Mathematics in Machine Learning - Thesis

Student: Fiorio Plà Paolo s276525

Index

1. • INTRODUCTION
   0.1 • Import Section
2. • DATA SET ANALYSIS
   1.1 • Data Description
   1.2 • Categorical features distribution
   1.3 • Numerical features distribution
   1.4 • Spatial statistics X and Y
   1.5 • Correlation
   1.6 • Target feature
3. • PREPROCESSING
   2.1 • Encoding data
   2.2 • Division of prediction features
   2.3 • DataSet partition
   2.4 • Standardization
   2.5 • Dimensionality reduction
4. • MODEL SELECTION
   3.1 • Metrics
   3.2 • Algorithms
   3.3 • Linear Regression
   3.4 • Random Forest
   3.5 • SVR
   3.6 • K-NN
   3.7 • Model selection results
5. • FEATURES SELECTION
   4.1 • STFWI
   4.2 • STM
   4.3 • FWI
   4.4 • Weather Conditions
6. • CONCLUSION
7. • REFERENCES

INTRODUCTION

A wildfire is an uncontrolled fire that consumes vegetation mostly in rural areas, and can be categorized by several factors such as cause of ignition, physical properties, combustable material, etc. **Wildfires can cause significant damages to human properties and claim lives** like in Australia's and Amazon's forest fires in early 2020 and Canadian's forest fires of 2018. In Canada, wildfires have consumed an average of 2.5 million hectares a year.
Every year, due to weather conditions and human bad behaviours the number of forest fires and the amplitude of their areas and their damage keep increasing.
In this context, **using as target the burned area of forests**, the aim of this project is to find the best model able to predict forest fires for resource management purposes and also to try to understand which are the factors that are most likely linked to them (such as temperature, wind,...).

Import Section

The main Python libraries used for this purpose are the following:

- **Pandas** and **Numpy** for the mathematical computations
- **Matplotlib** and **Seaborn** for the graphical part
- **Sklearn** for the modelling side

In [2]:
```
import pandas as pd
from matplotlib import pyplot
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import model_selection
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

DATA SET ANALYSIS

Data Description

The analysis is based on Forest Fires Data Set, which is a dataset of wildfires occurrences of Montesinho natural park, from the Tràs-os-Montes northeast region of Portugal.This park contains a high flora and fauna diversity and is inserted within a supra-Mediterranean climate. The data used in the experiments was collected from January 2000 to December 2003.
It is available at: http://archive.ics.uci.edu/ml/datasets/Forest+Fires (http://archive.ics.uci.edu/ml/datasets/Forest+Fires).

In [11]:
```
forest_fires = pd.read_csv('forestfires.csv')
forest_fires.head()
```

Out[11]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

Lets see how this dataset is composed and if there are missing values that will have to be treated.

In [3]:
```
print("Shape of Dataset:", forest_fires.shape)
print("Number of samples:", forest_fires.shape[0])
print("Number of features:", forest_fires.shape[1])
print("Features names:", list(forest_fires))
if forest_fires.isnull().sum().sum() == 0:
    print("\nIn this dataset there are no missing values.")
else:
    print(f"\nIn this dataset there are some missing values:{forest_fires.isnu
ll().sum()}")
```

```
Shape of Dataset: (517, 13)
Number of samples: 517
Number of features: 13
Features names: ['X', 'Y', 'month', 'day', 'FFMC', 'DMC', 'DC', 'ISI', 'tem
p', 'RH', 'wind', 'rain', 'area']

In this dataset there are no missing values.
```

The attributes in the dataset include:

- **X**: x-axis spatial coordinate within the Montesinho park map: 1 to 9
- **Y**: y-axis spatial coordinate within the Montesinho park map: 2 to 9
- **month**: month of the year: "jan" to "dec"
- **day**: day of the week: "mon" to "sun"
- **FFMC**: index from the FWI system: 18.7 to 96.20
- **DMC**: index from the FWI system: 1.1 to 291.3
- **DC**: index from the FWI system: 7.9 to 860.6
- **ISI**: index from the FWI system: 0.0 to 56.10
- **temp**: temperature in Celsius degrees: 2.2 to 33.30
- **RH**: relative humidity in %: 15.0 to 100
- **wind**: wind speed in km/h: 0.40 to 9.40
- **rain**: outside rain in mm/m2 : 0.0 to 6.4
- **area**: the burned area of the forest (in ha): 0.00 to 1090.84

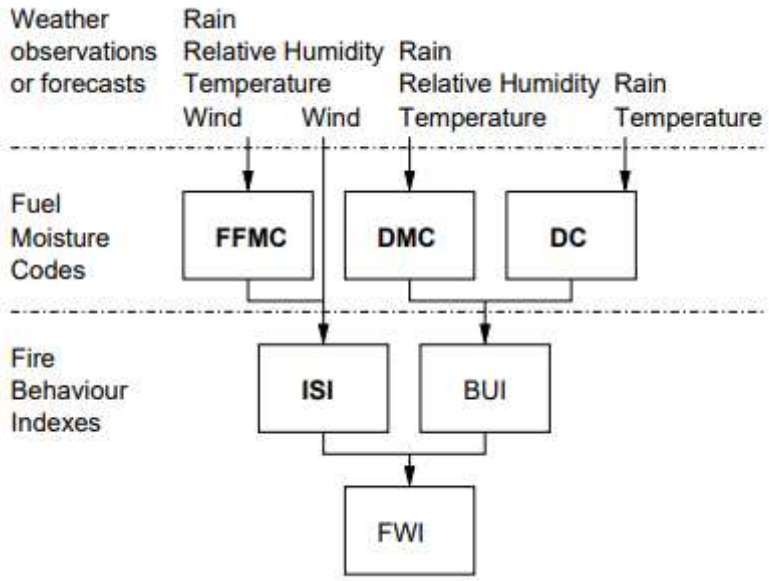
Fine Fuel Moisture Code(FFMC): a numeric rating of the moisture content of litter and other cured fine fuels. This code is an indicator of the relative ease of ignition and the flammability of fine fuel.

Duff Moisture Code(DMC): a numeric rating of the average moisture content of loosely compacted organic layers of moderate depth. This code gives an indication of fuel consumption in moderate duff layers and medium-size woody material.

Drought Code(DC):a numeric rating of the average moisture content of deep, compact organic layers. This code is a useful indicator of seasonal drought effects on forest fuels and the amount of smoldering in deep duff layers and large logs.

Initial Spread Index(ISI):a numeric rating of the expected rate of fire spread. It combines the effects of wind and the FFMC on rate of spread without the influence of variable quantities of fuel.

A relationships between those indexes can be seen in the image below, taken from a paper made from the authors of the dataset.



The dataset is composed both from numerical and categorical attributes.
It is possible to analize numerical ones by using the method describe, which prints the following stats:

- **Count** the total number of instances
- **Mean** of all the instances
- **Std** of all the instances
- **Min** return the minimum value
- **25%, 50%, 75%** are percentile values for each attribute
- **Max** return the maximum value

In [4]:
```
forest_fires.describe()
```

Out[4]:

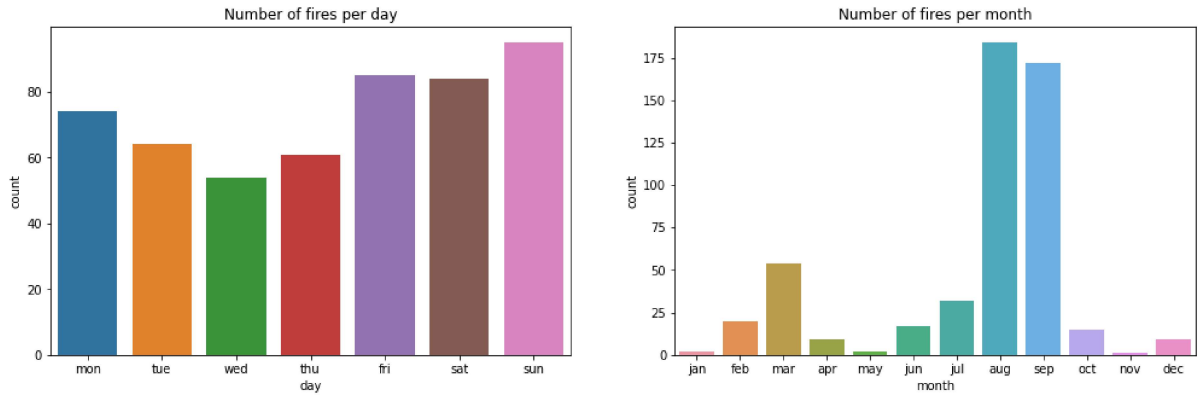
	X	Y	FFMC	DMC	DC	ISI	temp	
count	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517
mean	4.669246	4.299807	90.644681	110.872340	547.940039	9.021663	18.889168	44
std	2.313778	1.229900	5.520111	64.046482	248.066192	4.559477	5.806625	16
min	1.000000	2.000000	18.700000	1.100000	7.900000	0.000000	2.200000	15
25%	3.000000	4.000000	90.200000	68.600000	437.700000	6.500000	15.500000	33
50%	4.000000	4.000000	91.600000	108.300000	664.200000	8.400000	19.300000	42
75%	7.000000	5.000000	92.900000	142.400000	713.900000	10.800000	22.800000	53
max	9.000000	9.000000	96.200000	291.300000	860.600000	56.100000	33.300000	100

Categorical features distribution

Month and day are the only categorical features. They'll be shown with a proper graph.

In [11]:
```
days = ['mon','tue','wed','thu','fri','sat','sun']
forest_fires['day'] = pd.Categorical(forest_fires['day'], categories=
    ['mon','tue','wed','thu','fri','sat','sun'],
    ordered=True)

months = ['jan','feb','mar','apr','may','jun', 'jul','aug','sep','oct','nov',
'dec']
forest_fires['month'] = pd.Categorical(forest_fires['month'], categories=
    ['jan','feb','mar','apr','may','jun', 'jul','aug','sep','oct','nov','dec'
],
    ordered=True)

