

```

import numpy as np          # Одномерные и многомерные массивы (array)
import pandas as pd         # Таблицы и временные ряды (dataframe, series)
import matplotlib.pyplot as plt # Научная графика
import seaborn as sns        # Еще больше красивой графики для визуализации данных
import sklearn              # Алгоритмы машинного обучения
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

```

▼ Прогнозирование дождя на следующий день в Австралии

▼ 1. Описание задачи

Датасет weatherAUS.csv содержит данные ежедневных наблюдений за погодой в различных городах Австралии в период с 2007 по 2017 годы. Для каждого дня указаны различные метеорологические параметры, такие как температура, влажность, давление, скорость ветра, а также наличие осадков в течение дня. Всего в датасете содержится около 145 тысяч наблюдений и 23 признака. Данные были собраны автоматическими метеостанциями по всей территории Австралии.

Целевой переменной является бинарный признак RainTomorrow, который указывает, будет ли дождь на следующий день или нет (в колонке указывается "Да", если количество осадков в этот день составило 1 мм или более)

Данный датасет содержит следующие признаки:

- Date - дата наблюдения
- Location - местоположение, город наблюдения
- MinTemp - самая низкая температура дня (градусы Цельсия)
- MaxTemp - самая высокая температура дня (градусы Цельсия)
- Rainfall - количество осадков за день (единица измерения - мм)
- Evaporation - количество испарения воды, в течение 24 часов до 9 часов утра (единица измерения - мм)
- Sunshine - количество часов яркого солнечного света в течение дня.
- WindGustDir - направление сильнейшего ветра за 24 часа до полуночи
- WindGustSpeed - скорость самого сильного ветра за 24 часов до полуночи (км / ч)
- WindDir9am - направление ветра в 9 часов утра
- WindDir3pm - направление ветра в 3 часа дня
- WindSpeed9am - средняя скорость ветра за десять минут, с 8:50 до 9:00 (км / час)
- WindSpeed3pm - средняя скорость ветра за десять минут, с 14:50 до 15:00 (км / час)
- Humidity9am - влажность в 9 часов утра (%)
- Humidity3pm - влажность в 3 часа дня (%)
- Pressure9am - атмосферное давление в 9 часов утра (гпа)
- Pressure3pm - атмосферное давление в 3 часа дня (гпа)
- Cloud9am - часть неба, закрытая облаками в 9 часов утра, принимает значения из промежутка от 0 до 8, где значение 0 указывает на полностью ясное небо, а значение 8 - полностью затянуто облаками
- Cloud3pm - часть неба, закрытая облаками в 3 часа дня, принимает значения из промежутка от 0 до 8, где значение 0 указывает на полностью ясное небо, а значение 8 - полностью затянуто облаками
- Temp9am - температура в 9 часов утра (в градусах Цельсия)
- Temp3pm - температура в 3 часа дня (в градусах Цельсия)
- RainToday - идет ли дождь сегодня (да/нет)
- RainTomorrow - будет ли дождь завтра (да/нет)

▼ 2. Чтение данных

Загрузка данных из интернета:

```
url = "https://raw.githubusercontent.com/nik-se0/MLDA/main/weatherAUS.csv"
data_raw = pd.read_csv(url)
data_raw.shape

(145460, 23)
```

В текущем наборе данных содержится информация о 145460 наблюдениях, разделённая на 23 признака. Ознакомимся с информацией, хранящейся в нем

data_raw

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	38.0
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	45.0
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0
...
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	SE	...	51.0
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	SE	...	56.0
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	SE	...	53.0
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	SSE	...	51.0
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	NaN	ESE	...	62.0

145460 rows × 23 columns



data_raw.columns

```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

Подробная информация о столбцах набора данных

data_raw.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

```

По полученному описанию можно определить наличие пропущенных значений, а так же провести типизацию признаков:

- RainToday, RainTomorrow - являются категориальными, бинарными признаками.
- WindGustDir, WindDir9am, WindDir3pm - являются категориальными, номинальными признаками.
- Location - текстовой признак
- Остальные будут являться количественными (числовыми) признаками.

Однако все категориальные признаки являются по типу 'object', исправим это., заменив тип на специальный, предназначенный для хранения категориальных значений: 'category'

```

data_raw['WindGustDir'] = data_raw['WindGustDir'].astype('category')
data_raw['WindDir9am'] = data_raw['WindDir9am'].astype('category')
data_raw['WindDir3pm'] = data_raw['WindDir3pm'].astype('category')
data_raw['RainToday'] = data_raw['RainToday'].astype('category')
data_raw['RainTomorrow'] = data_raw['RainTomorrow'].astype('category')

```

```
data_raw['RainTomorrow'].dtype
```

```
CategoricalDtype(categories=['No', 'Yes'], ordered=False)
```

```
data_raw['WindGustDir'].dtype
```

```
CategoricalDtype(categories=['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE',
'SSE', 'SSW', 'SW', 'W', 'WNW', 'WSW'],
, ordered=False)
```

▼ 3. Визуализация данных, вычисление основных характеристик

Выведем описательную статистику по нашему датасету: математическое описание для каждого признака Сначала рассмотрим категориальные признаки

```
data_raw.describe(include=['category'])
```

	WindGustDir	WindDir9am	WindDir3pm	RainToday	RainTomorrow	
count	135134	134894	141232	142199	142193	grid
unique	16	16	16	2	2	bar
top	W	N	SE	No	No	
freq	9915	11758	10838	110319	110316	

- count - количество значений для каждого признака (значения признака Gender отличаются от всех остальных, так как в нем есть пропущенные значения)
- unique - количество уникальных значений
- top - значения, встречающиеся чаще всего
- freq - частота значений top

Теперь посмотрим на данные о количественных признаках

```
data_raw.describe()
```

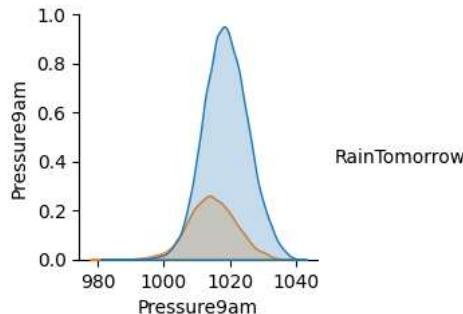
	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	135197.000000	143693.000000	142398.000000	142806.000000
mean	12.194034	23.221348	2.360918	5.468232	7.611178	40.035230	14.043426	18.662657	68.880000
std	6.398495	7.119049	8.478060	4.193704	3.785483	13.607062	8.915375	8.809800	19.029000
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000

- count – количество значений для каждого признака
- mean – среднее значение
- std – стандартное отклонение, значение которого показывает, на сколько в среднем отклоняются варианты от среднего значения
- min – минимальное значение
- 25% – значения, которые объект не превышает с вероятностью 25%
- 50% – медиана, то есть значения, которые объект не превышает с вероятностью 50%
- 75% – значения, которые объект не превышает с вероятностью 75%
- max – максимальное значение

Внимательно изучив таблицу можно решить, в каких признаках желательно обработать выбросы. Считаем необходимым рассмотреть такие признаки, как: Humidity9am, WindSpeed3pm, WindSpeed9am, WindGustSpeed, Evaporation, Rainfall, Pressure3pm, Pressure9am, так как их средние и медианные значения ушли далеко от половины максимального значения.

```
sns.pairplot(data_raw[["Pressure9am", "RainTomorrow"]], hue="RainTomorrow")
```

```
<seaborn.axisgrid.PairGrid at 0x7fb3cd9ab250>
```



```
sns.violinplot(y="Pressure9am", data=data_raw)
```

```
<Axes: ylabel='Pressure9am'>
```

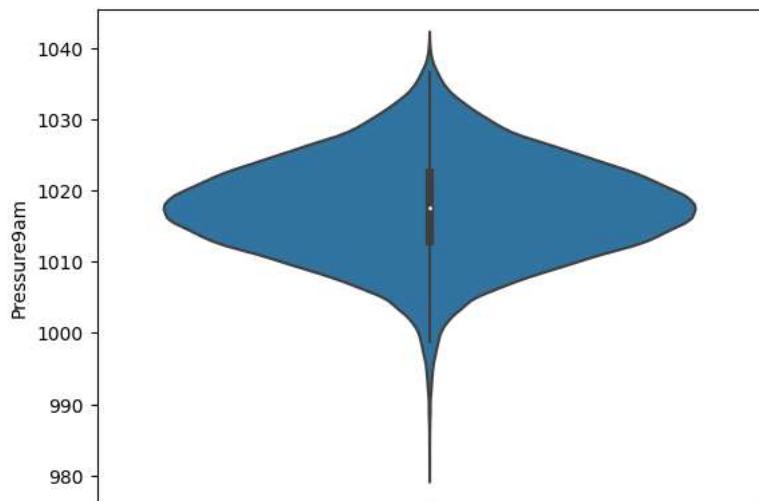
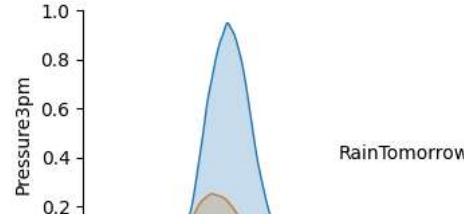


График напоминает нечто похожее на нормальное распределение, выбросы не портят, гармонично дополняют картину на наш взгляд

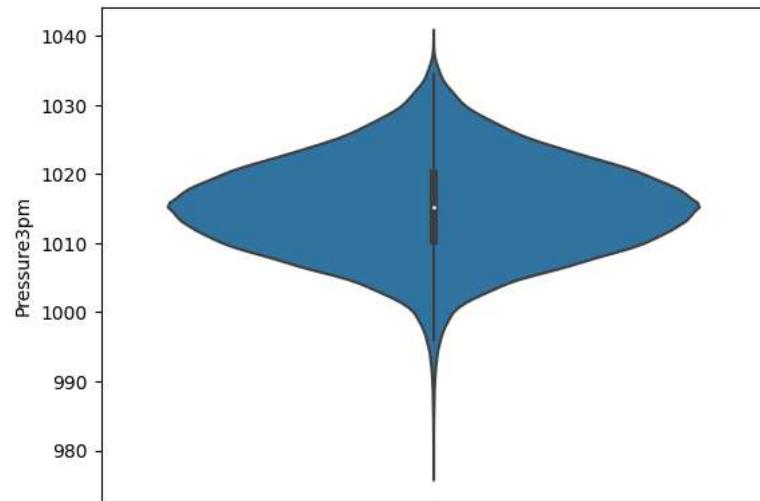
```
sns.pairplot(data_raw[["Pressure3pm", "RainTomorrow"]], hue="RainTomorrow")
```

```
<seaborn.axisgrid.PairGrid at 0x7fb3cc11ef80>
```



```
sns.violinplot(y="Pressure3pm", data=data_raw)
```

