

Machine Learning Project 2024

Lucas Jakin

Saša Nanut

Luca Marega

2024-07-12

Table of contents

Predicting next-day rain in Australia	1
About the dataset	2
Exploratory Data Analysis	2
Categorical values	7
Numerical values	8
Multicollinearity	21
Modeling	23
Artificial Neural Network	23
CatBoost	29
Logistic Regression	32
Conclusion	36

Predicting next-day rain in Australia

Introduction

As a group we decided to take on the **first project type**. The project focuses on utilizing DM & ML algorithms to address a specific problem chosen from Kaggle. The Goal of the project is to address the classification problem by utilizing more than one classification algorithm, in order to do a systematic experimentation with different algorithms to identify in what they differ and which one is the most effective one for the chosen dataset. We will consider different classification algorithms and make the comparison between three of them, more precisely *Artificial Neural Networks*, *CatBoost* and *Logistic Regression*.

We will divide the work as the following: as a group we will perform a brief analysis of the dataset and make some cleaning of it if needed. Afterwards, each one of us will implement one of the previously mentioned algorithms and then we will compare and interpret the results

About the dataset

The dataset found in [Kaggle](#) consists of about 10 years of daily weather observations from numerous locations across Australia.

The problem that is required to be solved from this dataset represents a classification problem, in this case a **binary classification** problem. The objective is to predict whether it will rain tomorrow or not with high accuracy. The dataframe contains 145460 observations (rows) and 23 attributes. The observations are weather conditions of days of a specific region including: date, location, minimum and maximum temperature, rain fall, humidity and so on.

The most important feature of the dataset is the last column “**RainTomorrow**”, which is the target variable for our ML models that we want to predict.

It has two values:

- Yes → It will rain tomorrow
- No → It will not rain tomorrow.

Exploratory Data Analysis

```
library(tidyverse)
library(dplyr)
library(skimr)
library(ggcorrplot)
library(gt)
library(ggplot2)

#use_python("C:\\Python312\\python.exe", required = T)
weatherAus <- read.csv("weatherAUS.csv", header = T)
```

Setting up python into Rstudio:

```
# SETTING UP PYTHON ON RSTUDIO
library(reticulate)

virtualenv_create("my-python", python_version = "3.12")
use_virtualenv("my-python", required = TRUE)

#virtualenv_install(envname = "my-python", "matplotlib", ignore_installed = FALSE, pip_opti
```

```
#virtualenv_install(envname = "my-python", "catboost",ignore_installed = FALSE, pip_option
#virtualenv_install(envname = "my-python", "numpy",ignore_installed = FALSE, pip_options =
#virtualenv_install(envname = "my-python", "pandas",ignore_installed = FALSE, pip_options
#virtualenv_install(envname = "my-python", "seaborn",ignore_installed = FALSE, pip_options
#virtualenv_install(envname = "my-python", "scikit-learn",ignore_installed = FALSE, pip_op
#virtualenv_install(envname = "my-python", "tensorflow",ignore_installed = FALSE, pip_opti
```

As we start, we first load the weather data and look at the first rows to identify the features:

```
head(weatherAus) %>% gt()
```

Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindG
2008-12-01	Albury	13.4	22.9	0.6	NA	NA	W	
2008-12-02	Albury	7.4	25.1	0.0	NA	NA	WNW	
2008-12-03	Albury	12.9	25.7	0.0	NA	NA	WSW	
2008-12-04	Albury	9.2	28.0	0.0	NA	NA	NE	
2008-12-05	Albury	17.5	32.3	1.0	NA	NA	W	
2008-12-06	Albury	14.6	29.7	0.2	NA	NA	WNW	

In the next step we check out the summary statistics of the dataset and identify the numerical and categorical attributes:

```
skim(weatherAus)
```

Table 2: Data summary

Name	weatherAus
Number of rows	145460
Number of columns	23

Column type frequency:

character 7
 numeric 16

Group variables None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Date	0	1.00	10	10	0	3436	0
Location	0	1.00	4	16	0	49	0
WindGustDir	10326	0.93	1	3	0	16	0
WindDir9am	10566	0.93	1	3	0	16	0
WindDir3pm	4228	0.97	1	3	0	16	0
RainToday	3261	0.98	2	3	0	2	0
RainTomorrow	3267	0.98	2	3	0	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
MinTemp	1485	0.99	12.19	6.40	-8.5	7.6	12.0	16.9	33.9	
MaxTemp	1261	0.99	23.22	7.12	-4.8	17.9	22.6	28.2	48.1	
Rainfall	3261	0.98	2.36	8.48	0.0	0.0	0.0	0.8	371.0	
Evaporation	62790	0.57	5.47	4.19	0.0	2.6	4.8	7.4	145.0	
Sunshine	69835	0.52	7.61	3.79	0.0	4.8	8.4	10.6	14.5	
WindGustSpeed	10263	0.93	40.04	13.61	6.0	31.0	39.0	48.0	135.0	
WindSpeed9am	1767	0.99	14.04	8.92	0.0	7.0	13.0	19.0	130.0	
WindSpeed3pm	3062	0.98	18.66	8.81	0.0	13.0	19.0	24.0	87.0	
Humidity9am	2654	0.98	68.88	19.03	0.0	57.0	70.0	83.0	100.0	
Humidity3pm	4507	0.97	51.54	20.80	0.0	37.0	52.0	66.0	100.0	
Pressure9am	15065	0.90	1017.65	7.11	980.5	1012.9	1017.6	1022.4	1041.0	
Pressure3pm	15028	0.90	1015.26	7.04	977.1	1010.4	1015.2	1020.0	1039.6	
Cloud9am	55888	0.62	4.45	2.89	0.0	1.0	5.0	7.0	9.0	
Cloud3pm	59358	0.59	4.51	2.72	0.0	2.0	5.0	7.0	9.0	
Temp9am	1767	0.99	16.99	6.49	-7.2	12.3	16.7	21.6	40.2	
Temp3pm	3609	0.98	21.68	6.94	-5.4	16.6	21.1	26.4	46.7	

As we can see from the figure above, there are 7 **categorical** attributes and 16 **numerical** attributes.

Before taking a deeper look on all other attributes, we first did a brief exploration of the target variable:

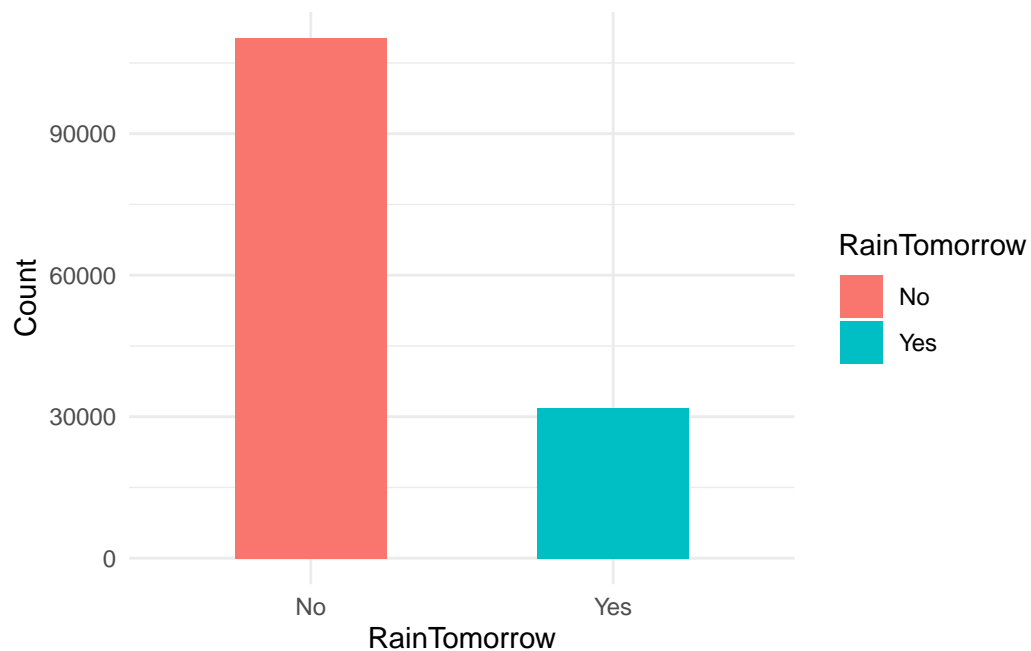
- **MISSING VALUES**

```
missingValues <- sum(is.na(weatherAus$RainTomorrow))  
missingValues
```

```
[1] 3267
```

