# Australia's Rainfall Prediction Using Artificial Neural Networks

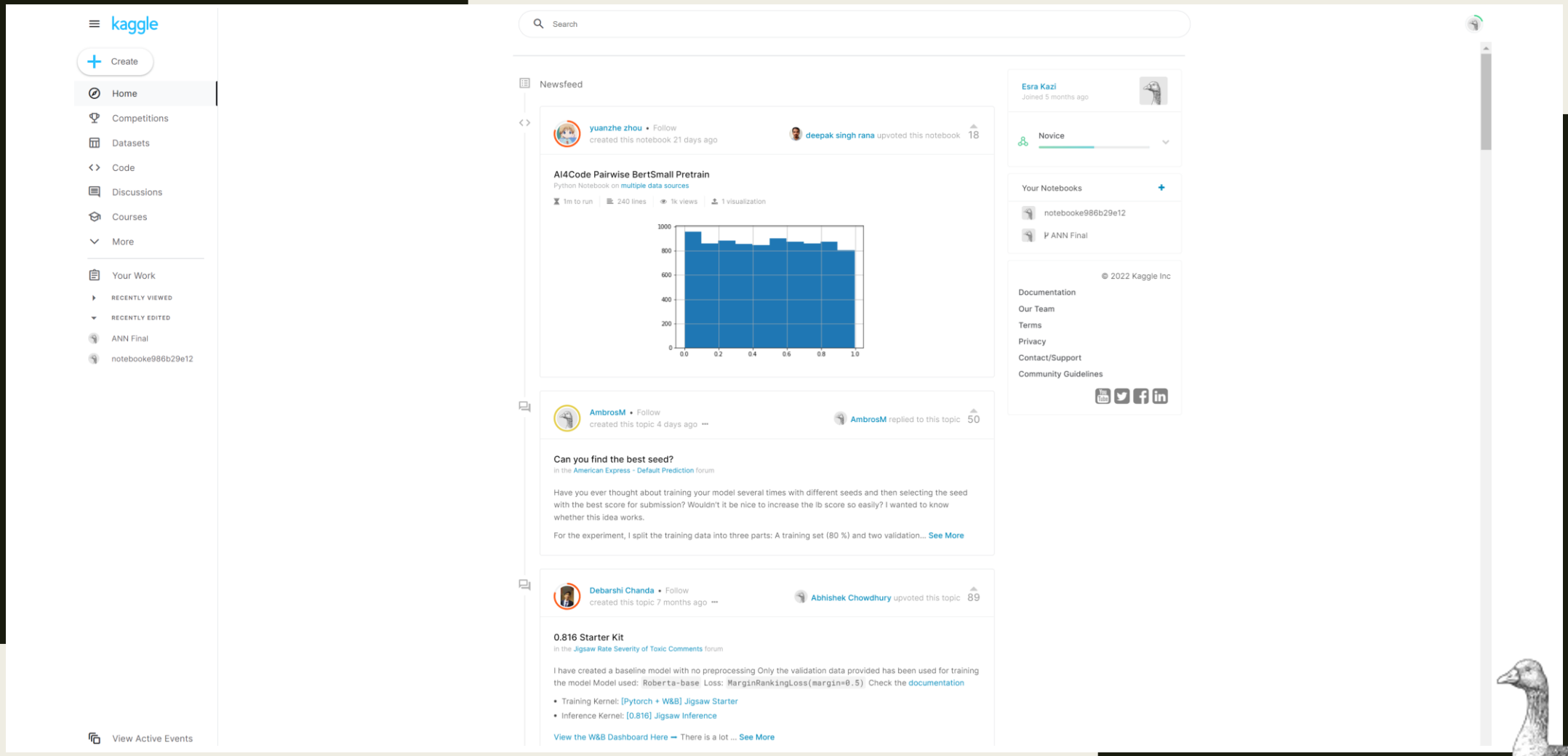
PRESENTED BY

Esra Kazi


# What is Kaggle?

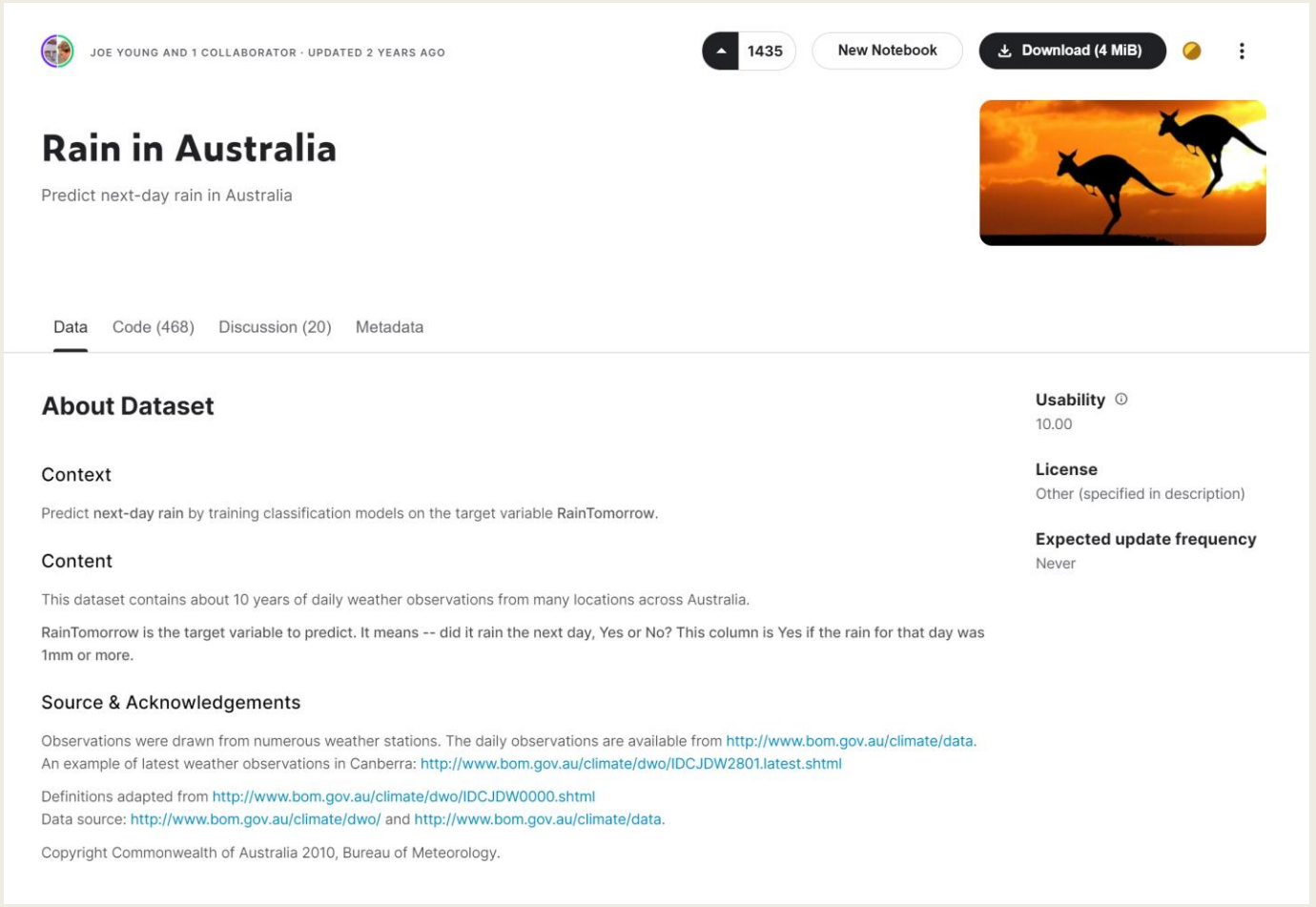
- Kaggle is an online community platform for data scientists and machine learning enthusiasts.
- Kaggle allows users to collaborate with other users, find and publish datasets, use GPU integrated notebooks, and compete with other data scientists to solve data science challenges.

