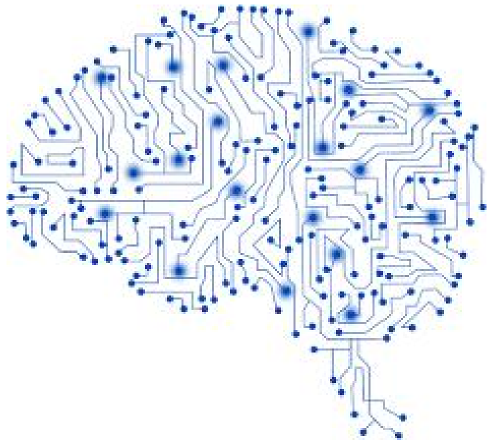
University of Minho
School of Engineering



Dados e Aprendizagem Automática

Support Vector Machine and Feature Engineering

DAA @ MEI-1º/MiEI-4º – 1º Semestre

César Analide, Bruno Fernandes, Filipa Ferraz, Filipe Gonçalves, Victor Alves

Part V

Contents

- Support Vector Machine
- Feature Engineering
- Hands On

Support Vector Machine

Support Vector Machine

- Exercise:
  - **Problem:** Development of a Machine Learning Model able to classify *if a patient has breast cancer*
  - **Classification Approach:** Support Vector Machine approach to solve this problem
  - **Dataset:** table with information regarding the *patient ID*, *diagnosis* and real-valued features computed for each *cell nucleus*, including:
    - Radius (mean of distances from center to points on the perimeter)
    - Texture (standard deviation of gray-scale values)
    - Perimeter
    - Area
    - Smoothness (local variation in radius lengths)
    - Compactness (perimeter^2 / area - 1.0)
    - Concavity (severity of concave portions of the contour)
    - Concave points (number of concave portions of the contour)
    - Symmetry
    - Fractal dimension ("coastline approximation" - 1)

Support Vector Machine

Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

We'll use the built in breast cancer dataset from Scikit Learn. We can get with the load function:

```
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
```

Support Vector Machine

The data set is presented in a dictionary form:

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

We can grab information and arrays out of this dictionary to set up our data frame and understanding of the features:

```
cancer['feature_names']
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

```
print(cancer['DESCR'])
```

…

Support Vector Machine

Set up DataFrame

```
df_feat = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df_feat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

Support Vector Machine


```python
cancer['target']
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```python
df_target = pd.DataFrame(cancer['target'],columns=['Cancer'])
```

Now let's actually check out the dataframe!

```python
df_target.head()
```

	Cancer
0	0
1	0
2	0
3	0
4	0

Support Vector Machine

Exploratory Data Analysis

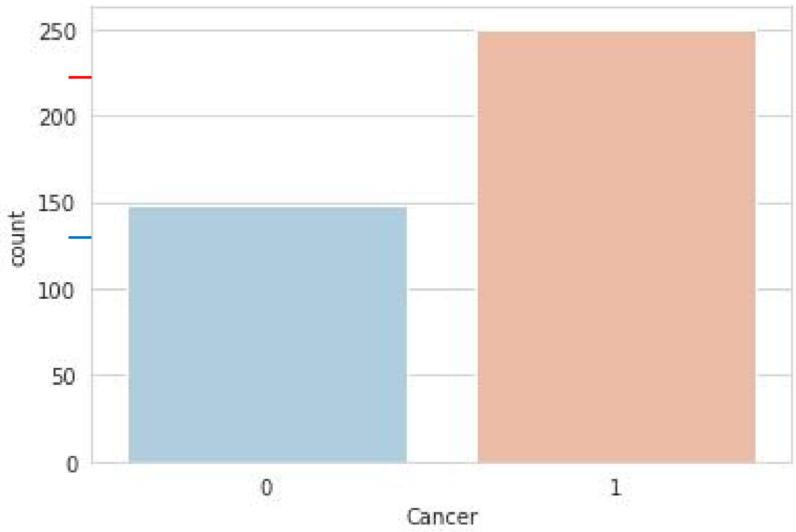
Train Test Split

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

```
X_train, X_test, y_train, y_test = train_test_split(df_feat, np.ravel(df_target), test_size=0.30, random_state=2021)
```