- **FREQUENCY DISTRIBUTION OF VALUES**

```
weatherAus %>% select(RainTomorrow) %>%  
count(RainTomorrow) %>% drop_na() %>%  
ggplot(., aes(RainTomorrow, n, fill=RainTomorrow)) +  
geom_col(width = 0.5)+  
labs(x = "RainTomorrow", y = "Count")+  
theme_minimal()
```

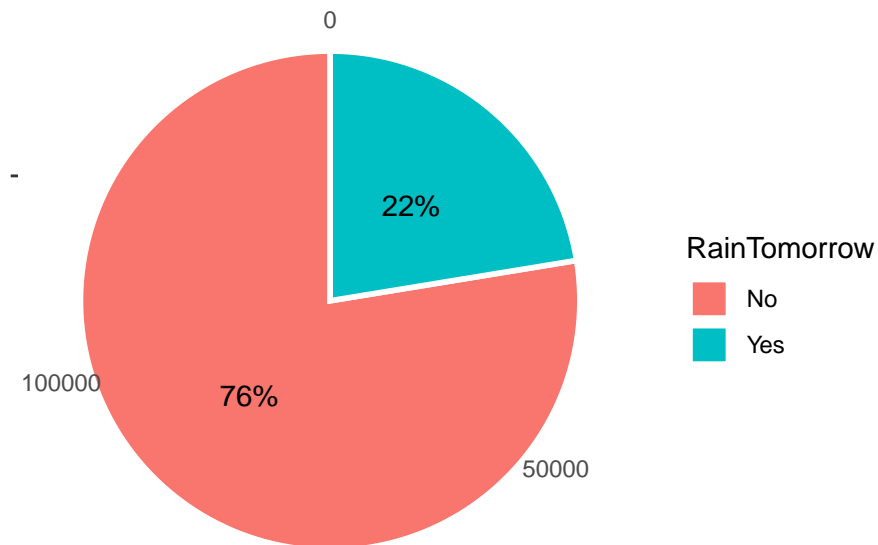


- **RATIO OF FREQUENCY DISTRIBUTION**

```

weatherAus %>% select(RainTomorrow) %>%
count(RainTomorrow) %>% drop_na() %>%
ggplot(., aes(x="", n, fill = RainTomorrow)) +
geom_bar(width = 1, size = 1, color = "white",stat = "identity") + coord_polar("y", star
geom_text(aes(label = paste0(round((n/145460)*100),"%")),
          position = position_stack(vjust = 0.5)) +
theme_classic() +
labs(x = NULL, y = NULL) +
theme(axis.line = element_blank())

```



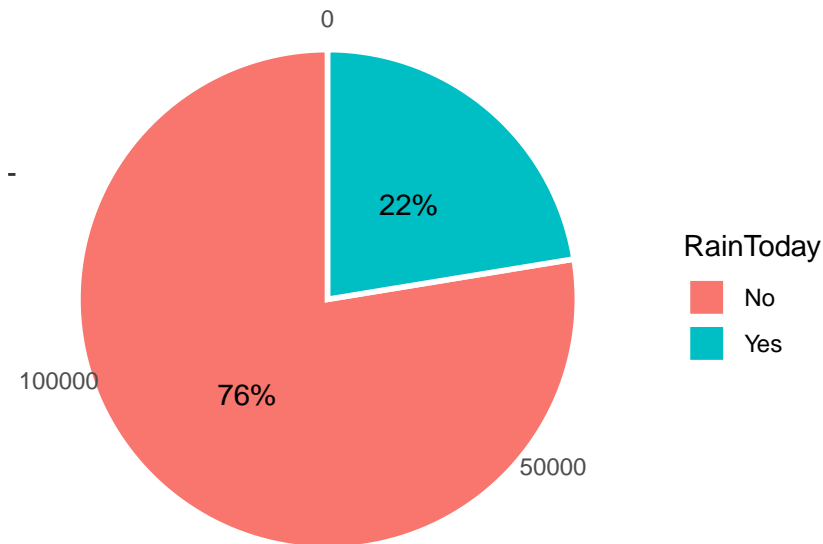
From the plots drawn above, we can clearly see that RainTomorrow has 2 categories of values: **Yes** and **No**. There are far more NEGATIVE values than POSITIVE. “No” and “Yes” appears 76% of time and 22% of time respectively after deleting all NA values from the attribute.

```

weatherAus %>% select(RainToday) %>%
count(RainToday) %>% drop_na() %>%
ggplot(., aes(x="", n, fill = RainToday)) +
geom_bar(width = 1, size = 1, color = "white",stat = "identity") + coord_polar("y", star
geom_text(aes(label = paste0(round((n/145460)*100),"%")),
          position = position_stack(vjust = 0.5)) +

```

```
theme_classic() +
labs(x = NULL, y = NULL) +
theme(axis.line = element_blank())
```



The variable **RainToday** has a very similar value distribution as the target variable.
!!!!!!

Categorical values

All together there are 6 categorical features + a Date column. In order to make the information about the date more useful, we decided to extract the year, the month and the day from the date into three separate columns.

This is done here below:

```
weatherAusNew <- weatherAus %>% mutate(
  Year = year(Date),
```

```

    Month = month(Date),
    Day = day(Date)
  ) %>% select(-Date)
  as_tibble(weatherAusNew)

```

```
# A tibble: 145,460 x 25
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	Albury	13.4	22.9	0.6	NA	NA	W
2	Albury	7.4	25.1	0	NA	NA	WNW
3	Albury	12.9	25.7	0	NA	NA	WSW
4	Albury	9.2	28	0	NA	NA	NE
5	Albury	17.5	32.3	1	NA	NA	W
6	Albury	14.6	29.7	0.2	NA	NA	WNW
7	Albury	14.3	25	0	NA	NA	W
8	Albury	7.7	26.7	0	NA	NA	W
9	Albury	9.7	31.9	0	NA	NA	NNW
10	Albury	13.1	30.1	1.4	NA	NA	W