# Home Page of Kaggle

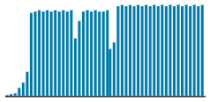


# Dataset

- Dataset used to predict rainfall is weatherAus.csv is dataset taken from Kaggle.
- This dataset contains about 10 years of daily weather observations from many locations across Australia.



# Data Categories

| Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGustSpeed |
|---|---|---|---|---|---|---|---|---|
| The date of observation | The common name of the location of the weather station | The minimum temperature in degrees celsius | The maximum temperature in degrees celsius | The amount of rainfall recorded for the day in mm | The so-called Class A pan evaporation (mm) in the 24 hours to 9am | The number of hours of bright sunshine in the day. | The direction of the strongest wind gust in the 24 hours to midnight | The speed (km/h) of the strongest wind gust in the 24 hours to midnight |
|  1Nov07 25Jun17 | Canberra 2% | NA 1% | 506 unique values | 0 63% | NA 43% | NA 48% | NA 7% | NA 7% |
| | Sydney 2% | 11 1% | | 0.2 6% | 4 2% | 0 2% | W 7% | 35 6% |
| | Other (138680) 95% | Other (143076) 98% | | Other (45619) 31% | Other (79331) 55% | Other (73266) 50% | Other (125219) 86% | Other (125982) 87% |

| WindDir9am | WindDir3pm | WindSpeed9am | WindSpeed3pm | Humidity9am | Humidity3pm | Pressure9am | Pressure3pm |
|---|---|---|---|---|---|---|---|
| Direction of the wind at 9am | Direction of the wind at 3pm | Wind speed (km/hr) averaged over 10 minutes prior to 9am | Wind speed (km/hr) averaged over 10 minutes prior to 3pm | Humidity (percent) at 9am | Humidity (percent) at 3pm | Atmospheric pressure (hpa) reduced to mean sea level at 9am | Atmospheric pressure (hpa) reduced to mean sea level at 3pm |
| N 8% | SE 7% | 9 9% | 13 9% | 99 2% | NA 3% | NA 10% | NA 10% |
| NA 7% | W 7% | 13 9% | 17 9% | 70 2% | 52 2% | 1016.4 1% | 1015.3 1% |
| Other (123136) 85% | Other (124512) 86% | Other (118679) 82% | Other (120341) 83% | Other (139043) 96% | Other (138202) 95% | Other (129579) 89% | Other (129646) 89% |

| Cloud9am | Cloud3pm | Temp9am | Temp3pm | RainToday | RainTomorrow |
|---|---|---|---|---|---|
| Fraction of sky obscured by cloud at 9am. This is measured in "oktas", which are a unit of eigths. It records how many | Fraction of sky obscured by cloud (in "oktas": eighths) at 3pm. See Cload9am for a description of the values | Temperature (degrees C) at 9am | Temperature (degrees C) at 3pm | Boolean: 1 if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0 | The amount of next day rain in mm. Used to create response variable RainTomorrow. A kind of measure of the "risk". |
| NA 38% | NA 41% | NA 1% | NA 2% | No 76% | No 76% |
| 7 14% | 7 13% | 17 1% | 20 1% | Yes 22% | Yes 22% |
| Other (69600) 48% | Other (67873) 47% | Other (142781) 98% | Other (140969) 97% | Other (3261) 2% | Other (3267) 2% |