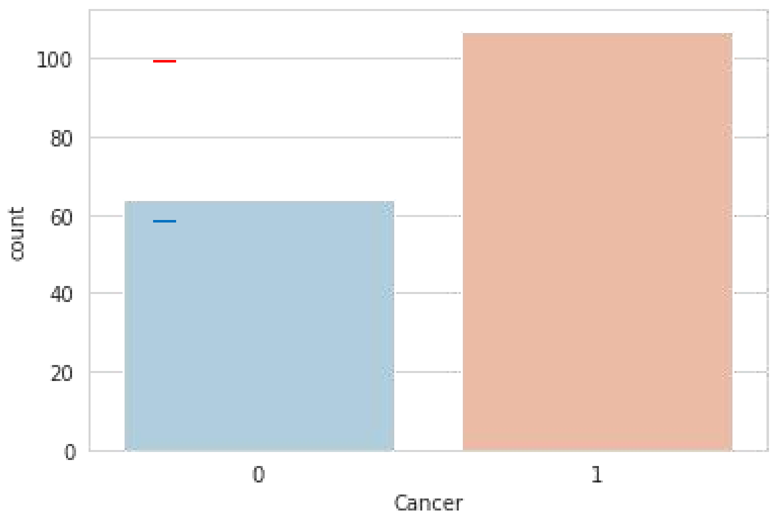
```
sns.set_style('whitegrid')
sns.countplot(x='Cancer', data = pd.DataFrame(y_train,columns=['Cancer']) ,palette='RdBu_r')
```

```
<AxesSubplot:xlabel='Cancer', ylabel='count'>
```




```
sns.set_style('whitegrid')
sns.countplot(x='Cancer', data = pd.DataFrame(y_test,columns=['Cancer']) ,palette='RdBu_r')
```

```
<AxesSubplot:xlabel='Cancer', ylabel='count'>
```



Support Vector Machine

Train the Support Vector Classifier

10-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
```

```
cross_valid_model = SVC(random_state=2021)
scores = cross_val_score(cross_valid_model, df_feat, np.ravel(df_target), cv=10)
scores
```

```
array([0.89473684, 0.84210526, 0.89473684, 0.92982456, 0.92982456,
       0.92982456, 0.94736842, 0.92982456, 0.92982456, 0.91071429])
```

```
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
0.91 accuracy with a standard deviation of 0.03
```

Hold-out

```
from sklearn.svm import SVC
```

```
model = SVC(random_state=2021)
```

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

```
SVC(random_state=2021)
```

Support Vector Machine

Predictions and Evaluations

Now let's predict using the trained model.

```
predictions = model.predict(X_test)
```

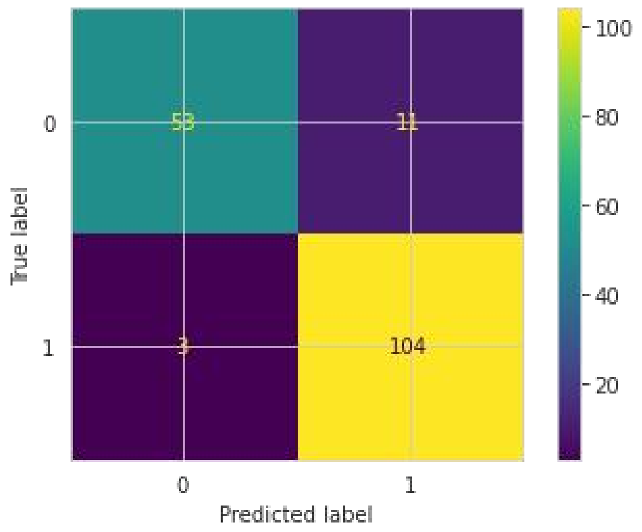
```
from sklearn.metrics import classification_report, plot_confusion_matrix, accuracy_score
```

```
print("%0.2f accuracy" % (accuracy_score(y_test, predictions)))
```

```
0.92 accuracy
```

```
plot_confusion_matrix(model, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f7800b8e0d0>
```




```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.83      0.88        64
           1       0.90      0.97      0.94       107

    accuracy                           0.92       171
   macro avg       0.93      0.90      0.91       171
weighted avg       0.92      0.92      0.92       171
```

Concepts

But first some concepts…

- **Model Parameters:** a model's (internal) configuration variable whose value is estimated from training data, i.e., the value is not set manually. Some examples include:
    - *Weights* in Artificial Neural Networks
    - *Support vectors* in Support Vector Machines

- **Model Hyperparameters:** a model's (external) configuration variable whose value can be set manually. It is difficult to know, beforehand, the best value of each hyperparameter. Tuning a model consists in finding the best (or, at least, a good) configuration of hyperparameters. Examples include:
    - *Optimizer and learning rate* in Artificial Neural Networks
    - *C and gamm*a in Support Vector Machines
    - *Quality measure and Pruning method* in Decision Trees

Support Vector Machine

GridSearch

- Finding the right parameters (like what C or gamma values to use) is a tricky task
- The idea of creating a 'grid' of parameters and trying out all the possible combinations is called a Gridsearch
  - This method is common enough that Scikit-learn has this functionality built in with GridSearchCV (CV stands for Cross-Validation)
  - GridSearchCV takes a dictionary that describes the parameters that should be tried and the model to train
  - The grid of parameters is defined as a dictionary where the keys are the parameters and the values are the settings to be tested

```
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'kernel': ['rbf']}
```

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

- GridSearchCV is a meta-estimator
- It takes an estimator like SVC and creates a new estimator that behaves exactly the same - in this case, like a classifier.
- You should add *refit=True* and choose verbose to whatever number you want (verbose means the text output describing the process).

