Modalità di utilizzo del codice

Query

Il codice è stato scritto utilizzando i Notebook Jupyter, integrati con librerie pandas, numpy, matplotlib e excel su linguaggio di programmazione python.

Le query sono state implementate usando Python con l'ausilio della libreria pandas.

Inizialmente è stato creato un dataframe **df_DatiSensori** direttamente dal file excel '**Dati_gruppo1.xlsx**'.

```
#Modifica la precisione nella visualizzazione delle cifre
pd.options.display.precision = 10

## Importa i dati dal file excel
df_DatiSensori = pd.read_excel('Dati_gruppo1.xlsx')

# Mostra tutte le colonne
# pd.set_option('display.max_rows', None)
```

Successivamente effettuiamo una pulizia del dataset appena importato. Innanzitutto andiamo a rinominare le colonne e creiamo un dataframe copia **df_Dati** per lavorarci.

```
#Rinomina le colonne con carattersi speciali e assegna il valore di soglia
df_DatiSensori.rename(columns = {'C6H6_ug/m3':'C6H6_ug_m3', 'H2S_ug/m3':'H2S_ug_m3', 'H2SJ_ug/m3':'H2SJ_ug_m3'}, inplace = True)
threshold = 24

# crea una copia del dataset per lavorarci
df_Dati = df_DatiSensori.copy()
```

Effettuiamo operazioni per il riconoscimento di eventuali valori 0 consecutivi che potrebbero significare fallimento dei sensori nell'invio dei dati attraverso una funzione definita **find_fail**.

```python
#Restituisce la lista con gli indici delle righe con almeno un valore di threshold di zeri consecutivi
def find_fail(colonna, df, threshold):
    index, dizionario = {} , {}
    listaTagli, out  = [], []
    df1 =df[[colonna]].copy()
    df1["Somma"] = df1.rolling(threshold).sum()
    for i in df1.index:
        if df1['Somma'][i] == 0:
            index[i] = df1['Somma'][i]
    indici = list(index.keys())
    for i in range(len(indici)-1):
        if (indici[i+1] - indici[i] > 1):
            listaTagli.append(i)
    listaTagli.append(0)
    listaTagli.sort()
    for i in range(len(listaTagli)-1):
        if i == 0:
            dizionario[i] = indici[listaTagli[i]:listaTagli[i+1]+1]
        else:
            dizionario[i] = indici[listaTagli[i]+1:listaTagli[i+1]+1]
    dizionario[len(dizionario.values())] = indici[listaTagli[-1]+1:]
    if (len(dizionario[0]) == 0):
      return out
    else:
      for key in dizionario.keys():
          maxIndice= dizionario[key][0]
          minIndice = maxIndice - threshold
          maxIndice = maxIndice + len(dizionario[key]) - 1
          out = out + list(np.arange(minIndice,maxIndice+1))
      return out
```

Questa funzione viene richiamata su dei dataframe temporanei creati per ogni coppia di sensore **valore/stato**, andando a resettare l'indice di ogni dataframe.

```python
# creo diversi dataframe quanti sono i sensori ed effettuo il controllo sui 24 zeri consecutivi ritenuti fallimento
# e infine effettuo il drop degli indici delle righe trovate
dfTRS = df_Dati[['TRS_ppb', 'TRS_stato']]
dfTRS = dfTRS.drop(find_fail('TRS_ppb', df_Dati, threshold)).reset_index()#TRS_ppb
dfVOC = df_Dati[['VOC_ppm', 'VOC_stato']]
dfVOC = dfVOC.drop(find_fail('VOC_ppm', df_Dati, threshold)).reset_index() #VOC
dfC6H6 = df_Dati[['C6H6_ug_m3', 'C6H6_stato']]
dfC6H6 = dfC6H6.drop(find_fail('C6H6_ug_m3', df_Dati, threshold)).reset_index().reset_index() #C6H6_ug/m3
dfH2S = df_Dati[['H2S_ug_m3', 'H2S_stato']]
dfH2S = dfH2S.drop(find_fail('H2S_ug_m3', df_Dati, threshold)).reset_index() #H2S_ug/m3
dfH2SJ = df_Dati[['H2SJ_ug_m3', 'H2SJ_stato']]
dfH2SJ = dfH2SJ.drop(find_fail('H2SJ_ug_m3', df_Dati, threshold)).reset_index() #H2SJ_ug/m3
dfPIDVOC = df_Dati[['PIDVOC_ppb', 'PIDVOC_stato']]
dfPIDVOC = dfPIDVOC.drop(find_fail('PIDVOC_ppb', df_Dati, threshold)).reset_index() #PIDVOC_ppb
```