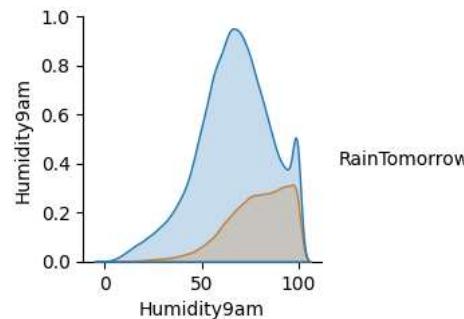
```
<Axes: ylabel='Pressure3pm'>
```



Такое же красивое распределение

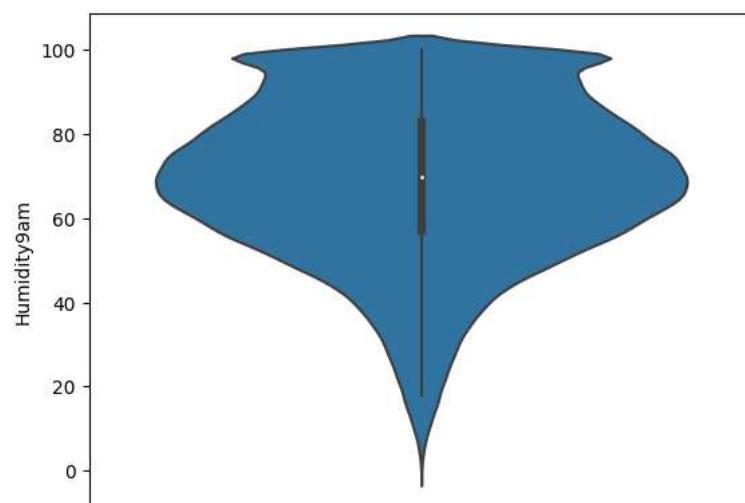
```
sns.pairplot(data_raw[["Humidity9am", "RainTomorrow"]], hue="RainTomorrow")
```

```
<seaborn.axisgrid.PairGrid at 0x7fb3d05f7400>
```



```
sns.violinplot(y="Humidity9am", data=data_raw)
```

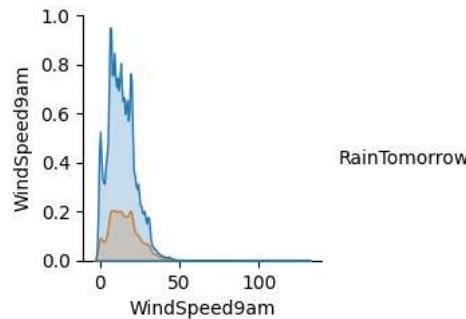
```
<Axes: ylabel='Humidity9am'>
```



Этот график уже меньше напоминает нормальное распределение, однако выбросы все так же не портят картину на наш взгляд

```
sns.pairplot(data_raw[["WindSpeed9am", "RainTomorrow"]], hue="RainTomorrow")
```

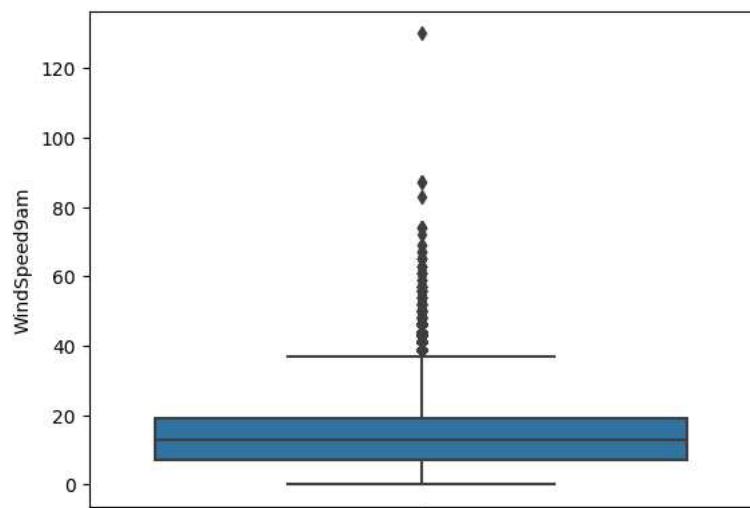
```
<seaborn.axisgrid.PairGrid at 0x7fb3d05f6170>
```



А вот тут явно есть выбросы, которые сильно расширяют график. Рассмотрим их подробнее и избавимся от них

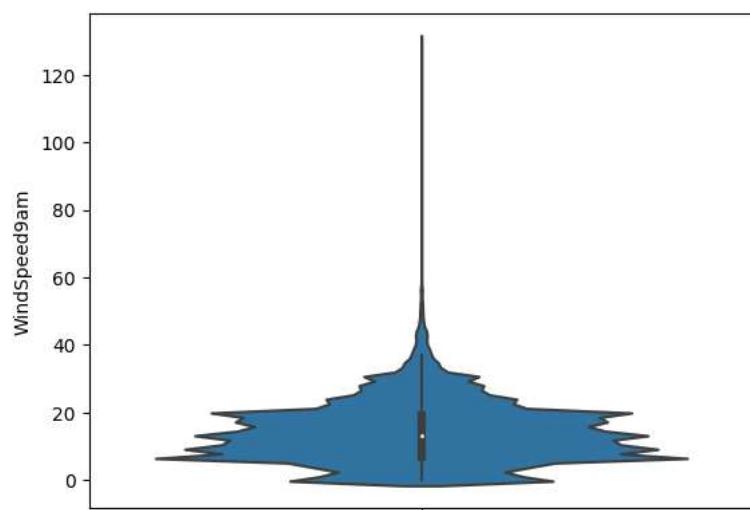
```
sns.boxplot(y="WindSpeed9am", data=data_raw)
```

```
<Axes: ylabel='WindSpeed9am'>
```



```
sns.violinplot(y="WindSpeed9am", data=data_raw)
```

```
<Axes: ylabel='WindSpeed9am'>
```

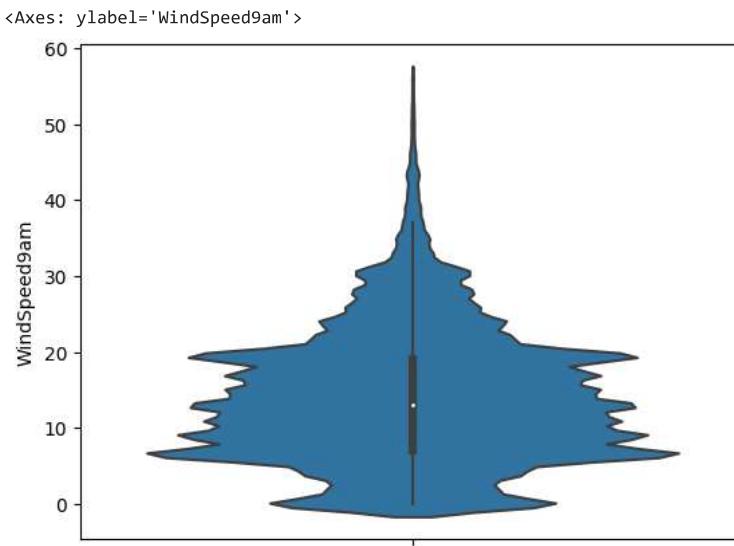


```
data_raw['WindSpeed9am'].quantile(0.9995)
```

```
56.0
```

```
rows_to_drop = data_raw[data_raw['WindSpeed9am'] > data_raw['WindSpeed9am'].quantile(0.9995)].index
len(rows_to_drop)
data_raw = data_raw.drop(rows_to_drop)
```

```
sns.violinplot(y="WindSpeed9am", data=data_raw)
```

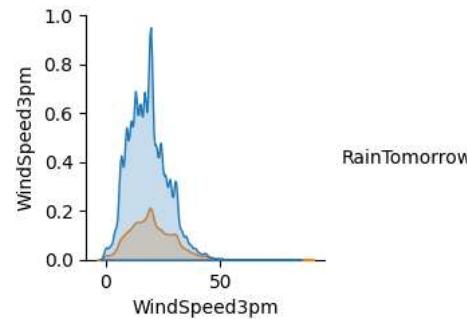


Теперь картина стала намного лучше.

