Predicting AUS rain occurance using Logistic Regression

```
df.info()
```

```
from google.colab import drive
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv("/content/drive/MyDrive/TIP S.Y.'s/TIP 2021-2022 (3rd
yr, 1st sem)/Predictive Analytics w  Machine Learning/CPE 312 -
Machine Learning/data/logistic regression/weatherAUS.csv")
```

```
df.head()
```

```
         Date Location  MinTemp  ...  Temp3pm  RainToday  RainTomorrow
0  2008-12-01   Albury     13.4  ...     21.8         No            No
1  2008-12-02   Albury      7.4  ...     24.3         No            No
2  2008-12-03   Albury     12.9  ...     23.2         No            No
3  2008-12-04   Albury      9.2  ...     26.5         No            No
4  2008-12-05   Albury     17.5  ...     29.7         No            No

[5 rows x 23 columns]
```

Objective(s):

This activity aims to solve classification problem using logistic regression

```
print(df.shape)
```

```
(145460, 23)
```

```
df.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        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object 
 8   WindGustSpeed  135197 non-null  float64
```

```
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object
 22  RainTomorrow   142193 non-null  object
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

- Remove null values for RainToday and RainTomorrow since the latter is needed for training and testing and the former feels like a good predictor.

```
df.dropna(subset = ['RainToday','RainTomorrow'], inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 140787 entries, 0 to 145458
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           140787 non-null  object
 1   Location       140787 non-null  object
 2   MinTemp        140319 non-null  float64
 3   MaxTemp        140480 non-null  float64
 4   Rainfall       140787 non-null  float64
 5   Evaporation    81093 non-null   float64
 6   Sunshine       73982 non-null   float64
 7   WindGustDir    131624 non-null  object
 8   WindGustSpeed  131682 non-null  float64
 9   WindDir9am     131127 non-null  object
 10  WindDir3pm     137117 non-null  object
 11  WindSpeed9am   139732 non-null  float64
 12  WindSpeed3pm   138256 non-null  float64
 13  Humidity9am    139270 non-null  float64
 14  Humidity3pm    137286 non-null  float64
 15  Pressure9am    127044 non-null  float64
 16  Pressure3pm    127018 non-null  float64
 17  Cloud9am       88162 non-null   float64
 18  Cloud3pm       84693 non-null   float64
 19  Temp9am        140131 non-null  float64
 20  Temp3pm        138163 non-null  float64
 21  RainToday      140787 non-null  object
```

```
 22  RainTomorrow   140787 non-null  object
dtypes: float64(16), object(7)
memory usage: 25.8+ MB
```

```
px.histogram(df, x = 'Location', title='Location vs. Rainy Days',
color ="RainToday")
```

```
Data seems to be evely distributed. Cities generally get more rains
than no. Additionally and generally, most cities experience 20% of the
time with rain for the last 10 years.
```

```
px.histogram(df, x = 'RainTomorrow', title='RainTomorrow vs. Rainy
Days', color ="RainToday")
```

More instances of not raining today and not raining tomorrow than other cases.

```
So we can observes that AUS predictor for tomorrow not raining is if
it did not rain yesterday. There seems to be no correlation to raining
today and then raining tomorrow VS NOT raining today and then raining
tomorrow. So this means using the RainToday predictor to whether it
WILL rain tomorrow would be bad since there is no clear distinction
that RainToday values reveals a tendency for it to rain tomorrow.

TLDR: Predictor column 'RainToday' is better at predicting "No" for
'RainTomorrow' than predicting "Yes".
```

```
px.histogram(df, x = 'Temp3pm', title='Temp at 3pm vs. Rainy Days',
color ="RainToday")
```

```
We can see that usually there are more instances of "No" for
"RainTomorrow" than "Yes" for any given temperature at 3PM. Most of
the days experience an average temperature of 15-25 degreees celcius.
It is also around these temperatures do we see the most amount of
rainy days, even tho it is around these temps that we also see the
most amount of non-rainy days. Higher temperatures of 35-45+ almost
guarantee no rain.
```

```
px.scatter(df.sample(5000), x = 'MinTemp', y = 'MaxTemp',
title="Minimum Temperature vs Maximum Temperature as predictor for
tomorrow's rain", color ="RainToday", opacity=0.5)
```

```
Generally, for the days with similar minimum temperatures, a higher
maximum temperature seems to indicate that it will not rain on that
day. The vertical shift of the otherwise similarly positioned scatter
plot of the two predictor columns (with RainToday as indicated) is due
to the fact that the higher maximum temperature is, the more likely to
not rain on that day.
```

Training, Validation, and Test sets

- training 60%
- validation 20%

- test 20%