```
grid = GridSearchCV(SVC(random_state=2021),param_grid,refit=True,verbose=3)
```

Support Vector Machine

What does fit do:

- Runs the same loop with cross-validation to find the best parameter combination
- Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation) to built a single new model using the best parameter settin

```
# May take awhile!
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.633, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.633, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.633, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.633, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.625, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.633, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.633, total=   0.0s
```

...

```
[CV] .... C=1000, gamma=0.0001, kernel=rbf, score=0.911, total=   0.0s
[Parallel(n_jobs=1)]: Done 125 out of 125 | elapsed:    1.8s finished
GridSearchCV(estimator=SVC(random_state=2021),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']},
             verbose=3)
```

Support Vector Machine

You can inspect the best parameters found by GridSearchCV in the best_params_ attribute, and the best estimator in the best_estimator_ attribute:

```
grid.best_params_
```

```
{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
```

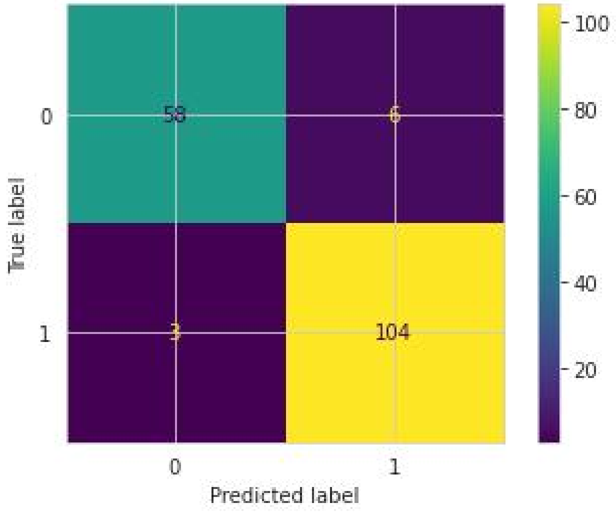
```
grid.best_estimator_
```

```
SVC(C=1, gamma=0.0001, random_state=2021)
```

Then you can re-run predictions on this grid object just like you would with a normal model.

```
plot_confusion_matrix(grid, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f777a61fc50>
```




```
grid_predictions = grid.predict(X_test)
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        64
           1       0.95      0.97      0.96       107

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

Feature Engineering

Feature Engineering

- Exercise:
  - 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

Feature Engineering

Load the data

```
data = pd.read_csv('tp5.csv')
```

```
data.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')
```

```
data.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           5000 non-null   object
dtypes: float64(5), int64(1), object(7)
memory usage: 507.9+ KB
```

```
data.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_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	None
1	Guimaraes	UNDEFINED	385	N101,	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	0.0	Sem Chuva	None
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	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
4	Guimaraes	UNDEFINED	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

Feature Engineering

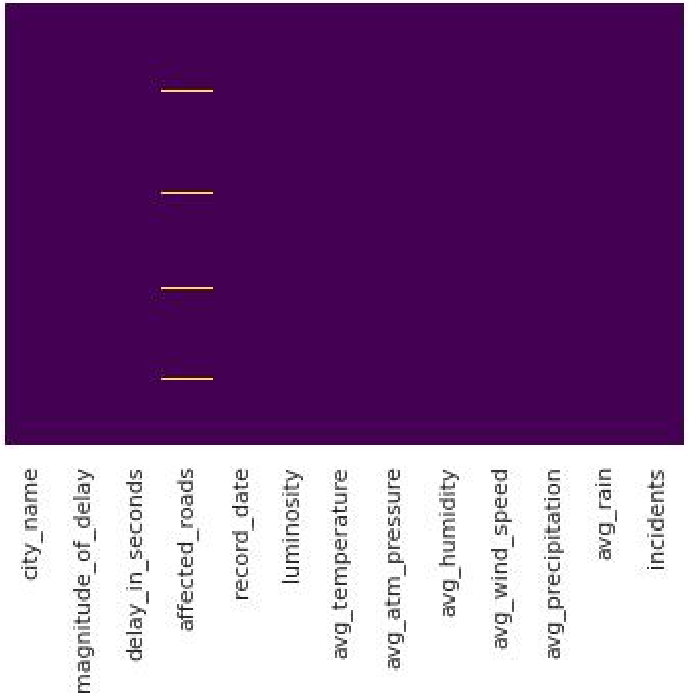
Handling missing data and possible data transformations

- Remove missing values, outliers, and unnecessary rows/ columns
- Check and impute null values
- Check Imbalanced data
- Re-indexing and reformatting our data

1. Missing Values

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