fig, (ax1, ax2) = plt.subplots(ncols = 2, nrows = 1, figsize = (17, 5))
sns.countplot(forest_fires['day'], order = days, ax = ax1).set_title("Number o
f fires per day")
sns.countplot(forest_fires['month'], order = months, ax = ax2).set_title("Numb
er of fires per month")
```

Out[11]: Text(0.5, 1.0, 'Number of fires per month')

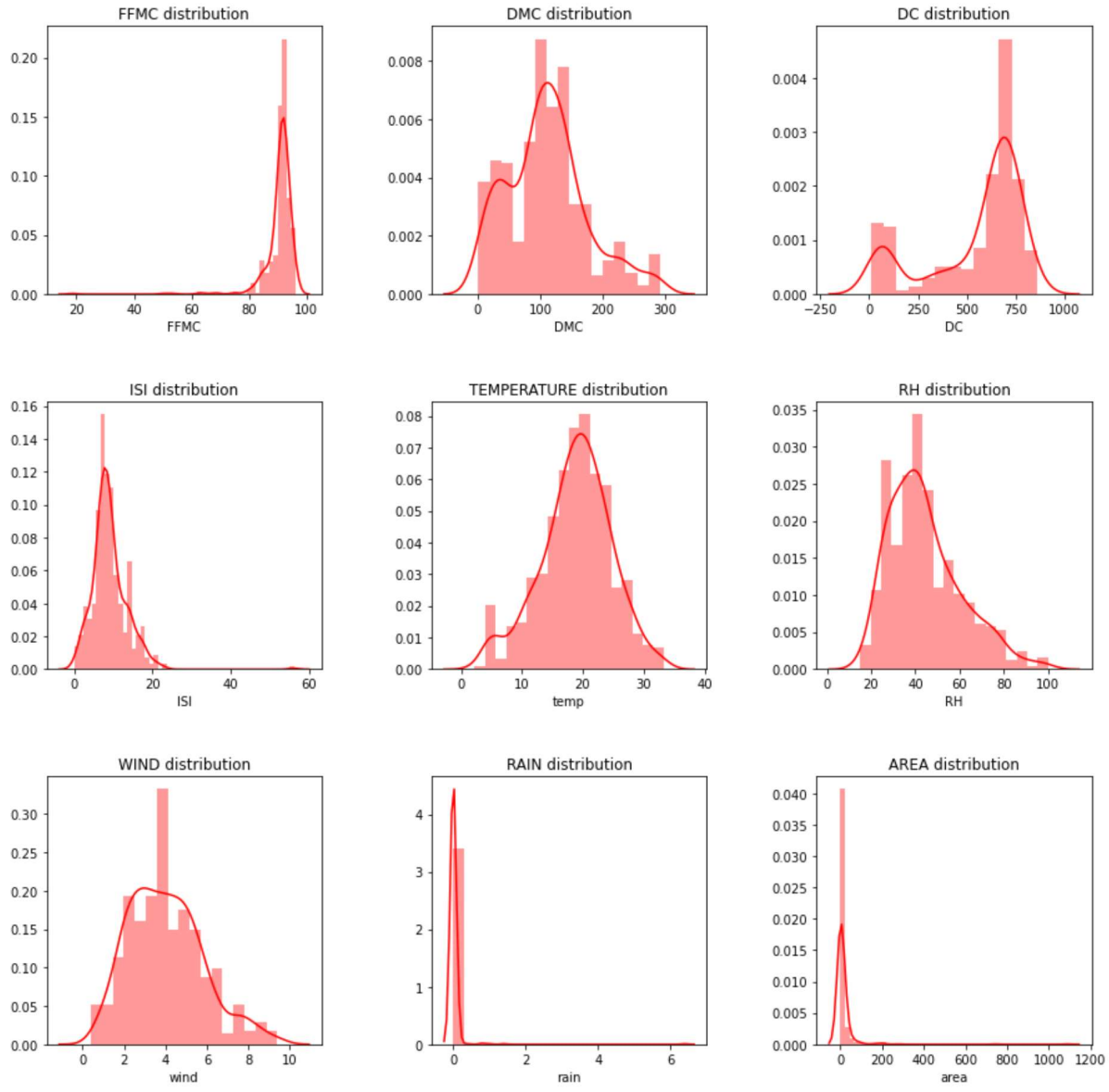


- Considering the daily graph we can see that the number of fires increases during the weekends.
- Considering the monthly graph we can see that the number of fires increases during the summer.
  So it seems quite obvious that the number of fires increases during the hottest season but also with an highest presence of humans, during the weekends.

Numerical features distribution

The following graphs will show the distributions of the remaining features, the numerical ones.

In [27]:
```
f, axes = plt.subplots(3, 3, figsize=(15, 15), sharex=False)
sns.distplot(forest_fires['FFMC'] , color="r", ax=axes[0, 0]).set_title("FFMC
 distribution")
sns.distplot(forest_fires['DMC'] , color="r", ax=axes[0, 1]).set_title("DMC di
stribution")
sns.distplot(forest_fires['DC'] , color="r", ax=axes[0, 2]).set_title("DC dist
ribution")
sns.distplot(forest_fires['ISI'] , color="r", ax=axes[1, 0]).set_title("ISI di
stribution")
sns.distplot(forest_fires['temp'] , color="r", ax=axes[1, 1]).set_title("TEMPE
RATURE distribution")
sns.distplot(forest_fires['RH'] , color="r", ax=axes[1, 2]).set_title("RH dist
ribution")
sns.distplot(forest_fires['wind'] , color="r", ax=axes[2, 0]).set_title("WIND
 distribution")
sns.distplot(forest_fires['rain'] , color="r", ax=axes[2, 1]).set_title("RAIN
 distribution")
sns.distplot(forest_fires['area'] , color="r", ax=axes[2, 2]).set_title("AREA
 distribution")
plt.subplots_adjust(hspace=0.4, wspace=0.4)
plt.show()
```



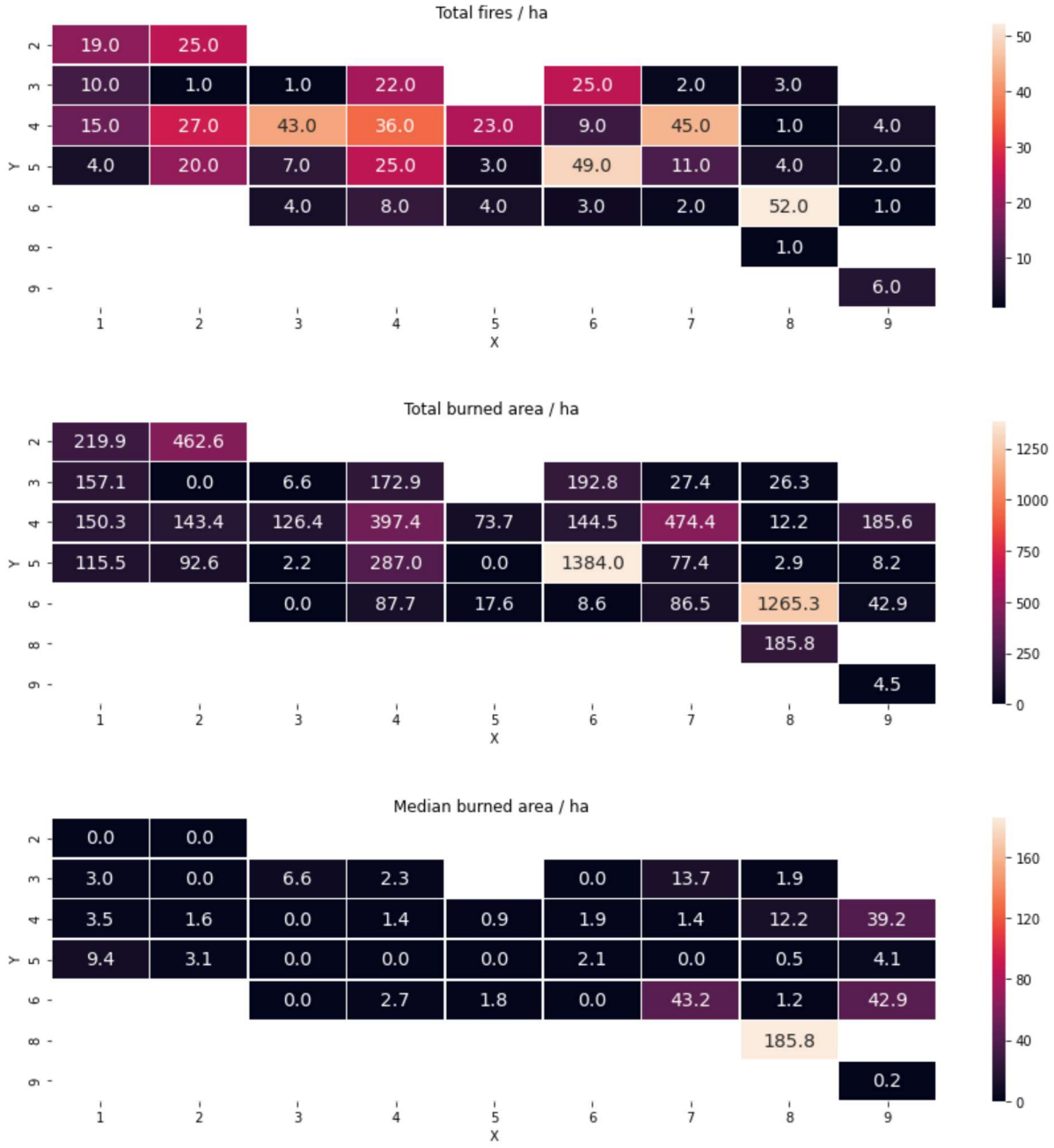
From these graphs we can have an idea of the range of values of each numerical feature. The ones with a smaller range seems to be FFMC, rain and area while the others have a larger range. The temperature feature distribution is the one that comes closer to a normal distribution.
The y-axis in a density plot is the probability density function: we need to be careful to specify that this is a probability density and not a probability. The only requirement of the density plot is that the total area under the curve integrates to one.

Spatial statistics X and Y

The following graphs are made to give a spatial localization in order to see the (X,Y) areas most affected by fires considering their numbers and their intensity.

In [33]:

```
group = forest_fires.groupby(['X', 'Y'], as_index = False)

xy_sum = group[['area']].sum()
xy_sum = xy_sum.pivot('Y', 'X', 'area')

xy_cnt = group[['area']].count()
xy_cnt = xy_cnt.pivot('Y', 'X', 'area')

xy_med = group[['area']].median()
xy_med = xy_med.pivot('Y', 'X', 'area')

fig, axes = plt.subplots(nrows = 3, ncols = 1, figsize = (15,15))


_ = sns.heatmap(xy_cnt, annot=True, fmt=".1f", linewidths=.5, ax = axes[0],
                annot_kws = {'fontsize': 14}).set_title('Total fires / ha ')

_ = sns.heatmap(xy_sum, annot=True, fmt=".1f", linewidths=.5, ax = axes[1],
                annot_kws = {'fontsize': 14}).set_title('Total burned area / ha
')

_ = sns.heatmap(xy_med, annot=True, fmt=".1f", linewidths=.5, ax = axes[2],
                annot_kws = {'fontsize': 14}).set_title('Median burned area / h
a ')

plt.subplots_adjust(hspace=0.4, wspace=0.4)
plt.show()
```



Comparing total fires with total burned area there is some evidence that fires at low X are small and numerous, where fires at high X are less frequent but larger. The median burned area reinforces the last bullet, that is to say, smaller fires dominate low-X regions wheres larger fires dominate at high-X regions.
A particular case is represented by the coordinate (8,8) were only one fire was able to burn an area of 185.8 ha.

Correlation

The correlation allows to see the relationship between two statistical variables X and Y. It takes into account the definition of covariance, and represents the relationship in a range of values between -1 and 1. This is possible because it divides the covariance value by the standard deviations values.

Considering two statistical variables X and Y the definition of covariance is:

$$cov(X,Y) = E[(X-\mu_x)(Y-\mu_y)] = E[XY] - E[X]E[Y] = E[XY] - \mu_x\mu_y$$

Instead, in case of two equal statistical variables:

$$cov(X,X) = Var(X) = \sigma_X^2$$

Between two statistical variables X and Y is possible to define a coefficient of correlation:

$$\rho_{XY} = \frac{cov(X,Y)}{\sigma_X\sigma_Y}$$

In case of two equal statistical variables, the coefficient becomes:

$$\rho_{XX} = \frac{cov(X,X)}{\sigma_X\sigma_X} = \frac{\sigma_X^2}{\sigma_X\sigma_X} = \frac{\sigma_X^2}{\sigma_X^2} = 1$$

Considering the value of this coefficient is possible to understand the correlation between the two involved variables:

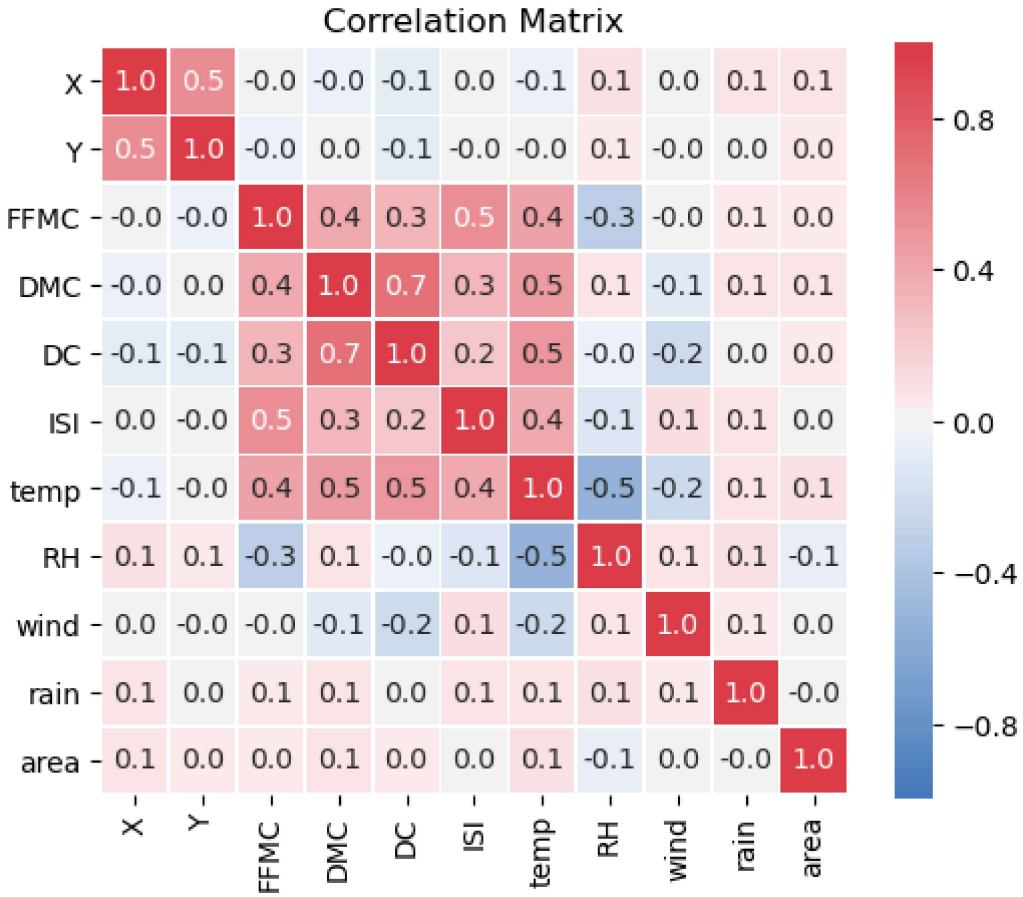
- if $\rho_{XY} > 0$, the variables X and Y are said to be directly related;
- if $\rho_{XY} = 0$, the variables X and Y are said to be uncorrelated;
- if $\rho_{XY} < 0$, the variables X and Y are said to be inversely related.

Finally, can be defined the correlation matrix as:

$$\rho_{ij} = \frac{cov(X_i,X_j)}{\sigma_{X_i}\sigma_{X_j}}$$

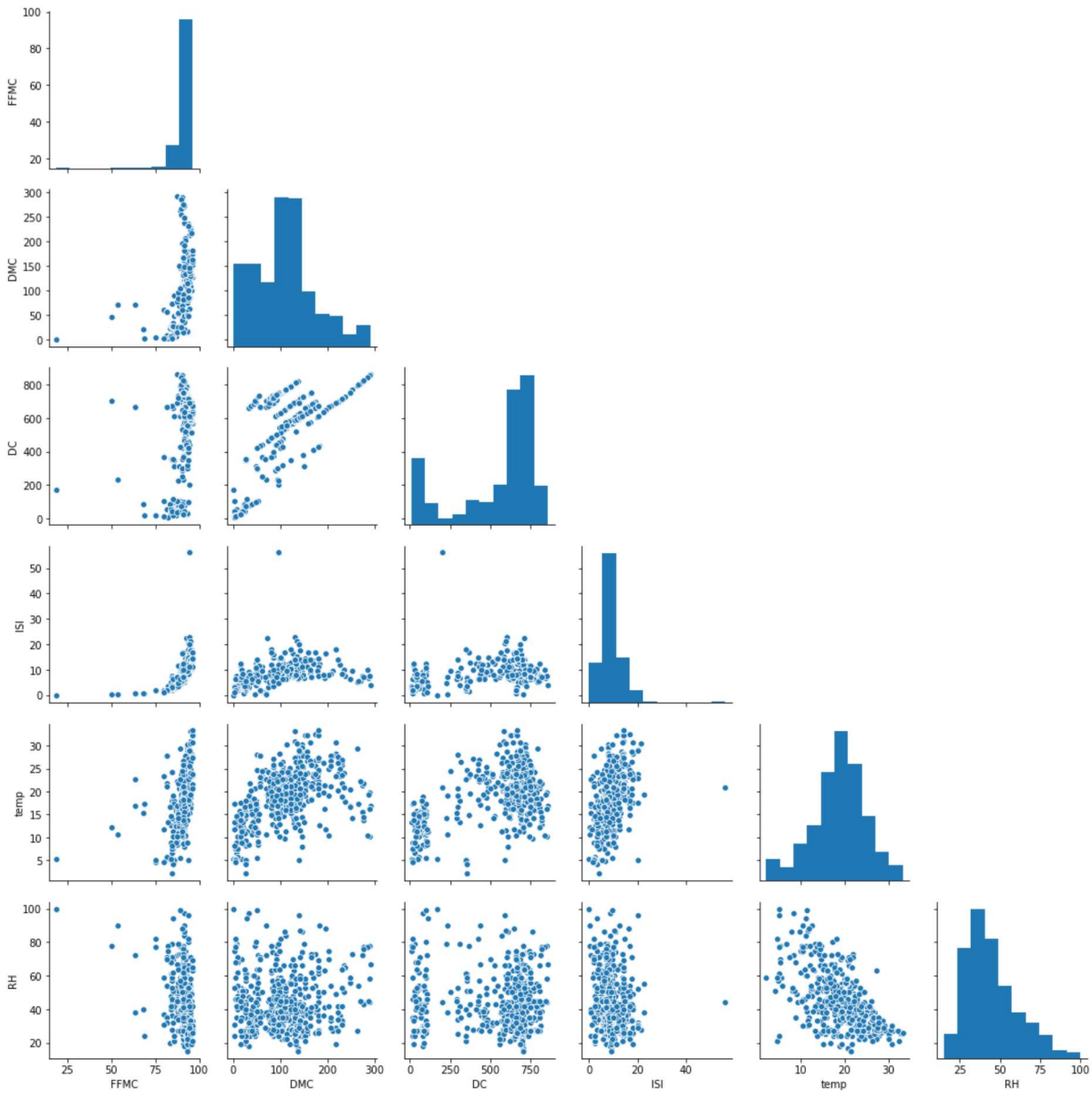
Where X is the dataset, i and j are related to the features being evaluated. This final matrix is symmetric ($\rho_{ij} = \rho_{ji}$) and the coefficients on the diagonal are equal to 1(as visible in the next graphs).

In [37]:
```
plt.figure(figsize=(6,5), dpi=100)