```
from sklearn.model_selection import train_test_split

train_val_df , test_df = train_test_split(df, test_size=0.2,
random_state=101)
train_df, val_df = train_test_split(train_val_df, test_size=0.25,
random_state=101)
```

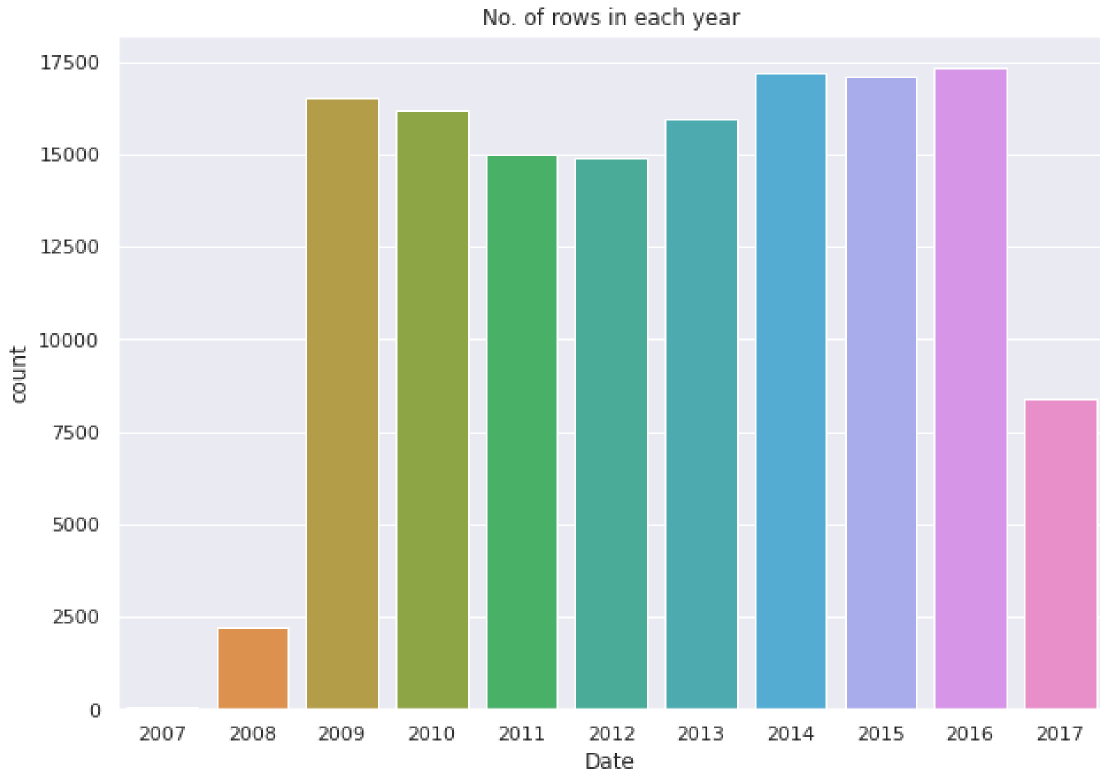
Conclusion:

```
print("shape of train_df is {}".format(train_df.shape))
print("shape of validity_df is {}".format(val_df.shape))
print("shape of test_df is {}".format(test_df.shape))
```

```
shape of train_df is (84471, 23)
shape of validity_df is (28158, 23)
shape of test_df is (28158, 23)
```

Since we are working with data that involves time it is best that we don't train our model with dates that would appear in our validation and test dataset. Meaning one should not train models based on data that is in the future or will be included in the test/validation dataset.

```
%matplotlib inline
fig, ax = plt.subplots(figsize=(10,7))
sns.set_theme(style="darkgrid")
ax = sns.countplot(data = df,x= pd.to_datetime(df["Date"]).dt.year)
ax.set_title("No. of rows in each year");
```



So the data for the test/training/validation will be broken down like this:

1. train_df will contain data points with years below 2015
2. val_df will contain data points with years equal to 2015
3. test_df will contain data points with years greater than 2015

```
year = pd.to_datetime(df["Date"]).dt.year
train_df = df[year < 2015]
val_df = df[year == 2015]
test_df = df[year > 2015]

print("shape of train_df is {}".format(train_df.shape))
print("shape of validity_df is {}".format(val_df.shape))
print("shape of test_df is {}".format(test_df.shape))

shape of train_df is (97988, 23)
shape of validity_df is (17089, 23)
shape of test_df is (25710, 23)

train_inputs = train_df.drop(['Date', 'RainTomorrow'], axis =1)
train_target = train_df["RainTomorrow"]

val_inputs = val_df.drop(['Date', 'RainTomorrow'], axis =1)
val_target = val_df["RainTomorrow"]
test_inputs = test_df.drop(['Date', 'RainTomorrow'], axis =1)
test_target = test_df["RainTomorrow"]
```

```
collect all the columns that are numerical and categorical and put
them into a list

num_cols =
train_inputs.select_dtypes(include=np.number).columns.tolist()
cat_cols =
train_inputs.select_dtypes(include=["object"]).columns.tolist()

train_inputs[num_cols].describe()

             MinTemp       MaxTemp  ...       Temp9am       Temp3pm
count   97674.000000  97801.000000  ...  97414.000000  97392.000000
mean       12.007831     23.022202  ...     16.835126     21.540138
std         6.347175      6.984397  ...      6.404586      6.831612
min        -8.500000     -4.100000  ...     -5.900000     -5.100000
25%         7.500000     17.900000  ...     12.200000     16.600000
50%        11.800000     22.400000  ...     16.600000     20.900000
75%        16.600000     27.900000  ...     21.400000     26.200000
max        33.900000     48.100000  ...     40.200000     46.100000

[8 rows x 16 columns]

train_inputs[cat_cols].nunique()