Продолжим анализ для оставшихся параметров

```
sns.pairplot(data_raw[["WindSpeed3pm", "RainTomorrow"]], hue="RainTomorrow")
```

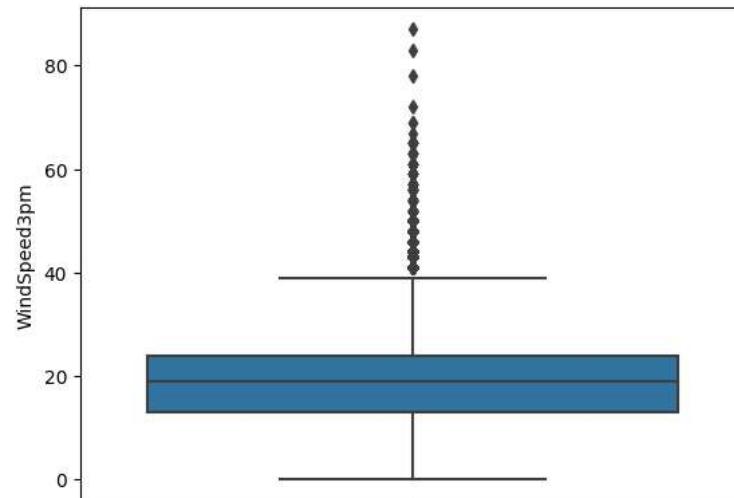
<seaborn.axisgrid.PairGrid at 0x7fb3ccdb53f0>



Ситуация аналогична предыдущей, избавимся от выбросов

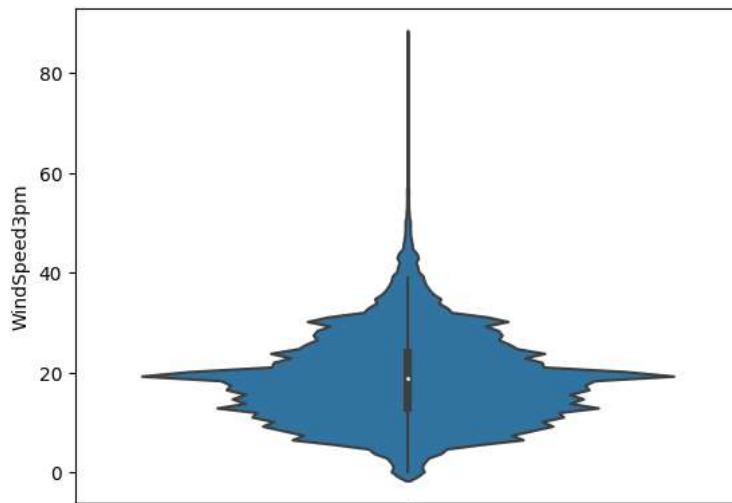
```
sns.boxplot(y="WindSpeed3pm", data=data_raw)
```

<Axes: ylabel='WindSpeed3pm'>



```
sns.violinplot(y="WindSpeed3pm", data=data_raw)
```

```
<Axes: ylabel='WindSpeed3pm'>
```



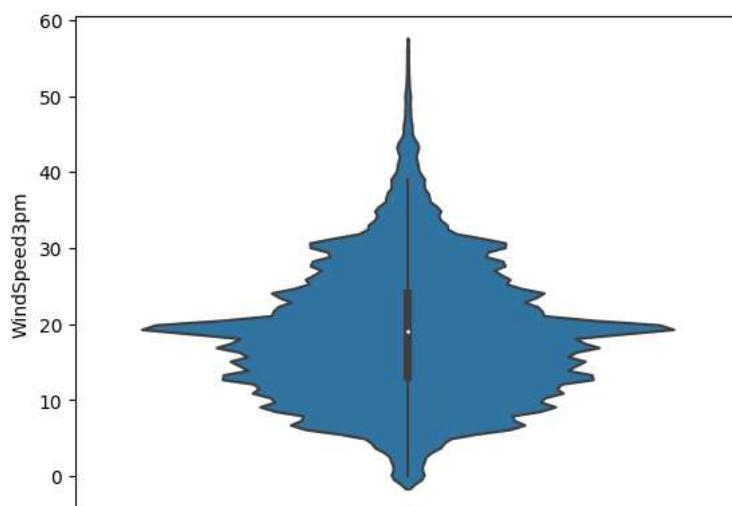
```
data_raw['WindSpeed3pm'].quantile(0.9993)
```

```
56.0
```

```
rows_to_drop = data_raw[data_raw['WindSpeed3pm'] > data_raw['WindSpeed3pm'].quantile(0.9993)].index
len(rows_to_drop)
data_raw = data_raw.drop(rows_to_drop)
```

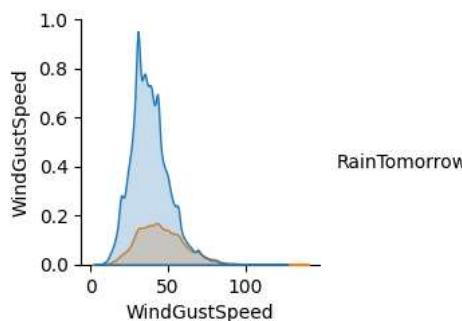
```
sns.violinplot(y="WindSpeed3pm", data=data_raw)
```

```
<Axes: ylabel='WindSpeed3pm'>
```



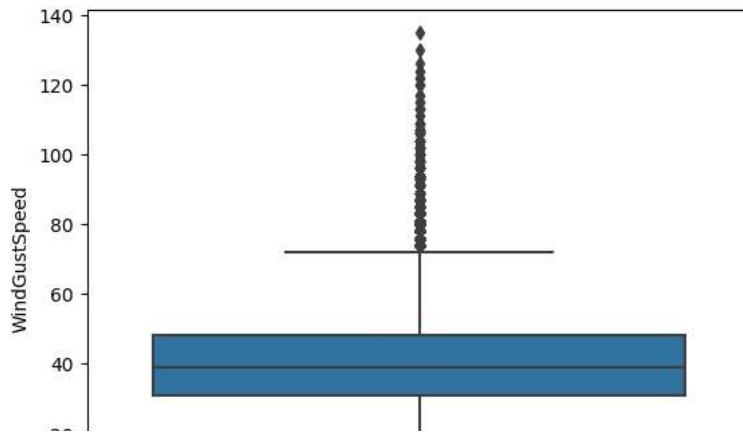
```
sns.pairplot(data_raw[["WindGustSpeed", "RainTomorrow"]], hue="RainTomorrow")
```

```
<seaborn.axisgrid.PairGrid at 0x7fb3d0621ea0>
```



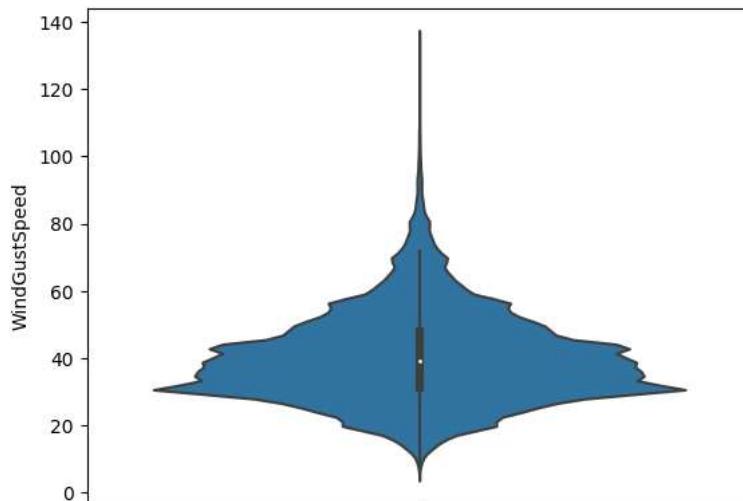
```
sns.boxplot(y="WindGustSpeed", data=data_raw)
```

<Axes: ylabel='WindGustSpeed'>



sns.violinplot(y="WindGustSpeed", data=data_raw)

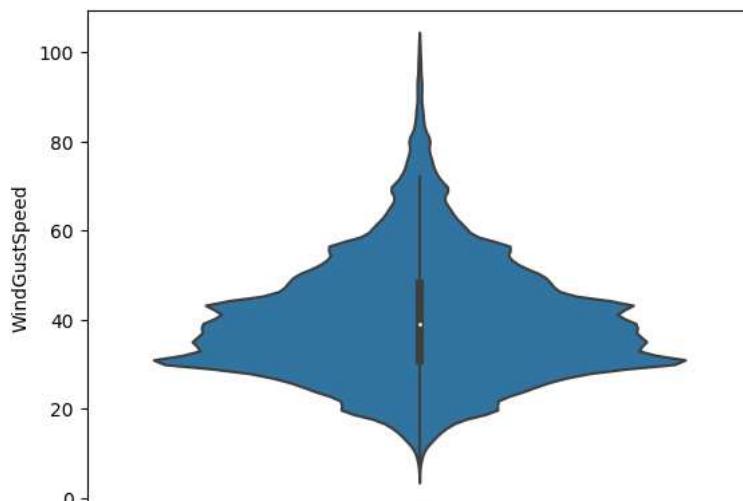
<Axes: ylabel='WindGustSpeed'>



```
rows_to_drop = data_raw[data_raw['WindGustSpeed'] > data_raw['WindGustSpeed'].quantile(0.9993)].index
len(rows_to_drop)
data_raw = data_raw.drop(rows_to_drop)
```

sns.violinplot(y="WindGustSpeed", data=data_raw)

<Axes: ylabel='WindGustSpeed'>



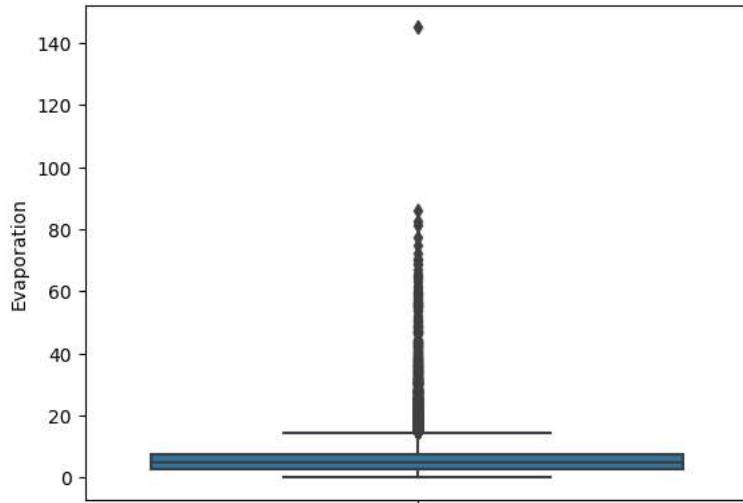
sns.pairplot(data_raw[["Evaporation", "RainTomorrow"]], hue="RainTomorrow")

```
<seaborn.axisgrid.PairGrid at 0x7fb3ccae4a90>
```



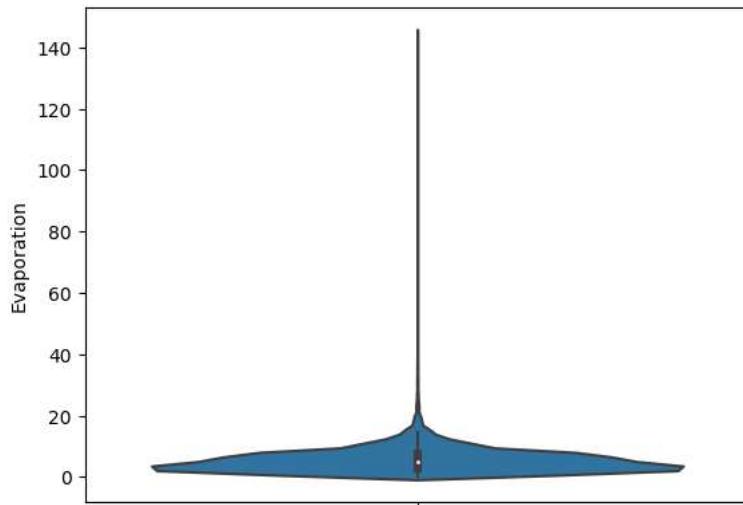
```
sns.boxplot(y="Evaporation", data=data_raw)
```

```
<Axes: ylabel='Evaporation'>
```



```
sns.violinplot(y="Evaporation", data=data_raw)
```

```
<Axes: ylabel='Evaporation'>
```