ax= sns.heatmap(forest_fires.corr(), fmt= '.1f', vmin=-1, vmax=1, center=0, li
newidths=.5,
    cmap=sns.diverging_palette(610, 730, n=100),
    square=True, annot=True).set_title("Correlation Matrix")

plt.show()
```



In [46]:
```
def hide_current_axis(*args, **kwds):
    plt.gca().set_visible(False)

plt.figure(figsize=(20,20), dpi=100)
print("Pairwise relationships without features(X,Y,rain, wind and area) with l
ess important results of correlation with the others because of graph's readab
ility")
df= forest_fires[['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH']]
sns.pairplot(df, kind="scatter").map_upper(hide_current_axis)
plt.show()
```

```
Pairwise relationships without features(X,Y,rain, wind and area) with less im
portant results of correlation with the others because of graph's readability

<Figure size 2000x2000 with 0 Axes>
```



The two graphs let to make some considerations about the features:

- the couple of features with the highest coefficient of correlation is DMC-DC, in fact both represent the moisture content of shallow and deep organic layers, which affect fire intensity.
- the couple of features with the lowest coefficient of correlation is temp-RH, this result sounds quite strange because usually temperature and humidity tend to be related.

Target feature

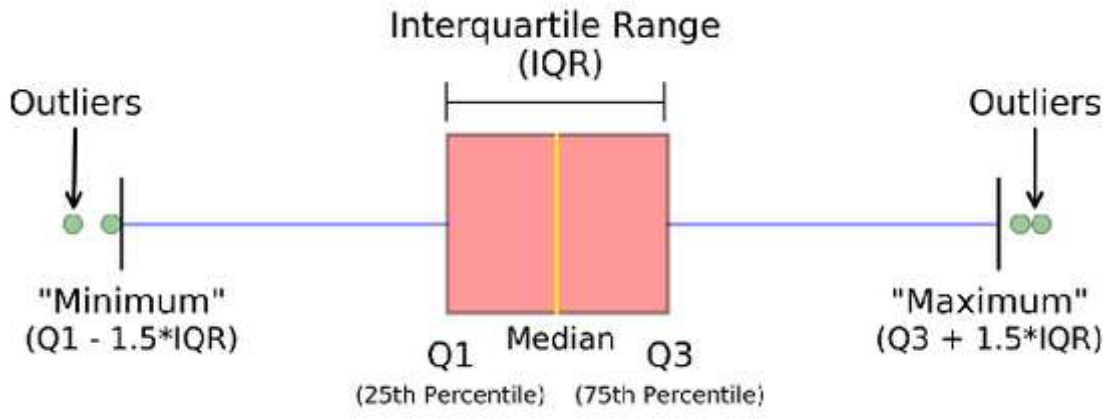
As anticipated in the introduction, the burned area will be used later as target of the model.
The previous correlation chapter already said that the area feature tend to be uncorrelated with all the other features.
According to the descriction of the features, a zero value means an area lower than 100 m^2.
Let's see how many records have an area greater than 100 m^2 and so may be more dangerous.

In [47]:
```
total_count = 0
positive_data_count = 0

for value in forest_fires['area']:
    if(value > 0):
        positive_data_count = positive_data_count + 1
    total_count = total_count + 1

print("The number of data records with 'burned area' > 0 are " + str(positive_
data_count) + " and the total number of records are " + str(total_count) + "."
)
print("The percentage value is " + str(positive_data_count/total_count * 100)
+ ".")
```

```
The number of data records with 'burned area' > 0 are 270 and the total numbe
r of records are 517.
The percentage value is 52.22437137330754.
```

Let's see with the following boxplot the distribution of its values.

Boxplot is a standardized way of displaying the dataset based on a five-number summary:



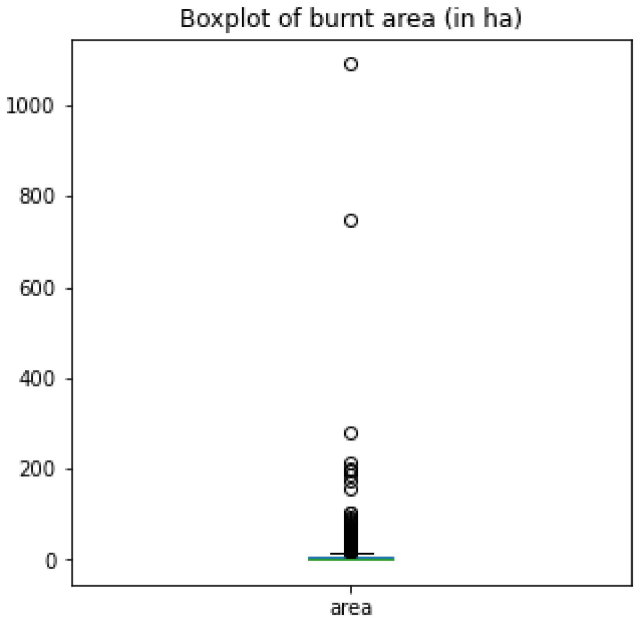
- **Minimum** : the lowest data point excluding any outliers.
- **Maximum** : the largest data point excluding any outliers.
- **Median** (Q2 / 50th percentile) : the middle value of the dataset.
- **First quartile** (Q1 / 25th percentile) : also known as the lower quartile qn(0.25), is the median of the lower half of the dataset.
- **Third quartile** (Q3 / 75th percentile) : also known as the upper quartile qn(0.75), is the median of the upper half of the dataset.

Important elements in Boxplot graphs are also:

- **Interquartile range** (IQR) : is the distance between the upper and lower quartiles.
- **Outliers**: individual points plotted as circles that stay outside the min-max representation.

One of the key aspects is to eliminate these outliers.

In [12]:
```
plt.figure(figsize=(5,5))
forest_fires['area'].plot(kind="box")
plt.title('Boxplot of burnt area (in ha)')
plt.show()
```



Burnt area is very skewed because as seen before there is just a 52% of data with an area greater than 0. In order to reduce skewness and improve symmetry can be applied a logarithmic function.

To obtain this result it will be applied the following transformation

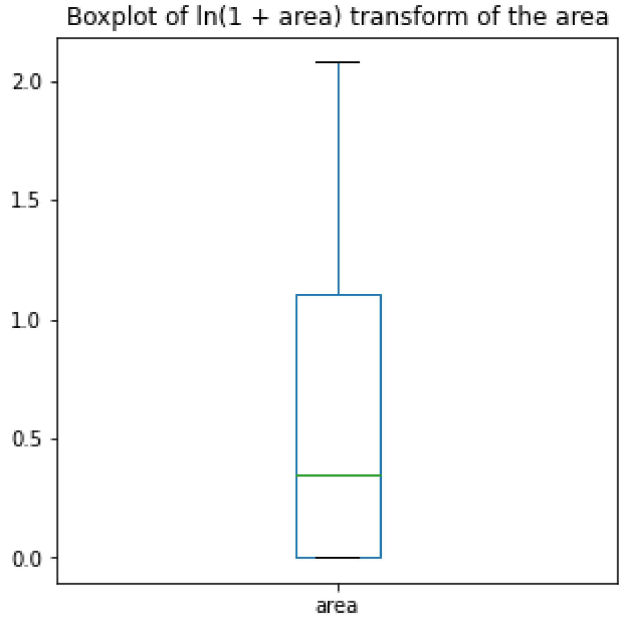
$$t(x) = \ln(x + 1)$$

According to this use of a logarithmic function, later on the model predictions will need to be post-processed with the inverse transformation:

$$t^{-1}(x) = e^x - 1$$

The effects of this trasformation are visible in the following boxplot.

In [13]:
```
forest_fires['area'] = np.log1p(forest_fires['area'])
plt.figure(figsize=(5,5))
forest_fires['area'].plot(kind="box")
plt.title('Boxplot of ln(1 + area) transform of the area')
plt.show()
```

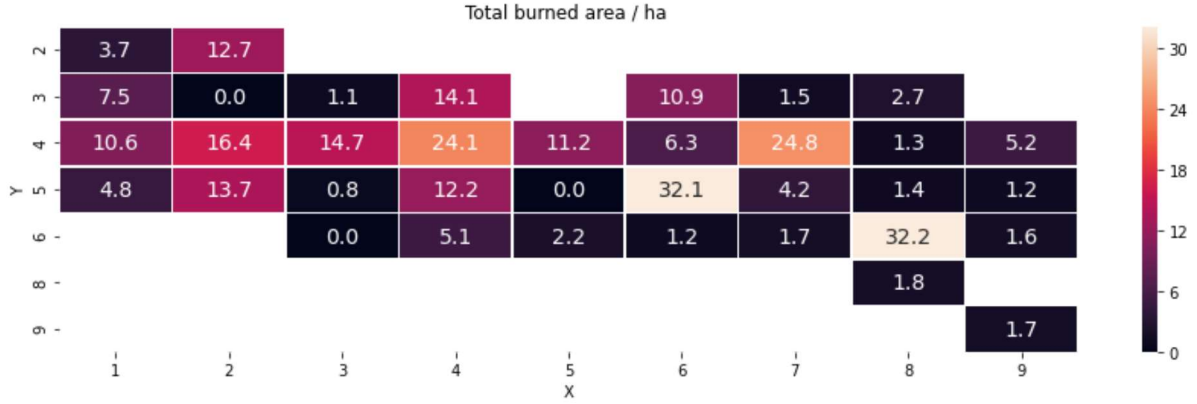


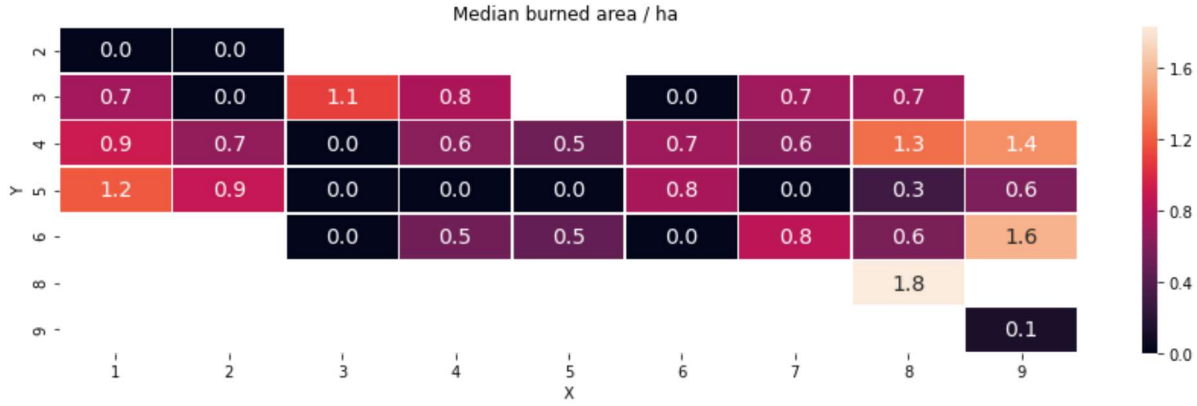
Using this log transformation the circles representing the outliers have been removed.
Let's see now the effects of this logarithmic function also on the spatial localization.

In [50]:

