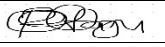
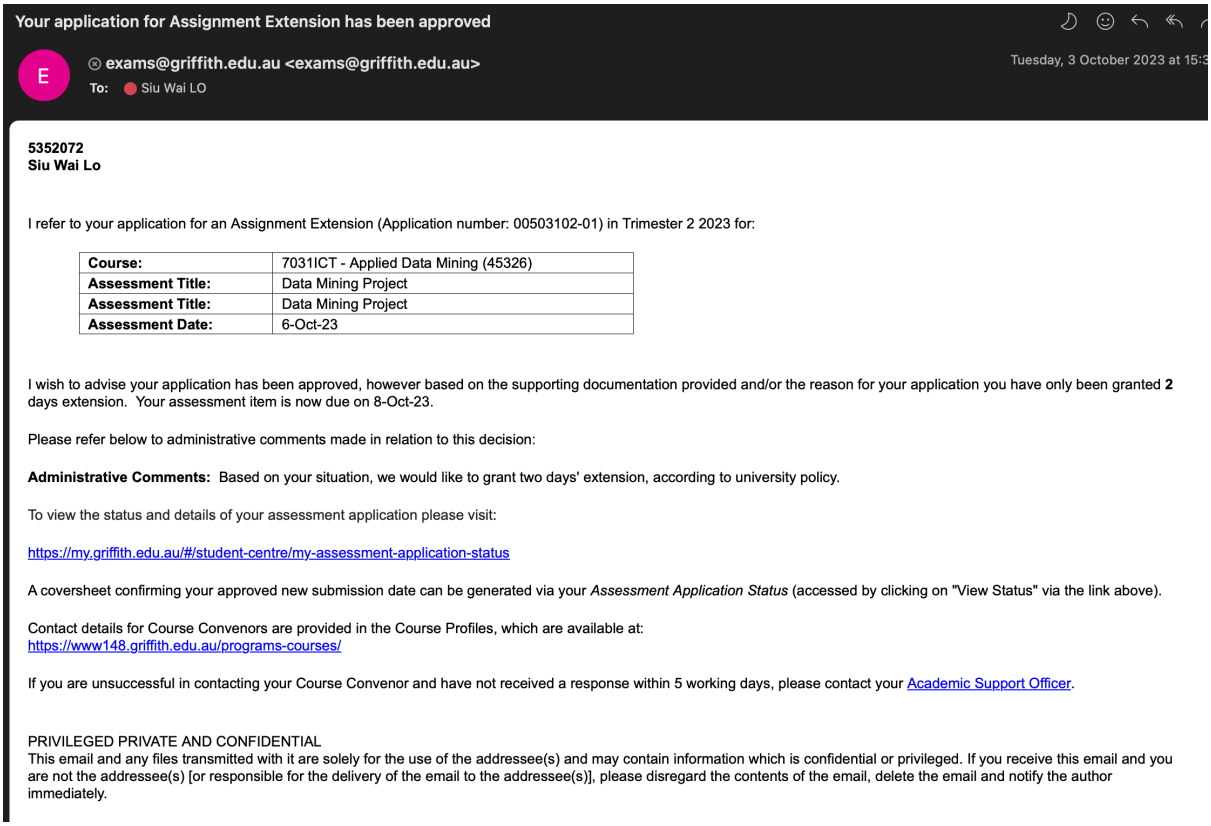
PREDICTIVE ANALYSIS OF AUSTRALIAN WEATHER CONDITIONS

7031ICT Assessment 2: Data Mining Project - Part 2

Group members:

Name	Student Number	Signature
Chimmy Rai	S5255377	
Erdenechimeg Darjaa	S5313342	
Ruvimbo Melody Chitagu	S5308164	
Siu Wai Lo	S5232072	

Predictive analysis of Australian weather conditions



Your application for Assignment Extension has been approved

E exams@griffith.edu.au <exams@griffith.edu.au>
To: Siu Wai LO
Tuesday, 3 October 2023 at 15:3

5352072
Siu Wai Lo

I refer to your application for an Assignment Extension (Application number: 00503102-01) in Trimester 2 2023 for:

Course:	7031ICT - Applied Data Mining (45326)
Assessment Title:	Data Mining Project
Assessment Title:	Data Mining Project
Assessment Date:	6-Oct-23

I wish to advise your application has been approved, however based on the supporting documentation provided and/or the reason for your application you have only been granted **2** days extension. Your assessment item is now due on 8-Oct-23.

Please refer below to administrative comments made in relation to this decision:

Administrative Comments: Based on your situation, we would like to grant two days' extension, according to university policy.

To view the status and details of your assessment application please visit:

https://my.griffith.edu.au/#/student-centre/my-assessment-application-status

A coversheet confirming your approved new submission date can be generated via your *Assessment Application Status* (accessed by clicking on "View Status" via the link above).

Contact details for Course Convenors are provided in the Course Profiles, which are available at:
https://www148.griffith.edu.au/programs-courses/

If you are unsuccessful in contacting your Course Convenor and have not received a response within 5 working days, please contact your Academic Support Officer.

PRIVILEGED PRIVATE AND CONFIDENTIAL
This email and any files transmitted with it are solely for the use of the addressee(s) and may contain information which is confidential or privileged. If you receive this email and you are not the addressee(s) [or responsible for the delivery of the email to the addressee(s)], please disregard the contents of the email, delete the email and notify the author immediately.

Table of Contents

Predictive analysis of Australian weather conditions

Introduction

In the fields of meteorology and environmental science, the task of predicting next-day rain carries significant weight and practical relevance. It revolves around the ability to forecast whether it will rain on the following day, framed as a binary classification problem with two outcomes: Yes or No. This project embarks on the journey to tackle the challenge of predicting next-day rain by harnessing historical weather data and employing classification models as the primary instruments for accomplishing this objective.

Motivation:

Precise predictions of next-day rain hold immense importance for individuals engaged in agriculture and related professions. The timely provision of rainfall information empowers farmers to make informed decisions concerning irrigation, crop management, and optimal harvest planning. This, in turn, enhances crop yields and resource allocation efficiency, directly impacting food production and agricultural sustainability. Apart from agriculture, weather forecasting agencies rely on these predictions to furnish the public with dependable weather forecasts. Such forecasts play a pivotal role in shaping everyday activities, influencing travel plans, and facilitating the planning of outdoor events. Hence, the accuracy of next-day rain predictions bears a direct impact on the quality of life for many individuals.

Moreover, water resource management agencies heavily depend on precise rainfall forecasts to oversee critical infrastructure, including reservoirs, dams, and water distribution systems. Accurate predictions assist in the efficient allocation and conservation of this finite resource, thereby contributing to responsible environmental stewardship.

Project Goal:

At its core, this project aspires to develop and assess classification models capable of predicting the occurrence of rain on the following day, represented as a binary outcome: Yes or No. To realize this ambition, we will delve into the realm of machine learning, exploring a variety of algorithms and techniques. In Python, we will employ widely recognized tools such as scikit-learn and XGBoost. In parallel, we will utilize the capabilities of the R programming language, employing packages such as 'glm' for model development. Our models will undergo training and evaluation using a comprehensive dataset spanning approximately a decade of daily weather observations from diverse geographic locations across Australia. This dataset, sourced from the Bureau of Meteorology (BoM), offers rich insights into Australia's dynamic climate patterns and is instrumental in our predictive modeling endeavors.

The targeted problem in this project is the prediction of Australian weather conditions, specifically focusing on whether it will rain tomorrow. The motivation behind this endeavor stems from the diverse and unpredictable nature of Australia's climate due to global climate changes. Accurate weather predictions are crucial for various sectors such as agriculture, water resource management, and natural disaster preparations. By enhancing the accuracy and reliability of weather forecasts, this project aims to facilitate better planning and decision-making in Australia's key industries, ultimately mitigating the impact of weather-related events.

Predictive analysis of Australian weather conditions

Solution and key results and metrics of algorithms

Linear Regression on dataset:

The objective of linear regression is to establish a linear relationship between one or more input features and a continuous target variable. It aims to find the best-fit straight line (or hyperplane in multiple linear regression) that minimizes the difference between the predicted values and the actual values of the target variable in the training data. In this project, we used logistic regression instead of linear regression likely because our problem involved a binary classification task, where we needed to predict one of two possible classes or outcomes. In binary classification, the target variable represents a categorical outcome, such as "yes" or "no," "1" or "0". Linear regression, on the other hand, is designed for predicting continuous numeric values, making it less appropriate for classifying discrete categories.

To do linear regression, we used:

In R Studio:

Import dataset into environment.

- library(tidyverse)
- model_rain_tomorrow <- glm(RainTomorrow ~ . - Date - Location - RainTomorrow, data = cleaned_file, family = binomial)
- summary(model_rain_tomorrow)
- library(coefplot)
- coefplot(model_rain_tomorrow)

Results from R:

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	4.778170e+01	1.544133e+00	30.94403902	3.055920e-210
...1	-1.889596e-08	2.041929e-07	-0.09253974	9.262692e-01
MinTemp	7.955892e-03	3.896506e-03	2.04180175	4.117120e-02
MaxTemp	-3.680238e-02	4.723478e-03	-7.79137277	6.628504e-15
Rainfall	5.107438e-01	3.451462e-02	14.79789699	1.511345e-49
Evaporation	6.481915e-03	3.964025e-03	1.63518500	1.020103e-01
Sunshine	-1.142803e-01	3.731952e-03	-30.62213287	6.211737e-206
WindGustDirENE	-8.989663e-03	5.362854e-02	-0.16762833	8.668757e-01

glm(formula = RainTomorrow ~ . - Date - Location - RainTomorrow, family = binomial, data = cleaned_file)

Deviance Residuals:
Min 1Q Median 3Q Max
-2.8915 -0.5547 -0.3172 -0.1354 3.2159

Deviance residuals measure the model's fit to the data.