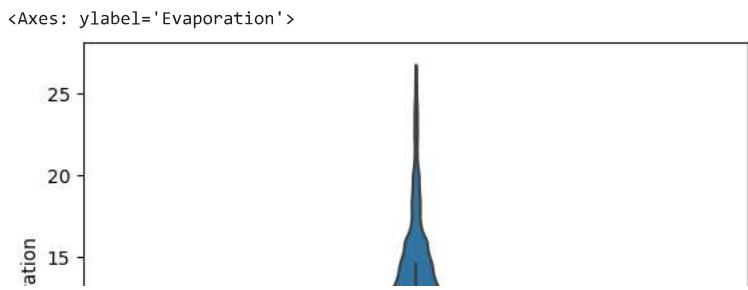


```
data_raw['Evaporation'].quantile(0.9965)
```

```
26.019900000002234
```

```
rows_to_drop = data_raw[data_raw['Evaporation'] > data_raw['Evaporation'].quantile(0.9965)].index
len(rows_to_drop)
data_raw = data_raw.drop(rows_to_drop)
```

```
sns.violinplot(y="Evaporation", data=data_raw)
```



Интересно посмотреть, сколько раз номинальные переменные принимают то или иное значение

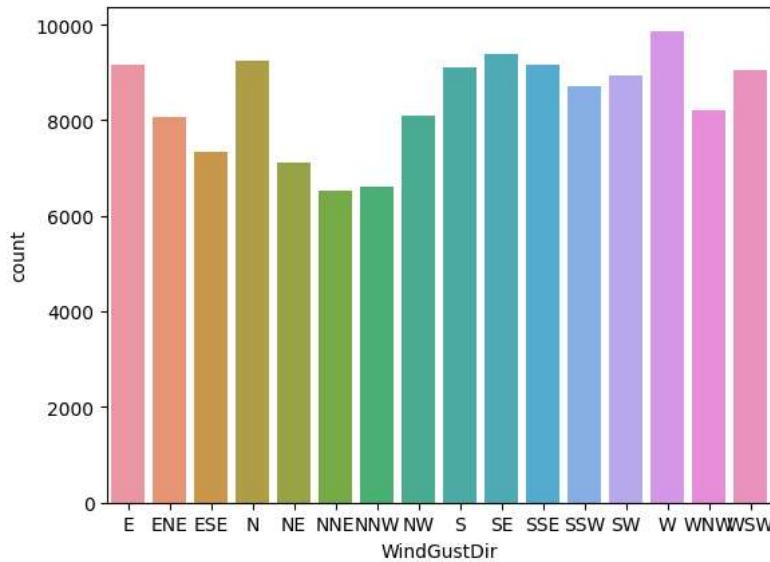
data_raw['WindGustDir'].value_counts()

```

W      9873
SE     9387
N      9253
SSE    9175
E      9160
S      9115
WSW   9041
SW    8947
SSW   8710
WNW   8215
NW    8087
ENE   8078
ESE   7350
NE    7110
NNW   6606
NNE   6530
Name: WindGustDir, dtype: int64

```

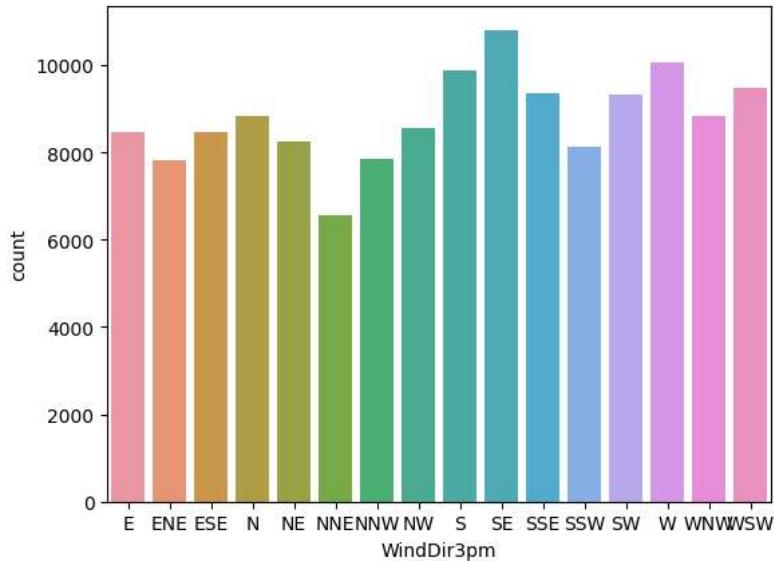
sns.countplot(x='WindGustDir', data=data_raw)
pass



sns.countplot(x='WindDir9am', data=data_raw)
pass



```
sns.countplot(x='WindDir3pm', data=data_raw)
pass
```



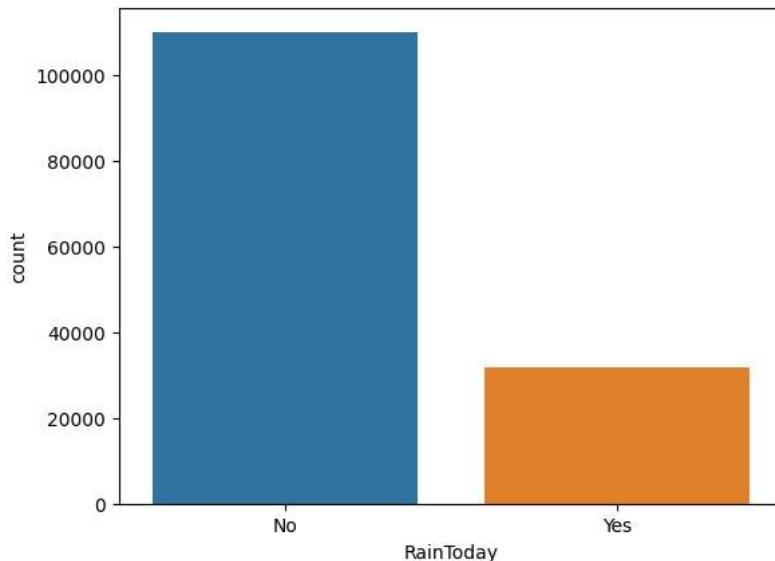
Аналогичным образом посмотрим, сколько раз бинарные признаки принимают свои значение

```
data_raw['RainToday'].value_counts()
```

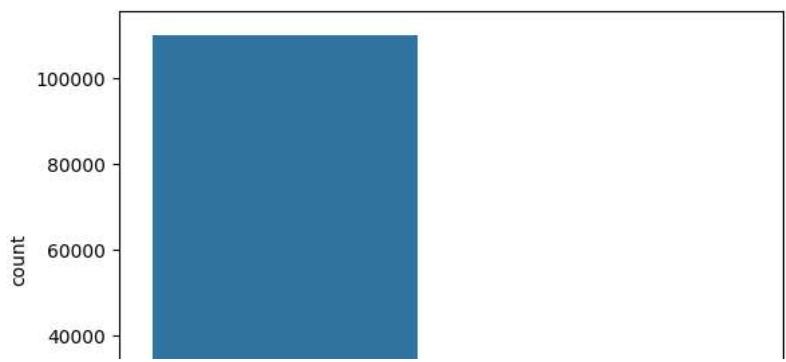
RainToday	Count
No	109948
Yes	31729

Name: RainToday, dtype: int64

```
sns.countplot(x='RainToday', data=data_raw)
pass
```

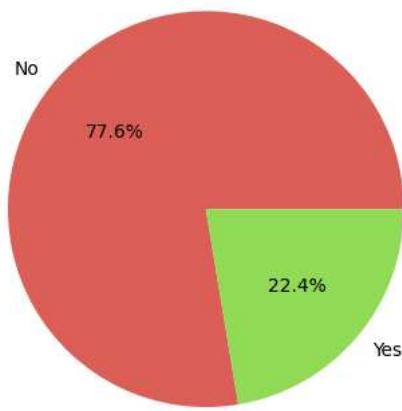


```
sns.countplot(x='RainTomorrow', data=data_raw)
pass
```



