Audric Gautier, Eowyn Renoud Hallereau
3A MAM
2023-2024

Projet Analyse de données
Dataset 1 : Pluviométrie

Pré-traitement des données

Notre base de données était constituée à l'origine de quatre tableurs, un par année, ayant la structure suivante. Les lignes caractérisaient les instants où de la pluie avait été détecté sur l'un des 31 capteurs, et les colonnes la localisation de ces capteurs. Deux sous-colonnes étaient présentes, l'une indiquant la hauteur en mm d'eau détectée, et l'autre si le capteur était en bon état de fonctionnement.

Nous avons donc souhaité changer ces tableaux à doubles entrées en tableau de caractéristiques avec pour chaque instant de détection de pluie la hauteur des précipitations, la localisation du capteur, ainsi que l'état de fonctionnement de ce capteur. Nous avons également fait le choix de sommer la quantité d'eau par jour sur chaque capteur afin d'avoir une échelle temporelle plus appropriée aux traitements postérieurs.

```
import pandas as pd

def extract_data_pluie(link,head,dys): #fonction d'extraction du tableur
    boo_dys=1 #extraction des sous-colonnes pluies, autrement dysfonctionnement
    if(dys):
        boo_dys=0
    DF=pd.DataFrame(pd.read_excel(link,header=head,engine="xlrd",usecols=[0]+[i for i in range(1,62) if(i%2==boo_dys)]))
    DF["Date"]=DF["Date"].apply(lambda x:str(x).split(" ")[0])#conservation de la date
    DF=DF.groupby(DF["Date"]).aggregate("sum") #sommation selon la date
    return(DF)


link=["donnees_pluvio_2018\\annee2018_6mn.xls","donnees_pluvio_2019\\annee2019_6mn.xls","donnees_pluvio_2020\\6mn_2020.xls","donnees_pluvio_2021\\6mn_2021.xls"]
head=[5 , 5 , 5 , 6 ] #les headers ne sont pas tous a la ligne selon le fichier
DF_pluie_all=[]
DF_dys_all=[]
for i in range(4):
    DF_pluie_all.append(extract_data_pluie(link[i],head[i],False))
    DF_dys_all.append(extract_data_pluie(link[i],head[i],True))
DF_concat_pluie=pd.concat(DF_pluie_all)
DF_concat_dys=pd.concat(DF_dys_all)
DF_concat_dys["Unnamed: 62"]=0 #cette colonne ne contient aucun dysfonctionnement visible, et etait considere comme une colonne vide par Python.

DF_concat_dys.columns=DF_concat_pluie.columns

DF_concat_pluie=pd.DataFrame(DF_concat_pluie.reset_index().melt('Date')) #transformation d'un tableau a doubles entrees par un tableau de caracteristiques
DF_concat_dys=pd.DataFrame(DF_concat_dys.reset_index().melt('Date'))
DF_concat_pluie=DF_concat_pluie.rename(columns={"variable":"Station", "value":"Hauteur"})
```

```
DF_concat_dys=DF_concat_dys.rename(columns={"variable":"Station","value":"Operationnel"})
DF_concat_dys["Operationnel"]=DF_concat_dys["Operationnel"].apply(lambda x:False if("*" in str(x)) else True) #concatenation de l'information apportee par les dysfonctionnements,
DF_concat_pluie["Operationnel"]=DF_concat_dys["Operationnel"]
DF_concat_pluie.to_csv("Data_Measure_All.csv",index=False)
```

Nous obtenons ainsi un tableau plus facilement utilisable, notamment pour conserver des données ne remplissant que certaines caractéristiques. Il existe cependant un biais dans les données : dès qu'un capteur présente un dysfonctionnement sur une mesure, il est indiqué comme défectueux pour la journée. En parcourant notre base de données, nous avons constaté que cela est souvent le cas, mais, dans la forme des données actuelles, certaines mesures sont qualifiées comme défectueuses alors qu'elles ne l'étaient pas.

Nous avons reçu des données supplémentaires avec un autre tableau contenant les coordonnées exactes des capteurs, ainsi que leurs altitudes. Nous avons donc rajouté ces données à la table nouvellement créée grâce à une jointure.

```
df["id"]=df["Station"].apply(lambda x:x.split(" ")[0])
station=pd.read_csv("liste_stations.csv",sep="\t",usecols=["nom","lon","lat","identifiant"])
station["identifiant"]=station["identifiant"].apply(lambda x:str(x) if(len(str(x))>=2) else "0"+str(x)) #formatage des donnees pour avoir une cle commune
station=station.drop("nom",axis=1)

df=df.join(station.set_index("identifiant"),on="id") #jointure des deux tables

df=df.drop("id",axis=1)
```

Dans ce format, nous pouvons maintenant plus aisément traiter nos données.

Évolution des précipitations durant les années

Dans un premier temps, il nous a semblé pertinent de constater l'évolution des précipitations sur la région lyonnaise entre 2018 et 2021, afin de détecter la présence d'une saisonnalité, ou d'une tendance générale sur cette évolution. Nous avons hésité entre utiliser un diagramme en barres et une courbe, permettant tous deux de montrer cette évolution, mais nous avons préféré le choix de la courbe, car elle avait l'avantage d'avoir des marqueurs temporels plus lisibles, bien qu'elles puisse introduire des biais en faisant croire qu'une évolution a été constante entre deux dates précises.

Pour réaliser ce graphique, nous avons utilisé le script suivant :

```
import matplotlib.pyplot as plt

def gen_evol_graphe_curve(df,col): #trace une courbe des precipitations
    df["Month"]=df["Date"].apply(lambda x:str(x).split("-")[1])
    df["Year"]=df["Date"].apply(lambda x:str(x).split("-")[0])
    df=df.drop(df[df["Year"]=="2022"].index)
    df=df.drop(["Month","Year"],axis=1)
    df["Date"]=df["Date"].apply(lambda x:str(x).split("-")[0]+"-"+str(x).split("-")[1])
    df_n=df.groupby(df["Date"]).aggregate("sum") #aggregation des mesures par mois
    df_n=df_n.reset_index()
    plt.plot(df_n["Date"],df_n["Hauteur"],color=col)
    plt.xticks(rotation=90)

def add_marks_season(): #ajoute des marqueurs temporels pour les saisons
    lst_year=["2018","2019","2020","2021"]
    lst_month_season=["12","03","06","09"]
    lst_color=["paleturquoise","limegreen","goldenrod","orange"]
```

```
    date_plot=[]
    for x in lst_year:
        for y in lst_month_season:
            date_plot.append(x+"-"+y)
    date_plot.append("2018-01") #ajout force du premier mois, afin d'afficher le marqueur temporel
    for i in range(len(date_plot)):
        plt.axvline(x=date_plot[i], color=lst_color[i%4],ls=":",alpha=0.5)

df_all=pd.DataFrame(pd.read_csv("Data_Measure_All.csv")) #toutes les mesures
df_sort=pd.DataFrame(pd.read_csv("Data_Measure_Trie.csv")) #toutes les mesures non dysfonctionnelles

gen_evol_graphe_curve(df_all,"red")
gen_evol_graphe_curve(df_sort,"darkblue")
add_marks_season()
plt.xlabel("Mois")
plt.ylabel("Hauteur (mm)")


plt.legend(["Avec Dysfonctionnements", "Sans Dysfonctionnements","Hiver","Printemps","Ete","Automne"])
plt.title("Hauteur des Precipitations dans la Region Lyonnaise")

plt.show()
```