Negative residuals suggest the model underpredicts the likelihood of rain tomorrow, while positive residuals suggest overprediction.
The range of residuals goes from -2.8915 (underprediction) to 3.2159 (overprediction).

Coefficients:
Coefficients represent the impact of predictor variables on the likelihood of rain.
Notable coefficients:
MaxTemp: A decrease in temperature reduces the likelihood of rain (statistically significant).
Rainfall: Increased rainfall significantly increases the likelihood of rain.
Sunshine: More sunshine significantly reduces the likelihood of rain.
Deviance:
Deviance measures how well the model fits the data.
Null deviance (152,973) is the model with no predictors; residual deviance (104,075) is the improved model with predictors.
AIC (Akaike Information Criterion):
AIC assesses model goodness of fit, considering predictor count.
Lower AIC (104,203) suggests your model provides a good fit to the data.
Number of Fisher Scoring Iterations:
The model required five iterations to estimate parameters.

	Accuracy	CI_95_Lower	CI_95_Upper	NIR	Kappa	Sensitivity	Specificity	PPV	NPV	Prevalence	Detection_Rate	Detection_Prevalence	Balanced_Accuracy	Positive_Class
1	0.8445	0.8426	0.8463	0.7809	0.4867	0.9451	0.4859	0.8676	0.7129	0.7809	0.738	0.8506	0.7155	0

Accuracy: The model's overall correctness, with a value of 0.8445 indicating 84.45% correct predictions.

CI_95_Lower and CI_95_Upper: Lower and upper bounds of a 95% confidence interval for accuracy, with values of 0.8426 and 0.8463, respectively.
NIR (Negative to Incidence Ratio): A measure evaluating the model's ability to distinguish negative from positive cases, with a higher value suggesting better separation.
Kappa: A statistic measuring the agreement between predicted and actual outcomes, yielding moderate agreement at 0.4867.
Sensitivity (True Positive Rate): The model's ability to correctly identify positive cases at 0.9451.
Specificity (True Negative Rate): The model's ability to correctly identify negative cases at 0.4859.

PPV (Positive Predictive Value): The proportion of true positive predictions out of all positive predictions, indicating 86.76% accuracy.
NPV (Negative Predictive Value): The proportion of true negative predictions out of all negative predictions, showing 71.29% accuracy.
Prevalence: The proportion of actual positive instances in the dataset at 0.7809.
Detection Rate: The rate at which positive cases are correctly detected, with a value of 0.738.
Detection Prevalence: The prevalence of positive predictions, at 0.8506.
Balanced Accuracy: An overall performance measure balancing true positives and true negatives, with a value of 0.7155.

Positive_Class: An identifier for the positive class, marked as 0.

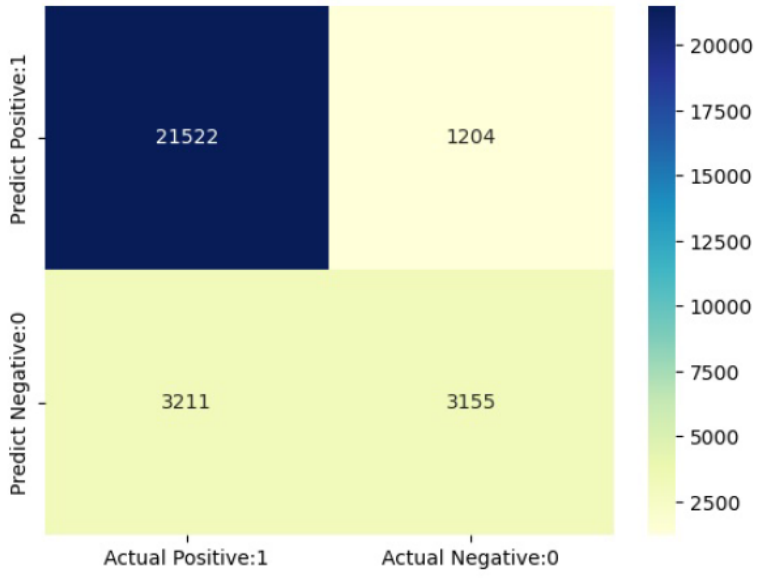
- We went on to do linear regression in Jupyter notebook(Python)
- from sklearn.linear_model import LogisticRegression to import the logistic regression model class.
- logreg = LogisticRegression(solver='liblinear', random_state=0) to create an instance of the logistic regression model named logreg with the solver parameter set to 'liblinear'.
- logreg.fit(X_train, y_train) this line of code fits the logistic regression model to training data (X_train and y_train). The fit method is used to train classification models.

Confusion Matrix:

```
Confusion matrix

 [[21522  1204]
 [ 3211  3155]]

True Positives(TP) =  21522

True Negatives(TN) =  3155

False Positives(FP) =  1204

False Negatives(FN) =  3211
```

```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```



Also checked for overfitting and underfitting:

```
print('Training set score: {:.4f}'.format(logreg.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.8487
Test set score: 0.8482
```

Statistics for Jupyter Notebook:

```
# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.9470
```

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.8702
```

```
true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 0.8702
```

```
false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.2762
```

```
# calculate cross-validated ROC AUC

from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(logreg, X_train, y_train, cv=5, scoring='roc_auc').mean()

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.8680
```

Classification Accuracy: 0.8482

Classification Error: 0.1518

Precision: 0.9470

Recall or Sensitivity: 0.8702 (True Positive Rate)

False Positive Rate: 0.2762

Cross-Validated ROC AUC: 0.8680

Comparison between the two tools on same Algorithm:

Performance Metrics in Jupyter:

- **Accuracy**: 0.8482

- **Precision**: 0.9470
- **Recall (Sensitivity)**: 0.8702
- **F1-Score**: 0.91 (for 'No' class) and 0.59 (for 'Yes' class)
- **ROC-AUC**: 0.8680

Performance Metrics in R Studio:

- **Accuracy**: 0.8445
- **Precision**: 0.8676 (for 'No' class) and 0.7129 (for 'Yes' class)
- **Recall (Sensitivity)**: 0.9451 (for 'No' class) and 0.4859 (for 'Yes' class)
- **F1-Score**: 0.8506 (for 'No' class) and 0.738 (for 'Yes' class)

Comparison:

1. **Accuracy**: The accuracy in Jupyter (0.8482) is slightly higher than in R Studio (0.8445).
2. **Precision**: In Jupyter, precision for the 'No' class is higher (0.9470) compared to R Studio (0.8676), indicating better performance in correctly identifying the 'No' class. However, for the 'Yes' class, R Studio has a higher precision (0.7129) compared to Jupyter (0.72).
3. **Recall (Sensitivity)**: Jupyter has a lower recall for the 'No' class (0.8702) compared to R Studio (0.9451), indicating that Jupyter's model is less sensitive in identifying 'No' instances. For the 'Yes' class, R Studio has a lower recall (0.4859) compared to Jupyter (0.50).
4. **F1-Score**: In terms of F1-score, Jupyter outperforms R Studio for the 'No' class with a higher F1-score (0.91 vs. 0.8506). However, R Studio has a higher F1-score for the 'Yes' class (0.738 vs. 0.59).
5. **ROC-AUC**: Jupyter's model has a slightly higher ROC-AUC (0.8680) compared to R Studio.

Overall, both environments produce models with reasonably similar performance, there are indeed some very slight differences in precision, recall, and F1-score for different classes. The decision on which tool to use would sorely depend on the type of project we would need the model for and which programming language one would resonate the best with.

Decision Tree Classifier

This classifier will be employed to decipher the patterns that lead to RainToday and RainTomorrow, and compare its results with the KNN classifier. Decision Trees are hierarchical models that split the dataset into subsets based on attribute values. This process is recursive, creating a tree of decisions. It's ability to capture complex relationships within the data makes it a good choice to achieve patterns in predicting rain.

RStudio and Juypter Notebook are used in this section. The dataset is pre-processed in the assessment 1 which removed the outliners and missing values in all cells except the columns of RainToday and RainTomorrow. The dataset is separated into 3 part:

a) Training set
b) Testing set
c) Prediction sets which Raintoday or RainTomorrow is empty

The classification tree created by the training set is being tested by the testing set. The accuracy of the classification tree should be higher than 0.8 before being implemented to predict the RainToday and RainTomorrow value in the prediction set.