Successivamente viene effettuato il concatenamento dei dataframe temporanei attraverso la funzione **pd.concat()**, scegliendo le colonne dei **valori** e dello **stato** dei sensori (così da non visualizzare la colonna **indice**). Effettuiamo una pulizia degli errori riportati con '**ND**' nel file excel.

```python
# elimina i valori ND
df_DatiPuliti = df_DatiPuliti[~df_Dati.TRS_stato.str.match('ND')]
df_DatiPuliti
```

Infine andiamo a effettuare la pulizia dei valori NaN che si sono creati.

```python
# elimina i valori NaN
df_DatiPuliti = df_Dati.dropna().reset_index(drop=True)
df_DatiPuliti
```

Le 100 registrazioni con il maggior livello di benzene

Nella prima query abbiamo creato un dataframe ordinato in modo discendete per i valori di benzene, andando ad eliminare qualsiasi duplicato riguardo alla data.

Il dataframe finale conterrà la postazione, la data e il valore del benzene, mostrando i primi cento.

```
# query effettuata con un unico comando

# creazione di un dataFrame ordinato in modo discendente per i valori di benzene
df_C6H6 = df_DatiPuliti.sort_values(by='C6H6_ug_m3', ascending =False)[['postazione', 'Data', 'C6H6_ug/m3']].head(100)
df_C6H6
```

Le 100 registrazioni con il maggior livello di acido solfidrico per i sensori H2S e H2SJ

Nella seconda query creiamo due dataframe, **df_H2S** e **df_H2SJ**, per i due sensori ed effettuiamo le stesse operazioni effettuate per la prima query.

```
# sensore H2S

# creazione di un dataFrame ordinato in modo discendente per i valori di acido solfidrico del sensore H2S
df_H2S = df_DatiPuliti.sort_values(by='H2S_ug_m3', ascending =False)[['postazione', 'Data', 'H2S_ug_m3']].head(100)
df_H2S
```

```
# sensore H2SJ

# creazione di un dataFrame ordinato in modo discendente per i valori di acido solfidrico del sensore H2SJ
df_H2SJ = df_DatiPuliti.sort_values(by='H2SJ_ug_m3', ascending =False)[['postazione', 'Data', 'H2SJ_ug_m3']].head(100)
df_H2SJ
```

Le 100 registrazioni con i più bassi livelli di VOC per i sensori VOC e PIDVOC

Nella terza query creiamo due dataframe, **df_VOC** e **df_PIDVOC**, per i due sensori ed effettuiamo le stesse operazioni effettuate per la prima query, con la differenza che in questo caso andremo ad ordinare i valori in maniera ascendente, cosi da prendere i 100 valori più bassi per i sensori.

```
# sensore VOC

# creazione di un dataFrame ordinato in modo discendente per i valori del sensore VOC
df_VOC = df_DatiPuliti.sort_values(by='VOC_ppm', ascending =True)[['postazione', 'Data', 'VOC_ppm']].head(100)
df_VOC
```

```
# sensore PIDVOC

# creazione di un dataFrame ordinato in modo discendente per i valori del sensore PIDVOC
df_PIDVOC = df_DatiPuliti.sort_values(by='PIDVOC_ppb', ascending =True)[['postazione', 'Data', 'PIDVOC_ppb']].head(100)
df_PIDVOC
```

Le 50 ore con il più alto/basso livello medio di benzene

Per questa query è stata creata una funzione che permette di eliminare i valori nulli, calcolare la media oraria e restituire un dataframe ordinato sull'ora e che contiene solo le colonne postazione, data e il composto interessato.

```
# funzione che permette di eliminare i valori nulli, calcola la media oraria e restituisce un dataframe ordinato sull'ora
def media_oraria(df, compound):
    df1 = df[['postazione', 'Data', compound]].copy()
    df1['Data'] = pd.to_datetime(df1['Data'].dt.strftime('%Y/%m/%d %H'))
    indexName = df1[(df1[compound]==0)].index
    df1.drop(indexName, inplace = True)
    return df1.groupby('Data').mean().dropna()
```

Dopodiché si è pensato di creare un dataframe richiamando la funzione **media_oraria**, andando ad aggiungere come parametri della funzione il dataframe **df_DatiPuliti** e il composto interessato, ovvero il benzene.

```
# Creazione di un dataframe ordinato sulla media oraria
df_C6H6avg = media_oraria(df_DatiPuliti, 'C6H6_ug_m3')
```