Grâce à ce script, nous avons obtenus le graphe suivant :

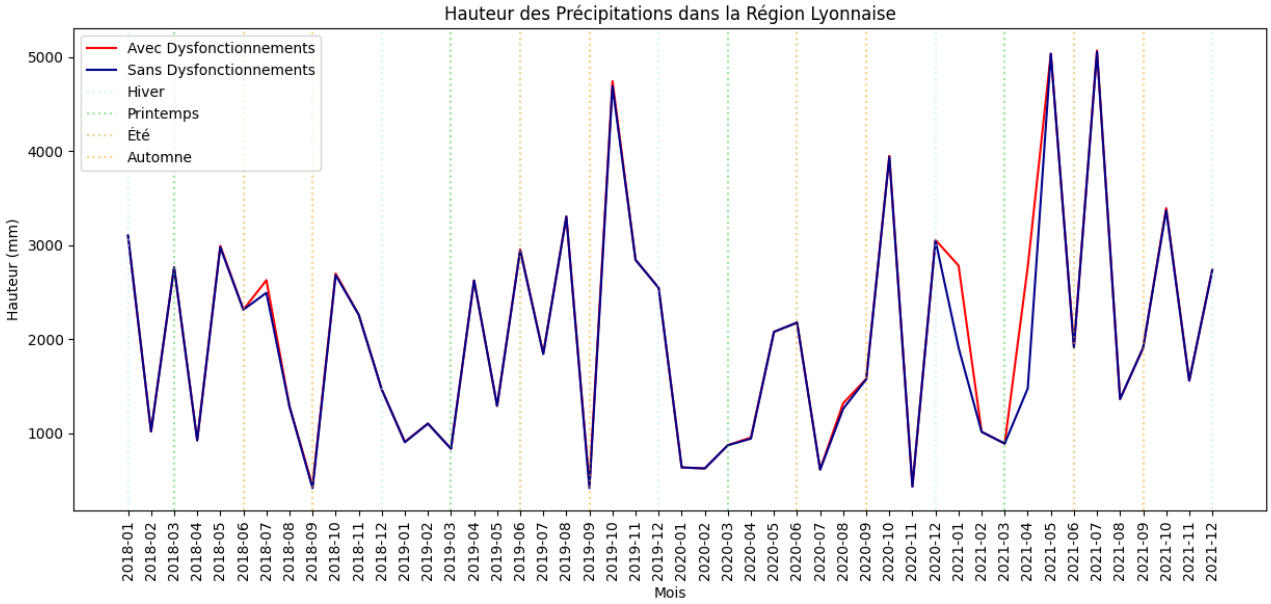

Figure 1: Courbe représentant l'évolution des précipitations (en mm) dans la région lyonnaise

Un premier constat est évident en regardant cette courbe. En effet, il ne semble pas d'y avoir de tendances à l'augmentation ou à la diminution des précipitations, mais plutôt des évènements extrêmes, comme le mois de mai 2021 ou le mois de janvier 2020. Ces évènements rendent d'ailleurs difficile la lecture de la saisonnalité de ces précipitations, bien que les hivers semblent secs, les printemps intermédiaires et les automnes pluvieux. Nous ne sommes même pas certains que nous puissions parler de saisonnalité à cette échelle de déplacement. Enfin, nous constatons que les dysfonctionnements ont généralement un impact faible en proportion des précipitations réellement détectés, mis à part dans la première moitié de

l'année 2021. Nous pouvons donc considérer les capteurs en place fiables, car même en cas de dysfonctionnement, il n'y a pas de grande influence sur les résultats généraux.

Détection des périodes où des évènements extrêmes ont eu lieu

En regardant le précédent graphique, il nous a semblé pertinent de vouloir comparer la quantité de pluie qui est tombée en proportion durant chaque saison, puis les comparer pour voir durant quelles périodes y-a-t-il eu des évènements extrêmes. Les diagrammes circulaires nous ont semblé pertinents pour une telle tâche, car ils sont facilement lisibles et permettent de comparer en "proportion" (et non pas en quantité, comme pour des diagrammes en barres), les précipitations totales durant les saisons.

```
from datetime import datetime

def give_season(x): #retourne la saison selon la date
    prop=datetime.strptime(x, '%Y-%m-%d').timetuple().tm_yday #position du jour dans l'
    annee
    if(80 <= prop < 172):
        return("Printemps")
    elif(172 <= prop < 264):
        return("Ete")
    elif(264 <= prop < 355):
        return("Automne")
    else:
        return("Hiver")

df=pd.read_csv("Data_Measure_Trie.csv")
df["id"]=df["Station"].apply(lambda x:x.split(" ")[0])
station=station.drop("nom",axis=1)
df=df.drop("Operationnel",axis=1)
df["Saison"]=df["Date"].apply(lambda x:give_season(x))
df["Aggreg"]=df["Date"].apply(lambda x:give_season(x)+"-"+x.split("-")[0])
n_df=df.groupby(df["Aggreg"]).agg({"Hauteur":np.sum})
n_df=n_df.reset_index()
n_df["Saison"]=n_df["Aggreg"].apply(lambda x:x.split("-")[0])
n_df["Annee"]=n_df["Aggreg"].apply(lambda x:x.split("-")[1])
n_df=n_df.drop("Aggreg",axis=1)

lst_season=["Hiver","Printemps","Ete","Automne"]

n_df=n_df.pivot(index="Annee", columns="Saison", values="Hauteur") #conversion en tableau
    a double entree

lst_color=[["lightcyan","paleturquoise","turquoise","lightseagreen"],["lightgreen","
    palegreen","limegreen","forestgreen"],["lemonchiffon","palegoldenrod","gold","
    goldenrod"],["papayawhip","moccasin","sandybrown","peru"]]

fig=plt.figure()

ax=[plt.subplot(int("22"+str(i+1))) for i in range(4)]

fig.suptitle("Proportion de pluie selon l'annee et la saison",fontsize=16)

for i in range(4):
    ax[i].pie(n_df[lst_season[i]],labels=n_df.index,colors=lst_color[i])
    ax[i].legend().remove()
    ax[i].set_title(lst_season[i])

plt.show()
```

