

# [Grupa 3] Praca Domowa nr3 Kacper Kurowski

April 19, 2021

## 1 [WUM] PD3

### 1.1 Kacper Kurowski

Wpierw wczytajmy dane

```
[1]: import os
      os.getcwd()
```

```
[1]: '/home/kurowskik'
```

```
[2]: import numpy as np
      import pandas as pd

      import seaborn as sns
      sns.set_theme(style="darkgrid")

      import matplotlib.pyplot as plt

      import warnings
      warnings.filterwarnings("ignore")
      import requests
```

```
[3]: aus_weather = pd.read_csv( "/home/kurowskik/kaggle/weatherAUS.csv", sep = ",",
      ↪header=0)
```

```
[51]: aus_weather
```

```
[51]:
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	\
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	
...	...	...	...	...	...		
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	

145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN

	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	\
0	NaN	W	44.0	W	...	71.0	
1	NaN	WNW	44.0	NNW	...	44.0	
2	NaN	WSW	46.0	W	...	38.0	
3	NaN	NE	24.0	SE	...	45.0	
4	NaN	W	41.0	ENE	...	82.0	
...	...	...	...	...	...	...	
145455	NaN	E	31.0	SE	...	51.0	
145456	NaN	NNW	22.0	SE	...	56.0	
145457	NaN	N	37.0	SE	...	53.0	
145458	NaN	SE	28.0	SSE	...	51.0	
145459	NaN	NaN	NaN	ESE	...	62.0	

	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	\
0	22.0	1007.7	1007.1	8.0	NaN	16.9	
1	25.0	1010.6	1007.8	NaN	NaN	17.2	
2	30.0	1007.6	1008.7	NaN	2.0	21.0	
3	16.0	1017.6	1012.8	NaN	NaN	18.1	
4	33.0	1010.8	1006.0	7.0	8.0	17.8	
...	...	...	...	...	...	...	
145455	24.0	1024.6	1020.3	NaN	NaN	10.1	
145456	21.0	1023.5	1019.1	NaN	NaN	10.9	
145457	24.0	1021.0	1016.8	NaN	NaN	12.5	
145458	24.0	1019.4	1016.5	3.0	2.0	15.1	
145459	36.0	1020.2	1017.9	8.0	8.0	15.0	

	Temp3pm	RainToday	RainTomorrow
0	21.8	No	No
1	24.3	No	No
2	23.2	No	No
3	26.5	No	No
4	29.7	No	No
...	...	...	...
145455	22.4	No	No
145456	24.5	No	No
145457	26.1	No	No
145458	26.0	No	No
145459	20.9	No	NaN

[145460 rows x 23 columns]

Możemy szybko zapoznać się z danymi

```
[7]: from pandas_profiling import ProfileReport
```

```
[33]: profile = ProfileReport(aus_weather, title="Pandas Profiling Report")
```

```
[1]: #profile.to_notebook_iframe() #Jak chcemy zobaczyć
```

Możemy zauważyć, że zmienne Date, Location, WindGustDir, WindDir9am, WindDir3pm, RainToday i RainTomorrow mają wartości nieliczbowe. Dlatego postaramy się zakodować je przy pomocy liczb. Możemy również zauważyć, że jest dużo wierszy z Evaporation i Sunshine na NaN. Podobnie Cloud9am i Cloud3pm. Z tego powodu usuniemy te kolumny.

```
[4]: del aus_weather["Evaporation"]
del aus_weather["Sunshine"]
del aus_weather["Cloud9am"]
del aus_weather["Cloud3pm"]
```

```
[5]: direction_to_encoding = {
    "N" : [1.0,0.0,0.0,0.0],
    "NNW" : [0.75,0.25,0.0,0.0],
    "NW" : [0.5,0.5,0.0,0.0],
    "WNW" : [0.25,0.66,0.0,0.0],
    "W" : [0.0,1.0,0.0,0.0],
    "WSW" : [0.0,0.75,0.25,0.0],
    "SW" : [0.0,0.5,0.5,0.0],
    "SSW" : [0.0,0.75,0.66,0.0],
    "S" : [0.0,0.0,1.0,0.0],
    "SSE" : [0.0,0.0,0.75,0.25],
    "SE" : [0.0,0.0,0.5,0.5],
    "ESE" : [0.0,0.0,0.25,0.75],
    "E" : [0.0,0.0,0.0,1.0],
    "ENE" : [0.25,0.0,0.0,0.75],
    "NE" : [0.5,0.0,0.0,0.5],
    "NNE" : [0.75,0.66,0.0,0.25],
    "nan" : [0.0,0.0,0.0,0.0]
}
```

```
[6]: GustDir = pd.DataFrame(
    aus_weather["WindGustDir"].fillna("nan").map(direction_to_encoding).
    ↪tolist(),
    columns=['WindGustDirN','WindGustDirW','WindGustDirS','WindGustDirE'],
    index = aus_weather.index)
aus_weather = aus_weather.merge(GustDir, left_index=True, right_index=True)

GustDir9am = pd.DataFrame(
    aus_weather["WindDir9am"].fillna("nan").map(direction_to_encoding).
    ↪tolist(),
    columns=['WindDir9amN','WindDir9amW','WindDir9amS','WindDir9amE'],
    index = aus_weather.index)

aus_weather = aus_weather.merge(GustDir9am, left_index=True, right_index=True)
```

```
GustDir3pm = pd.DataFrame(
    aus_wheather["WindDir3pm"].fillna("nan").map(direction_to_encoding).
    ↪ tolist(),
    columns=['WindDir3pmN', 'WindDir3pmW', 'WindDir3pmS', 'WindDir3pmE'],
    index = aus_wheather.index)
aus_wheather = aus_wheather.merge(GustDir3pm, left_index=True, right_index=True)
```

