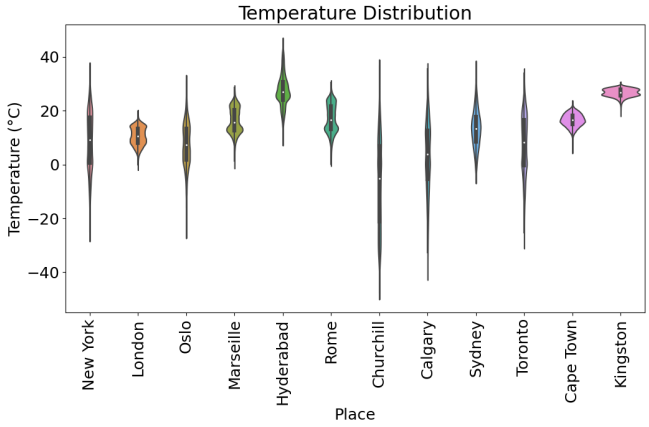


Figure 5: A violin plot comparing the distributions of temperature, in Celsius, in each of the 12 locations. As can be observed, New York stays relatively hot all year round, whereas Marseille has large fluctuations in temperature.

A violin plot is a visualisation tool that illustrates the probability density of the data at different values, with the width of each ”violin” indicating the frequency of occurrence of temperatures at each level. This serves as a valuable insight to the temperature ranges, median temperatures, temperature extremes and data skewdness. Violin plots are invaluable in the exploratory data analysis phase of machine learning. They allow for the assessment of data distribution properties, identification of outliers, and comparison between different locations within the dataset.

Cities like Churchill and Calgary exhibit wide temperature ranges, showcasing significant seasonal variation, while tropical locations like Cape Town and Kingston demonstrate narrower temperature bands, indicating more stable year-round conditions.

The plot’s median markers can immediately indicate locations that higher temperatures all year round. Cities like Kingston tend to have higher median temperatures, contrasting with colder cities like Churchill.

The extremes in the plot, seen in the tails of each violin, highlight cities like Churchill experiencing severe cold, whereas cities like Kingston endure higher temperatures at the other extreme. These tails are crucial for understanding the full range of climatic conditions experienced.

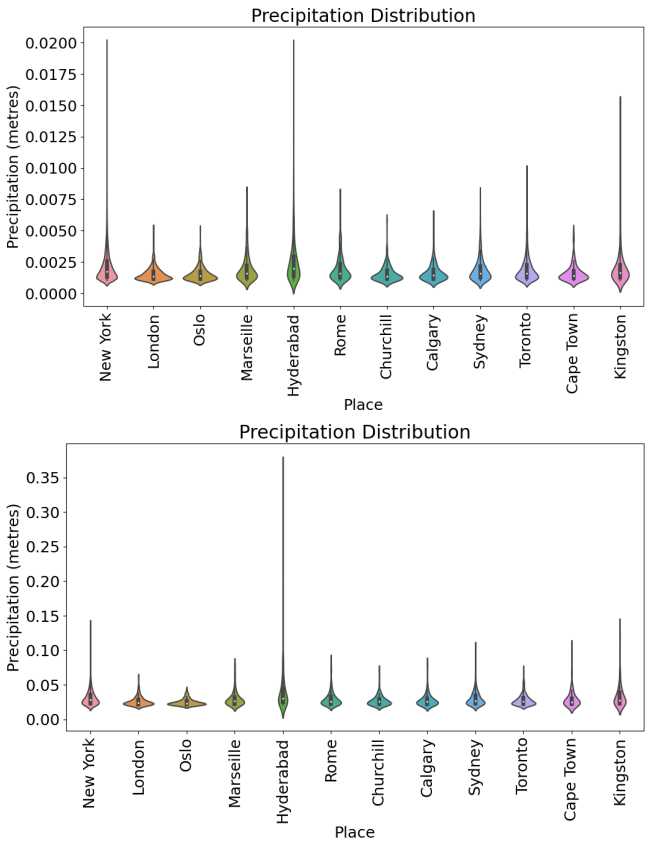


Figure 6: Violin comparing the distributions of precipitation both hourly and as a daily aggregate, top and bottom, respectively. This is measured metres, in each of the 12 locations. Data values below 0.001 metres for hourly, and 0.02 metres for daily. This is because the data is largely skewed by an extremely large number of zero-valued time-steps.As can be observed, all locations have extremely low mean values with very little deviation, but a lot of outliers. This is a testament to how sporadic rainfall is. Furthermore, it can be observed that Hyderabad experiences the most extreme rainfall, which makes sense due to its tropical climate.

Although there is a focus on temperature, it is also important to consider the other features. Firstly, considering precipitation, from Figure 6 we can see that the distributions for most cities are relatively narrow, indicating less variation in precipitation amounts during individual hours throughout a year. The peaks are sharp for most cities, suggesting that low precipitation rates dominate hourly data.

The medians are low across all cities, typically close to zero, reflecting the rarity of high hourly precipitation. Most of the timesteps are dominated by little rainfall. This is typical for many urban climates where intense rainfall is infrequent but possible.