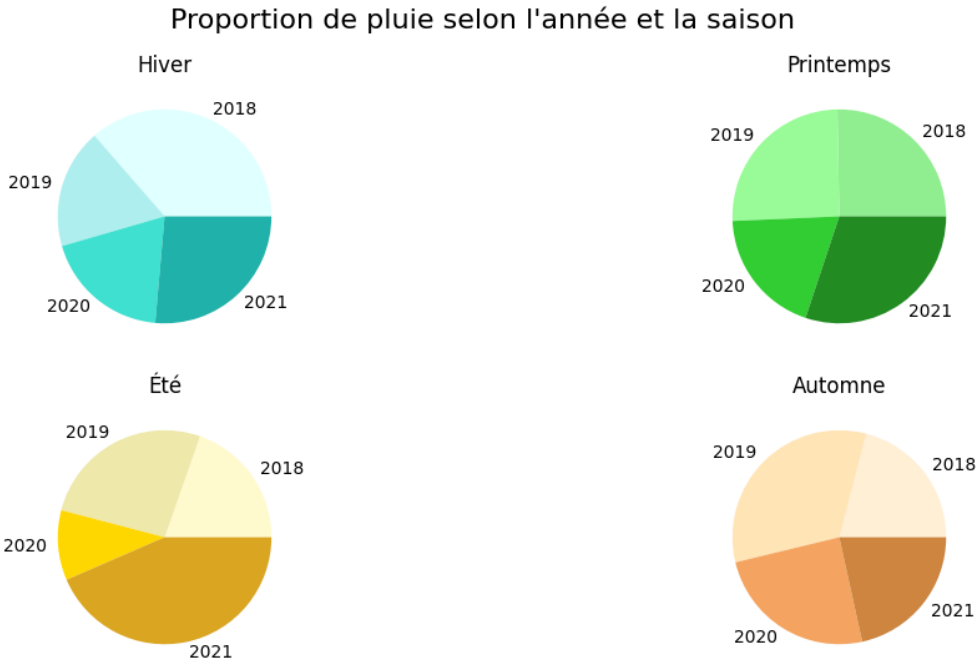

Figure 2: Diagrammes circulaires représentant la proportion des précipitations tombées dans la région lyonnaise entre 2018 et 2021 pendant chaque saison

Ce diagramme permet de comparer par saison la proportion de pluie qui est tombée dans la région par rapport aux saisons suivantes/précédentes :

- L'hiver 2018 a été particulièrement pluvieux, notamment car il y a eu de nouvelles précipitations en mars de cette année , contrairement aux années précédentes.
- La relative proportion de pluie identique entre tous les printemps est étonnant, au vu de la grande quantité de pluie tombée en mai 2021. Cependant, le début du printemps en 2021 était plus sec, ce qui rend cohérent cette hypothèse.
- C'est durant cette saison que les évènements sont les plus extrêmes, l'été 2020 et 2018 ont été très secs, et au contraire l'été 2021 a été très pluvieux.
- Enfin, les automnes de la région lyonnaise ont des précipitations à peu près uniforme, mis à part l'automne 2019. Cependant, il s'agit d'une des saisons où il pleut le plus en général.

Nous n'avons pas aperçu de causalité spécifique entre les saisons sèches et les saisons pluvieuses. Il faudrait sûrement plus d'années, et un autre genre de graphe, ou une analyse ACP adaptée pour déterminer s'il existe des facteurs causaux, ou du moins une corrélation.

Carte des précipitations

Nous avons souhaité voir si la localisation du capteur influençait la quantité de pluie qu'il recevait. Par exemple, est-ce que les capteurs qui reçoivent le plus de pluie sont au Nord, au Sud etc. Le diagramme qui nous a paru alors le pertinent est une carte avec des marqueurs indiquant à la position des capteurs la hauteur totale des précipitations depuis 2018.

```
import folium as fol

df=pd.read_csv("Data_Measure_Trie.csv")
print(sum(df["Hauteur"]))
df=df.drop("Date",axis=1)
df=df.drop("Operationnel",axis=1)
df=df.groupby(df["Station"]).aggregate("sum").reset_index() #regroupement des valeurs par stations

df["Station"]=df["Station"].apply(lambda x:x[3:]) #suppression de l'identifiant

print(df)
map=fol.Map(location=[45.5723403774766,4.79955516570804]) #affichage d'une carte vierge

for i in range(31):
    fol.CircleMarker(location=[df.iloc[i]["lat"],df.iloc[i]["lon"]], #ajout des marqueurs des capteurs
                radius=(df.iloc[i]["Hauteur"]-min(df["Hauteur"]))/25+5,
                popup=df.iloc[i]['Station'],
                color="blue",
                fill=True,
                fill_opacity=(df.iloc[i]["Hauteur"]-min(df["Hauteur"]))/(max(df["Hauteur"])-min(df["Hauteur"]))+0.1
    ).add_to(map)

legend_html = ''' #ajout d'une legende
 <div style="position: fixed;
             bottom: 100px; left: 50px; width: 200px; height: 150px;
             background-color: white; z-index:9999; font-size:14px;
             border:2px solid grey; border-radius:5px;">
       <b> Hauteur des Precipitations </b> <br>
       2750mm   <i style="background:blue;width:10px;height:10px;opacity:0.1;border-radius:50%;display:inline-block;border:2px solid blue;border-radius:50%;"></i><br>
       3000mm   <i style="background:blue;width:35px;height:35px;opacity:0.55;border-radius:50%;display:inline-block;border:2px solid blue;border-radius:50%;"></i><br>
       3250mm   <i style="background:blue;width:60px;height:60px;opacity:1;border-radius:50%;display:inline-block;border:2px solid blue; border-radius:50%;"></i>
 </div>
 '''

map.get_root().html.add_child(fol.Element(legend_html))

map.save("map.html") #sauvegarde du fichier html
```

Voici le graphe résultant (capture d'écran prise de la carte interactive)

