BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

MACHINE LEARNING AND DATA SCIENCE – ENCS5341

Course Project Report

Student Names: Mazen Batrawi – 1190102 & Khalid Dabe – 1190507

Instructor: Dr. Yazan Abu Farha

Jan – 2024

Table of Contents

1. Introduction 1
   1.1 K-Nearest Neighbor (KNN) 1
   1.2 Multilayer Perceptron (MLP) 1
   1.3 Support Vector Machine (SVM) 1
   1.4 Evaluation Metrics 1
   1.5 Cross-Validation Grid Search 2
2. Dataset 2
   2.1 Processing the data 2
3. Procedure, Results, and Analysis 4
   3.1 KNN 4
   3.2 MLP 6
   3.3 SVC 7
4. Conclusion 8
5. References 9

Table of Figures

Figure 1: Data information. 3
Figure 2: Categorical data distribution. 4
Figure 3: Numeric data distribution. 4
Figure 4: Parameters for the KNN. 4
Figure 5: Confusion matrix for KNN with k = 1. 5
Figure 6: Confusion matrix for KNN with k = 3. 5
Figure 7: ROC Curve for KNN. 6
Figure 8: Parameters for the MLP. 6
Figure 9: Confusion Matrix for MLP.. 7
Figure 10: ROC Curve for MLP. 7
Figure 11: Parameters for the SVC. 7
Figure 12: ROC Curve for SVC. 8
Figure 13: Confusion matrix for SVC. 8

Table of Tables

Table 1: Location count. 3
Table 2 - KNN Best parameters 5
Table 3: KNN Results. 5
Table 4 - MLP best parameters 6
Table 5- MLP Results 7
Table 6 - SVC Results and best parameters 7

1. Introduction

It has become clear that machine learning (ML) is a powerful tool for solving challenging issues and extracting valuable insights from a variety of datasets. With the use of a carefully selected dataset, we hope to apply ML techniques to a real-world predicting task and assess the performance of several models. Our research primarily focuses on applying the k-Nearest Neighbors (KNN), Multilayer Perceptron (MLP), and Support Vector Machine (SVM) models. These models were selected based on their individual qualities and capacity to provide high levels of prediction accuracy.

1.1 K-Nearest Neighbor (KNN)

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It is heavily used in pattern recognition, data mining, and intrusion detection and is a member of the supervised learning domain. Because of its simplicity and ease of implementation, this machine learning technique is both frequently used and adaptable. No presumptions on the distribution of the underlying data are necessary. It is a versatile option for a range of dataset types in classification and regression applications because it can handle both numerical and categorical data. This non-parametric technique bases its predictions on how similar the data points in a particular dataset are to one another. By comparison, K-NN is less susceptible to outliers than other algorithms. [1]

1.2 Multilayer Perceptron (MLP)

Multilayer Perceptron MLP is a group of dense layers that are entirely connected and can convert any input dimension to the required dimension. A neural network with several layers is called a multi-layer perception. Neurons are joined together to form neural networks, with some neurons' outputs acting as inputs for other neurons. [2]

1.3 Support Vector Machine (SVM)

Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. The primary goal of Support Vector Machines (SVMs) is to identify the hyperplane that maximally divides the various classes in the training set. Finding the hyperplane with the biggest margin—defined as the separation between the hyperplane and the nearest data points from each class—is how this is accomplished. New data can be categorized by identifying which side of the hyperplane it falls on once the hyperplane has been identified. SVMs are especially helpful in situations where the data has a large number of features and/or a distinct margin of separation. [3]

1.4 Evaluation Metrics

A critical aspect of assessing the performance of machine learning models lies in the selection of appropriate evaluation metrics. In our project, we employ a set of well-established metrics that

align with the nature of our predictive task. The chosen metrics offer a comprehensive view of the models' performance and cater to both classification and regression scenarios.

The performance of the models was evaluated using the following metrics:

- ROC Curves and AUC: Visualize the models' ability to distinguish between positive and negative classes effectively.
- Confusion Matrices: Reveal the distribution of correct and incorrect predictions for each class.
- Weighted Average Precision, Recall, and F1-Score: Provide a comprehensive assessment of the models' accuracy, taking into account class imbalances.
- Accuracy: Measure the overall proportion of correct predictions.