Location       49
WindGustDir    16
WindDir9am     16
WindDir3pm     16
RainToday       2
dtype: int64
```

Imputing missing data

```
from sklearn.impute import SimpleImputer

#create an imputer object
imputer = SimpleImputer(strategy="mean")

# check the columns that have nan values and how many
train_inputs[num_cols].isnull().sum()

MinTemp            314
MaxTemp            187
Rainfall             0
Evaporation      36331
Sunshine         40046
WindGustSpeed     6828
WindSpeed9am       874
WindSpeed3pm      1069
Humidity9am       1052
Humidity3pm       1116
Pressure9am       9112
```

```
Pressure3pm       9131
Cloud9am         34988
Cloud3pm         36022
Temp9am            574
Temp3pm            596
dtype: int64
```

```
# fit the imputer model to fill each column with missing values the
mean value for that column
imputer.fit(train_inputs[num_cols])
```

```
SimpleImputer()
```

```
# the object imputer now contains an atribute called .statistics_
which contains the mean value for each column. We can access this:
list(imputer.statistics_)
```

```
[12.0078311526097,
 23.0222022269711,
 2.37293546148508,
 5.289991404057933,
 7.609003831417626,
 40.21587318999561,
 14.09226270156723,
 18.76460755888938,
 68.62874473879673,
 51.46954744404988,
 1017.5137337413925,
 1015.132351891241,
 4.302952380952381,
 4.410676822773779,
 16.835126367873205,
 21.54013779365862]
```

```
# we need to inject these values in the predictor variable for all our
datasets.

train_inputs[num_cols] = imputer.fit_transform(train_inputs[num_cols])
val_inputs[num_cols] = imputer.fit_transform(val_inputs[num_cols])
test_inputs[num_cols] = imputer.fit_transform(test_inputs[num_cols])
```

Normalizing our Numerical columns

```
from sklearn.preprocessing import MinMaxScaler

#create an object for MinMaxScaler
scaler = MinMaxScaler()

#scaler.transform({data fram with num cols}) will result in the
scaling of the values from (0,1)

train_inputs[num_cols] = scaler.fit_transform(train_inputs[num_cols])
```

```
val_inputs[num_cols] = scaler.fit_transform(val_inputs[num_cols])
test_inputs[num_cols] = scaler.fit_transform(test_inputs[num_cols])

# verify that the scaling worked (val_inputs is used to see the max
and min but train_inputs and test_inputs both have a max of 1 and min
of 0)
val_inputs[num_cols].describe().loc[["min","max"]]
```

```
     MinTemp  MaxTemp  Rainfall  ...  Cloud3pm  Temp9am  Temp3pm
min      0.0      0.0       0.0  ...       0.0      0.0      0.0
max      1.0      1.0       1.0  ...       1.0      1.0      1.0

[2 rows x 16 columns]
```

Encoding Categorical Data

```
# check all of our categorical columns
train_inputs[cat_cols].nunique()
```

```
Location       49
WindGustDir    16
WindDir9am     16
WindDir3pm     16
RainToday       2
dtype: int64
```

```
train_inputs[cat_cols].isnull().sum()
```

```
Location          0
WindGustDir    6868
WindDir9am     7019
WindDir3pm     1952
RainToday         0
dtype: int64
```

```
from sklearn.preprocessing import OneHotEncoder

# encoder object
encoder = OneHotEncoder(sparse = False, handle_unknown='ignore')
```

Try to impute missing categorical values using logistic regression

```
# Seperate the test, train, and val data into two dataframes: one w/
missing data and one w/o.
```

Try to impute missing categorical values by inputting "Unknown"

```
test_inputs.fillna("Unknown", inplace = True)
train_inputs.fillna("Unknown", inplace = True)
val_inputs.fillna("Unknown", inplace = True)

#check to see if imputation works (val_df2 is shown here but test_df2
and train_df2 also are modified)
train_inputs
```

```
       Location   MinTemp   MaxTemp  ...   Temp9am   Temp3pm
RainToday
0         Albury  0.516509  0.517241  ...  0.494577  0.525391
No
1         Albury  0.375000  0.559387  ...  0.501085  0.574219
No
2         Albury  0.504717  0.570881  ...  0.583514  0.552734
No
3         Albury  0.417453  0.614943  ...  0.520607  0.617188
No
4         Albury  0.613208  0.697318  ...  0.514100  0.679688
No
...          ...       ...       ...  ...       ...       ...        ..
.
144548    Uluru  0.599057  0.714559  ...  0.642082  0.720703
No
144549    Uluru  0.556604  0.783525  ...  0.754881  0.779297
No
144550    Uluru  0.608491  0.802682  ...  0.772234  0.796875
No
144551    Uluru  0.674528  0.816092  ...  0.774403  0.826172
No
144552    Uluru  0.731132  0.837165  ...  0.780911  0.830078
No

[97988 rows x 21 columns]
```

```
# We must first fit our encoder object with our dataframe so it can
recognize the categorical columns
encoder.fit(train_inputs[cat_cols])
```