# DEALING WITH MISSING DATA

```
for i in num_cols:
    print(i, data[i].isnull().sum())
```

```
MinTemp 1485
MaxTemp 1261
Rainfall 3261
Evaporation 62790
Sunshine 69835
WindGustSpeed 10263
WindSpeed9am 1767
WindSpeed3pm 3062
Humidity9am 2654
Humidity3pm 4507
Pressure9am 15065
Pressure3pm 15028
Cloud9am 55888
Cloud3pm 59358
Temp9am 1767
Temp3pm 3609
```

```
for i in num_cols:
    data[i].fillna(data[i].median(), inplace=T

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  object
 1   Location       145460 non-null  object
 2   MinTemp        145460 non-null  float64
 3   MaxTemp        145460 non-null  float64
 4   Rainfall       145460 non-null  float64
 5   Evaporation    145460 non-null  float64
 6   Sunshine       145460 non-null  float64
 7   WindGustDir    135134 non-null  object
 8   WindGustSpeed  145460 non-null  float64
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   145460 non-null  float64
 12  WindSpeed3pm   145460 non-null  float64
 13  Humidity9am    145460 non-null  float64
 14  Humidity3pm    145460 non-null  float64
 15  Pressure9am    145460 non-null  float64
 16  Pressure3pm    145460 non-null  float64
 17  Cloud9am       145460 non-null  float64
 18  Cloud3pm       145460 non-null  float64
 19  Temp9am        145460 non-null  float64
 20  Temp3pm        145460 non-null  float64
```

```
data[i].isnull().sum()
```

```
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Temp9am          0
Temp3pm          0
```

# ENCODING STRING DATA

# Why Encode String Data?

- When creating a machine learning model, mathematical functions like ReLu, softmax, sigmoid functions are used on data.
- Every single data value should be encoded as series of numbers to ‘compute’ the string input.

| Location | MinTemp | MaxTemp | Rainfall | WindGustDir | WindGustSpeed | WindDir9am | WindDir3pm |
|---|---|---|---|---|---|---|---|
| Albury | 13.4 | 22.9 | 0.6 | W | 44.0 | W | WNW |
| Albury | 7.4 | 25.1 | 0.0 | WNW | 44.0 | NNW | WSW |
| Albury | 12.9 | 25.7 | 0.0 | WSW | 46.0 | W | WSW |
| Albury | 9.2 | 28.0 | 0.0 | NE | 24.0 | SE | E |
| Albury | 17.5 | 32.3 | 1.0 | W | 41.0 | ENE | NW |

| Location | MinTemp | MaxTemp | Rainfall | WindGustDir | WindGustSpeed | WindDir9am | WindDir3pm |
|---|---|---|---|---|---|---|---|
| -1.531666 | 0.191328 | -0.041360 | -0.203581 | 1.045228 | 0.327736 | 1.328766 | 1.366458 |
| -1.531666 | -0.751052 | 0.268745 | -0.275097 | 1.258262 | 0.327736 | -0.221338 | 1.586813 |
| -1.531666 | 0.112796 | 0.353318 | -0.275097 | 1.471296 | 0.479465 | 1.328766 | 1.586813 |
| -1.531666 | -0.468338 | 0.677518 | -0.275097 | -0.872075 | -1.189550 | 0.442992 | -1.718521 |
| -1.531666 | 0.835287 | 1.283631 | -0.155903 | 1.045228 | 0.100143 | -1.328556 | -0.176032 |

```
sc = StandardScaler()
x = sc.fit_transform(x)
```

```
print(x)
```

```
[[-1.53166617  0.19132753 -0.04135977 ... -0.01407077  0.02310362
  -0.52979545]
 [-1.53166617 -0.75105231  0.26874452 ...  0.03244663  0.387799
  -0.52979545]
 [-1.53166617  0.11279588  0.35331842 ...  0.62166712  0.22733303
  -0.52979545]
 ...
 [ 1.20928479 -1.06517892  0.52246622 ... -0.69632607  0.65037966
  -0.52979545]
 [ 1.20928479 -0.68822699  0.53656187 ... -0.29317521  0.63579185
  -0.52979545]
 [ 1.20928479  0.42692249 -0.45013361 ... -0.30868102 -0.10818671
  -0.52979545]]
```

- Encoded value for *data['Location']== 'Albury'* as *-1.53166617*

# SPLITTING DATA TO TRAIN AND TEST SETS

```
1 x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3, random_state = 42)
```

While training, I faced an issue caused by the difference in data types. So I used this piece of code to convert the data type as float

```
1 x_train = np.asarray(x_train).astype('float32')
2 y_train = np.asarray(y_train).astype('float32')
```