The methods for creating classification tree in Juypter Notebook and RStudio:

- **i) Load required library**
  - (1) create Dataframe:
    - (a) Jupyter Notebook:
      - (i) import pandas as pd: for dataframe
  - (2) create classification tree
    - (a) Jupyter Notebook:
      - (i) from sklearn import metrics
      - (ii) from sklearn import tree
      - (iii) from sklearn.tree import DecisionTreeClassifier
      - (iv) from sklearn.model_selection import train_test_split
    - (b) RStudio: require(tree)
  - (3) Hyperparameter Tuning and Cross Validation to Decision Tree
    - (a) Jupyter Notebook:
      - (i) from sklearn.model_selection import GridSearchCV
      - (ii) from sklearn.model_selection import RandomizedSearchCV
      - (iii) from sklearn.metrics import confusion_matrix
      - (iv) from scipy.stats import randint
    - (b) RStudio: library(caret)
- **ii) Pre-process the dataset to suit for classification tree**
  - (1) all missing values and outliners in the dataset is removed during previous assessment 1.
  - (2) extract the rows with NaN value in RainToday or RainTomorrow to new dataframes and delete them in the dataset, which will be used for prediction later.
- **iii) Split dataset into training set and test set (80% training and 20% test)**
  - (1) Jupyter Notebook:
    - (a) Assign the variable to feature_cols = []
    - (b) Assign Features to X = df[feature_cols]
    - (c) Assign target variable to y= df.RainTomorrow
    - (d) Generate the training and test set by train_test_split(X, y, test_size=0.2, random_state=1)
  - (2) RStudio:
    - (a) Assign the percentage to training set = nrow(weather)*0.8
    - (b) Generate the training set by sample(1:nrow(weather), training)
- **iv) Create and Train the Decision Tree classifer object**

(1) Jupyter Notebook:
  (a) Create the tree by DecisionTreeClassifier()
  (b) Train the tree by clf.fit(X_train,y_train)
(2) RStudio:
  (a) Assign the variables and subset to tree = tree(as.factor(RainToday)~ variables, data = weather, subset = train_tdy)

v) Predict the response for test dataset

(1) Jupyter Notebook: predict the target value by clf.predict(X_test)
(2) RStudio: predict the target value by predict(tree.tdy, weather[-train_tdy,], type="class")

vi) **Find the accuracy and recal scores for the classification tree**

(1) Jupyter Notebook:
  (a) Generate the confusion matrix
  (b) Generate the tree model accuracy by metrics.accuracy_score
  (c) Generate the tree model recall by metrics.recall_score
(2) RStudio:
  (a) Use caret::confusionMatrix to create the confusion matrix and accuracy.

vii) Cross Validation to Decision Tree

  (a) Jupyter Notebook:
    (i) use RandomizedSearchCV to find the optimized model
    (ii) Find the best parameters by best_params_
    (iii) Find the best scores by cv.best_score_
  (b) RStudio:
    (i) Use cross-validation to prune the tree cv.tree(tree.tmr, FUN = prune.misclass)

viii) Use the best parameters to generate the classification tree and use the tree to predict the RainToday and RainTomorrow in the predication sets.

ix) Store the predicted value to the dataframe and export to csv.

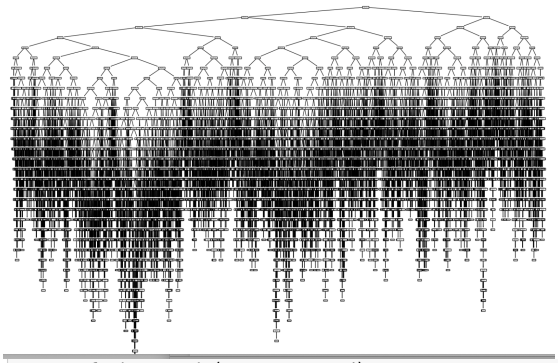
Predictive analysis of Australian weather conditions

Key results and metrics of your algorithms.

1) Classification Tree for RainTomorrow

Jupyter Notebook

```
fig = plt.figure(figsize=(20,15))
tree.plot_tree(clf, feature_names=feature_cols)
fig.savefig("decistion_tree_before_tune.png")
```



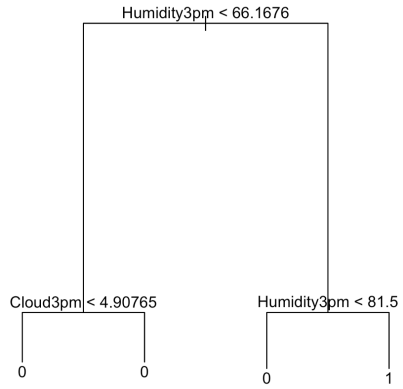
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[18790  3128]
 [ 2917  3323]]
```

```
#and get the accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.7853185595567868
Recall: 0.5325320512820513
```

RStudio



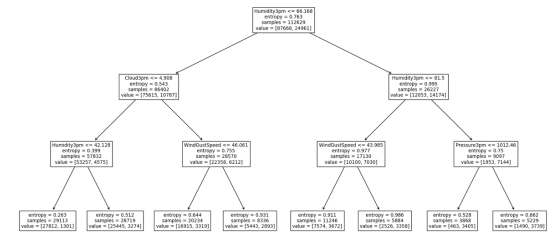

```
                RainTomorrow
tree_tmr.pred     0     1
            0 21526  4421
            1   489  1722

               Accuracy : 0.8256
                 95% CI : (0.8211, 0.83)
    No Information Rate : 0.7818
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3355
```

Sensitivity: 0.280319062
Specificity: 0.977787872

From the above tree diagram, we can see the tree model generated by Jupyter Notebook is more complicated and harder to read comparing to the neat and clear tree model by RStudio. Also, the accuracy of Jupyter Notebook's model is lower than RStudio after tested in the test set.

a) Hyperparameter Tuning and Cross Validation to Decision Tree

```
fig = plt.figure(figsize=(20,10))
tree.plot_tree(tree_f, feature_names=feature_cols)
fig.savefig("decistion_tree.png")
```

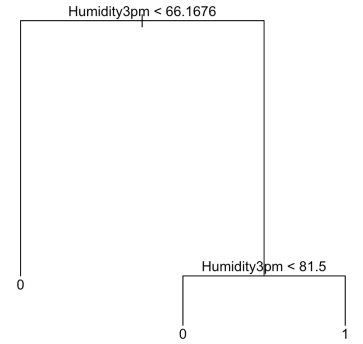


```
#Predict the test set and get the accuracy
y_pred = cv.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
[[20834  1084]
 [ 3618  2622]]
Accuracy: 0.8330137083599688
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.8330137083599688
Recall: 0.4201923076923077
```




```
                RainTomorrow
tree_tmr.pred.p     0     1
              0 21526  4421
              1   489  1722

               Accuracy : 0.8256
                 95% CI : (0.8211, 0.83)
    No Information Rate : 0.7818
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3355
```

Sensitivity: 0.280319062
Specificity: 0.977787872

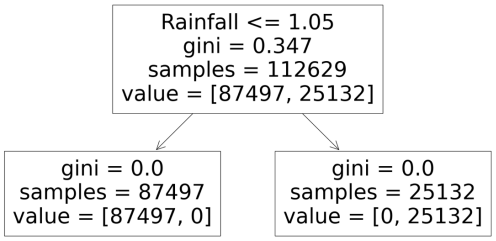
After Cross Validation to the tree model, both model in Jupyter Notebook and RStudio is simplified and neater. The Jupyter Notebook's model become better in accuracy, but the recall is lower than the original one, on the other hand, RStudio's model has no changes to the accuracy and recall.

Comparing the pruned tree model, Jupyter Notebook's model has better accuracy and recall than RStudio's.

2) Classification Tree for RainToday

Jupyter Notebook

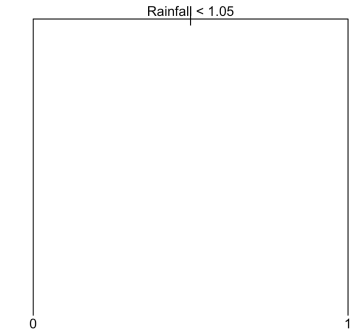
```
fig = plt.figure(figsize=(20,10))
tree.plot_tree(tree_tdy, feature_names=feature_cols)
fig.savefig("decistion_tree_for_Raintdy.png")
```




```
# Create Decision Tree classifer object
tree_tdy = DecisionTreeClassifier()

# Train Decision Tree Classifer
tree_tdy.fit(X_train,y_tdy_train)

#Predict the response for test dataset
y_tdy_pred = tree_tdy_f.predict(X_test)

cm = confusion_matrix(y_tdy_test, y_tdy_pred)
print(cm)
```

```
[[21835     0]
 [    0  6323]]
```

RStudio




```
             RainToday
tree_tdy.pred     0     1
            0 21855     0
            1     0  6303

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.7762
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
```

Both platforms have perfect accuracy and recall in the RainToday classification tree.

3) Predict the RainToday and RainTomorrow

Jupyter Notebook

```
Predict RainToday       0.0
dtype: object
Predict RainTomorrow    215.0
dtype: object
```

RStudio