```
# i 145,450 more rows
```

```

# i 18 more variables: WindGustSpeed <int>, WindDir9am <chr>, WindDir3pm <chr>,
#   WindSpeed9am <int>, WindSpeed3pm <int>, Humidity9am <int>,
#   Humidity3pm <int>, Pressure9am <dbl>, Pressure3pm <dbl>, Cloud9am <int>,
#   Cloud3pm <int>, Temp9am <dbl>, Temp3pm <dbl>, RainToday <chr>,
#   RainTomorrow <chr>, Year <dbl>, Month <dbl>, Day <int>

```

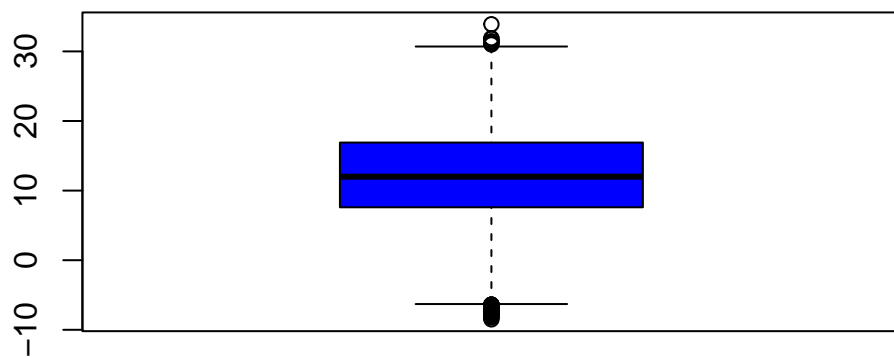
Numerical values

There are 16 numerical attributes in the raw dataset, after adding three columns for year, month and day, there are in total 19 numerical attributes. The main goal when analyzing numerical data is to find the outliers. Outliers are data information that differ significantly from other observations.

The most efficient way to detect outliers is to draw box plots:

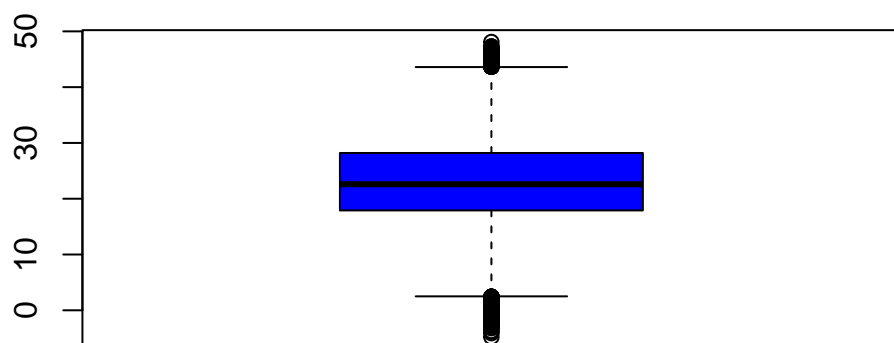
MinTemp

```
boxplot(weatherAusNew$MinTemp, col = "blue", border = "black")
```

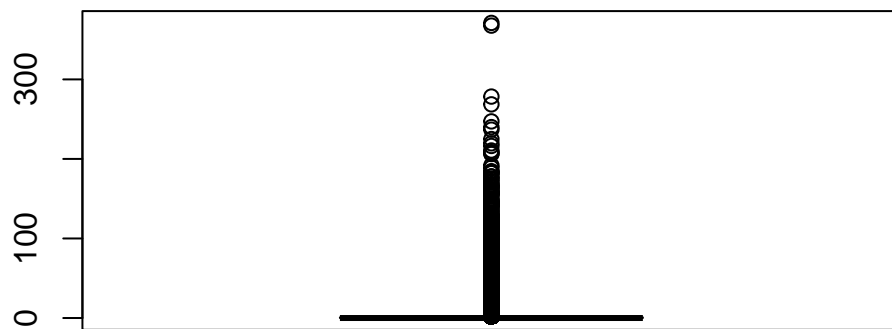
MaxTemp

```
boxplot(weatherAusNew$MaxTemp, col = "blue", border = "black")
```



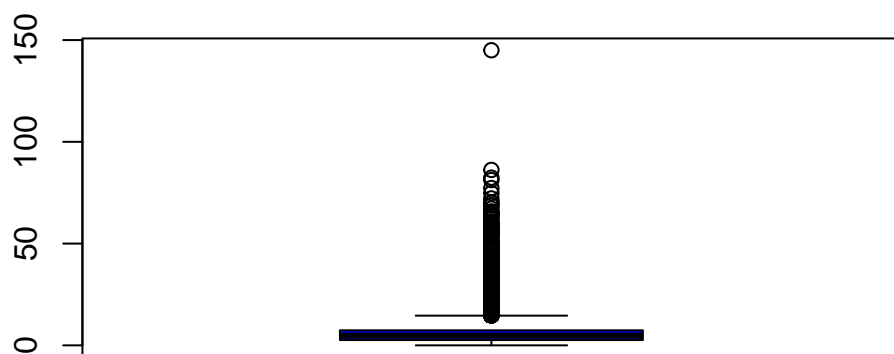
Rainfall

```
boxplot(weatherAusNew$Rainfall, col = "blue", border = "black")
```



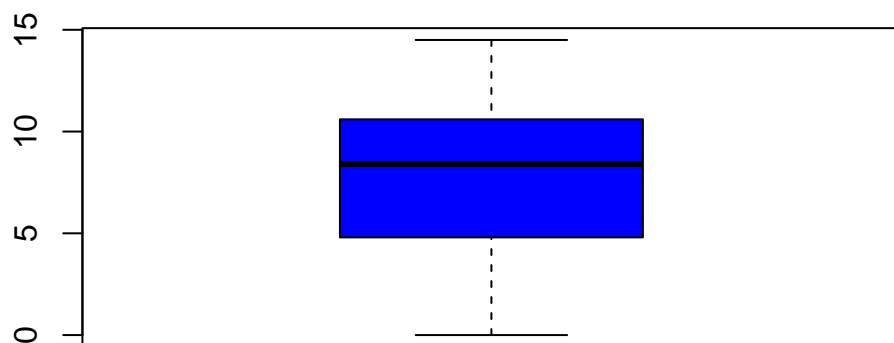
Evaporation

```
boxplot(weatherAusNew$Evaporation, col = "blue", border = "black")
```



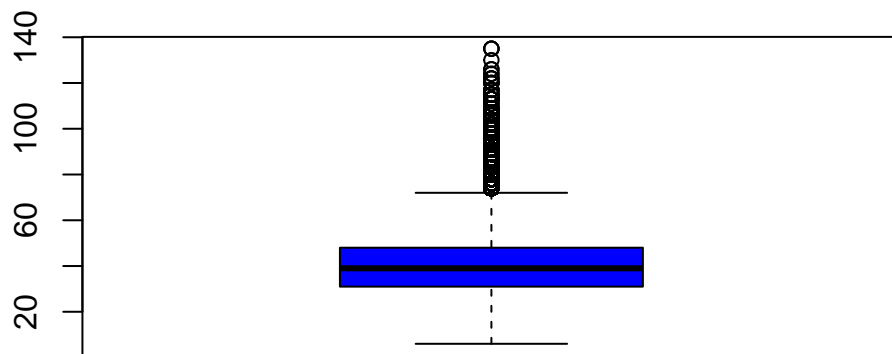
Sunshine

```
boxplot(weatherAusNew$Sunshine, col = "blue", border = "black")
```



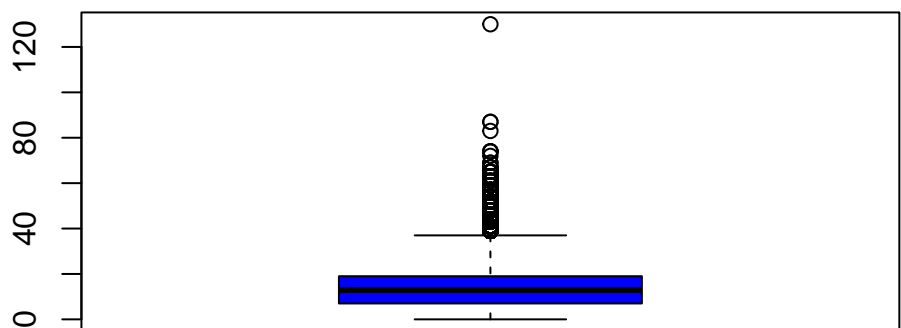
WindGustSpeed