```
[7]: def encode_dates(x):
      tmp = x.split("-")
      return [float( tmp[0]), float(tmp[1]), float(tmp[2]) ]
```

```
[8]: dates = pd.DataFrame(
    aus_wheather['Date'].map( encode_dates).tolist(),
    columns=["Year", "Month", "Day"],
    index = aus_wheather.index)
aus_wheather = aus_wheather.merge(dates, left_index=True, right_index=True)
```

```
[9]: def encodeRain(x):
      if x == "Yes":
          return 1
      elif x == "No":
          return 0
```

```
[10]: aus_wheather['RainTomorrow'] = aus_wheather['RainTomorrow'].map( encodeRain)
      aus_wheather['RainToday'] = aus_wheather['RainToday'].map( encodeRain)
```

```
[11]: aus_wheather.columns
```

```
[11]: Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustDir',
            'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am',
            'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am',
            'Pressure3pm', 'Temp9am', 'Temp3pm', 'RainToday', 'RainTomorrow',
            'WindGustDirN', 'WindGustDirW', 'WindGustDirS', 'WindGustDirE',
            'WindDir9amN', 'WindDir9amW', 'WindDir9amS', 'WindDir9amE',
            'WindDir3pmN', 'WindDir3pmW', 'WindDir3pmS', 'WindDir3pmE', 'Year',
            'Month', 'Day'],
            dtype='object')
```

```
[12]: tmp = aus_wheather['Location'].map( lambda x: sum(bytearray(x, 'utf-8'))+len(x))
      ↪ # Kodujemy lokację, niestety nieróżnowartościowo
```

```
[13]: print( len(pd.unique(aus_wheather['Location'])))
      print( len(pd.unique(tmp))) # Niestety kodowanie nie jest różnowartościowe w
      ↪ tym przypadku. Trudno.
```

49

46

```
[14]: aus_weather['Location'] = tmp
```

```
[15]: aus_weather.head()
```

```
[15]:
```

	Date	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	\
0	2008-12-01	629	13.4	22.9	0.6	W	
1	2008-12-02	629	7.4	25.1	0.0	WNW	
2	2008-12-03	629	12.9	25.7	0.0	WSW	
3	2008-12-04	629	9.2	28.0	0.0	NE	
4	2008-12-05	629	17.5	32.3	1.0	W	

	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	...	WindDir9amW	\
0	44.0	W	WNW	20.0	...	1.00	
1	44.0	NNW	WSW	4.0	...	0.25	
2	46.0	W	WSW	19.0	...	1.00	
3	24.0	SE	E	11.0	...	0.00	
4	41.0	ENE	NW	7.0	...	0.00	

	WindDir9amS	WindDir9amE	WindDir3pmN	WindDir3pmW	WindDir3pmS	\
0	0.0	0.00	0.25	0.66	0.00	
1	0.0	0.00	0.00	0.75	0.25	
2	0.0	0.00	0.00	0.75	0.25	
3	0.5	0.50	0.00	0.00	0.00	
4	0.0	0.75	0.50	0.50	0.00	

	WindDir3pmE	Year	Month	Day
0	0.0	2008.0	12.0	1.0
1	0.0	2008.0	12.0	2.0
2	0.0	2008.0	12.0	3.0
3	1.0	2008.0	12.0	4.0
4	0.0	2008.0	12.0	5.0

[5 rows x 34 columns]

Usuwamy kolumny zakodowane (przy pomocy innych kolumn)

```
[15]: del aus_weather["Date"]
del aus_weather["WindGustDir"]
del aus_weather["WindDir9am"]
del aus_weather["WindDir3pm"]
```