```
group = forest_fires.groupby(['X', 'Y'], as_index = False)

xy_sum = group[['area']].sum()
xy_sum = xy_sum.pivot('Y', 'X', 'area')

xy_cnt = group[['area']].count()
xy_cnt = xy_cnt.pivot('Y', 'X', 'area')

xy_med = group[['area']].median()
xy_med = xy_med.pivot('Y', 'X', 'area')

fig, axes = plt.subplots(nrows = 3, ncols = 1, figsize = (15,15))


_ = sns.heatmap(xy_cnt, annot=True, fmt=".1f", linewidths=.5, ax = axes[0],
                annot_kws = {'fontsize': 14}).set_title('Total fires / ha ')

_ = sns.heatmap(xy_sum, annot=True, fmt=".1f", linewidths=.5, ax = axes[1],
                annot_kws = {'fontsize': 14}).set_title('Total burned area / ha
')

_ = sns.heatmap(xy_med, annot=True, fmt=".1f", linewidths=.5, ax = axes[2],
                annot_kws = {'fontsize': 14}).set_title('Median burned area / h
a ')

plt.subplots_adjust(hspace=0.4, wspace=0.4)
plt.show()
```


Total fires / ha
19.0 25.0
10.0 1.0 1.0 22.0 25.0 2.0 3.0
15.0 27.0 43.0 36.0 23.0 9.0 45.0 1.0 4.0
4.0 20.0 7.0 25.0 3.0 49.0 11.0 4.0 2.0
4.0 8.0 4.0 3.0 2.0 52.0 1.0
1.0
6.0
Y
X
1 2 3 4 5 6 7 8 9
2 3 4 5 6 8 9
50 40 30 20 10


Total burned area / ha
3.7 12.7
7.5 0.0 1.1 14.1 10.9 1.5 2.7
10.6 16.4 14.7 24.1 11.2 6.3 24.8 1.3 5.2
4.8 13.7 0.8 12.2 0.0 32.1 4.2 1.4 1.2
0.0 5.1 2.2 1.2 1.7 32.2 1.6
1.8
1.7
Y
X
1 2 3 4 5 6 7 8 9
2 3 4 5 6 8 9
30 24 18 12 6 0


Median burned area / ha
0.0 0.0
0.7 0.0 1.1 0.8 0.0 0.7 0.7
0.9 0.7 0.0 0.6 0.5 0.7 0.6 1.3 1.4
1.2 0.9 0.0 0.0 0.0 0.8 0.0 0.3 0.6
0.0 0.5 0.5 0.0 0.8 0.6 1.6
1.8
0.1
Y
X
1 2 3 4 5 6 7 8 9
2 3 4 5 6 8 9
1.6 1.2 0.8 0.4 0.0

Now the fires tend to have a more uniform distribution within the entire park.

PREPROCESSING

Encoding data

In order to be able to use categorical variables in machine learning month and day columns need to be modified. To overcame this problem, the most commonly used technique is one hot encoding. Each category value is converted into a new column: is assigned 1 to the corresponding value and 0 to everything else.

Day of Week	One-Hot Encoding						
Sunday	1	0	0	0	0	0	0
Monday	0	1	0	0	0	0	0
Tuesday	0	0	1	0	0	0	0
Wednesday	0	0	0	1	0	0	0
Thursday	0	0	0	0	1	0	0
Friday	0	0	0	0	0	1	0
Saturday	0	0	0	0	0	0	1

Following this technique a new Dummy dataset will be created.

In [53]:
```
forest_fires['day'] = pd.Categorical(forest_fires['day'])
forest_fires['month'] = pd.Categorical(forest_fires['month'])

le = LabelEncoder()
forest_fires['day'] = le.fit_transform(forest_fires['day'])
forest_fires['month'] = le.fit_transform(forest_fires['month'])
# Creating dummy variables
day_dummies = pd.get_dummies(forest_fires['day'], prefix="day")
month_dummies = pd.get_dummies(forest_fires['month'], prefix="month")

forest_fires_dummy= pd.concat([forest_fires, day_dummies, month_dummies], axis
=1)
```

Let's see how this new dataset looks like

In [54]:
```
forest_fires_dummy.head()
```

Out[54]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	...	month_2	month_3	month_4	mc
0	7	5	7	0	86.2	26.2	94.3	5.1	8.2	51	...	0	0	0	
1	7	4	10	5	90.6	35.4	669.1	6.7	18.0	33	...	0	0	0	
2	7	4	10	2	90.6	43.7	686.9	6.7	14.6	33	...	0	0	0	
3	8	6	7	0	91.7	33.3	77.5	9.0	8.3	97	...	0	0	0	
4	8	6	7	3	89.3	51.3	102.2	9.6	11.4	99	...	0	0	0	

5 rows × 32 columns

Division of prediction features

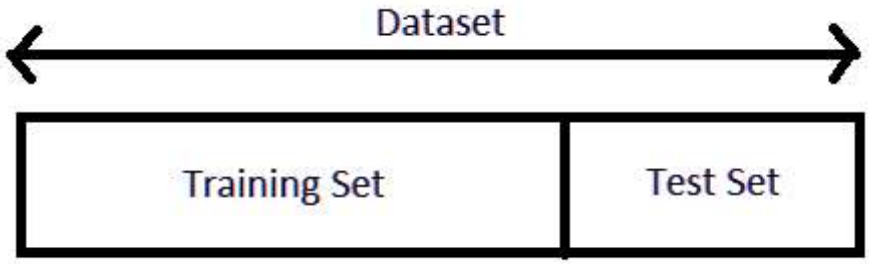
The first thing to do is to divide the column representing the target from the rest of the columns. The target value is used to perform the learning of the model and also the test on it. In this dataset, the target value is represented by the area feature.

In [59]:
```
X = forest_fires_dummy.drop('area', axis=1).values.astype(float)

Y = forest_fires_dummy['area']
```

DataSet partition

During the study of the various models is important to understand how the newly generated models will perform on data they have never seen. To do this, the dataset can be split into two parts:

- **Training set**: is the set where the model goes to learn;
- **Test set**: is the part of dataset where models can be evaluated.



The most common splits tend to use at least 2/3 of the entire dataset for the training part and to evaluate the model using the remaining 1/3. The division, in this case 70% for training set and 30% for test set, is implemented using the train_test_split library function which, given a proportion for test set size, returns the divided dataset.

In [60]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.3, rand
om_state=0)
```

Standardization

Standardization of a dataset is a common requirement for many machine learning algorithms. It is a type of data transformation, whose values are adapted into a smaller range of numeric values. It is used in order to have for all data a mean equal to 0 and a unit variance. In this case the z-score transformation can be chosen and it is defined as:

$$z = \frac{X - \mu}{\sigma}$$

Using the SandardScaler library class, the transformation can be calculated using the fit method on the training set and then applying the changes on the training set and test set.

In [61]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Pre-compute post-processed Ys
#needed after log trasformation of area
Y_train_post = np.expm1(Y_train)
Y_test_post = np.expm1(Y_test)
```

Dimensionality reduction

The use of one-hot encoding has increased the number of features to 31 so, having more features, we can run into a performance and memory space problem. It must be considered that not all features are relevant to the problem to be solved.

Principal Component Analysis PCA is a decomposition technique that extracts from a multivariate dataset a set of successive orthogonal components that explain a maximum amount of variance. The new coordinate system is composed by the -so called- principal components that are orthogonal linear transformation of the original features and are uncorrelated.

In other words, it is an approximation of the dataset in a lower dimensional space, preserving largest variances in the data.

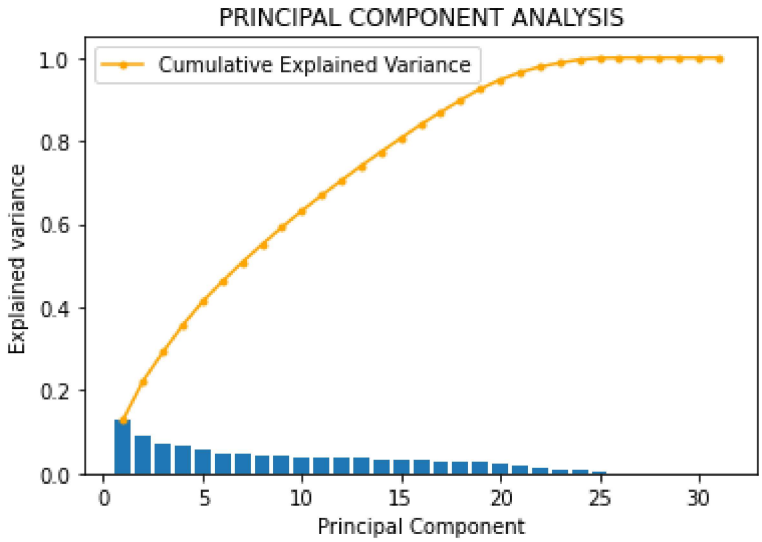
A useful measure to decide how many principal components to keep for the new feature space is the explained variance. It tells how much information (variance) can be attributed to each of the principal components, so that the proper number of components can be maintained depending on the target total variance we are willing to reach.

The following function will be used to see both PCA dataframe and PCA plot, results of PCA application on training set, and determine the right number of principal components to choose (they'll have at least to preserve the 90% of explained variance).

In [70]:

```
def tryPca(X_train_):
    pca = PCA()
    pca.fit(X_train_)

    n_components = len(pca.components_)
    pcaDataFrame =pd.DataFrame({'Principal Component': range(1, n_components+
1),
                  'Cumulative percentage of variance': np.cumsum(pca.explained_
variance_ratio_*100)})
    print(pcaDataFrame)
    plt.bar(range(1, n_components+ 1), pca.explained_variance_ratio_)
    plt.title("PRINCIPAL COMPONENT ANALYSIS")
    plt.ylabel('Explained variance')
    plt.xlabel('Principal Component')
    plt.plot(range(1, n_components+ 1), np.cumsum(pca.explained_variance_ratio
_),
             c='orange', marker='.', label='Cumulative Explained Variance')

    plt.legend();

tryPca(X_train)
```

	Principal Component	Cumulative percentage of variance
0	1	13.007006
1	2	22.233446
2	3	29.179076
3	4	35.635081
4	5	41.334497
5	6	46.201979
6	7	50.744938
7	8	55.062778
8	9	59.209246
9	10	63.155049
10	11	66.874030
11	12	70.458333
12	13	73.995560
13	14	77.348395
14	15	80.665063
15	16	83.910148
16	17	86.920752
17	18	89.782706
18	19	92.507451
19	20	94.704469
20	21	96.475744
21	22	97.819951
22	23	98.752595
23	24	99.537509
24	25	99.927030
25	26	100.000000
26	27	100.000000
27	28	100.000000
28	29	100.000000
29	30	100.000000
30	31	100.000000



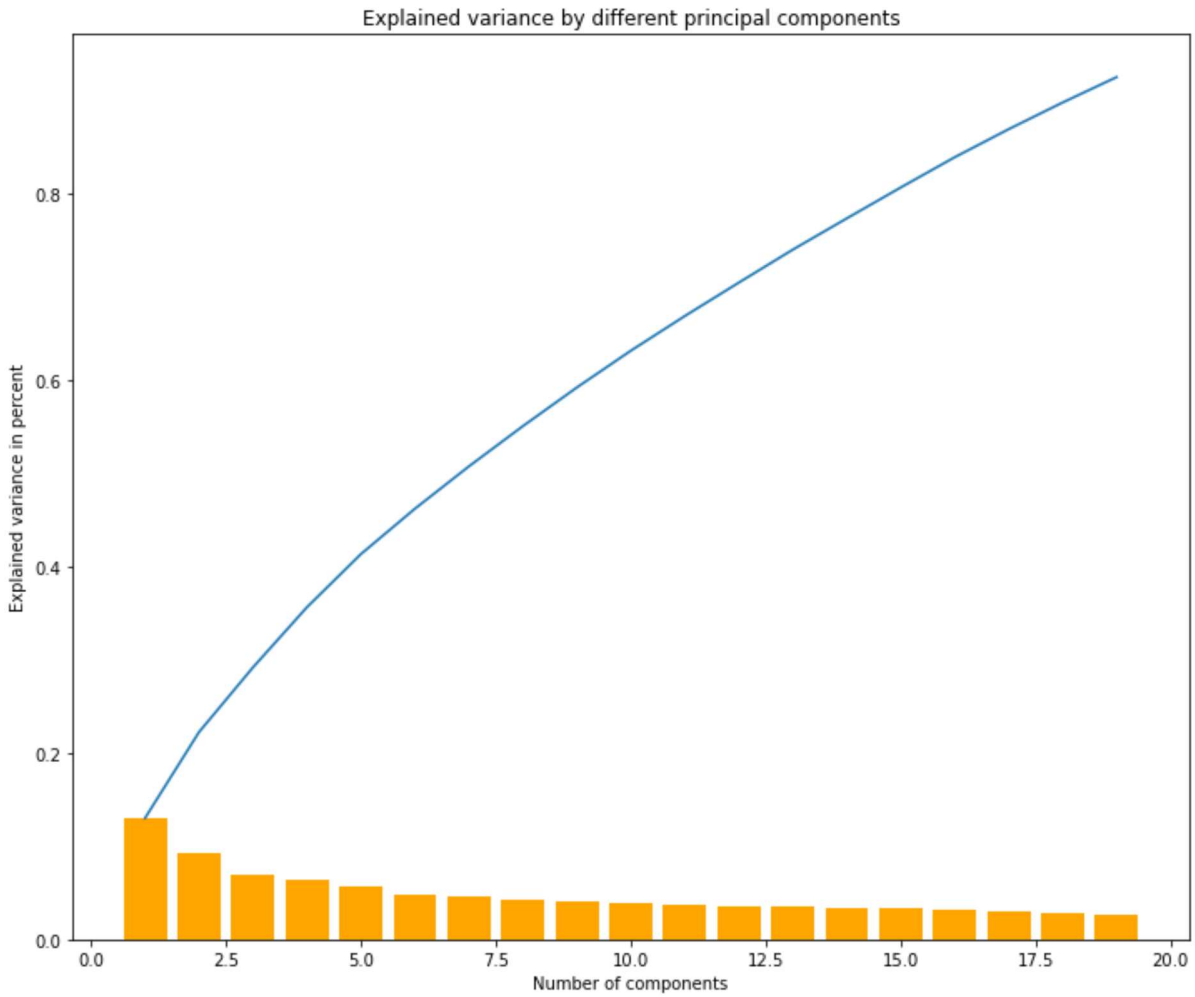
As visible in this graph, reducing the number of principal components from 31 to 19 can guarantee more than 90% of explained variance.

The following fuctions will use the selected number of principal component to make PCA and plot its cumulative explained variance.

In [71]:
```
def plotPCACumulativeExplainedVariance(pca):
    plt.figure(figsize=(12,10))
    plt.bar(np.arange(1, len(pca.explained_variance_ratio_)+1), pca.explained_
variance_ratio_, color='orange')
    plt.plot(np.arange(1, len(pca.explained_variance_ratio_)+1), np.cumsum(pca
.explained_variance_ratio_))
    plt.title('Explained variance by different principal components')
    plt.xlabel('Number of components')
    plt.ylabel('Explained variance in percent')
    plt.show()