```
<AxesSubplot:>
```




```
data.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              0
dtype: int64
```

Feature Engineering

Drop or fill

Let's verify how the data is presented in the feature *affected_roads*

```
data['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
```

```
data[data['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	None
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	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4785	Guimaraes	MAJOR	298	NaN	2021-12-22 13:00	LIGHT	16.0	1015.0	71.0	3.0	0.0	Sem Chuva	None
4811	Guimaraes	UNDEFINED	96	NaN	2021-03-11 15:00	LIGHT	13.0	1025.0	89.0	3.0	0.0	chuva fraca	Medium
4838	Guimaraes	UNDEFINED	36	NaN	2021-03-10 13:00	LIGHT	14.0	1025.0	65.0	2.0	0.0	Sem Chuva	Low
4854	Guimaraes	UNDEFINED	233	NaN	2021-01-29 20:00	DARK	11.0	1017.0	92.0	1.0	0.0	Sem Chuva	High
4910	Guimaraes	UNDEFINED	324	NaN	2021-02-03 08:00	LIGHT	10.0	1012.0	90.0	2.0	0.0	Sem Chuva	Low

85 rows × 13 columns

Feature Engineering

Copy of the data to experiment the options

```
data_m1 = data.copy()
data_m2 = data.copy()
```

a) Drop

```
data_m1.drop(['affected_roads'], axis = 1, inplace = True)
data_m1.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	None
1	Guimaraes	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	0.0	Sem Chuva	None
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

Feature Engineering

b) Fill with zero

```
data_m2.fillna(0, inplace = True)
data_m2.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_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	None
1	Guimaraes	UNDEFINED	385	N101,	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	0.0	Sem Chuva	None
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	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
4	Guimaraes	UNDEFINED	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

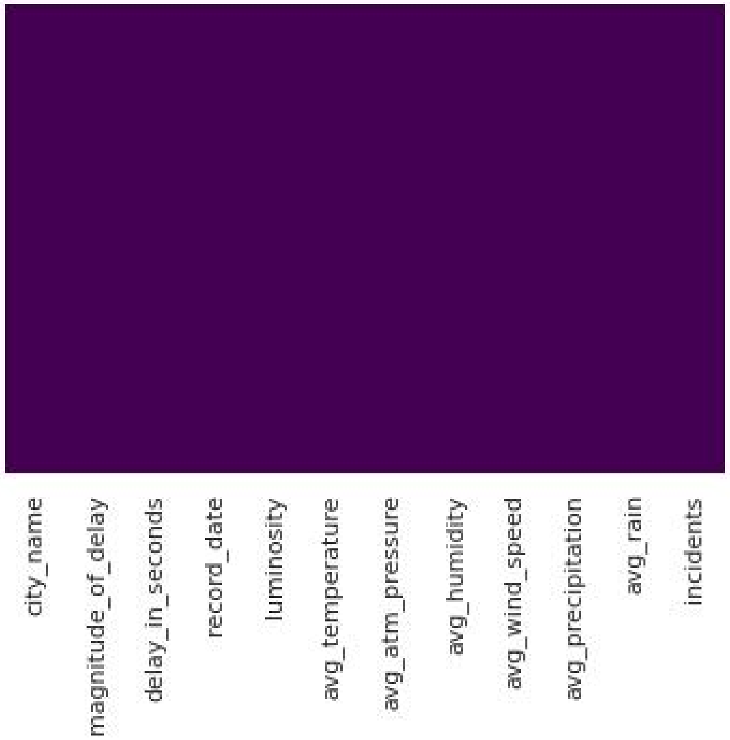
We need to choose one of the options to keep going. We will choose to drop the column since it does not bring added value to our goal.

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

Feature Engineering


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

```
<AxesSubplot:>
```




```
data.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            0
dtype: int64
```

```
data.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           5000 non-null   object
dtypes: float64(5), int64(1), object(6)
memory usage: 468.9+ KB
```

Feature Engineering


```
data.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	None
1	Guimaraes	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	0.0	Sem Chuva	None
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

There are features that are of the type *object*: *city_name*, *magnitude_of_delay*, *record_date*, *luminosity*, *avg_rain* and *incidents*.

Let's see how many different values each feature has.