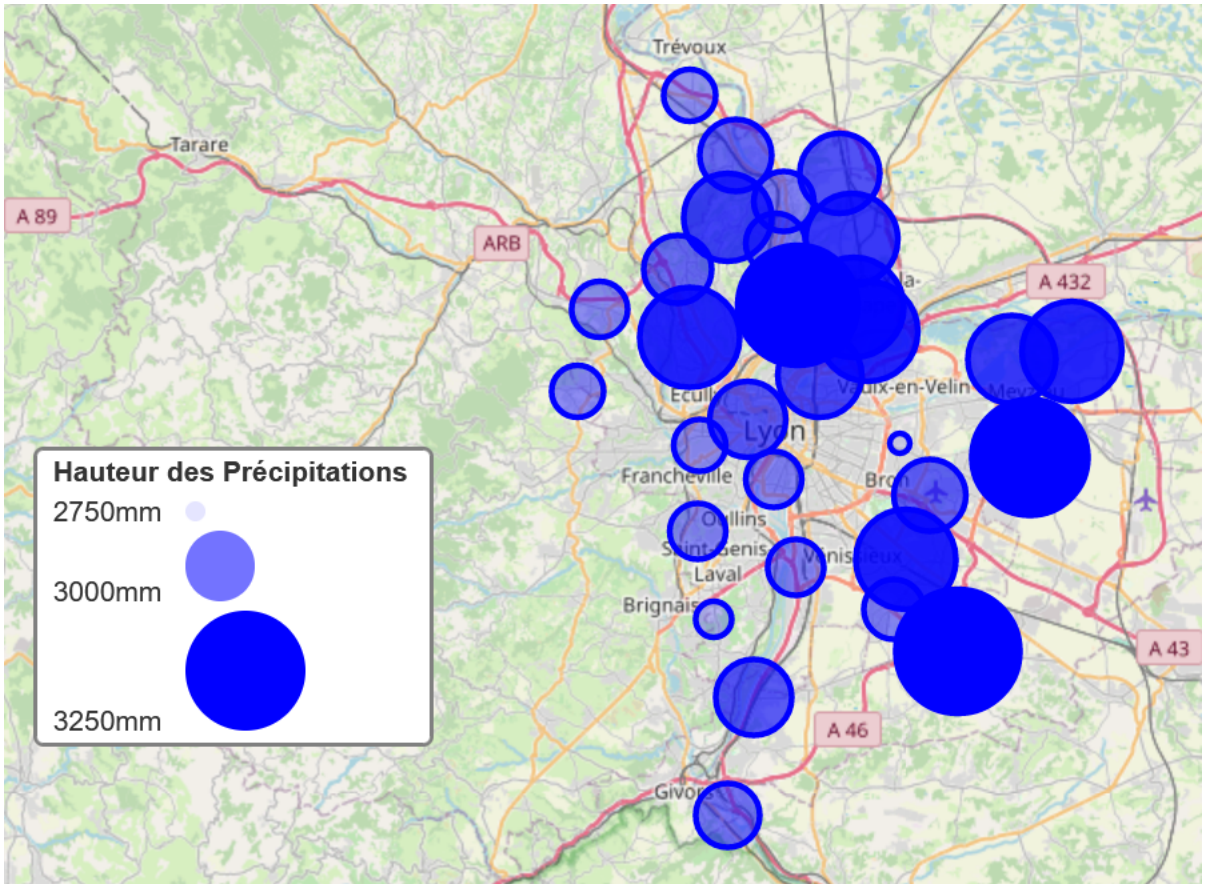

Figure 3: Carte représentant la quantité totale de précipitations détectées par les capteurs entre 2018 et 2021

Cette carte nous donne des informations précieuses sur la quantité totale de pluie tombée selon la localisation du capteur. En effet, la majorité des précipitations détectées se trouvent vers le nord-ouest de Lyon, ainsi que dans l'est. Cela nous donne une information également sur l'influence de l'altitude sur les pluies. En effet, le nord-ouest de Lyon est à une altitude moyenne plus élevée que l'est, donc nous pouvons penser que l'altitude n'a pas une influence importante. Nous constatons également que les capteurs du sud-ouest ont reçus moins de précipitations en moyenne (à $250,500$mm près). Cependant, en proportion, il n'y a pas de grande influences (cela représente une différence de $1/13 - 2/13$ par rapport au capteur qui en a détecté le plus). La localisation doit tout de même être un facteur explicatif de la variation des résultats, bien que nous ne sommes pas en mesure de l'expliquer avec des hypothèses crédibles.

Analyse en composantes principales

Première ACP

Nous souhaitons faire une analyse en composantes principales avec comme individus les hauteurs de précipitation.

Nous choisissons de ne pas utiliser comme caractéristique la date entière mais uniquement le mois, tout en gardant bien toutes les lignes. En effet, la précipitation selon les jours n'est pas suffisamment régulière, et nous ne travaillons que sur quatres années. Une échelle temporelle par mois est donc la plus adaptée pour notre ACP, donc nous transmettons uniquement cette caractéristique de la date comme variable.

```
data = data.set_index("Hauteur")
data["Annee"] = data["Date"].apply(lambda x : x.split("-")[0])
data["Mois"] = data["Date"].apply(lambda x : x.split("-")[1])
data["Mois"] = data["Mois"].apply(lambda x : int(x))
data = data.drop(["Date","Annee"], axis="columns")
n = data.shape[0]
p = data.shape[1]
```

Nous avons donc maintenant comme caractéristiques la longitude de la station météorologique, sa latitude, et le mois de la précipitation.

Pour notre ACP, nous centrons et réduisons nos données, puis nous choisissons le *svd_solver='full'* qui calcule l'ACP sur toutes les composantes.

```
acp = PCA(svd_solver='full')
sc = StandardScaler(with_mean=True, with_std=True)
data2 = sc.fit_transform(data)
coord=acp.fit_transform(data2)
```

On peut vérifier que le modèle trouve bien trois composantes. Puis on regarde le pourcentage de variables expliqué par nos composantes : la première composante explique 39% des données, la deuxième 33% des données et la troisième 28% des données.

```
print(acp.n_components_)
print(acp.explained_variance_ratio_)
plt.plot(np.arange(1,p+1),np.cumsum(acp.explained_variance_ratio_))
plt.title("Variance expliques par rapport au composante")
plt.ylabel("Pourcentage de variance explique cumule")
plt.xlabel("Composantes")
plt.show()
```

On peut également le voir sur le graphe suivant.