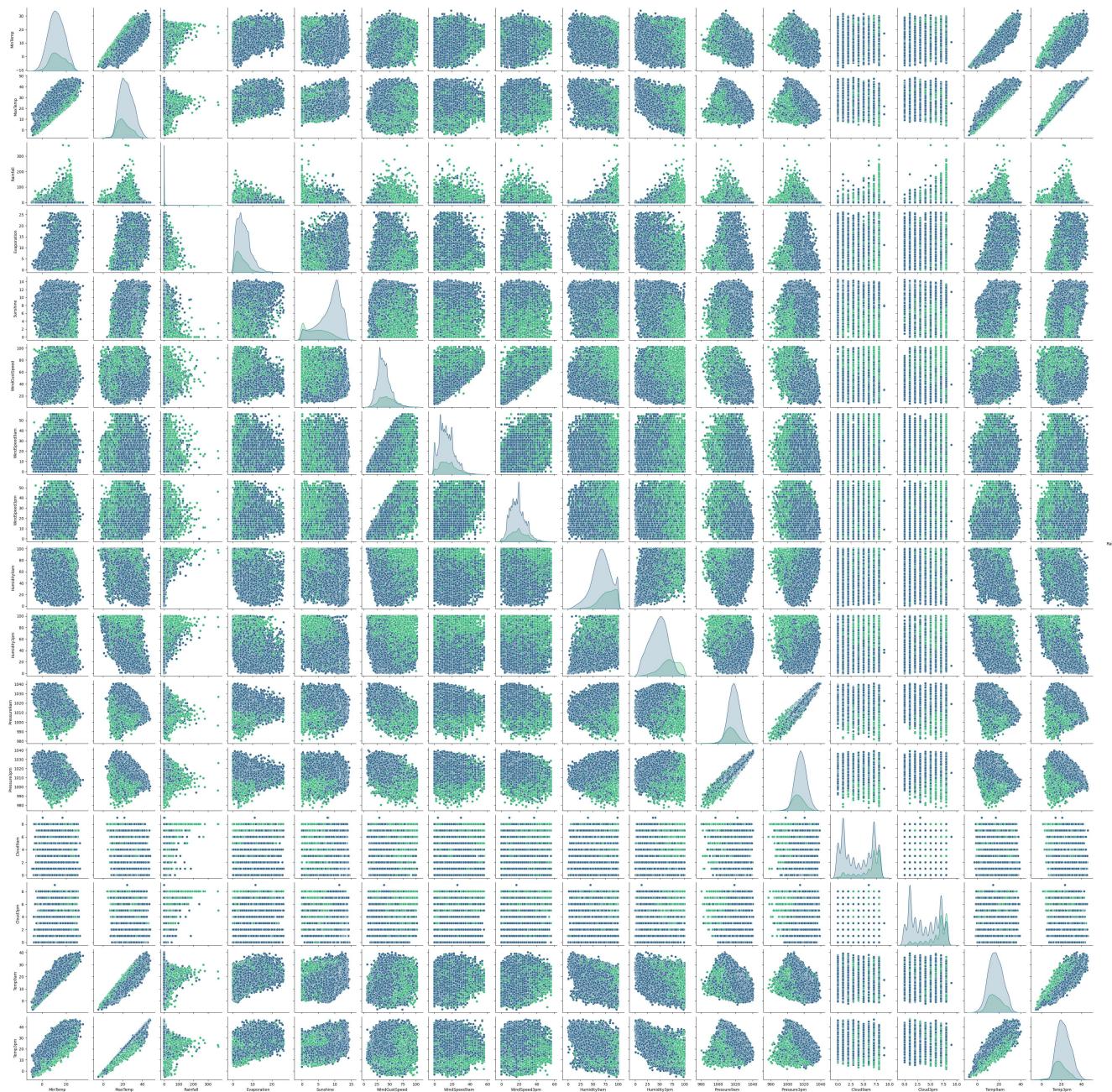
Видим что данные не сбалансированные. Можем посмотреть на соотношение классов более подробно

```
20000 | [REDACTED] |  
plt.pie(data_raw['RainTomorrow'].value_counts(), labels = ["No", "Yes"], colors = sns.color_palette("hls",4), autopct = '%1.1f%%')  
pass
```



Построение диаграммы рассеивания для признаков.

```
sns.pairplot(data_raw, hue="RainTomorrow", palette="viridis")  
pass
```

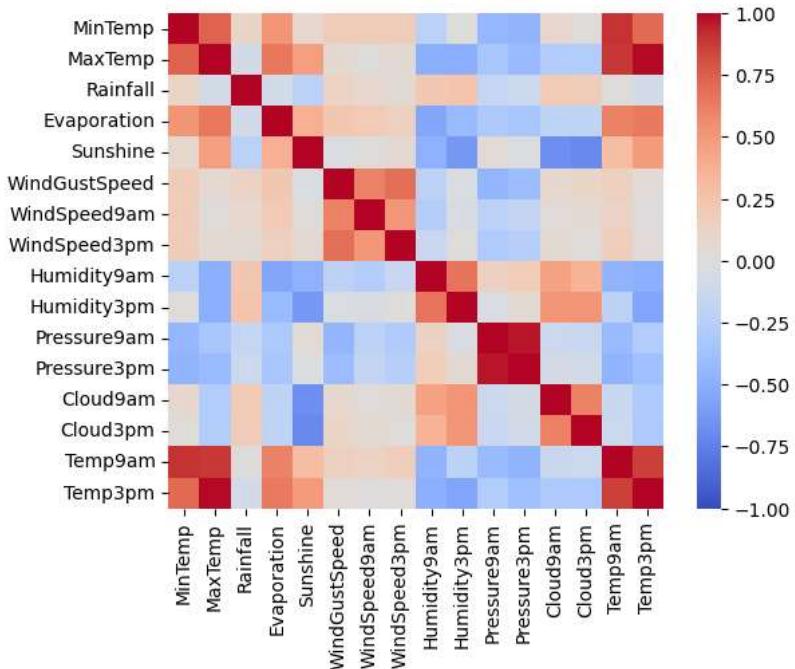


Выведем матрицу корреляции наших численных признаков:

```
corr_mat = data_raw.corr(numeric_only=True)
corr_mat
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidi
MinTemp	1.000000	0.736282	0.104172	0.522646	0.071849	0.179228	0.176524	0.177627	-0.231884	0.0
MaxTemp	0.736282	1.000000	-0.075179	0.651887	0.469904	0.069259	0.015588	0.052938	-0.503147	-0.5
Rainfall	0.104172	-0.075179	1.000000	-0.081570	-0.227087	0.129855	0.081965	0.053259	0.224866	0.2

```
sns.heatmap(corr_mat, square=True, vmin=-1, vmax=1, cmap='coolwarm')
pass
```



Получим список признаков в которых наблюдается корреляция

```
import numpy as np
corr_mat.where(np.triu(corr_mat > 0.5, k=1)).stack()
```

MinTemp	MaxTemp	0.736282
	Evaporation	0.522646
	Temp9am	0.901819
	Temp3pm	0.708645
MaxTemp	Evaporation	0.651887
	Temp9am	0.887023
	Temp3pm	0.984525
Evaporation	Temp9am	0.608916
	Temp3pm	0.634632
WindGustSpeed	WindSpeed9am	0.602326
	WindSpeed3pm	0.682473
WindSpeed9am	WindSpeed3pm	0.515201
Humidity9am	Humidity3pm	0.666096
Humidity3pm	Cloud9am	0.516606
	Cloud3pm	0.523238
Pressure9am	Pressure3pm	0.961420
Cloud9am	Cloud3pm	0.603239
Temp9am	Temp3pm	0.860427

dtype: float64

Исходя из этих данных заметна высокая, очевидная корреляция:

- между MinTemp, MaxTemp, Temp9am, Temp3pm - признаки температуры
- между Evaporation и MinTemp, MaxTemp, Temp9am, Temp3pm - признаки температуры и испарения
- между WindGustSpeed, WindSpeed9am, WindSpeed3pm - признаки скорости ветра
- между Humidity9am, Humidity3pm - признаки влажности
- между Cloud9am, Cloud3pm - признаки облачности

Более интересные корреляции:

- между Sunshine, Temp3pm - признаки солнечности и температуры
- между Humidity3pm, Cloud9am, Cloud3pm - признаки влажности и облачности

```
import numpy as np
corr_mat.where(np.triu(corr_mat < -0.5, k=1)).stack()
```

```

MaxTemp      Humidity9am   -0.503147
              Humidity3pm   -0.507599
Evaporation  Humidity9am   -0.554284
Sunshine     Humidity3pm   -0.628934
              Cloud9am      -0.674977
              Cloud3pm      -0.704064
Humidity3pm  Temp3pm     -0.556671
dtype: float64

```

Так же заметная обратная корреляции:

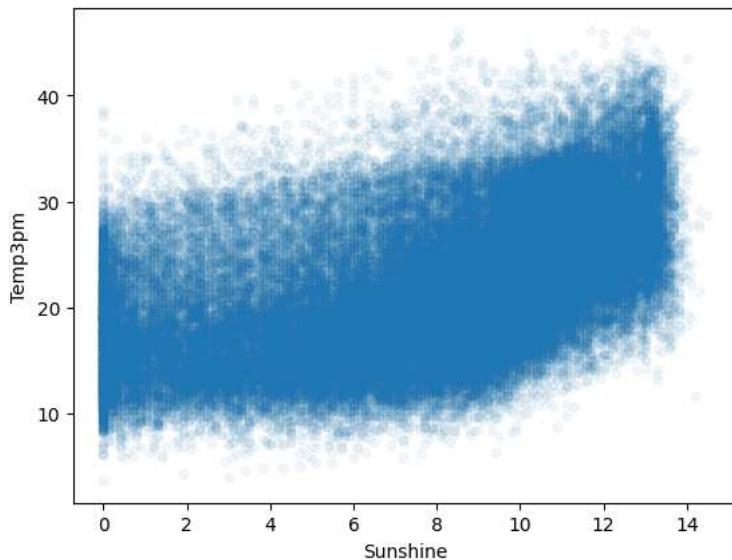
- между MaxTemp, Humidity9am, Humidity3pm - признаки температуры и влажности
- между Sunshine, Cloud9am, Cloud3pm - признаки солнечности и облачности

Выведем графики, нескольких интересующих нас пар:

