Machine learning

Final project

Omer Ofir 203769187

Roee Zvi 316491661

Task:

חיזוי גשם למחרת על ידי אימון מודלים של סיווג על משתנה היעד

תוֹכֶן:

מערך נתונים זה מכיל כ-10 שנים של תצפיות מזג אוויר יומיות ממקומות רבים ברחבי אוסטרליה.

RainTomorrow הוא משתנה היעד לניבוי

זה אומר -- האם ירד גשם למחרת, כן או לא? עמודה זו היא כן אם הגשם באותו יום היה 1 מ"מ או יותר.

עליך בנות אלגוריתם חיזוי לפי הנתונים הצורפים

Description:

The task is to define if it will rain tomorrow in Australia. for this task, we are using the NN TensorFlow model and SKleran tools. This is a neural network that uses binary classification to predict whether, given meteorological observations of a given day at a given weather station in Australia, it will rain there the next day. The model is trained and tested on a dataset containing about 10 years of daily weather observations from numerous Australian weather stations.

As we said, we decided to implement this project with Tensorflow 2 and Keras and SKlearn.

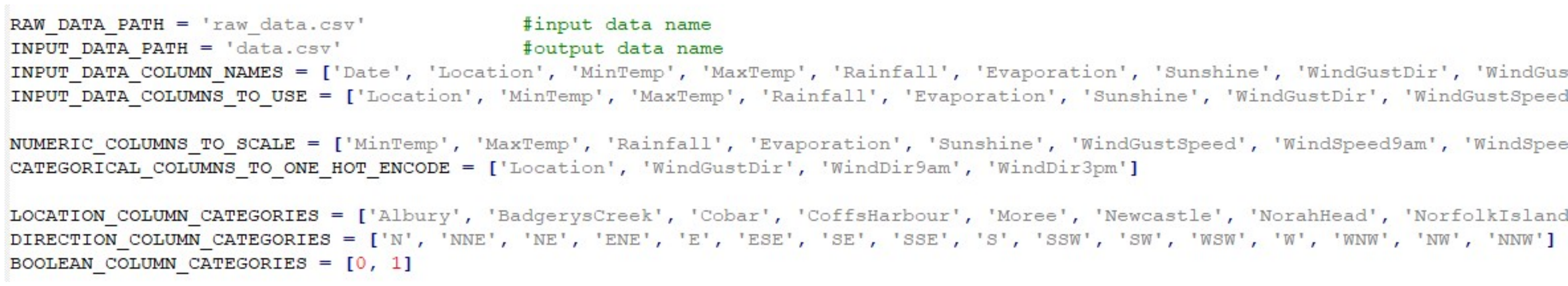
Milestones

- **Data Pre-processing -** get ready the data set.
- **Build the TensorFlow model -** define the layers, training set/learning set, Beach size and test data size.
- **Training the model -** define the number of Epoch.
- **Predict** - input the same data - predict if it will rain tomorrow.

Data Pre-processing

In this step, we organize the data, delete the unnecessary column, and create new data set to work with.

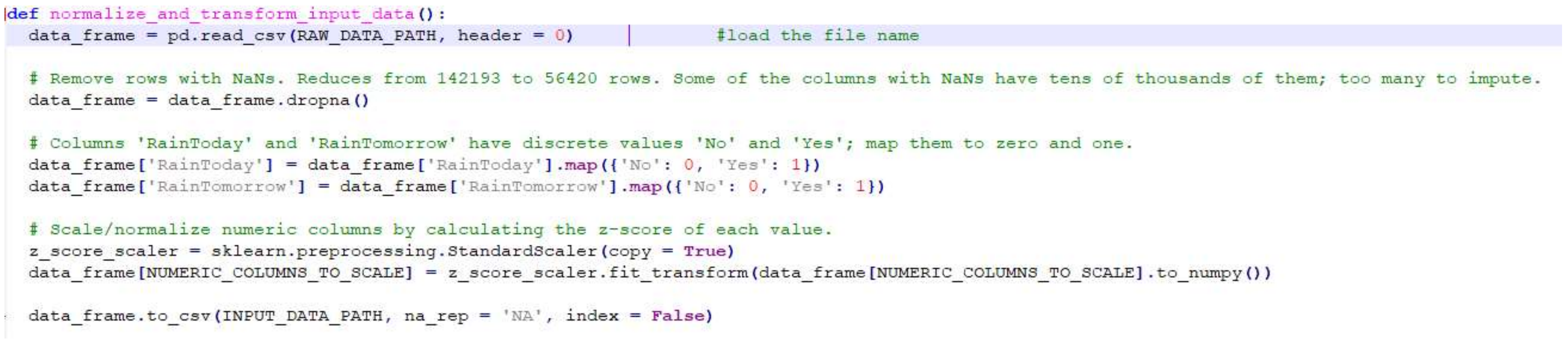
For this task, we build the python code: preprocessing.py



```
RAW_DATA_PATH = 'raw_data.csv'              #input data name
INPUT_DATA_PATH = 'data.csv'                #output data name
INPUT_DATA_COLUMN_NAMES = ['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustDir', 'WindGus
INPUT_DATA_COLUMNS_TO_USE = ['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustDir', 'WindGustSpeed

NUMERIC_COLUMNS_TO_SCALE = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpee
CATEGORICAL_COLUMNS_TO_ONE_HOT_ENCODE = ['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm']

LOCATION_COLUMN_CATEGORIES = ['Albury', 'BadgerysCreek', 'Cobar', 'CoffsHarbour', 'Moree', 'Newcastle', 'NorahHead', 'NorfolkIsland
DIRECTION_COLUMN_CATEGORIES = ['N', 'NNE', 'NE', 'ENE', 'E', 'ESE', 'SE', 'SSE', 'S', 'SSW', 'SW', 'WSW', 'W', 'WNW', 'NW', 'NNW']
BOOLEAN_COLUMN_CATEGORIES = [0, 1]
```

The first thing that we need to do is to load the data file, then, we create the new data set (.csv).

Define the input column and the columns we want to use, then, define all the numeric labels and the non-numeric label.

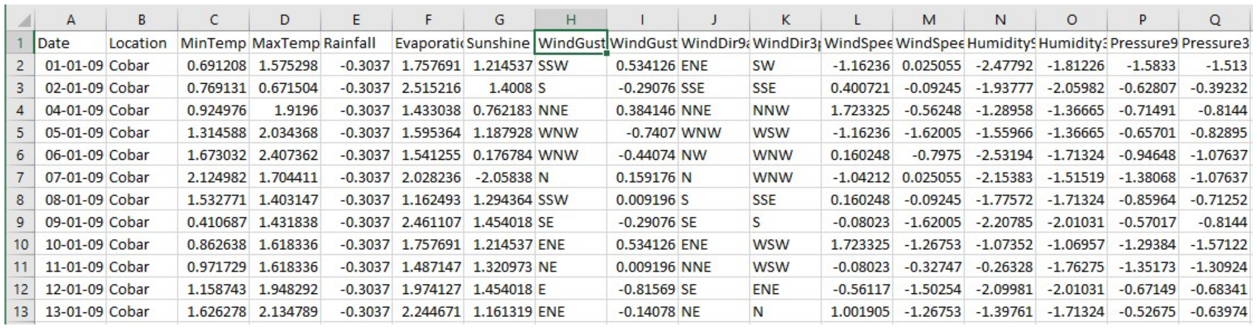


```
def normalize_and_transform_input_data():
  data_frame = pd.read_csv(RAW_DATA_PATH, header = 0)              #load the file name

  # Remove rows with NaNs. Reduces from 142193 to 56420 rows. Some of the columns with NaNs have tens of thousands of them; too many to impute.
  data_frame = data_frame.dropna()

  # Columns 'RainToday' and 'RainTomorrow' have discrete values 'No' and 'Yes'; map them to zero and one.
  data_frame['RainToday'] = data_frame['RainToday'].map({'No': 0, 'Yes': 1})
  data_frame['RainTomorrow'] = data_frame['RainTomorrow'].map({'No': 0, 'Yes': 1})

  # Scale/normalize numeric columns by calculating the z-score of each value.
  z_score_scaler = sklearn.preprocessing.StandardScaler(copy = True)
  data_frame[NUMERIC_COLUMNS_TO_SCALE] = z_score_scaler.fit_transform(data_frame[NUMERIC_COLUMNS_TO_SCALE].to_numpy())

  data_frame.to_csv(INPUT_DATA_PATH, na_rep = 'NA', index = False)
```

The next step is to remove all the rows that contains NaN (Reduce from 142193 rows to 56420 rows.).

Convert all the 'Yes/No' labels to 1/0, Scale/normalize numeric columns by calculating the z-score of each value.

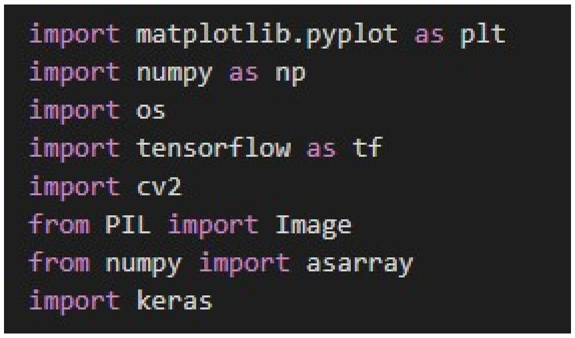
At the end of the process, the code will create new .csv file called 'data.csv