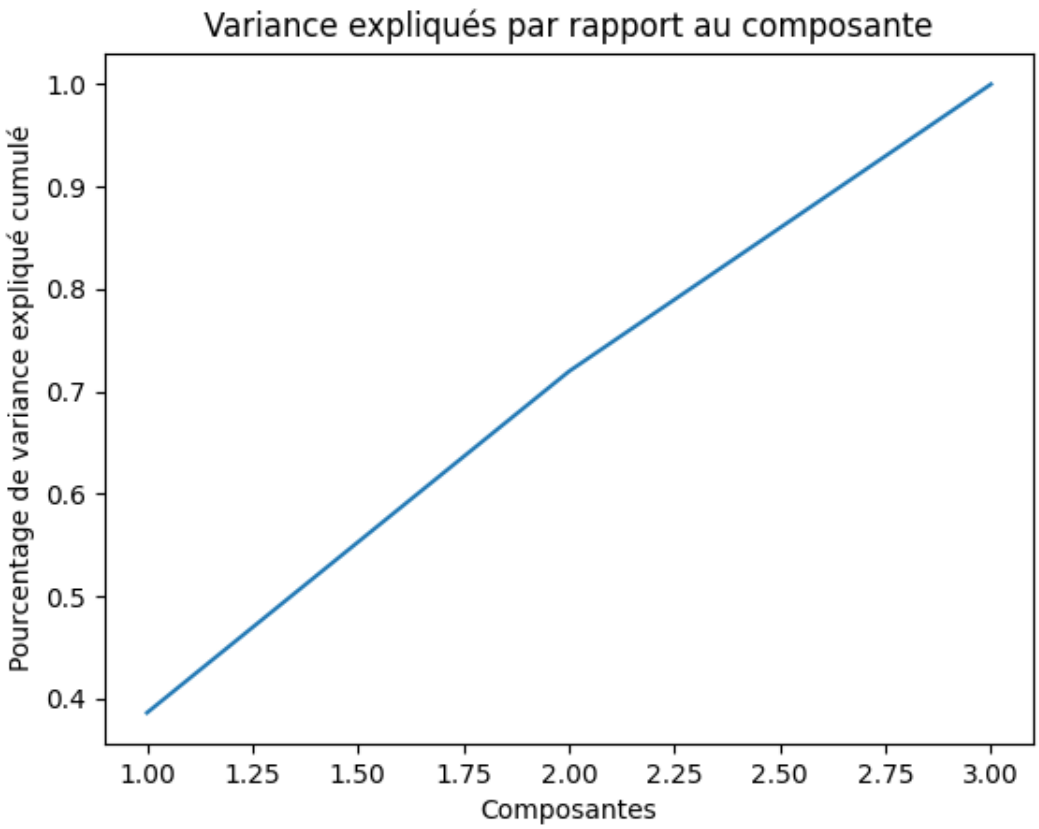

Figure 4: Pourcentage de variables expliqué par les composantes

On voit déjà qu'il y a n'y a pas de composantes réellement plus importantes que les autres, nous allons donc toujours conserver ces trois composantes principales.

Nous enregistrons dans un fichier csv les valeurs des méta variables pour chaque composante, afin de pouvoir les parcourir plus aisément. Nous en aurons besoin pour l'interprétation de l'ACP.

```
cf = pd.DataFrame(coord,index=data.index,columns=np.arange(1,p+1))
cf.to_csv("cf.csv")
```

Nous cherchons à caractériser les composantes à l'aide des individus, pour cela nous calculons ce qu'on appelle le cosinus carré, ce qui correspond à la corrélation au carré. Cette caractérisation permet de connaître la qualité de représentation de la variable selon les trois composantes. Etant donnée la quantité d'individus que nous avons, nous sauvegardons ce DataFrame en fichier csv, afin de pouvoir le lire séparément.

```
di = np.sum(data2**2,axis=1)
cos2 = coord**2
for j in range(p):
  cos2[:,j] = cos2[:,j]/di
qual_rep_var_selon_comp=pd.DataFrame({'id':data.index,'COS2_1':cos2[:,0],
  'COS2_2':cos2[:,1], 'COS2_3':cos2[:,2]})
qual_rep_var_selon_comp.to_csv("qual_rep_var_selon_comp.csv")
```

Après avoir passé en revu toutes les lignes obtenues nous remarquons différentes choses. Si nous nous concentrons sur une seule station météorologique, on remarque que la première composante représente bien le mois de juillet, et de manière générale la période d'avril à septembre et donc la période du printemps et de l'été. La deuxième composante représente quant à elle bien le mois de janvier, et de manière générale la

période d'octobre à mars, c'est- à-dire l'automne et l'hiver. En ce qui concerne la troisième composante, elle représente bien le mois de juillet, comme la première composante.

Cependant la troisième composante est plus intéressante à regarder selon chaque station. En effet, le niveau de représentation de cette composante varient surtout en fonction de la station et non du mois. Pour certaines stations cette composante est plus représentative que les deux premières.

Nous affichons maintenant la corrélation de chaque variable avec chaque composante. On peut voir que la longitude est corrélée positivement à la première composante et négativement à la troisième. La lattitude est corrélé négativement à la première et troisième composante. Le mois est quant à lui corrélé négativement à la deuxième composante.

```
eigval = (n-1)/n*acp.explained_variance_
sqrt_eigval = np.sqrt(eigval)
corvar = np.zeros((p,p))
for k in range(p):
  corvar[:,k] = acp.components_[k,:] * sqrt_eigval[k]
fig, axes = plt.subplots(figsize=(8,8))
axes.set_xlim(-1,1)
axes.set_ylim(-1,1)
for j in range(p):
  plt.annotate(data.columns[j],(corvar[j,0],corvar[j,1]))
plt.plot([-1,1],[0,0],color='silver',linestyle='-',linewidth=1)
plt.plot([0,0],[-1,1],color='silver',linestyle='-',linewidth=1)
cercle = plt.Circle((0,0),1,color='blue',fill=False)
axes.add_artist(cercle)
plt.show()
```

Nous traçons ensuite les cercles de corrélations selon les composantes deux par deux.

