University of Minho
School of Engineering

Intelligent Systems Associate Laboratory

Dados e Aprendizagem Automática

Feature Engineering

Decision Tree Pruning and Hyperparameter Tuning

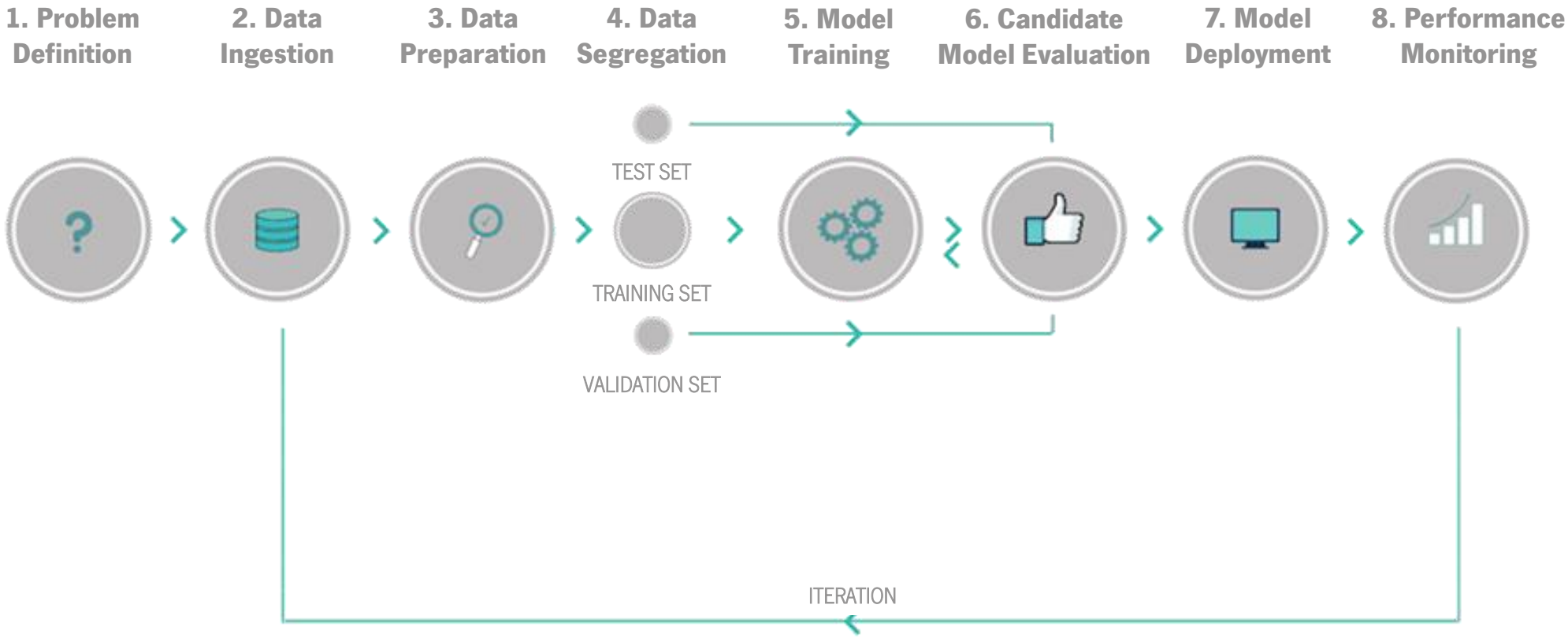
DAA @ MEI-1º/MiEI-4º – 1º Semestre

Bruno Fernandes, Dalila Alves, Filipa Ferraz, Victor Alves

Part V

Contents

- Feature Engineering
- Decision Tree
  - Hyperparameter Tuning with GridSearch
  - Pruning
- Hands On


1. Problem Definition
2. Data Ingestion
3. Data Preparation
4. Data Segregation
5. Model Training
6. Candidate Model Evaluation
7. Model Deployment
8. Performance Monitoring
TEST SET
TRAINING SET
VALIDATION SET
ITERATION

Feature Engineering

The Problem and the Data

Problem: development of a Machine Learning model capable of **predicting incidents** on road structure in the city of Guimarães

Dataset: table with information regarding the **incidents on the road** with 5000 entries and 13 features, including:

- *city_name*
- *magnitude_of_delay*
- *delay_in_seconds*
- *affected_roads*
- *record_date*
- *luminosity*
- *avg_temperature*
- *avg_atm_pressure*
- *avg_humidity*
- *avg_wind_speed*
- *avg_precipitation*
- *avg_rain*
- *incidents*

Exploratory Data Analysis


```
incid.columns
```

```
Index(['city_name', 'magnitude_of_delay', 'delay_in_seconds', 'affected_roads',
       'record_date', 'luminosity', 'avg_temperature', 'avg_atm_pressure',
       'avg_humidity', 'avg_wind_speed', 'avg_precipitation', 'avg_rain',
       'incidents'],
      dtype='object')
```

```
incid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   city_name           5000 non-null   object 
 1   magnitude_of_delay  5000 non-null   object 
 2   delay_in_seconds    5000 non-null   int64  
 3   affected_roads      4915 non-null   object 
 4   record_date         5000 non-null   object 
 5   luminosity          5000 non-null   object 
 6   avg_temperature     5000 non-null   float64
 7   avg_atm_pressure    5000 non-null   float64
 8   avg_humidity        5000 non-null   float64
 9   avg_wind_speed      5000 non-null   float64
 10  avg_precipitation   5000 non-null   float64
 11  avg_rain            5000 non-null   object 
 12  incidents           2972 non-null   object 
dtypes: float64(5), int64(1), object(7)
memory usage: 507.9+ KB
```

Exploratory Data Analysis

7

```
incid.head()
```

	city_name	magnitude_of_delay	delay_in_seconds	affected_roads	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_preci
0	Guimaraes	UNDEFINED	0	,	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	
1	Guimaraes	UNDEFINED	385	N101,	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	
2	Guimaraes	UNDEFINED	69	,	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	
3	Guimaraes	MAJOR	2297	N101,R206,N105,N101,N101,N101,N101,N101,N101,N...	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	
4	Guimaraes	UNDEFINED	0	N101,N101,N101,N101,N101,	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	

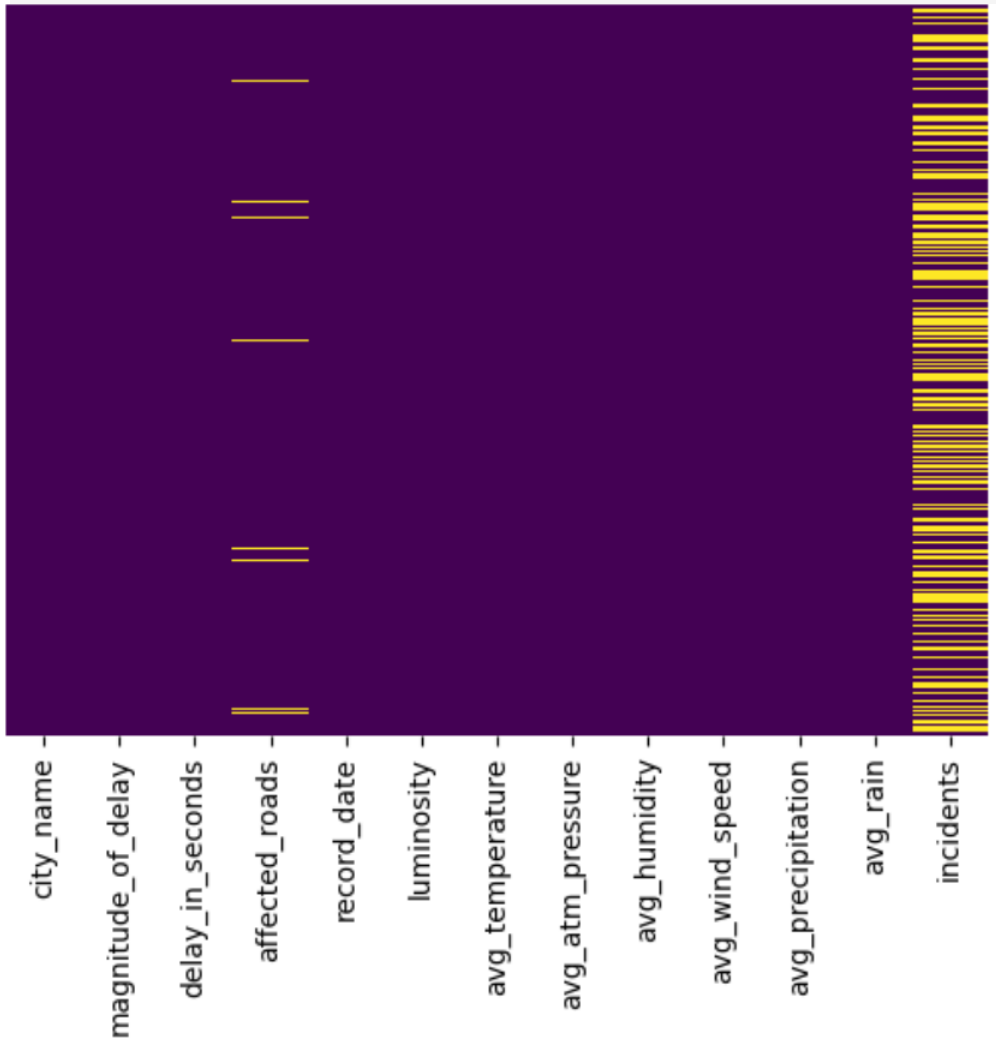
Feature Engineering

Handling Missing Data and Data Transformations

- Remove missing values, outliers, and unnecessary entries/features
- Check and impute null values
- Check imbalanced data
- Re-indexing and reformatting the data

Dealing with Missing Values


```
sns.heatmap(incid.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```




```
incid.isnull().sum()
```

```
city_name                0
magnitude_of_delay       0
delay_in_seconds         0
affected_roads          85
record_date              0
luminosity               0
avg_temperature          0
avg_atm_pressure         0
avg_humidity             0
avg_wind_speed           0
avg_precipitation        0
avg_rain                 0
incidents             2028
dtype: int64
```

```
incid['affected_roads'].head()
```

```
0                                                    ,
1                                                 N101,
2                                                    ,
3    N101,R206,N105,N101,N101,N101,N101,N101,N101,N...
4                              N101,N101,N101,N101,N101,
Name: affected_roads, dtype: object
```

Dealing with Missing Values

10