Infine è stata calcolata la più alta e la più bassa media oraria di benzene, andando poi a mostrare solo i primi 50 valori, modificando solo il parametro **ascending** della funzione **sort_values**.

```
# Stampa le 50 ore con i livelli di benzene più alti
df_C6H6avg.sort_values('C6H6_ug_m3', ascending = False).head(50)
```

```
# Stampa le 50 ore con i livelli di benzene più bassi
df_C6H6avg.sort_values('C6H6_ug_m3', ascending = True).head(50)
```

Le 50 ore con il più alto/basso livello medio di acido solfidrico secondo i sensori H2S e H2SJ

Sono state effettuate le stesse operazioni della query precedente, duplicate per entrambe i sensori **H2S** e **H2SJ**.

```
#Elimina i valori nulli e calcola la media oraria sul dataframe creato per il sensore H2S
df_H2S, df_H2Savg = media_oraria(df_DatiPuliti, 'H2S_ug_m3')
```

```
# Stampa le 50 ore con i livelli di acido solfidrico del sensore H2S più alti
df_H2SMax= df_H2Savg.sort_values('H2S_ug_m3', ascending = False).head(50)
df_H2SMax
```

```
# Stampa le 50 ore con i livelli di acido solfidrico del sensore H2S più basso
df_H2SMin= df_H2Savg.sort_values('H2S_ug_m3', ascending = True).head(50)
df_H2SMin
```

```
#Elimina i valori nulli e calcola la media oraria sul dataframe creato per il sensore H2SJ
df_H2SJ, df_H2SJavg = media_oraria(df_DatiPuliti, 'H2SJ_ug_m3')
```

```
# Stampa le 50 ore con i livelli di acido solfidrico del sensore H2SJ più alti
df_H2SJMax= df_H2SJavg.sort_values('H2SJ_ug_m3', ascending = False).head(50)
df_H2SJMax
```

```
# Stampa le 50 ore con i livelli di acido solfidrico del sensore H2SJ più bassi
df_H2SJMin= df_H2SJavg.sort_values('H2SJ_ug_m3', ascending = True).head(50)
df_H2SJMin
```

Le 50 ore con il più alto/basso livello medio di VOC secondo i sensori VOC e PIDVOC

Sono state effettuate le stesse operazioni della query precedente, duplicate per entrambe i sensori **VOC** e **PIDVOC**.

```
#Elimina i valori nulli e calcola la media oraria sul dataframe creato per il sensore VOC
df_VOC, df_VOCavg = media_oraria(df_DatiPuliti, 'VOC_ppm')
```

```
# Stampa le 50 ore con i livelli di VOC del sensore VOC più alti
df_VOCMax= df_VOCavg.sort_values('VOC_ppm', ascending = False).head(50)
df_VOCMax
```

```
# Stampa le 50 ore con i livelli di VOC del sensore VOC più bassi
df_VOCMin= df_VOCavg.sort_values('VOC_ppm', ascending = True).head(50)
df_VOCMin
```

```
#Elimina i valori nulli e calcola la media oraria sul dataframe creato per il sensore PIDVOC
df_PIDVOC, df_PIDVOCavg = media_oraria(df_DatiPuliti, 'PIDVOC_ppb')
```

```
# Stampa le 50 ore con i livelli di VOC del sensore PIDVOC più alti
df_PIDVOCMax= df_PIDVOCavg.sort_values('PIDVOC_ppb', ascending = False).head(50)
df_PIDVOCMax
```

```
# Stampa le 50 ore con i livelli di VOC del sensore PIDVOC più bassi
df_PIDVOCMin= df_PIDVOCavg.sort_values('PIDVOC_ppb', ascending = True).head(50)
df_PIDVOCMin
```

Le 3 giornate con il maggior numero di fallimenti nell'invio dei dati

Per questa query è stato creato un dataframe df_Fail_ND che ha come colonne la Data e i valori numerici dei sensori. Creiamo una nuova colonna settando il tipo data a 'giorno/mese/anno' ed effettuiamo il conteggio dei valori NaN sulla colonna TRS_ppb (il conteggio risulta identico su qualunque altra colonna poiché il fallimento di una stazione coincide con il fallimento di invio dei dati di

ogni sensore) effettuando il groupby sul 'Giorno', sommando e infine resettando gli indici, creando così una nuova colonna 'count'.

```
# Giornate con fallimenti ND (i fallimenti nell'invio dei dati sono uguali per ogni sensore)
df_Fail_ND = df_DatiSensori.copy()
df_Fail_ND['Giorno'] = pd.to_datetime(df_Fail_ND['Data'].dt.strftime('%Y/%m/%d'))
df_Fail_ND = df_Fail_ND.TRS_ppb.isnull().groupby(df_Fail_ND['Giorno']).sum().transform(int).reset_index(name='count')
df_Fail_ND
```