```
boxplot(weatherAusNew$WindGustSpeed, col = "blue", border = "black")
```



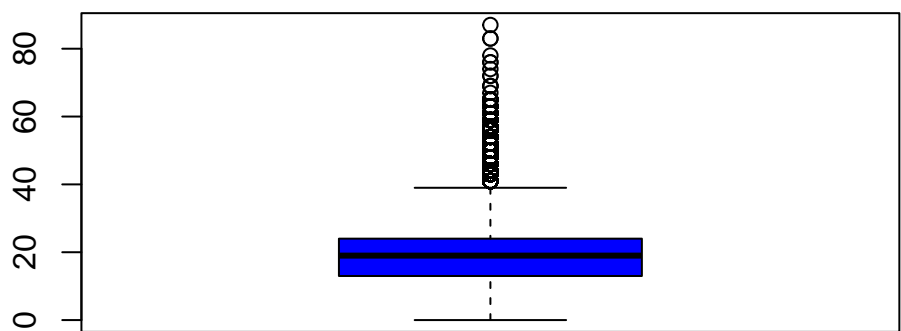
WindSpeed9am

```
boxplot(weatherAusNew$WindSpeed9am, col = "blue", border = "black")
```



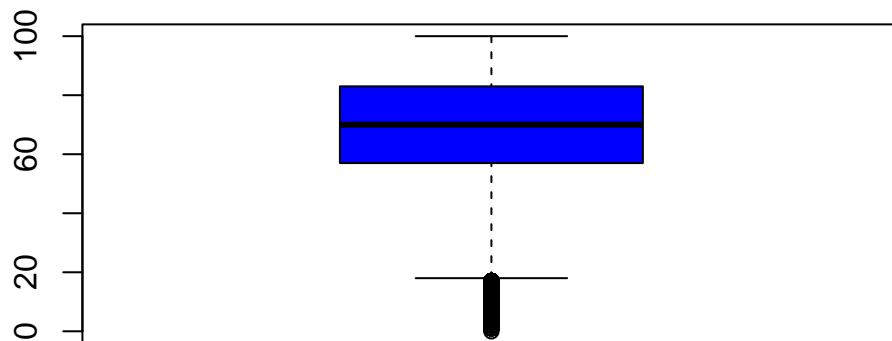
WindSpeed3pm

```
boxplot(weatherAusNew$WindSpeed3pm, col = "blue", border = "black")
```



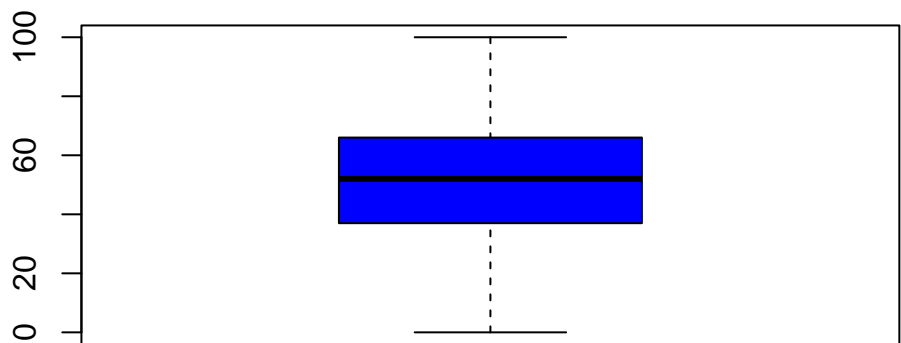
Humidity9am

```
boxplot(weatherAusNew$Humidity9am, col = "blue", border = "black")
```



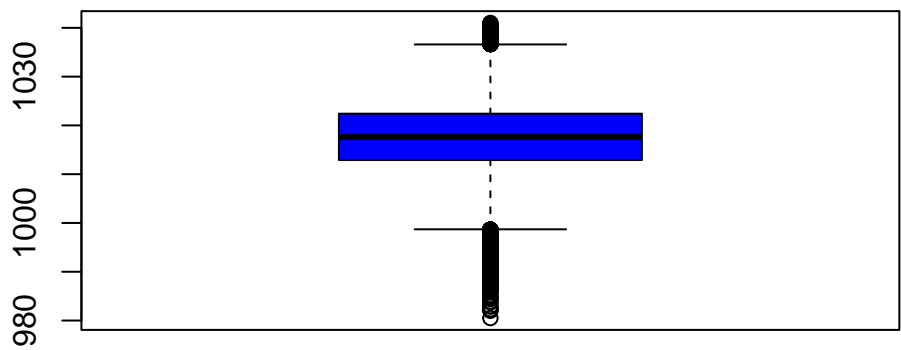
Humidity3pm

```
boxplot(weatherAusNew$Humidity3pm, col = "blue", border = "black")
```



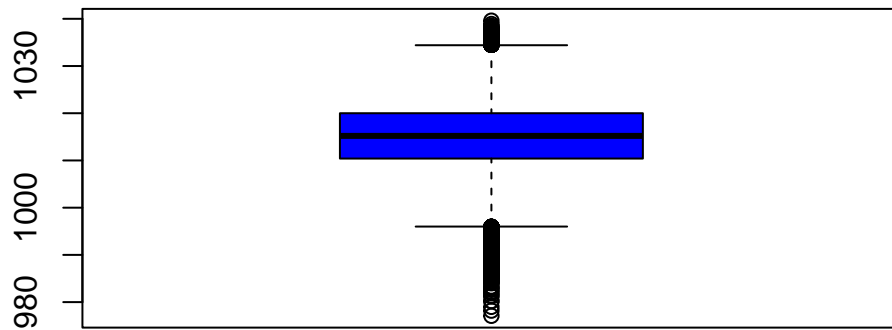
Pressure9am

```
boxplot(weatherAusNew$Pressure9am, col = "blue", border = "black")
```



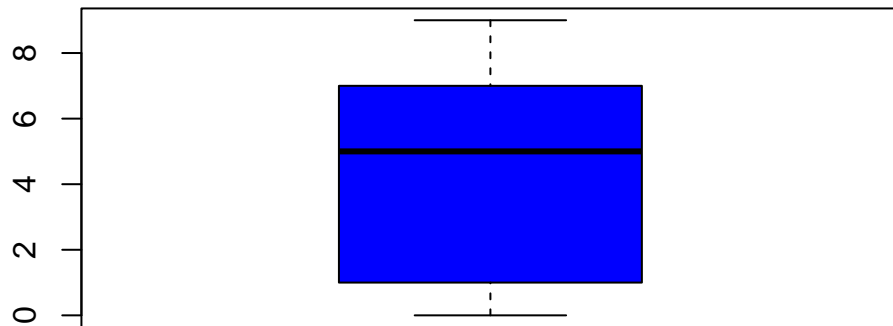
Pressure3pm

```
boxplot(weatherAusNew$Pressure3pm, col = "blue", border = "black")
```



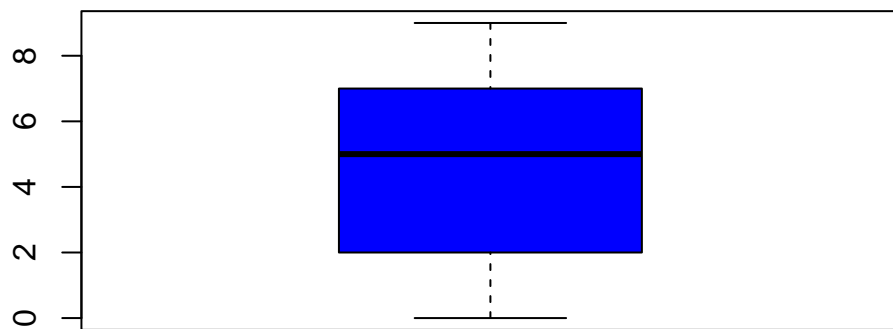
Cloud9am