```
incid[incid['affected_roads'].isnull()]
```

	city_name	magnitude_of_delay	delay_in_seconds	affected_roads	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_precipitation	avg_rain	incidents
29	Guimaraes	UNDEFINED	64	NaN	2021-01-22 09:00	LIGHT	8.0	1012.0	91.0	4.0	0.0	Sem Chuva	Medium
76	Guimaraes	UNDEFINED	223	NaN	2021-01-29 08:00	LIGHT	11.0	1022.0	92.0	1.0	0.0	Sem Chuva	High
79	Guimaraes	MAJOR	80	NaN	2021-12-24 21:00	DARK	11.0	1004.0	92.0	0.0	0.0	Sem Chuva	NaN
91	Guimaraes	UNDEFINED	52	NaN	2021-03-02 13:00	LIGHT	13.0	1024.0	78.0	2.0	0.0	Sem Chuva	Low
109	Guimaraes	UNDEFINED	139	NaN	2021-12-27 13:00	LIGHT	15.0	1014.0	88.0	5.0	0.0	Sem Chuva	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Dealing with Missing Values

11

```
incid[incid['affected_roads'].isnull()]
```

	city_name	magnitude_of_delay	delay_in_seconds	affected_roads	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_precipitation	avg_rain	incidents
29	Guimaraes	UNDEFINED	64	NaN	2021-01-22 09:00	LIGHT	8.0	1012.0	91.0	4.0	0.0	Sem Chuva	Medium
76	Guimaraes	UNDEFINED	223	NaN	2021-01-29 08:00	LIGHT	11.0	1022.0	92.0	1.0	0.0	Sem Chuva	High
79	Guimaraes	MAJOR	80	NaN	2021-12-24 21:00	DARK	11.0	1004.0	92.0	0.0	0.0	Sem Chuva	NaN
91	Guimaraes	UNDEFINED	52	NaN	2021-03-02 13:00	LIGHT	13.0	1024.0	78.0	2.0	0.0	Sem Chuva	Low
109	Guimaraes	UNDEFINED	139	NaN	2021-12-27 13:00	LIGHT	15.0	1014.0	88.0	5.0	0.0	Sem Chuva	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Let's see how can missing values be handled. We will start by creating data copies to experiment different solutions:

```
incid1 = incid.copy()
incid2 = incid.copy()
```

Dealing with Missing Values

a) Dropping the column

```
incid1.drop(['affected_roads'], axis = 1, inplace = True)
incid1.head()
```

	city_name	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_precipitation	avg_rain	incidents
0	Guimaraes	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	0.0	Sem Chuva	NaN
1	Guimaraes	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	0.0	Sem Chuva	NaN
2	Guimaraes	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	0.0	Sem Chuva	Low
3	Guimaraes	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	0.0	Sem Chuva	Very_High
4	Guimaraes	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	0.0	Sem Chuva	High

Dealing with Missing Values

b) Filling the missing values with zero

```
incid2.fillna(0, inplace = True)
incid2.head()
```

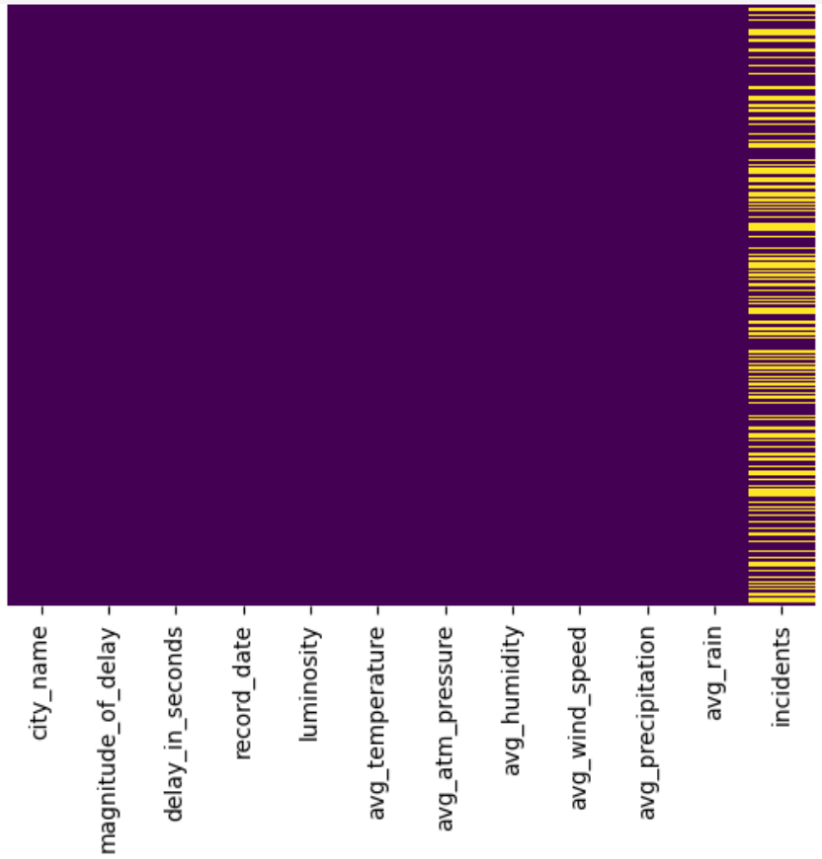
_delay	delay_in_seconds	affected_roads	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_precipitation	avg_rain	incidents
FINED	0	,	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	0.0	Sem Chuva	0
FINED	385	N101,	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	0.0	Sem Chuva	0
FINED	69	,	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	0.0	Sem Chuva	Low
MAJOR	2297	N101,R206,N105,N101,N101,N101,N101,N101,N101,N...	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	0.0	Sem Chuva	Very_High
FINED	0	N101,N101,N101,N101,N101,	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	0.0	Sem Chuva	High

Dealing with Missing Values

We will choose to drop the column since it does not bring added value to our goal:

```
incid.drop(['affected_roads'], axis = 1, inplace = True)
```

```
sns.heatmap(incid.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```




```
incid.isnull().sum()
```

```
city_name              0
magnitude_of_delay     0
delay_in_seconds       0
record_date            0
luminosity             0
avg_temperature        0
avg_atm_pressure       0
avg_humidity           0
avg_wind_speed         0
avg_precipitation      0
avg_rain               0
incidents           2028
dtype: int64
```

```
incid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   city_name           5000 non-null   object
 1   magnitude_of_delay  5000 non-null   object
 2   delay_in_seconds    5000 non-null   int64
 3   record_date         5000 non-null   object
 4   luminosity          5000 non-null   object
 5   avg_temperature     5000 non-null   float64
 6   avg_atm_pressure    5000 non-null   float64
 7   avg_humidity        5000 non-null   float64
 8   avg_wind_speed      5000 non-null   float64
 9   avg_precipitation   5000 non-null   float64
 10  avg_rain            5000 non-null   object
 11  incidents           2972 non-null   object
dtypes: float64(5), int64(1), object(6)
memory usage: 468.9+ KB
```

Dealing with Missing Values


```
incid.nunique()
```

```
city_name              1
magnitude_of_delay     3
delay_in_seconds    1186
record_date         5000
luminosity             3
avg_temperature       35
avg_atm_pressure      36
avg_humidity          83
avg_wind_speed        11
avg_precipitation      1
avg_rain               4
incidents              4
dtype: int64
```

The features *city_name* and *avg_precipitation* have only one value. We will start with *avg_precipitation*:

```
incid['avg_precipitation'].nunique()
```

```
1
```

```
incid['avg_precipitation'].describe()
```

```
count    5000.0
mean        0.0
std         0.0
min         0.0
25%         0.0
50%         0.0
75%         0.0
max         0.0
Name: avg_precipitation, dtype: float64
```

Dealing with Missing Values

Since 0 is the unique value of *avg_precipitation* and all entries have the same value, we will drop this feature:

```
incid.drop(['avg_precipitation'], axis = 1, inplace = True)
incid.head()
```

	city_name	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
0	Guimaraes	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	NaN
1	Guimaraes	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	NaN
2	Guimaraes	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	Low
3	Guimaraes	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	Very_High
4	Guimaraes	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	High

Handling Categoric Data

Even tough some models accept categoric data, we will explore how it can be transformed.

```
incid['city_name'].head()
```

```
0    Guimaraes
1    Guimaraes
2    Guimaraes
3    Guimaraes
4    Guimaraes
Name: city_name, dtype: object
```

The unique value of *city_name* is “Guimarães” - we can drop this feature as well:

```
incid.drop('city_name',axis=1,inplace=True)
incid.dropna(inplace=True)
```

Handling Categoric Data


```
print(incid['incidents'].value_counts())
```

```
incidents
High         1073
Low           718
Very_High     603
Medium        578
Name: count, dtype: int64
```

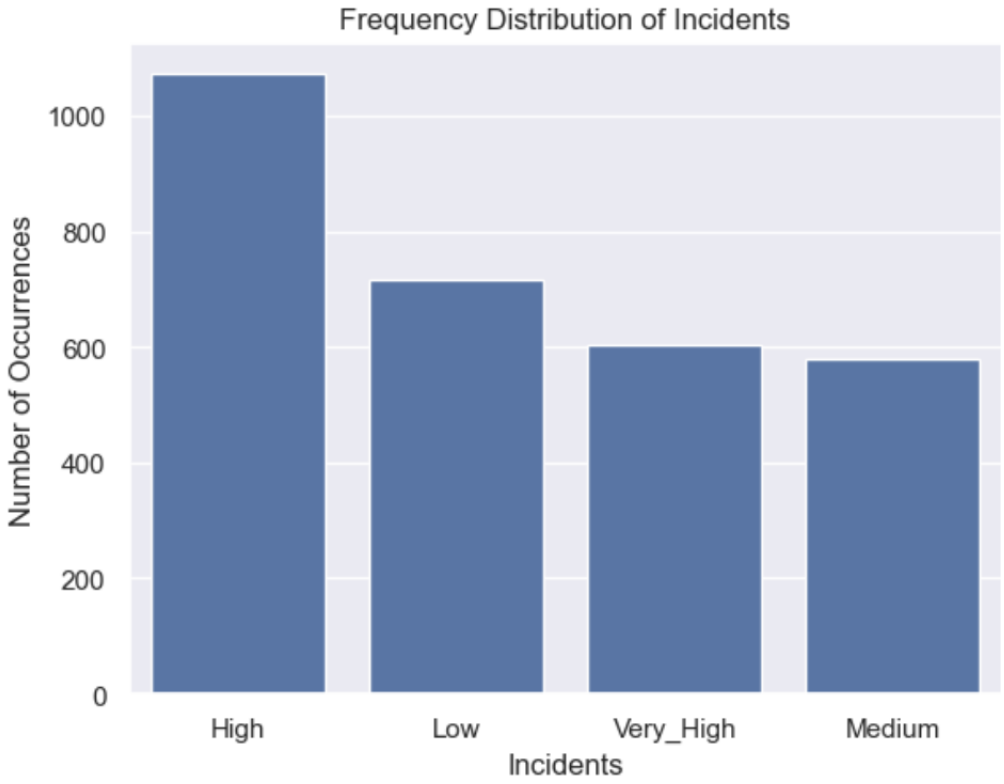
```
print(incid['incidents'].value_counts().count())
```

```
4
```