```
R_predicted_RainToday       0.0
dtype: object
R_predicted_RainTomorrow    149.0
dtype: object
```

The result of prediction of RainToday on both platforms are same. However, the prediction of RainTomorrow is quite different in Jupyter Notebook and RStudio, which is 215 and 149 respectively and 18% difference.

To conclude, the Jupyter Notebook is generally perform better, which the accuracy and recall of the final classification tree are higher than RStudio. However, the classification tree in RStudio is simpler and easier to read than Jupyter Notebook's tree. Although the final classification tree in Jupyter Notebook has higher accuracy percentage, the recall score for the tree is very low, which means the number of false negative value is high or true positive value is low. The lower sensitivity in RStudio's model also explained why its RainTomorrow count is lower than Jupyter Notebook's.

KNN algorithm

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for both classification and regression tasks. It is a type of instance-based learning or lazy learning, where the algorithm memorizes the training dataset instead of learning explicit patterns from it. KNN makes predictions based on the similarity between input data points and the data points in the training set.

We have implemented the KNN algorithm requirement using two tools. The first one was implemented in python using libraries like pandas, numpy for general preprocessing and matplotlib and seaborn for the visualization portion(when required). For the main task of generating a prediction model and subsequent prediction, we use the the **sklearn** library.

Here we are using KNN to classify whether it will rain tomorrow or not by analyzing the other features that are related to the target feature (RainTomorrow). Before training a model, the data set has two binary qualitative variables RainToday and RainTomorrow (Yes-No), which we transformed into quantitative variables (1-0). The categorical variables (WindGustDir, WindDir3pm, WindDir9am), we converted each category of this columns into an additional column, with a dummy variable (1-0). We used the **MinMaxScaler**() function from the **sklearn** preprocessing module to scale numerical features to a specific range, (i.e. between 0 and 1). Then we calculated the pairwise correlation between columns using the corr() function of pandas library so that we can drop features/columns that had very negligible correlation with the target variable (RainTomorow). This was done to reduce the size of the data and thus reduce the computational time required for training the model.

```
#Droping data that are not relevant to the analysis as Date, Location and the rest of the missing values (if theres is any left after cleaning)

df= df.drop(columns=['Date','Location'],axis=1)
df = df.dropna(how='any')


# Replace No and Yes for 0 and 1 in RainToday and RainTomorrow

df['RainToday'].replace({'No': 0, 'Yes': 1},inplace = True)
df['RainTomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)
# Categorical variables WindGustDir, WindDir3pm and WindDir9am in dummy variables for each category.
categoric_c = ['WindGustDir', 'WindDir3pm', 'WindDir9am']
datafinal = pd.get_dummies(df, columns=categoric_c)
print(datafinal.shape)
datafinal.replace({False: 0, True: 1}, inplace=True)
datafinal.head()



#data normalization
from sklearn import preprocessing
```

```
standa = preprocessing.MinMaxScaler()
standa.fit(datafinal)
datafinal = pd.DataFrame(standa.transform(datafinal), index=datafinal.index, columns=datafinal.columns)
datafinal.head()

# eliminating variables that have a correlation less than 0.5% with the variable of interest(RainTomorrow)
corr = datafinal.corr()
corr1 = pd.DataFrame(abs(corr['RainTomorrow']),columns = ['RainTomorrow'])
nonvals = corr1.loc[corr1['RainTomorrow'] < 0.005]
print('Var correlation < 0.5%',nonvals)
nonvals = list(nonvals.index.values)

# We extract variables with correlation less than 5% and drop those variables from the table
datafinal1 = datafinal.drop(columns=nonvals,axis=1)
print('Data Final',datafinal1.shape)
```

To implement a KNN algorithm, we must consider the number of neighbors (given by K), as the algorithm assigns the class label that is most common among its k-nearest neighbors to the input data point. The choice of the number of neighbors, 'k,' is crucial. A smaller 'k' value makes the model sensitive to noise, while a larger 'k' value can smooth out decision boundaries but might miss local patterns. Thus, finding the most optimal K is an important task before the model is generated. For this, we use the cross-fold validation to find the optimal number of neighbors to consider. Since the dataset was too large, we made a subset of the data that included 500 random rows from the original data. We split the data in Independent Features and Dependent Features. For this we used the **KNeighborsClassifier**() function from the neighbors module of the sklearn library **and cross_val_score()** function from the **model_selection** module of the sklearn library. The reason for using the subset and not the entire data was because the dataset would be very large for the cross-fold validation and computationally take a lot of time for convergence. We took the k value with the maximum cross_val_score. The cross_val_score() used accuracy as the metric for comparison.

```
#load the class from teh sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
import math

subset_df = datafinal1.sample(n=500, random_state=42)
subset_df

# Splitting data into Independent Features and Dependent Features:
Y = subset_df['RainTomorrow']
X = subset_df.drop(columns=['RainTomorrow'])
```

```
k_values = list(range(1,50))
# Initialize an empty list to store the cross-validation scores
cross_val_scores = []

# Loop through each k value and perform cross-validation
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k,n_jobs=-1)
    scores = cross_val_score(knn, X, Y, cv=10,scoring='accuracy')  # 10-fold cross-validation
    cross_val_scores.append(np.mean(scores))

optimal_k = k_values[np.argmax(cross_val_scores)]
print("Optimal value of k is:", optimal_k)
```

After we estimated the optimal K value, we splited the whole dataset into independent and dependen features as we did with the subset dataset. We use the train_test_split function from the sklearn.model_selection to divide the data into test and train data with 20% used for the test set and 80% used for training or KNN model. Then we train the model using the train data and use the model with the test data for testing.

```
from sklearn.model_selection import train_test_split

# Splitting data into Independent Features and Dependent Features:
Y = datafinal1['RainTomorrow']
X = datafinal1.drop(columns=['RainTomorrow'])
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=9)


# We define the model
knnclassifier = KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1)


# We train model
knnclassifier.fit(X_train, Y_train)

# We predict target values using the trained model
Y_predict = knnclassifier.predict(X_test)
```

Then we use the confusion_matric() from the metric module of metric module of sklearn to generate a confusion matrix and plot the confusion matrix using heatmap from the seaborn data visualization module. We use other functions from the same metric module to calculate other measurables like f1score, reacall and precision as shown below:

```
from sklearn.metrics import confusion_matrix
# The confusion matrix
knncla_cm = confusion_matrix(Y_test, Y_predict)
```

```
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(knncla_cm, annot=True, linewidth=0.7, linecolor='red', fmt='g', ax=ax, cmap="PiYG")
plt.title('KNN Classification Confusion Matrix')
plt.xlabel('Y predict')
plt.ylabel('Y test')
plt.show()
```

```
# Test score
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
score_knnclassifier= knnclassifier.score(X_test, Y_test)
print(score_knnclassifier)

precision = precision_score(Y_test, Y_predict)
recall = recall_score(Y_test, Y_predict)
f1 = f1_score(Y_test, Y_predict)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