Infine è stato effettuato un ordinamento decrescente sulla colonna 'count' e presi in considerazione solo le prime 3 righe.

```
# 3 giorni con il maggior numero di fallimenti
df_Fail_ND.sort_values('count', ascending = False).head(3)
```

Le 3 giornate con il minor numero di fallimenti nell'invio dei dati

Per questa query sono state effettuate le stesse operazioni di quella precedente andando a modificare accuratamente i parametri interessati.

```
# 3 giorni con il minor numero di fallimenti
df_Fail_ND.sort_values('count', ascending = True).head(3)
```

Il numero medio di fallimenti nell'invio per sensore

Per questa query sono state create due liste, **inquinanti** e **stato_inquinanti**, da usare nella funzione successiva.

```
# crea due liste con tutti gli inquinanti e i relativi stati
inquinanti = ['TRS_ppb','VOC_ppm','C6H6_ug_m3','H2S_ug_m3','H2SJ_ug_m3','PIDVOC_ppb']
stato_Inquinanti = ['TRS_stato','VOC_stato','C6H6_stato', 'H2S_stato','H2SJ_stato', 'PIDVOC_stato']
```

È stato sviluppato un metodo che permette di contare la somma dei fallimenti per ogni sensore andando poi ad inserirli in un nuovo dataframe **df_NFails**. Innanzitutto creo una lista di fallimenti, **fails**, e poi inizializzo il nuovo dataframe **df_NFails** a 2 colonne di nome **Sensori** e **Fallimenti**. Vado ad effettuare un ciclo sulla lunghezza di una delle due liste (quale è indifferente, avendo entrambe la stessa lunghezza). Nel ciclo creo un dataframe temporaneo contenente solo le colonne **inquinante** e **stato_inquinante**. La variabile **errors** restituisce il numero di fallimenti dovuti alla presenza di 0 consecutivi (come fatto per la pulizia del dataset, abbiamo scelto un valore di 24 zeri consecutivi che equivalgono a 2 ore di invii di dati considerati falliti), mentre **num_ND** restituisce il numero di valori **ND** presenti. Infine questi valori vengono sommati e aggiunti alla lista **fails**. Fuori

dal ciclo for viene popolato il dataframe **df_NFails** con la colonna sensori uguagliata alla lista inquinanti e la colonna fallimenti con **fails**.

```
fails = []
df_NFails = pd.DataFrame({'Fallimenti':np.arange(6)})
df_NFails = pd.DataFrame({'Sensori':np.arange(6)})
for column in range(len(inquinanti)):
    df_Fail = df_Dati[[inquinanti[column], stato_Inquinanti[column]]].copy()
    errors = df_Fail[inquinanti[column]].ne(df_Fail
                                [inquinanti[column]].shift()).cumsum()[df_Fail
                                                                       [inquinanti[column]].eq(0.0)].value_counts().ge(24).sum(
    num_ND= df_Fail[stato_Inquinanti[column]].value_counts()["ND"]
    fails.append(num_ND + errors)

df_NFails['Sensori'] = inquinanti
df_NFails['Fallimenti'] = fails
df_NFails
```

Viene aggiunta una colonna '**Dati TOT**' che comprende la lunghezza dei dati per ogni sensore ed infine viene creata un'ultima colonna chiamata '**Media**' contente la media tra la colonna '**Fallimenti**' e '**Dati TOT**'.

```
df_NFails['Dati TOT'] = [len(dfTRS), len(dfVOC), len(dfC6H6), len(dfH2S), len(dfH2SJ), len(dfPIDVOC)]
df_NFails['Media'] = df_NFails['Fallimenti'].div(df_NFails['Dati TOT'])
df_NFails
```

Il sensore con il numero massimo di fallimenti

Per questa query è stato utilizzato lo stesso dataframe creato nella query precedente ed è stata fatta una chiamata **max()** al dataframe **df_NFails** sulla colonna '**Fallimenti**'.

```
# Prende il massimo dal dataframe
df_NFails['Fallimenti'].max()
```