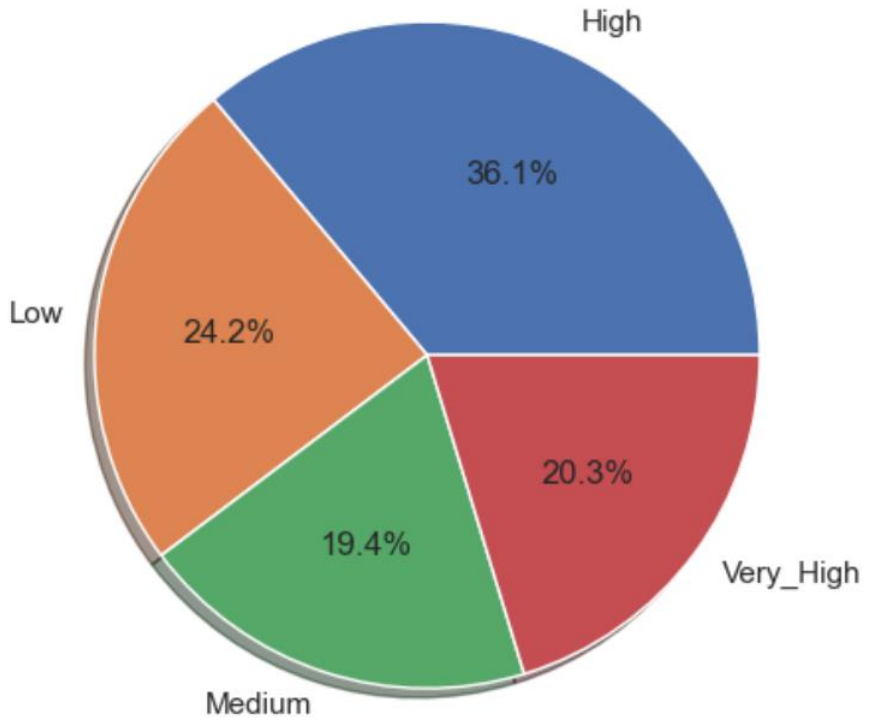
Let's visualize the distribution of values of feature *incidents*.

Handling Categoric Data


```
incidents_count = incid['incidents'].value_counts()
sns.set(style="darkgrid")
sns.barplot(x=incidents_count.index, y=incidents_count.values)
plt.title('Frequency Distribution of Incidents')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Incidents', fontsize=12)
plt.show()
```




```
labels = incid['incidents'].astype('category').cat.categories.tolist()
counts = incid['incidents'].value_counts()
sizes = [counts[var_cat] for var_cat in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True)
ax1.axis('equal')
plt.show()
```



Handling Categoric Data

We have several options of how to deal with qualitative data.

a) Replacing values

```
incid3=incid.copy()
incid3.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	Low
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	Very_High
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	High
5	UNDEFINED	0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	Medium
6	UNDEFINED	0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	Low

Label association: *None* - 0, *Low* - 1, *Medium* - 2, *High* - 3, *Very_High* - 4

```
replace_map = {'incidents': {'None': 0, 'Low': 1, 'Medium': 2, 'High': 3, 'Very_High': 4}}
```

Handling Categoric Data

We can create a replacement map in another way:

```
labels = incid3['incidents'].astype('category').cat.categories.tolist()
replace_map_comp = {'incidents' : {k: v for k,v in zip(labels, list(range(1, len(labels)+1)))}}
print(replace_map_comp)
```

```
{'incidents': {'High': 1, 'Low': 2, 'Medium': 3, 'Very_High': 4}}
```

```
incid3.replace(replace_map_comp, inplace=True)
incid3.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	2
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	4
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	1
5	UNDEFINED	0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	3
6	UNDEFINED	0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	2

```
print(incid3['incidents'].dtypes)
```

```
int64
```

Handling Categoric Data

b) Label encoding

```
incid4=incid.copy()
```

Similar to the previous examples, each string is assigned a number. Instead of replacing the values under the column *incidents*, we will create a new column for each label created. To complete the process, it is necessary to replicate for each label and then drop the column *incidents*.

```
print(incid4.dtypes)
```

```
magnitude_of_delay     object
delay_in_seconds        int64
record_date            object
luminosity             object
avg_temperature       float64
avg_atm_pressure      float64
avg_humidity          float64
avg_wind_speed        float64
avg_rain               object
incidents              object
dtype: object
```

```
incid4['None'] = np.where(incid4['incidents'].str.contains('None'), 1, 0)
incid4.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents	None
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	Low	0
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	Very_High	0
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	High	0
5	UNDEFINED	0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	Medium	0
6	UNDEFINED	0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	Low	0

Handling Categoric Data

b) Label encoding using `LabelEncoder()`

```
incid5=incid.copy()
incid6=incid.copy()

from sklearn.preprocessing import LabelEncoder

le_make = LabelEncoder()
incid5['incidents_code'] = le_make.fit_transform(incid6['incidents'])

incid5.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents	incidents_code
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	Low	1
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	Very_High	3
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	High	0
5	UNDEFINED	0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	Medium	2
6	UNDEFINED	0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	Low	1

A new column, *incidents_code*, is created with the labels assigned to feature *incidents*. The numeric values have been assigned randomly, as the crescent order does not apply to the meaning of the qualifying words.

Handling Categoric Data

c) One-Hot Encoding

This alternative creates a matrix with bits regarding each label.

```
incid7 = incid.copy()

from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
lb_results = lb.fit_transform(incid7['incidents'])
lb_results_df = pd.DataFrame(lb_results, columns=lb.classes_)

lb_results_df.head()
```

	High	Low	Medium	Very_High
0	0	1	0	0
1	0	0	0	1
2	1	0	0	0
3	0	0	1	0
4	0	1	0	0

Handling Categoric Data


```
result_df = pd.concat([incid7, lb_results_df], axis=1)
result_df.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents	High	Low	Medium	Very_High
2	UNDEFINED	69.0	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	Low	1.0	0.0	0.0	0.0
3	MAJOR	2297.0	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	Very_High	0.0	0.0	1.0	0.0
4	UNDEFINED	0.0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	High	0.0	1.0	0.0	0.0
5	UNDEFINED	0.0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	Medium	1.0	0.0	0.0	0.0
6	UNDEFINED	0.0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	Low	0.0	0.0	0.0	1.0

Handling Categoric Data

d) Binary Encoding

Similar to the previous technique, it creates a matrix of the states of the values, but this time with binary values. See the comparison between the techniques below:

Level	"Decimal encoding"	Binary encoding	One-Hot encoding
None	0	000	000001
Low	1	001	000010
Medium	2	010	000100
High	3	011	001000
Very_High	4	100	010000

This technique requires the `category_encoders` package to be installed:

```
pip install category_encoders
```

Handling Categoric Data


```
incid8 = incid.copy()

import category_encoders as ce

bencoder = ce.BinaryEncoder(cols=['incidents'])
df_binary = bencoder.fit_transform(incid8)

df_binary.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents_0	incidents_1	incidents_2
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	0	0	1
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	0	1	0
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	0	1	1
5	UNDEFINED	0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	1	0	0
6	UNDEFINED	0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	0	0	1

Handling Categoric Data

e) Backward Difference Encoding

The values are normalized in the range from -1 to 1.

```
incid9 = incid.copy()

bde = ce.BackwardDifferenceEncoder(cols=['incidents'])
df_bd = bde.fit_transform(incid9)

df_bd.head()
```

	intercept	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents_0	incidents_1	incidents_2
2	1	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	-0.75	-0.5	-0.25
3	1	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	0.25	-0.5	-0.25
4	1	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	0.25	0.5	-0.25
5	1	UNDEFINED	0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	0.25	0.5	0.75
6	1	UNDEFINED	0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	-0.75	-0.5	-0.25

Handling Categoric Data

f) Factorizing

This technique encodes the object as an enumerated type or categorical variable.

```
incid10 = incid.copy()
```

```
incid10['incidents'] = pd.factorize(incid10['incidents'])[0] + 1
incid10.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	1
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	2
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	3
5	UNDEFINED	0	2021-12-07 23:00	DARK	9.0	1015.0	94.0	0.0	Sem Chuva	4
6	UNDEFINED	0	2021-12-05 05:00	DARK	8.0	1026.0	87.0	1.0	Sem Chuva	1

We will choose to the factorize technique to keep going:

```
incid['incidents'] = pd.factorize(incid['incidents'])[0] + 1
```

Handling Categoric Data

Regarding the features *magnitude_of_delay*, *luminosity* and *avg_rain*, we will factorize for now:

```
incid['magnitude_of_delay'] = pd.factorize(incid['magnitude_of_delay'])[0] + 1
incid['luminosity'] = pd.factorize(incid['luminosity'])[0] + 1
incid['avg_rain'] = pd.factorize(incid['avg_rain'])[0] + 1

incid.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
2	1	69	2021-03-12 15:00	1	14.0	1025.0	64.0	0.0	1	1
3	2	2297	2021-09-29 09:00	1	15.0	1028.0	75.0	1.0	1	2
4	1	0	2021-06-13 11:00	1	27.0	1020.0	52.0	1.0	1	3
5	1	0	2021-12-07 23:00	2	9.0	1015.0	94.0	0.0	1	4
6	1	0	2021-12-05 05:00	2	8.0	1026.0	87.0	1.0	1	1

Handling Dates

There are several techniques to handle dates. We will explore just a few. Here can be consulted the `datetime` properties and methods.

```
incid11 = incid.copy()
incid11['record_date'].head()
```

```
2    2021-03-12 15:00
3    2021-09-29 09:00
4    2021-06-13 11:00
5    2021-12-07 23:00
6    2021-12-05 05:00
Name: record_date, dtype: object
```