```
[16]: aus_weather.head()
```

```
[16]:
```

	Location	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	\
0	629	13.4	22.9	0.6	44.0	20.0	
1	629	7.4	25.1	0.0	44.0	4.0	
2	629	12.9	25.7	0.0	46.0	19.0	
3	629	9.2	28.0	0.0	24.0	11.0	

4	629	17.5	32.3	1.0	41.0	7.0
---	-----	------	------	-----	------	-----

	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	...	WindDir9amW	\
0	24.0	71.0	22.0	1007.7	...	1.00	
1	22.0	44.0	25.0	1010.6	...	0.25	
2	26.0	38.0	30.0	1007.6	...	1.00	
3	9.0	45.0	16.0	1017.6	...	0.00	
4	20.0	82.0	33.0	1010.8	...	0.00	

	WindDir9amS	WindDir9amE	WindDir3pmN	WindDir3pmW	WindDir3pmS	\
0	0.0	0.00	0.25	0.66	0.00	
1	0.0	0.00	0.00	0.75	0.25	
2	0.0	0.00	0.00	0.75	0.25	
3	0.5	0.50	0.00	0.00	0.00	
4	0.0	0.75	0.50	0.50	0.00	

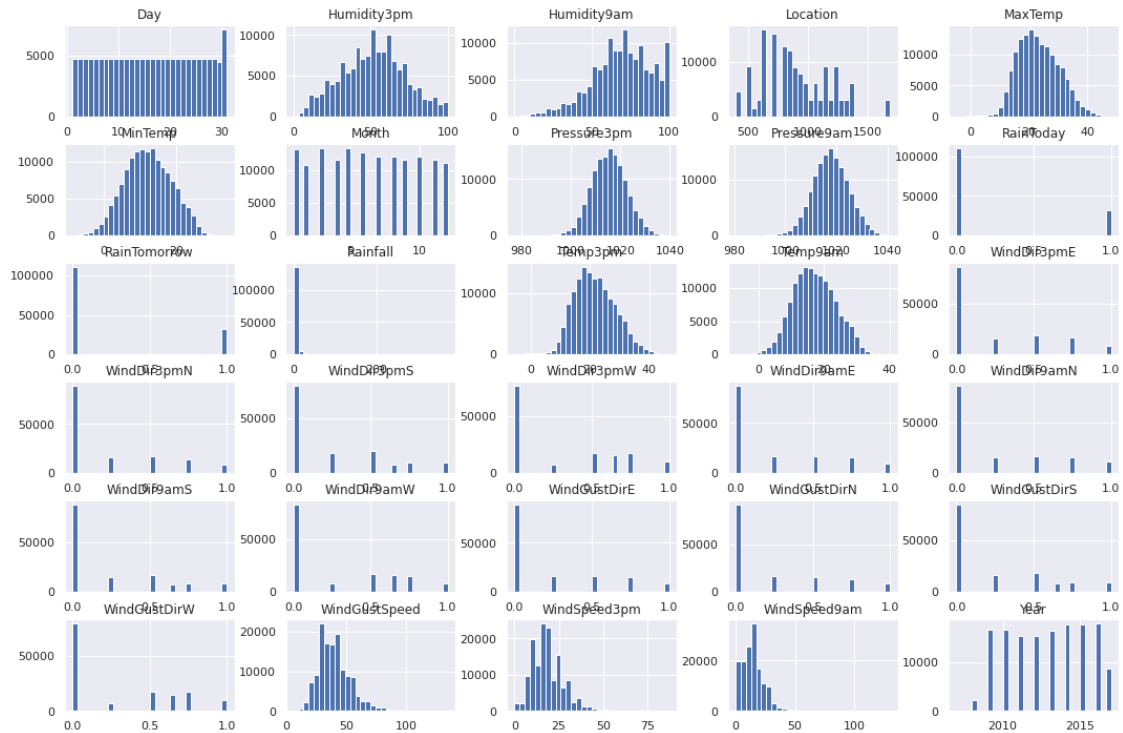
  

	WindDir3pmE	Year	Month	Day
0	0.0	2008.0	12.0	1.0
1	0.0	2008.0	12.0	2.0
2	0.0	2008.0	12.0	3.0
3	1.0	2008.0	12.0	4.0
4	0.0	2008.0	12.0	5.0

[5 rows x 30 columns]

Możemy również popatrzeć na wykresy samodzielnie (może coś się uda zauważyć)

```
[25]: aus_weather.hist(figsize=(18, 12), bins=30)
plt.show()
```



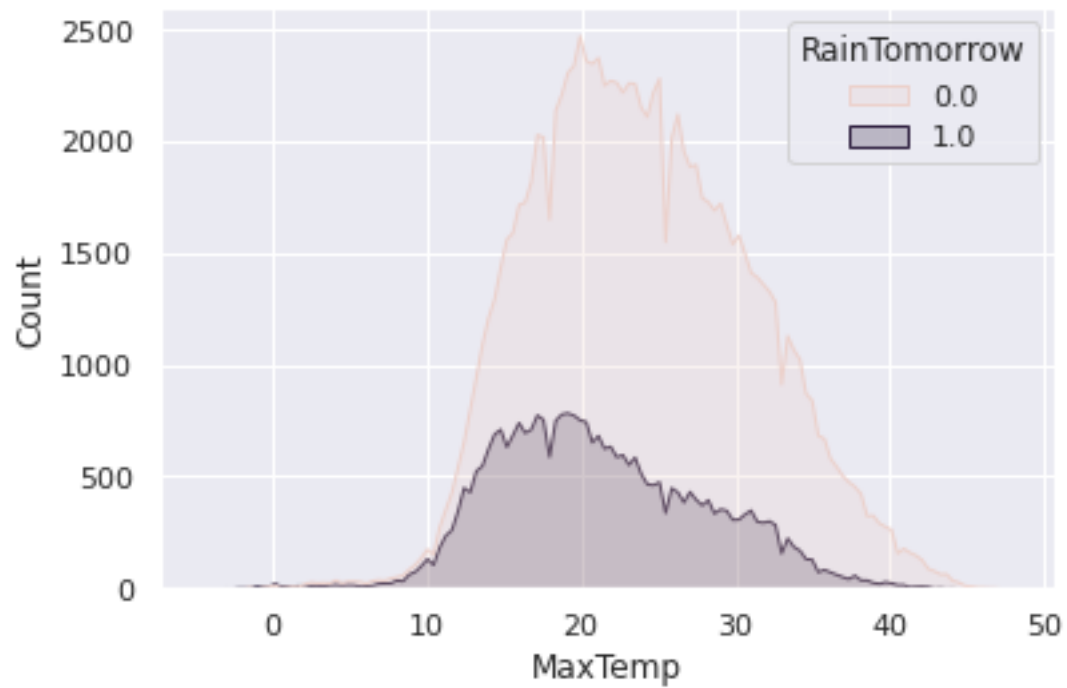
```
[26]: import seaborn as sns
```