Il sensore con il numero minimo di fallimenti

Per questa query è stato utilizzato lo stesso dataframe creato nella query precedente ed è stata fatta una chiamata **min()** al dataframe **df_NFails** sulla colonna '**Fallimenti**'

```
# Prende il minimo dal dataframe
df_NFails['Fallimenti'].min()
```

Correlazioni

Inizialmente è stata creata una funzione correlazione **corr_Sensori** che mi permette di ricevere in output due valori, ovvero la correlazione di **Pearson** e la correlazione di **Spearman**. Questa funzione prende in input il dataframe su cui effettuare le correlazioni, i due composti come stringa che si intendono mettere in correlazione e la postazione, anch'essa stringa, che è settata come **None** (dato che questa funzione è stata adattata per essere utilizzata per due tipi di correlazioni diverse). Inizialmente verifica che il valore di **postazione** è diverso da **None** così da poter creare un dataframe **df1** prendendo solo la colonna postazione interessata, altrimenti fa una copia dell'intero dataframe che è stato passato. Viene effettuata la regressione lineare e graficati i valori e calcoliamo i coefficienti di **Pearson** e di **Spearman**.

```python
# Funzione correlazione
def corr_Sensori(df, colonna1, colonna2, postazione = None):
    cc = -2
    cs = pd.DataFrame([{'a': 0},{'a': -2}])
    ck = pd.DataFrame([{'a': 0},{'a': -2}])
    if postazione != None:
        df1 = df[df['postazione'] == postazione]
    else:
        df1 = df.copy()
    df1.plot.scatter(x= colonna1, y=colonna2)
    if np.sum(df1[colonna1].to_list())!=0 and np.sum(df1[colonna2].to_list()) !=0:
        a,b = np.polyfit(df1[colonna1].to_list(), df1[colonna2].to_list(), 1) #Inferiamo y =ax + b
        x1 = min(df1[colonna1].to_list())
        x2 = max(df1[colonna1].to_list())
        plt.plot([x1,x2], [a*x1 +b, a*x2 +b], color = 'red')
        plt.show()
        cc = np.corrcoef(df1[colonna1], df1[colonna2])[1,0]
        cs = df1[[colonna1, colonna2]].corr(method = 'spearman')
        ck = df1[[colonna1, colonna2]].corr(method = 'kendall')
    return cc, cs.iloc[1,0], ck.iloc[1,0]
```

Correlazione tra H2S e H2SJ in una data stazione

Nella prima correlazione creiamo un dataframe df_AcidoSolf prendendo dal dataframe df_DatiPuliti le colonne di **postazione**, **Data**, **H2S_ug_m3** e **H2SJ_ug_m3**, effettuiamo l'eliminazione dei valori nulli attraverso la funzione **dropna()** e infine richiamiamo la funzione **corr_Sensori** passando le come variabili il dataframe **df_AcidoSolf**, **H2S_ug_m3**, **H2SJ_ug_m3** e la **postazione** (viene effettuata su tutte e 4). I valori ottenuti vengono associati alle variabili **pearson** e **spearman** e poi stampate a video.

```
# Crea un dataframe contenente entrambi i sensori, rinomina le colonne ed elimina i valori nulli
df_AcidoSolf = df_DatiPuliti[['postazione', 'Data', 'H2S_ug_m3', 'H2SJ_ug_m3']]
df_AcidoSolf = df_AcidoSolf.dropna()
df_AcidoSolf
```

```
# correlazione per la stazione ATM05_01479
pearson, spearman, kendall= corr_Sensori(df_AcidoSolf, 'H2S_ug_m3', 'H2SJ_ug_m3','ATM05_01479')
print('Correlazione tra H2S e H2SJ:\npearson: ', pearson,' spearman: ', spearman,' kendall: ', kendall)
```

Correlazione tra VOC e PIDVOC in una data stazione

Per questa correlazione sono state effettuate le stesse operazioni della correlazione precedente andando ad effettuare le opportune modifiche sui composti interessati, **VOC** e **PIDVOC**.

```
# Crea un dataframe contenente entrambi i sensori, rinomina le colonne ed elimina i valori nulli
df_SensoriVOC = df_DatiPuliti[['postazione', 'Data', 'VOC_ppm', 'PIDVOC_ppb']]
df_SensoriVOC = df_SensoriVOC.dropna()
df_SensoriVOC
```

```
# correlazione per la stazione ATM05_01479
pearson, spearman, kendall= corr_Sensori(df_SensoriVOC, 'VOC_ppm', 'PIDVOC_ppb', 'ATM05_01479' )
print('Correlazione tra VOC e PIDVOC:\npearson: ', pearson,' spearman: ', spearman,' kendall: ', kendall)
```