We will convert the dates from *object* to *datetime*, specifying the format we want:

```
incid11['record_date'] = pd.to_datetime(incid11['record_date'], format = '%Y-%m-%d %H:%M', errors='coerce')
assert incid11['record_date'].isnull().sum() == 0, 'missing record date'
incid11['record_date'].head()
```

```
2   2021-03-12 15:00:00
3   2021-09-29 09:00:00
4   2021-06-13 11:00:00
5   2021-12-07 23:00:00
6   2021-12-05 05:00:00
Name: record date, dtype: datetime64[ns]
```

Handling Dates

We can extract parts of the date and create new columns:

```
incid11['record_date_year'] = incid11['record_date'].dt.year
incid11['record_date_month'] = incid11['record_date'].dt.month
incid11['record_date_day'] = incid11['record_date'].dt.day
incid11['record_date_hour'] = incid11['record_date'].dt.hour
incid11['record_date_minute'] = incid11['record_date'].dt.minute
```

```
incid11.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents	record_date_year	record_date_month	rec
2	1	69	2021-03-12 15:00:00	1	14.0	1025.0	64.0	0.0	1	1	2021	3	
3	2	2297	2021-09-29 09:00:00	1	15.0	1028.0	75.0	1.0	1	2	2021	9	
4	1	0	2021-06-13 11:00:00	1	27.0	1020.0	52.0	1.0	1	3	2021	6	
5	1	0	2021-12-07 23:00:00	2	9.0	1015.0	94.0	0.0	1	4	2021	12	
6	1	0	2021-12-05 05:00:00	2	8.0	1026.0	87.0	1.0	1	1	2021	12	

Handling Dates


```
incid11.nunique()
```

```
magnitude_of_delay       3
delay_in_seconds      1167
record_date           2972
luminosity               3
avg_temperature         34
avg_atm_pressure        34
avg_humidity            80
avg_wind_speed          11
avg_rain                 4
incidents                4
record_date_year         1
record_date_month       11
record_date_day         31
record_date_hour        24
record_date_minute       1
dtype: int64
```

Since the *year* and the *minute* have only one value, we will drop them.

```
incid11.drop('record_date_year',axis=1,inplace=True)
incid11.drop('record_date_minute',axis=1,inplace=True)
incid11.drop('record_date',axis=1,inplace=True)
incid11.dropna(inplace=True)
```

Handling Dates


```
incid11.head()
```

	magnitude_of_delay	delay_in_seconds	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents	record_date_month	record_date_day	record_date_hour
2	1	69	1	14.0	1025.0	64.0	0.0	1	1	3	12	15
3	2	2297	1	15.0	1028.0	75.0	1.0	1	2	9	29	9
4	1	0	1	27.0	1020.0	52.0	1.0	1	3	6	13	11
5	1	0	2	9.0	1015.0	94.0	0.0	1	4	12	7	23
6	1	0	2	8.0	1026.0	87.0	1.0	1	1	12	5	5

Other methods to deal with dates:

```
incid12 = incid.copy()
```

```
incid12['record_date'] = pd.to_datetime(incid12['record_date'])
incid12.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2972 entries, 2 to 4995
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   magnitude_of_delay  2972 non-null   int64
 1   delay_in_seconds    2972 non-null   int64
 2   record_date         2972 non-null   datetime64[ns]
 3   luminosity          2972 non-null   int64
 4   avg_temperature     2972 non-null   float64
 5   avg_atm_pressure    2972 non-null   float64
 6   avg_humidity        2972 non-null   float64
 7   avg_wind_speed      2972 non-null   float64
 8   avg_rain            2972 non-null   int64
 9   incidents           2972 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(5)
memory usage: 319.9 KB
```

```
incid12['record_date'].head()
```

```
2   2021-03-12 15:00:00
3   2021-09-29 09:00:00
4   2021-06-13 11:00:00
5   2021-12-07 23:00:00
6   2021-12-05 05:00:00
Name: record_date, dtype: datetime64[ns]
```

Handling Dates


```
import datetime
today = datetime.datetime.today()
today
```

```
datetime.datetime(2024, 10, 14, 17, 50, 16, 977144)
```

```
today - incid12['record_date']
```

```
2      1312 days 02:50:16.977144
3      1111 days 08:50:16.977144
4      1219 days 06:50:16.977144
5      1041 days 18:50:16.977144
6      1044 days 12:50:16.977144
                  ...
4991   1115 days 23:50:16.977144
4992   1254 days 06:50:16.977144
4993   1209 days 01:50:16.977144
4994   1208 days 19:50:16.977144
4995   1280 days 17:50:16.977144
Name: record_date, Length: 2972, dtype: timedelta64[ns]
```

```
(today - incid12['record_date']).dt.days
```

```
2       1312
3       1111
4       1219
5       1041
6       1044
        ...
4991    1115
4992    1254
4993    1209
4994    1208
4995    1280
Name: record_date, Length: 2972, dtype: int64
```

```
incid12['day'] = incid12['record_date'].dt.day
incid12['month'] = incid12['record_date'].dt.month
incid12['hour'] = incid12['record_date'].dt.hour
incid12['time'] = incid12['record_date'].dt.time
incid12.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents	day	month	hour	time
2	1	69	2021-03-12 15:00:00	1	14.0	1025.0	64.0	0.0	1	1	12	3	15	15:00:00
3	2	2297	2021-09-29 09:00:00	1	15.0	1028.0	75.0	1.0	1	2	29	9	9	09:00:00
4	1	0	2021-06-13 11:00:00	1	27.0	1020.0	52.0	1.0	1	3	13	6	11	11:00:00
5	1	0	2021-12-07 23:00:00	2	9.0	1015.0	94.0	0.0	1	4	7	12	23	23:00:00

Handling Dates


```
incid['record_date'] = pd.to_datetime(incid['record_date'], format = '%Y-%m-%d %H:%M', errors='coerce')
```

```
incid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2972 entries, 2 to 4995
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   magnitude_of_delay  2972 non-null   int64
 1   delay_in_seconds    2972 non-null   int64
 2   record_date         2972 non-null   datetime64[ns]
 3   luminosity          2972 non-null   int64
 4   avg_temperature     2972 non-null   float64
 5   avg_atm_pressure    2972 non-null   float64
 6   avg_humidity        2972 non-null   float64
 7   avg_wind_speed      2972 non-null   float64
 8   avg_rain            2972 non-null   int64
 9   incidents           2972 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(5)
memory usage: 319.9 KB
```

```
incid.head()
```

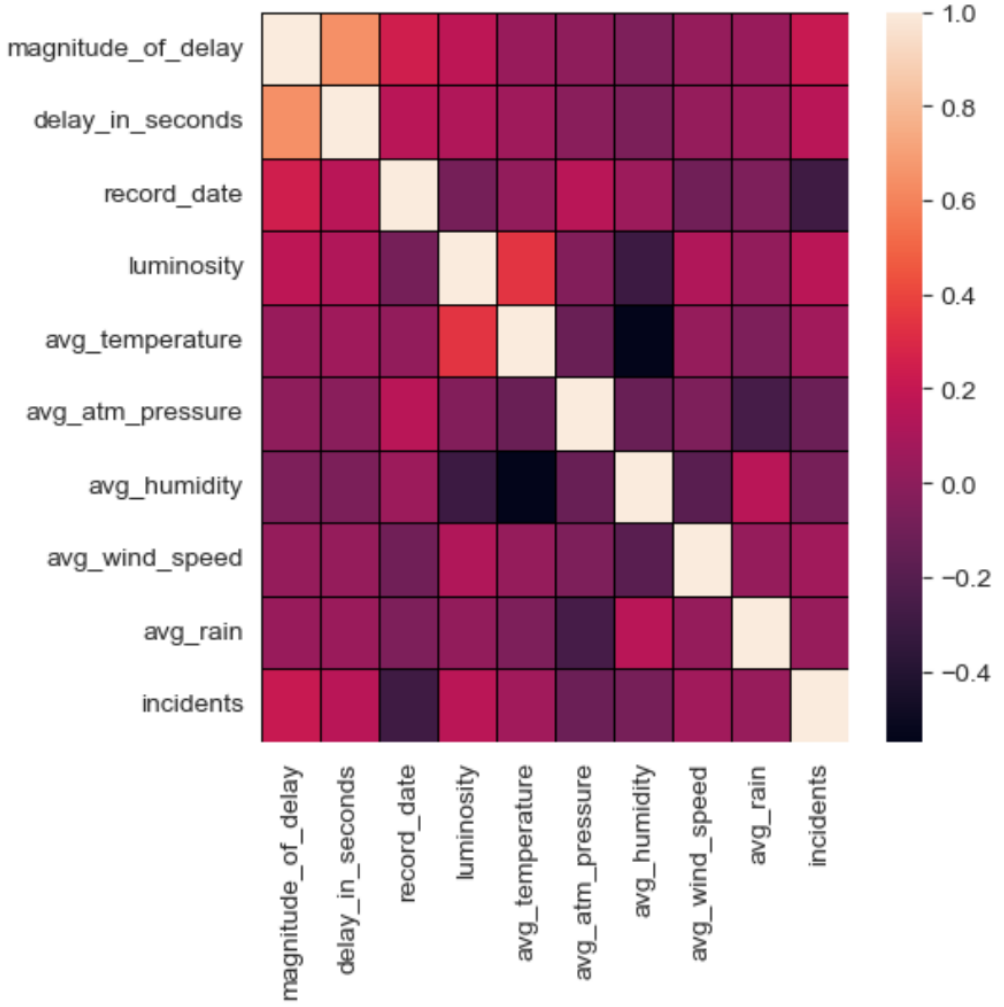
	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
2	1	69	2021-03-12 15:00:00	1	14.0	1025.0	64.0	0.0	1	1
3	2	2297	2021-09-29 09:00:00	1	15.0	1028.0	75.0	1.0	1	2
4	1	0	2021-06-13 11:00:00	1	27.0	1020.0	52.0	1.0	1	3
5	1	0	2021-12-07 23:00:00	2	9.0	1015.0	94.0	0.0	1	4
6	1	0	2021-12-05 05:00:00	2	8.0	1026.0	87.0	1.0	1	1

Decision Tree

EDA

Before approaching the model, we need to analyze the data a search for relations between features:

```
fig = plt.figure(figsize = (5,5))
incidents_corr = incid.corr( method = 'pearson')
sns.heatmap(incidents_corr, linecolor='black', linewidths=0.5)
```