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporati	Sunshine	WindGust	WindGust	WindDir9a	WindDir3p	WindSpee	WindSpee	Humidity9	Humidity3	Pressure9	Pressure3
2	01-01-09	Cobar	0.691208	1.575298	-0.3037	1.757691	1.214537	SSW	0.534126	ENE	SW	-1.16236	0.025055	-2.47792	-1.81226	-1.5833	-1.513
3	02-01-09	Cobar	0.769131	0.671504	-0.3037	2.515216	1.4008	S	-0.29076	SSE	SSE	0.400721	-0.09245	-1.93777	-2.05982	-0.62807	-0.39232
4	04-01-09	Cobar	0.924976	1.9196	-0.3037	1.433038	0.762183	NNE	0.384146	NNE	NNW	1.723325	-0.56248	-1.28958	-1.36665	-0.71491	-0.8144
5	05-01-09	Cobar	1.314588	2.034368	-0.3037	1.595364	1.187928	WNW	-0.7407	WNW	WSW	-1.16236	-1.62005	-1.55966	-1.36665	-0.65701	-0.82895
6	06-01-09	Cobar	1.673032	2.407362	-0.3037	1.541255	0.176784	WNW	-0.44074	NW	WNW	0.160248	-0.7975	-2.53194	-1.71324	-0.94648	-1.07637
7	07-01-09	Cobar	2.124982	1.704411	-0.3037	2.028236	-2.05838	N	0.159176	N	WNW	-1.04212	0.025055	-2.15383	-1.51519	-1.38068	-1.07637
8	08-01-09	Cobar	1.532771	1.403147	-0.3037	1.162493	1.294364	SSW	0.009196	S	SSE	0.160248	-0.09245	-1.77572	-1.71324	-0.85964	-0.71252
9	09-01-09	Cobar	0.410687	1.431838	-0.3037	2.461107	1.454018	SE	-0.29076	SE	S	-0.08023	-1.62005	-2.20785	-2.01031	-0.57017	-0.8144
10	10-01-09	Cobar	0.862638	1.618336	-0.3037	1.757691	1.214537	ENE	0.534126	ENE	WSW	1.723325	-1.26753	-1.07352	-1.06957	-1.29384	-1.57122
11	11-01-09	Cobar	0.971729	1.618336	-0.3037	1.487147	1.320973	NE	0.009196	NNE	WSW	-0.08023	-0.32747	-0.26328	-1.76275	-1.35173	-1.30924
12	12-01-09	Cobar	1.158743	1.948292	-0.3037	1.974127	1.454018	E	-0.81569	SE	ENE	-0.56117	-1.50254	-2.09981	-2.01031	-0.67149	-0.68341
13	13-01-09	Cobar	1.626278	2.134789	-0.3037	2.244671	1.161319	ENE	-0.14078	NE	N	1.001905	-1.26753	-1.39761	-1.71324	-0.52675	-0.63974

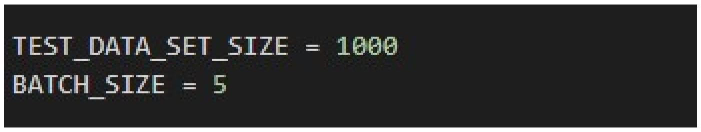
Build the TensorFlow model

The first thing we need to do is to call all the libraries:



```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
import cv2
from PIL import Image
from numpy import asarray
import keras
```

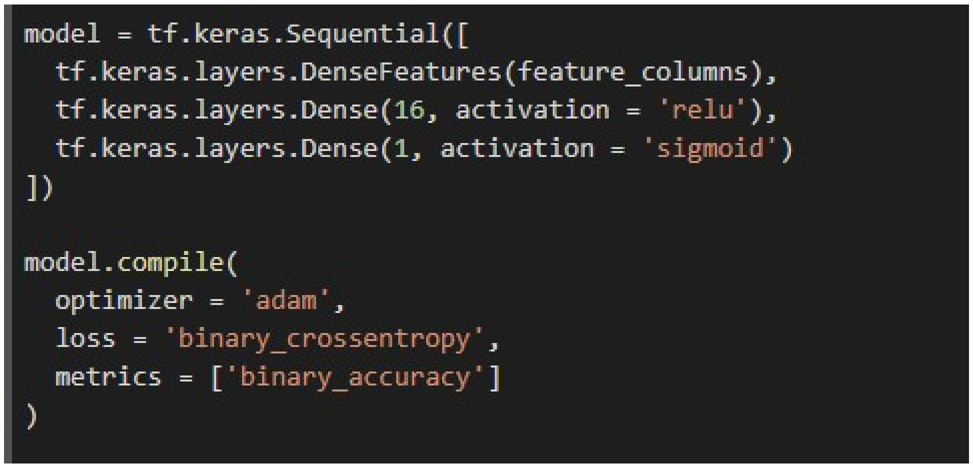
As you can see, we have used the TensorFlow and Keras libraries and a couple more libraries for execution.



```
TEST_DATA_SET_SIZE = 1000
BATCH_SIZE = 5
```

The number lines that we take from the file.

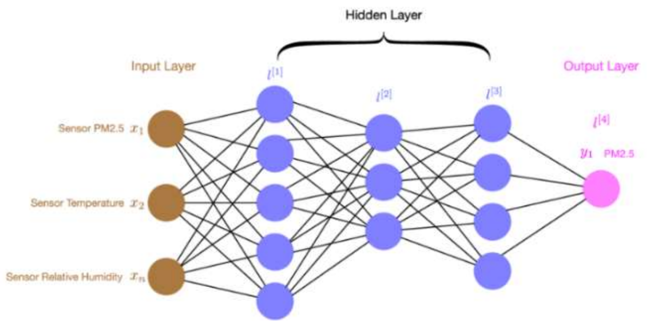
The number of batch (how many lines the ML learn in peralla).



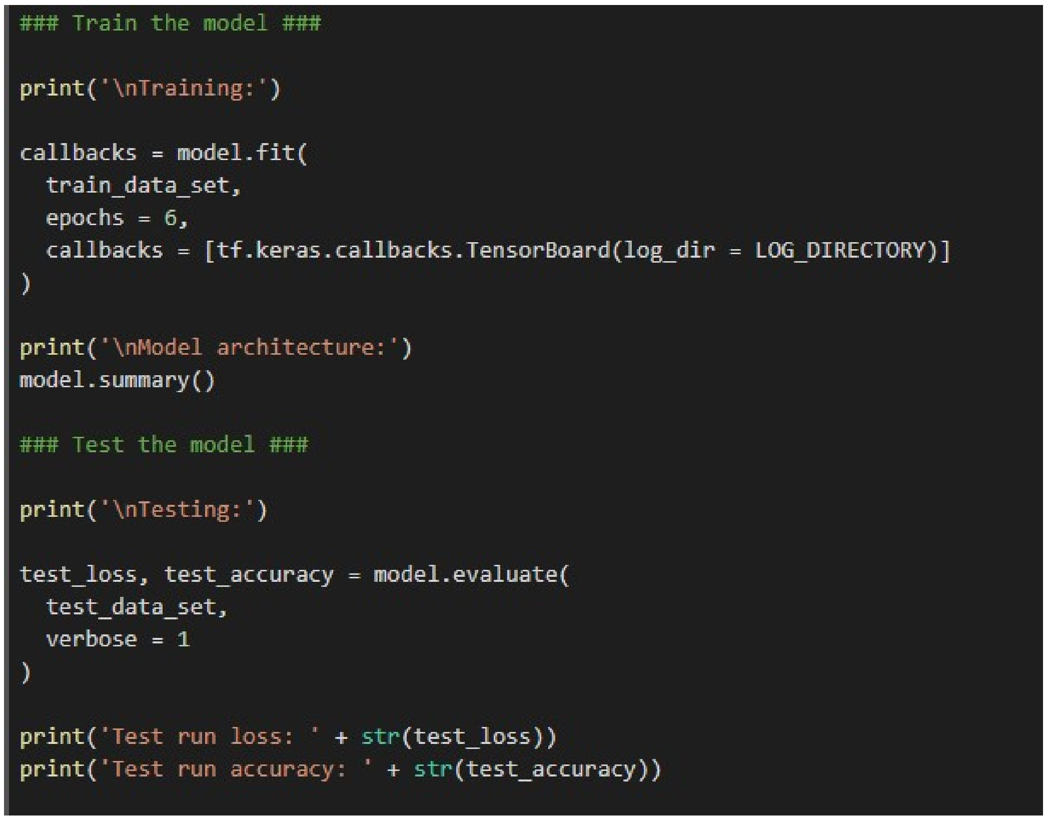
```
model = tf.keras.Sequential([
  tf.keras.layers.DenseFeatures(feature_columns),
  tf.keras.layers.Dense(16, activation = 'relu'),
  tf.keras.layers.Dense(1, activation = 'sigmoid')
])

model.compile(
  optimizer = 'adam',
  loss = 'binary_crossentropy',
  metrics = ['binary_accuracy']
)
```

Define the batch size and the data set size (we don't need to use all the data sets).

build the NN and use "Adam" binary decision.