```
OneHotEncoder(handle_unknown='ignore', sparse=False)
```

```
encoder.categories_
```

```
[array(['Adelaide', 'Albany', 'Albury', 'AliceSprings',
'BadgerysCreek',
        'Ballarat', 'Bendigo', 'Brisbane', 'Cairns', 'Canberra',
'Cobar',
        'CoffsHarbour', 'Dartmoor', 'Darwin', 'GoldCoast', 'Hobart',
        'Katherine', 'Launceston', 'Melbourne', 'MelbourneAirport',
        'Mildura', 'Moree', 'MountGambier', 'MountGinini',
'Newcastle',
        'Nhil', 'NorahHead', 'NorfolkIsland', 'Nuriootpa',
'PearceRAAF',
        'Penrith', 'Perth', 'PerthAirport', 'Portland', 'Richmond',
'Sale',
        'SalmonGums', 'Sydney', 'SydneyAirport', 'Townsville',
        'Tuggeranong', 'Uluru', 'WaggaWagga', 'Walpole', 'Watsonia',
        'Williamtown', 'Witchcliffe', 'Wollongong', 'Woomera'],
       dtype=object),
```

```
 array(['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE',
'SSE',
        'SSW', 'SW', 'Unknown', 'W', 'WNW', 'WSW'], dtype=object),
 array(['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE',
'SSE',
        'SSW', 'SW', 'Unknown', 'W', 'WNW', 'WSW'], dtype=object),
 array(['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE',
'SSE',
        'SSW', 'SW', 'Unknown', 'W', 'WNW', 'WSW'], dtype=object),
 array(['No', 'Yes'], dtype=object)]
```

```
# generate columns names to label our new one-hot encoded columns
encoded_cols = list(encoder.get_feature_names(cat_cols))

print(encoded_cols)
```

```
['Location_Adelaide', 'Location_Albany', 'Location_Albury',
'Location_AliceSprings', 'Location_BadgerysCreek',
'Location_Ballarat', 'Location_Bendigo', 'Location_Brisbane',
'Location_Cairns', 'Location_Canberra', 'Location_Cobar',
'Location_CoffsHarbour', 'Location_Dartmoor', 'Location_Darwin',
'Location_GoldCoast', 'Location_Hobart', 'Location_Katherine',
'Location_Launceston', 'Location_Melbourne',
'Location_MelbourneAirport', 'Location_Mildura', 'Location_Moree',
'Location_MountGambier', 'Location_MountGinini', 'Location_Newcastle',
'Location_Nhil', 'Location_NorahHead', 'Location_NorfolkIsland',
'Location_Nuriootpa', 'Location_PearceRAAF', 'Location_Penrith',
'Location_Perth', 'Location_PerthAirport', 'Location_Portland',
'Location_Richmond', 'Location_Sale', 'Location_SalmonGums',
'Location_Sydney', 'Location_SydneyAirport', 'Location_Townsville',
'Location_Tuggeranong', 'Location_Uluru', 'Location_WaggaWagga',
'Location_Walpole', 'Location_Watsonia', 'Location_Williamtown',
'Location_Witchcliffe', 'Location_Wollongong', 'Location_Woomera',
'WindGustDir_E', 'WindGustDir_ENE', 'WindGustDir_ESE',
'WindGustDir_N', 'WindGustDir_NE', 'WindGustDir_NNE',
'WindGustDir_NNW', 'WindGustDir_NW', 'WindGustDir_S',
'WindGustDir_SE', 'WindGustDir_SSE', 'WindGustDir_SSW',
'WindGustDir_SW', 'WindGustDir_Unknown', 'WindGustDir_W',
'WindGustDir_WNW', 'WindGustDir_WSW', 'WindDir9am_E',
'WindDir9am_ENE', 'WindDir9am_ESE', 'WindDir9am_N', 'WindDir9am_NE',
'WindDir9am_NNE', 'WindDir9am_NNW', 'WindDir9am_NW', 'WindDir9am_S',
'WindDir9am_SE', 'WindDir9am_SSE', 'WindDir9am_SSW', 'WindDir9am_SW',
'WindDir9am_Unknown', 'WindDir9am_W', 'WindDir9am_WNW',
'WindDir9am_WSW', 'WindDir3pm_E', 'WindDir3pm_ENE', 'WindDir3pm_ESE',
'WindDir3pm_N', 'WindDir3pm_NE', 'WindDir3pm_NNE', 'WindDir3pm_NNW',
'WindDir3pm_NW', 'WindDir3pm_S', 'WindDir3pm_SE', 'WindDir3pm_SSE',
'WindDir3pm_SSW', 'WindDir3pm_SW', 'WindDir3pm_Unknown',
'WindDir3pm_W', 'WindDir3pm_WNW', 'WindDir3pm_WSW', 'RainToday_No',
'RainToday_Yes']
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/
deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is
deprecated in 1.0 and will be removed in 1.2. Please use
get_feature_names_out instead.
```

```
# Now that we have generated the one-hot encoded columns, we shall
append them to our train, validation, and test datasets
train_inputs[encoded_cols] = encoder.fit_transform(train_df[cat_cols])
test_inputs[encoded_cols] = encoder.fit_transform(test_df[cat_cols])
val_inputs[encoded_cols] = encoder.fit_transform(val_df[cat_cols])

# Delete redundant columns such as the categori columns such as the
ones we used the one-encoded on
train_inputs.drop(cat_cols, axis = 1, inplace = True)
test_inputs.drop(cat_cols, axis = 1, inplace = True)
val_inputs.drop(cat_cols, axis = 1, inplace = True)