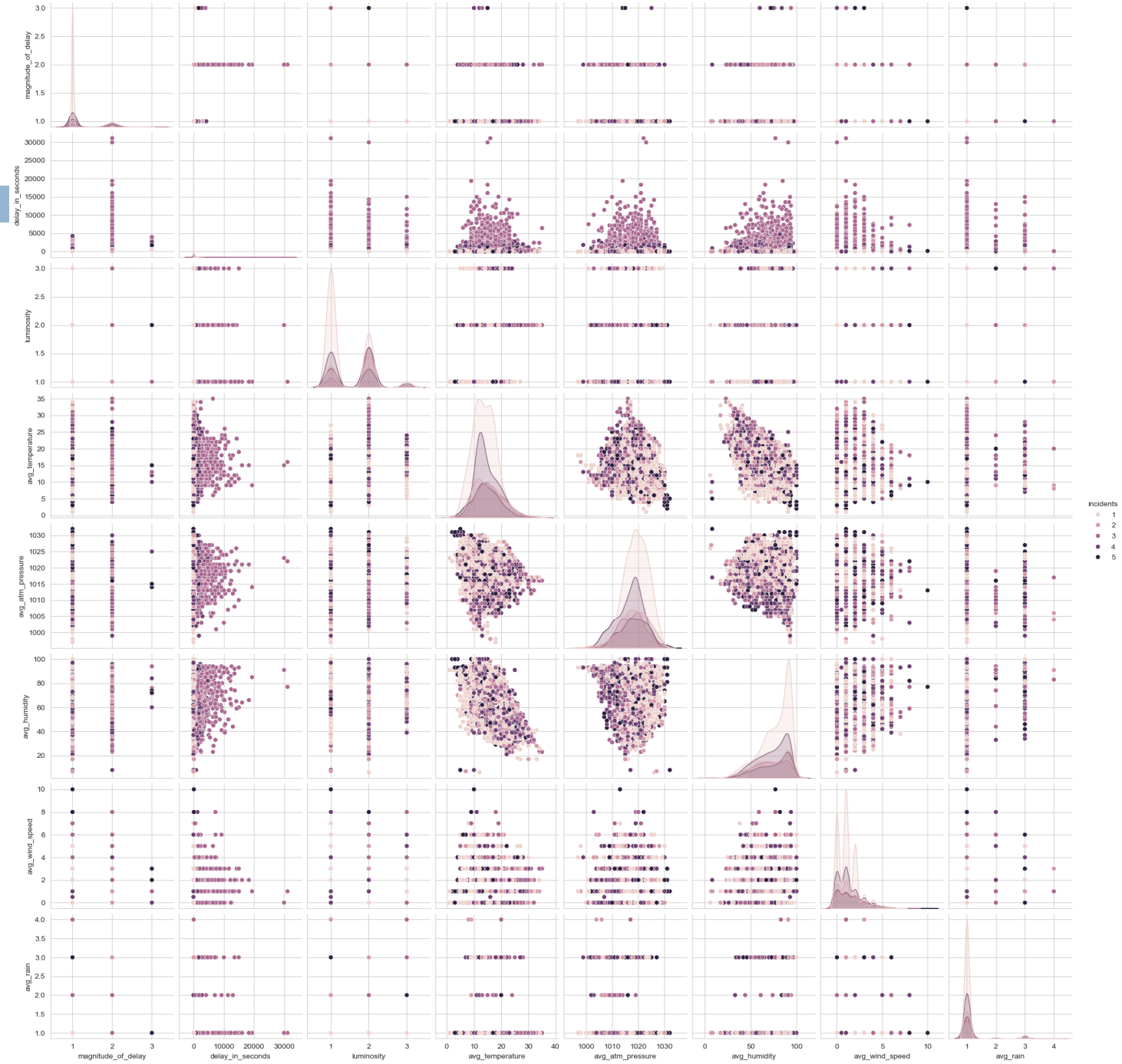
Highlighted relations:

- magnitude_of_delay and *delay_in_seconds*
- *magnitude_of_delay* and *record_date*
- *avg_humidity* and *luminosity*

EDA

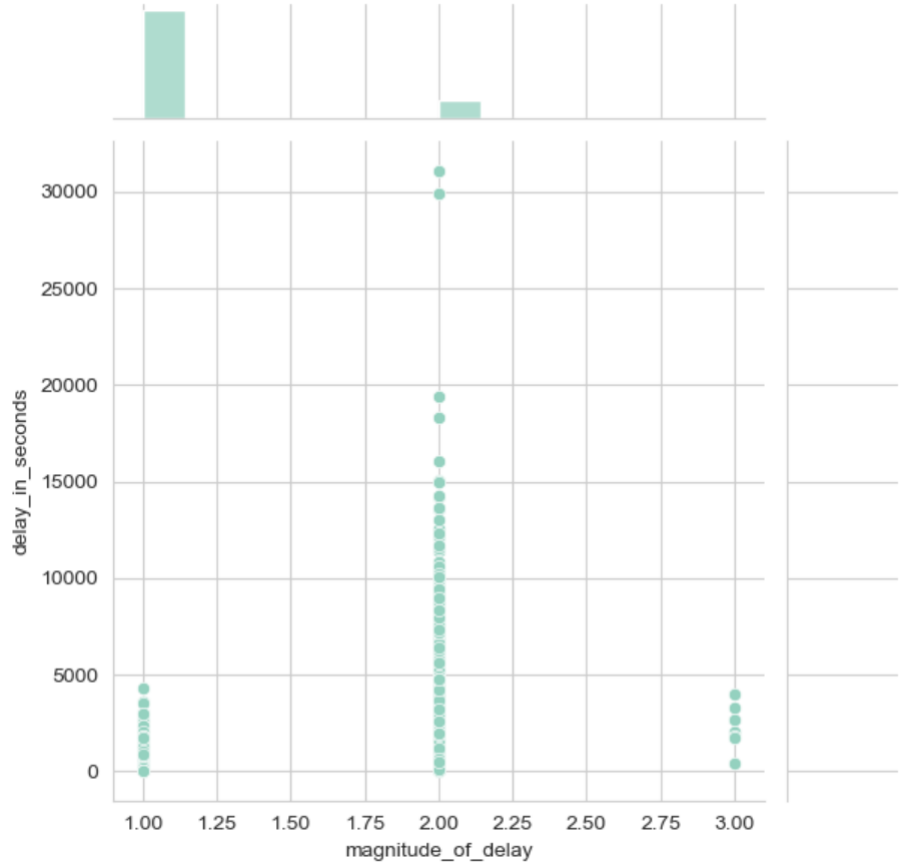

```
sns_plot = sns.pairplot(incid, hue="incidents")
sns_plot.savefig('pp.png')
```

It's difficult to analyze the relationship between all the features. Let's create joint plots between the features that show a relationship.

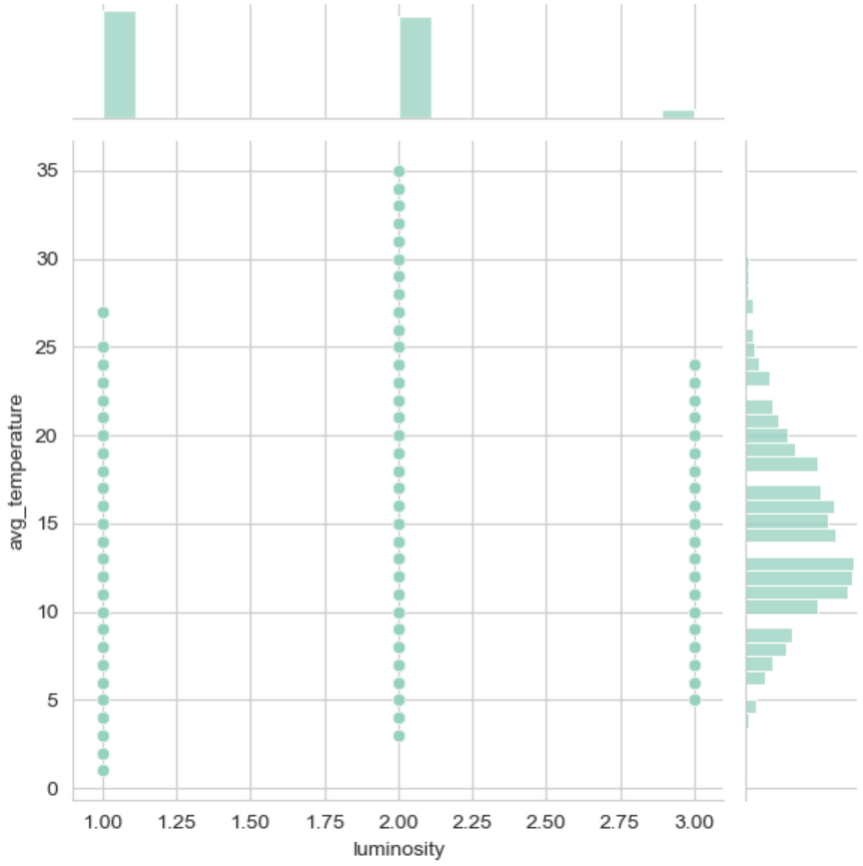


EDA


```
sns.set_palette("GnBu_d")
sns.set_style('whitegrid')
sns.jointplot(x='magnitude_of_delay', y='delay_in_seconds', data=incid)
```