Correlazione tra TRS e H2S in una data stazione

Per questa correlazione sono state effettuate le stesse operazioni della correlazione precedente andando ad effettuare le opportune modifiche sui composti interessati, **TRS** e **H2S**.

```
df_SenTRS_H2S = df_DatiPuliti[['postazione', 'Data', 'TRS_ppb', 'H2S_ug_m3']]
df_SenTRS_H2S = df_SenTRS_H2S.dropna()
df_SenTRS_H2S.fillna(0)
```

```
# correlazione per la stazione ATM05_01479
pearson, spearman, kendall = corr_Sensori(df_SenTRS_H2S, 'TRS_ppb', 'H2S_ug_m3', 'ATM05_01479' )
print('Correlazione tra TRS e H2S:\npearson: ', pearson,' spearman: ', spearman,' kendall: ', kendall)
```

Correlazione tra TRS e H2SJ in una data stazione

Per questa correlazione sono state effettuate le stesse operazioni della correlazione precedente andando ad effettuare le opportune modifiche sui composti interessati, **TRS** e **H2SJ**.

```
df_SenTRS_H2SJ = df_DatiPuliti[['postazione', 'Data', 'TRS_ppb', 'H2SJ_ug_m3']]
df_SenTRS_H2SJ = df_SenTRS_H2SJ.dropna()
df_SenTRS_H2SJ
```

```
# correlazione per la stazione ATM05_01479
pearson, spearman, kendall= corr_Sensori(df_SenTRS_H2SJ, 'TRS_ppb', 'H2SJ_ug_m3' , 'ATM05_01479')
print('Correlazione tra TRS e H2SJ:\npearson: ', pearson,' spearman: ', spearman,' kendall: ', kendall)
```

Correlazione tra VOC e C6H6 in una data stazione

Per questa correlazione sono state effettuate le stesse operazioni della correlazione precedente andando ad effettuare le opportune modifiche sui composti interessati, **VOC** e **C6H6**.

```
df_SenVOC_C6H6 = df_DatiPuliti[['postazione', 'Data', 'VOC_ppm', 'C6H6_ug_m3']]
df_SenVOC_C6H6 = df_SenVOC_C6H6.dropna()
df_SenVOC_C6H6
```

```
# correlazione per la stazione ATM05_01479
pearson, spearman, kendall= corr_Sensori(df_SenVOC_C6H6, 'VOC_ppm', 'C6H6_ug_m3' , 'ATM05_01479')
print('Correlazione tra VOC e C6H6:\npearson: ', pearson,' spearman: ', spearman,' kendall: ', kendall)
```

Correlazione tra PIDVOC e C6H6 in una data stazione

Per questa correlazione sono state effettuate le stesse operazioni della correlazione precedente andando ad effettuare le opportune modifiche sui composti interessati, **PIDVOC** e **C6H6**.

```
df_SenPIDVOC_C6H6 = df_DatiPuliti[['postazione', 'Data', 'PIDVOC_ppb', 'C6H6_ug_m3']]
df_SenPIDVOC_C6H6 = df_SenPIDVOC_C6H6.dropna()
df_SenPIDVOC_C6H6
```

```
# correlazione per la stazione ATM05_01479
pearson, spearman, kendall= corr_Sensori(df_SenPIDVOC_C6H6, 'PIDVOC_ppb', 'C6H6_ug_m3', 'ATM05_01479' )
print('Correlazione tra PIDVOC e C6H6:\npearson: ', pearson,' spearman: ', spearman,' kendall: ', kendall)
```