```

data_raw.plot(kind = 'scatter', x = 'Sunshine', y = 'Temp3pm', alpha=.05)
pass

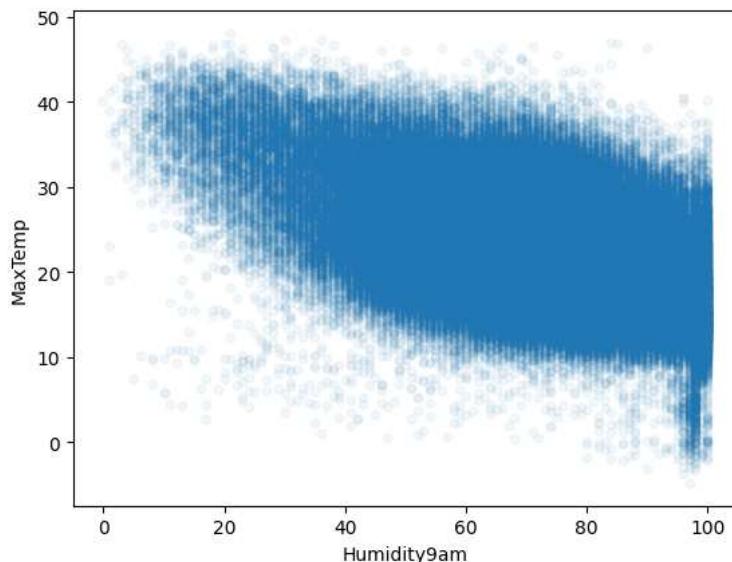
```



```

data_raw.plot(kind = 'scatter', x = 'Humidity9am', y = 'MaxTemp', alpha=.05)
pass

```



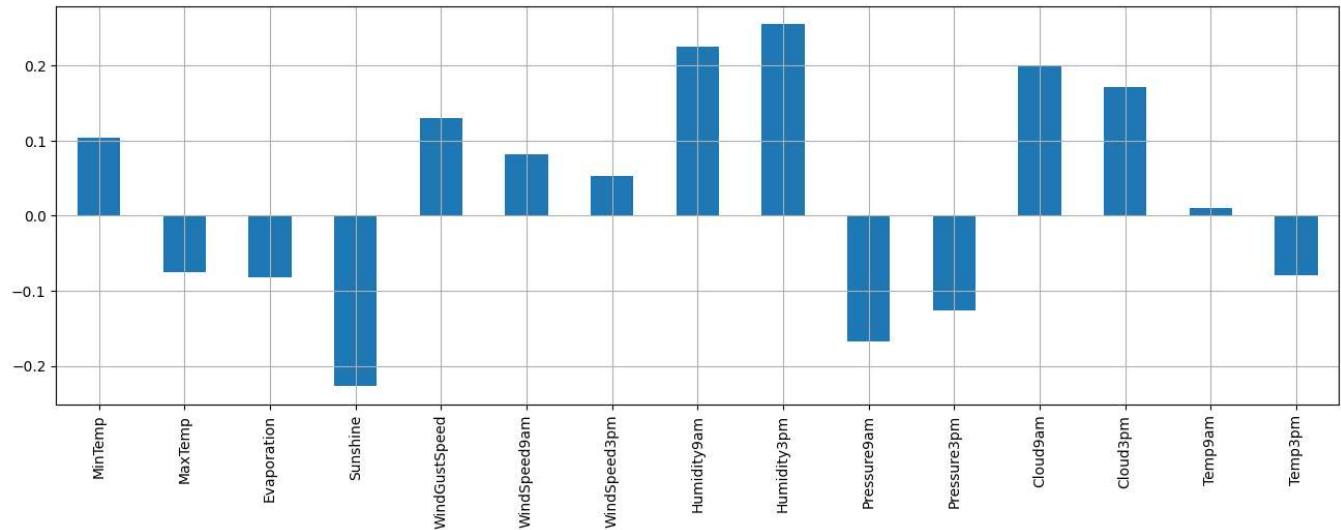
```

data_raw.drop('Rainfall', axis=1).corrwith(data_raw['Rainfall']).plot(kind='bar', grid=True, figsize=(16, 5));
pass

```

```
<ipython-input-454-9042fc1deb36>:1: FutureWarning: The default value of numeric_only in DataFrame.corrwith is deprecated. In a future version, it will default to False.
```

```
data_raw.drop('Rainfall', axis=1).corrwith(data_raw['Rainfall']).plot(kind='bar', grid=True, figsize=(16, 5));
```



Проверим взаимосвязь между численными признаками и дождем завтра, а именно как влияет на дождь:

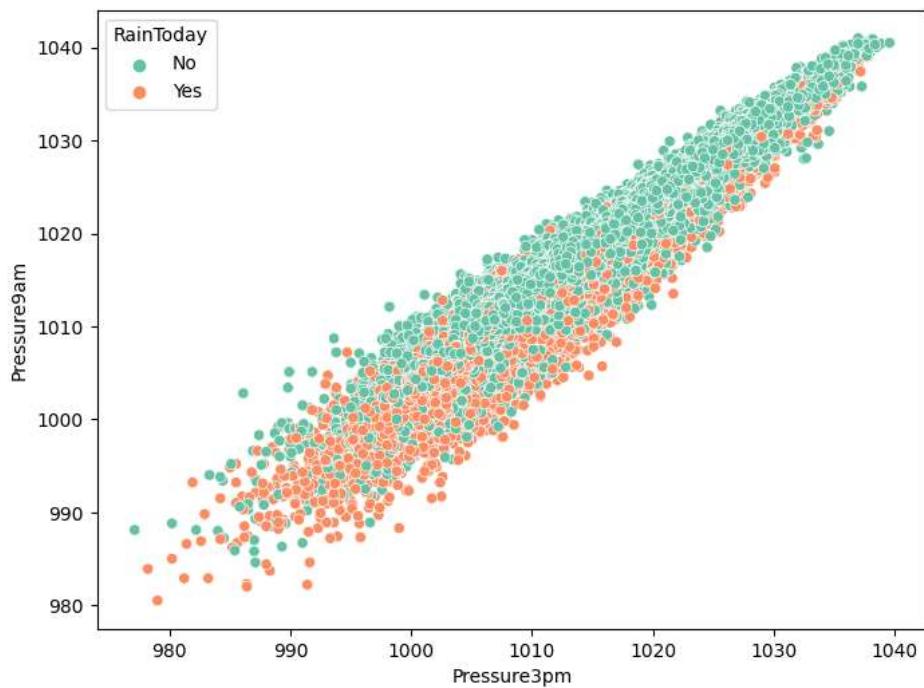
- Humidity
- Pressure
- Temp

А так же проверим взаимосвязь между:

- Rain today ~ Rain tomorrow

Атмосферное давление

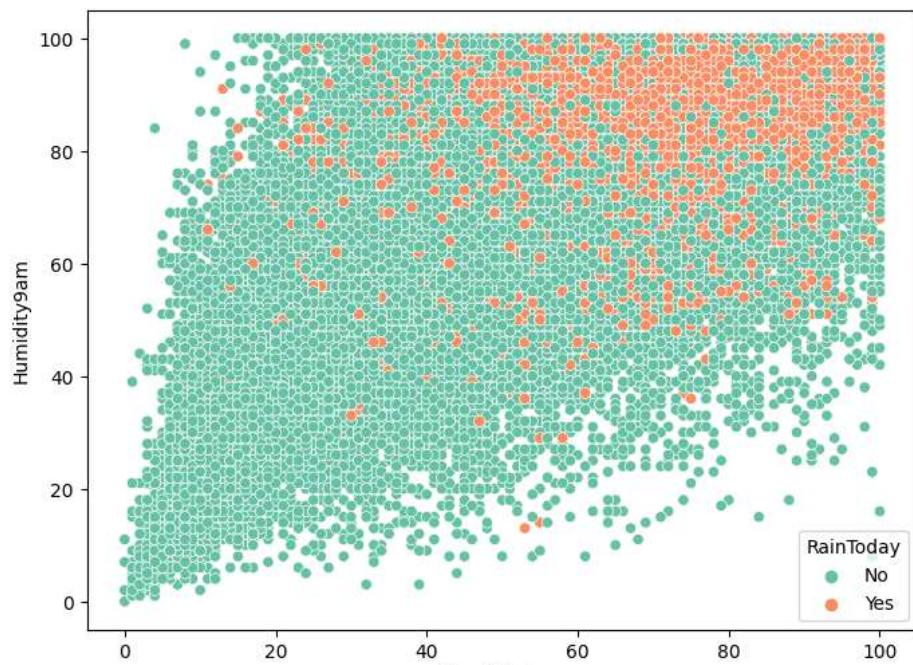
```
plt.figure(figsize=(8,6))
sns.scatterplot(data=data_raw,x='Pressure3pm',y='Pressure9am',hue='RainToday', palette='Set2');
```



Низкое давление увеличивает вероятность дождя

Влажность

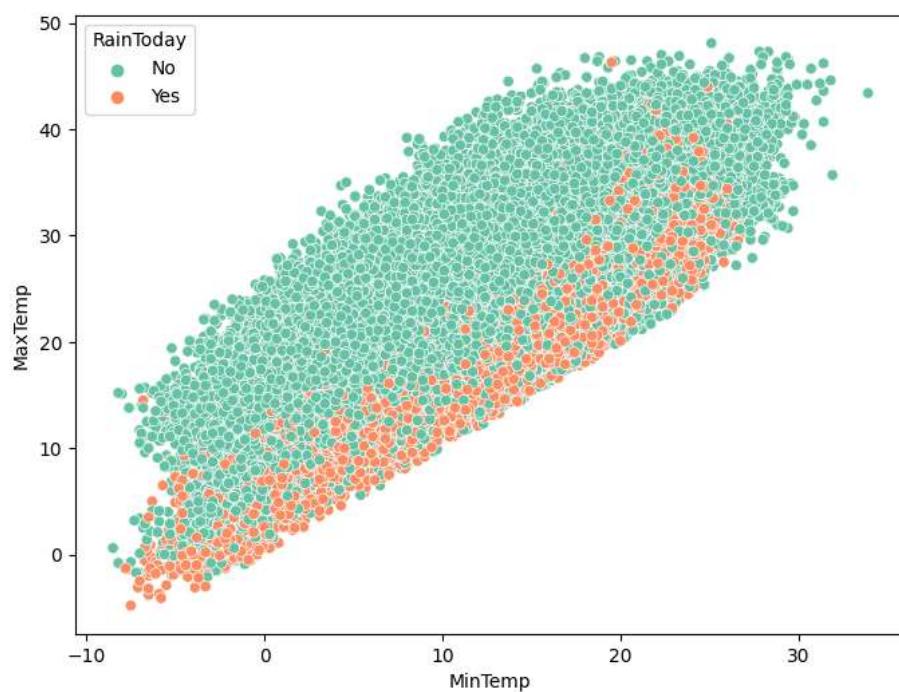
```
plt.figure(figsize=(8,6))
sns.scatterplot(data=data_raw,x='Humidity3pm',y='Humidity9am',hue='RainToday', palette='Set2');
pass
```



Высокая влажность увеличивает вероятность дождя

Температура

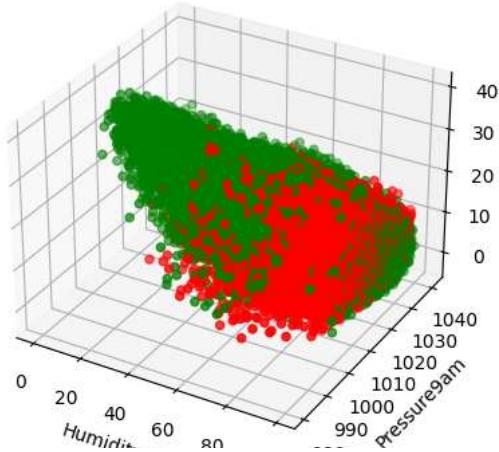
```
plt.figure(figsize=(8,6))
sns.scatterplot(x='MinTemp', y='MaxTemp', data=data_raw, hue='RainToday', palette='Set2');
```



Вероятность дождя завтра больше, если максимальная и минимальная температуры дня примерно равны