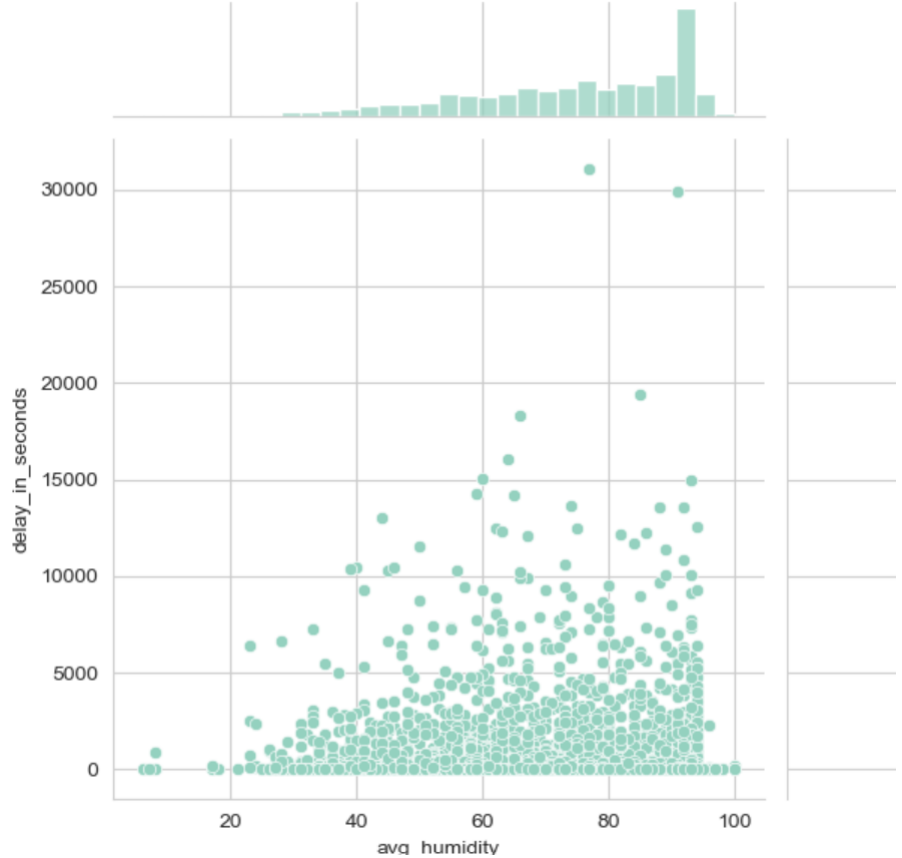



```
sns.jointplot(x='luminosity', y='avg_temperature', data=incid)
```

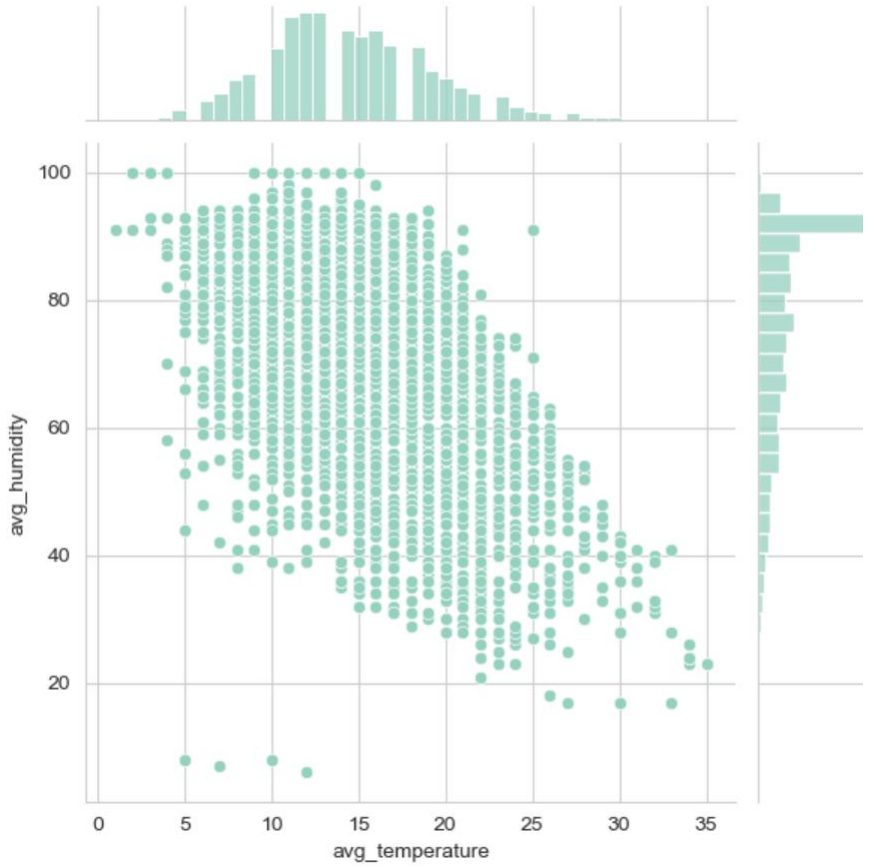


EDA


```
sns.jointplot(x='avg_humidity', y='delay_in_seconds', data=incid)
```



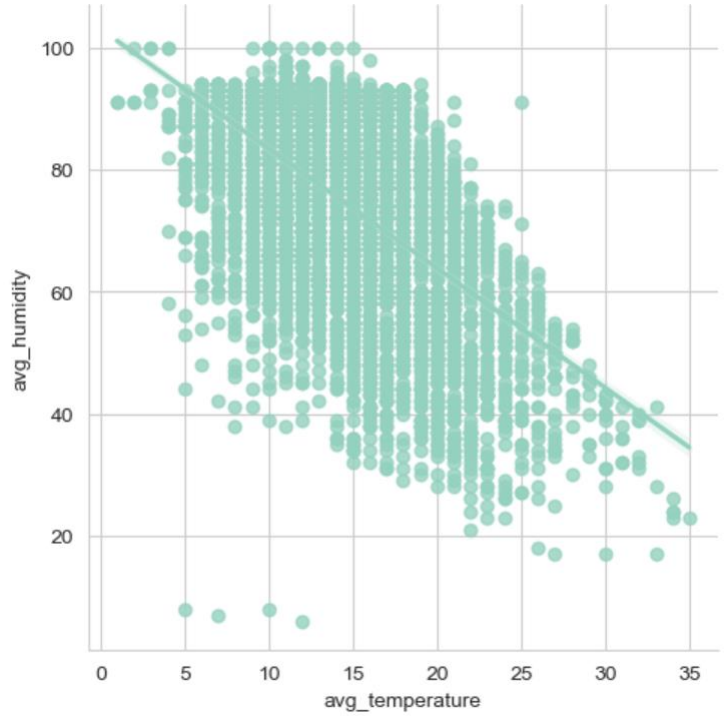

```
sns.jointplot(x='avg_temperature', y='avg_humidity', data=incid)
```



EDA

There seems to be a relationship between *avg_temperature* and *avg_humidity*:

```
sns.lmplot(x='avg_temperature', y='avg_humidity', data=incid)
```



Decision Tree Classifier

Let's now begin to train our model. The target are the *incidents* so we will implement a DT Classifier model. The feature *record_date* will not be considered since it is not relevant to the context.

```
from sklearn.model_selection import train_test_split
```

```
X = incid.drop(['record_date', 'incidents'], axis=1)
y = incid['incidents']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=2022)
```

Decision Tree Classifier


```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
                                    min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
                                    min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0, monotonic_cst=None)
```

```
dt_model = DecisionTreeClassifier(random_state=2022)
```

```
dt_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=2022)
```

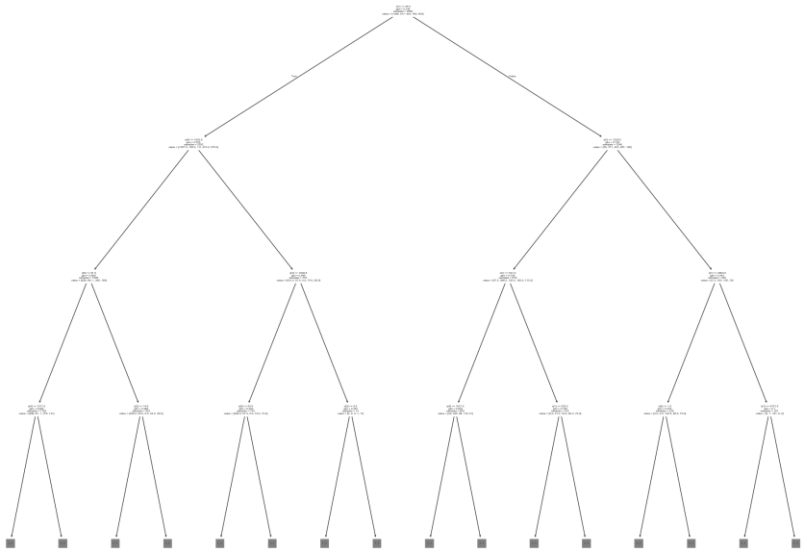
Plot the resultant tree, save to figure or export to text:

```
fig = plt.figure(figsize=(25, 20))
tree.plot_tree(dt_model, max_depth = 3)
plt.show()

fig.savefig("dt_plot.png")
```

```
text_representation = tree.export_text(dt_model)
print(text_representation)

with open("dt_text.log", "w") as fout:
    fout.write(text_representation)
```



Decision Tree Classifier

Evaluating the model:

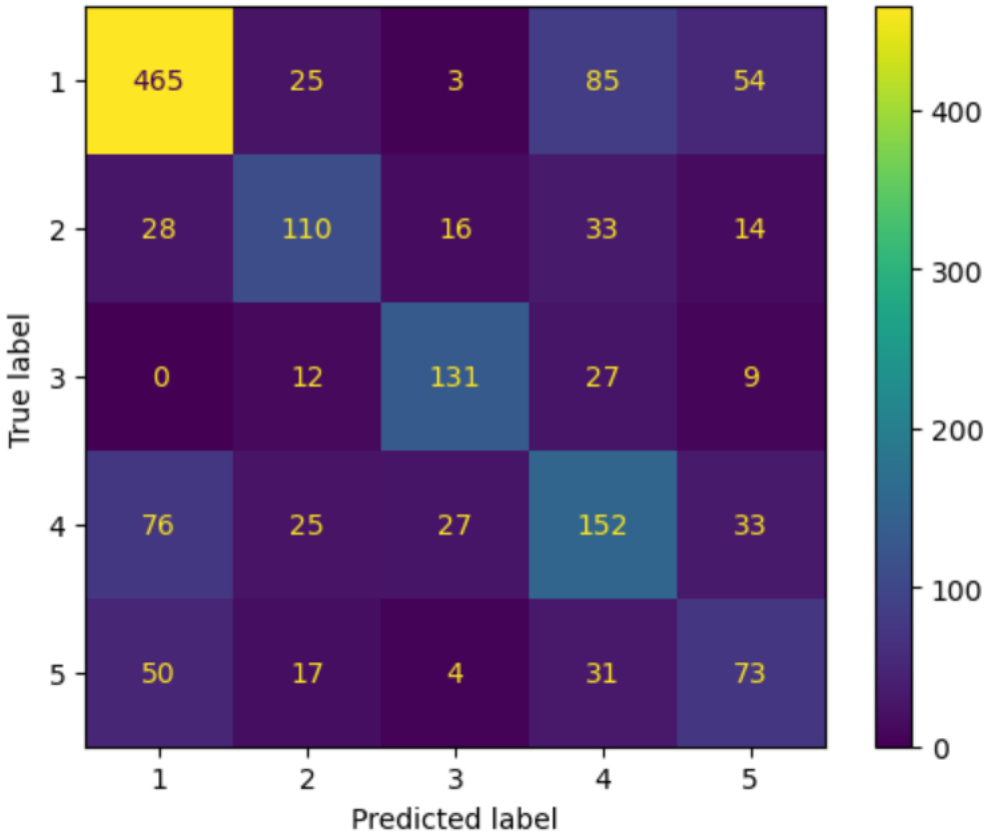
```
dt_predictions = dt_model.predict(X_test)
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
print(classification_report(y_test, dt_predictions))
```

```
              precision    recall  f1-score   support

           1       0.75      0.74      0.74       632
           2       0.58      0.55      0.56       201
           3       0.72      0.73      0.73       179
           4       0.46      0.49      0.47       313
           5       0.40      0.42      0.41       175

    accuracy                           0.62      1500
   macro avg       0.58      0.58      0.58      1500
weighted avg       0.62      0.62      0.62      1500
```

```
ConfusionMatrixDisplay.from_predictions(y_test, dt_predictions)
plt.show()
```