Cities such as Hyderabad, New York and Kingston exhibit longer tails in their distributions, pointing to occasional heavy rainfall within single hours. However, most cities do not show significant extremes on an hourly basis. This is aligned with what might be expected from these locations. Kingston and Hyderabad being tropical locations experience very intense bursts of rainfall and can be susceptible to natural disasters such as hurricane which bring a huge amount of rainfall in a very short amount of time. Similarly, New York being situation on the east coast of the USA is susceptible to aggressive storms coming in from the Atlantic Ocean.

Figure **??** shows that some cities like New York show potential for high precipitation totals in a day, indicated by upper tails reaching up to 0.35 meters. This could be due to underlying natural phenomena that might contribute to certain fluctuations and extremities is also of valuable insight. An example of this could be a natural occurring weather cycle. El Nino and La Nina are two opposing climate patterns that break what are considered normal conditions. These phenomena are part of the El Nino-Southern Oscillation (ENSO) cycle. El Nino and La Nina can both have global impacts on the weather conditions specifically rainfall. New York is one such locations that is greatly affect by this particular weather phenomenon and could be the reason why it sees such extreme outliers when it comes to rain. Natural phenomena are an import thing to consider when developing machine learning models as they can have completely different patterns and may cause unrepresentative predictions.

Coastal cities or those known for wet climates, like London, might show broader distributions, whereas drier cities like Hyderabad and Rome maintain narrow violins even on a daily scale, indicating consistent low daily precipitation.

Comparing these distributions helps understand the climatic conditions of each city—coastal vs. inland or tropical vs. temperate.

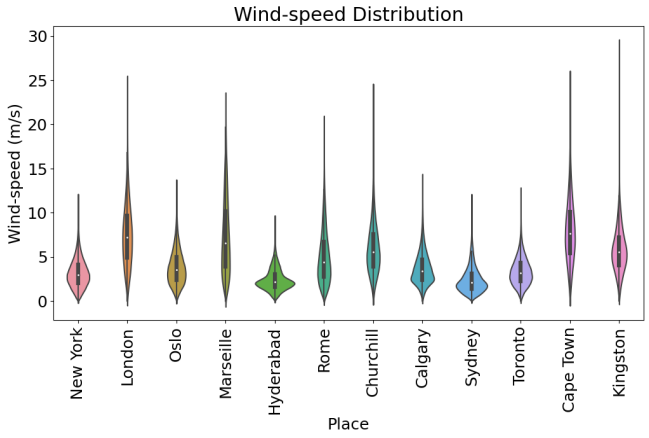


Figure 7: Violin plot showing the distribution of wind-speed magnitude, in metres per second, in each of the 12 locations. As can be observed, values are in a similar range across each location, but with some differences.

New York, Hyderabad and Sydney show moderate wind speeds with similar ranges but but Hyderabad has more of a data skew towards lower windspeeds, suggesting more consistent but also lower windspeeds.

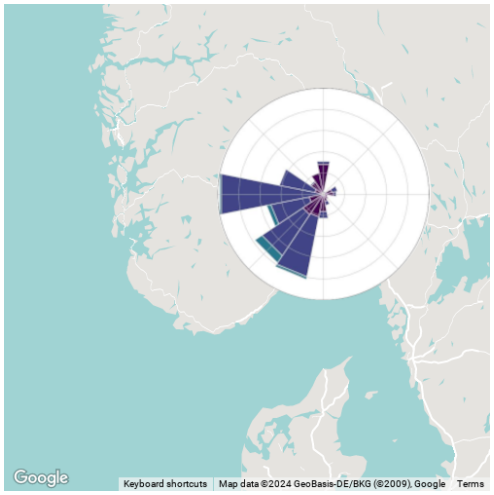


Figure 8: Windrose displayed on map of Oslo [4]

Kingston stands out with a high upper extreme, potentially due to its less obstructed location on the coast. In fact Coastal cities like Cape Town and Kingston generally show moderate to high windspeeds with a great amount of variability. Coastal areas are subject to sea breezes which cause more wind extremes. Being coastal they also have very little protection from these sea breezes so experience the greatest range of windspeeds. In contrast, inland cities like Calgary and Hyderabad might experience less variation due to their inland nature and surrounding topography such as hills and mountains that protect them very extreme wind. Wind measurements in u and v directions can be complicated to understand and interpret. As an alternative to this, the data can be converted to speed and bearing, then visualised as a polar bar plot or "windrose" as seen in figure 8. This is easier to comprehend as it can be read to be the direction the wind is blowing from, where the bars represent the distributions of the strength of this wind. The plot shows a windrose for Oslo over a map of the region. Here it is easy to see a connection between the average prevailing wind and the North sea on the south east.

4. Data Pre-Processing for Modelling

Now, to leverage our deeper understanding of the data to produce models that can make useful predictions on the data. There are a variety of ways machine learning can be applied to a weather dataset. For example, we could train a model that uses all other features (e.g. precipitation, wind) to predict a target feature (such as temperature). Alternatively, we could use past weather data to attempt to produce a forecast of future weather data. Furthermore, this could come in the form of a daily forecast, weekly forecast, or something more long term like for predictions in climate change. Ultimately, we decided do explore the usage of both of the discussed model types, and compare how different model types suit different applications.