```
boxplot(weatherAusNew$Cloud9am, col = "blue", border = "black")
```

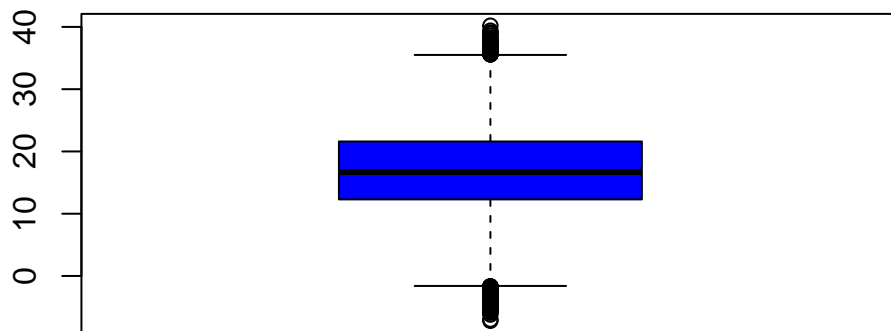
Cloud3pm

```
boxplot(weatherAusNew$Cloud3pm, col = "blue", border = "black")
```



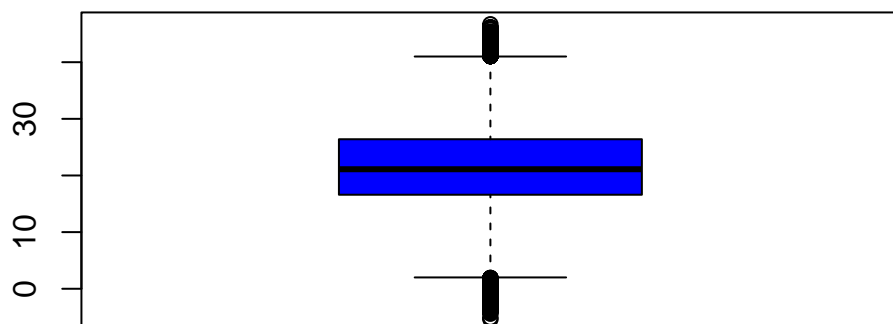
Temp9am

```
boxplot(weatherAusNew$Temp9am, col = "blue", border = "black")
```



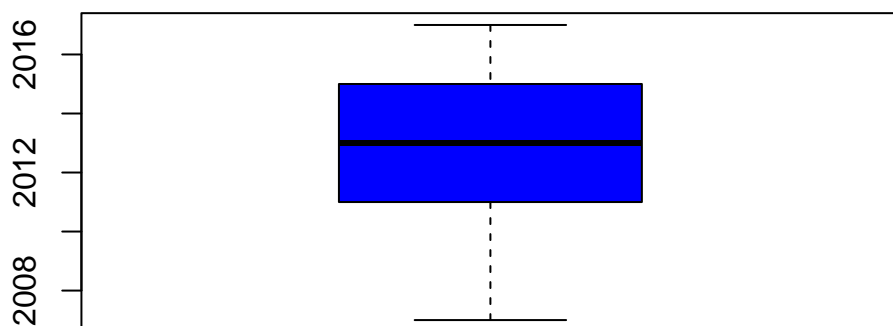
Temp3pm

```
boxplot(weatherAusNew$Temp3pm, col = "blue", border = "black")
```



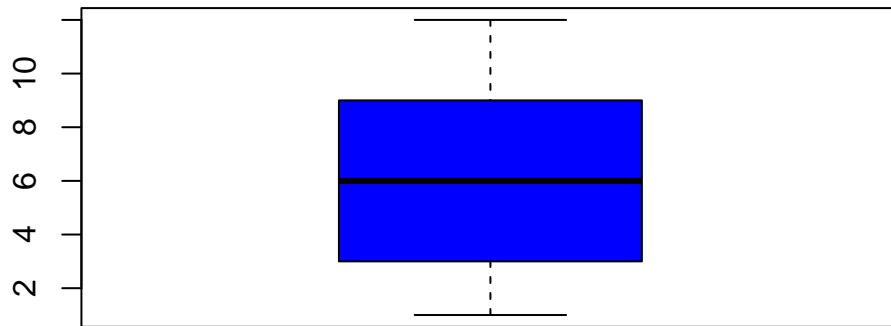
Year

```
boxplot(weatherAusNew$Year, col = "blue", border = "black")
```



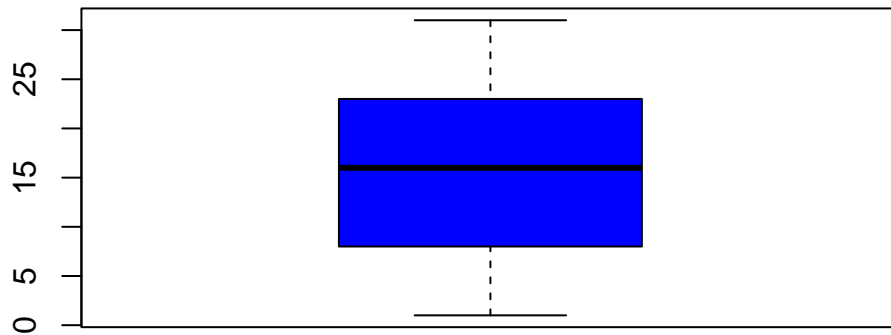
Month

```
boxplot(weatherAusNew$Month, col = "blue", border = "black")
```



Day

```
boxplot(weatherAusNew$Day, col = "blue", border = "black")
```



Multicollinearity

```
weatherAusNew %>% select(where(is.numeric)) %>% model.matrix(~0+.,
  data=.) %>%
  cor(use="pairwise.complete.obs") %>%
  ggcorrplot(show.diag = FALSE, type="full",
    lab=TRUE, legend.title = "Correlation" ,lab_size
    = 2,lab_col = "black" ,ggtheme =
    ggplot2::theme_gray,
    colors = c("white","green","darkgreen"),
    outline.color = "black")

write.csv(weatherAusNew, file = "weatherNewToPython.csv", row.names = FALSE)
colnames(weatherAusNew)
```