```
[17]: aus_weather.columns
```

```
[17]: Index(['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed',
          'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
          'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm', 'RainToday',
          'RainTomorrow', 'WindGustDirN', 'WindGustDirW', 'WindGustDirS',
          'WindGustDirE', 'WindDir9amN', 'WindDir9amW', 'WindDir9amS',
          'WindDir9amE', 'WindDir3pmN', 'WindDir3pmW', 'WindDir3pmS',
          'WindDir3pmE', 'Year', 'Month', 'Day'],
          dtype='object')
```

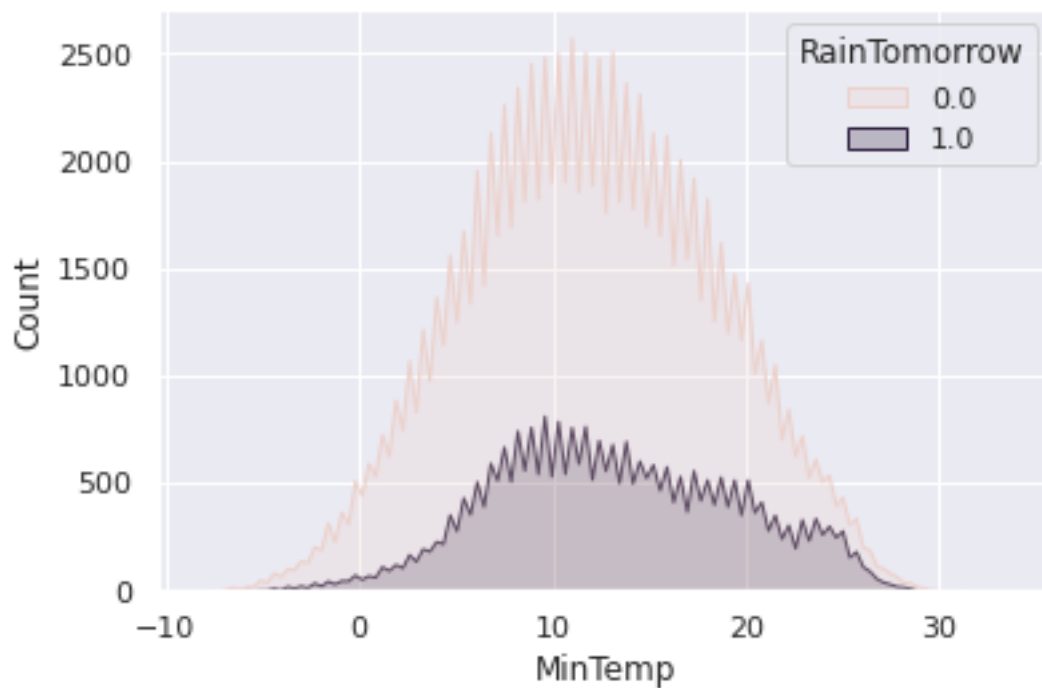
```
[18]: sns.histplot(data=aus_weather, x="MaxTemp", hue="RainTomorrow", element="poly")
```

```
[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8d22f0dca0>
```



```
[133]: sns.histplot(data=aus_weather, x="MinTemp", hue="RainTomorrow", element="poly")
```

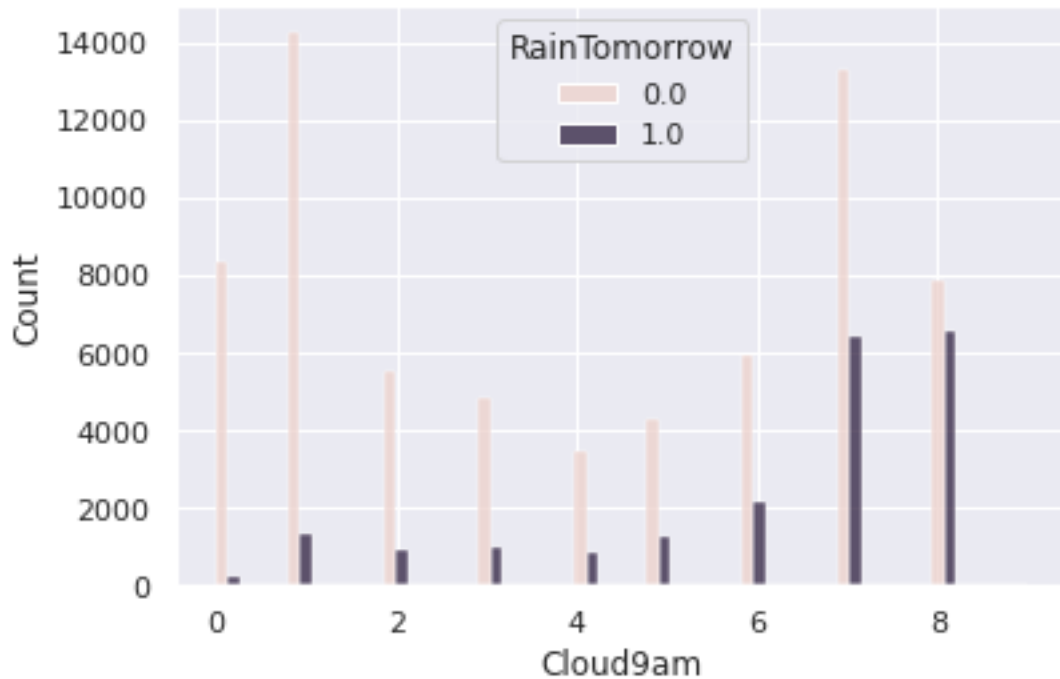
```
[133]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd005c48130>
```





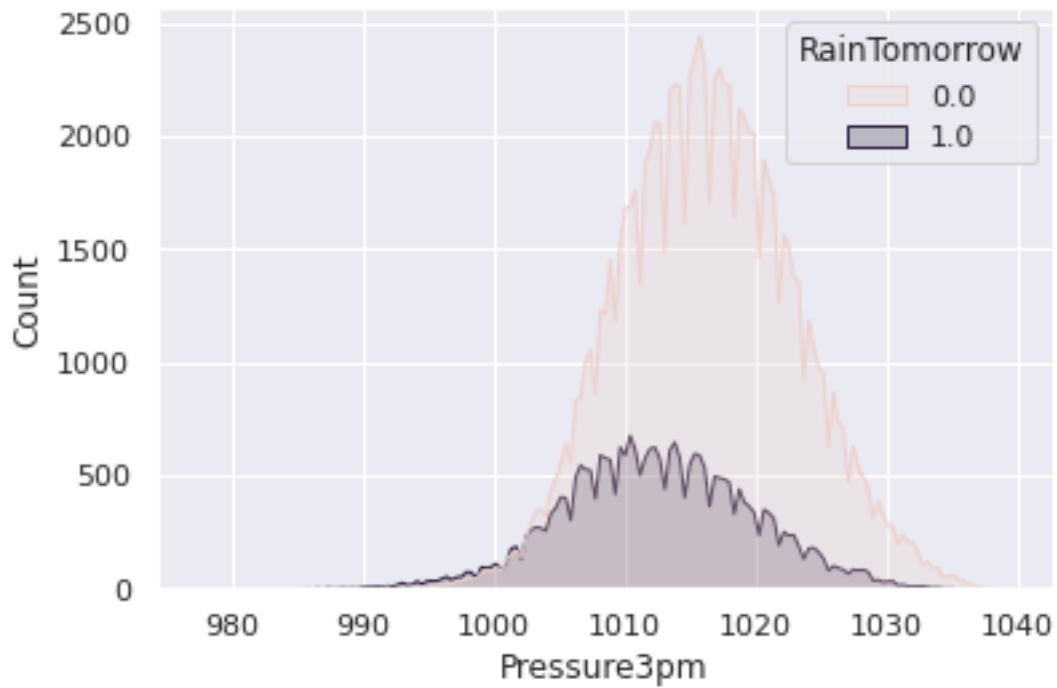
```
[134]: sns.histplot(data=aus_weather, x="Cloud9am", hue="RainTomorrow",  
↳multiple='dodge')
```