Machine learning model performance can be highly influenced on how its input data is formatted; wrangling our data into new formats can enhance our models ability to recognize patterns and make better predictions. For example, cyclical encoding is a method that allows models to better understand the repeating nature of time series data. By using our prior knowledge about daily and yearly periodicity, we can help the model by giving it data that captures this frequential pattern. To do this, we converted days and hours into sine and cosine signals with the following equations:

$$\text{day}_{\text{sin}} = \sin(\frac{2\pi t}{t_{day}}), \tag{3}$$

$$\text{day}_{\text{cos}} = \cos(\frac{2\pi t}{t_{day}}), \tag{4}$$

$$\text{year}_{\text{sin}} = \sin(\frac{2\pi t}{t_{year}}), \tag{5}$$

$$\text{year}_{\text{cos}} = \cos(\frac{2\pi t}{t_{year}}), \tag{6}$$

where t is the POSIX time in seconds, t_{day} is the number of seconds in a day, and t_{year} is the number of seconds in a year. As a result, the model should be stronger at spotting important frequential features in the data, such as hourly, and seasonal trends. Now, both days and years can be interpreted as a combination of sine and cosine signals, such that one wavelength is equal to 24 hours, and 365.25 days respectively.

Also, the data was split into training, testing, and validation datasets, with corresponding ratios 80%, 10%, and 10%, respectively. Usually, this involves randomly allocating data from the dataset in the described ratios. However, with our data being time-series, it does not make sense to take away its sequential property. For this reason, the data was split whilst retaining its chronological order, with the first 80% being used for training data, the next 10% being used for validation, and the final 10 % being used for testing. By doing this, it emulates a real-world scenario in which future data is unseen, and allows the model to make predictions on the future, as it would in real life applications.
Then, we used sklearn's StandardScaler to scale the feature values using the following equation:

$$z = \frac{x - \mu}{\sigma} \tag{7}$$

where μ is the mean of the training data, and σ is its standard deviation. It is important we do this after splitting the data, and using the mean and standard deviation of the training dataset, because otherwise outliers from the test and validation set could affect μ and σ. Ultimately, this would influence the model training, causing some bias in its predictions. By scaling the data, it allows the model to attribute equal importance to each feature by putting them on a similar scale. Otherwise, the model may give more weighting to higher-valued features such as temperature, and cause the model to overfit. Also, we didn't scale the target

feature, as it meant any output was more interpretable, and it should make no difference to training. The violin plot in Figure is a great way of displaying the effect of scaling the data [UP TO HERE IN REWRITE OF MODEL SECTION].

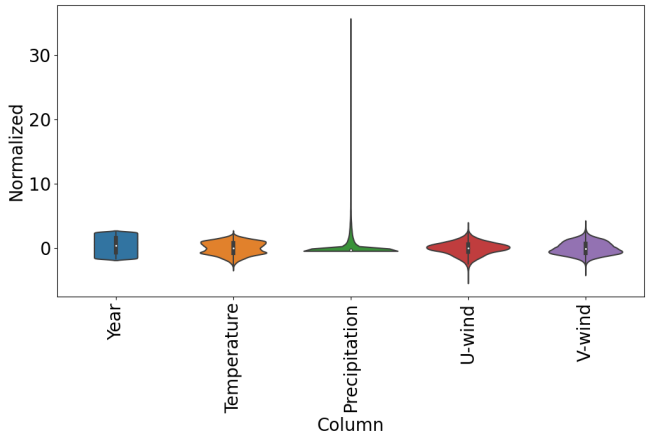


Figure 9: A violin plot showing the distributions of each feature once scaled.

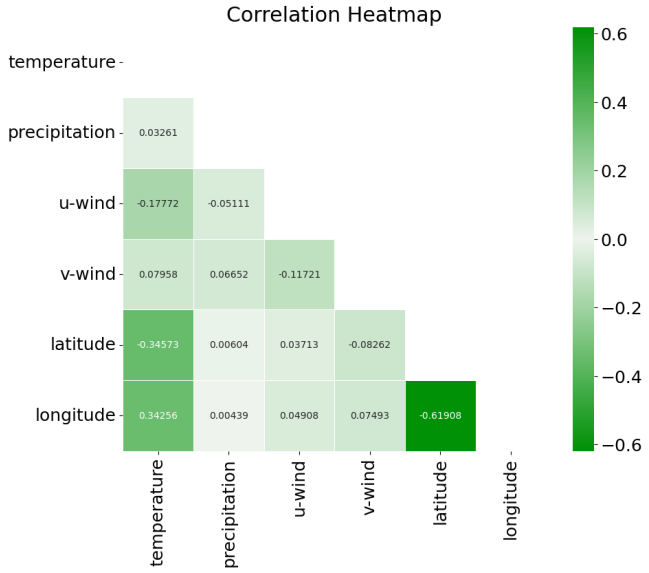


Figure 10: A correlation matrix showing relationships between our feature variables, as well as latitude and longitude. Notably, latitude and longitude have a strong negative correlation with temperature, which is simply suggesting as you go further north or east, the colder it gets. More interestingly, temperature seems to be influenced by u-wind, as the more windy, the less hot it is.