1.5 Cross-Validation Grid Search

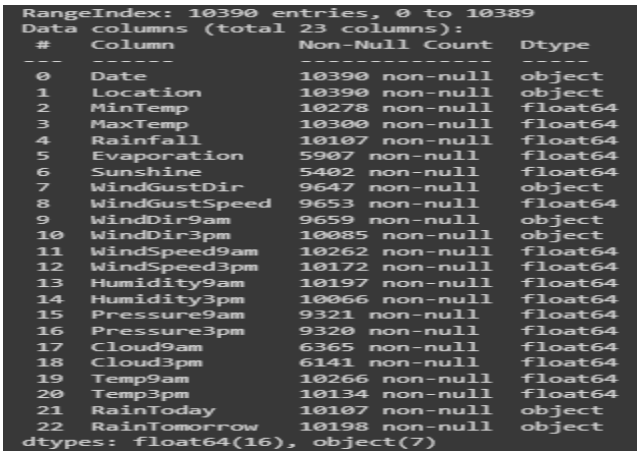
Cross-validation grid search is a crucial technique in machine learning for optimizing model performance. It involves utilizing cross-validation to properly evaluate various combinations of hyperparameter values in order to improve the model's generalization to unknown data. Grid search examines a predefined grid of parameters, evaluating the effect of each configuration on the correctness of the model. It evaluates performance over several dataset subsets by utilizing cross-validation folds, providing a more accurate estimate of the model's efficiency. By avoiding over- or underfitting, this iterative procedure helps determine the ideal hyperparameter values. Although cross-validation grid search requires a lot of computing power, its advantages—such as improved model accuracy—make it a vital tool for optimizing machine learning models. [4]

2. Dataset

The dataset employed for this project is the "Rain in Australia" [5] dataset, publicly available on Kaggle. It encompasses meteorological data collected from various locations across Australia between 1900 and 2015. The dataset comprises 23 attributes, including rainfall observations, temperature, humidity, atmospheric pressure, and wind speeds. Notably, it features a binary target variable indicating whether or not rain occurred on that day and the day after it. The dataset contains approximately 145k entries, which made the runtime too long (above 5 hours), so we minimized the data by taking every 14^{th} entry from it.

2.1 Processing the data

We started processing the data by showcasing the features as shown in figure 1.



```
RangeIndex: 10390 entries, 0 to 10389
Data columns (total 23 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            10390 non-null  object
 1   Location        10390 non-null  object
 2   MinTemp         10278 non-null  float64
 3   MaxTemp         10300 non-null  float64
 4   Rainfall        10107 non-null  float64
 5   Evaporation     5907 non-null   float64
 6   Sunshine        5402 non-null   float64
 7   WindGustDir     9647 non-null   object
 8   WindGustSpeed   9653 non-null   float64
 9   WindDir9am      9659 non-null   object
 10  WindDir3pm      10085 non-null  object
 11  WindSpeed9am    10262 non-null  float64
 12  WindSpeed3pm    10172 non-null  float64
 13  Humidity9am     10197 non-null  float64
 14  Humidity3pm     10066 non-null  float64
 15  Pressure9am     9321 non-null   float64
 16  Pressure3pm     9320 non-null   float64
 17  Cloud9am        6365 non-null   float64
 18  Cloud3pm        6141 non-null   float64
 19  Temp9am         10266 non-null  float64
 20  Temp3pm         10134 non-null  float64
 21  RainToday       10107 non-null  object
 22  RainTomorrow    10198 non-null  object
dtypes: float64(16), object(7)
```

Figure 1: Data information.

We can see that the total number of rows is 10390, some features have missing values so we need to fill them. And we can also see that we have numeric and categorical data.

Then we showcased the correlation between the data in the figure in section 3.1 in the colab file. After that, we decided to remove the date because the date is considered as noise, so we need to remove it, but the month is important because it is related to the raining periods so we will keep it. Then we started filling the categorical data with the mode and the numeric data with the median. We tried forward filling but the results got worse, so we stayed on filling with the median and the mode. Table 1 and figures 2 and 3 show visualizations for the data.

Table 1: Location count.