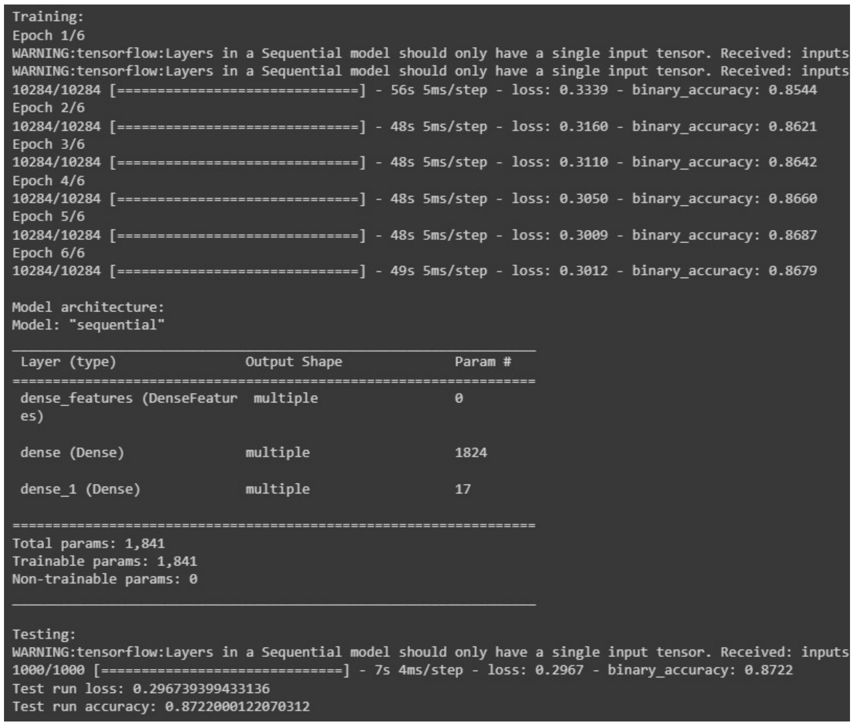


Training the model



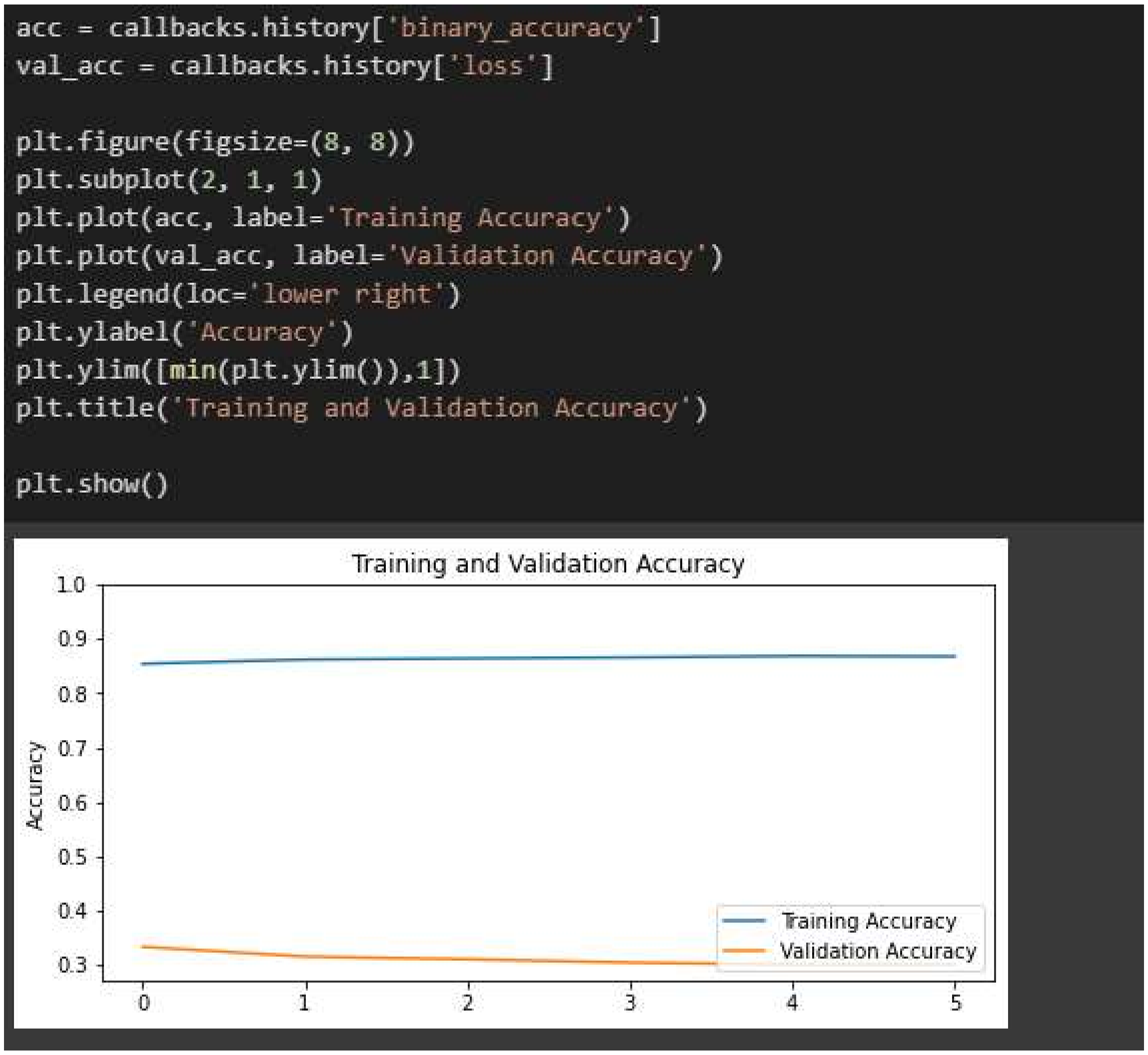
```
### Train the model ###

print('\nTraining:')

callbacks = model.fit(
  train_data_set,
  epochs = 6,
  callbacks = [tf.keras.callbacks.TensorBoard(log_dir = LOG_DIRECTORY)]
)

print('\nModel architecture:')
model.summary()

### Test the model ###

print('\nTesting:')

test_loss, test_accuracy = model.evaluate(
  test_data_set,
  verbose = 1
)

print('Test run loss: ' + str(test_loss))
print('Test run accuracy: ' + str(test_accuracy))
```

For this run use only 6 epochs. We have called the .fit function (from Kares) to start the training).

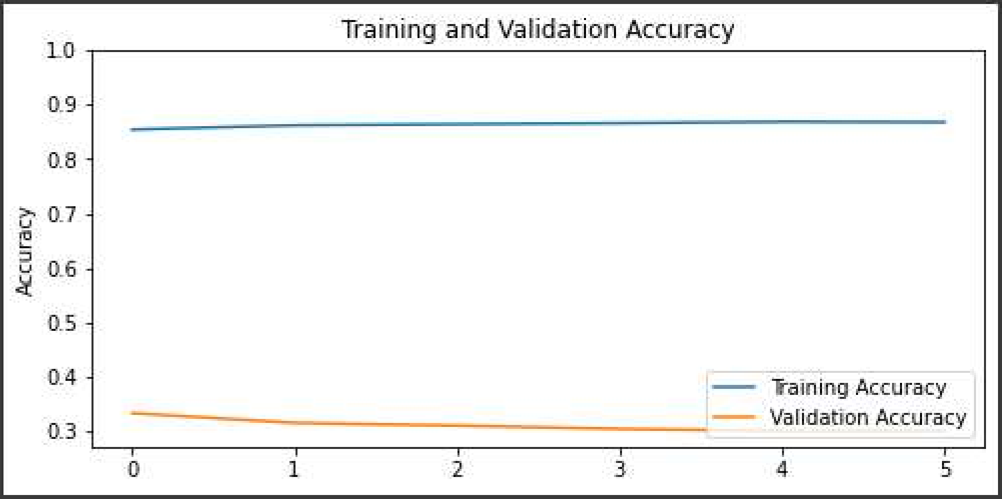


```
Training:
Epoch 1/6
WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor. Received: inputs
WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor. Received: inputs
10284/10284 [==============================] - 56s 5ms/step - loss: 0.3339 - binary_accuracy: 0.8544
Epoch 2/6
10284/10284 [==============================] - 48s 5ms/step - loss: 0.3160 - binary_accuracy: 0.8621
Epoch 3/6
10284/10284 [==============================] - 48s 5ms/step - loss: 0.3110 - binary_accuracy: 0.8642
Epoch 4/6
10284/10284 [==============================] - 48s 5ms/step - loss: 0.3050 - binary_accuracy: 0.8660
Epoch 5/6
10284/10284 [==============================] - 48s 5ms/step - loss: 0.3009 - binary_accuracy: 0.8687
Epoch 6/6
10284/10284 [==============================] - 49s 5ms/step - loss: 0.3012 - binary_accuracy: 0.8679

Model architecture:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_features (DenseFeatur  multiple                  0
 es)

 dense (Dense)               multiple                  1824

 dense_1 (Dense)             multiple                  17

=================================================================
Total params: 1,841
Trainable params: 1,841
Non-trainable params: 0
_________________________________________________________________

Testing:
WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor. Received: inputs
1000/1000 [==============================] - 7s 4ms/step - loss: 0.2967 - binary_accuracy: 0.8722
Test run loss: 0.296739399433136
Test run accuracy: 0.8722000122070312
```

As you can see, we got an accuracy of 87% and a loss of 30% (if we will train the algorithm more we can get better results).

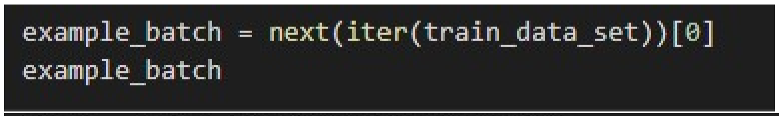


```
acc = callbacks.history['binary_accuracy']
val_acc = callbacks.history['loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.show()
```

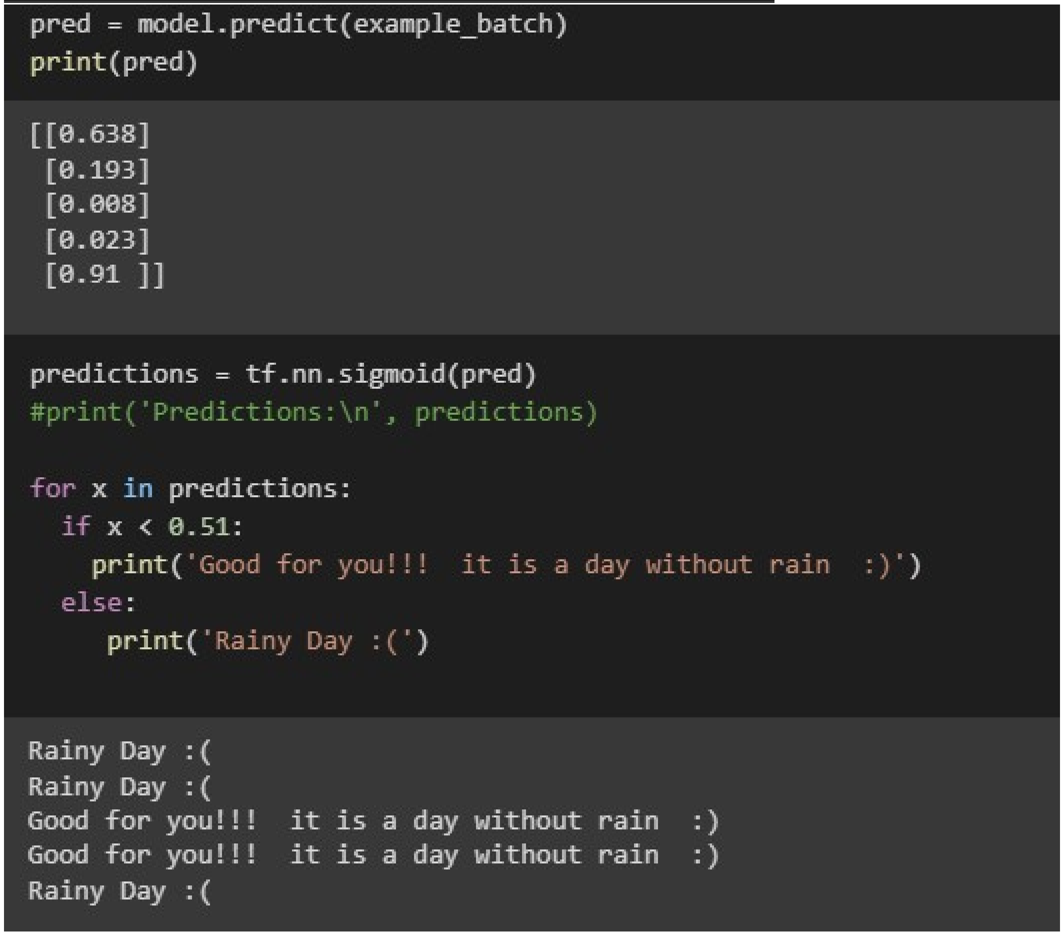


Predict-

for the prediction we have used 2 data, 1 from the train and one from the test. for the training dataset, the algorithm needs to predict 100 % because the algorithm knows the data, the test dataset is the real test.



```
example_batch = next(iter(train_data_set))[0]
example_batch
```



```
pred = model.predict(example_batch)
print(pred)
```