A correlation matrix was used to examine relationships between the feature variables. Since correlations are both negative and positive, centered approximately around zero, the color map used is uniform to ensure that high correlations can be seen easily, whether negative or positive. Figure 10 shows these results, and demonstrates that most

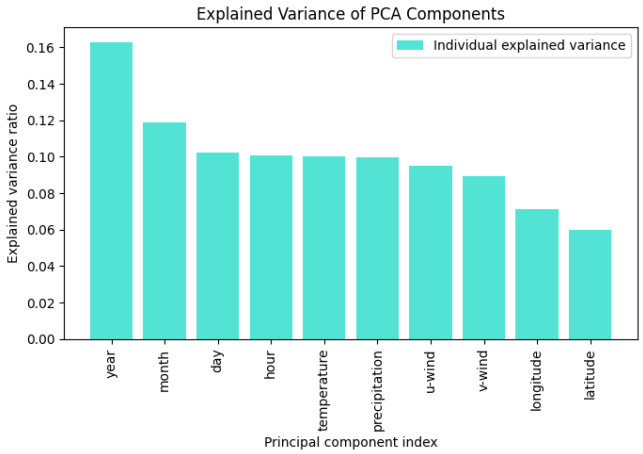


Figure 11: Bar chart showing explained variances of each feature.

of the features are relatively uncorrelated, however, it does confirm the influence of latitude and longitude on temperature.

Next, we used principal component analysis (PCA) to explore the most influential features in the dataset. PCA is a statistical method that aims to reduce the dimensions of a dataset whilst retaining as much variance as possible. It does this by looking for orthogonal features, as these ones are uncorrelated and contain more unique information about the dataset. To do this, we used sklearn's built-in PCA function on the feature data, and then we plotted the explained variance of each feature on a bar chart, which can be seen in Figure 11. From this, we can see that no feature stands out in explained variance, and for this reason we decided to keep all features in for training because they are all fairly equally descriptive.

5. Data Modelling

As discussed, there are a number of different routes we can take for modelling on the dataset. Ultimately, we decided to explore 2 different paths, exploring both weather forecasting, and predicting a missing feature. Modelling requires splitting the dataset into input features, **X**, and target features, **y**. For predicting a missing feature, this simply meant making a target dataset with just one feature, and dropping this feature from the original dataset. On the other hand, for weather forecasting, this meant making a new column which is the same as the target feature, but with its values shifted **N** time-steps, depending on how far in the future we are predicting. Also, shifting the rows down means that there will be **N** rows containing a NaN value, which we discard because they are not useful for predictions.

5.1. Forecasting Future Values

The models in this section focus on predicting the temperature, as it is both useful and interesting. Knowing the future temperature can be beneficial for a range of usages, from planning a day trip to the beach, to a farmer choosing a harvest day. At this point, the models aim for predictions 6 hours in the future, to give ourselves a reasonable chance of getting useful results in a shorter time frame. This meant making a duplicate of the temperature column and shifting it down 6 rows to create a new target column.
Finding benchmarks is another crucial step for any machine learning task. It is important that there is a standard to compare our model with, so that there is some measure of success. The first, and simplest, benchmark we used is an averaging model. This model makes a prediction that assumes the target feature, at a particular time-step, will be equal to the mean average of the target feature on that specific time-step in other years. Next, we utilized the persistence model for a slightly more sophisticated prediction. This model simply predicts that the weather on one day will be exactly the same as the day prior to it. Finally, we used a rolling model which assumes that the weather will remain the same for the next 6 hours. These simple models provide a baseline performance to compare our machine learning models with, and the results can be seen in 4.

When training a machine learning model, the hyperparameters which it is initialized with can bear a great impact on how the model performs. For this reason, GridSearchCV was used to undergo a search for the best hyperparameters for the model. During a grid search, you give the model a list of options for each hyperparameter, then it tries every possible combination and assesses its performance with mean squared error. From this, we note the best hyperparameters and then complete further analysis of model performance. Also, it is important that we properly interpret the results of each model. To analyze the performance of our model, we will look at the learning curves, checking for divergence, in addition to looking at loss scores and predictions on the test set.

To begin with, we trained a multi-layer perceptron regressor (MLPRegressor) to predict the temperature 6 hours ahead of the current time-step. We chose to use MLP due to its relative simplicity to train, and its suitability for modelling complex non-linear relationships between features, such as meteorological feature data. To train the MLPRegressor, we used Sklearn and GridSearchCV to find the best hyperparameters for training. The search space used for the grid search can be seen in Table 2