Location	Count
Canberra	245
Sydney	239
Perth	229
Adelaide	229
Melbourne	229
Darwin	228
Hobart	228
Brisbane	228
Bendigo	218
Albury	218
Newcastle	217
MountGinini	217
Townsville	217
MountGambier	217
Albany	217
GoldCoast	217
Ballarat	217
Cairns	217
Tuggeranong	217
Penrith	217
Wollongong	217
Launceston	217
AliceSprings	217
Walpole	215
PearceRAAF	215
Witchcliffe	215
Woomera	215
Nuriootpa	215
Moree	215
CoffsHarbour	215
Richmond	215
SydneyAirport	215
Dartmoor	215
Watsonia	215
Portland	215
BadgerysCreek	215
NorahHead	215
Sale	215
NorfolkIsland	215
Williamtown	215
WaggaWagga	215
MelbourneAirport	214
PerthAirport	214
SalmonGums	214
Cobar	214
Mildura	214
Katherine	113
Nhil	113
Uluru	112

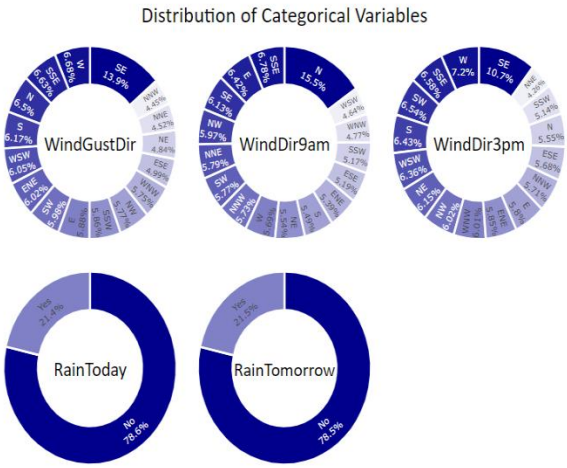


Figure 2: Categorical data distribution.

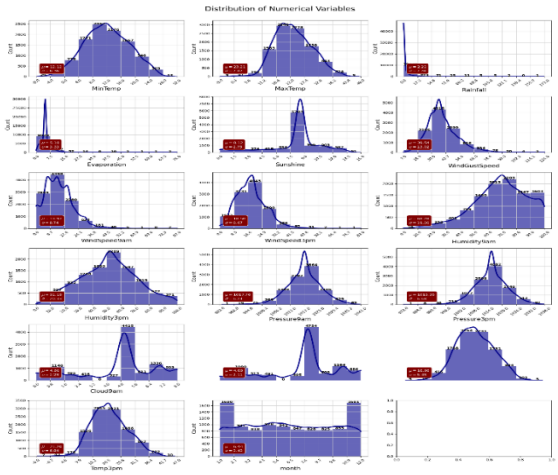


Figure 3: Numeric data distribution.

Figure 2 shows that some of the data are balanced and the others are unbalanced, which is normal, and figure 3 shows some statistics about the numeric data, like the mean, standard deviation, minimum, and maximum values.

Then we encoded the categorical values into numeric values, to process them in the models. After encoding the data, we scaled it. We had large and small values, which is not good for the training, so we wrapped the values around the mean so that all values become close and small. Section 3.6 in the colab file shows the data before and after scaling. The last step in data processing was detecting the outliers and removing them manually, as shown in section 3.7 in the colab file.

3. Procedure, Results, and Analysis

3.1 KNN

We applied the KNN algorithm on the dataset with K = 1 and K = 3. We applied hyper-parameter tuning using grid search cross validation at 5 folds. The parameters and their values were as shown in figure 4.

```
param_grid_knn_1 = {
    'n_neighbors': [1],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30, 40],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
```

Figure 2: Parameters for the KNN.

The n_neighbors variable represents the K value for the KNN algorithm. For the weights parameter, the uniform means that all points in each neighborhood are weighted equally, while distance means that closer neighbors of a query point will have a greater influence than neighbors which are further away. The algorithm parameter is used to determine the nearest neighbor. The leaf size controls the minimum number of points in a given node and is passed to BallTree or KDTree. The metric parameter is used for distance computation. Table 2 demonstrates the results for K = 1 and K = 3.