Parameters and Hyperparameters

Model Parameter: an (internal) configuration variable of the model whose value is estimated from training data, i.e., the value is not set manually. Some examples are:

- *Weights* in Artificial Neural Networks
- *Support vectors* in Support Vector Machines

Model Hyperparameter: an (external) configuration variable of the model whose value can be set manually. It is difficult to know the best value of each hyperparameter in advance. Tuning a model consists of finding the best (or at least a good) configuration of hyperparameters. Examples are:

- *Optimizer* and *learning rate* in Artificial Neural Networks
- *C* and *gamma* in Support Vector Machines
- *Quality measure* and *pruning method* in Decision Trees

Hyperparamenter Tuning with GridSearch

GridSearch in practice

- Finding the right parameters is a tricky business;
- The idea of creating a "grid" of parameters and trying all possible combinations is called GridSearch;
- This method is so common that Scikit-learn has this functionality built in with GridSearchCV ("CV" stands for "Cross Validation");
- GridSearchCV takes a dictionary describing the parameters to be tried and the model to be trained.
- The parameter grid is defined as a dictionary where the keys are the parameters, and the values are the settings to be tested.
- GridSearchecv is a meta-estimator;
- It takes an estimator like DTC and creates a new estimator that behaves the same (like a classifier);
- You should add `refit=True` and set `verbose` to any number you want (*verbose* means the text output describing the process).

```
sklearn.model_selection.GridSearchCV(estimator, param_grid, , scoring=None, n_jobs=None,
            refit=True, cv=None, verbose=0, pre_dispatch='2n_jobs', error_score=nan,
            return_train_score=False)
```

```
from sklearn.model_selection import GridSearchCV
```

Hyperparamenter Tuning with GridSearch

Depth of the tree and number of leaves obtained from the previous model:

```
print(dt_model.get_depth())
print(dt_model.get_n_leaves())
```

```
24
1189
```

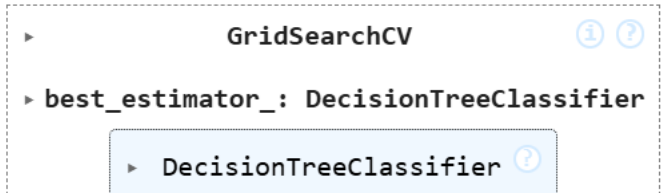
Hyperparameters to test: *criterion* and *max_depth*

```
param_grid_dt = {'criterion': ['gini', 'entropy'], 'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
estimator_dt = DecisionTreeClassifier(random_state=2022)
grid_dt = GridSearchCV(estimator_dt, param_grid_dt, refit=True, verbose=2)
```

The fit:

- Runs the same loop with CV to find the best combination of parameters;
- Once it has the best combination, it runs the fit again on all the data passed to the fit (without CV) to buils a single new model using the best parameter setting.

```
grid_dt.fit(X_train, y_train)
[CV] END ....................criterion=entropy, max_depth=10; total time=   0.0s
```



Hyperparamenter Tuning with GridSearch

Evaluating the model:

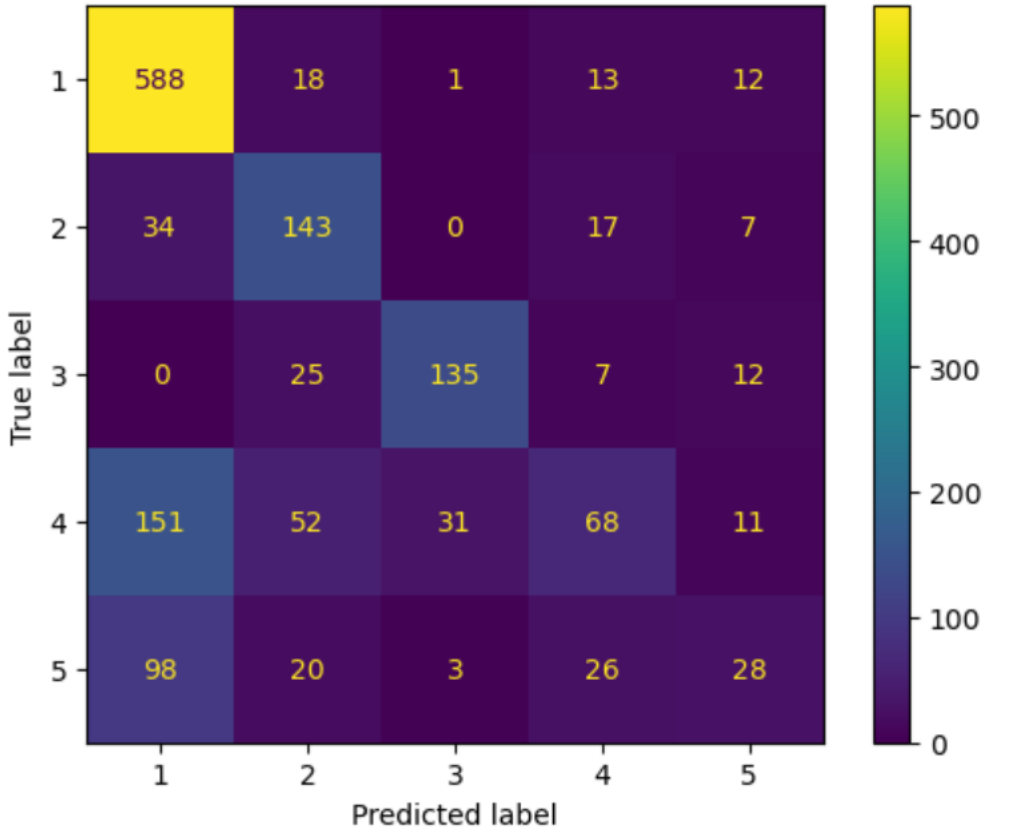
```
print(classification_report(y_test, grid_dt.predict(X_test)))
```

```
              precision    recall  f1-score   support

           1       0.68      0.93      0.78       632
           2       0.55      0.71      0.62       201
           3       0.79      0.75      0.77       179
           4       0.52      0.22      0.31       313
           5       0.40      0.16      0.23       175

    accuracy                           0.64      1500
   macro avg       0.59      0.55      0.54      1500
weighted avg       0.61      0.64      0.60      1500
```

```
ConfusionMatrixDisplay.from_predictions(y_test, grid_dt.predict(X_test))
plt.show()
```



Decision Tree Pruning

The DT is pruned by replacing an entire subtree with a leaf node. If the expected error rate in the subtree is greater than that of the single leaf, it is replaced.

When to apply:

- If the node becomes very small, do not split further;
- Minimum error (CV) pruning without early stopping is a good technique;
- Build a full depth tree and work backwards, using a statistical test at each stage;
- Prune an inner node and raise the subtree below it one level.

Maximum depth of the tree:

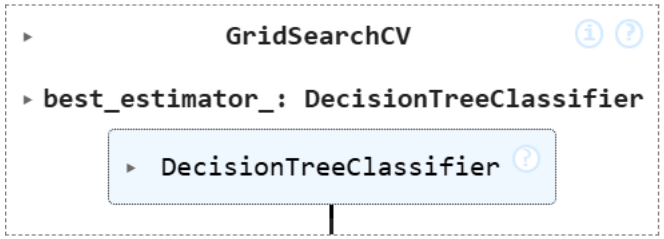
```
max_depth = dt_model.get_depth()
max_depth
```

```
24
```

Pré-pruning: *max_depth*

```
param_grid_pru = {'max_depth': [max_depth for max_depth in range(1, max_depth + 1)]}
estimator_pru = DecisionTreeClassifier(random_state=2022)
max_depth_gs = GridSearchCV(estimator_pru, param_grid_pru)
```

```
max_depth_gs.fit(X_train, y_train)
```



Decision Tree Pruning

Best parameters and estimator configuration:

```
max_depth_gs.best_params_
```

```
{'max_depth': 6}
```

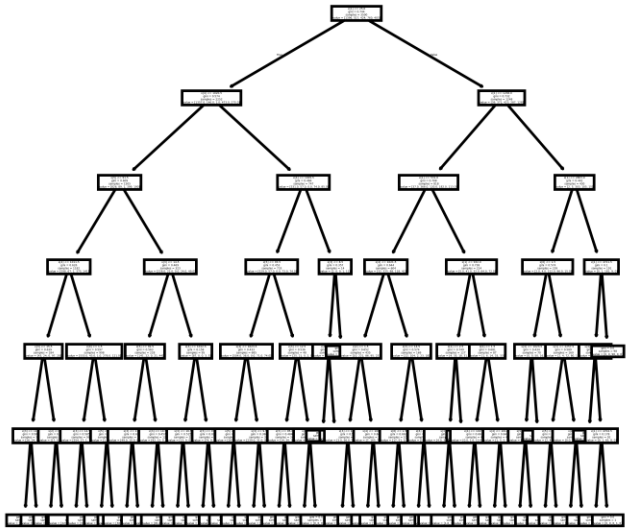
```
max_depth_tree = max_depth_gs.best_estimator_
print(max_depth_tree)
```

```
DecisionTreeClassifier(max_depth=6, random_state=2022)
```