After finding the optimal hyperparameters, the model was initialized with

- Hidden layer size of (50,50)
- ReLu activation function

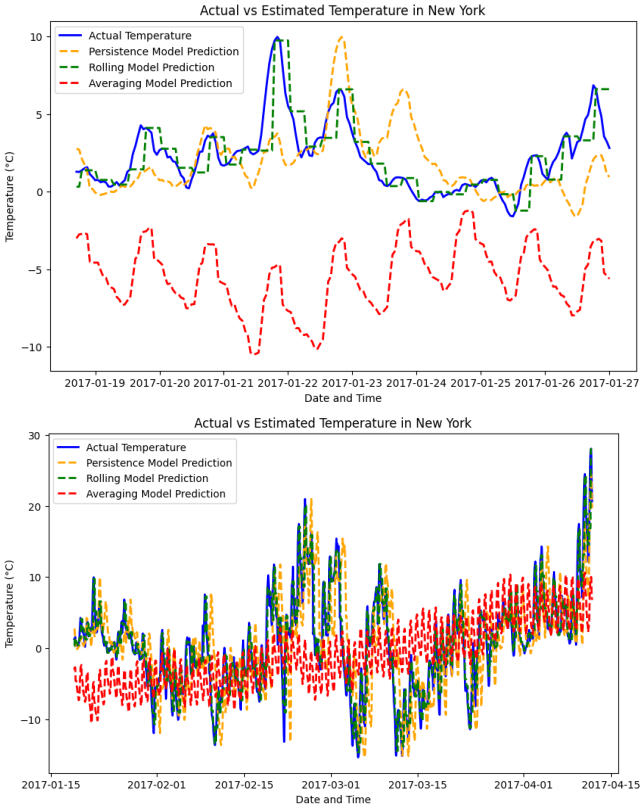


Figure 12: Graph showing our benchmark model performance over about 1 week and 3 months top and bottom, respectively. Clearly the persistence and rolling models have predictions closer to the ground truth, but the averaging model is able to show a general trend over a longer time period.

MLPRegressor's GridSearchCV Setup		
Hyperparameter	**Search Space**	**Description**
Hidden Layer Size	(25,25,25), (50), (50,50), (100)	Defining how many nodes each hidden layer should have. Larger/more hidden layers require more time per epoch.
Activation Function	tanh, ReLu	The activation function for the neurons.
Alpha	0.0001, 0.05	Strength of the L2 regularization
Solver	SGD, Adam	The optimizer used for altering node gradients.
Learning Rate	Constant, Adaptive	Defines if/how the learning rate should decay during training.

TABLE 2: A table showing the search space used for MLPRegressor with GridSearchCV.

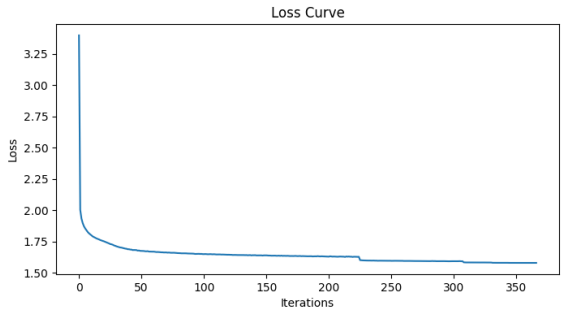


Figure 13: Graph showing loss at each epoch of training, exhibiting clear divergence.

- SGD solver
- Alpha value of 0.05
- Adaptive learning rate

and was allowed to run for up to 200 epochs but had early stopping enabled. The training required all 200 epochs to finish, and gave us a significant improvement on the bench-marking models. Our MLPRegressor achieved a mean square error of 4.82, and a mean absolute error of 1.74. The lower mean square error means that the model makes less large mistakes in its predictions, whereas the higher mean absolute error means our predictions are frequently further away from the true temperature.

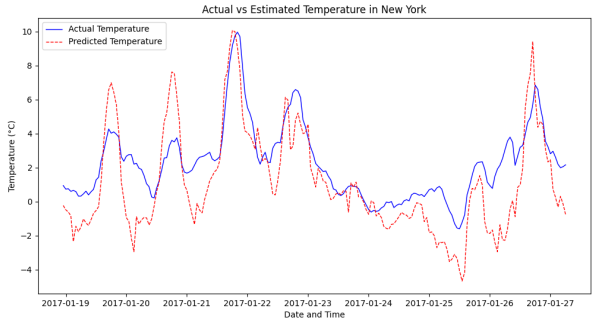


Figure 14: Graph showing predictions from MLPRegressor over the course of a week in New York. The results show that the model has learnt about the periodicity of the temperature and for the most part can get a close estimate of the peaks and troughs, but can be liable to underestimating and overestimating.

Subsequently, we decided to train a random forest regressor due to their versatility to different data modelling tasks, and robustness to counteract overfitting. Furthermore, using a random forest regressor allows us to determine the relative importance of each feature. We used Sklearn to initialize a RandomForestRegressor to see if we could improve our predictions. Using GridSearchCV with the search space in Table 3 to find the best hyperparameters, we then initialized the model with the following setup:

- 50 Estimators

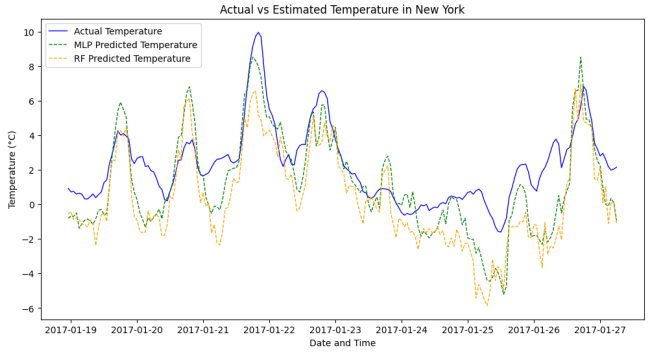


Figure 15: A graph comparing predictions of MLPRegressor and RandomForestRegressor. It can be seen that they both follow a similar trajectory, and tend to make the same mistakes.