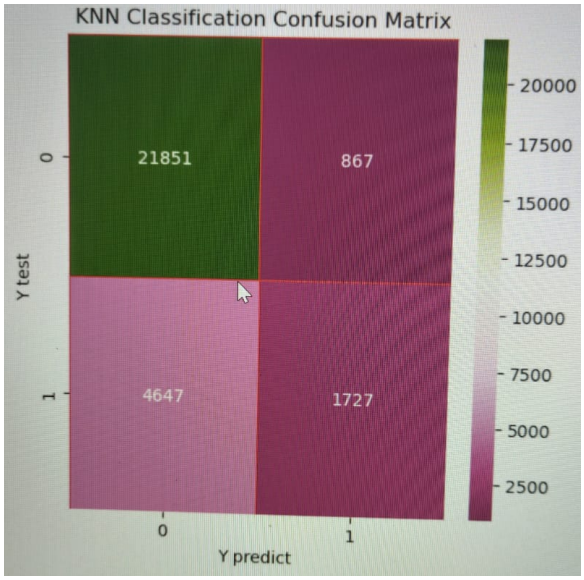
The same KNN algorithm was executed in R**.** I performed the same preprocessing task in R where we used the tidyverse library to define a tibble in place of dataframe. Then I used a subset of the original dataframe (tibble in our case). Using this subset, we use the **caret** library to run a cross-fold validation to find the optimal value of neighbors to consider when training the KNN model. Using the optimal k(neighbors), we used the **train()** function from the caret library to train the model and use the trained model to perform the prediction on the test data using the **predict()** function.

```
setwd('C:/Users/mohun/Desktop/proposal/knn in R')
library(tidyverse)
library(caret)
data <- read_csv('finaldataKNN.csv', show_col_types = FALSE)
head(data,3)

#chekcing if any row has a null value in any of the column
rows_with_null <- sum(!complete.cases(data))
print(rows_with_null)

#determine correlation of every column with the the RainTomorrow column
data_cor <- cor(data[ ,colnames(data)!="RainTomorrow"],data$RainTomorrow)
data_cor

#Find column indices with correlation less than the threshold (5%)
columns_to_drop <- which(data_cor < 0.005)
print(columns_to_drop)

column_name <- names(data)[20]
print(column_name)

value_to_remove <- 20
```

```
columns_to_drop<-columns_to_drop[columns_to_drop != value_to_remove]
print(columns_to_drop)


#drops column with correlation less than 0.005
filtered_data <- data[, -columns_to_drop]
print(filtered_data)
#check for any null values in any column of the filtered dataframe
print(rows_with_null <- sum(!complete.cases(filtered_data)))
filtered_data$RainTomorrow <- factor(data$RainTomorrow, levels = c(0, 1), labels = c("no", "yes"))
print(filtered_data)

# Randomly sample a subset from the filtereddata
subset_size <- 500

random_subset_indices <- sample(1:nrow(filtered_data), subset_size)
random_subset <- data[random_subset_indices, ]
random_subset$RainTomorrow <- factor(random_subset$RainTomorrow, levels = c(0, 1), labels = c("no", "yes"))

# Perform k-fold cross-validation on the random subset

levels(random_subset$RainTomorrow)

cv <- trainControl(method = "cv", number = 10)

knn_model <- train(RainTomorrow ~ ., data = random_subset, method = "knn", trControl = cv, tuneLength = 30)

knn_model

plot(knn_model)
library(caret)

set.seed(3033)
intrain <- createDataPartition(y =filtered_data$RainTomorrow, p= 0.8, list = FALSE)
training <- filtered_data[intrain,]
testing <- filtered_data[-intrain,]

param_grid <- expand.grid(k = 1:24)
knn_model <- train(RainTomorrow ~ ., data = training, method = "knn", trControl = cv,  tuneGrid = param_grid,tuneLength = 4)
predictions <- predict(knn_model, newdata = testing)
cf <- confusionMatrix(predictions, testing$RainTomorrow)
print(cf) setwd('C:/Users/mohun/Desktop/proposal/knn in R')
library(tidyverse)
library(caret)
data <- read_csv('finaldataKNN.csv', show_col_types = FALSE)
head(data,3)
```

```
#chekcing if any row has a null value in any of the column
rows_with_null <- sum(!complete.cases(data))
print(rows_with_null)
#determine correlation of every column with the the RainTomorrow column
data_cor <- cor(data[ ,colnames(data)!="RainTomorrow"],data$RainTomorrow)
data_cor

#Find column indices with correlation less than the threshold
columns_to_drop <- which(data_cor < 0.005)
print(columns_to_drop)

column_name <- names(data)[20]
print(column_name)

value_to_remove <- 20
columns_to_drop<-columns_to_drop[columns_to_drop != value_to_remove]
print(columns_to_drop)


#drops column with correlation less than 0.005
filtered_data <- data[, -columns_to_drop]
print(filtered_data)
#check for any null values in any column of the filtered dataframe
print(rows_with_null <- sum(!complete.cases(filtered_data)))
filtered_data$RainTomorrow <- factor(data$RainTomorrow, levels = c(0, 1), labels = c("no", "yes"))
print(filtered_data)

subset_size <- 500
# Randomly sample a subset from the filtereddata
random_subset_indices <- sample(1:nrow(filtered_data), subset_size)
random_subset <- data[random_subset_indices, ]
random_subset$RainTomorrow <- factor(random_subset$RainTomorrow, levels = c(0, 1), labels = c("no", "yes"))
#check for any null values in any column of the filtered dataframe
levels(random_subset$RainTomorrow)


cv <- trainControl(method = "cv", number = 10)
# Perform k-fold cross-validation on the random subset
knn_model <- train(RainTomorrow ~ ., data = random_subset, method = "knn", trControl = cv, tuneLength = 30)

knn_model # we see that theoptimal k value for the knn is 21

plot(knn_model)
library(caret)

set.seed(3033)
intrain <- createDataPartition(y =filtered_data$RainTomorrow, p= 0.8, list = FALSE)
```

```
training <- filtered_data[intrain,]
testing <- filtered_data[-intrain,]

param_grid <- expand.grid(k = 1:24)
knn_model <- train(RainTomorrow ~ ., data = training, method = "knn", trControl = cv,  tuneGrid = param_grid,tuneLength = 4)
predictions <- predict(knn_model, newdata = testing)
cf <- confusionMatrix(predictions, testing$RainTomorrow)
print(cf)
```

KNN in python using scikit-learn library	**KNN in R using the Caret library**
Both libraries offer functionalities for KNN, including classification and regression tasks.	
scikit-learn in Python is well-documented, widely used, and integrates seamlessly with other Python libraries, making it a popular choice for machine learning in Python.	caret in R provides a more extensive set of algorithms and models, making it versatile for various machine learning tasks.
Comparison based on our use case	
Parameter tuning Cross fold validation generated 4 as the optimal neighbors to consider when performing KNNclassier	**Parameter tuning:** Cross fold validation generated 21 as the optimal neighbors to consider when performing KNNclassier
Prediction accuracy was 81.05%	Prediction accuracy was 82.3%
Confusion matrix: 	**Confusion matrix:** Confusion Matrix and Statistics Reference Prediction no yes no 21633 4067 yes 1083 2308 Accuracy : 0.823 95% CI : (0.8185, 0.8273) No Information Rate : 0.7809 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.378 Mcnemar's Test P-Value : < 2.2e-16
Compared to Caret library in R, the scikit learning library was little complex for us	implementation of KNN using caret library in R was simpler than scikit learn library for our use case.

K-Means Clustering

K-Means clustering is an unsupervised machine learning algorithm used for partitioning a dataset into distinct, non-overlapping subsets (clusters). The algorithm aims to group similar data points together and discover underlying patterns in the data. Each cluster is represented by its centroid, which is the mean of all data points in that cluster. We used the K-Means clustering to cluster/group together different locations/cities based on the four season's average rainfall. We derived the seasons from the Date variable and grouped the data based on Location and season and calculated the mean rainfall for each location in each season.

```
#function to get seasons from the Date field
def get_season(month):
    if month in [12, 1, 2]:  # Dec, Jan, Feb -> Summer
        return 'Summer'
    elif month in [3, 4, 5]:  # Mar, Apr, May -> Autumn
        return 'Autumn'
    elif month in [6, 7, 8]:  # Jun, Jul, Aug -> Winter
        return 'Winter'
    else:  # Sep, Oct, Nov -> Spring
        return 'Spring'
```

```
# Apply the get_season function to the 'Date' column and create a new 'Season' column
df['Season'] = df['Date'].dt.month.apply(get_season)
df
```

```
# Group data by 'Location' and 'Season', then calculate the average rainfall for each group
average_rainfall_by_season = df.groupby(['Location', 'Season'])['Rainfall'].mean().reset_index()
average_rainfall_by_season
```

Then we pivot the dataframe such that the seasons (Autumn, Winter, Spring and Summer) becomes the columns using the pivot method of pandas library.

```
# Pivot the DataFrame to have columns for each season's rainfall values
average_rainfall_by_season = average_rainfall_by_season.pivot(index='Location', columns='Season', values='Rainfall').reset_index()
average_rainfall_by_season
```

We use the **LabelEncoder** utility class from the preprocessing module of sklearn library to encode the categorical field (Location) integers labels.

```
from sklearn.preprocessing import LabelEncoder
```

```
# Encode categorical data (Location)

location_label_encoder = LabelEncoder()
average_rainfall_by_season['Location'] =
location_label_encoder.fit_transform(average_rainfall_by_season['Location'])
# Store the classes for later inverse transformation
location_classes = location_label_encoder.classes_

average_rainfall_by_season
```

In this algorithm too, finding the best number of groups for grouping the data was necessary. Thus, we tried to find the best number of groups using the elbow method. The elbow method involved plotting the within-cluster sum of squares (WCSS) against the number of clusters. WCSS measures the compactness of the clusters. As the number of clusters increases, WCSS decreases. The idea was to identify the "elbow" point on the plot, where the rate of decrease sharply changes. This point is often considered the optimal number of clusters/groups. We analyzed the plotted graph and estimated the optimum number of clusters to be 8.

```
wcss = []
for k in range(2, 50):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(average_rainfall_by_season)
    wcss.append(kmeans.inertia_)
# Calculate the first and second derivatives of the WCSS curve
first_derivative = [wcss[i] - wcss[i-1] for i in range(1, len(wcss))]
second_derivative = [first_derivative[i] - first_derivative[i-1] for i in range(1, len(first_derivative))]

# Plot the elbow curve with the identified elbow point
plt.figure(figsize=(8, 6))
plt.plot(range(2, 50), wcss, marker='o', linestyle='--')
plt.plot(elbow_index, wcss[elbow_index - 1], marker='o', markersize=8, color='red', label='Elbow
Point')
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.title('Elbow Method for Optimal K')
plt.legend()
plt.show()
```