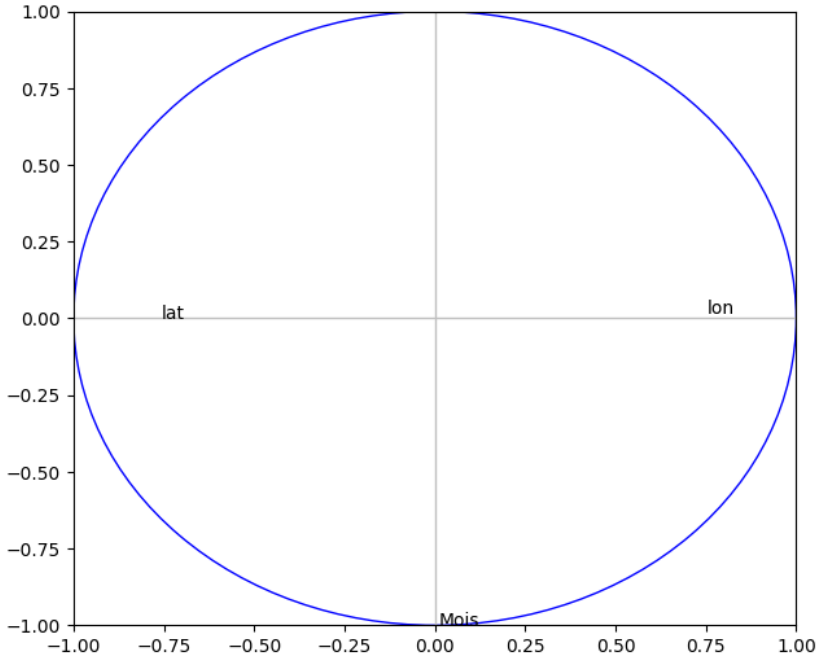

Figure 5: Cercle de corrélations selon première et deuxième composante

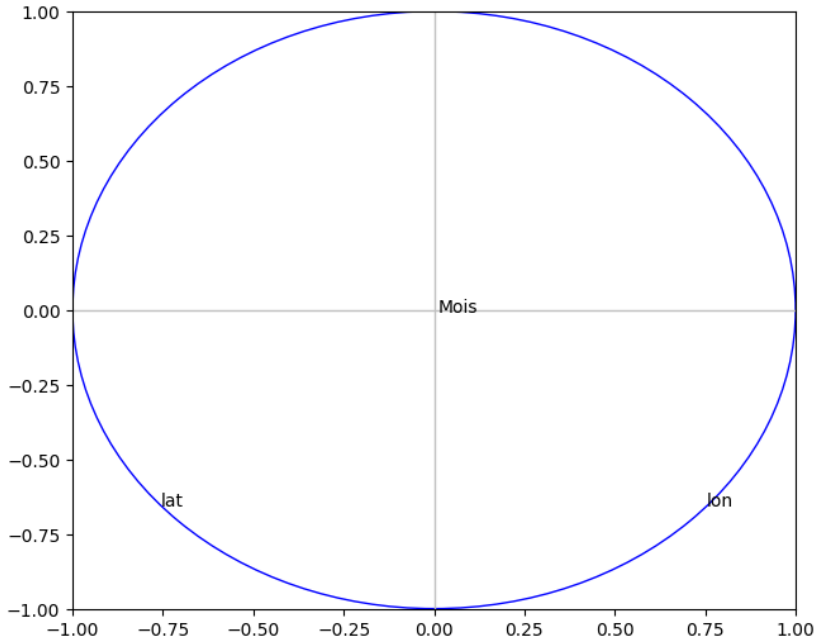

Figure 6: Cercle de corrélations selon première et troisième composante

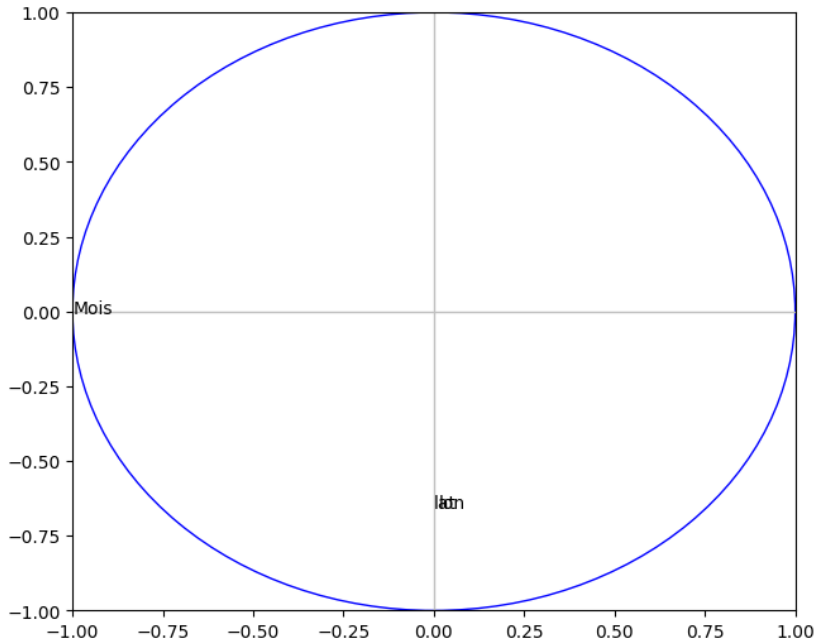

Figure 7: Cercle de corrélations selon deuxième et troisième composante

La longitude et la latitude étant corrélées négativement aux hauteurs de précipitations, selon la

troisième composante, on peut en déduire qu'il pleut le plus dans les stations dont la troisième composante, est négative, et qu'il pleut moins quand la méta variable est positive. On regarde dans notre fichier cf enregistré plus tôt, et on peut voir que les station dont la troisième coposante est positive se situent au Sud - Ouest, ainsi il pleut le moins dans le Sud - Ouest de Lyon.

La deuxième composante nous indique la corrélation entre les mois de l'année et la hauteur des précipitations. Comme les mois sont corrélées négativement, il faut regarder les mois dont la deuxième composante est négative. Nous pouvons en déduire qu'il pleut le plus en été et en automne.

La première composante est corrélée négativement à la latitude, donc il pleut plus dans les stations dont les valeurs de la première composante est négative, cela correspond aux latitudes du Nord de Lyon. Elle est cependant corrélée positivement à la longitude, donc il pleut plus dans les stations dont la méta variable de la première composante est positive, cela correspond aux longitudes de l'Est de Lyon. On retrouve bien le même résultat qu'avec la troisième composante.

Deuxième ACP

Nous souhaitons maintenant utiliser une caractéristique que nous n'avons pas utilisée précédemment, c'est-à -dire l'altitude des stations météorologiques (zsol).

Nous effectuons le même prétraitement de données qu'expliquée pour la première ACP.

```
data = pd.DataFrame(pd.read_csv("Data_Measure_Trie.csv"))
data=data.drop("Operationnel",axis="columns")
station = pd.DataFrame(pd.read_csv("liste_stations.csv", sep = "\t",
  usecols = ["identifiant","lon","lat","zsol"]))
station["identifiant"] = station["identifiant"].apply(lambda x: str(x) if len(str(x)) >=2
    else "0"+str(x))
station = station.rename(columns={"identifiant":"Station"})
data["Station"]=data["Station"].apply(lambda x : x.split(" ")[0])
data=data.merge(station, on = "Station")
data = data.set_index("Hauteur")
data["Annee"] = data["Date"].apply(lambda x : x.split("-")[0])
data["Mois"] = data["Date"].apply(lambda x : x.split("-")[1])
data["Mois"] = data["Mois"].apply(lambda x : int(x))
data = data.drop(["Date","Annee","Station"], axis="columns")
n = data.shape[0]
p = data.shape[1]
```

On effectue l'ACP :

```
acp = PCA(svd_solver='full')
sc = StandardScaler(with_mean=True, with_std=True)
data2 = sc.fit_transform(data)
coord=acp.fit_transform(data2)
```

La première composante explique 39% des données, la deuxième 25%, la troisième 22% et la dernière 14%. On voit déjà que la dernière composante explique moins les données.

```
print(acp.explained_variance_ratio_)
plt.plot(np.arange(1,p+1),np.cumsum(acp.explained_variance_ratio_))
plt.title("Variance expliques par rapport au composante")
plt.ylabel("Pourcentage de variance explique cumule")
plt.xlabel("Composantes")
plt.show()
```