- Square root number of features
- Maximum depth of 30
- Minimum samples split of 5
- Minimum samples leaf of 2

[TALK ABOUT RESULTS WHEN GOT]

RandomForestRegressor's GridSearchCV Setup		
Hyperparameter	**Search Space**	**Description**
Number of Estimators	50, 100, 200	Defining how many decision trees to include in the bagging ensemble.
Maximum Features	Auto, Square Root	The number of features to consider at each split in the trees.
Maximum Depth	10, 20, 30, None	The upper limit of how long a branch of the trees can be.
Minimum Samples Split	2, 5, 10	The minimum number of samples required to make a new split.
Minimum Samples Leaf	1,2,4	The minimum number of samples for a leaf node to have.

TABLE 3: A table showing the search space used for RandomForestRegressor with GridSearchCV.

After some moderate success with the discussed models, there were still desirable improvements. Despite its simplicity, the rolling benchmark model still manages to give more consistent estimates. For this reason, we experimented with using recurrent neural networks (RNNs), a more complex mode, for making predictions. RNNs have in recent years become a popular choice for processing time series data as they exhibit a more long term memory of data, which allows it to pick up more complex patterns. Training an RNN meant that the data needed further pre-processing because of the input format of RNNs. For this reason, we rearranged the data into data windows. A data window is defined by its total width, input width, label width, and offset. The input width is the number of timesteps fed as input into the RNN, the label width is the number of desired outputs, and the offset is how many time-steps after the input the output starts. Finally, the total width is equal to one less than the sum

Model	Mean Absolute Error	Mean Square Error
Averaging	4.09	28.43
Perseverance	3.45	20.86
Rolling	1.59	5.95
MLPRegressor	1.74	4.82
RandomForestRegressor	1.78	5.10
RNN	**1.32**	**3.06**

TABLE 4: A table showing the mean absolute error and mean square error for each model, including benchmarks and machine learning models. As can be seen, our rolling model will be the toughest comparison for the machine learning models. We observed that our MLPRegressor is the best performing in Mean Square Error while no other models beat the rolling benchmark in mean absolute error (yet).

of these values (demonstrated in Figure 16a). To do this, we generate indices for the window, and use Tensorflow's timeseries_dataset_from_array() on each window to prepare it for input into the RNN. Our RNN is defined with the following structure:

- LSTM layer of size 32, with return_sequences enabled
- Dense layer of size 1
- Early stopping enabled
- Adam optimizer

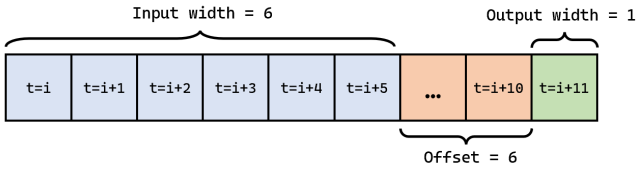
On each epoch, the model iterates through each window, and uses mean squared error to calculate the loss of its predictions. The model is allowed to run for a maximum of 50 epochs, and then is used to produce predictions on the test set. The model had finished training after 30 epochs and achieved a mean absolute error of 1.32, and a mean squared error of 3.06.

5.2. Predicting the Missing Feature

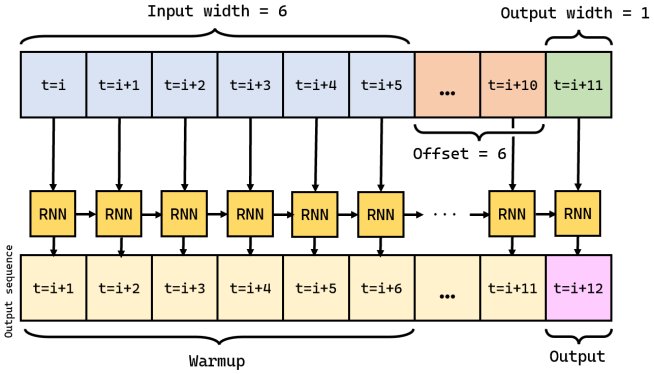
Another part of our work on modelling involved answering a different question; how can we accurately predict the value of a single variable at a given time, when only given the others? In our case, we again chose to work on predicting the temperature value at a specific time in the dataset when provided with only the u-wind, v-wind and precipitation.

This task led us back through similar steps as in our RNN forecasting work, with the end goal of training another neural network to predict temperature. Our previous findings from sections 3 and 4 were applicable here once more; from our PCA reduction investigation we knew that there were no variables that stood out as having more significant explained variance so all features were kept, and we kept our cyclical encoding of time variables as well as our application of sklearn's StandardScaler.

There's a significant difference in the data that we provide the single variable prediction models, compared with the forecasters; in the RNN for example, we are required to



(a) Diagram demonstrating the structure of a data window, which is given as input into the RNN. In this case the total window width is 12.



(b) Diagram demonstrating the data window being input into the RNN to receive an output prediction. The recursive use of RNN is due to return sequences being enabled.