```
# Создание трехмерного графика
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('Humidity9am')
ax.set_ylabel('Pressure9am')
ax.set_zlabel('Temp9am')
c = ['r' if val == 'Yes' else 'g' for val in data_raw['RainToday']]
ax.scatter(data_raw['Humidity9am'], data_raw['Pressure9am'], data_raw['Temp9am'], c=c)

# Отображение графика
plt.show()
```

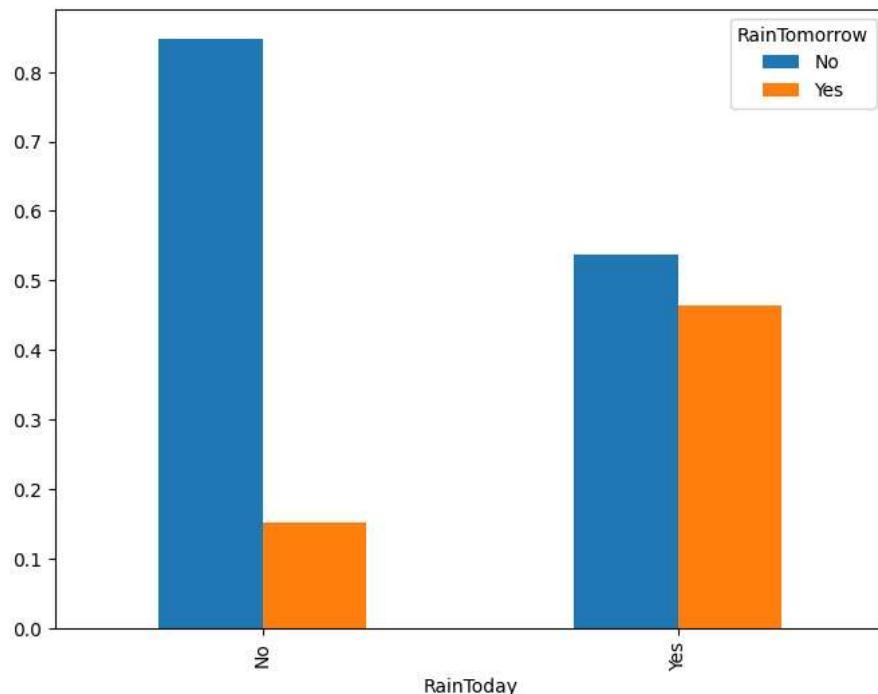


Получившийся график подтверждает наши утверждения: низкое давление и высокая влажность повышают вероятность дождя

Дождь сегодня - дождь завтра

Посмотрим зависимость между тем, есть ли дождь сегодня, и будет ли дождь завтра

```
ct = pd.crosstab(data_raw['RainToday'], columns=data_raw['RainTomorrow'], normalize='index')
ct.plot(kind="bar", figsize=(8,6));
```



Если сегодня нет дождя, завтра дождь пойдет с вероятностью = 15% Если же дождь идет сегодня, вероятность дождя завтра = 47%

Рассмотрим более подробно.

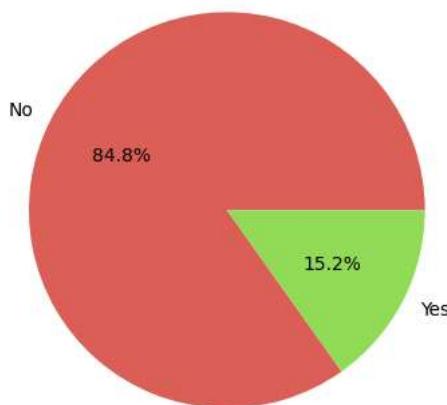
Если сегодня есть дождь до вероятность, что дождь будет завтра выглядит следующим образом

```
plt.pie([ct['No']['Yes'], ct['Yes']['Yes']], labels=['No', 'Yes'], colors = sns.color_palette("hls",4), autopct='%.1f%%')
pass
```



Аналогично, если сегодня есть дождь до вероятность, что дождь будет завтра будет такова

```
plt.pie([ct['No']['No'], ct['Yes']['No']], labels=['No','Yes'], colors = sns.color_palette("hls",4), autopct='%1.1f%%')
pass
```



```
data_raw.drop('Date', axis=1, inplace=True)
```

▼ 4. Обработка пропущенных значений

Теперь проверим датасет на наличие пропущенных значений:

```
data_raw.isnull().sum()
```

Column	Count
Location	0
MinTemp	1481
MaxTemp	1258
Rainfall	3254
Evaporation	62676
Sunshine	69528
WindGustDir	10294
WindGustSpeed	10231
WindDir9am	10553
WindDir3pm	4218
WindSpeed9am	1763
WindSpeed3pm	3052
Humidity9am	2644
Humidity3pm	4486
Pressure9am	15041
Pressure3pm	15003
Cloud9am	55740
Cloud3pm	59196
Temp9am	1763
Temp3pm	3593
RainToday	3254
RainTomorrow	3252

dtype: int64

Мы видим столбцы с пропущенными значениями. Можно было бы просто удалить строки с этими ячейками, но их достаточно много.

Поэтому заполним медианами пропущенные значения в столбцах, соответствующих числовым признакам.

```
data_raw.fillna(data_raw.median(axis = 0), axis=0 , inplace=True)

<ipython-input-79-dcd17f8fdec0>:1: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future
  data_raw.fillna(data_raw.median(axis = 0), axis=0 , inplace=True)
```

Заполним наиболее частыми значениями в столбцах (модой), соответствующих категориальным признакам

```
data_raw.WindDir9am.fillna(data_raw.WindDir9am.mode().iloc[0], inplace=True)
data_raw.WindDir3pm.fillna(data_raw.WindDir3pm.mode().iloc[0], inplace=True)
data_raw.WindGustDir.fillna(data_raw.WindGustDir.mode().iloc[0], inplace=True)
data_raw.RainTomorrow.fillna(data_raw.RainTomorrow.mode().iloc[0], inplace=True)
data_raw.RainToday.fillna(data_raw.RainToday.mode().iloc[0], inplace=True)
```

Проверим изменения

```
data_raw.isna().sum()
```

Location	0
MinTemp	0
MaxTemp	0
Rainfall	0
Evaporation	0
Sunshine	0
WindGustDir	0
WindGustSpeed	0
WindDir9am	0
WindDir3pm	0
WindSpeed9am	0
WindSpeed3pm	0
Humidity9am	0
Humidity3pm	0
Pressure9am	0
Pressure3pm	0
Cloud9am	0
Cloud3pm	0
Temp9am	0
Temp3pm	0
RainToday	0
RainTomorrow	0
dtype: int64	

▼ 5. Бинаризация номинальных признаков

Алгоритмы из библиотеки scikit-learn (почти) не умеют работать напрямую с категориальными признаками. Поэтому их вначале надо закодировать с помощью числовых признаков.

Заменяем текстовые значения на числовые

```
data_raw['Location'] = LabelEncoder().fit_transform(data_raw['Location'])
```

```
data_raw.head()
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	Humidity9am
0	2	13.4	22.9	0.6	4.6	8.4	W	44.0	W	WNW	...	71.0
1	2	7.4	25.1	0.0	4.6	8.4	WNW	44.0	NNW	WSW	...	44.0
2	2	12.9	25.7	0.0	4.6	8.4	WSW	46.0	W	WSW	...	38.0
3	2	9.2	28.0	0.0	4.6	8.4	NE	24.0	SE	E	...	45.0
4	2	17.5	32.3	1.0	4.6	8.4	W	41.0	ENE	NW	...	82.0

5 rows × 22 columns

Среди категориальных признаков есть бинарные и небинарные.

К бинарным относятся такие признаки, как RainToday и RainTomorrow. Заменим их значения на 0 и 1.

```
data_raw['RainToday'] = pd.factorize(data_raw['RainToday'])[0]
data_raw['RainTomorrow'] = pd.factorize(data_raw['RainTomorrow'])[0]
```

```
data_raw.head()
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	Humidity9am
0	2	13.4	22.9	0.6	4.6	8.4	W	44.0	W	WNW	...
1	2	7.4	25.1	0.0	4.6	8.4	WNW	44.0	NNW	WSW	...
2	2	12.9	25.7	0.0	4.6	8.4	WSW	46.0	W	WSW	...
3	2	9.2	28.0	0.0	4.6	8.4	NE	24.0	SE	E	...
4	2	17.5	32.3	1.0	4.6	8.4	W	41.0	ENE	NW	...

5 rows × 22 columns

Для категориальных (небинарных) признаках 'WindGustDir', 'WindDir9am', 'WindDir3pm' применим метод бинаризации (one-hot encoding)

```
enc = OneHotEncoder(drop='if_binary', sparse_output=False)
enc.fit(data_raw[['WindGustDir', 'WindDir9am', 'WindDir3pm']])
dummies = pd.DataFrame(enc.transform(data_raw[['WindGustDir', 'WindDir9am', 'WindDir3pm']]),
columns=enc.get_feature_names_out(), index=data_raw.index)
```

```
dummies.head()
```

	WindGustDir_E	WindGustDir_ENE	WindGustDir_ESE	WindGustDir_N	WindGustDir_NE	WindGustDir_NNE	WindGustDir_NNW	WindGustDir_NW
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 48 columns

Добавим эти dummy-столбцы к таблице и удалим исходные столбцы

```
data_raw = pd.concat((data_raw, dummies), axis=1).drop(['WindGustDir', 'WindDir9am', 'WindDir3pm'], axis=1)
```

```
data_raw.head()
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	...	WindDir
0	2	13.4	22.9	0.6	4.6	8.4	44.0	20.0	24.0	71.0	...
1	2	7.4	25.1	0.0	4.6	8.4	44.0	4.0	22.0	44.0	...
2	2	12.9	25.7	0.0	4.6	8.4	46.0	19.0	26.0	38.0	...
3	2	9.2	28.0	0.0	4.6	8.4	24.0	11.0	9.0	45.0	...
4	2	17.5	32.3	1.0	4.6	8.4	41.0	7.0	20.0	82.0	...