```
data.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              5
dtype: int64
```

Feature Engineering

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

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

```
1
```

```
data['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
```

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

```
data.drop(['avg_precipitation'], axis = 1, inplace = True)
data.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	None
1	Guimaraes	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	None
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

Feature Engineering

2. Handling categoric data

Feature *city_name*

```
data['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.

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

Let's see the feature *incidents*:

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

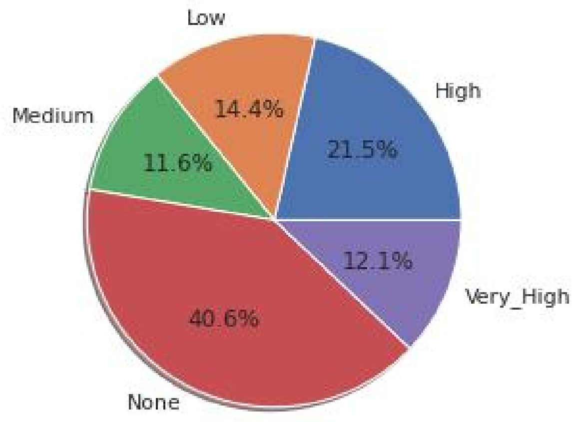
```
None         2028
High         1073
Low           718
Very_High     603
Medium        578
Name: incidents, dtype: int64
```

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

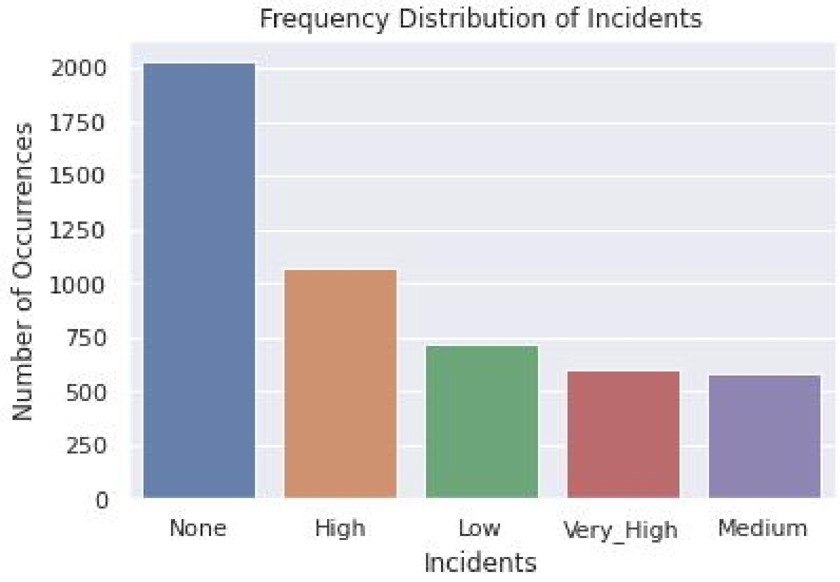
```
5
```

Feature Engineering


```
labels = data['incidents'].astype('category').cat.categories.tolist()
counts = data['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) #autopct is show the % on plot
ax1.axis('equal')
plt.show()
```




```
incidents_count = data['incidents'].value_counts()
sns.set(style="darkgrid")
sns.barplot(incidents_count.index, incidents_count.values, alpha=0.9)
plt.title('Frequency Distribution of Incidents')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Incidents', fontsize=12)
plt.show()
```



Feature Engineering

We have several options how to deal with qualitative data:

a) Replace the values

Again, we are using data copies to experiment all options.

```
data_r1=data.copy()

data_r1.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	None
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	None
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

We need to create a dictionary assigning the string to a numeric value:

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

Feature Engineering

Then we create the labels and associate:

```
labels = data_r1['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, 'None': 4, 'Very_High': 5}}
```

```
data_r1.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	None
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	None
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

Feature Engineering

Now we need to replace with the new values:

```
data_r1.replace(replace_map_comp, inplace=True)
data_r1.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	4
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	4
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	5
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	1

Done! Now we can see that the type of values are *int64*:

```
print(data_r1['incidents'].dtypes)
```

```
int64
```

Feature Engineering

b) Label encoding

```
data_r2=data.copy()

data_r2.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
0	1	0	2021-03-15 23:00	1	12.0	1013.0	70.0	1.0	1	None
1	1	385	2021-12-25 18:00	1	12.0	1007.0	91.0	1.0	1	None
2	1	69	2021-03-12 15:00	2	14.0	1025.0	64.0	0.0	1	Low
3	2	2297	2021-09-29 09:00	2	15.0	1028.0	75.0	1.0	1	Very_High
4	1	0	2021-06-13 11:00	2	27.0	1020.0	52.0	1.0	1	High

```
print(data_r2.dtypes)
```

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

Feature Engineering

Similar to the previous examples, each string will be assigned a number. Instead of replacing the values under the column *incidents*, it will be created a new colum to each created label.