#same outputs for the train and validation datasets.
test_inputs.columns.tolist()
```

```
['MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm',
 'Location_Adelaide',
 'Location_Albany',
 'Location_Albury',
 'Location_AliceSprings',
 'Location_BadgerysCreek',
 'Location_Ballarat',
 'Location_Bendigo',
 'Location_Brisbane',
 'Location_Cairns',
 'Location_Canberra',
 'Location_Cobar',
 'Location_CoffsHarbour',
```

```
'Location_Dartmoor',
'Location_Darwin',
'Location_GoldCoast',
'Location_Hobart',
'Location_Katherine',
'Location_Launceston',
'Location_Melbourne',
'Location_MelbourneAirport',
'Location_Mildura',
'Location_Moree',
'Location_MountGambier',
'Location_MountGinini',
'Location_Newcastle',
'Location_Nhil',
'Location_NorahHead',
'Location_NorfolkIsland',
'Location_Nuriootpa',
'Location_PearceRAAF',
'Location_Penrith',
'Location_Perth',
'Location_PerthAirport',
'Location_Portland',
'Location_Richmond',
'Location_Sale',
'Location_SalmonGums',
'Location_Sydney',
'Location_SydneyAirport',
'Location_Townsville',
'Location_Tuggeranong',
'Location_Uluru',
'Location_WaggaWagga',
'Location_Walpole',
'Location_Watsonia',
'Location_Williamtown',
'Location_Witchcliffe',
'Location_Wollongong',
'Location_Woomera',
'WindGustDir_E',
'WindGustDir_ENE',
'WindGustDir_ESE',
'WindGustDir_N',
'WindGustDir_NE',
'WindGustDir_NNE',
'WindGustDir_NNW',
'WindGustDir_NW',
'WindGustDir_S',
'WindGustDir_SE',
'WindGustDir_SSE',
'WindGustDir_SSW',
'WindGustDir_SW',
```

```
 'WindGustDir_Unknown',
 'WindGustDir_W',
 'WindGustDir_WNW',
 'WindGustDir_WSW',
 'WindDir9am_E',
 'WindDir9am_ENE',
 'WindDir9am_ESE',
 'WindDir9am_N',
 'WindDir9am_NE',
 'WindDir9am_NNE',
 'WindDir9am_NNW',
 'WindDir9am_NW',
 'WindDir9am_S',
 'WindDir9am_SE',
 'WindDir9am_SSE',
 'WindDir9am_SSW',
 'WindDir9am_SW',
 'WindDir9am_Unknown',
 'WindDir9am_W',
 'WindDir9am_WNW',
 'WindDir9am_WSW',
 'WindDir3pm_E',
 'WindDir3pm_ENE',
 'WindDir3pm_ESE',
 'WindDir3pm_N',
 'WindDir3pm_NE',
 'WindDir3pm_NNE',
 'WindDir3pm_NNW',
 'WindDir3pm_NW',
 'WindDir3pm_S',
 'WindDir3pm_SE',
 'WindDir3pm_SSE',
 'WindDir3pm_SSW',
 'WindDir3pm_SW',
 'WindDir3pm_Unknown',
 'WindDir3pm_W',
 'WindDir3pm_WNW',
 'WindDir3pm_WSW',
 'RainToday_No',
 'RainToday_Yes']
```

```python
print("train_inputs shape is {}".format(train_inputs.shape))
print("train_target shape is {}".format(train_target.shape))
print("val_inputs shape is {}".format(val_inputs.shape))
print("val_target shape is {}".format(val_target.shape))
print("test_inputs shape is {}".format(test_inputs.shape))
print("test_target shape is {}".format(test_target.shape))
```

```
train_inputs shape is (97988, 118)
train_target shape is (97988,)
val_inputs shape is (17089, 118)
```

```
val_target shape is (17089,)
test_inputs shape is (25710, 118)
test_target shape is (25710,)
```

Train the training set using Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='liblinear')

model.fit(train_inputs, train_target)

LogisticRegression(solver='liblinear')

weight_df = pd.DataFrame(
    {"feature": train_inputs.columns.tolist(),
    "weight": model.coef_.tolist()[0]}
)
weight_df

            feature    weight
0           MinTemp  0.894303
1           MaxTemp -2.854273
2          Rainfall  3.162893
3       Evaporation  0.640867
4          Sunshine -1.649886
..              ...       ...
113    WindDir3pm_W -0.043347
114  WindDir3pm_WNW -0.285430
115  WindDir3pm_WSW  0.068019
116    RainToday_No -1.559546
117   RainToday_Yes -1.060584

[118 rows x 2 columns]

import plotly.graph_objects as go

neg_weights = weight_df[weight_df["weight"] < 0].sort_values("weight",
ascending = False)
pos_weights = weight_df[weight_df["weight"] > 0].sort_values("weight",
ascending = False)

fig = go.Figure()
fig.add_trace(go.Bar(x=neg_weights.feature, y=neg_weights.weight,
                marker_color='crimson',
                name='negative weights'))
fig.add_trace(go.Bar(x=pos_weights.feature, y=pos_weights.weight,
                marker_color='lightslategrey',
                name='positive weights'
                ))
#fig.update_xaxes(visible = False)
fig.update_xaxes(nticks = 100)
```

```
fig.update_yaxes(nticks=20, ticks = "outside")
fig.update_layout(
    margin=dict(l=20, r=20, t=20, b=20),
    paper_bgcolor="LightSteelBlue"
)
fig.show()

'''
press on the legends to see/remove the negative and positive weights.
Negative Weights is turned off currently. press the negative weights
legend to show the negative weights
'''
```