```
[1] "Location"      "MinTemp"      "MaxTemp"      "Rainfall"
[5] "Evaporation"  "Sunshine"     "WindGustDir"  "WindGustSpeed"
[9] "WindDir9am"   "WindDir3pm"   "WindSpeed9am" "WindSpeed3pm"
[13] "Humidity9am"  "Humidity3pm"  "Pressure9am"  "Pressure3pm"
[17] "Cloud9am"     "Cloud3pm"     "Temp9am"      "Temp3pm"
[21] "RainToday"    "RainTomorrow" "Year"          "Month"
```

[25] "Day"

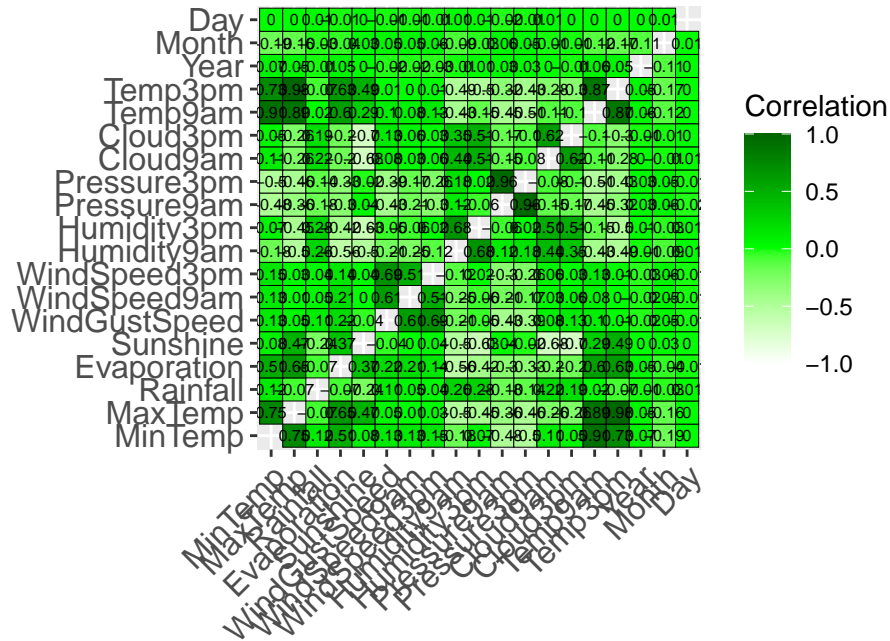


Figure 1: Correlation Heatmap

Outliers & Missing Values

Outliers

After drawing a boxplot for each numerical attribute in the dataset, we compared the mean of each column with the min/max value and we have noticed that that attributes **Rainfall**, **Evaporation**, **WindSpeed9am** and **WindSpeed3pm** might have a large number of outliers as there's a considerable difference between average value and max value. This also can be seen from their plots, as there is a huge amount of points (values) that differ from the average. Outliers can be identified by using some visualization tools as we have seen above, or also with some statistical methods. Once detected, outliers can be addressed by removing them, transforming the data, or using robust models less sensitive to outliers.

When implementing the models, we will split the dataset into training and testing sets. The training set will be used to train machine learning models, allowing the algorithms to learn from the data. The testing set, on the other hand, will be used to evaluate the models' performance on unseen data, ensuring that the models generalize well and provide accurate predictions in real-world scenarios.

Missing Values

Addressing missing values is crucial during data preprocessing. Missing values can result from data entry errors, collection issues, and they can degrade performance. Because of this we are going to impute the missing values at each implementation of the three models. **Missing values in categorical columns** will be filled up using the Python function `mode()` that fills in the cells with the most common/occurring element from all other instances. **Missing values in numerical columns** will be filled up with the median value from all the values of the other instances.

Modeling

After performing the Exploratory Data Analysis, we will proceed to the modeling part. In this phase we will implement three models: **Artificial Neural Network**, **CatBoost**, and **Logistic Regression**.

Till now the analysis was made using code chunks performed in R language. For this implementation part the models will be implemented in Python.

Artificial Neural Network

- The ANN model derives from Biological neural networks that have the structure of the human brain. It contains neurons(nodes) interconnected to one another in various layers of the network. It consists of three layers: Input layer, Hidden layers (can be several of them) and Output layer. The input layer accepts inputs in several formats, the hidden layer is in-between the inputs and outputs and performs calculations to find hidden features and patterns. The output layer outputs the results of calculations.
- *ADVANTAGES*
 - **Parallel processing**
 - Information can produce **output** even with **inadequate data**
 - **Success** proportional to **chosen instances**
- *DISADVANTAGES*
 - Depends on **hardware**

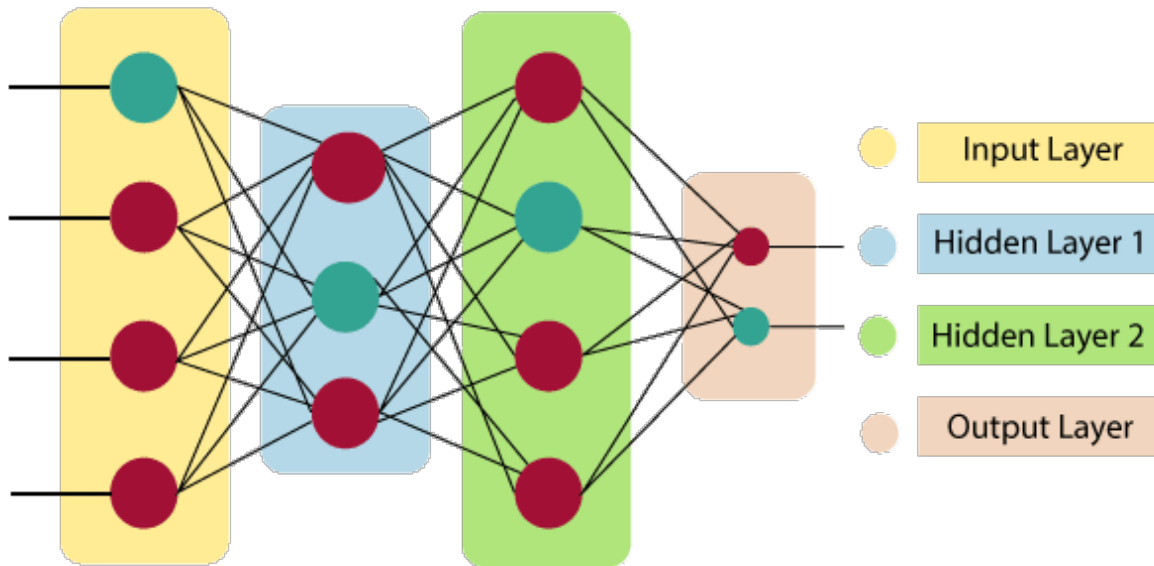


Figure 2: ANN layers

- ANNs work with **numerical data**
- Duration of network is **unknown**

- **FUNCTIONING**

Each input is multiplied by its corresponding weights (strength of interconnections between neurons). All weighted inputs are summarized inside the computing unit. Each neuron has its **bias**, which is added to the weighted sum to make it non-zero, so the total sum of weighted inputs can be from 0 to plus infinity. The maximum value is **benchmarked** to keep the response in the limits. This is performed in *TRANSFER FUNCTIONS*.

ACTIVATION FUNCTIONS choose whether a node should fire or not. Only those who are *fired* make it to the output layer. Activation functions are distinctive depending on the task that is performed.

- **FEED-BACK**

- Feed-back networks feed information back to itself

- **FEED-FORWARD**

- Assessment of outputs by reviewing its inputs
- Input \rightarrow Neuron layer \rightarrow Output

```

# ANN IMPLEMENTATIONN
import pandas as pd
import numpy as np
import seaborn as sns
from catboost import CatBoostClassifier, Pool
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from keras.layers import Dense, BatchNormalization, Dropout
from keras.models import Sequential
from keras import callbacks
from keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder

df = pd.read_csv('weatherNewToPython.csv')

# Categorical columns
s = (df.dtypes == "object")
cat_cols = list(s[s].index)

print("Categorical variables")

print(cat_cols)

for cols in cat_cols:

    df[cols] = df[cols].fillna(df[cols].mode()[0])

# Numerical columns
t = (df.dtypes == "float64")
num_cols = list(t[t].index)

print("Numeric variables:")

```

```

print(num_cols)

for cols in num_cols:

    df[cols] = df[cols].fillna(df[cols].median())
df.info()

df.head(10)

# Categorical columns of type "object" into "float64"
label_encoder = LabelEncoder()
for cols in cat_cols:
    df[cols] = label_encoder.fit_transform(df[cols])

df.info()

# dropping target and extra columns
features = df.drop(['RainTomorrow', 'Year', 'Month', 'Day'], axis=1)
target = df['RainTomorrow']