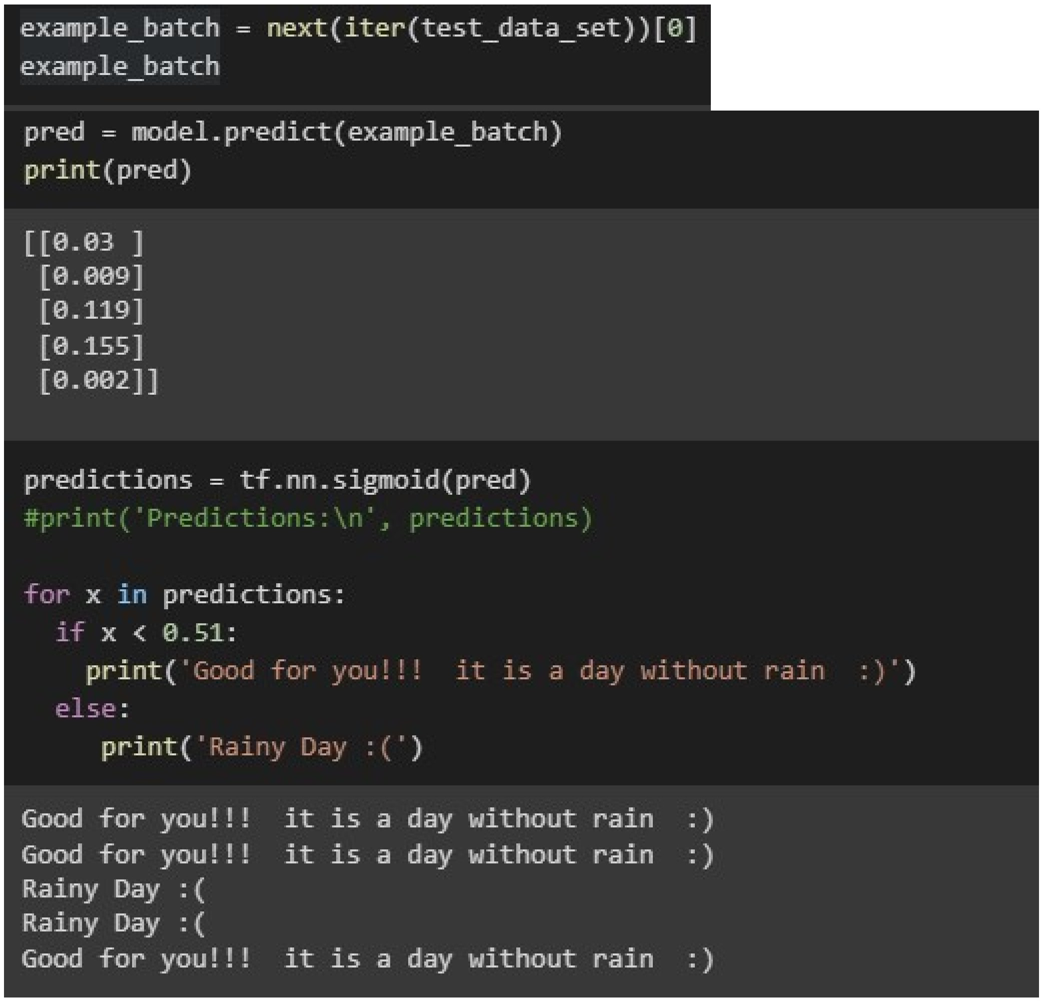
```
[[0.638]
 [0.193]
 [0.008]
 [0.023]
 [0.91 ]]
```

```
predictions = tf.nn.sigmoid(pred)
#print('Predictions:\n', predictions)

for x in predictions:
  if x < 0.51:
    print('Good for you!!!  it is a day without rain  :)')
  else:
     print('Rainy Day :(')
```

```
Rainy Day :(
Rainy Day :(
Good for you!!!  it is a day without rain  :)
Good for you!!!  it is a day without rain  :)
Rainy Day :(
```

We have used 1 batch (that includes 5 rows from the dataset). and the predicted results are at the bottom.

For the test set:



```
example_batch = next(iter(test_data_set))[0]
example_batch
```

```
pred = model.predict(example_batch)
print(pred)
```

```
[[0.03 ]
 [0.009]
 [0.119]
 [0.155]
 [0.002]]
```

```
predictions = tf.nn.sigmoid(pred)
#print('Predictions:\n', predictions)

for x in predictions:
  if x < 0.51:
    print('Good for you!!!  it is a day without rain  :)')
  else:
     print('Rainy Day :(')
```

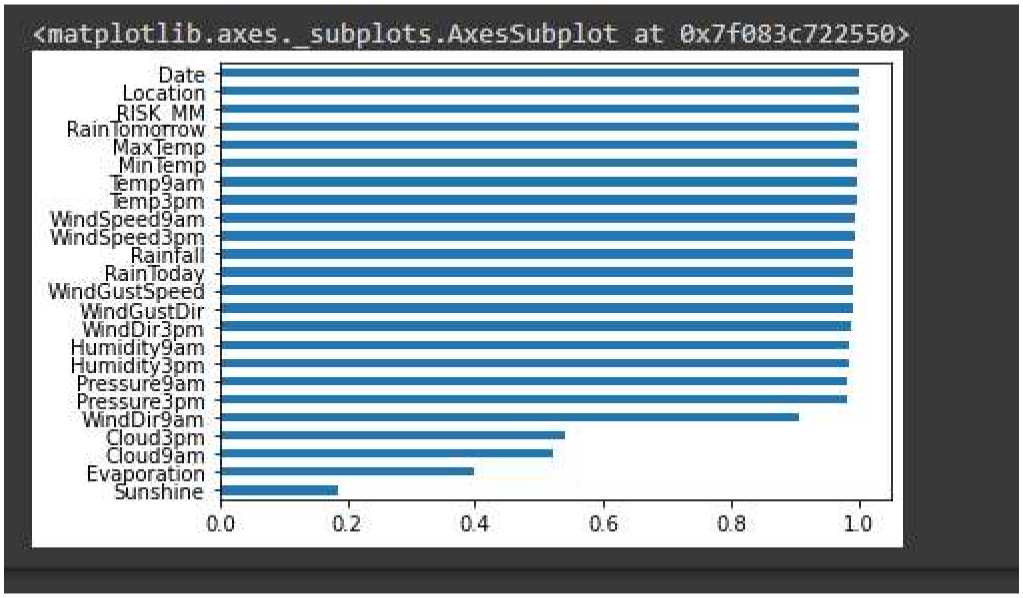
```
Good for you!!!  it is a day without rain  :)
Good for you!!!  it is a day without rain  :)
Rainy Day :(
Rainy Day :(
Good for you!!!  it is a day without rain  :)
```

The full code is an attached.

Second method -Sklearn

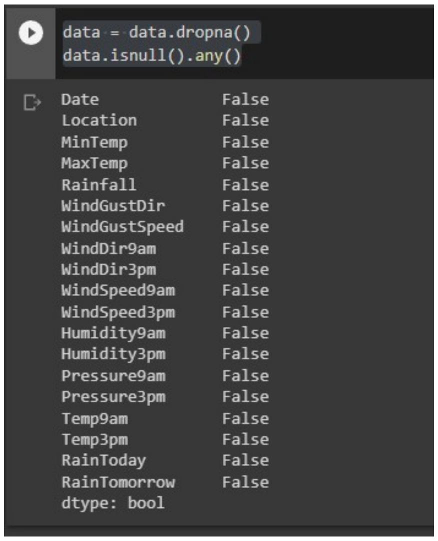
The type of machine learning we will be doing is called classification, because when we make predictions, we are classifying each day as rainy or not. More specifically, we are performing binary classification, which means that there are only two different states we are classifying.

Null values Let's get rid of columns with significant amount of null values. And in the rest columns we will drop rows with null values.


```
<matplotlib.axes._subplots.AxesSubplot at 0x7f083c722550>
```

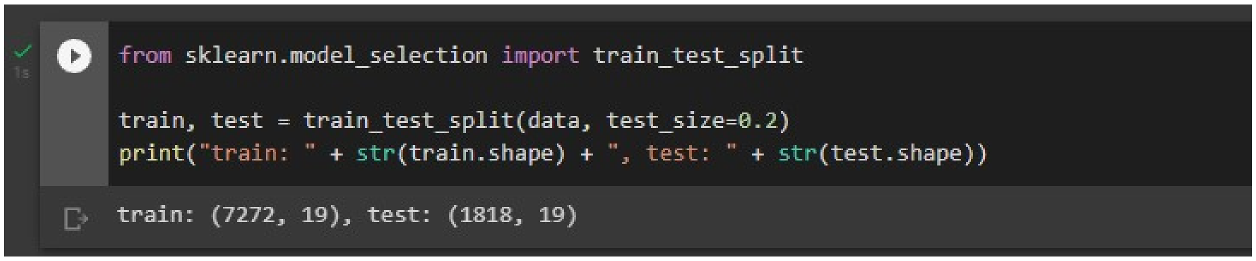
Cloud9pm, Cloud3pm, Evaporation, and Sunshine must be dropped since significant amount of records in these columns is missed. Also, we should exclude RISK_MM because it can leak the answers to the model and reduce its predictability.

Let's drop rows with null values in them.


```
data = data.dropna()
data.isnull().any()
```

```
Date             False
Location         False
MinTemp          False
MaxTemp          False
Rainfall         False
WindGustDir      False
WindGustSpeed    False
WindDir9am       False
WindDir3pm       False
WindSpeed9am     False
WindSpeed3pm     False
Humidity9am      False
Humidity3pm      False
Pressure9am      False
Pressure3pm      False
Temp9am          False
Temp3pm          False
RainToday        False
RainTomorrow     False
dtype: bool
```

Split into train and test, we must be aware of one important thing: any change we make to the train data, we also need to make to the test data, otherwise we will be unable to use our model.