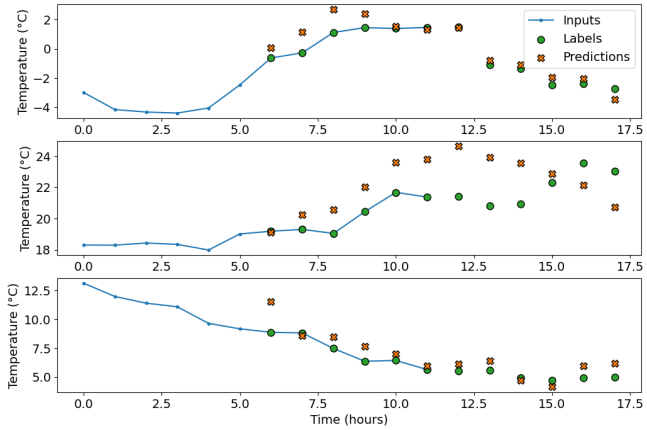


Figure 17: Graph showing the RNN's short-term predictions of temperature in New York.

define a separate model for each of the 12 locations as the distributions in temperatures were dramatically different - and due to the necessity of consecutive data the model would fail to learn patterns in the data if we combined the locations. In this case, we can produce a singular dataset containing data from all locations and define one model that learns and predicts over the full set.

However, we still want the model to be able to identify the differences in temperature distributions across various geographical locations and as such we add additional columns for longitude and latitude. Cyclical encoding of the date features was reintroduced, and the temperature

column was cut to be its own target value set. The final dataset for temperature predictions can be seen in tables 5, 6 & 7.

TABLE 5: Temperature Prediction Dataset (Part 1)

Precipitation	U-Wind	V-Wind	Longitude	Latitude
2.797500×10^{-6}	1.54910	0.77725	151.00	-33.50
8.866100×10^{-7}	8.38360	4.76240	10.75	59.92
...	...	...	...	...

TABLE 6: Temperature Prediction Dataset (Part 2)

month_sin	month_cos	day_sin
5.000000×10^{-1}	0.866025	2.012985×10^{-1}
$-2.449294 \times 10^{-16}$	1.000000	$-2.449294 \times 10^{-16}$
...	...	...

TABLE 7: Temperature Prediction Dataset (Part 3)

day_cos	hour_sin	hour_cos
0.97953	0.258819	0.965926
1.00000	0.258819	0.965926
...	...	...

Notably when it came to producing a test/train split for this data, we did not have to concern ourselves with keeping data sequential as we did with RNNs. We produced a test/train split sampled randomly but evenly across locations, relying on the presence of the longitude and latitude columns for the model to identify location-based trends. We defined another scikit-learn MLP Regresor model and performed a largely identical grid search to that mentioned in table 2 just with some tweaked parameters.

MLPRegressor's GridSearchCV Setup	
Hyperparameter	**Search Space**
Hidden Layer Size	(128), (64, 64), (128, 64, 32)
Activation Function	tanh, ReLU
Alpha	0.0001, 0.001, 0.01
Solver	SGD, Adam
Learning Rate	Constant, Adaptive

TABLE 8: A table showing the search space used for MLPRegressor with GridSearchCV.

Our grid search returned the following optimal hyperparameters which we used to initialise the MLP Regressor for training.

- Hidden layer size of (128, 64, 32)
- ReLu activation function
- Adam solver
- Alpha value of 0.001
- Constant learning rate

We trained the model for 100 epochs with early stopping triggering at epoch 52. Once trained, produced predictions over the full unseen test set - containing 820510 values; spread across all 12 locations in a random order.

To assess the regressor's average accuracy over all data points, we calculated the R^2 score of our model. R^2 is the coefficient of determination regression score function, it's a statistical measure that represents the proportion of variance in the dependent variable that can be explained by the independent variable.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Where:

$$SS_{res} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

And where:

- n is the number of observations.
- y_i is the actual value of the dependent variable for observation i.
- $\hat{y}_i$ is the predicted value of the dependent variable for observation i.
- $\bar{y}_i$ is the mean of the actual values of the dependent variable.

So R^2 is essentially a representation of how well our model fits the data - with a value closer to 100% meaning a closer representation of the variability in the ground truth values. Our model scored 92.392% over the full prediction set - indicating that our model effectively fits the data we have trained it on and produces close to perfect predictions.
To get a better idea of the MLP's performance, we then split the predictions back up into their 12 locations using the longitude and latitude columns, sorted the values by their date value, and then generated plots of actual vs predicted temperature for each location.

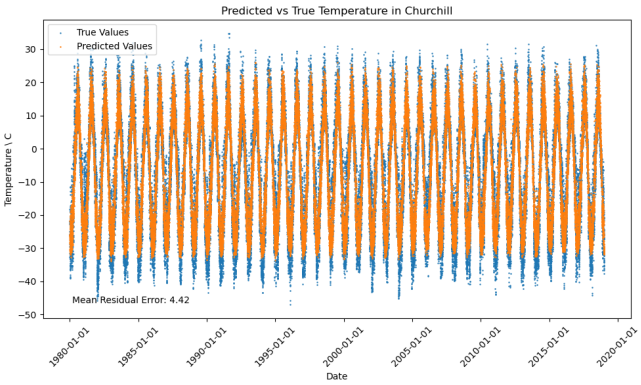


Figure 18: Predicted vs Actual Temperature in Churchill

Figures 18, 19 and 20 were selected, from the 12 generated, for their interesting features and perceived variability in performance. A mean residual error was calculated for each location, determined by calculating the mean of the absolute differences between each predicted value and its ground