Table 2 - KNN Best parameters

K	Algorithm	Leaf_size	metric	N_neighbors	Weights
1	Auto	10	Manhattan	1	uniform
3	Auto	10	Manhattan	3	uniform

Table 3: KNN Results.

K	Best Cross-validated Accuracy	Training Accuracy	Precision	Recall	F1-Score	AUC
1	0.80	1.0	0.76	0.77	0.77	0.62
3	0.82	0.89	0.77	0.80	0.78	0.69

Note that the values of the precision, recall, and F1-Score are the weighted averages. Figures 5 and 6 show the confusion matrix for each K. We can see from the table above that the training accuracy for k=1 is equal to 1 and the testing accuracy is 0.80, which implicates overfitting. while in k=3, the training accuracy is 0.89 and the testing accuracy is 0.82, which indicates less overfitting. At k=1, the KNN tends to closely follow the training data and thus shows a high training score. However, in comparison, the test score is quite low, thus indicating overfitting. So, as we increase k, we will notice less overfitting.

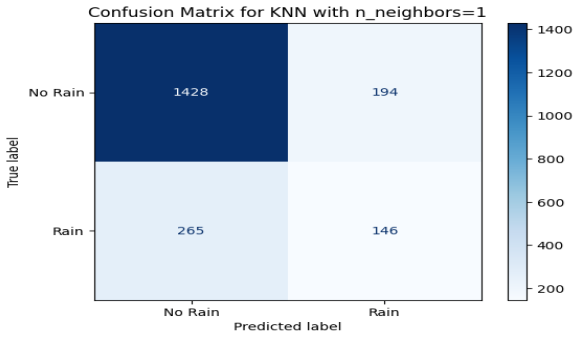


Figure 5: Confusion matrix for KNN with k = 1.

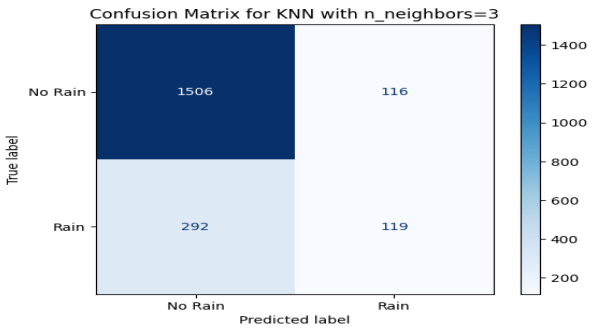


Figure 6: Confusion matrix for KNN with k = 3.

And for the Receiver Operating Characteristic (ROC) Curve for KNN, figure 7 demonstrates it.

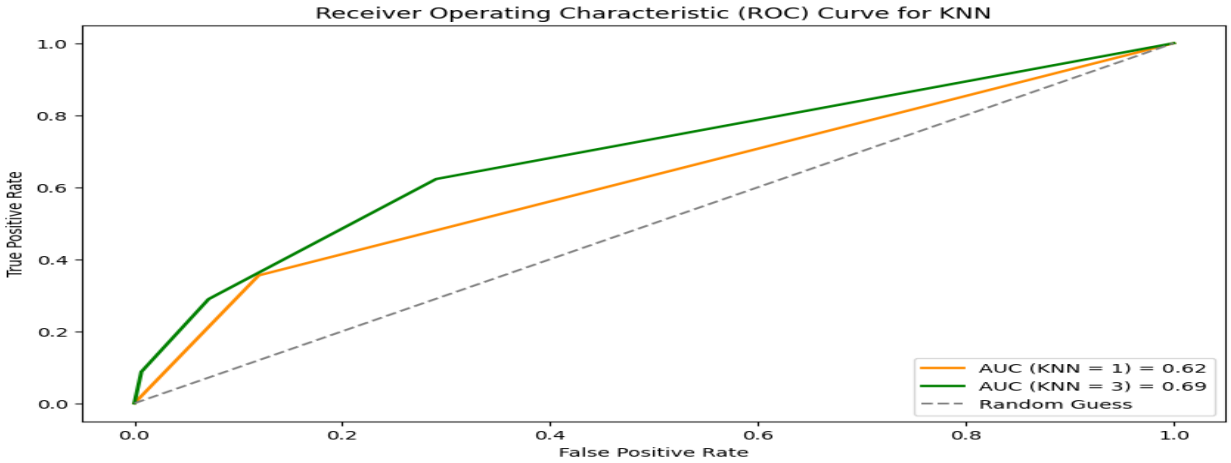


Figure 3: ROC Curve for KNN.