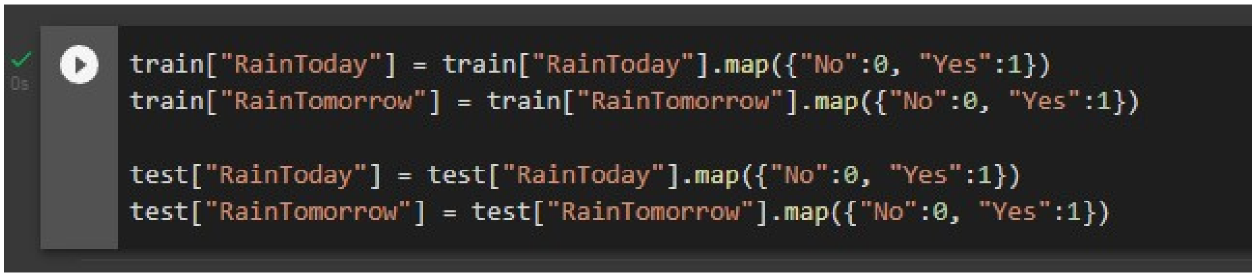


```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, test_size=0.2)
print("train: " + str(train.shape) + ", test: " + str(test.shape))
```

```
train: (7272, 19), test: (1818, 19)
```

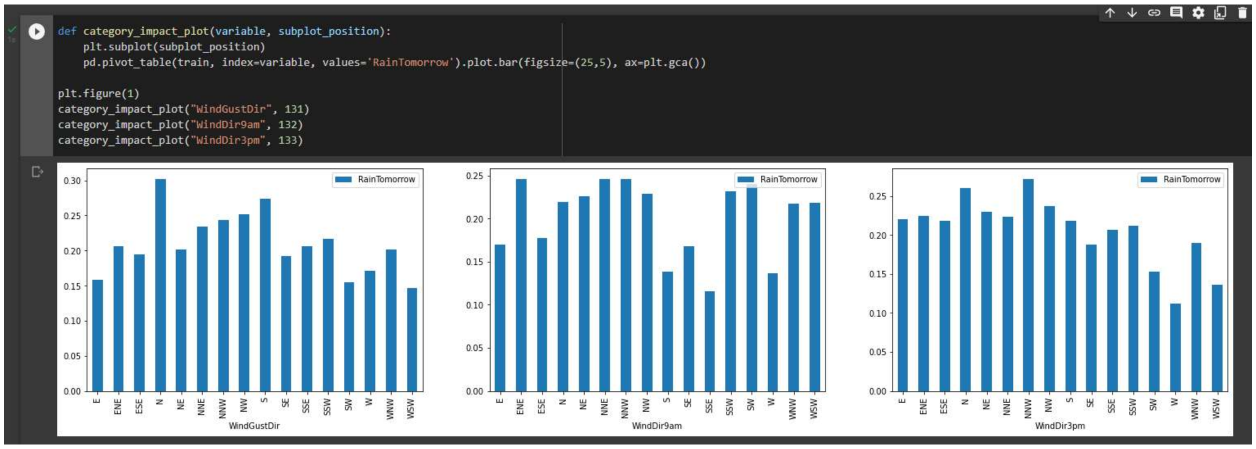
Deal with categorical variables, to apply such algorithms as Logistic Regression we need to convert the non-numeric data into numeric data. Categorical variables with only 2 possible values can be converted into variables with 0s and 1s as values. For categorical variables with 3 and more possible value we will create dummy variables.

Convert values in columns "RainToday" and "RainTomorrow" from "No" and "Yes" to 0 and 1

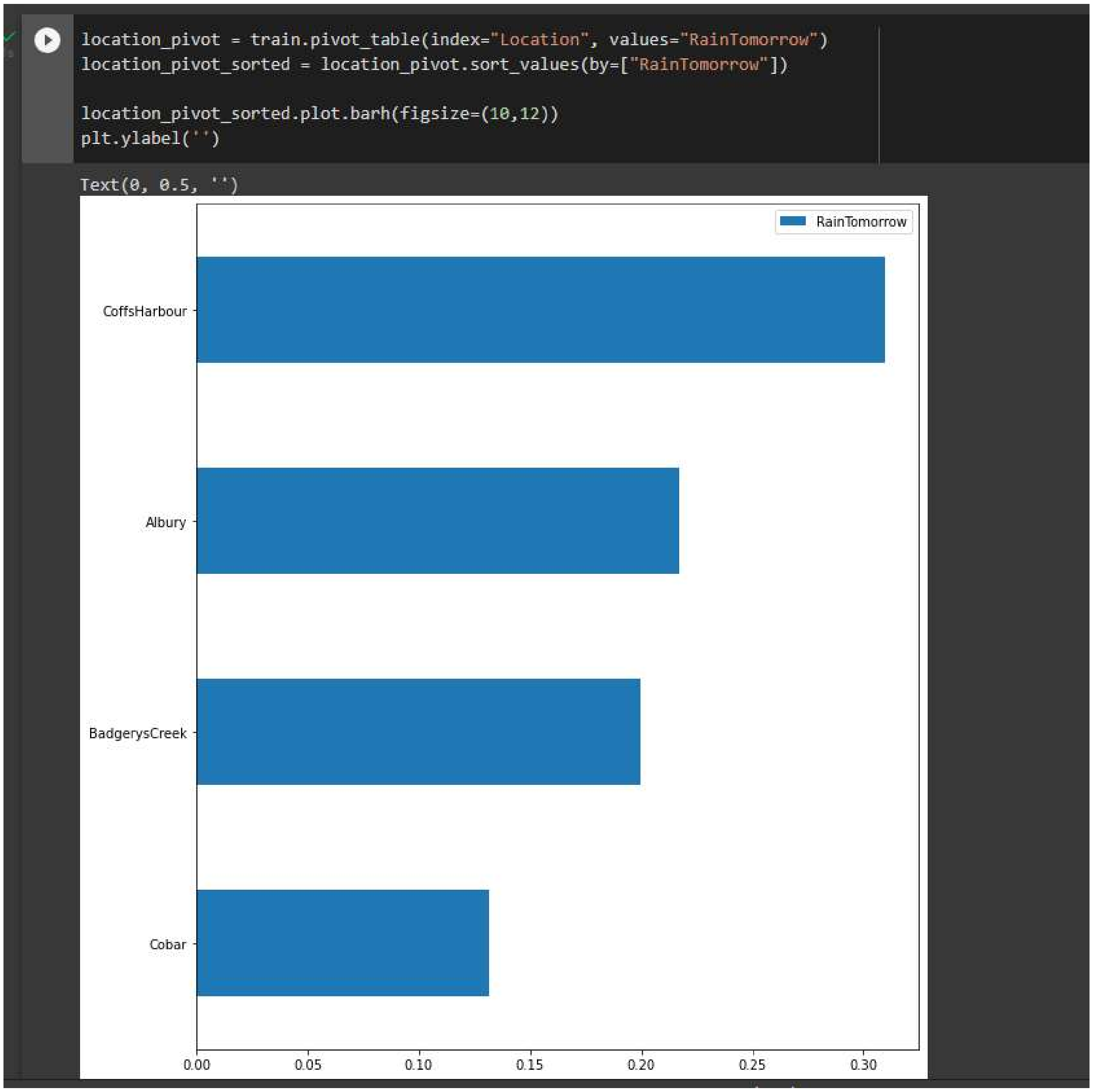


```
train["RainToday"] = train["RainToday"].map({"No":0, "Yes":1})
train["RainTomorrow"] = train["RainTomorrow"].map({"No":0, "Yes":1})

test["RainToday"] = test["RainToday"].map({"No":0, "Yes":1})
test["RainTomorrow"] = test["RainTomorrow"].map({"No":0, "Yes":1})
```

Visualization of how categorical variables impact on forming tomorrow's rain.



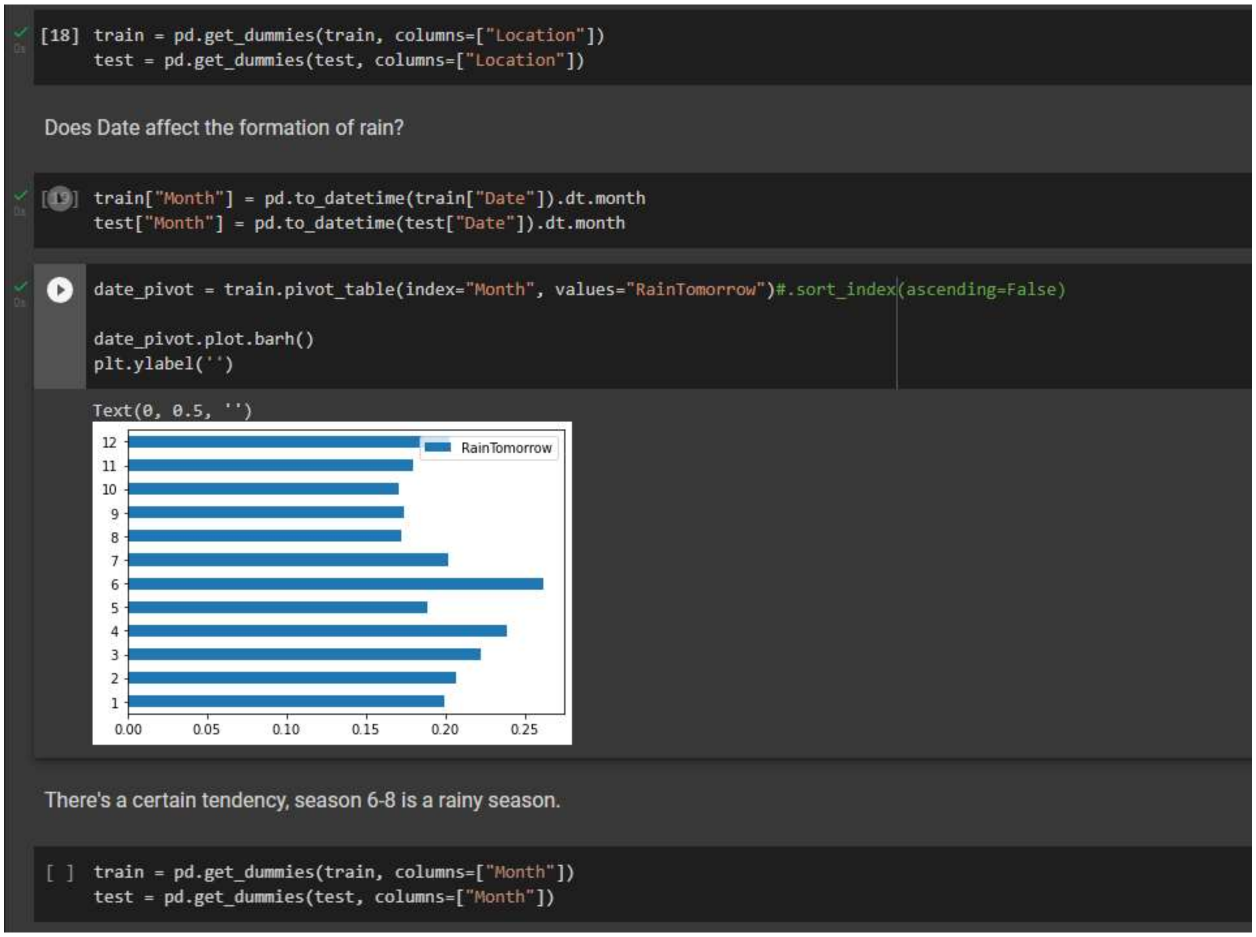
```
def category_impact_plot(variable, subplot_position):
    plt.subplot(subplot_position)
    pd.pivot_table(train, index=variable, values='RainTomorrow').plot.bar(figsize=(25,5), ax=plt.gca())

plt.figure(1)
category_impact_plot("WindGustDir", 131)
category_impact_plot("WindDir9am", 132)
category_impact_plot("WindDir3pm", 133)
```



```
location_pivot = train.pivot_table(index="Location", values="RainTomorrow")
location_pivot_sorted = location_pivot.sort_values(by=["RainTomorrow"])

location_pivot_sorted.plot.barh(figsize=(10,12))
plt.ylabel('')
```

```
Text(0, 0.5, '')
```