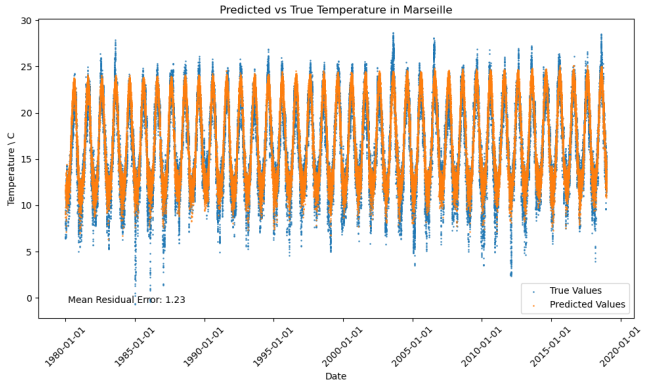


Figure 19: Predicted vs Actual Temperature in Marseille

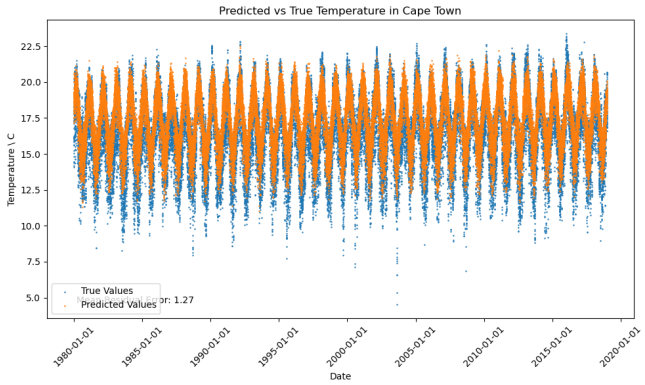


Figure 20: Predicted vs Actual Temperature in Cape Town

truth value. With the absolute residual errors for Cape Town, Churchill and Marseille being 1.268, 4.439 and 1.231 degrees Celsius respectively. While it may seem apparent that the model performed worst when predicting temperature in Churchill based on these results, we must consider that the range of temperatures in Churchill is significantly larger than in Cape Town for example.
And so to accurately compare the accuracy of the model across locations, we can instead calculate the relative residual error.
The relative residual error can be calculated using the following formula:

$$\text{Relative Residual Error } (\%) = \frac{\text{Absolute Error}}{\text{Temperature Range}} \times 100 \tag{8}$$

6. Discussion and Conclusion

In this report, we experimented with the application of machine learning to forecast temperature and to predict one variable given the other variables on this subset of the ERA5 Global Atmospheric Reanalysis Dataset.

Our data preparation was fairly trivial, the raw data only requiring some changes to conventional units and checking for outliers or missing values. Exploration of the dataset revealed expected meteorological phenomena, such as higher temperatures and less precipitation during summer months. We also investigated the variances of wind, precipitation and temperature in the different locations using box plots. The data then required processing to be used for machine learning techniques. We manipulated the data using cyclical encoding and split it into training, validation and test sequential subsets and finally, scaled it to make it suitable for machine learning. The variables are also checked for correlation to reduce redundancy but none of the variables are significantly correlated. Given the thoroughness of the data exploration and pre-processing we can be confident in the reliability of the results from models that use this data.

TABLE 9: Residual & Absolute Errors in Temperature

Location	Absolute Residual (Degrees)	Relative Residual (%)
Cape Town	1.268	6.723
London	1.337	6.339
Toronto	3.645	5.760
Calgary	4.288	5.674
Sydney	2.336	5.655
New York	3.472	5.517
Churchill	4.439	5.429
Kingston	0.662	5.382
Hyderabad	1.777	4.685
Rome	1.396	4.669
Oslo	2.417	4.221
Marseille	1.231	4.193

Our first approach to modelling, forecasting temperature 6 hours in the future, used several models including an MLPRegressor, RandomForestRegressor and an RNN. A rolling model was used as a baseline to compare to the performance of more complex models. While the MLPRegressor and RandomForestGenerator were able to improve on the rolling model's mean square error, they were worse in mean absolute error. This indicates they may have struggled with capturing short-term fluctuations present in the data. Despite their superior performance in mean square error, the discrepancy in mean absolute error suggests that these models might have overemphasised large deviations while overlooking smaller, yet significant, variations in temperature prediction. Only the RNN was able to improve on the baseline's absolute error and square error, proving it to be the most appropriate model for forecasting temperature. The RNN captures temporal dependencies and long-term patterns in time series data, unlike MLP and RandomForest which treat each input independently. This is likely why the RNN is more accurate and consistent in predicting fluctuating temperatures in meteorological data.

References

[1] Copernicus and ECMWF. ECMWF Reanalysis v5 (ERA5).

https://www.ecmwf.int/en/forecasts/dataset/ecmwf-reanalysis-v5. URL https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-complete?tab=overvie Accessed on 2023-05-25.

[2] Google Maps. Annotated World Map. http://maps.google.co.uk, . Accessed on 01/05/2024.

[3] Afan Galih Salman, Bayu Kanigoro, and Yaya Heryadi. Weather forecasting using deep learning techniques. In *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 281–285, 2015. doi: 10.1109/ICACSIS.2015.7415154.

[4] Google Maps. Oslo. http://maps.google.co.uk, . Accessed on 28/04/2024.