X = features
y = target

# Splitting test and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =

X.shape

#Early stopping
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001, # minimum amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True,
)

# Initialising the Neural Network
model = Sequential()

```

```

# Adding layers to the network
model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu', input_dim=1000))
model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
opt = Adam(learning_rate=0.001)
model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])

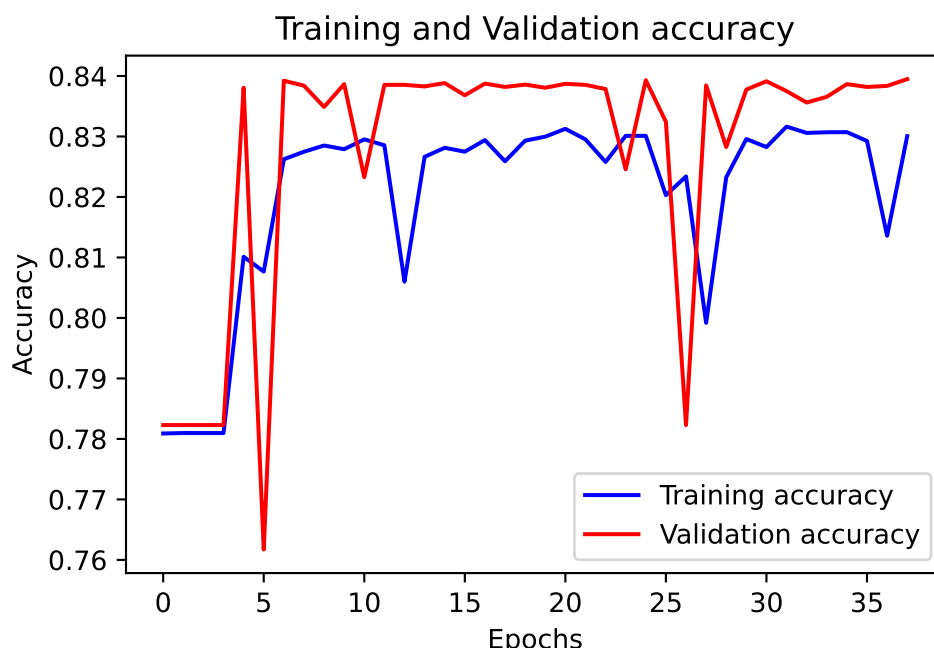
# Train the ANN
history = model.fit(X_train, y_train, batch_size = 32, epochs = 150, callbacks=[early_stop])

# Plotting the TRAINING & VALIDATION accuracy
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['accuracy']], "blue", label='Training accuracy')
plt.plot(history_df.loc[:, ['val_accuracy']], "red", label='Validation accuracy')

plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```



```
# Predicting the test set results
y_pred = model.predict(X_test)
```

```
y_pred = (y_pred > 0.5)
accuracy = accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.86	0.95	0.90	22672
1	0.72	0.43	0.54	6420
accuracy			0.84	29092
macro avg	0.79	0.69	0.72	29092
weighted avg	0.83	0.84	0.82	29092

CatBoost

- Catboost is a machine learning algorithm that excels in classification and regression tasks. As a **gradient boosting** algorithm, it builds an ensemble of decision trees, where each tree corrects the errors of the previous ones. This iterative process enhances the model's accuracy and robustness.

- **FUNCTIONING**

Catboost follows the principles of gradient boosting but introduces several unique innovations:

- *Initialization:* The algorithm begins with a simple initial model, such as predicting the **mean value** for regression or **uniform probability** for classification.
- *Sequential Tree Building:* Decision trees are constructed one by one. Each tree is designed to minimize a specified loss function by addressing the errors from previous trees.
- *Model Update:* Newly built tree is added to the model, refining it.
- *Iteration:* The process repeats until the stopping criterion is met.

- **Key Concepts:**

- Gradient Boosting: Ensemble learning technique that combines weak prediction models, typically prediction trees, to form a strong predictive model. It iteratively adds new models to the ensemble.
- Handling Categorical Features: It handles categorical features directly, without requiring extensive preprocessing. Very effective for real-world datasets that contain qualitative data, improving the performance.
- Learning Rate: The learning rate in CatBoost controls the step size at which the model learns during boosting. CatBoost automatically selects an optimal learning rate based on the dataset's features, striking a **balance** between **learning speed** and **model accuracy**. This helps achieving better performance with minimal tuning.

```
# Load data
data = pd.read_csv("weatherAUS.csv")

# Define columns with missing values
numeric_features = data.select_dtypes(include=['float64', 'int64']).columns
```

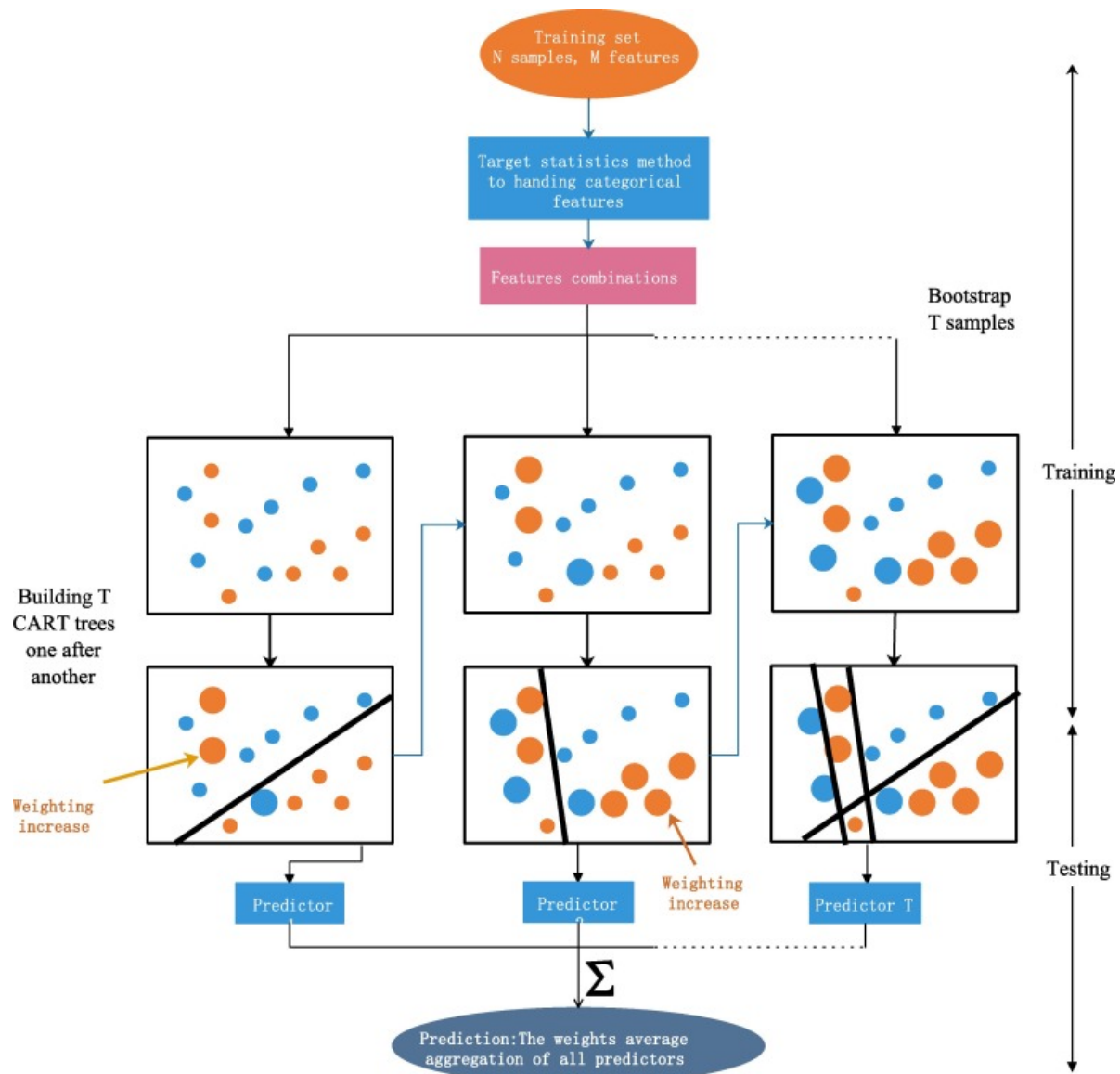


Figure 3: Sequential Tree Building approach