```
[134]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcff10c1730>
```



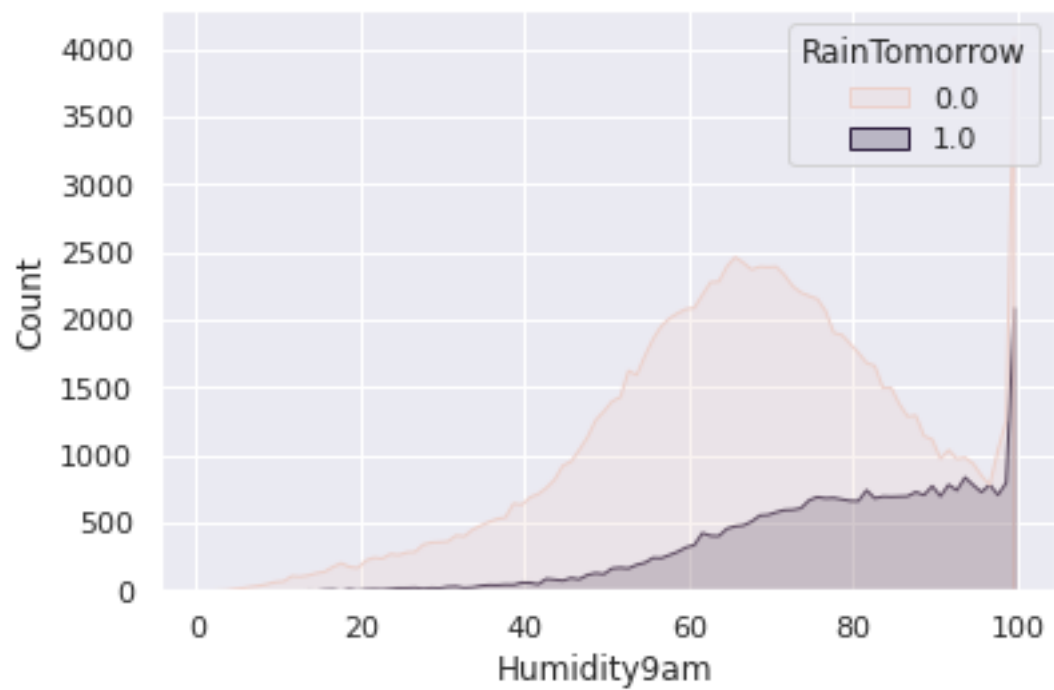
```
[137]: sns.histplot(data=aus_weather, x="Pressure3pm", hue="RainTomorrow",  
↳element="poly")
```

```
[137]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcff6630fd0>
```



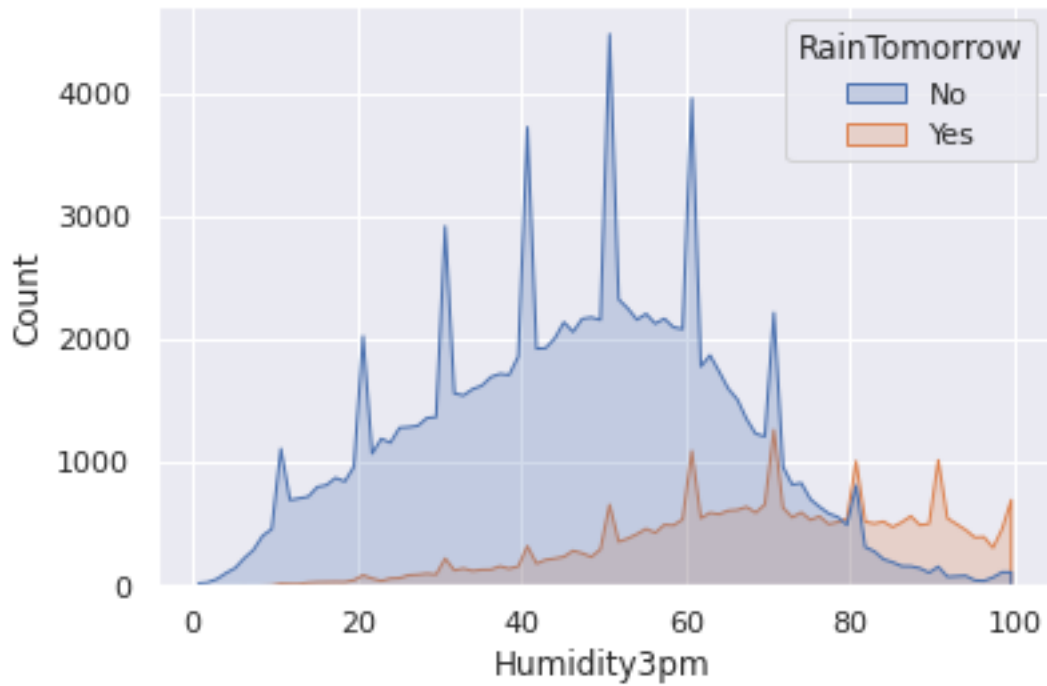
```
[138]: sns.histplot(data=aus_weather, x="Humidity9am", hue="RainTomorrow",  
↪ element="poly")
```