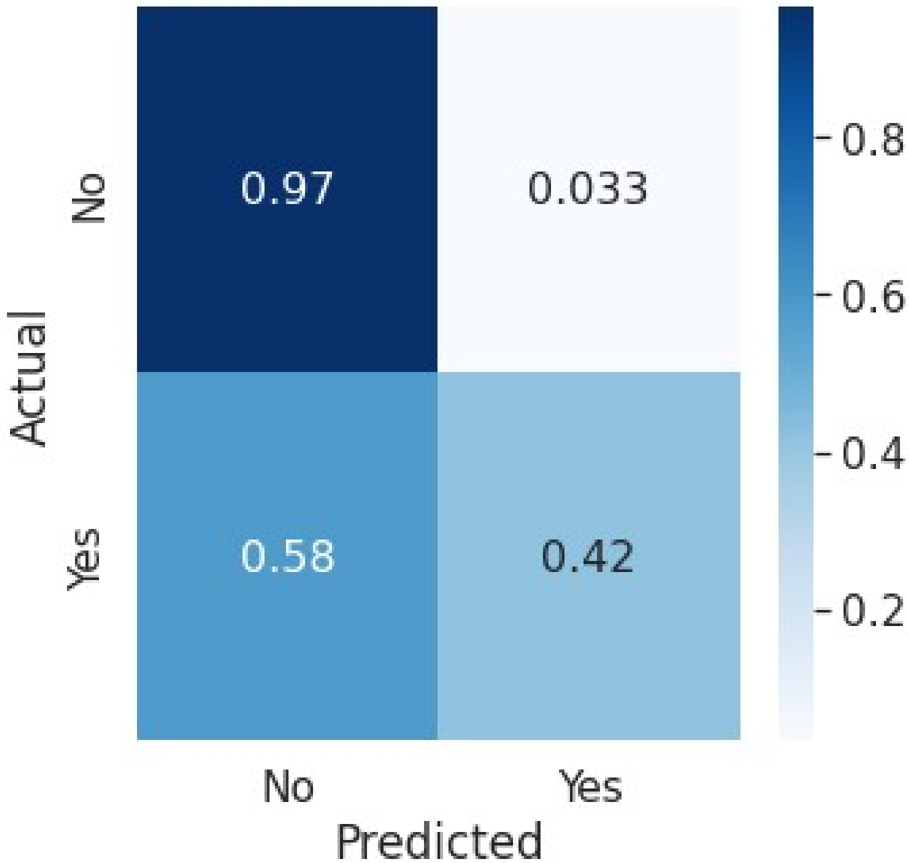
{"type":"string"}

```
generally the higher the coefficient the more the feature is able to
accurately predict the target variable. A high negative coefficient
value means that the relationship is inverse, but strong.
```

Making Predictions on the validation set

```
val_predictions = model.predict(val_inputs)
val_predictions.tolist()

# check the accuracy of the train_target and train_predictions
from sklearn.metrics import accuracy_score
print("accuracy is: {}".format(accuracy_score(val_target,
val_predictions)))

accuracy is: 0.8516004447305284

# check the prediction probabilities for the observations
val_proba = model.predict_proba(val_inputs)
print(model.classes_)
print(val_proba)

['No' 'Yes']
[[0.98930488 0.01069512]
 [0.96371828 0.03628172]
 [0.94248961 0.05751039]
 ...
 [0.97879253 0.02120747]
 [0.9711122  0.0288878 ]
 [0.94725931 0.05274069]]

from sklearn.metrics import confusion_matrix
confusion_matrix(val_target, val_predictions, normalize = 'true')

array([[0.96661979, 0.03338021],
       [0.58272778, 0.41727222]])

data = confusion_matrix(val_target, val_predictions, normalize =
'true')
```

```
df_cm = pd.DataFrame(data, columns=np.unique(val_predictions), index =
np.unique(val_target))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



as one can see the confusion matrix reveals that the model is better at predicting whether it will not rain tomorrow than it can predict whether it will rain tomorrow.

```
from sklearn.metrics import classification_report
print(classification_report(val_target, val_predictions))
```

```
              precision    recall  f1-score   support

          No       0.86      0.97      0.91     13511
         Yes       0.77      0.42      0.54      3578

    accuracy                           0.85     17089
   macro avg       0.82      0.69      0.73     17089
```

```
weighted avg       0.84      0.85      0.83     17089
```

first attempt at optimizing model by changing the hyperparameters of the logistic regression model

```
param_grid = [
    {'penalty':['l1', 'l2','elasticnet','none'],
     'solver':['liblinear','newton-cg','lbfgs','sag','saga'],
     'max_iter': [100,1000,1500,2000]
    }
]

from sklearn.model_selection import GridSearchCV

logmodel = LogisticRegression()

clf = GridSearchCV(logmodel, param_grid, cv=2, verbose=True, n_jobs=-
1)

best_clf = clf.fit(train_inputs, train_target)
```

```
Fitting 2 folds for each of 80 candidates, totalling 160 fits

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/
_validation.py:372: FitFailedWarning:


72 fits failed out of a total of 160.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
----------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties,
```

```
got l1 penalty.