def makePCA(X_train_, Y_train_, X_test_, n_comp, labels=None):
    pca = PCA(n_components=n_comp)
    pca.fit(X_train_)
    X_train_pca_ = pca.transform(X_train_)
    X_test_pca_ = pca.transform(X_test_)

    plotPCACumulativeExplainedVariance(pca)

    return X_train_pca_, X_test_pca_
```

Each time PCA is applied one graph is plotted. This graph describes the components obtained from PCA in terms of variance. It contains a bar plot for individual explained variance and a line plot for cumulative explained variance. The individual explained variance is the variance explained by a single component in percentage, and the cumulative explained variance is the cumulative sum of these percentages.

In [72]:
```
X_train_pca, X_test_pca = makePCA(X_train, Y_train, X_test,19, list(forest_fir
es_dummy))
```



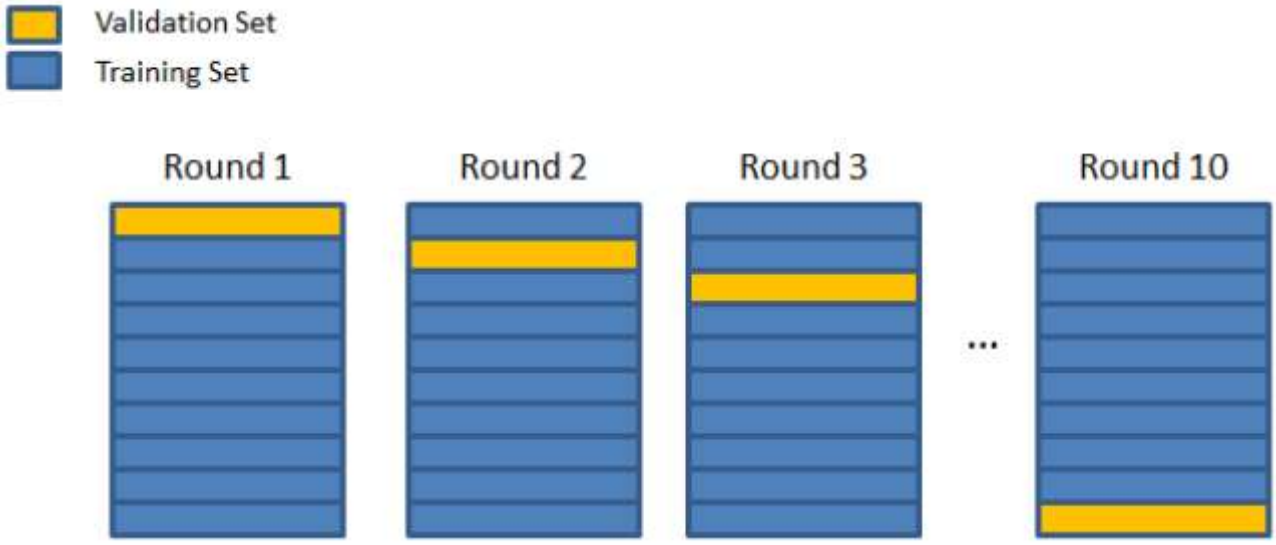
PCA reduces the number of components to 19 and explains more than the 90% of variance

MODEL SELECTION

For model evaluation there exist several partition techniques:

- **Holdout**: fixed partitioning in train and validation sets; less accurate than other methods, but the only one appropriate for large datasets;
- **Cross validation (or k-fold validation)**: first partition the dataset into k disjoint subsets (called folds), then train on $k-1$ partitions and test on the remaining one; repeating this method for each partition provides a reliable accuracy estimation, at the expense of a slower training
- **Leave-one-out**: variation on cross validation, each fold contains only one record. It is only feasible with small datasets.

In this case study cross validation is used with $k = 10$.



Metrics

As suggested in the paper(available in the references) made by the authors of this dataset,the metrics used to evaluate the models are:

Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

RMSE is more sensitive to errors than MAE

Algorithms

From now on will be presented the algorithms that will be used to compute the metrics both on PCA and NotPCA datasets.

The algorithms are:

- Linear Regression
- Random Forest Regression
- Support Vector Regression
- KNeighborsRegression

Later, comparing the results obtained, the best algorithm will be selected for the following part.

Linear Regression

This type of algorithm is the simplest that can be found in the habit of regression, it is based on a linear relationship between X and Y. The model can be defined starting from the one-dimensional case:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

The idea is to try to estimate the coefficients of the model ($\hat{\beta}_0$ and $\hat{\beta}_1$), so as to obtain an estimate of the target ($\hat{y}$). The algorithm is based on the calculation of the residuals:

$$e_i = y_i - \hat{y_i},\ i \in n\ (number\ of\ samples)$$

A functions that has to be optimized is defined as following:

$$Loss(\hat{y}, y) = \sum_{i}^{n} (y_i - \hat{\beta}_0 + \hat{\beta}_1 x_i)^2$$

This mode is called least square method which allows to choose the coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the loss function. In the multidimensional case, we can estimate p coefficients: $\hat{\beta}_0,\ \hat{\beta}_1,\ \ldots,\ \hat{\beta}_p$.
The resulting model is given by the following formulation:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \ \ldots\ + \hat{\beta}_p x_p$$

In [73]:
```
def makeLinearRegression(X_train_, Y_train_, X_test_, Y_test_post_):
    model = LinearRegression()

    model.fit(X_train_, Y_train_)

    Y_predict = model.predict(X_test_)
    # Post-processing
    Y_predict_post = np.expm1(Y_predict)

    print("RMSE:",np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_p
ost)))
    print("MAE:", metrics.mean_absolute_error(Y_test_post_, Y_predict_post))

    return np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_post)),
metrics.mean_absolute_error(Y_test_post_, Y_predict_post)
```

In [75]:
```
print("Linear Regression without PCA:")
lr_RMSE, lr_MAE = makeLinearRegression(X_train, Y_train, X_test, Y_test_post)
print("Linear Regression with PCA:")
lr_pca_RMSE, lr_pca_MAE = makeLinearRegression(X_train_pca, Y_train, X_test_pc
a, Y_test_post)
```

```
Linear Regression without PCA:
RMSE: 1.6295207840827597
MAE: 1.2405518462189002
Linear Regression with PCA:
RMSE: 1.601680987515177
MAE: 1.2161072063478902
```

For the Linear Regression model, PCA has been able to reduce both errors.

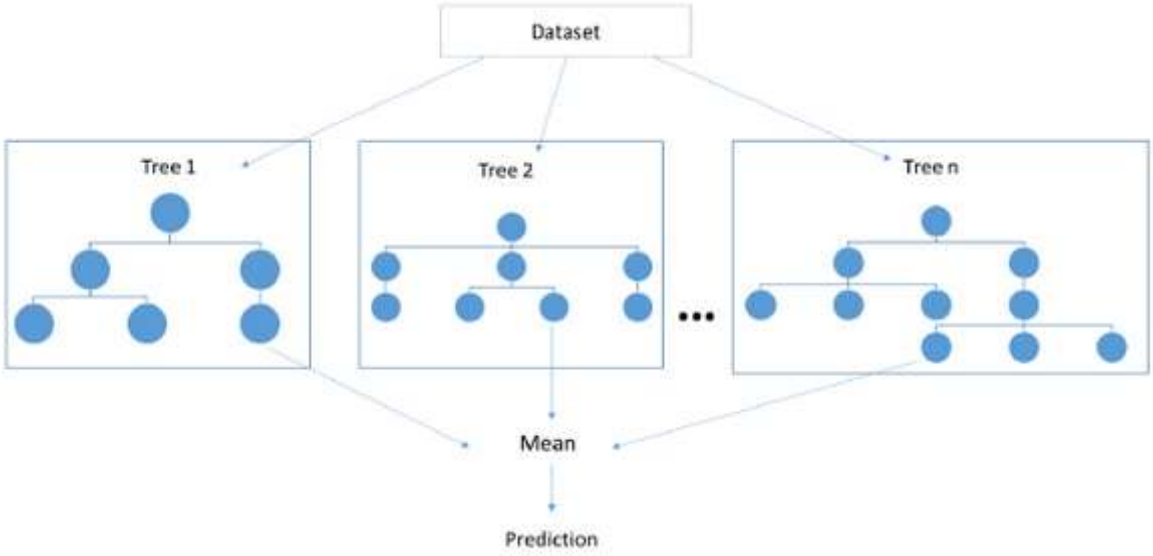
Random Forest

Random forests are part of the so called ensamble techniques, that combine together different models in order to improve accuracy and stability over a single model.

With random forest a set of decision trees is trained independently and the final outcome is obtained by means of majority vote.

In details, given a training set D of n instances, B decision trees are trained, each one using $n' \leq n$ random samples taken from D with replacement. For each split in the learning process a random subset of $\sqrt{p}$ features can be selected among the total p features of the initial training set D.

This ensures that the trees are decorrelated, hence different features could be selected as best attribute for the split.
It usually produces better results then a single Regression Tree, with an higher predictive accuracy and better control of over-fitting.



Search for the best parameter will be performed thanks to the GridSearch method (max depth as stopping criterion and different split quality measures like MAE and MSE).

In [76]:
```
def makeRandomForest(X_train_, Y_train_, X_test_, Y_test_post_):
    randomForest = GridSearchCV(estimator=RandomForestRegressor(n_estimators=1
00),
                                param_grid={'max_depth': [1,5, 10, 15, 20], 'c
riterion': ["mse", "mae"]},
                                cv=10, n_jobs=-1)
    randomForest.fit(X_train_, Y_train_)
    print("Best parameters: ", randomForest.best_params_)
    Y_predict = randomForest.predict(X_test_)
    # Post-processing
    Y_predict_post = np.expm1(Y_predict)

    print("RMSE:", np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_
post)))
    print("MAE:", metrics.mean_absolute_error(Y_test_post_, Y_predict_post))

    return np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_post)),
metrics.mean_absolute_error(Y_test_post_, Y_predict_post)
```

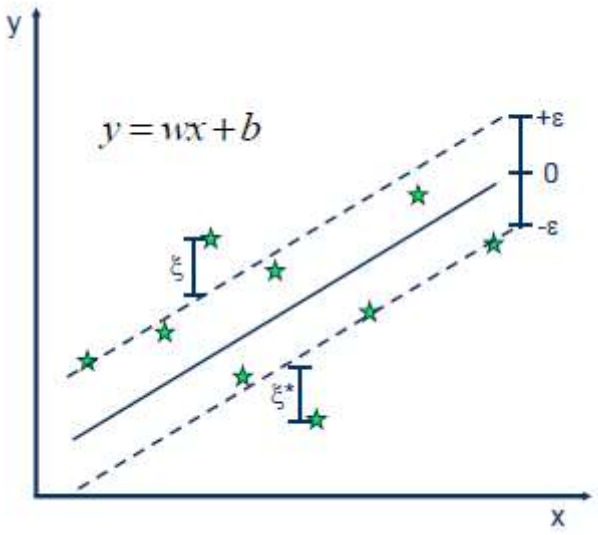
In [77]:
```
print("Random Forest without PCA:")
rf_RMSE, rf_MAE = makeRandomForest(X_train, Y_train, X_test, Y_test_post)
print("Random Forest with PCA:")
rf_pca_RMSE, rf_pca_MAE = makeRandomForest(X_train_pca, Y_train, X_test_pca, Y
_test_post)
```