```
[138]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcff64c6160>
```



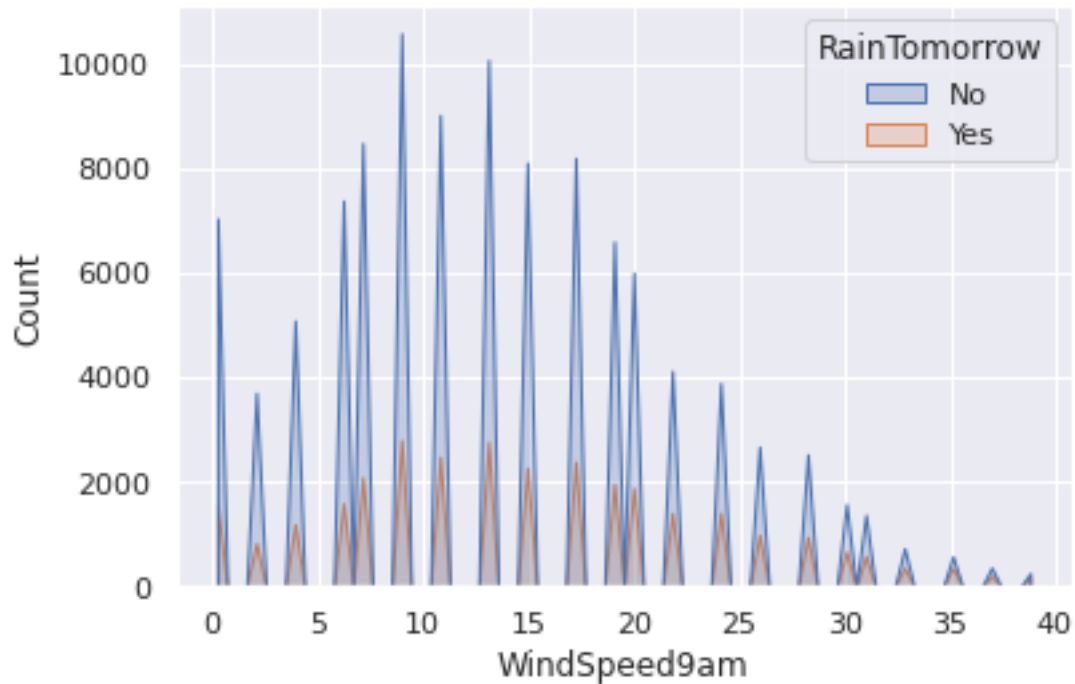
```
[33]: sns.histplot(data=aus_weather, x="Humidity3pm", hue="RainTomorrow",  
↪ element="poly")
```

```
[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f38115338e0>
```



```
[43]: sns.histplot(data=aus_weather[aus_weather["WindSpeed9am"]<40],  
↪ x="WindSpeed9am", hue="RainTomorrow", element="poly")
```

```
[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3811c27c70>
```



## 1.2 Dzielenie zbiorów

Podział na zbiór treningowy (a ten na treningowy i walidacyjny) i na zbiór testowy

```
[18]: from sklearn.model_selection import train_test_split
      from sklearn.metrics import confusion_matrix
```

```
[19]: X = pd.DataFrame( aus_weather)
      y = pd.DataFrame( aus_weather["RainTomorrow"])
```

```
[20]: y_fill = y.fillna('0')
```

```
[23]: #imputer = KNNImputer(n_neighbors=10, weights="uniform") #imputancja powodowała
      ↪ jakieś problemy, więc po prostu uzupełnimy dane zerami
      #imputer.fit_transform(X.sample(n=1000, random_state=1))
      #X_tr = imputer.transform(X)
      #from sklearn.experimental import enable_iterative_imputer
      #from sklearn.impute import IterativeImputer
      #imp = IterativeImputer(max_iter=10, random_state=0)
      #imp.fit(X.sample(n=1000, random_state=1))
      #X_tr = imp.transform(X)
      X_fill = X.fillna(0)
```

```
[24]: X_fill["RainToday"] = X_fill["RainToday"].astype( int)
X_fill.drop(columns = ["RainTomorrow"], inplace=True)
y_fill["RainTomorrow"] = y_fill["RainTomorrow"].astype( int)
```

```
[25]: X_train, X_test, y_train, y_test \
      = train_test_split(X_fill, y_fill, stratify = y_fill, test_size=0.2, \
      ↪random_state=1)

X_train, X_val, y_train, y_val \
      = train_test_split(X_train, y_train, stratify = y_train, test_size=0.25, \
      ↪random_state=1)
```

### 1.3 AdaBoostClassifier

```
[26]: from sklearn.ensemble import AdaBoostClassifier
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import f1_score
```

```
[27]: alf = AdaBoostClassifier(n_estimators=100, random_state=0, learning_rate=0.9)
      alf.fit(X_train, y_train)
```