The results are kind of good, but other models must produce higher results and more correct predictions.

3.2 MLP

For the second learning algorithm we've applied MLP (Multilayer Perceptron) algorithm

```
param_grid_mlp = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50), (100, 100)],
    'activation': ['relu', 'logistic'],
    'solver': ['adam'],
    'max_iter': [1000],
    'random_state': [0],
}
```

Figure 4: Parameters for the MLP.

- **hidden_layer_sizes:** This parameter allows us to set the number of layers and the number of nodes we wish to have in the Neural Network Classifier. Each element in the tuple represents the number of nodes at the i^{th} position where i is the index of the tuple. Thus, the length of tuple denotes the total number of hidden layers in the network.
- **max_iter:** It denotes the number of epochs.
- **activation:** The activation functions for the hidden layers.
- **solver:** This parameter specifies the algorithm for weight optimization across the nodes.
- **random_state:** The parameter allows setting a seed for reproducing the same results.

Table 4 - MLP best parameters

Activation	hidden_layer_sizes	max_iter	random_state	solver
logistic	(100,)	1000	0	adam

Table 5- MLP Results

Hidden Layers	Best Cross-validated Accuracy	Training Accuracy	Precision	Recall	F1-Score	AUC
(100,)	0.84	0.94	0.79	0.81	0.80	0.79

The model shows improvements on the KNN model.

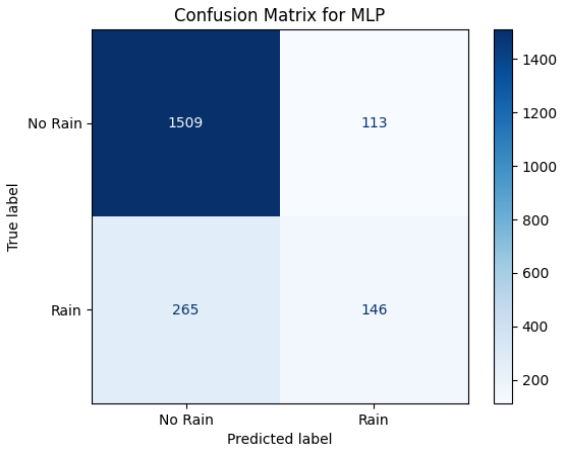


Figure 5: Confusion Matrix for MLP.

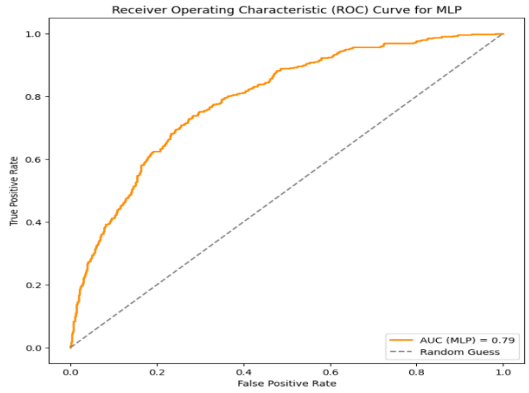


Figure 6: ROC Curve for MLP.

3.3 SVC

For the second learning algorithm we've applied SVM (Support Vector Machine) algorithm.

```
param_grid = {
    'C': [0.1, 1, 10, 100],
}
```

Figure 7: Parameters for the SVC.

The 'C' parameter in figure above represents Regularization parameter and it tells the SVM optimization how much it wants to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.