# CREATING, TRAINING AND TESTING THE MODEL

# Training Models

[1]:
```
#Early stopping
early_stopping = EarlyStopping(
    min_delta=0.001, # minimium amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True,
)

# Initialising the NN
model = Sequential()

# Layers

model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu', input_dim = 21))
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=0.00009)
model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])

# Train the ANN
history = model.fit(x_train, y_train, batch_size = 26, epochs = 5, callbacks=[early_stopping], validation_split=0.2)
```

```
Epoch 1/5
3133/3133 [==============================] - 3s 965us/step - loss: 0.4747 - accuracy: 0.7818 - val_loss: 0.4006 - val_accuracy: 0.7810
Epoch 2/5
3133/3133 [==============================] - 3s 914us/step - loss: 0.4173 - accuracy: 0.7819 - val_loss: 0.3949 - val_accuracy: 0.7810
Epoch 3/5
3133/3133 [==============================] - 3s 890us/step - loss: 0.4144 - accuracy: 0.7819 - val_loss: 0.3907 - val_accuracy: 0.7810
Epoch 4/5
3133/3133 [==============================] - 3s 890us/step - loss: 0.4109 - accuracy: 0.7819 - val_loss: 0.3885 - val_accuracy: 0.7810
Epoch 5/5
3133/3133 [==============================] - 3s 893us/step - loss: 0.4087 - accuracy: 0.8355 - val_loss: 0.3869 - val_accuracy: 0.8431
```

# Early Stopping

- EarlyStopping() is a programmer friendly function that calls back the model training process, if the model has repeatedly the same or with such small difference of accuracy improvement during the epoch.

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

- In our model;
  - *Change amount is declared as min_delta = 0.001*
  - *Number of epoch's EarlyStopping() function will pass is =20.*
    *On 20th epoch, if minimum change is still 0.001, training will stop.*

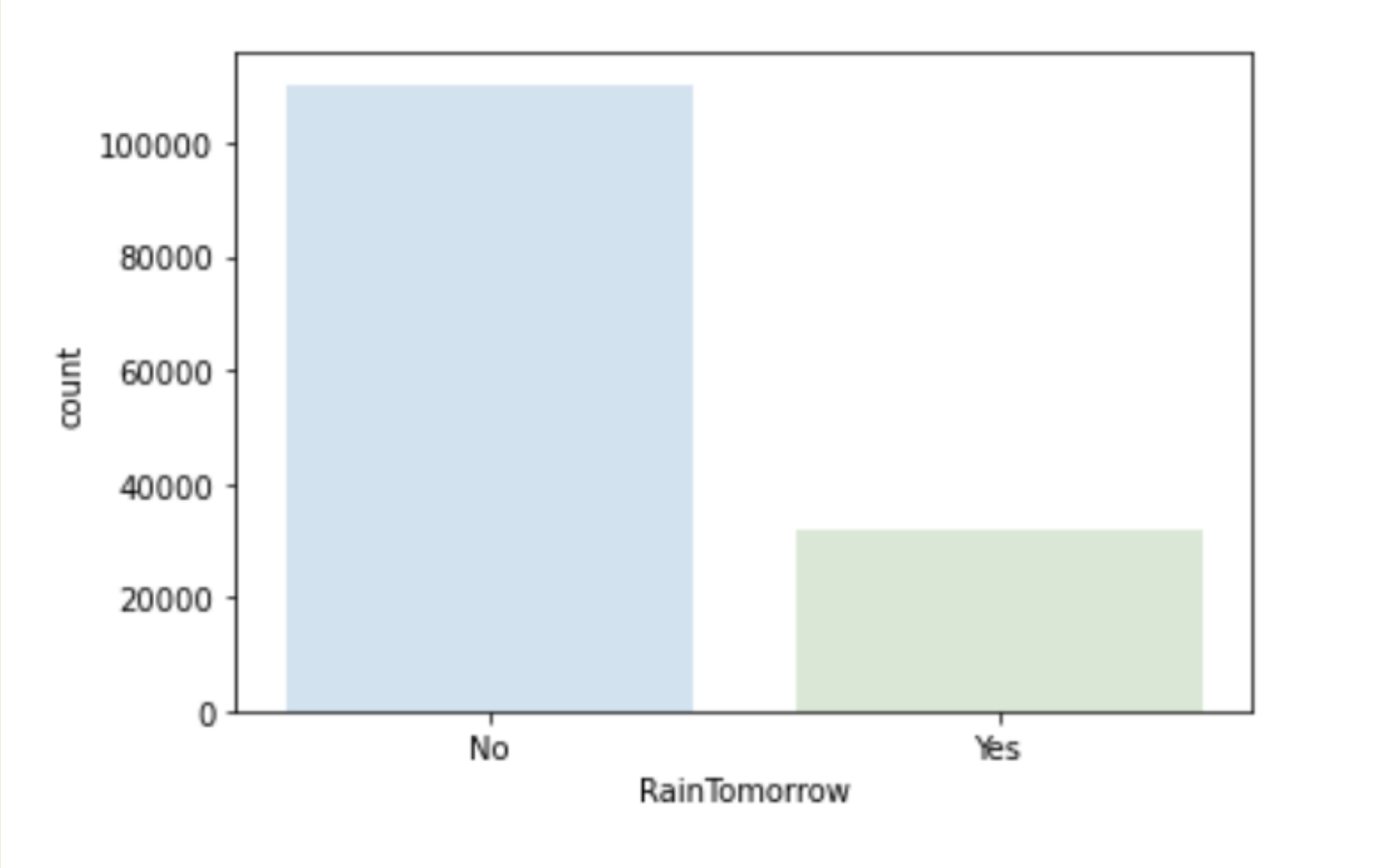
```
model = Sequential()
```

```
model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu', input_dim = 23))
```

```
model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))

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'))
```