--------------------------------------------------------------------------------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

--------------------------------------------------------------------------------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1
penalty.

--------------------------------------------------------------------------------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
```

```
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 459, in _check_solver
    solver
ValueError: Only 'saga' solver supports elasticnet penalty, got
solver=liblinear.

----------------------------------------------------------------------
----------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties,
got elasticnet penalty.

----------------------------------------------------------------------
----------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

----------------------------------------------------------------------
----------
8 fits failed with the following error:
```

```
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver sag supports only 'l2' or 'none' penalties, got
elasticnet penalty.

----------------------------------------------------------------------
----------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1473, in fit
    % self.l1_ratio
ValueError: l1_ratio must be between 0 and 1; got (l1_ratio=None)

----------------------------------------------------------------------
----------
8 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 464, in _check_solver
    raise ValueError("penalty='none' is not supported for the
liblinear solver")
ValueError: penalty='none' is not supported for the liblinear solver
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search
.py:972: UserWarning:

One or more of the test scores are non-finite: [0.821968          nan
nan        nan 0.82289668 0.82062089
 0.822366   0.82195779 0.8223558  0.82234559        nan        nan
        nan        nan        nan        nan 0.78861697 0.81428338
 0.80577214 0.81313018 0.82187615        nan        nan        nan
 0.82289668 0.82062089 0.822366   0.82232518 0.822366   0.82234559
        nan        nan        nan        nan        nan        nan
 0.78861697 0.80469037 0.79499531 0.80854799 0.82166184        nan
        nan        nan 0.82289668 0.82062089 0.822366   0.82232518
 0.8223558  0.82233539        nan        nan        nan        nan
        nan        nan 0.78861697 0.80469037 0.80569049 0.80853778
 0.82174348        nan        nan        nan 0.82289668 0.82062089
 0.822366   0.82232518 0.82237621 0.82234559        nan        nan
        nan        nan        nan        nan 0.78861697 0.80469037
 0.80542515 0.80854799]
```

```
best_clf.best_params_
```

```
{'max_iter': 100, 'penalty': 'l1', 'solver': 'saga'}
```

```
print("previous accuracy is: {}".format(accuracy_score(val_target,
val_predictions)))
logmodel_optimized = LogisticRegression(max_iter = 100,penalty='l1',
solver='saga').fit(train_inputs, train_target)
```

```
previous accuracy is: 0.8516004447305284
```

```
val_predictions_optimized = logmodel_optimized.predict(val_inputs)
print("new accuracy is: {}".format(accuracy_score(val_target,
val_predictions_optimized)))
```

```
new accuracy is: 0.85171747908011
```

```
print("difference in accruacy scores are:
{}".format(accuracy_score(val_target, val_predictions_optimized) -
accuracy_score(val_target, val_predictions)))
```

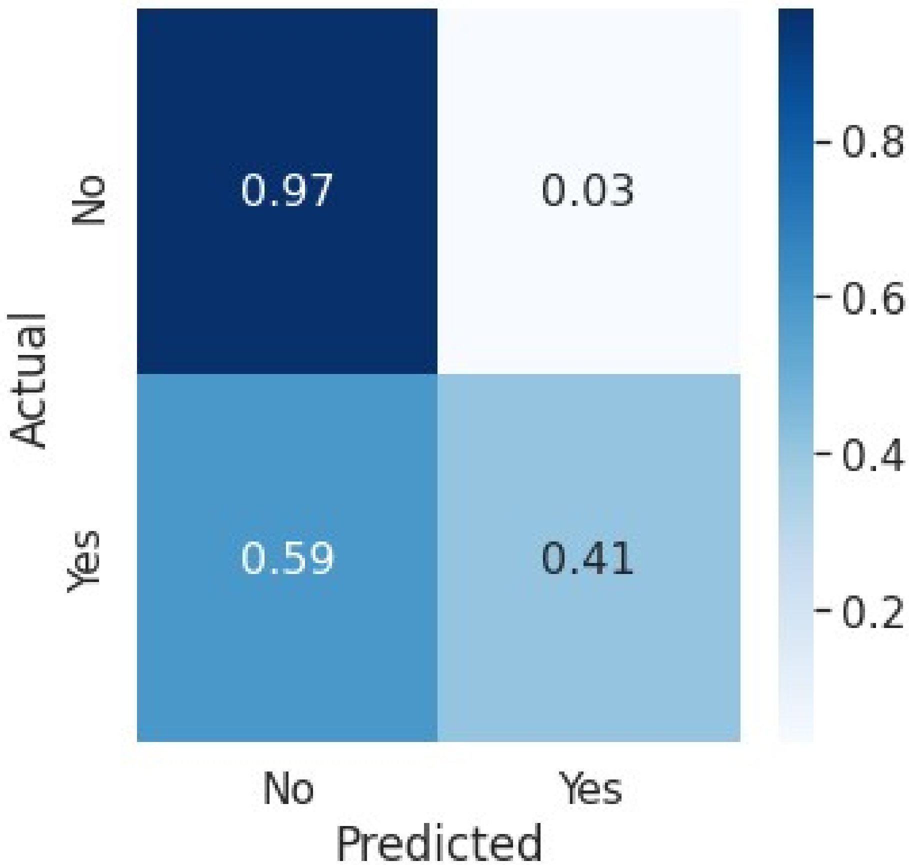
```
difference in accruacy scores are: 0.00011703434958154624
```

The logistical regression model was optimized by changing the hyperparameters. However, after several fits using the hyperparameter combinations of param_grid, the accuracy score did not change that much. There is an improvement of 0.00011703434958154624.

Now we wil check the confusion matrix