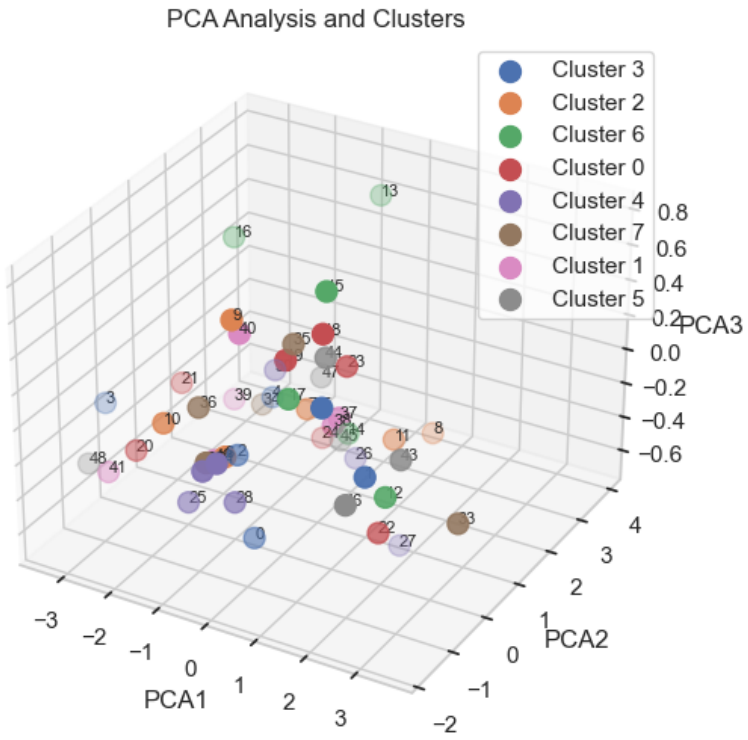
Using the **KMeans** function from the cluster module of sklearn library, we performed the clustering on the data and added a new field called Clusters. Then we decoded back our location field from numerical labels back to original Labels.

```
from sklearn.cluster import KMeans

#Choose the optimal number of clusters (8 from elbow method)
optimal_k = 8

#Perform K-means clustering
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(average_rainfall_by_season)

#Add cluster labels to the original DataFrame
average_rainfall_by_season['Cluster'] = clusters
average_rainfall_by_season

#Decode 'Location' columns back to original names
average_rainfall_by_season['Location']=location_label_encoder.inverse_transform(average_rainfall_
by_season['Location'])
average_rainfall_by_season
```

We tried to plot the dataset to display the clusters but there were many variables that needed to be considered when plotting the graph which was not possible within the 3-D graph. Therefore, we performed the Principal Component Analysis on the four seasons to transform into 3 components for 3D visualization.

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

df=average_rainfall_by_season
# Extract seasonal data for PCA
seasonal_data = df[['Autumn', 'Spring', 'Summer', 'Winter']]

# Standardize the data (important for PCA)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(seasonal_data)

# Perform PCA with 3 components (for 3D visualization)
pca = PCA(n_components=3)
pca_features = pca.fit_transform(scaled_data)

# Add PCA features to DataFrame
df['PCA1'] = pca_features[:, 0]
df['PCA2'] = pca_features[:, 1]
df['PCA3'] = pca_features[:, 2]

# 3D Scatter Plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
```

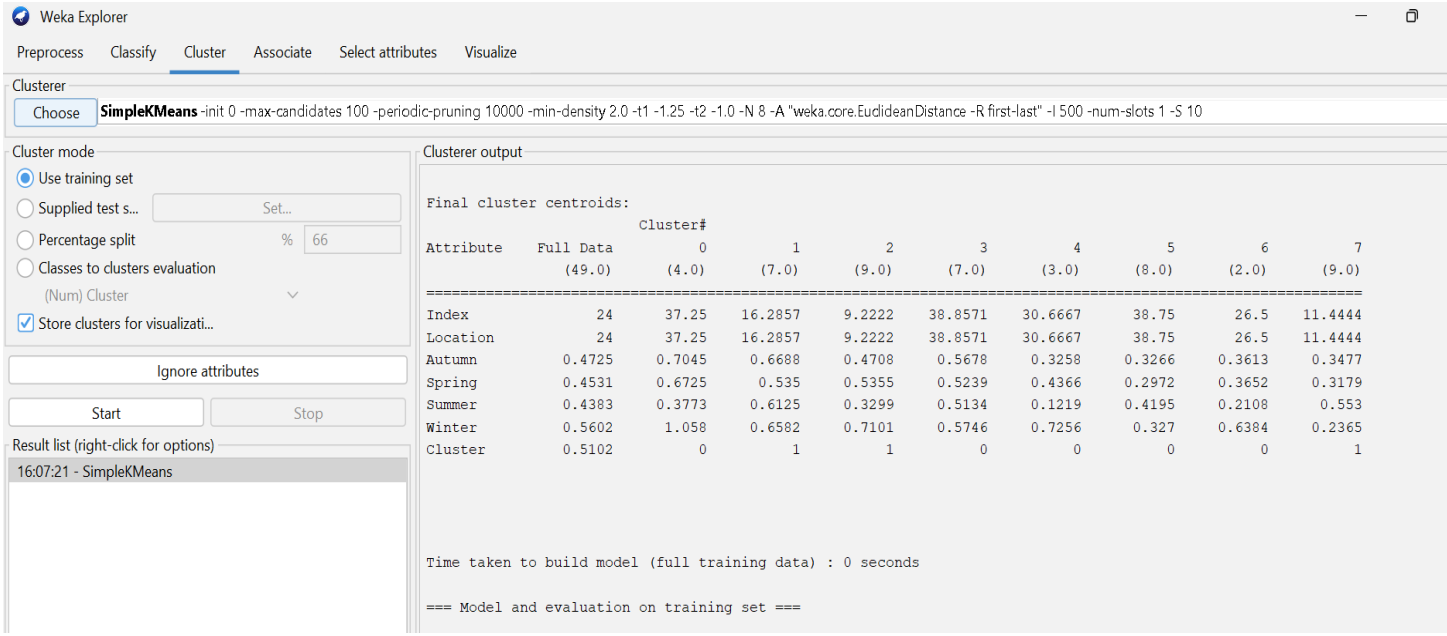
```
# Plotting points based on PCA components and cluster labels
for cluster_label in df['Cluster'].unique():
    cluster_data = df[df['Cluster'] == cluster_label]
    ax.scatter(cluster_data['PCA1'], cluster_data['PCA2'], cluster_data['PCA3'],
               label=f'Cluster {cluster_label}', s=100)

# Plotting location names
for i, location in enumerate(df['Location']):
    ax.text(df['PCA1'][i], df['PCA2'][i], df['PCA3'][i], location, fontsize=8)