Yes, Location obviously affects the formation of tomorrow's rain! So, we're going to use this variable, and to use this categorical variable we must create dummies.



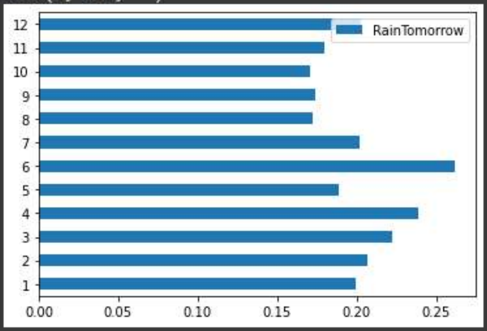
```
[18] train = pd.get_dummies(train, columns=["Location"])
     test = pd.get_dummies(test, columns=["Location"])
```

Does Date affect the formation of rain?

```
[19] train["Month"] = pd.to_datetime(train["Date"]).dt.month
     test["Month"] = pd.to_datetime(test["Date"]).dt.month
```

```
date_pivot = train.pivot_table(index="Month", values="RainTomorrow")#.sort_index(ascending=False)

date_pivot.plot.barh()
plt.ylabel('')
```

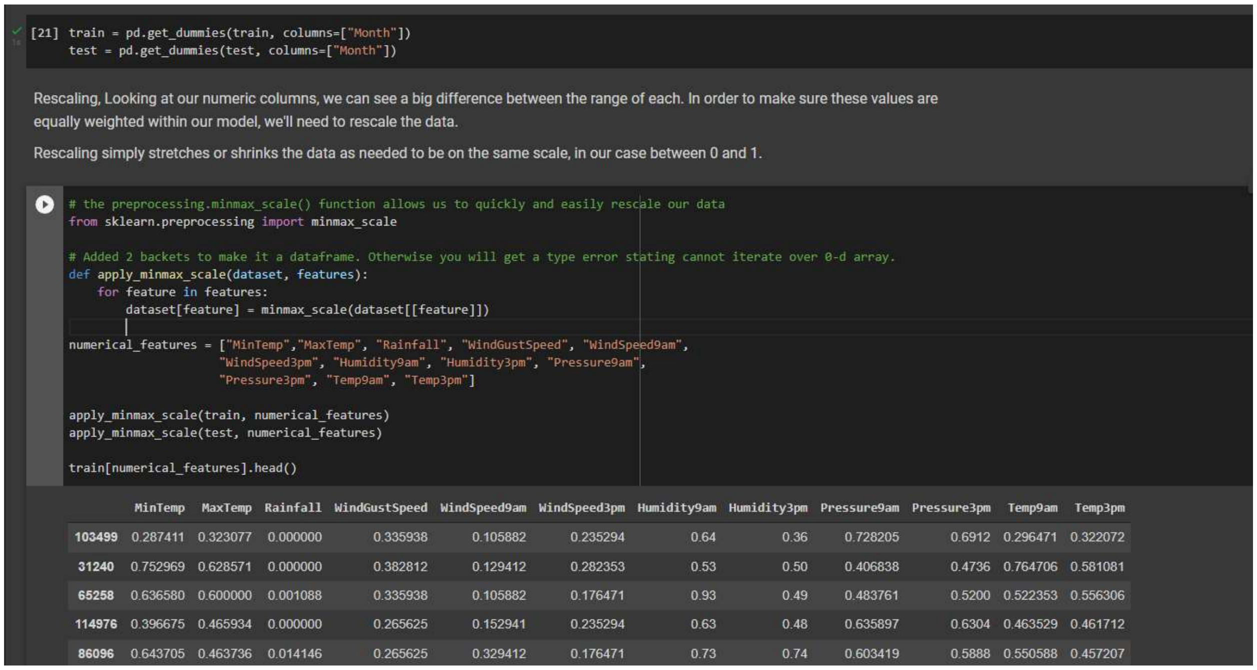
```
Text(0, 0.5, '')
```



There's a certain tendency, season 6-8 is a rainy season.

```
[ ] train = pd.get_dummies(train, columns=["Month"])
    test = pd.get_dummies(test, columns=["Month"])
```

There's a certain tendency, season 6-8 is a rainy season.



```
[21] train = pd.get_dummies(train, columns=["Month"])
     test = pd.get_dummies(test, columns=["Month"])
```

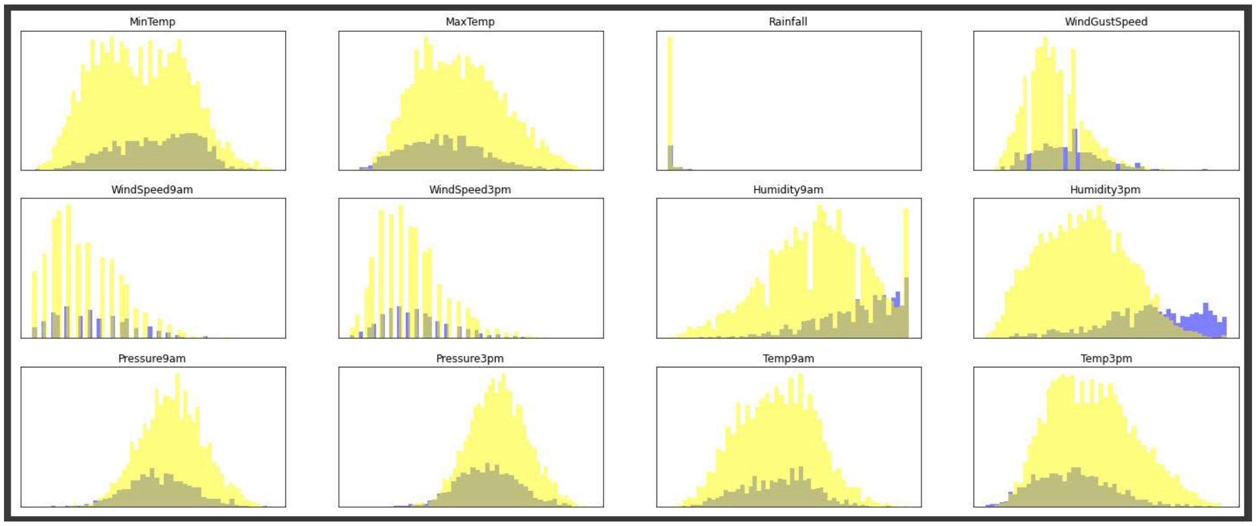
Rescaling, Looking at our numeric columns, we can see a big difference between the range of each. In order to make sure these values are equally weighted within our model, we'll need to rescale the data.

Rescaling simply stretches or shrinks the data as needed to be on the same scale, in our case between 0 and 1.

```
# the preprocessing.minmax_scale() function allows us to quickly and easily rescale our data
from sklearn.preprocessing import minmax_scale

# Added 2 backets to make it a dataframe. Otherwise you will get a type error stating cannot iterate over 0-d array.
def apply_minmax_scale(dataset, features):
    for feature in features:
        dataset[feature] = minmax_scale(dataset[[feature]])

numerical_features = ["MinTemp","MaxTemp", "Rainfall", "WindGustSpeed", "WindSpeed9am",
                     "WindSpeed3pm", "Humidity9am", "Humidity3pm", "Pressure9am",
                     "Pressure3pm", "Temp9am", "Temp3pm"]

apply_minmax_scale(train, numerical_features)
apply_minmax_scale(test, numerical_features)

train[numerical_features].head()
```

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Temp9am	Temp3pm
103499	0.287411	0.323077	0.000000	0.335938	0.105882	0.235294	0.64	0.36	0.728205	0.6912	0.296471	0.322072
31240	0.752969	0.628571	0.000000	0.382812	0.129412	0.282353	0.53	0.50	0.406838	0.4736	0.764706	0.581081
65258	0.636580	0.600000	0.001088	0.335938	0.105882	0.176471	0.93	0.49	0.483761	0.5200	0.522353	0.556306
114976	0.396675	0.465934	0.000000	0.265625	0.152941	0.235294	0.63	0.48	0.635897	0.6304	0.463529	0.461712
86096	0.643705	0.463736	0.014146	0.265625	0.329412	0.176471	0.73	0.74	0.603419	0.5888	0.550588	0.457207

Visualization of how numerical variables impact on forming tomorrow's rain.