```
data = confusion_matrix(val_target, val_predictions_optimized,
normalize = 'true')
df_cm = pd.DataFrame(data,
```

```
columns=np.unique(val_predictions_optimized), index =
np.unique(val_target))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



looks like the model's ability to predict certainty of actually raining decreased.

Evaluate the FINAL model (using test dataset) using classification report, accuracy and confusion matrix

```
test_predictions = logmodel_optimized.predict(test_inputs)
```

```
print(classification_report(test_target,test_predictions))
```

```
              precision    recall  f1-score   support

          No       0.85      0.96      0.90     19885
         Yes       0.77      0.43      0.55      5825

    accuracy                           0.84     25710
   macro avg       0.81      0.70      0.73     25710
```

```
weighted avg       0.83      0.84      0.82     25710
```

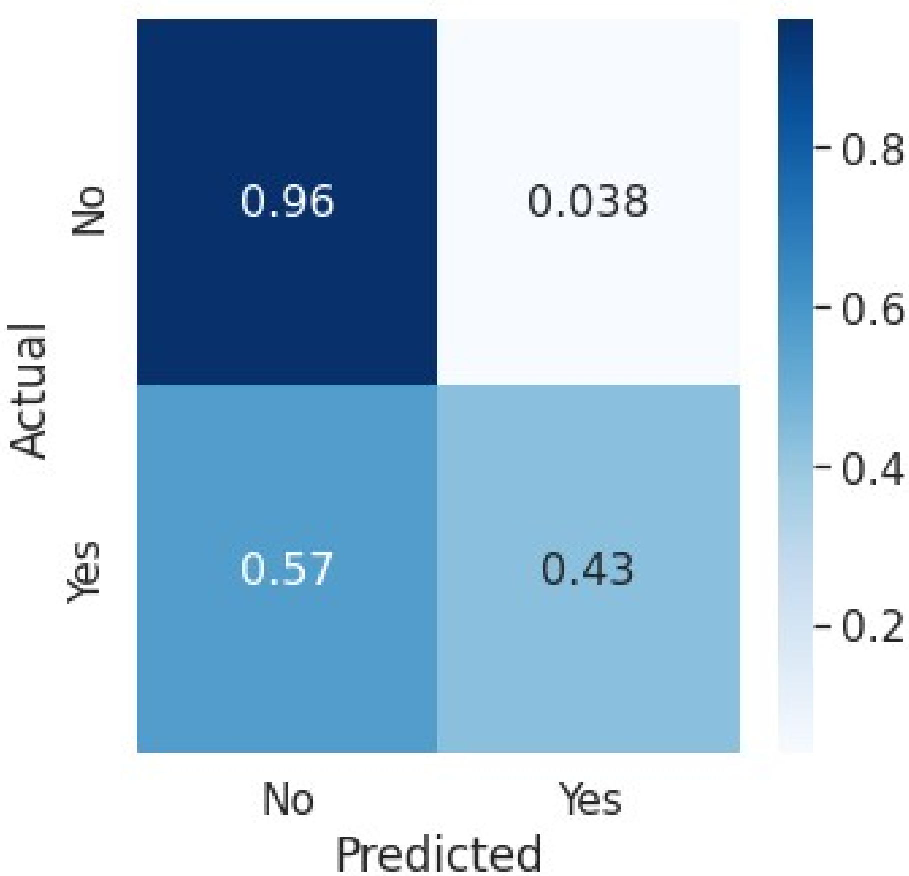
```
print(accuracy_score(test_target, test_predictions))
```

```
0.841267989109296
```

```
print(confusion_matrix(test_target, test_predictions, normalize =
'true'))
```

```
[[0.96208197 0.03791803]
 [0.5711588  0.4288412 ]]
```

```
data = confusion_matrix(test_target, test_predictions, normalize =
'true')
df_cm = pd.DataFrame(data, columns=np.unique(test_predictions), index
= np.unique(test_target))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



type your conclusion here

In conclusion the resulting accuracy score, confusion matrix, and classification report are undesribale in my opninion. I have initially tried to optimize the model by increasing the combinations of the hyperparameters; however, the processing of determingin which combination was the best took so much time. When the process was not done until 3 hours, I repeated the entire process again using lesser parameters. I went from 4800 fits to 160 fits using GridSearchCV.

According to the confusion matrix, the model is better at predicting No rain than it is at predicting that it would rain. In fact, if the model were used in weather forcasts and it outputs a value of "Yes", people are slightly likely to see it not rain as opposed to not raining. People should be more confident that it would not rain if the model tells them it would not rain.