Table 6 - SVC Results and best parameters

C (best Param)	Best Cross-validated Accuracy	Training Accuracy	Precision	Recall	F1-Score	AUC
1	0.85	0.87	0.83	0.84	0.81	0.80

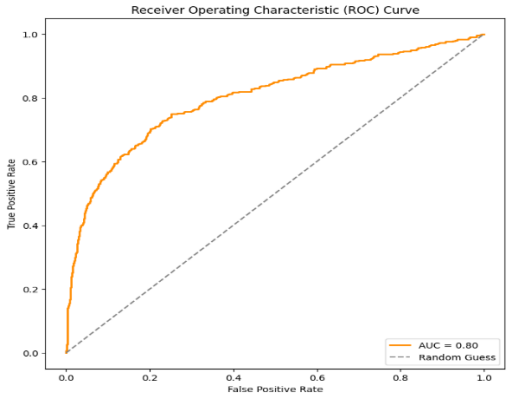


Figure 8: ROC Curve for SVC.

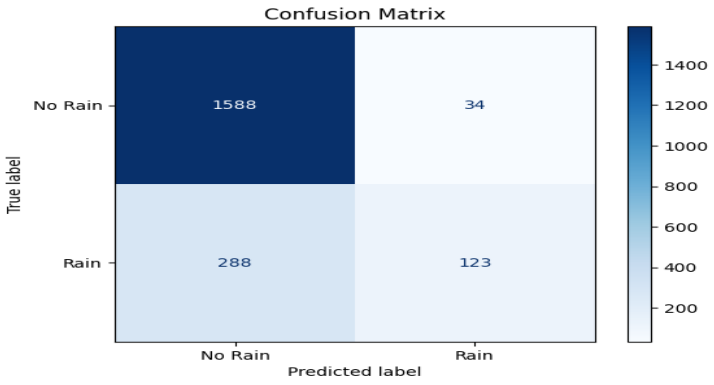


Figure 9: Confusion matrix for SVC.

4. Conclusion

In summary, three machine learning models—K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), and Support Vector Classifier (SVC)—were trained and evaluated on a dataset. Let's look at each model's performance.

The KNN model, when set with K=3, showed better results compared to K=1. It achieved a higher cross-validation accuracy of 0.82, more accurately predicting instances with 1625 true values. K=1, on the other hand, exhibited overfitting with a training accuracy of 1.0 and a lower testing accuracy of 0.80. As the value of K increased, the KNN model showed less overfitting.

Moving on to the MLP model, it demonstrated improvement over the KNN model. The best parameters for this model included an activation function of 'logistic' and a hidden layer with 100 neurons. While the cross-validation accuracy reached 0.84, the high training accuracy of 0.94 suggested potential overfitting. The area under the ROC curve (AUC) was 0.79, and the model correctly predicted 1655 instances.

The SVC model stood out as the most effective among the three. With a high accuracy of 0.85 and closely aligned training and testing accuracies (0.87), the SVC model did not show significant overfitting. The best parameter for this model was 'C=1,' and it correctly predicted 1711 instances. The AUC for the ROC curve was 0.80.

Comparing the models, the SVC model performed the best with a high accuracy and balanced training and testing results. KNN with K=3 also showed competitive results, outperforming K=1 and demonstrating less overfitting. The MLP model, while achieving high training accuracy, raised concerns about potential overfitting with a lower cross-validation accuracy. Choosing the best model depends on the specific goals and constraints of your application, as well as considerations like interpretability and computational efficiency. Further tuning of hyperparameters could potentially improve the models' performance.

5. References

[1] "K-Nearest Neighbor(KNN) Algorithm." *GeeksforGeeks*, 9 Nov. 2023, www.geeksforgeeks.org/k-nearest-neighbours/. Accessed 19 Jan. 2024.

[2] "Classification Using Sklearn Multi-Layer Perceptron." *GeeksforGeeks*, 11 Oct. 2023, www.geeksforgeeks.org/classification-using-sklearn-multi-layer-perceptron/. Accessed 19 Jan. 2024.

[3] "Introduction to Support Vector Machines (SVM)." *GeeksforGeeks*, 2 Feb. 2023, www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/. Accessed 19 Jan. 2024.

[4] Shah, Rahul. "Tune Hyperparameters with GRIDSEARCHCV." *Analytics Vidhya*, 6 Dec. 2023, www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/. Accessed 15 Jan. 2024.

[5] Young, Joe. "Rain in Australia." *Kaggle*, 11 Dec. 2020, www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package/. Accessed 15 Jan. 2024.