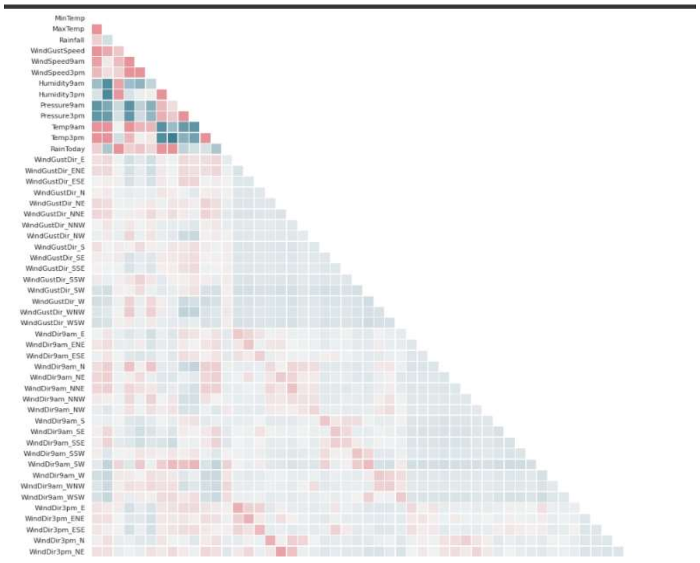


We are interested in variables with plots where blue and yellow areas have different shapes. Such variables have impact (positive or negative) on forming tomorrow's rain. The most obvious one is Humidity3pm! The rest is not that clear, we will use another feature selection method.

Collinearity, we now have 73 possible feature columns we can use to train our model. One thing to be aware of as you start to add more features is a concept called collinearity. Collinearity occurs where more than one feature contains data that are similar.

The effect of collinearity is that your model will overfit - you may get great results on your test data set, but then the model performs worse on unseen data (like the test set).

A common way to spot collinearity is to plot correlations between each pair of variables in a heatmap.



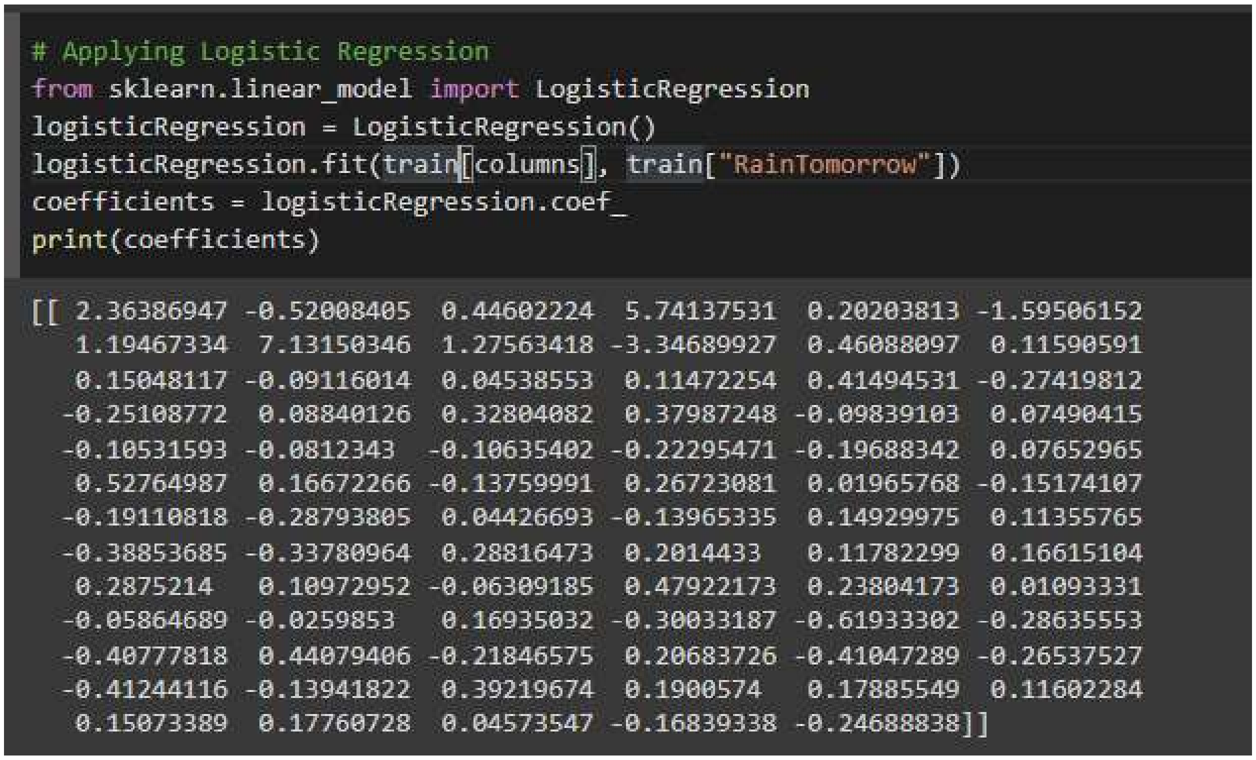
We can see that there is correlation about 30-50% between some variables. That's not enough to remove one of them and rely on the other.

Apart from that, we should remove one of each of our dummy variables to reduce the collinearity in each. We'll remove:

WindGustDir_E WindDir9am_E WindDir3pm_E

Feature selection to select the best-performing features, we need a way to measure which of our features are relevant to our outcome - in this case, the impact on forming tomorrow's rain. One effective way is by training a logistic regression model using all our features, and then looking at the coefficients of each feature.

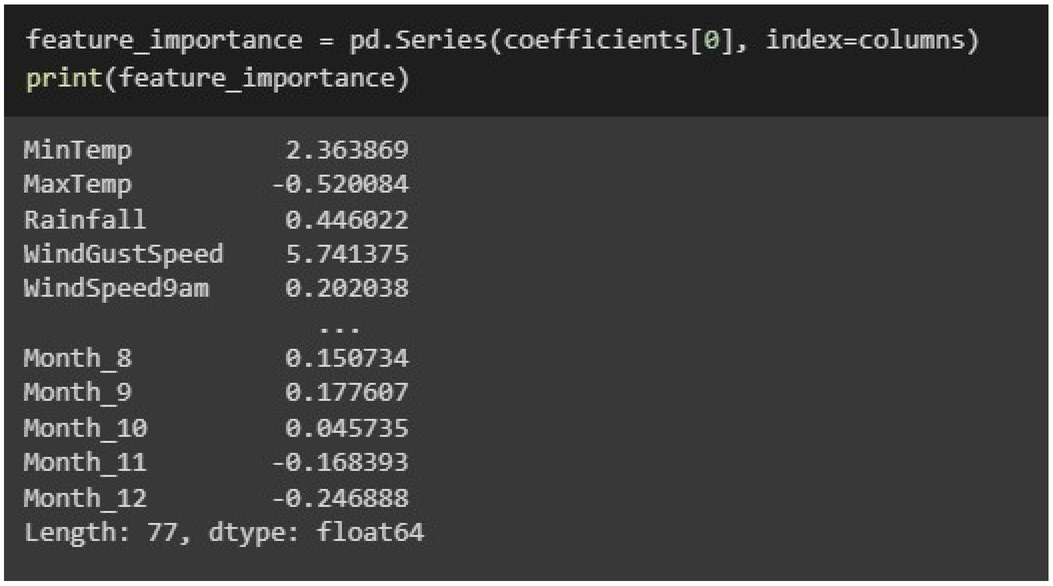
The scikit-learn Logistic Regression class has an attribute in which coefficients are stored after the model is fit, LogisticRegression.coef_. We first need to train our model, after which we can access this attribute.



```python
# Applying Logistic Regression
from sklearn.linear_model import LogisticRegression
logisticRegression = LogisticRegression()
logisticRegression.fit(train[columns], train["RainTomorrow"])
coefficients = logisticRegression.coef_
print(coefficients)
```

```
[[ 2.36386947 -0.52008405  0.44602224  5.74137531  0.20203813 -1.59506152
   1.19467334  7.13150346  1.27563418 -3.34689927  0.46088097  0.11590591
   0.15048117 -0.09116014  0.04538553  0.11472254  0.41494531 -0.27419812
  -0.25108772  0.08840126  0.32804082  0.37987248 -0.09839103  0.07490415
  -0.10531593 -0.0812343  -0.10635402 -0.22295471 -0.19688342  0.07652965
   0.52764987  0.16672266 -0.13759991  0.26723081  0.01965768 -0.15174107
  -0.19110818 -0.28793805  0.04426693 -0.13965335  0.14929975  0.11355765
  -0.38853685 -0.33780964  0.28816473  0.2014433   0.11782299  0.16615104
   0.2875214   0.10972952 -0.06309185  0.47922173  0.23804173  0.01093331
  -0.05864689 -0.0259853   0.16935032 -0.30033187 -0.61933302 -0.28635553
  -0.40777818  0.44079406 -0.21846575  0.20683726 -0.41047289 -0.26537527
  -0.41244116 -0.13941822  0.39219674  0.1900574   0.17885549  0.11602284
   0.15073389  0.17760728  0.04573547 -0.16839338 -0.24688838]]
```

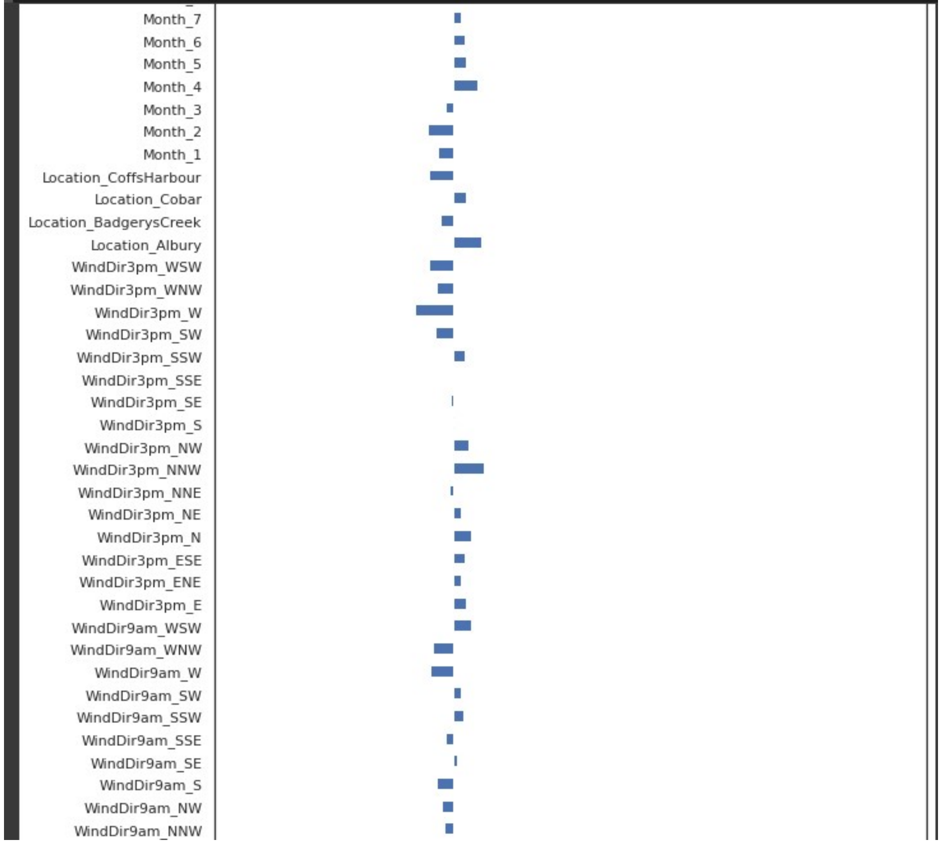
The coef() method returns a NumPy array of coefficients, in the same order as the features that were used to fit the model. To make these easier to interpret, we can convert the coefficients to a pandas series, adding the column names as the index:



```
feature_importance = pd.Series(coefficients[0], index=columns)
print(feature_importance)
```