```
[27]: AdaBoostClassifier(learning_rate=0.9, n_estimators=100, random_state=0)
```

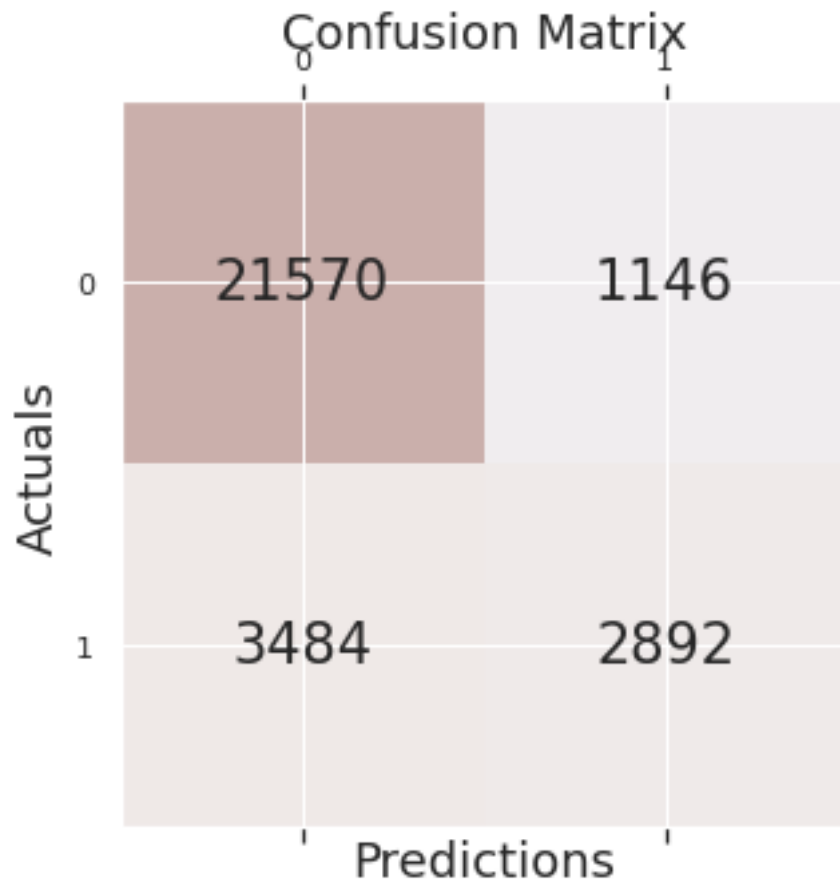
```
[28]: y_val_hat = alf.predict(X_val)
      y_test_hat = alf.predict(X_test)
```

```
[29]: alf.score(X_test, y_test)
```

```
[29]: 0.844218341812182
```

```
[30]: conf_matrix = confusion_matrix(y_true=y_val, y_pred=y_val_hat.round())
      fig, ax = plt.subplots(figsize=(5, 5))
      ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)
      for i in range(conf_matrix.shape[0]):
          for j in range(conf_matrix.shape[1]):
              ax.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center', \
              ↪size='xx-large')

      plt.xlabel('Predictions', fontsize=18)
      plt.ylabel('Actuals', fontsize=18)
      plt.title('Confusion Matrix', fontsize=18)
      plt.show()
```



```
[31]: print('F1 Score: %.3f' % f1_score(y_test, y_test_hat))
```

F1 Score: 0.571

```
[32]: print( accuracy_score(y_val, y_val_hat))
print( accuracy_score(y_test, y_test_hat))
```

0.84084971813557

0.844218341812182

Jak widzimy, predykcje Adaboosta są nieidealne.

## 1.4 GradientBoostingClassifier

```
[33]: from sklearn.ensemble import GradientBoostingClassifier
```

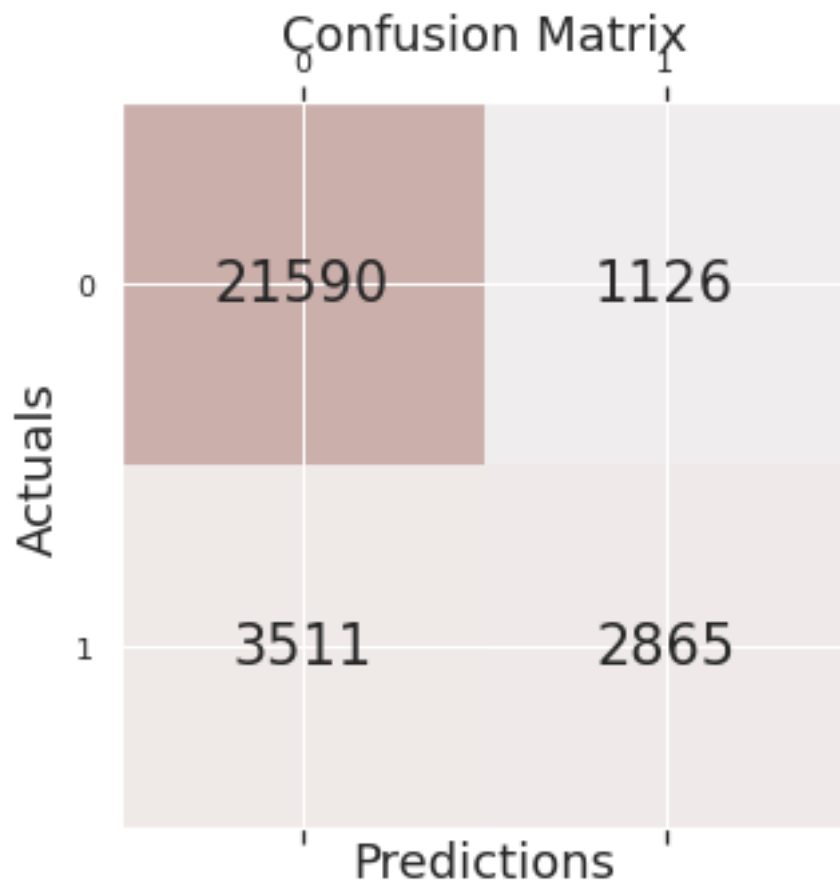
```
[34]: clf = GradientBoostingClassifier(n_estimators=90, learning_rate=0.8,
    max_depth=1, random_state=0).fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
[34]: 0.8430496356386635
```