```
Random Forest without PCA:
Best parameters:  {'criterion': 'mse', 'max_depth': 1}
RMSE: 1.6139386764560677
MAE: 1.2112251039396296
Random Forest with PCA:
Best parameters:  {'criterion': 'mse', 'max_depth': 1}
RMSE: 1.622698836130552
MAE: 1.2349724440637904
```

In the Random Forest case the use of PCA doesn't change the best parameters selected by the GridSearch and results also less useful because both the errors tend to be higher than in the no PCA case.

SVR

Support Vector Machine is an algorithm that tries to find a hyperplane which separates the classes as best as possible and this is achieved maximizing the margin defined by the closest points in the hyperplane's neighbourhood, called support vectors. This algorithm can be used also in the case of regression. In this case you take the mode of support vector regression and use the same concepts used in the classification. At the end of the process you always get a maximal margin, also keeping in mind a possible error tolerance.



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right)$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon + \xi_i$$
$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \ge 0$$

In this case the problem turns:

$$min\ \frac{1}{2}||w||^2 + C\sum_{i}^{N}|\xi_i|$$
$$subject\ to: |y_i - \langle w, x_i\rangle| \le \epsilon + |\xi_i|,\ \ \xi_i \ge 0$$

- ϵ: it is a margin of error, which allows you to consider the points in the margin as tolerated errors.
- ξ_i: they are called slack variables. In the case of regression, it is a value assigned to points outside the margin.
- regularization parameter C: represents the trade-off between smoothness and correctness. With a large C, more train points are classified correctly and the resulting margin is smaller. With a small C, the boundary is smoother and the models generalize better (less prone to overfit); in this case the margin is larger. If $C = 0$ a hard-margin classifier is obtained, not allowing samples to be misclassified;
- gamma γ: parameter of the Gaussian/RBF (radial basis function) kernel; a high value of gamma can lead to overfitting.

In case of classification, mapping allows to map the points into the space where they are linearly separable and then apply the dot product. However, this operation can be very expensive.

Instead in the case of regression, the mapping performs one possible way to make the curve more flexible to the training data.

Also to solve the problem and to avoid mapping points in another space, can be used the concept of kernel. The kernel function must meet the following requirement:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle$$

This important result lead us to the conclusion that is enough to know the kernel function in order to resolve the maximization problem. The guidelines for a function to be considered kernel are given by the **Mercer's theorem**:

- The kernel must be symmetric:
$$k(x, y) = k(y, x)$$
- The Gram matrix must be positive semi-definite,it is defined as:
$$G_{i,j} = k(x_i, x_j)$$

Using a kernel is more efficient than mapping into another space. In this case, using GridSearch method were considered two types of kernels:

- Linear kernel:
$$k(x, x') = \langle x, x' \rangle$$
- Residual Basis Function (rbf):
$$k(x, x') = e^{-\gamma |x - x'|^2}$$

In [78]:
```
def makeSVR(X_train_, Y_train_, X_test_, Y_test_post_):
    svr = GridSearchCV(estimator=SVR(),
                                param_grid={'C': [0.001, 0.01, 0.1, 1, 10], 'g
amma': [0.001, 0.01, 0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'epsilon': [0.0
1, 0.1, 1, 10, 100]},
                                cv=10, n_jobs=-1)
    svr.fit(X_train_, Y_train_)
    print("Best parameters: ", svr.best_params_)
    Y_predict = svr.predict(X_test_)
    # Post-processing
    Y_predict_post = np.expm1(Y_predict)

    print("RMSE:", np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_
post)))
    print("MAE:", metrics.mean_absolute_error(Y_test_post_, Y_predict_post))

    return np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_post)),
metrics.mean_absolute_error(Y_test_post_, Y_predict_post)
```

In [79]:
```
print("SVR without PCA:")
svr_RMSE, svr_MAE = makeSVR(X_train, Y_train, X_test, Y_test_post)
print("SVR with PCA:")
svr_pca_RMSE, svr_pca_MAE = makeSVR(X_train_pca, Y_train, X_test_pca, Y_test_p
ost)
```

```
SVR without PCA:
Best parameters:  {'C': 1, 'epsilon': 0.1, 'gamma': 10, 'kernel': 'rbf'}
RMSE: 1.5910228113789509
MAE: 1.1824209848701162
SVR with PCA:
Best parameters:  {'C': 1, 'epsilon': 0.1, 'gamma': 10, 'kernel': 'rbf'}
RMSE: 1.598285044108477
MAE: 1.1845422590810086
```

Also in this case PCA doesn't change the best parameters and doesn't reduce the errors.

K-NN

Also this algorithm can be extended to the case of regression. This technique allows to fix k points of the training set to evaluate the prediction.

A possible pseudo algorithm can be, for p in point:

- select k points close the point p
- assign:

$$\hat{y} : \hat{y} = \frac{1}{K} \sum_{i}^{n} y_i$$

The algorithm proceeds as follows:

- compute the distance $d(x, x_i)$ to every training example x_i;
- select the k closest instances and their labels;
- output the class $\hat{y}$ which is the most frequent among the k selected.

As can be seen from the pseudo-algorithm, this type of model is based on the definition of distance. In this case two distances were used for the evaluation:

- Manhattan (p = 1):

$$ManhattanDistance = \sum_{i=1}^{k} |x_i - y_i|$$

- Euclidean (p = 2).

$$EuclideanDistance = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

- These are actually both forms of what is called ‘Minkowski distance’, whose formula is:

$$d(X, Y) = (\sum_{i}^{k} (x_i - y_i)^p)^{\frac{1}{p}}$$

In addition to two types of distances were evaluated different number of neighbors.

In [80]:
```
def makeKNN(X_train_, Y_train_, X_test_, Y_test_post_):
    knn = GridSearchCV(estimator=KNeighborsRegressor(),
                                    param_grid={'n_neighbors': [1, 3, 5, 7, 9, 13,
15, 17, 19], 'p': [1, 2]},
                                    cv=10, n_jobs=-1)
    knn.fit(X_train_, Y_train_)
    print("Best parameters: ", knn.best_params_)
    Y_predict = knn.predict(X_test_)
    # Post-processing
    Y_predict_post = np.expm1(Y_predict)

    print("RMSE:", np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_
post)))
    print("MAE:", metrics.mean_absolute_error(Y_test_post_, Y_predict_post))

    return np.sqrt(metrics.mean_squared_error(Y_test_post_, Y_predict_post)),
metrics.mean_absolute_error(Y_test_post_, Y_predict_post)
```

In [81]:
```
print("K-NN without PCA:")
knn_RMSE, knn_MAE = makeKNN(X_train, Y_train, X_test, Y_test_post)
print("K-NN with PCA:")
knn_pca_RMSE, knn_pca_MAE = makeKNN(X_train_pca, Y_train, X_test_pca, Y_test_p
ost)
```

```
K-NN without PCA:
Best parameters:  {'n_neighbors': 19, 'p': 2}
RMSE: 1.6473052894061866
MAE: 1.2444997415218144
K-NN with PCA:
Best parameters:  {'n_neighbors': 19, 'p': 1}
RMSE: 1.6356103558284811
MAE: 1.2393121139031011
```

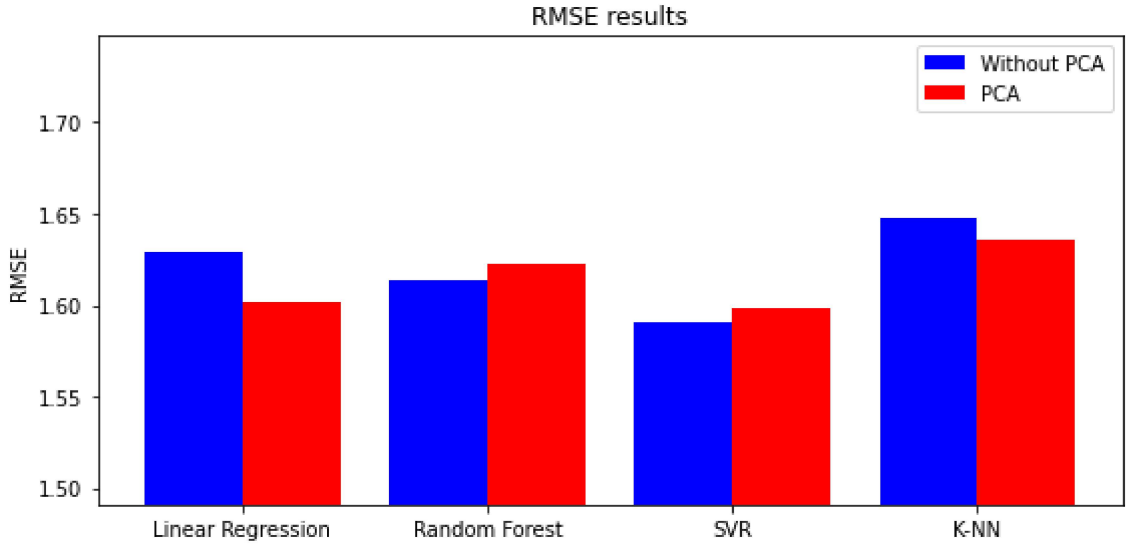
In K-NN PCA algorithm there is the first change of parameters selected (Manhattan distance and not Euclidean) and also an improvement with the errors.

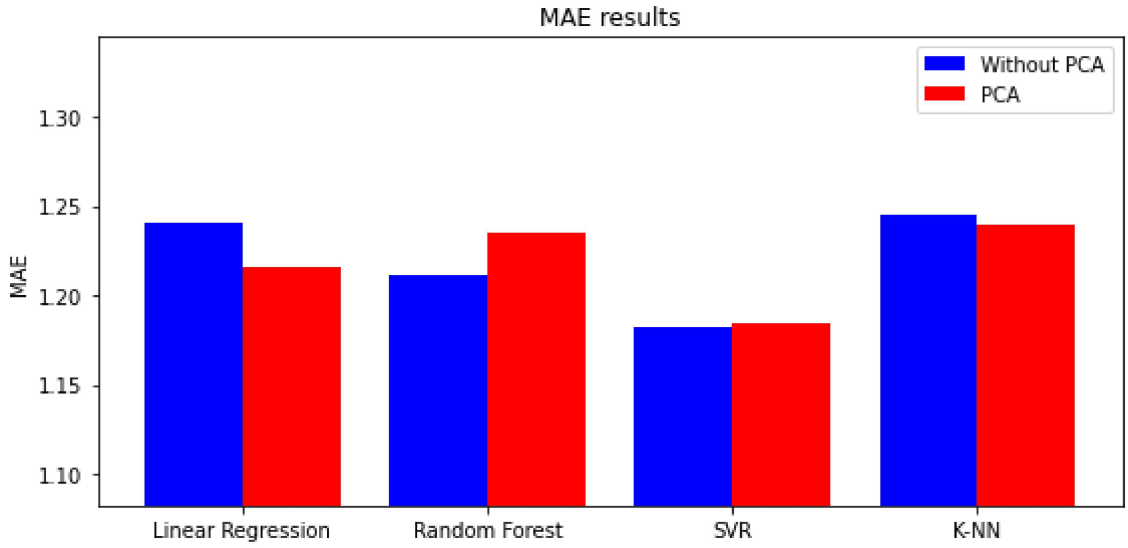
Model selection results

Let's see a final recap about the models and the errors that they lead to. The best algorithm will be selected for the following part.

In [82]:
```
def plotResult(RMSEs, RMSEs_pca, MAEs, MAEs_pca,labels):

    x = np.arange(len(labels))
    width = 0.4
    fig, ax = plt.subplots(figsize=(8,4))
    rects1 = ax.bar(x - width/2, RMSEs, width, label='Without PCA',color='blu
e')
    rects2 = ax.bar(x + width/2, RMSEs_pca, width, label='PCA',color='red')

    ax.set_ylabel('RMSE')
    ax.set_title('RMSE results')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.legend()
    plt.ylim(min(RMSEs) - 0.1, max(RMSEs) + 0.1)
    fig.tight_layout()
    plt.show()


    x = np.arange(len(labels))
    width = 0.4
    fig, ax = plt.subplots(figsize=(8,4))
    rects1 = ax.bar(x - width/2, MAEs, width, label='Without PCA', color='blu
e')
    rects2 = ax.bar(x + width/2, MAEs_pca, width, label='PCA', color='red')