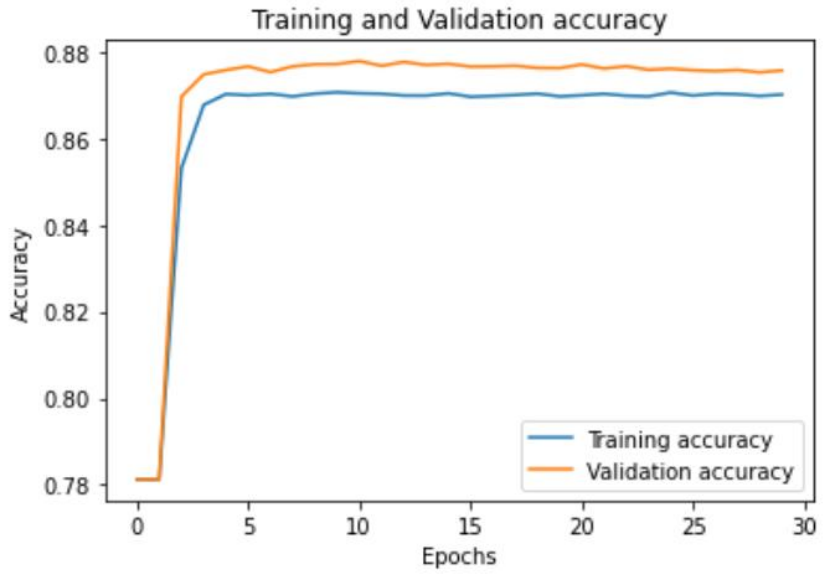
```
model.add(Flatten())
```

```
opt = Adam(learning_rate=0.00009)
model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])

# Train the ANN
history = model.fit(x_train, y_train, batch_size = 26, epochs = 30, callbacks=[early_stopping], validation_split=0.2)
```




- RainTomorrow is the target variable to predict. It means did it rain the next day, Yes or No?
- This column is Yes if the rain for that day was 1mm or more.

```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['accuracy']], label='Training accura
plt.plot(history_df.loc[:, ['val_accuracy']], label='Validation

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




```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['loss']], label='Training loss')
plt.plot(history_df.loc[:, ['val_loss']], label='Validation l
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc="best")

plt.show()
```