```
data_r2['None'] = np.where(data_r2['incidents'].str.contains('None'), 1, 0)
data_r2.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents	None
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	None	1
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	None	1
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

To complete the process, it is needed to replicate for each label and then drop the column *incidents*.

Feature Engineering

Let's see another way to label encoding. This uses the *LabelEncoder* from *sklearn*.

```
data_r2_skl = data.copy()
data_r22=data.copy()

from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
data_r2_skl['incidents_code'] = lb_make.fit_transform(data_r22['incidents'])

data_r2_skl.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
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	None	3
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	None	3
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	4
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	High	0

It creates a new column, *incidents_code*, with the labels assigned to feature *incidents*. The numeric values were assigned randomly, being the crescent order not apllicable to the meaning of the qualifying words.

Feature Engineering

c) One-Hot encoding

This alternative uses *LabelBinarizer* of *sklearn* and creates a matrix with bits regarding each label.

```
data_r3 = data.copy()

from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
lb_results = lb.fit_transform(data_r3['incidents'])
lb_results_df = pd.DataFrame(lb_results, columns=lb.classes_)

lb_results_df.head()
```

	High	Low	Medium	None	Very_High
0	0	0	0	1	0
1	0	0	0	1	0
2	0	1	0	0	0
3	0	0	0	0	1
4	1	0	0	0	0

Feature Engineering


```
result_df = pd.concat([data_r3, 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	None	Very_High
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	None	0	0	0	1	0
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	None	0	0	0	1	0
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	Low	0	1	0	0	0
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	Very_High	0	0	0	0	1
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	High	1	0	0	0	0

Feature Engineering

d) Binary Encoding

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

For this technique it is needed to have the *category_encoders* installed: `!pip install category_encoders`

```
data_r4 = data.copy()

import category_encoders as ce

encoder = ce.BinaryEncoder(cols=['incidents'])
df_binary = encoder.fit_transform(data_r4)

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
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	0	0	1
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	0	0	1
2	UNDEFINED	69	2021-03-12 15:00	LIGHT	14.0	1025.0	64.0	0.0	Sem Chuva	0	1	0
3	MAJOR	2297	2021-09-29 09:00	LIGHT	15.0	1028.0	75.0	1.0	Sem Chuva	0	1	1
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	1	0	0

Feature Engineering

e) Backward difference encoding

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

```
data_r5 = data.copy()

encoder = ce.BackwardDifferenceEncoder(cols=['incidents'])
df_bd = encoder.fit_transform(data_r5)

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	incidents_3
0	1	1	0	2021-03-15 23:00	1	12.0	1013.0	70.0	1.0	1	-0.8	-0.6	-0.4	-0.2
1	1	1	385	2021-12-25 18:00	1	12.0	1007.0	91.0	1.0	1	-0.8	-0.6	-0.4	-0.2
2	1	1	69	2021-03-12 15:00	2	14.0	1025.0	64.0	0.0	1	0.2	-0.6	-0.4	-0.2
3	1	2	2297	2021-09-29 09:00	2	15.0	1028.0	75.0	1.0	1	0.2	0.4	-0.4	-0.2
4	1	1	0	2021-06-13 11:00	2	27.0	1020.0	52.0	1.0	1	0.2	0.4	0.6	-0.2

Feature Engineering

f) Factorize

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

```
data_r6 = data.copy()
```

```
data_r6['incidents'] = pd.factorize(data_r6['incidents'])[0] + 1
data_r6.head()
```

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	1
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	1
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	3
4	UNDEFINED	0	2021-06-13 11:00	LIGHT	27.0	1020.0	52.0	1.0	Sem Chuva	4

We will choose the factorize technique to keep going.

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

	magnitude_of_delay	delay_in_seconds	record_date	luminosity	avg_temperature	avg_atm_pressure	avg_humidity	avg_wind_speed	avg_rain	incidents
0	UNDEFINED	0	2021-03-15 23:00	DARK	12.0	1013.0	70.0	1.0	Sem Chuva	1
1	UNDEFINED	385	2021-12-25 18:00	DARK	12.0	1007.0	91.0	1.0	Sem Chuva	1
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	3

Feature Engineering

Regarding the features *magnitude_delay*, *luminosity* and *avg_rain*, we will factorize for now.