```

categorical_features = data.select_dtypes(include=['object']).columns

# Fill missing values
for col in numeric_features:
    data[col] = data[col].fillna(data[col].median())

for col in categorical_features:
    data[col] = data[col].fillna(data[col].mode()[0])

# Prepare features and labels
X = data.drop(columns=['RainTomorrow'])
y = data['RainTomorrow']

# Ensure that 'RainTomorrow' is not included in categorical_features
categorical_features = [col for col in categorical_features if col != 'RainTomorrow']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create CatBoost Pool
train_pool = Pool(data=X_train, label=y_train, cat_features=categorical_features)
test_pool = Pool(data=X_test, label=y_test, cat_features=categorical_features)

# Initialize CatBoost model
model = CatBoostClassifier(iterations=100, depth=6, learning_rate=0.1, loss_function='LogLoss')

# Train the model
model.fit(train_pool)

# Get evaluation results
evals_result = model.get_evals_result()

# Print the loss at each iteration
losses = evals_result['learn']['Logloss']
for i, loss in enumerate(losses):
    print(f'Iteration {i}: Loss = {loss}')

# Make predictions
preds = model.predict(test_pool)

# Evaluate the model

```

```
accuracy = (preds == y_test).mean()
print(f'Accuracy: {accuracy}')
```

Accuracy: 0.8512649525642788

```
print(classification_report(y_test, preds))
```

	precision	recall	f1-score	support
No	0.87	0.95	0.91	22672
Yes	0.74	0.50	0.60	6420
accuracy			0.85	29092
macro avg	0.81	0.73	0.75	29092
weighted avg	0.84	0.85	0.84	29092

Logistic Regression

- **Logistic Regression** is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not. This model is used for binary classification where we use the sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.
- **Key Concepts:**
 - Logistic Regression predicts the output of a categorical dependent variable. In our case we have the target variable “*RainTomorrow*” that has values “yes” and “no”.
 - It gives the probabilistic values which is between 0 and 1.
 - Instead of fitting a regression line, we fit an **S shaped** logistic function, which predicts two maximum values.
- **Sigmoid function** is a mathematical function used to map the predicted values to probabilities. It maps any real value into another value within the range of 0 and 1. This range is called the **Threshold value**.

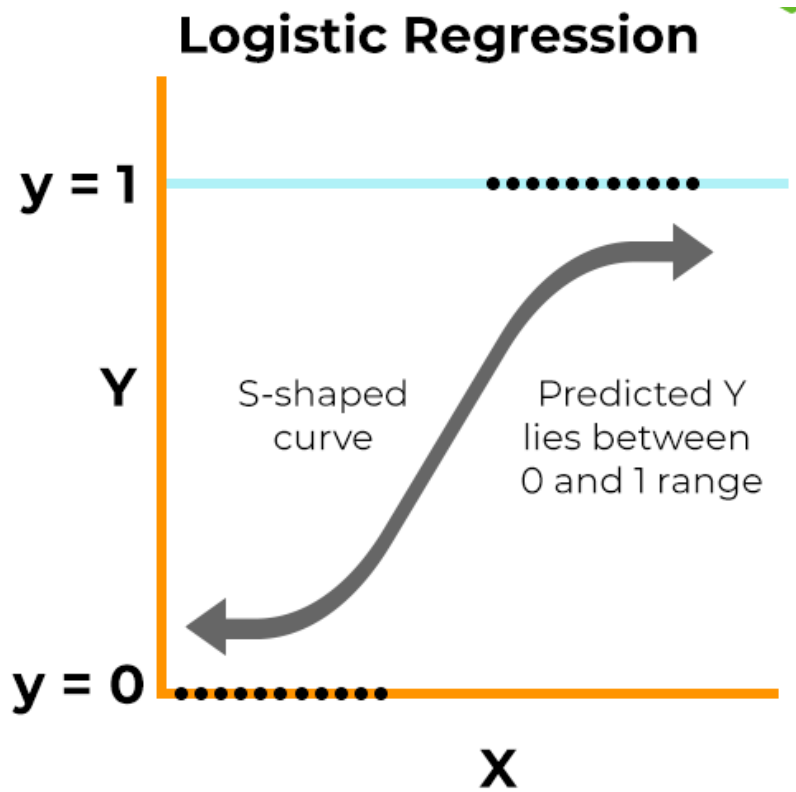


Figure 4: Sigmoid Function

- **Types of Logistic regression**

- Binomial: there can be only two possible types of dependent variables, such as 0 and 1, Pass or Fail, etc.
- Multinomial: there can be 3 or more possible unordered types of the dependent variable, such as “cat”, “dogs”, or “sheep”.
- Ordinal: there can be 3 or more possible ordered types of dependent variables, such as “low”, “medium”, “high”.

```
data = pd.read_csv("weatherAUS.csv")

# Define numerical and categorical columns
numerical_cols = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSp',
                  'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3p',
                  'Temp9am', 'Temp3pm']

categorical_cols = ['WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']

# Preprocessing for numerical data: Impute missing values with median and scale
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Preprocessing for categorical data: Impute missing values with mode and one-hot encode
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Prepare the data
```

```

X = data.drop(columns=['Date', 'Location', 'RainTomorrow'])
y = data['RainTomorrow']

# Apply label encoding to the target variable
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the logistic regression model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])

model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, zero_division=0)

print("Accuracy:", accuracy)

```

Accuracy: 0.8231472569778633

```
print("Confusion Matrix:\n", conf_matrix)
```

Confusion Matrix:

```

[[20779 1233    0]
 [ 3252 3168    0]
 [  584   76    0]]

```

```
print("Classification Report:\n", class_report)
```

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.94	0.89	22012
1	0.71	0.49	0.58	6420
2	0.00	0.00	0.00	660
accuracy			0.82	29092
macro avg	0.52	0.48	0.49	29092
weighted avg	0.79	0.82	0.80	29092

Conclusion

In our machine learning project we aimed at predicting next-day rain, implementing three distinct models: Artificial Neural Networks (ANN), CatBoost, and Logistic Regression. Each model was assessed based on its accuracy in predicting the binary target variable *RainTomorrow*. The results of our evaluations revealed that CatBoost achieved the highest accuracy at 0.85, that can be attributed to its advanced gradient boosting algorithm, which is particularly effective in handling categorical features and capturing complex interactions within the data. CatBoost's ability to automatically handle categorical variables contributed to its slightly higher accuracy.

At second place we got ANN with an accuracy of 0.84. The strength of ANN lies in its ability to model non-linear relationships through multiple layers of interconnected neurons, but it typically requires more tuning and computational resources, which could explain why it performed slightly less than CatBoost.

Logistic Regression, while the simplest of the three models, achieved an accuracy of 0.82. This model may have performed worse than the other two models, because of its linear nature, which may not fully captured the complexities of the data as effectively. Logistic Regression remains a valuable model due to its simplicity and lower computational cost.