```
[35]: y_val_hat = clf.predict(X_val)
      y_test_hat = clf.predict(X_test)
```

```
[36]: conf_matrix = confusion_matrix(y_true=y_val, y_pred=y_val_hat)
      fig, ax = plt.subplots(figsize=(5, 5))
      ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)
      for i in range(conf_matrix.shape[0]):
          for j in range(conf_matrix.shape[1]):
              ax.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center',
                      size='xx-large')

      plt.xlabel('Predictions', fontsize=18)
      plt.ylabel('Actuals', fontsize=18)
      plt.title('Confusion Matrix', fontsize=18)
      plt.show()
```



```
[37]: print('F1 Score: %.3f' % f1_score(y_test, y_test_hat))
```

F1 Score: 0.562

```
[38]: print( accuracy_score(y_val, y_val_hat))  
      print( accuracy_score(y_test, y_test_hat))
```

0.840609102158669

0.8430496356386635

GradientBoostingClassifier zdaje się być nieco gorszy od Adaboosta.

## 1.5 HistGradientBoostingRegressor

```
[49]: from sklearn.experimental import enable_hist_gradient_boosting  
      from sklearn.ensemble import HistGradientBoostingRegressor
```

```
[50]: est = HistGradientBoostingRegressor( l2_regularization= 2, learning_rate=0.7).  
      ↪fit(X_train, y_train)  
      est.score(X_val, y_val)
```

```
[50]: 0.34110104593782364
```

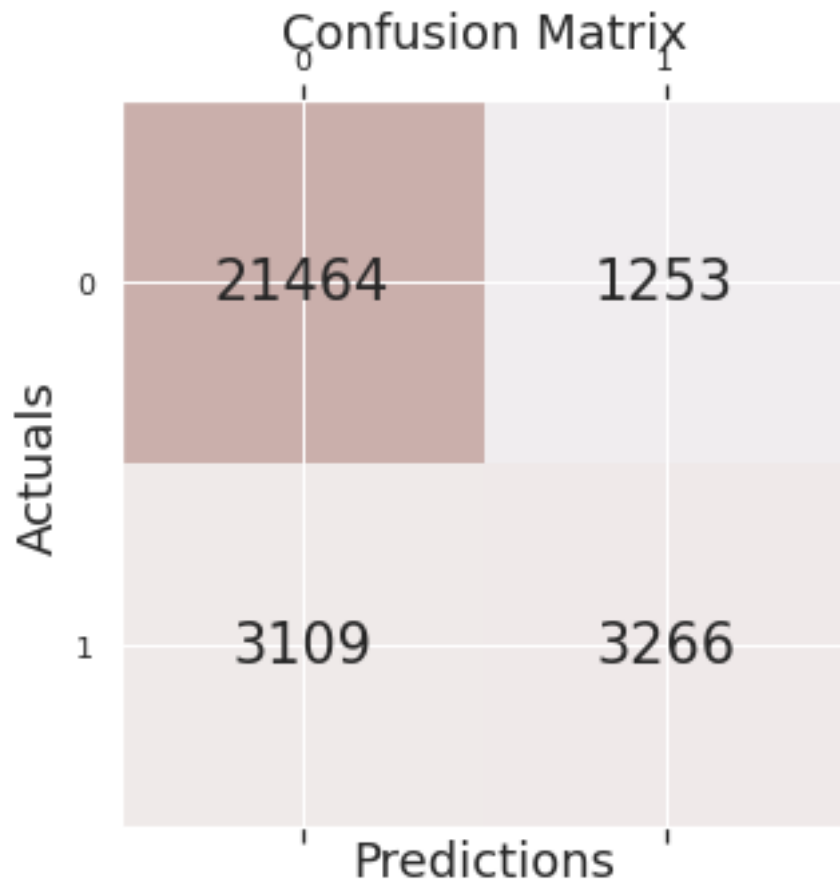
```
[51]: y_val_hat = est.predict(X_val).round()  
      y_test_hat = est.predict(X_test).round()
```

```
[52]: from sklearn.metrics import precision_score, recall_score, f1_score,  
      ↪accuracy_score  
      from sklearn.metrics import confusion_matrix
```

```
[53]: conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_test_hat)
```

```
[54]: fig, ax = plt.subplots(figsize=(5, 5))  
      ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)  
      for i in range(conf_matrix.shape[0]):  
          for j in range(conf_matrix.shape[1]):  
              ax.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center',  
                  ↪size='xx-large')  
  
      plt.xlabel('Predictions', fontsize=18)  
      plt.ylabel('Actuals', fontsize=18)  
      plt.title('Confusion Matrix', fontsize=18)  
      plt.show()
```





```
[55]: print('F1 Score: %.3f' % f1_score(y_test, y_test_hat))
```

F1 Score: 0.600

```
[56]: print( accuracy_score(y_val, y_val_hat))  
print( accuracy_score(y_test, y_test_hat))
```

0.8475182180668225

0.8500618726797745

Zdaje się zatem, że to HistGradientBoostingRegressor daje najlepsze rezultaty.

## 1.6 Wnioski

Zasadniczo każdy klasyfikator daje nieidealny rezultat. Należy jednak zaznaczyć, że ostatni z nich, HistGradientBoostingRegressor zdaje się dawać nieznacznie lepsze rezultaty od pozostałych.