Correlazione in stazioni diverse per lo stesso sensore

Per questa correlazione è stata sviluppata una funzione che permette di creare e restituire un dataframe costituito dalle colonne necessarie. Le colonne sono 2 e sono chiamate '**colonna1**' e '**colonna2**' a causa della riusabilità del codice della funzione (utilizzato anche per correlazioni che hanno diverse finalità). La funzione **creazione_df** prende in input un dataframe **df**, le due colonne, il composto e poi presenta un parametro di default, **df2 = None**,per l'elaborazione delle correlazioni successive. Viene inizializzato il primo dataframe temporaneo

df_1 prendendo le righe con postazione uguale al valore di **colonna1**, eliminati i valori e la colonna corrispondente al composto. Infine vengono resettati gli indici e reso un dataframe. Viene effettuato il controllo su **df2**, per verificare se è stato passato come parametro. Infine viene creato il dataframe **df_Full** concatenando i due dataframe, rinominate le colonne e sostituiti i valori **NaN** con 0.

```
def creazione_df(df, colonna1, colonna2, composto, df2 = None, ):
    df_1 = df[df['postazione'] == colonna1].dropna()[composto].reset_index(drop = True).to_frame()
    if df2 is None:
        df_2 = df.copy()
        df_2 = df_2[df_2['postazione'] == colonna2].dropna()[composto].reset_index(drop = True).to_frame()
    else:
        df_2 = df2.copy()
        df_2 = df_2[colonna2].dropna().reset_index(drop = True).to_frame()
    df_Full = pd.concat([df_1, df_2], axis=1)
    df_Full.columns = [colonna1,colonna2]
    df_Full = df_Full[[colonna1,colonna2]].fillna(0)
    return df_Full
```

Per effettuare la correlazione viene richiamata la funzione **corr_Sensori** e passati, come parametri, la funzione **creazione_df** con i suoi parametri e la **colonna1** e la **colonna2** come stringa. Infine vengono visualizzate le correlazioni di **Pearson** e **Spearman**. Questa operazione viene effettuata per tutte le combinazioni di sensori e stazioni.

```
# correlazione per il sensore TRS_ppb
pearson, spearman, kendall= corr_Sensori(creazione_df(df_Dati, 'ATM05_01479', 'ATM07_01480', 'TRS_ppb' ),
                                         'ATM05_01479', 'ATM07_01480')
print('pearson: ', pearson, ' spearman: ', spearman, ' kendall: ', kendall)
```

Correlazione con la temperatura per ogni stazione e sensore

Innanzitutto carichiamo il dataset dei dati meteo relativi alla zona di analisi.

```
df_DatiMeteo = pd.read_excel('Dati_Meteo.xlsx')
```

Definiamo una funzione che mi crea un dataframe ordinato con secondo la media oraria e per ogni postazione chiamato **media_Stazione**.

```
def media_Stazione(df, postazione):
    df1 = df[df['postazione'] == postazione]
    df2 = pd.concat([media_oraria(df1, 'C6H6_ug_m3')['C6H6_ug_m3'],
                     media_oraria(df1, 'H2S_ug_m3')['H2S_ug_m3'],
                     media_oraria(df1, 'H2SJ_ug_m3')['H2SJ_ug_m3'],
                     media_oraria(df1, 'VOC_ppm')['VOC_ppm'],
                     media_oraria(df1, 'PIDVOC_ppb')['PIDVOC_ppb']], axis=1)
    if postazione == 'ATM14_01486':
        df2['TRS_ppb'] = 0
    else:
        df2['TRS_ppb'] = media_oraria(df1, 'TRS_ppb')['TRS_ppb']
    return df2
```

Creiamo tre liste, una con tutte le postazioni, una con tutti i sensori e una con tutte le informazioni meteo necessarie.

```
stazione = ['ATM05_01479','ATM07_01480','ATM10_01481', 'ATM14_01486']
sensore = ['TRS_ppb','C6H6_ug_m3', 'VOC_ppm', 'PIDVOC_ppb', 'H2S_ug_m3', 'H2SJ_ug_m3']
meteo = ['temperatura_gradiC', 'direzione_vento_gradi','pressione_hPa', 'intensità_vento_km_h',
         'umidità_relativa', 'precipitazioni']
```

Creo un dataframe usando la funzione **media_Stazione** e gli aggiungo una colonna **'postazione'** utile per la funzione correlazione creata in precedenza.

```
# correlazione con la temperatura per il sensore ATM05_01479
df_ATM5 = media_Stazione(df_DatiPuliti, stazione[0]).fillna(0)
df_ATM5['postazione'] = 'ATM05_01479'
```