Plot the tree:

```
fig = plt.figure(figsize=(4, 4), dpi=1000)
tree.plot_tree(max_depth_tree)
plt.show()

fig.savefig("dt_plot_md.png")
```



Decision Tree Pruning

Evaluating the model:

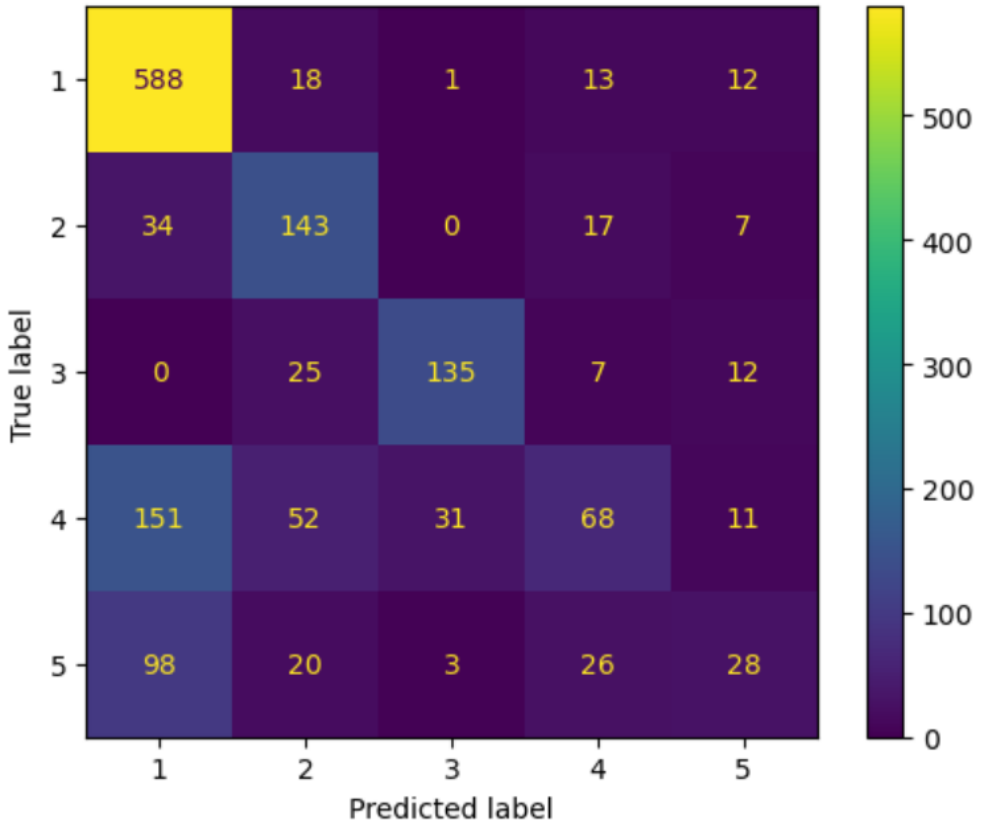
```
print(classification_report(y_test, max_depth_tree.predict(X_test)))
```

```
              precision    recall  f1-score   support

           1       0.68      0.93      0.78       632
           2       0.55      0.71      0.62       201
           3       0.79      0.75      0.77       179
           4       0.52      0.22      0.31       313
           5       0.40      0.16      0.23       175

    accuracy                           0.64      1500
   macro avg       0.59      0.55      0.54      1500
weighted avg       0.61      0.64      0.60      1500
```

```
ConfusionMatrixDisplay.from_predictions(y_test, max_depth_tree.predict(X_test))
plt.show()
```



Decision Tree Pruning

Prost-pruning: *ccp_alpha*

```
ccp_alphas = dt_model.cost_complexity_pruning_path(X_train, y_train)["ccp_alphas"]
```

```
ccp_alphas
```

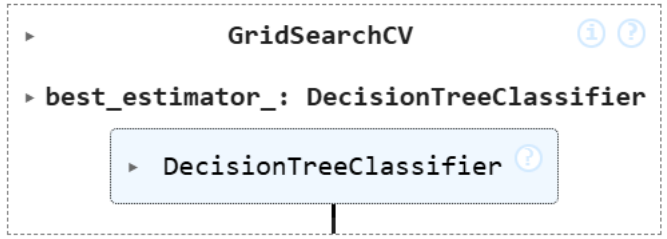
```
array([0.00000000e+00, 1.90476190e-05, 2.38095238e-05, 3.78151261e-05,
       4.76190476e-05, 6.66666667e-05, 7.14285714e-05, 7.14285714e-05,
       7.61904762e-05, 7.61904762e-05, 8.57142857e-05, 9.52380952e-05,
```

```
estimator_pru.get_params().keys()
```

```
dict_keys(['ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_features', 'max_le
af_nodes', 'min_impurity_decrease', 'min_samples_leaf', 'min_samples_split', 'min_weight_
fraction_leaf', 'monotonic_cst', 'random_state', 'splitter'])
```

```
param_grid_pru2 = {'ccp_alpha': [alpha for alpha in ccp_alphas]}
estimator_pru2 = DecisionTreeClassifier(random_state=2022)
ccp_alpha_gs = GridSearchCV(estimator_pru2, param_grid_pru2)
```

```
ccp_alpha_gs.fit(X_train, y_train)
```



Best parameters:

```
ccp_alpha_gs.best_params_
```

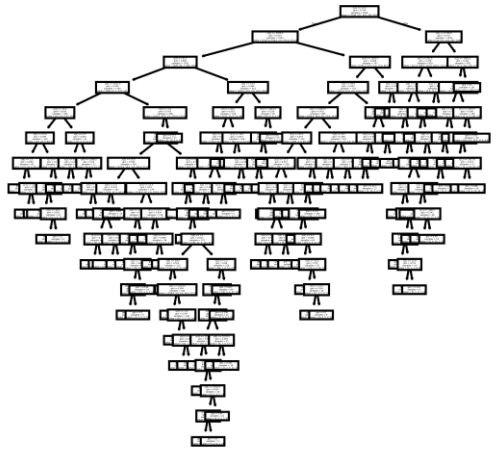
```
{'ccp_alpha': 0.0008316875137151618}
```

```
best_ccp_alpha_tree = ccp_alpha_gs.best_estimator_
print(best_ccp_alpha_tree)
```

```
DecisionTreeClassifier(ccp_alpha=0.0008316875137151618, random_state=2022)
```

```
fig = plt.figure(figsize=(4, 4), dpi=1000)
tree.plot_tree(best_ccp_alpha_tree)
plt.show()

fig.savefig("dt_plot_alpha.png")
```



Decision Tree Pruning

Evaluating the model:

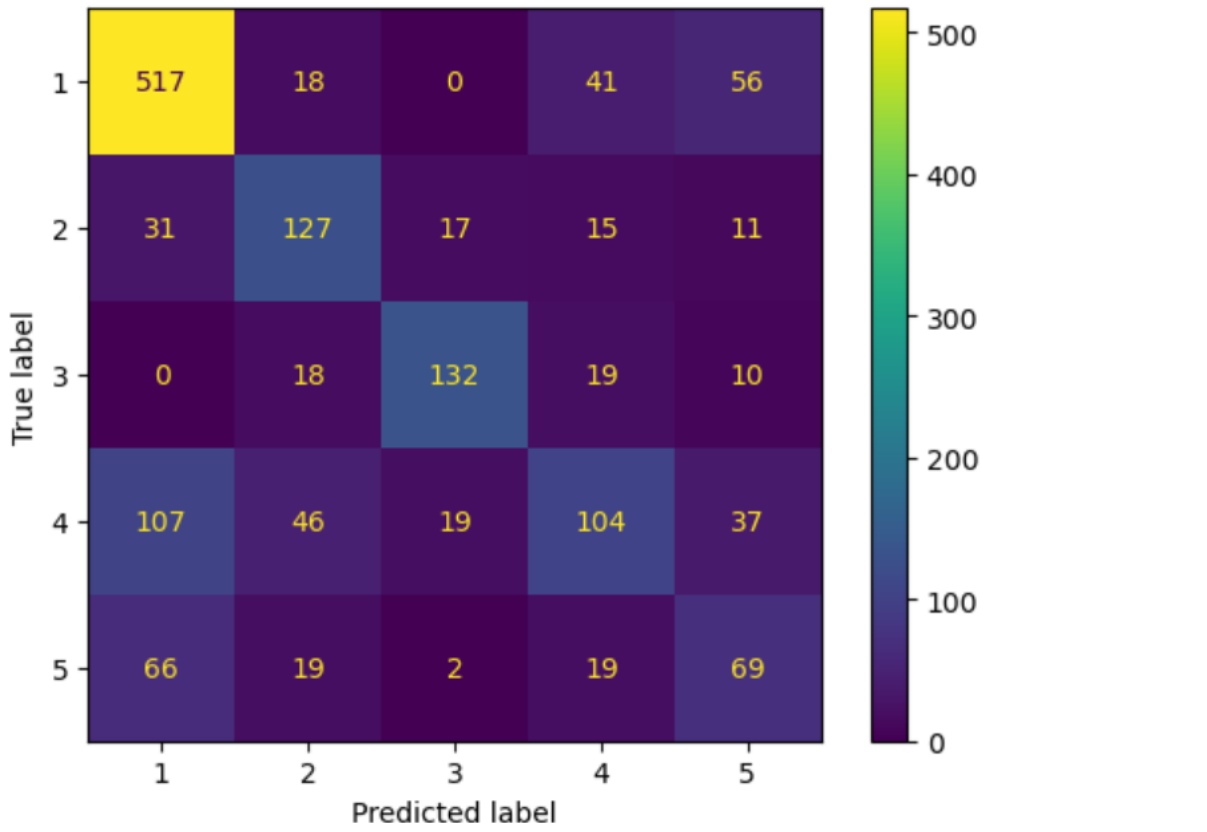
```
print(classification_report(y_test, best_ccp_alpha_tree.predict(X_test)))
```

```
              precision    recall  f1-score   support

           1       0.72      0.82      0.76       632
           2       0.56      0.63      0.59       201
           3       0.78      0.74      0.76       179
           4       0.53      0.33      0.41       313
           5       0.38      0.39      0.39       175

    accuracy                           0.63      1500
   macro avg       0.59      0.58      0.58      1500
weighted avg       0.62      0.63      0.62      1500
```

```
ConfusionMatrixDisplay.from_predictions(y_test, best_ccp_alpha_tree.predict(X_test))
plt.show()
```



Hands On