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

data.head()
```

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

Feature Engineering

3. Handling dates

Datetime Properties and Methods (https://pandas.pydata.org/pandas-docs/version/0.23/api.html#datetimelike-properties)

```
data_dt = data.copy()
```

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

```
0    2021-03-15 23:00
1    2021-12-25 18:00
2    2021-03-12 15:00
3    2021-09-29 09:00
4    2021-06-13 11:00
Name: record_date, dtype: object
```

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

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

```
assert data_dt['record_date'].isnull().sum() == 0, 'missing record date'
```

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

```
0   2021-03-15 23:00:00
1   2021-12-25 18:00:00
2   2021-03-12 15:00:00
3   2021-09-29 09:00:00
4   2021-06-13 11:00:00
Name: record_date, dtype: datetime64[ns]
```

Feature Engineering

We can extract parts of the date and create newm columns with that:

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

```
data_dt.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	record_date_day	record_date_hour	record_date_minute
0	1	0	2021-03-15 23:00:00	1	12.0	1013.0	70.0	1.0	1	1	2021	3	15	23	0
1	1	385	2021-12-25 18:00:00	1	12.0	1007.0	91.0	1.0	1	1	2021	12	25	18	0
2	1	69	2021-03-12 15:00:00	2	14.0	1025.0	64.0	0.0	1	2	2021	3	12	15	0
3	2	2297	2021-09-29 09:00:00	2	15.0	1028.0	75.0	1.0	1	3	2021	9	29	9	0
4	1	0	2021-06-13 11:00:00	2	27.0	1020.0	52.0	1.0	1	4	2021	6	13	11	0

Feature Engineering


```
data_dt.nunique()
```

```
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_rain                  4
incidents                 5
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 it.

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

Feature Engineering

Other functions to deal with dates

```
data_dt2 = data.copy()
```

```
data_dt2['record_date'] = pd.to_datetime(data_dt2['record_date'])
data_dt2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 0 to 4999
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   magnitude_of_delay  5000 non-null   int64
 1   delay_in_seconds    5000 non-null   int64
 2   record_date         5000 non-null   datetime64[ns]
 3   luminosity          5000 non-null   int64
 4   avg_temperature     5000 non-null   float64
 5   avg_atm_pressure    5000 non-null   float64
 6   avg_humidity        5000 non-null   float64
 7   avg_wind_speed      5000 non-null   float64
 8   avg_rain            5000 non-null   int64
 9   incidents           5000 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(5)
memory usage: 429.7 KB
```

```
data_dt2['record_date'].head()
```

```
0   2021-03-15 23:00:00
1   2021-12-25 18:00:00
2   2021-03-12 15:00:00
3   2021-09-29 09:00:00
4   2021-06-13 11:00:00
Name: record_date, dtype: datetime64[ns]
```

Feature Engineering

We can use *datetime.today* and fetch the actual date.

```
import datetime

today = datetime.datetime.today()

today
```

```
datetime.datetime(2022, 10, 26, 10, 27, 52, 327533)
```

It can be measured the time elapsed between the dates on the dataset and today.

```
today - data_dt2['record_date']
```

```
0      589 days 11:27:52.327533
1      304 days 16:27:52.327533
2      592 days 19:27:52.327533
3      392 days 01:27:52.327533
4      499 days 23:27:52.327533
                  ...
4995   561 days 10:27:52.327533
4996   476 days 20:27:52.327533
4997   587 days 07:27:52.327533
4998   358 days 04:27:52.327533
4999   310 days 08:27:52.327533
Name: record_date, Length: 5000, dtype: timedelta64[ns]
```

```
(today - data_dt2['record_date']).dt.days
```

```
0       589
1       304
2       592
3       392
4       499
       ...
4995    561
4996    476
4997    587
4998    358
4999    310
Name: record_date, Length: 5000, dtype: int64
```

Feature Engineering


```
data_dt2['day'] = data_dt2['record_date'].dt.day
data_dt2['month'] = data_dt2['record_date'].dt.month
data_dt2['hour'] = data_dt2['record_date'].dt.hour
data_dt2['time'] = data_dt2['record_date'].dt.time
data_dt2.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
0	1	0	2021-03-15 23:00:00	1	12.0	1013.0	70.0	1.0	1	1	15	3	23	23:00:00
1	1	385	2021-12-25 18:00:00	1	12.0	1007.0	91.0	1.0	1	1	25	12	18	18:00:00
2	1	69	2021-03-12 15:00:00	2	14.0	1025.0	64.0	0.0	1	2	12	3	15	15:00:00
3	2	2297	2021-09-29 09:00:00	2	15.0	1028.0	75.0	1.0	1	3	29	9	9	09:00:00
4	1	0	2021-06-13 11:00:00	2	27.0	1020.0	52.0	1.0	1	4	13	6	11	11:00:00

Feature Engineering

Now we need to choose how to deal with the *record_date*.