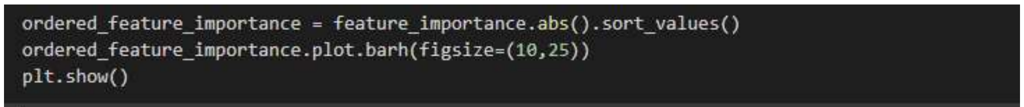
```
MinTemp          2.363869
MaxTemp         -0.520084
Rainfall         0.446022
WindGustSpeed    5.741375
WindSpeed9am     0.202038
                   ...
Month_8          0.150734
Month_9          0.177607
Month_10         0.045735
Month_11        -0.168393
Month_12        -0.246888
Length: 77, dtype: float64
```

```
# Plotting as a horizontal Bar chart
feature_importance.plot.barh(figsize=(10,25))
plt.show()
```

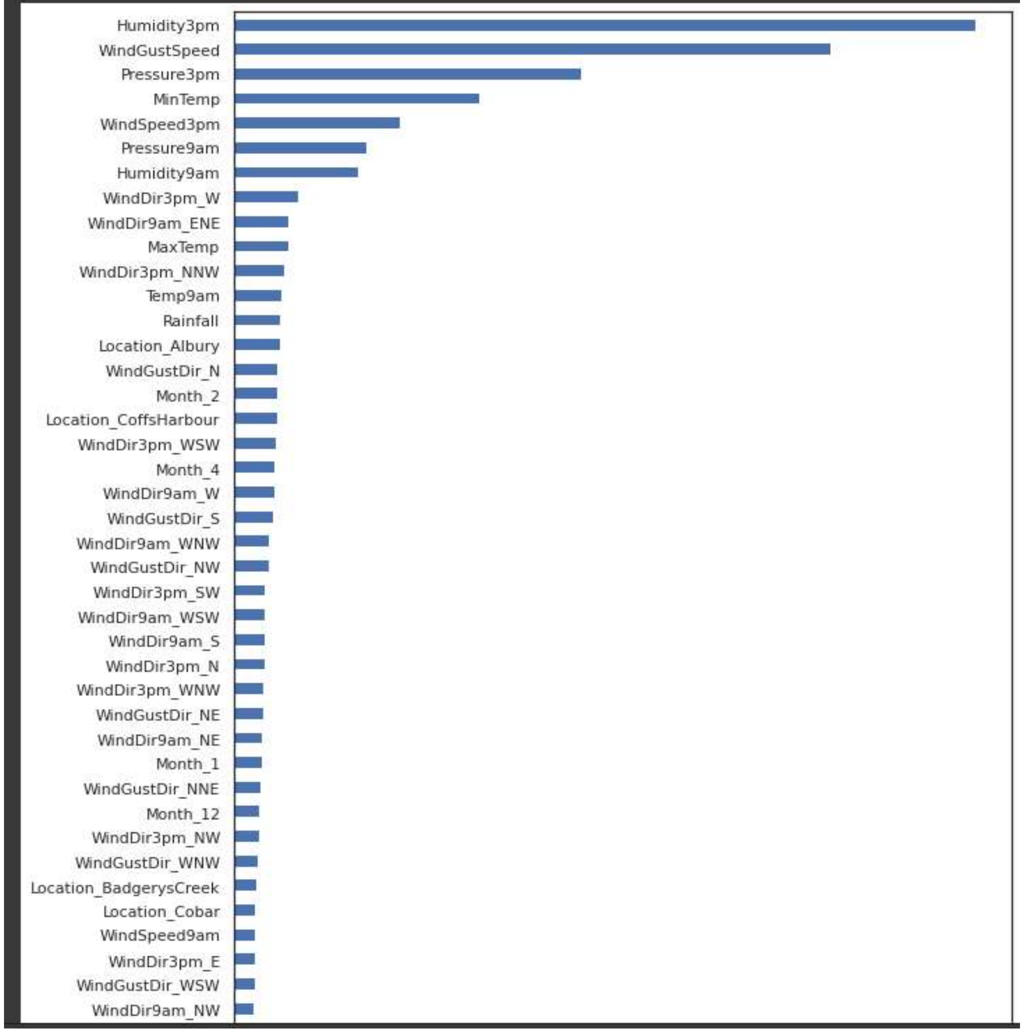


The plot we generated shows a range of both positive and negative values. Whether the value is positive or negative isn't as important in this case, relative to the magnitude of the value. If you think about it, this makes sense. A feature that indicates strongly whether it’s not going to rain tomorrow is just as useful as a feature that indicates strongly that it’s going to rain tomorrow, given they are mutually exclusive outcomes.

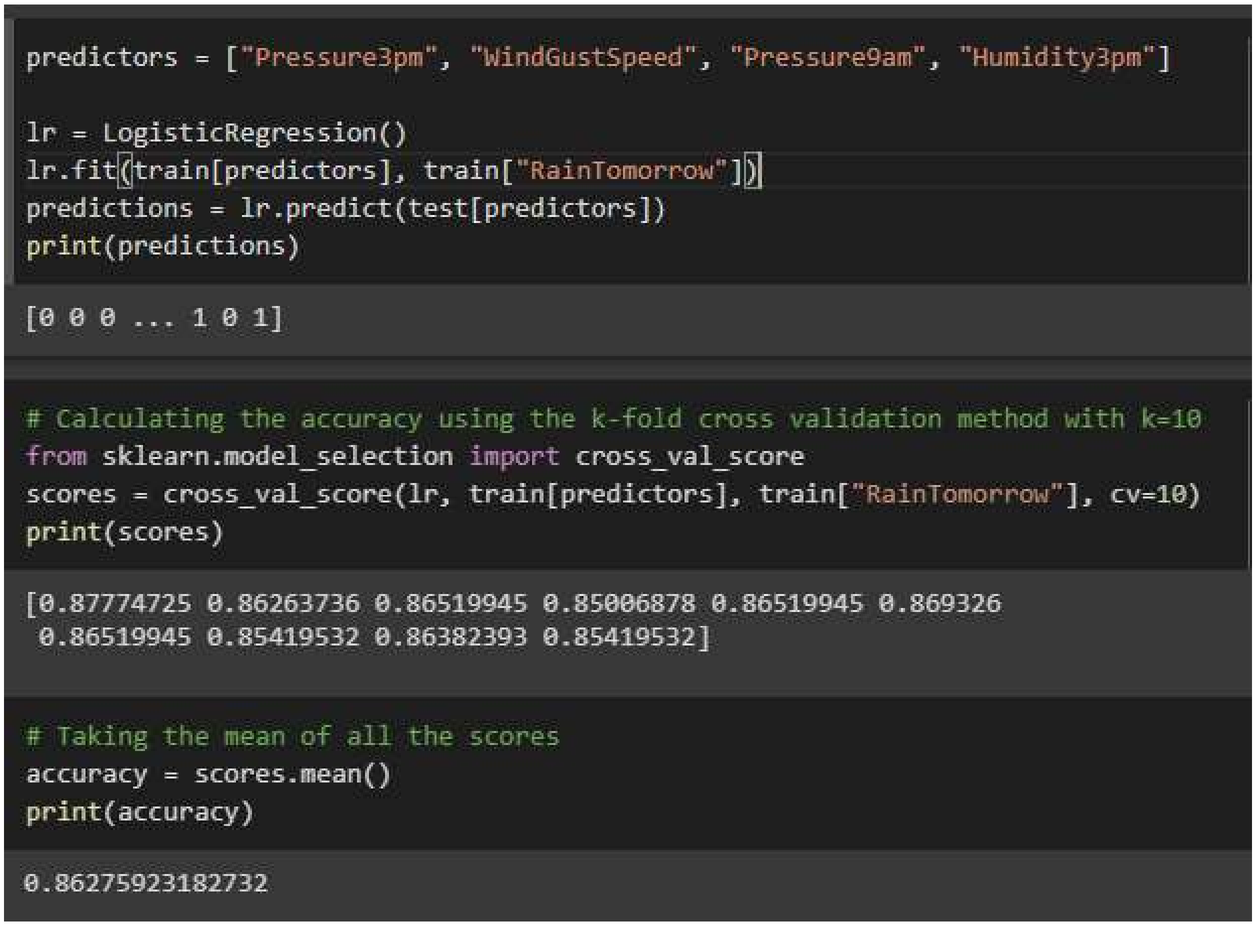
To make things easier to interpret, we'll alter the plot to show all positive values, and have sorted the bars in order of size:



```
ordered_feature_importance = feature_importance.abs().sort_values()
ordered_feature_importance.plot.barh(figsize=(10,25))
plt.show()
```



We'll train a model with the top 4 scores.



```
predictors = ["Pressure3pm", "WindGustSpeed", "Pressure9am", "Humidity3pm"]

lr = LogisticRegression()
lr.fit(train[predictors], train["RainTomorrow"])
predictions = lr.predict(test[predictors])
print(predictions)
```

```
[0 0 0 ... 1 0 1]
```

```
# Calculating the accuracy using the k-fold cross validation method with k=10
from sklearn.model_selection import cross_val_score
scores = cross_val_score(lr, train[predictors], train["RainTomorrow"], cv=10)
print(scores)
```

```
[0.87774725 0.86263736 0.86519945 0.85006878 0.86519945 0.869326
 0.86519945 0.85419532 0.86382393 0.85419532]
```

```
# Taking the mean of all the scores
accuracy = scores.mean()
print(accuracy)
```

```
0.86275923182732
```