5 rows × 67 columns

Перед применением алгоритмов машинного обучения количественные признаки полезно нормализовать. Будем использовать стандартизацию - линейное преобразование, приводящее все значения к нулевому среднему и единичному стандартному отклонению

Выполним стандартизацию всех признаков.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data_raw)
data_raw = pd.DataFrame(scaler.transform(data_raw), columns=data_raw.columns, index=data_raw.index)
```

```
data_raw.describe()
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3
count	1.449310e+05	1.449310e+							
mean	-9.413045e-17	2.321884e-16	-1.041710e-15	5.569385e-17	-4.706523e-17	-3.137682e-18	2.529756e-17	7.922646e-17	4.549639e-
std	1.000003e+00	1.000003e+							
min	-1.672308e+00	-3.249143e+00	-3.956225e+00	-2.759223e-01	-1.837282e+00	-2.897399e+00	-2.619066e+00	-1.594663e+00	-2.160984e+
25%	-8.987521e-01	-7.039547e-01	-7.349634e-01	-2.759223e-01	-3.744123e-01	7.623944e-02	-6.857370e-01	-7.961568e-01	-6.521324e-
50%	1.545034e-02	-2.838006e-02	-8.505976e-02	-2.759223e-01	-1.549818e-01	1.487672e-01	-6.707177e-02	-1.117229e-01	4.426042e-
75%	8.593295e-01	7.257498e-01	7.061273e-01	-2.037297e-01	6.444864e-02	2.575588e-01	4.742603e-01	5.727110e-01	6.245878e-

▼ 6. Разбить данные на обучающую и тестовую выборки

Для предсказания дождя завтра будем использовать все входные признаки

```
X = data_raw.drop(['RainTomorrow'], axis=1)
y = data_raw['RainTomorrow']
```

Разобьем данные на обучающую и тестовую выборки в пропорции 3:1 (75% - обучающая выборка, 25% - тестовая):

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.25, random_state = 42)

N_train, _ = X_train.shape
N_test, _ = X_test.shape

N_train, N_test

(108698, 36233)
```

Интерпретируем все значения у как целые числа:

```
Y_test = Y_test.astype("int")
Y_train = Y_train.astype("int")
```

▼ 7. Используем классификатор ближайших соседей (kNN), вычислим ошибки на выборках.

Для начала запустим классификатор к-ближайших соседей с 3 соседями

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, Y_train)

Y_train_predict = model.predict(X_train)
Y_test_predict = model.predict(X_test)

RMSE_train = mean_squared_error(Y_train, Y_train_predict)**.5
RMSE_test = mean_squared_error(Y_test, Y_test_predict)**.5

RMSE_train, RMSE_test

(0.3380531490068957, 0.44518478833348574)
```

▼ 8. Подберем оптимальное значение гиперпараметра (к-ва ближайших соседей)

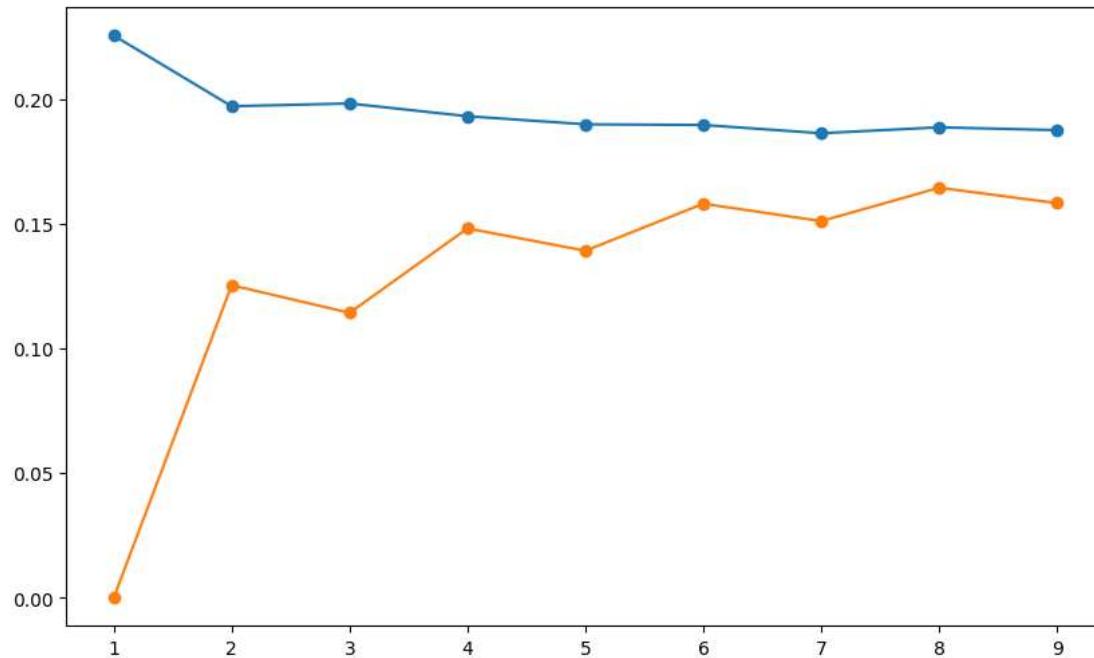
Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

Рассмотрим различное количество соседей. Наименьшая ошибка покажет нам наилучший выбор параметра k для метода.

```
model = KNeighborsClassifier()
error_train = []
error_test = []
for i in range(1, 10):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train, Y_train)
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)
    error_train.append(np.mean(test_predict != Y_test))
    error_test.append(np.mean(train_predict != Y_train))

plt.figure(figsize=(10,6))
plt.plot(range(1,10),error_train, linestyle='solid', marker='o')
plt.plot(range(1,10),error_test, linestyle='solid', marker='o')
```

[<matplotlib.lines.Line2D at 0x7e7c0ee7a770>]



Лучший результат в 16% достигается при n=10

Во всех случаях ошибка на тестовой выборке оказывалась больше, чем на тренировочной, что может свидетельствовать о некотором переобучении модели.

Подводя итог, можно сделать вывод, что "предобработка" данных была выполнена успешно и задача классификации решена.

▼ 9. Борьба с несбалансированностью классов

Ранее мы выяснили что в классах присутствует сильная несбалансированность

```
plt.pie(data_raw['RainTomorrow'].value_counts(), labels = ["No", "Yes"], colors = sns.color_palette("hls",4), autopct = '%1.1f%%')
pass
```



Интерпретируем все значения RainTomorrow как целые числа:

78.1%

```
RainTomorrow2 = data_raw["RainTomorrow"].astype("int")
data_raw.drop('RainTomorrow', axis=1, inplace=True)
data_raw = pd.concat((data_raw, RainTomorrow2), axis=1)
```



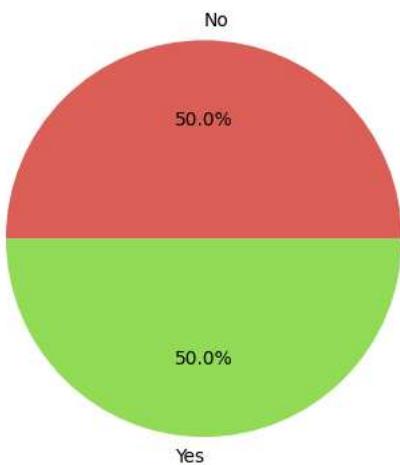
Попробуем исправить ситуацию выкинув часть значений из класса "No"

```
class_0 = int((data_raw[data_raw["RainTomorrow"] == 0])["RainTomorrow"].value_counts())
class_1 = int((data_raw[data_raw["RainTomorrow"] == 1])["RainTomorrow"].value_counts())
class_0 - class_1
```

81509

```
data_balanced = data_raw.copy()
data_balanced = data_balanced.drop(data_balanced.loc[data_balanced["RainTomorrow"] == 0].sample(n=class_0 - class_1).index)
```

```
plt.pie(data_balanced['RainTomorrow'].value_counts(), labels = ["No", "Yes"], colors = sns.color_palette("hls",4), autopct = '%1.1f%')
pass
```



Теперь мы уравняли количество данных по классам, попробуем обучить на них классификатор

```
X = data_balanced.drop("RainTomorrow", axis=1)
Y = data_balanced["RainTomorrow"]
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 42)
```

```
N_train, _ = X_train.shape
N_test, _ = X_test.shape
```

```
N_train, N_test
```

(47566, 15856)

```
model = KNeighborsClassifier(n_neighbors=9)
model.fit(X_train, Y_train)
Y_train_prediction = model.predict(X_train)
Y_test_prediction = model.predict(X_test)
```

```
RMSE_train = mean_squared_error(Y_train, Y_train_prediction)**.5
RMSE_test = mean_squared_error(Y_test, Y_test_prediction)**.5
RMSE_train, RMSE_test
```

(0.4652721318395574, 0.5333835222769154)

Матрица для тестовых данных

```
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_test_prediction))

precision    recall    f1-score   support

          0       0.70      0.76      0.73      7987
          1       0.73      0.67      0.70      7869

   accuracy                           0.72      15856
  macro avg       0.72      0.72      0.71      15856
weighted avg       0.72      0.72      0.71      15856
```

Таким образом после борьбы с несбалансированностью точность метода ближайших соседей ожидаемо упала. Однако класс 1 стал определяться намного лучше

▼ Общие выводы

В данной работе мы познакомились с датасетом weatherAUS.csv. Выяснили некоторые подробности о взаимосвязи температурных, и иных показателей на наличие/отсутствие дождя на завтра. Нашли и разрешили проблему с наличием пропущенных значений. Провели некоторую подготовку данных: векторизацию категориальных признаков и стандартизацию количественных. Для визуализации данных мы использовали библиотеку seaborn и построили диаграмму violinplot для всех параметров. В результате наша работа позволила успешно обучить модели для предсказания наличия/отсутствия дождя на завтра и получить хорошие метрики качества. Успешно обучили модели kNN, подобрали наиболее удачное количество соседей, немного поэкспериментировали. Также рассмотрели проблему несбалансированности данных.