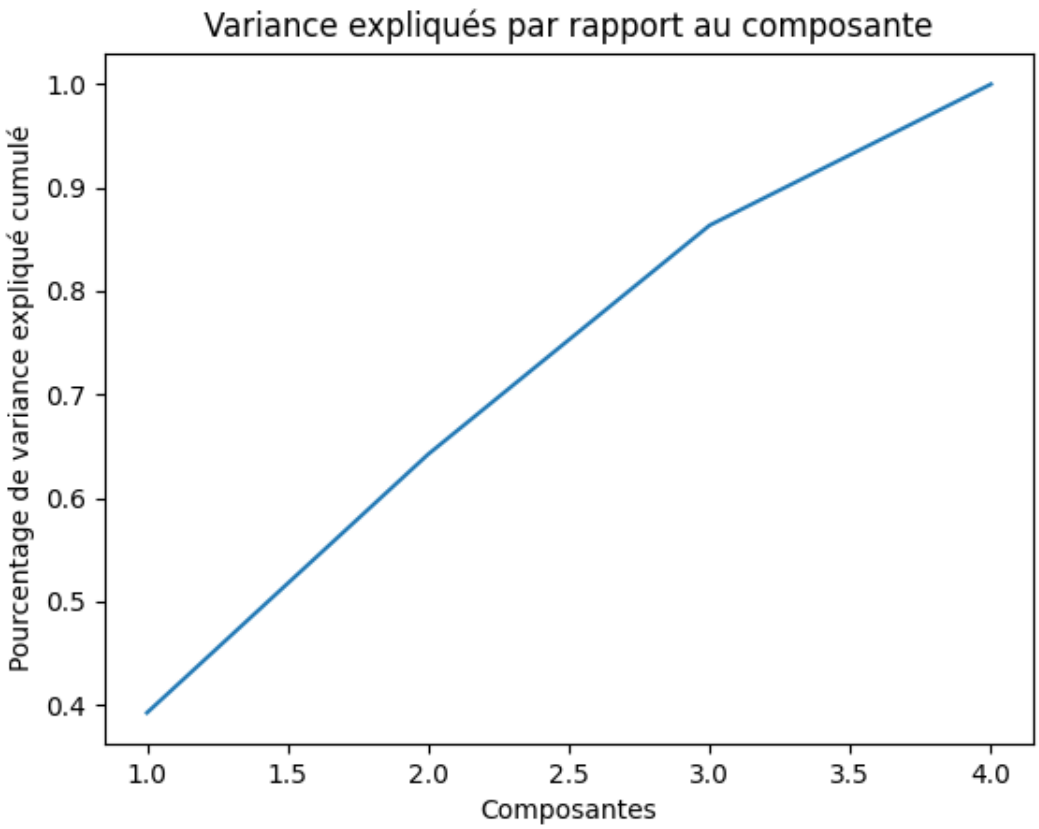

Figure 8: Pourcentage de variables expliqué par les composantes

Regardons maintenant les cercles de corrélations des différentes composantes.

```
eigval = (n-1)/n*acp.explained_variance_
sqrt_eigval = np.sqrt(eigval)
corvar = np.zeros((p,p))
for k in range(p):
  corvar[:,k] = acp.components_[k,:] * sqrt_eigval[k]

fig, axes = plt.subplots(figsize=(8,8))
axes.set_xlim(-1,1)
axes.set_ylim(-1,1)
for j in range(p):
  plt.annotate(data.columns[j],(corvar[j,0],corvar[j,1]))
plt.plot([-1,1],[0,0],color='silver',linestyle='-',linewidth=1)
plt.plot([0,0],[-1,1],color='silver',linestyle='-',linewidth=1)
cercle = plt.Circle((0,0),1,color='blue',fill=False)
axes.add_artist(cercle)
plt.show()
```