ax.set_xlabel('PCA1')
ax.set_ylabel('PCA2')
ax.set_zlabel('PCA3')
ax.set_title('PCA Analysis and Clusters')
ax.legend()
plt.show()
```



KMeans clustering using the weka software suite:

We exported the dataframe that contained the location and average rainfall for each seasons into the csv file by using the **to_csv()** method of pandas library. We then used the Weka Explorer tool to load this csv and convert into ARFF(Attribute-Relation File Format). We selected the required fields/columns required for the clustering from the preprocessing tab of the Weka GUI. Then under

the cluster tab, we choose the SimpleKMeans as the clusterer and set the **numClusters** value to 8 from the generic object editor. Then we started the clustering algorithm and showed the generated clusters as below:



We could also get the cluster visualization by going under the Visualize tab of the Weka GUI

KMeans clustering using sklearn library	KMeans clustering suing weka
One needs to know the basic python programming with the ability to use the required functions and classes	Clustering can be performed through GUI
Based on python	Based on Java
Provides a high degree of flexibility, allowing users to customize various parameters of the KMeans algorithm such as the number of clusters, initialization methods, and distance metrics. Users have fine-grained control over the algorithm's behavior.	While Weka offers some customization options, it might not be as extensive as scikit-learn. Advanced users might find scikit-learn more suitable when they need highly customized KMeans clustering.
We must write the visualization code to visualize the clusters	Visuals are auto generated when we use the GUI tool for visualization and clustering.
Comparison based on our use case	
We get the clusters but do not know what specific values on which the clusters are created.	We can see how each value have been distributed for each of the clusters when using the WEKA GUI

Summary

Effective Teamwork: One thing that worked really well was how we all collaborated as a team. We talked openly and helped each other with our ideas, which made it easier to make decisions and solve problems together.

Good Model Performance: It was quite surprising to see that both the R and Jupyter models we made for linear regression worked really well. They were accurate in predicting things, and the numbers we used to measure their accuracy showed that.

Useful Pictures: We were also surprised by how helpful pictures and graphs were in our project. They made it much easier to understand and explain our data and results.

Flexibility of Tools: Another good thing was that both R and Jupyter were flexible tools. R was easy to use for quick testing, and Jupyter gave us more options for advanced analysis.

Learning from Challenges: We faced some difficulties, especially when dealing with data and adjusting our models. But these challenges taught us a lot and helped us become better at what we were doing.

Data Challenges: One of the challenges we faced was with the dataset itself. It turned out to be more messy and incomplete than we initially anticipated. This made the data pre-processing stage more time-consuming and complex than we had hoped.

Model Complexity: While our linear regression models performed well, we encountered difficulties when we tried to implement more complex models. These models sometimes didn't improve predictions significantly and added unnecessary complexity to our analysis.

Overall, the task was very demanding. We tried to implement every algorithm as per our best efforts. Type conversion was one of the major issues faced when creating the different models. We had tough time with the visualization as well. The implementation of the model was easy in almost all the case but the performance tuning was quit difficult as well as subjective in our case as we did not the proper domain knowledge which is a good asset in performance tuning.

Reference

Bhalla, D. (n.d.). *Decision tree in R : Step by step guide*. ListenData. https://www.listendata.com/2015/04/decision-tree-in-r.html

Brownlee, J. (2019, December 11). *How to implement the decision tree algorithm from scratch in Python*. MachineLearningMastery.com. https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

Commonwealth of Australia. (2023). Rain in Australia [Dataset]. https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package

Confusionmatrix: Create a confusion matrix. RDocumentation. (n.d.). https://www.rdocumentation.org/packages/caret/versions/3.45/topics/confusionMatrix

John & Dave, A. (1967, April 1). *How to obtain confusion matrix using caret package?*. Stack Overflow. https://stackoverflow.com/questions/65604276/how-to-obtain-confusion-matrix-using-caret-package

Johnson, D. (2023, September 30). *Decision tree in R: Classification tree with example*. Guru99. https://www.guru99.com/r-decision-trees.html

Max, K. (2023, March 31). *Confusionmatrix: Create a confusion matrix in caret: Classification and regression training*. confusionMatrix: Create a confusion matrix in caret: Classification and Regression Training. https://rdrr.io/cran/caret/man/confusionMatrix.html

Minkoo S. (1959, May 1). *R's randomforest can not handle more than 32 levels. what is workaround?* Cross Validated. https://stats.stackexchange.com/questions/49243/rs-randomforest-can-not-handle-more-than-32-levels-what-is-workaround

Raj, A. (2021, December 11). *An exhaustive guide to decision tree classification in python 3.X*. Medium. https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f

Ski, P. (2020, June 22). *Visualize a decision tree in 4 ways with Scikit-Learn and python*. MLJAR. https://mljar.com/blog/visualize-decision-tree/

Thevapalan, A., & Le, J. (2023, June 1). *R decision trees tutorial: Examples & code in R for Regression & Classification*. DataCamp. https://www.datacamp.com/tutorial/decision-trees-R#

YouTube. (2018, May 1). *Hyperparameter tuning and cross validation to decision tree classifier (machine learning by python)*. YouTube. https://www.youtube.com/watch?time_continue=541&v=dA_x2xHTYQE&embeds_referring_euri=https%3A%2F%2Fwww.google.com%2F&source_ve_path=MTM5MTE3LDM2ODQyLDI4NjYzLDEzNzcyMSwxMzkxMTcsMTM5MTE3LDI4NjYzLDEzNzcyMSwyODY2MywyODY2Ng&feature=emb_logo&themeRefresh=1

Contribution

Chimmy Rai:

Played a key role in our project by focusing on K-Nearest Neighbors (KNN) analysis. My main contributions revolved around understanding, implementing, and fine-tuning KNN algorithms. I designed experiments to assess KNN's performance, compared it with linear regression, and created visualizations to help the team understand the results. I made sure everyone on the team understood how KNN works and actively collaborated to integrate our findings. Overall, I brought KNN expertise to the team, and my work added an important dimension to our project, allowing us to explore different machine learning techniques.

Erdenechimeg Darjaa:

I've been working with the K-Nearest Neighbors (KNN) method on our weatherAUS data. I first cleaned up our data, changing words to numbers and filling in missing bits. I then set up the KNN tool with 5 neighbors and trained it. After that, I used it to guess rain patterns on a test set. I checked how well it did using accuracy measures and other simple checks. Moving ahead, we could try adjusting some settings to see if we get better results.

Ruvimbo Melody Chitagu:

My individual contribution to this project encompassed several key aspects. I was responsible for data collection and preprocessing, which included sourcing and cleaning the dataset used for our linear regression experiments. Additionally, I played a central role in implementing the linear regression models in both R and Jupyter environments, as well as conducting the experiments and recording the results. I also contributed significantly to the analysis of the results, including the interpretation of metrics and the creation of visualizations. Finally, I actively collaborated with team members by providing insights and feedback during discussions and brainstorming sessions. Overall, my contributions were around data preparation, model implementation, experimentation, and collaboration, ensuring the successful completion of this project.

Siu Wai Lo:

I took a leading role in conducting decision tree analysis in both Jupyter and R Studio environments for our project. My contributions encompassed implementing decision tree models, preprocessing the data, evaluating model performance, comparing the outcomes of the two environments, creating visualizations, and actively collaborating with team members. These efforts collectively enhanced the project's depth and understanding of decision tree techniques in different software environments.

Appendix

Decision Tree Classifier

Jupyter Notebook source codes

Classification Tree for RainTomorrow

reference: https://saturncloud.io/blog/how-to-use-cross-validation-with-decision-trees-in-scikitlearn/

```
#import lib
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
```

```
#extract the rows containing NaN in RainTmr & RainTdy to a new dataframe, use decision tree to predict their value l
df_predict_tmr=df_weather[df_weather['RainTomorrow'].isnull()]
df_predict_tdy=df_weather[df_weather['RainToday'].isnull()]
```

```
#remove the rows with NaN
clean_df = df_weather.dropna()
clean_df
```

	Year	Month	Day	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	...	Humidity9am	Humidity3pm	Pressure9am	Pressu
0	2008	12	01	Albury	13.4	22.9	0.6	7.078959	8.629676	W	...	71.0	22.0	1007.7	
1	2008	12	02	Albury	7.4	25.1	0.0	7.078959	8.629676	WNW	...	44.0	25.0	1010.6	
2	2008	12	03	Albury	12.9	25.7	0.0	7.078959	8.629676	WSW	...	38.0	30.0	1007.6	
3	2008	12	04	Albury	9.2	28.0	0.0	7.078959	8.629676	NE	...	45.0	16.0	1017.6	
4	2008	12	05	Albury	17.5	32.3	1.0	7.078959	8.629676	W	...	82.0	33.0	1010.8	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145454	2017	06	20	Uluru	3.5	21.8	0.0	2.741813	6.264984	E	...	59.0	27.0	1024.7	
145455	2017	06	21	Uluru	2.8	23.4	0.0	2.741813	6.264984	E	...	51.0	24.0	1024.6	
145456	2017	06	22	Uluru	3.6	25.3	0.0	2.741813	6.264984	NNW	...	56.0	21.0	1023.5	

```
print([column_name for column_name in df_weather.columns if df[column_name].dtype != 'O'])
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Hum
idity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'RainToday',
'RainTomorrow']
```

```
#split dataset in features and target variable
feature_cols = ['Month', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am
X = clean_df[feature_cols] # Features
y = clean_df.RainTomorrow # Target variable
```

```
X
```

```
...
```

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1) # 80% training and 20% test
```

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf=clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[18790  3128]
 [ 2917  3323]]
```

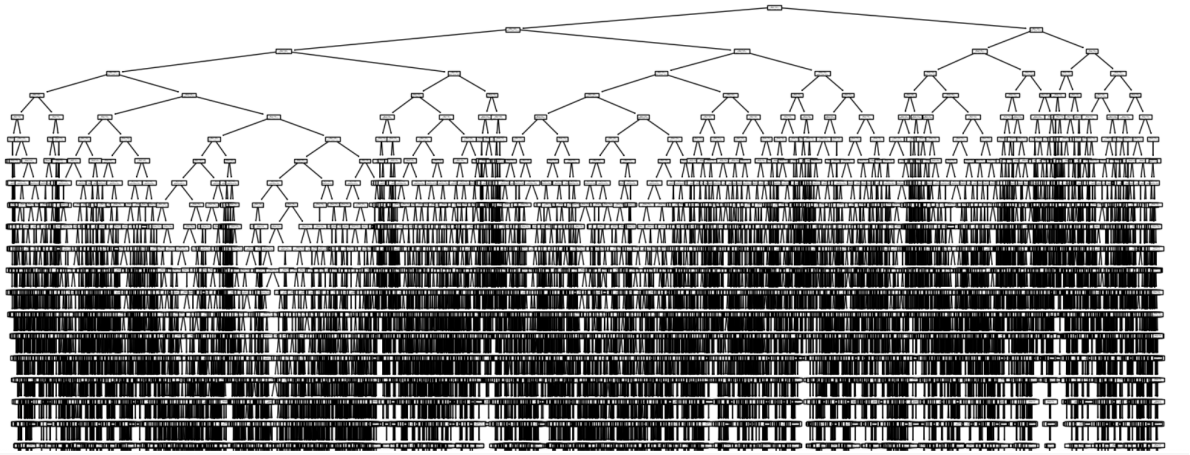
```
#and get the accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.7853185595567868
Recall: 0.5325320512820513
```

```
#print the tree
tree.plot_tree(clf, feature_names=feature_cols)
```

...