    ax.set_ylabel('MAE')
    ax.set_title('MAE results')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.legend()
    plt.ylim(min(MAEs) - 0.1, max(MAEs) + 0.1)
    fig.tight_layout()
    plt.show()
```

In [83]:
```
RMSEs = [lr_RMSE, rf_RMSE, svr_RMSE, knn_RMSE]
RMSEs_pca = [lr_pca_RMSE, rf_pca_RMSE, svr_pca_RMSE, knn_pca_RMSE]
MAEs = [lr_MAE, rf_MAE, svr_MAE, knn_MAE]
MAEs_pca = [lr_pca_MAE, rf_pca_MAE, svr_pca_MAE, knn_pca_MAE]
plotResult(RMSEs, RMSEs_pca, MAEs, MAEs_pca,['Linear Regression', 'Random Fore
st', 'SVR', 'K-NN'])
```





As we can see from the tables, the application of PCA guarantees better result only Linear Regression and K-NN methods. For both the metrics MAE and RMSE, the algorithm that guarantees lower errors is the SVR and it will be used for the last step of the project.

FEATURES SELECTION

The dataset is made up of multiple sources. To infer about the impact of each of them, in this paragraph will be applied features selection, also reducing the dimensionality of the dataset. The four selections made, suggested by the authors of the dataset in their paper, are the following:

- **STFWI**: containing spatial, temporal features and the four components of the FWI index;
- **STM**: containing spatial, temporal and meteorological features;
- **FWI**: with only the components of the FWI index;
- **W**: with only weather features.

Also in this case will be made a confront between the results obtained with and without the use of PCA, excluding the last two selections because they contain only 4 features and PCA doesn't change their results. Considering the well balanced content of each selection, the idea is that PCA will not improve the results.
For each of the features selection will be used only an algorithm, the SVR, wich performed better in the model selection.

STFWI

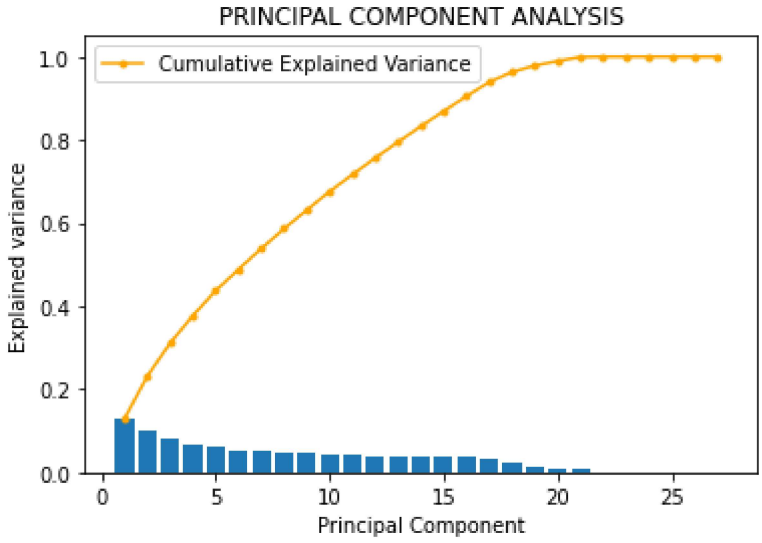
The STFWI feature selection stands for spatial, temporal and the four FWI components. In this process, the involved features are:

- X
- Y
- month
- day
- FFMC
- DMC
- DC
- ISI

In [84]:
```
X_STFWI = forest_fires_dummy.drop(columns=['temp', 'RH', 'wind', 'rain', 'are
a']).values
Y_STFWI = forest_fires_dummy['area'].values
```

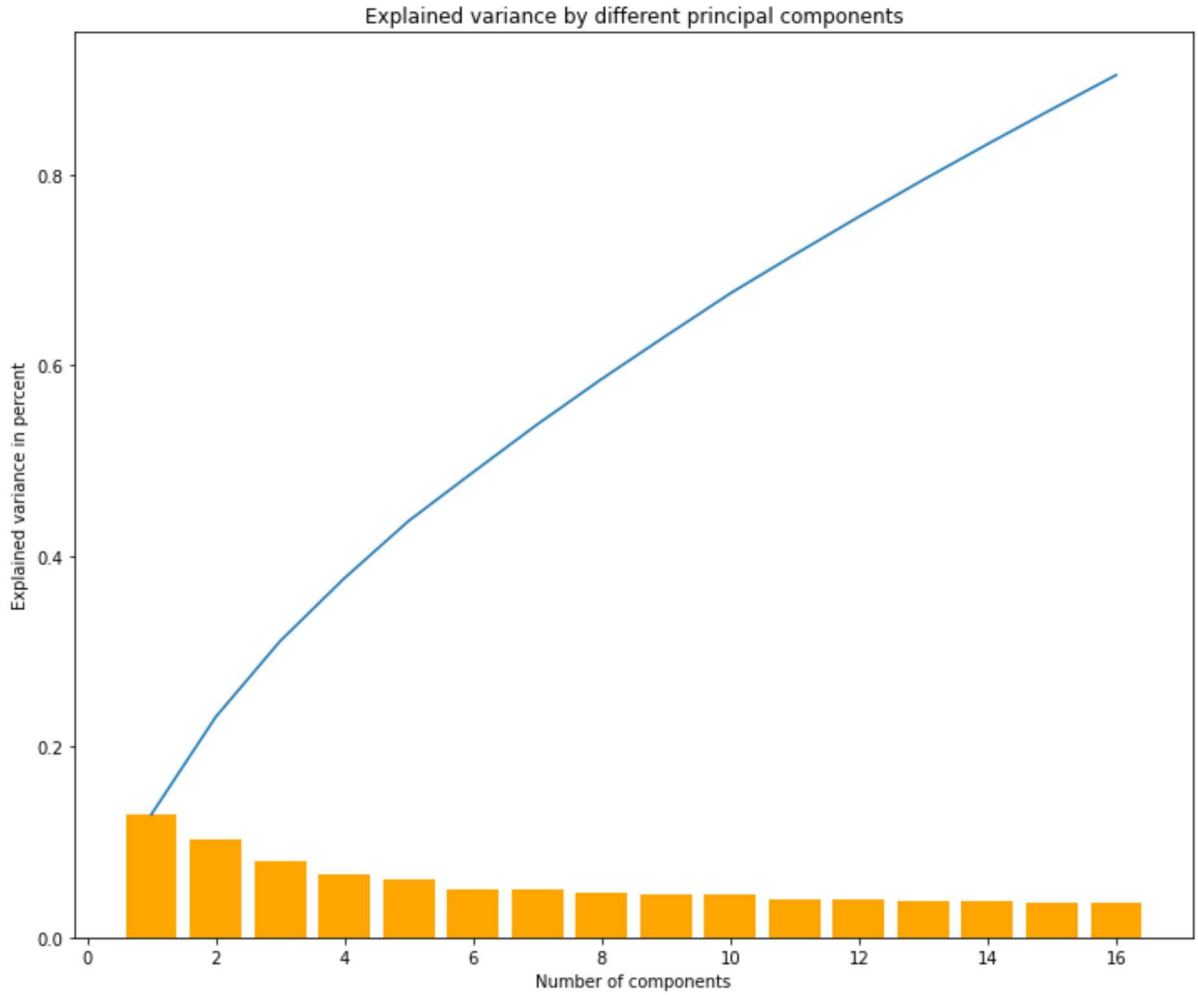
In [85]:
```
X_train_STFWI, X_test_STFWI, Y_train_STFWI, Y_test_STFWI = train_test_split( X
_STFWI, Y_STFWI, test_size=0.30, random_state=0)
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train_STFWI)
# Apply transform to both the training set and the test set.
X_train_STFWI = scaler.transform(X_train_STFWI)
X_test_STFWI = scaler.transform(X_test_STFWI)
# Pre-compute post-processed Ys
Y_train_STFWI_post = np.expm1(Y_train_STFWI)
Y_test_STFWI_post = np.expm1(Y_test_STFWI)
tryPca(X_train_STFWI)
```

	Principal Component	Cumulative percentage of variance
0	1	12.897942
1	2	23.114878
2	3	31.069198
3	4	37.672021
4	5	43.690398
5	6	48.794494
6	7	53.821532
7	8	58.579531
8	9	63.107620
9	10	67.564215
10	11	71.630760
11	12	75.610308
12	13	79.479798
13	14	83.226826
14	15	86.889643
15	16	90.472811
16	17	93.907578
17	18	96.308550
18	19	97.871108
19	20	98.937299
20	21	99.914973
21	22	100.000000
22	23	100.000000
23	24	100.000000
24	25	100.000000
25	26	100.000000
26	27	100.000000



In this case reducing the number of components to 16 can guarantee a 90% of explained variance.

In [86]:
```
X_train_pca_STFWI, X_test_pca_STFWI = makePCA(X_train_STFWI, Y_train_STFWI, X_
test_STFWI,16, list(X_STFWI))
```



In [88]:
```
print("SVR without PCA for STFWI selection::")
svr_RMSE_STFWI, svr_MAE_STFWI = makeSVR(X_train_STFWI, Y_train_STFWI, X_test_S
TFWI, Y_test_STFWI_post)
print("SVR with PCA for STFWI selection:")
svr_pca_RMSE_STFWI, svr_pca_MAE_STFWI = makeSVR(X_train_pca_STFWI, Y_train_STF
WI, X_test_pca_STFWI, Y_test_STFWI_post)
```

```
SVR without PCA for STFWI selection::
Best parameters:  {'C': 10, 'epsilon': 0.1, 'gamma': 10, 'kernel': 'rbf'}
RMSE: 1.6659015267905788
MAE: 1.2113028492193996
SVR with PCA for STFWI selection:
Best parameters:  {'C': 10, 'epsilon': 0.1, 'gamma': 10, 'kernel': 'rbf'}
RMSE: 1.704178016096849
MAE: 1.2355208057879161
```

The STFWI selection is already good and doesn't need PCA to improve the results.

STM

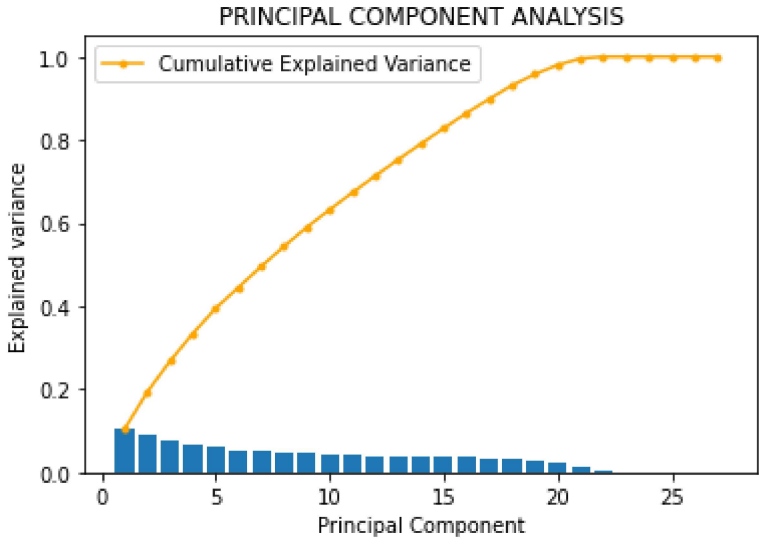
The STM feature selection stands for spatial, temporal and the four weather variables. In this process, the involved features are:

- X
- Y
- month
- day
- temp
- RH
- wind
- rain

In [89]:
```
X_STM = forest_fires_dummy.drop(columns=['FFMC', 'DMC', 'DC', 'ISI', 'area']).
values
Y_STM = forest_fires_dummy['area'].values
```

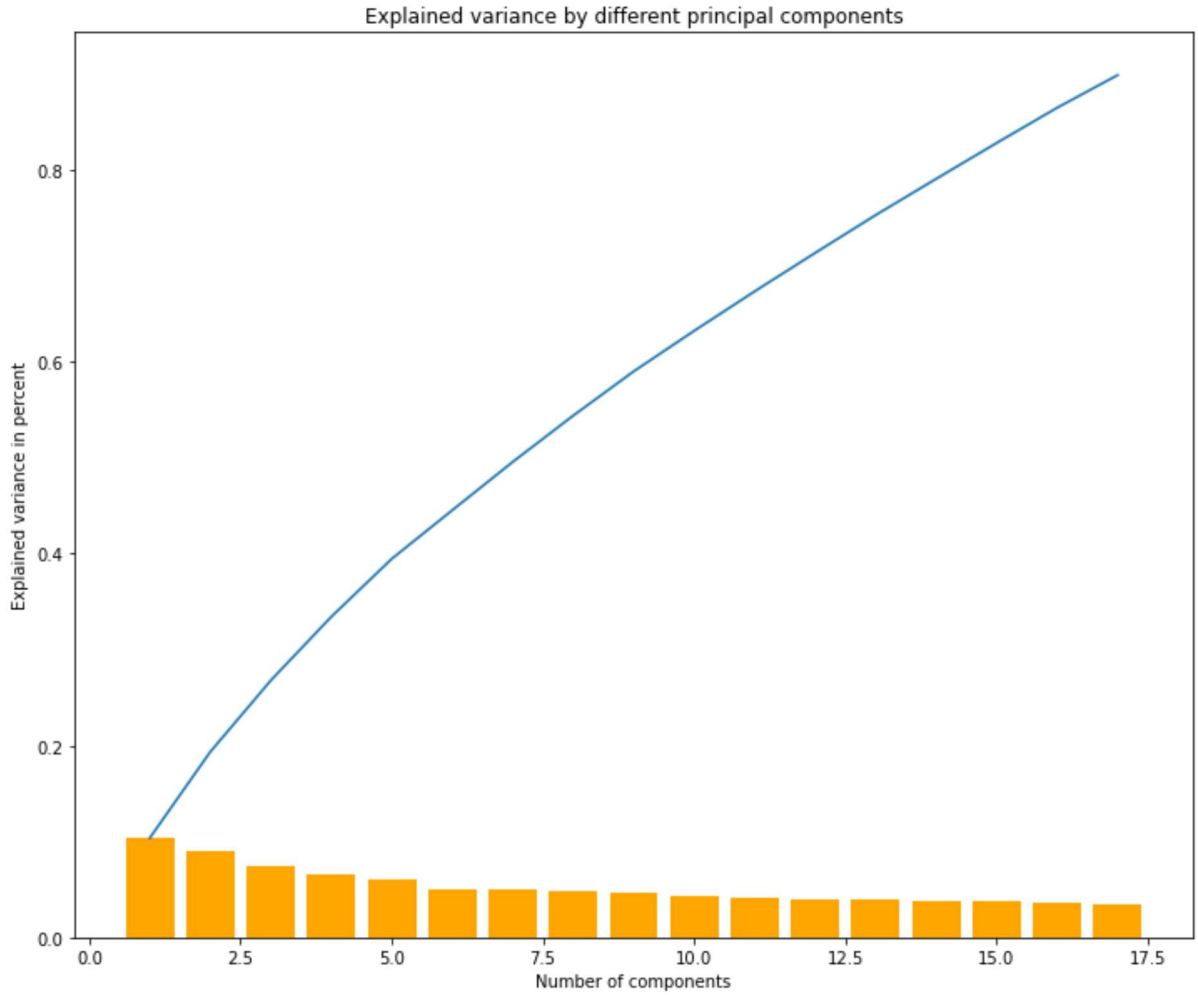
In [90]:
```
X_train_STM, X_test_STM, Y_train_STM, Y_test_STM = train_test_split( X_STM, Y_
STM, test_size=0.30, random_state=0)

scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train_STM)
# Apply transform to both the training set and the test set.
X_train_STM = scaler.transform(X_train_STM)
X_test_STM = scaler.transform(X_test_STM)