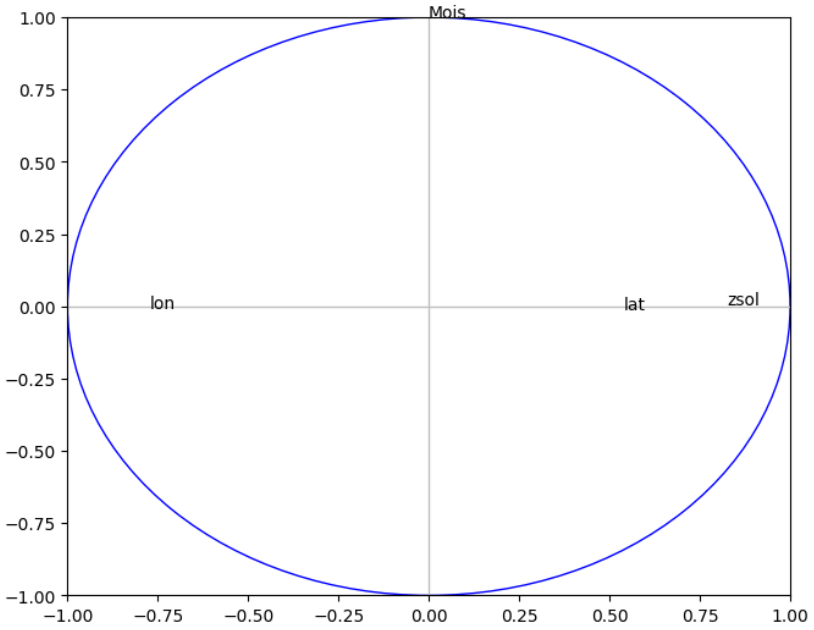

Figure 9: Cercle de corrélations selon première et deuxième composante

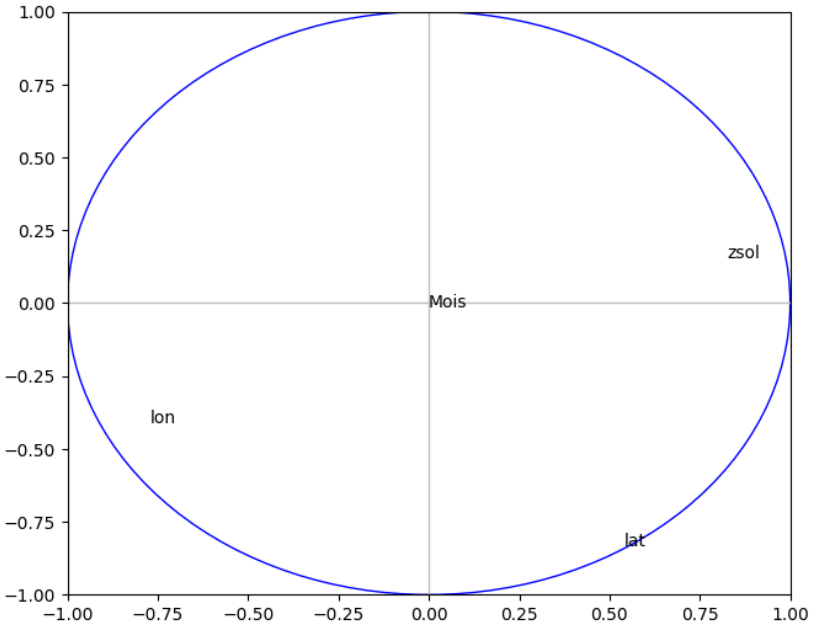

Figure 10: Cercle de corrélations selon première et troisième composante

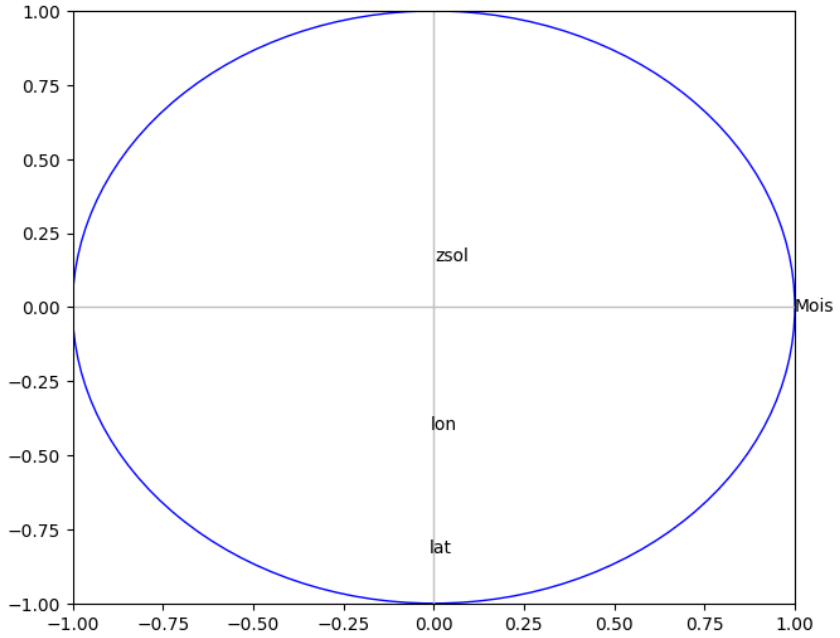

Figure 11: Cercle de corrélations selon deuxième et troisième composante

.

Si on regarde le premier cercle (9), la première chose qui peut nous étonner est que les trois composantes préalablement étudiées, sont maintenant corrélées de manière opposée. Cependant les valeurs des méta variables sont aussi opposées à celles que nous avions précédemment, donc le résultat ne change pas.

Nous pouvons également remarquer que l'altitude des stations est particulièrement représentée par la première et la troisième composante (9, 11), comme la latitude et la longitude.

On affiche également le tableau de corrélation, il représente la même chose que les cercles, mais nous évite de tracer les 6 cercles.

```
print(pd.DataFrame({'id':data.columns,'COR_1':corvar[:,0],'COR_2':corvar[:,1],
  'COR_3':corvar[:,2], 'COR_4':corvar[:,3]}))
```

id	COR_1	COR_2	COR_3	COR_4
zsol	0.825840	0.004130	0.157032	-0.541584
lon	-0.772469	-0.006978	-0.412036	-0.483188
lat	0.539230	-0.011756	-0.830814	0.137261
Mois	-0.002462	0.999909	-0.013292	0.000479

Nous pouvons lire sur le tableau des corrélations que zsol est principalement corrélé avec la première et la quatrième composante, nous avons déjà affiché la première composante (9), nous affichons donc maintenant la quatrième composante (12).

```
fig, axes = plt.subplots(figsize=(8,8))
axes.set_xlim(-1,1)
axes.set_ylim(-1,1)
for j in range(p):
   plt.annotate(data.columns[j],(corvar[j,3],corvar[j,1]))
plt.plot([-1,1],[0,0],color='silver',linestyle='-',linewidth=1)
plt.plot([0,0],[-1,1],color='silver',linestyle='-',linewidth=1)
```

```
cercle = plt.Circle((0,0),1,color='blue',fill=False)
axes.add_artist(cercle)
plt.show()
```

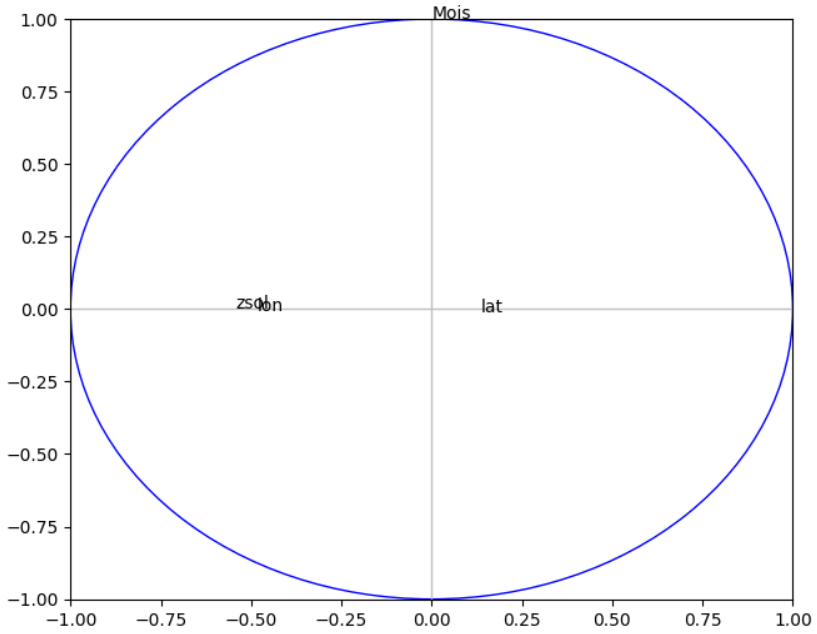

Figure 12: Cercle de corrélations selon deuxième et troisième composante

.

L'altitude étant corrélé positivement à la première composante, il faut regarder les valeurs positives des méta variables de la première composante. Cela correspond aux stations avec la plus haute altitude, donc il semble pleuvoir plus dans les stations plus hautes. L'altitude est également corrélée négativement à la quatrième composante (12), regardons donc les valeurs négatives des méta variables de cette composante. Les stations dont la quatrième composantes est négatives sont celles avec les plus hautes altitudes, donc cela est cohérent avec l'analyse que nous venons de faire sur la première composante.

Cependant, regardons la corrélation entre ces caractéristiques.

```
print("correlation", data.corr())
```

Nous pouvons voir que l'altitude est assez corrélée avec la longitude et la latitude. En effet, à Lyon, le relief se situe plutôt du côté Nord de la ville. Nous aurions donc pu déduire ces analyses avec notre première ACP, mais les résultats auraient étés moins évident.

	zsol	lon	lat	Mois
zsol	1.000000	-0.440980	0.240467	-0.000250
lon	-0.440980	1.000000	-0.140455	0.000170
lat	0.240467	-0.140455	1.000000	-0.001974
Mois	-0.000250	0.000170	-0.001974	1.000000