```
fig = plt.figure(figsize=(20,15))
tree.plot_tree(clf, feature_names=feature_cols)
fig.savefig("decistion_tree_before_tune.png")
```



Hyperparameter Tuning and Cross Validation to Decision Tree

reference: https://www.youtube.com/watch?v=dA_x2xHTYQE

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix
from scipy.stats import randint
```

```
#use RandomizedSearchCV tpo find the optimized model
param_dist = {"max_depth": [3, None],
              "min_samples_leaf": randint(1,9),
              "criterion": ["gini", "entropy"]}
tree = DecisionTreeClassifier()
cv = RandomizedSearchCV(tree, param_dist, cv=5)
cv.fit(X_train, y_train)
print("Tuned Tree parameters: {}".format(cv.best_params_))
print("Best Score is: {}".format(cv.best_score_))
```

```
Tuned Tree parameters: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 2}
Best Score is: 0.8316685612669261
```

```
#Predict the test set and get the accuracy
y_pred = cv.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
[[20834  1084]
 [ 3618  2622]]
Accuracy: 0.8330137083599688
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

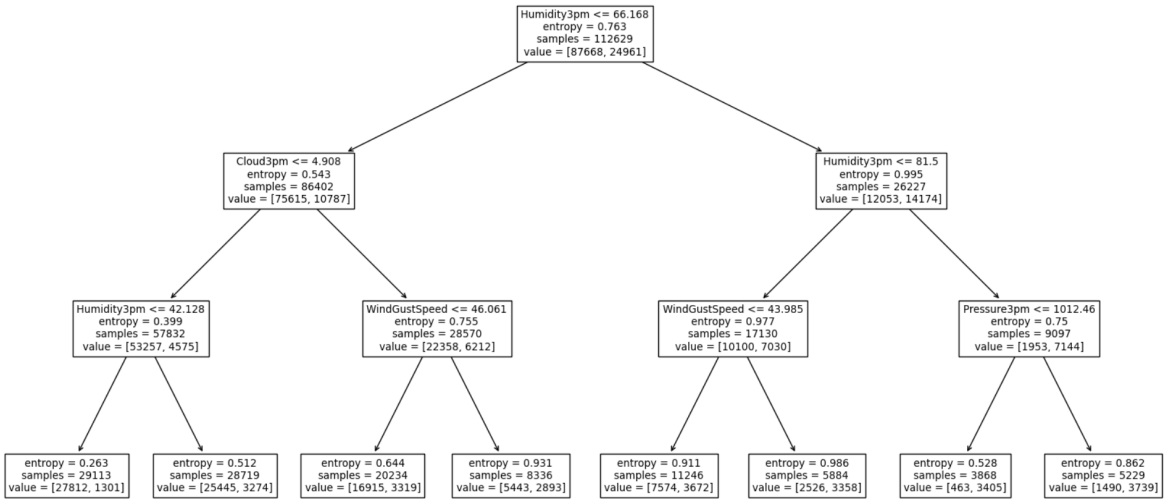
```
Accuracy: 0.8330137083599688
Recall: 0.4201923076923077
```

```
#tuned tree
tree_f = DecisionTreeClassifier(criterion= 'entropy', max_depth= 3, min_samples_leaf= 2)
tree_f.fit(X_train,y_train)
```

```
from sklearn import tree
#print tree
tree.plot_tree(tree_f, feature_names=feature_cols)
```

...

```
fig = plt.figure(figsize=(20,10))
tree.plot_tree(tree_f, feature_names=feature_cols)
fig.savefig("decistion_tree.png")
```



Classification Tree for RainToday

```
y_tdy = clean_df.RainToday # Target variable
```

```
# Split dataset into training set and test set
X_train, X_test, y_tdy_train, y_tdy_test = train_test_split(X, y_tdy, test_size=0.2, random_state=1)
```

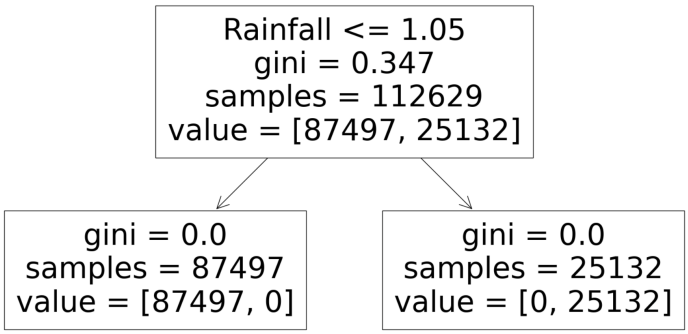
```
# Create Decision Tree classifer object
tree_tdy = DecisionTreeClassifier()

# Train Decision Tree Classifer
tree_tdy.fit(X_train,y_tdy_train)

#Predict the response for test dataset
y_tdy_pred = tree_tdy_f.predict(X_test)

cm = confusion_matrix(y_tdy_test, y_tdy_pred)
print(cm)
```

```
[[21835     0]
 [    0  6323]]
```

```
#and get the accuracy
print("Accuracy:",metrics.accuracy_score(y_tdy_test, y_tdy_pred))
print("Recall:",metrics.recall_score(y_tdy_test, y_tdy_pred))
```

```
Accuracy: 1.0
Recall: 1.0
```

```
fig = plt.figure(figsize=(20,10))
tree.plot_tree(tree_tdy, feature_names=feature_cols)
fig.savefig("decistion_tree_for_Raintdy.png")
```



Predict the RainToday and RainTomorrow

```
#RainToday
# The final tree model
tree_tdy = DecisionTreeClassifier()
tree_tdy.fit(X_train,y_tdy_train)
X_tdy = df_predict_tdy[feature_cols]
#Predict the RainToday
df_predict_tdy["Predict RainToday"] = tree_tdy.predict(X_tdy)
df_predict_tdy.head()
```

...

```
#RainTomorrow
# The final tree model
X_tmr = df_predict_tmr[feature_cols]
#Predict the RainToday
df_predict_tmr["Predict RainTomorrow"] = tree_f.predict(X_tmr)
df_predict_tmr.head()
```

RStudio source codes

```
#Decision Tree

library(tidyverse)

library(ggplot2)

library(ISLR)

require(tree)

library(caret)

#1.Read the dataset into a data frame

weather <- read.csv("removedNaN_for_decision_tree.csv", stringsAsFactors=TRUE)

#2.Create two sets: training (80% observations of the drug dataset) and test (20% observations of the drug dataset) sets.

training = nrow(weather)*0.8

set.seed(101)

train_tdy=sample(1:nrow(weather), training)

training_tdyset = weather[train_tdy,]

testing_tdyset = weather[-train_tdy,]


#3.Create a classification tree using the training set and calculate the classification accuracy of the tree using the test set.

#for RainToday

#train set

tree.tdy =
tree(as.factor(RainToday)~Month+MinTemp+MaxTemp+Rainfall+Evaporation+Sunshine+WindGustDir+WindGustSpeed+WindDir9am+WindDir3pm+WindSpeed9am+WindSpeed3pm+Humidity9am+Humidity3pm+Pressure9am+Pressure3pm+Cloud9am+Cloud3pm+Temp9am+Temp3pm,

        data = weather, subset = train_tdy)

plot(tree.tdy)

text(tree.tdy, pretty=0)

#predict

tree_tdy.pred = predict(tree.tdy, weather[-train_tdy,], type="class")

cm <- caret::confusionMatrix(with(weather[-train_tdy,], table(RainToday,tree_tdy.pred)))

print(cm)


#for RainTmr
```

```
set.seed(102)

train_tmr=sample(1:nrow(weather), training)

training_tmrset = weather[train_tmr,]

testing_tmrset = weather[-train_tmr,]

#train set

tree.tmr =
tree(as.factor(RainTomorrow)~Month+MinTemp+MaxTemp+Rainfall+Evaporation+Sunshine+WindGustDir
+WindGustSpeed+WindDir9am+WindDir3pm+WindSpeed9am+WindSpeed3pm+Humidity9am+Humidity3
pm+Pressure9am+Pressure3pm+Cloud9am+Cloud3pm+Temp9am+Temp3pm,

        data = weather, subset = train_tmr)

plot(tree.tmr)

text(tree.tmr, pretty=0)

#predict

tree_tmr.pred = predict(tree.tmr, weather[-train_tmr,], type="class")

with(weather[-train_tmr,], table(tree_tmr.pred,RainTomorrow))

cm_tmr <- caret::confusionMatrix(with(weather[-train_tmr,], table(RainTomorrow,tree_tmr.pred)))

print(cm_tmr)

#4.Use cross-validation to prune the tree (tree from Step 3) optimally. You can use the misclassification error
as the basis for pruning. Calculate the classification accuracy of the pruned tree using the test set.

cv.tmr = cv.tree(tree.tmr, FUN = prune.misclass)

cv.tmr

plot(cv.tmr)

#best class=3

prune.tmr = prune.misclass(tree.tmr, best=4)

plot(prune.tmr)

text(prune.tmr,pretty=0)

#classification accuracy

tree_tmr.pred.p = predict(prune.tmr, weather[-train_tmr,], type="class")

with(weather[-train_tmr,], table(tree_tmr.pred.p,RainTomorrow))

cm_tmr.p <- caret::confusionMatrix(with(weather[-train_tmr,], table(RainTomorrow,tree_tmr.pred.p)))

print(cm_tmr.p)

#5.Predict the RainTmr

#read csv

weather_raintmr <- read.csv("RainTmr_Predict.csv", stringsAsFactors=TRUE)
```

```
tree_raintmr.pred = predict(prune.tmr, weather_raintmr, type="class")
#write csv
weather_R_raintmr <- weather_raintmr %>%
  mutate( "R_predicted_RainTomorrow" = tree_raintmr.pred)
write.csv(weather_R_raintmr,file='R_RainTmr_Predict.csv', row.names=FALSE)

#6.Predict the RainTdy
#read csv
weather_raintdy <- read.csv("RainTdy_Predict.csv", stringsAsFactors=TRUE)
tree_raintdy.pred = predict(tree.tdy, weather_raintdy, type="class")
#write csv
weather_R_raintdy <- weather_raintdy %>%
  mutate( "R_predicted_RainToday" = tree_raintdy.pred)
write.csv(weather_R_raintdy,file='R_RainTdy_Predict.csv', row.names=FALSE)
```