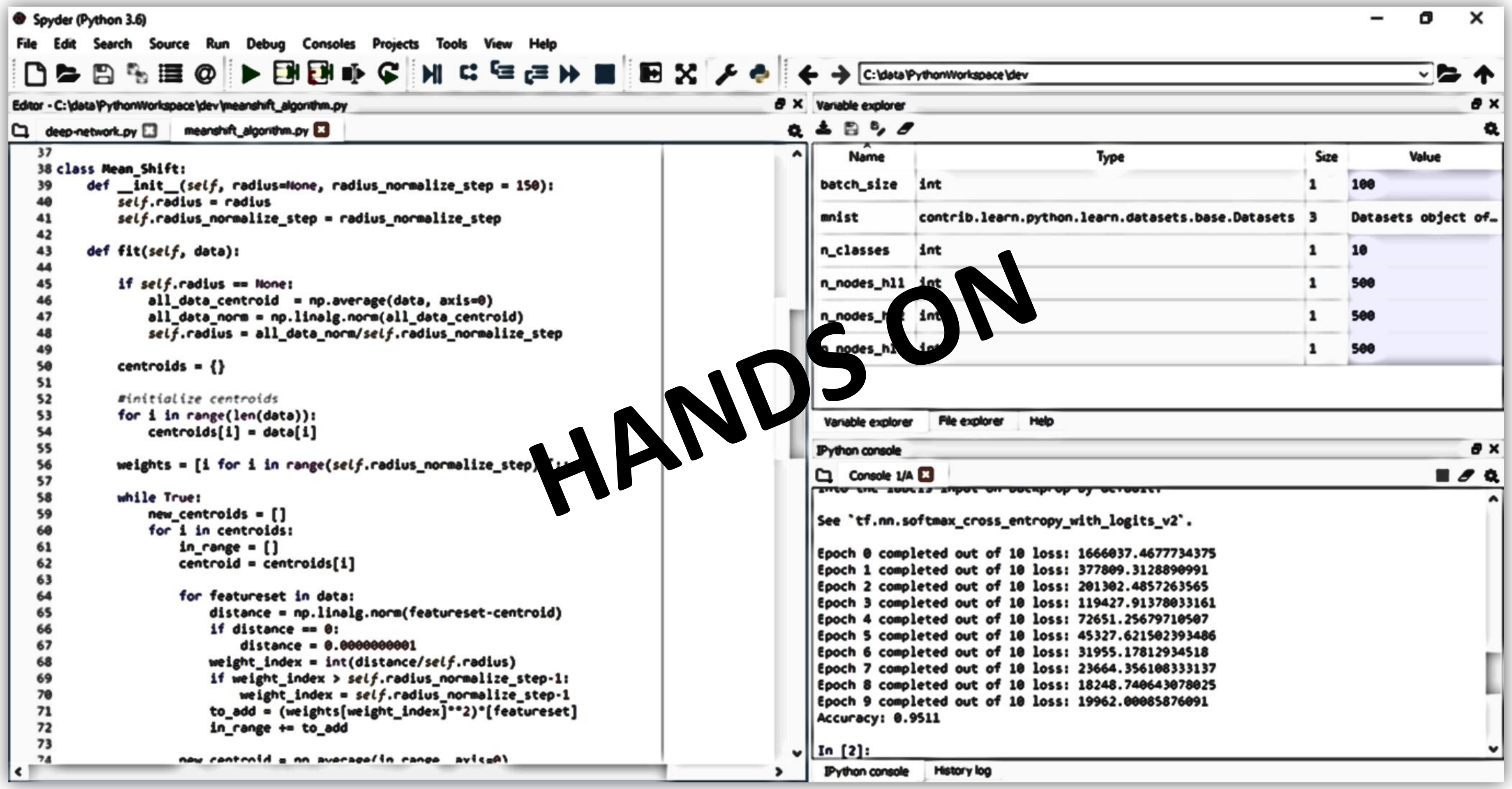
```
data['record_date'] = pd.to_datetime(data['record_date'], format = '%Y-%m-%d %H:%M', errors='coerce')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 0 to 4999
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   magnitude_of_delay  5000 non-null   int64
 1   delay_in_seconds    5000 non-null   int64
 2   record_date         5000 non-null   datetime64[ns]
 3   luminosity          5000 non-null   int64
 4   avg_temperature     5000 non-null   float64
 5   avg_atm_pressure    5000 non-null   float64
 6   avg_humidity        5000 non-null   float64
 7   avg_wind_speed      5000 non-null   float64
 8   avg_rain            5000 non-null   int64
 9   incidents           5000 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(5)
memory usage: 429.7 KB
```

There are other features that need to be worked on, but it's up to you now!

Hands On


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
HANDS ON
Variable explorer
Name Type Size Value
batch_size int 1 100
mnist contrib.learn.python.learn.datasets.base.Datasets 3 Datasets object of…
n_classes int 1 10
IPython console
Epoch 0 completed out of 10 loss: 1666037.4677734375
Epoch 9 completed out of 10 loss: 19962.00085876091
Accuracy: 0.9511
In [2]:
IPython console History log