# Pre-compute post-processed Ys
Y_train_STM_post = np.expm1(Y_train_STM)
Y_test_STM_post = np.expm1(Y_test_STM)
tryPca(X_train_STM)
```

	Principal Component	Cumulative percentage of variance
0	1	10.354955
1	2	19.327310
2	3	26.762091
3	4	33.396103
4	5	39.435024
5	6	44.496519
6	7	49.532177
7	8	54.353231
8	9	58.965687
9	10	63.188977
10	11	67.301491
11	12	71.304888
12	13	75.244968
13	14	79.020575
14	15	82.758604
15	16	86.427133
16	17	89.827572
17	18	93.096894
18	19	95.792300
19	20	98.009197
20	21	99.561420
21	22	100.000000
22	23	100.000000
23	24	100.000000
24	25	100.000000
25	26	100.000000
26	27	100.000000



In this dataset reducing the number of components to 17 explains 90% of variance.

In [91]:
```
X_train_pca_STM, X_test_pca_STM = makePCA(X_train_STM, Y_train_STM, X_test_STM
,17, list(X_STM))
```



In [92]:
```
print("SVR without PCA for STM selection:")
svr_RMSE_STM, svr_MAE_STM = makeSVR(X_train_STM, Y_train_STM, X_test_STM, Y_te
st_STM_post)
print("SVR with PCA for STM selection:")
svr_pca_RMSE_STM, svr_pca_MAE_STM = makeSVR(X_train_pca_STM, Y_train_STM, X_te
st_pca_STM, Y_test_STM_post)
```

```
SVR without PCA for STM selection:
Best parameters:  {'C': 1, 'epsilon': 0.1, 'gamma': 10, 'kernel': 'rbf'}
RMSE: 1.592757395235446
MAE: 1.1832044323570163
SVR with PCA for STM selection:
Best parameters:  {'C': 1, 'epsilon': 0.1, 'gamma': 10, 'kernel': 'rbf'}
RMSE: 1.6042598164164172
MAE: 1.185638633130634
```

Also in this case PCA didn't improve the result, original dataset is already well balanced.

FWI

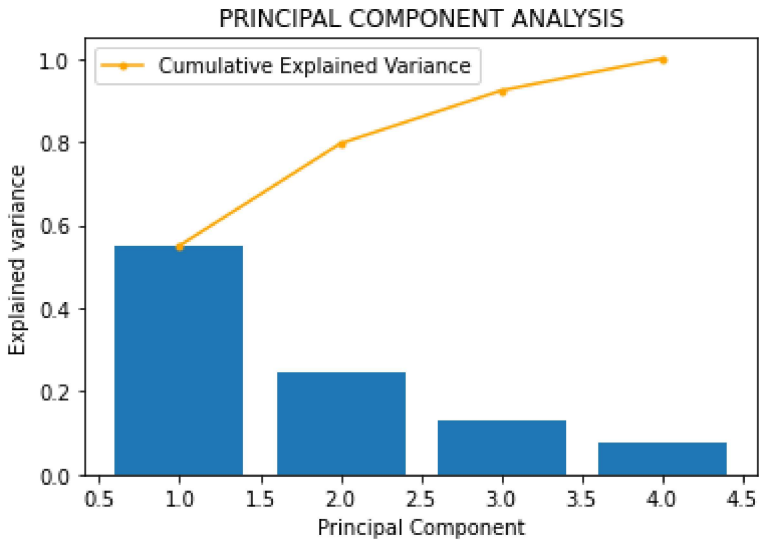
The FWI feature selection stands for the four FWI components. In this process, the involved features are:

- FFMC
- DMC
- DC
- ISI

In [93]:
```
X_FWI = forest_fires_dummy[['FFMC', 'DMC', 'DC', 'ISI']].values
Y_FWI = forest_fires_dummy['area'].values
```

In [94]:
```
X_train_FWI, X_test_FWI, Y_train_FWI, Y_test_FWI = train_test_split( X_FWI, Y_
FWI, test_size=0.30, random_state=0)

scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train_FWI)
# Apply transform to both the training set and the test set.
X_train_FWI = scaler.transform(X_train_FWI)
X_test_FWI = scaler.transform(X_test_FWI)
# Pre-compute post-processed Ys
Y_train_FWI_post = np.expm1(Y_train_FWI)
Y_test_FWI_post = np.expm1(Y_test_FWI)
tryPca(X_train_FWI)
```

	Principal Component	Cumulative percentage of variance
0	1	55.046378
1	2	79.595124
2	3	92.344218
3	4	100.000000



In this case the number of components is already small, the entire number will be kept.

In [95]:
```
print("SVR without PCA for FWI selection:")
svr_RMSE_FWI, svr_MAE_FWI = makeSVR(X_train_FWI, Y_train_FWI, X_test_FWI, Y_te
st_FWI_post)
```

```
SVR without PCA for FWI selection:
Best parameters:  {'C': 0.1, 'epsilon': 0.1, 'gamma': 10, 'kernel': 'rbf'}
RMSE: 1.7658912962974755
MAE: 1.2343548917430955
```

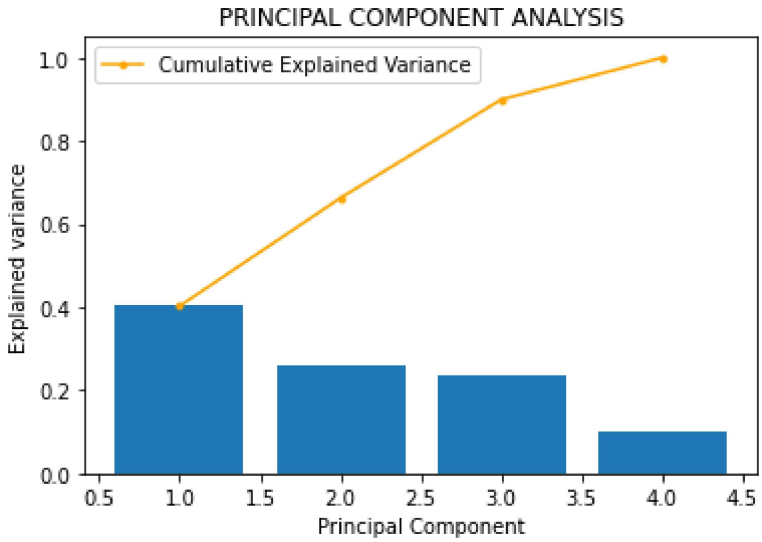
Weather Conditions

The W feature selection stands the weather conditions. In this process, the involved features are:

- temp
- RH
- wind
- rain

In [96]:
```
X_W = forest_fires_dummy[['temp', 'RH', 'wind', 'rain']].values
Y_W = forest_fires_dummy['area'].values
X_train_W, X_test_W, Y_train_W, Y_test_W = train_test_split( X_W, Y_W, test_si
ze=0.30, random_state=0)


scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train_W)
# Apply transform to both the training set and the test set.
X_train_W = scaler.transform(X_train_W)
X_test_W = scaler.transform(X_test_W)
# Pre-compute post-processed Ys
Y_train_W_post = np.expm1(Y_train_W)
Y_test_W_post = np.expm1(Y_test_W)
tryPca(X_train_W)
```

	Principal Component	Cumulative percentage of variance
0	1	40.245384
1	2	66.228603
2	3	89.924798
3	4	100.000000



Also in this dataset the number of principal components is already small and will not be reduced

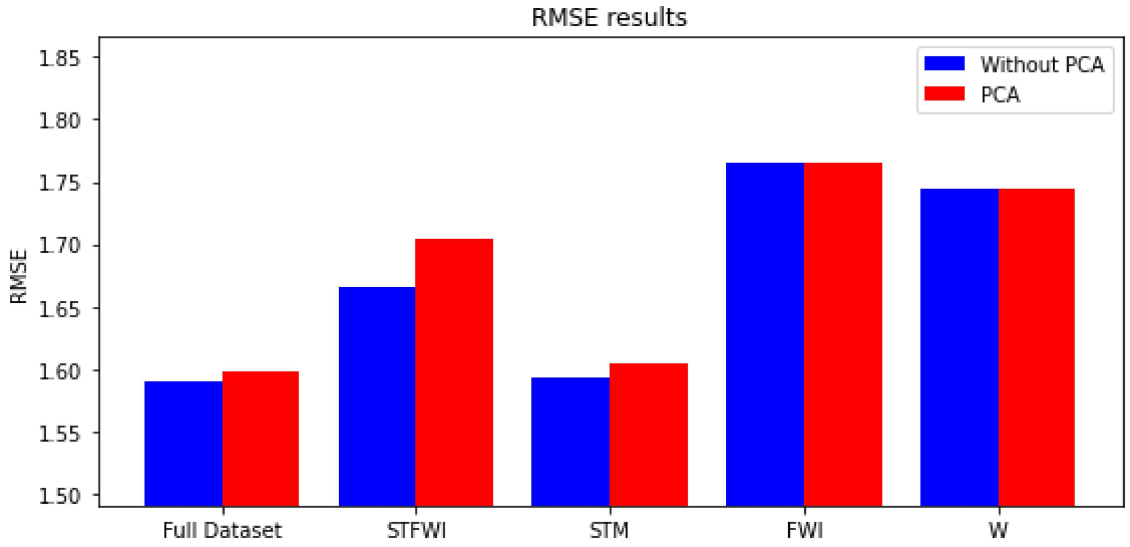
In [97]:
```
print("SVR without PCA for Weather selection:")
svr_RMSE_W, svr_MAE_W = makeSVR(X_train_W, Y_train_W, X_test_W, Y_test_W_post)
```

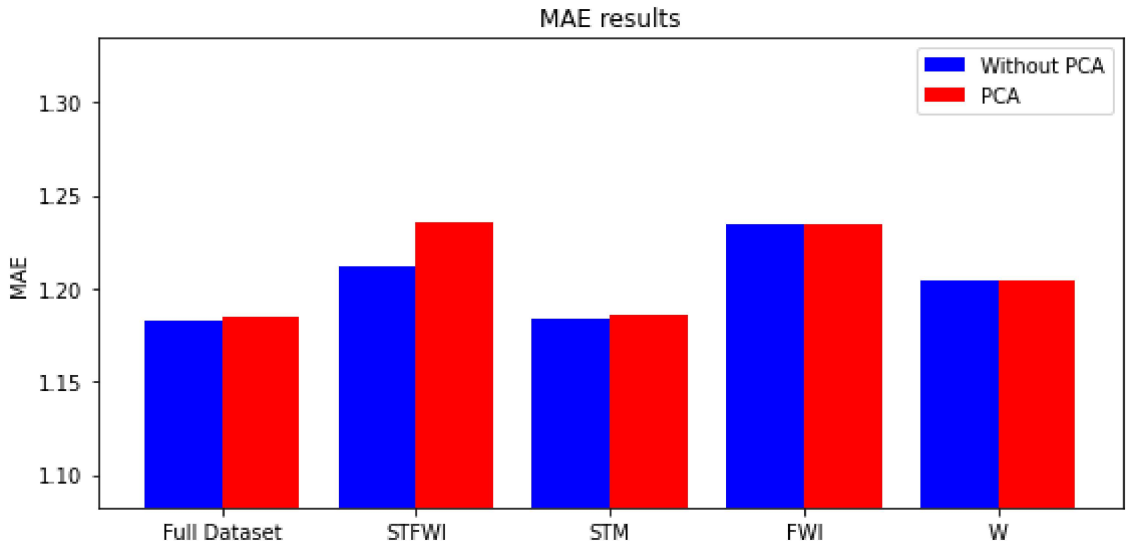
```
SVR without PCA for Weather selection:
Best parameters:  {'C': 1, 'epsilon': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}
RMSE: 1.7437467468701675
MAE: 1.2042265862449653
```

CONCLUSION

The aim of this project was to find the best model able to predict the burned area in case of forest fires and to evaluate which are the parameters that can increase the amplitude of fires. From the model selection part, the SVR was choosen as best algorithm for this topic. Now, let's see in this final graph the SVR results corresponding to the RMSE and MAE for each dataset used, in order to find the features that can increase the burnt area of fires.
Just for plot visualization in the FWI and W cases will be used the same results for both PCA/without PCA.

In [100]:
```
RMSEs = [svr_RMSE,svr_RMSE_STFWI,svr_RMSE_STM,svr_RMSE_FWI,svr_RMSE_W]
MAEs = [svr_MAE, svr_MAE_STFWI, svr_MAE_STM,svr_MAE_FWI,svr_MAE_W]
RMSEs_pca = [svr_pca_RMSE, svr_pca_RMSE_STFWI,svr_pca_RMSE_STM,svr_RMSE_FWI,svr_RMSE_W]
MAEs_pca = [svr_pca_MAE,svr_pca_MAE_STFWI, svr_pca_MAE_STM,svr_MAE_FWI,svr_MAE_W]
plotResult(RMSEs, RMSEs_pca, MAEs, MAEs_pca,['Full Dataset','STFWI','STM', 'FWI', 'W'])
```





As suggested before, PCA hasn't been useful because this premade feature selections are already well balanced. From the result obtained is visible that only the STM dataset is able to guarantee the same low errors as the original one. This because also the entire dataset isn't populated by tons of data without a meaning, but has already been perfectioned by the authors. Considering the STM model as the best obtained from the original dataset, is possible to determine that the factors which are more related to the presence of fires are the spatial-temporal and the four weather variables, in particular: X, Y, month, day, temp, RH, wind and rain.

In conclusion:

- Best model: **SVR**
- Features selected: **Spatial, temporal and weather variables**

REFERENCES

- "P. Cortez and A. Morais. A Data Mining Approach to Predict Forest Fires using Meteorological Data." Fires.pdf (http://www.dsi.uminho.pt/~pcortez/fires.pdf)