A questo punto viene richiamata la funzione **corr_Sensori** con i parametri relativi al dataframe, richiamando la funzione **creazione_df**, la **colonna 1** e la **colonna2**. Per la creazione del dataframe sono stati utilizzati il **df_ATM5** per **df**, **postazione[ ]** per indicare la stazione in **df_DatiPuliti, meteo[0]** per indicare le informazioni meteo da correlare, ovvero la temperatura, **sensore[ ]** per indicare il composto da analizzare e infine il dataframe **df_DatiMeteo**. Infine sono state mostrate a video le correlazioni di **Pearson** e **Spearman**.

```
pearson, spearman, kendall = corr_Sensori(creazione_df(df_ATM5,
                                          stazione[0], meteo[0], sensore[0], df2 = df_DatiMeteo), stazione[0], meteo[0])
print('pearson: ', pearson, ' spearman: ', spearman, ' kendall: ', kendall)
```

Correlazione con la direzione del vento per ogni stazione e sensore

Per questa correlazione sono state effettuate le stesse operazione per la correlazione precedente andando a modificare opportunamente i valori necessari, ovvero **meteo [1]** per la direzione del vento e **stazione[]** e **sensore[]** per confrontarli.

```
pearson, spearman, kendall = corr_Sensori(creazione_df(df_ATM5,
                                          stazione[0], meteo[1], sensore[0], df2 = df_DatiMeteo), stazione[0], meteo[1])
print('pearson: ', pearson, ' spearman: ', spearman, ' kendall: ', kendall)
```

Correlazione con la pressione atmosferica per ogni stazione e sensore

Per questa correlazione sono state effettuate le stesse operazione per la correlazione precedente andando a modificare opportunamente i valori necessari, ovvero **meteo [2]** per la pressione atmosferica e **stazione[]** e **sensore[]** per confrontarli.

```
pearson, spearman, kendall = corr_Sensori(creazione_df(df_ATM5,
                                        stazione[0], meteo[2], sensore[0], df2 = df_DatiMeteo), stazione[0], meteo[2])
print('pearson: ', pearson, ' spearman: ', spearman, ' kendall: ', kendall)
```

Correlazione con l'intensità del vento per ogni stazione e sensore

Per questa correlazione sono state effettuate le stesse operazione per la correlazione precedente andando a modificare opportunamente i valori necessari, ovvero **meteo [3]** per l'intensità del vento e **stazione[]** e **sensore[]** per confrontarli.

```
pearson, spearman, kendall = corr_Sensori(creazione_df(df_ATM5,
                                        stazione[0], meteo[3], sensore[0], df2 = df_DatiMeteo), stazione[0], meteo[3])
print('pearson: ', pearson, ' spearman: ', spearman, ' kendall: ', kendall)
```

Correlazione spurie tra i dati meteo e i fallimenti giornalieri

Per questa correlazione ci siamo creati un nuovo dataframe andando a copiare il dataframe **df_DatiMeteo**, raggruppando i valori per il giorno facendo una media. Infine è stato fatto un merge tra il nuovo dataframe e il dataframe **df_Fail_ND** sul giorno, calcolato in precedenza nella query 7.

```
df1 = df_DatiMeteo.copy()
df1['Giorno'] = pd.to_datetime(df1['Data'].dt.strftime('%Y/%m/%d'))
df1 = df1.groupby(['Giorno']).mean()
df_nuovo = df1.merge(df_Fail_ND, on= 'Giorno')
```

Per calcolare la correlazione è stata richiamata la funzione corr_Sensori sul nuovo dataframe, per ogni evento atmosferico.

```
# correlazione tra fallimenti e temperatura in gradi
pearson, spearman , kendall = corr_Sensori(df_nuovo,meteo[0], 'count')
print('pearson: ', pearson, ' spearman: ', spearman, ' kendall: ', kendall)
```

Odori

Per verificare che i composti emessi dal centro oli siano rilevabili a livello olfattivo abbiamo, innanzitutto, verificato un range di valori nel quale l'H2S viene rilevato, identificata come soglia minima.

```
# Considerando che la massa molare del H2S è 34.08 g/mol
# Per convertire il ppm in ug/m3 basta moliplicarlo
# per il peso molecolare (PM) della sostanza espressa in grammi
# e dividere tutto per 24,45 m3mol-1
puzza_min = (0.0005*34.08)/(24.45)
puzza_max = (0.3*34.08)/(24.45)
print('La soglia minima per la puzza in ug_m3 è: ',puzza_min)
print('La soglia massima per la puzza in ug_m3 è: ',puzza_max)
```

Successivamente è stato creato un dataframe per calcolare il livello di H2S sui sensori H2S e H2SJ nelle diverse stazioni andando a prendere la data e il sensore. È stata effettuata una formattazione della data in base al giorno, andando quindi a fare una media dopo aver raggruppato i valori sulla data. Infine è stata aggiunta una colonna H2S_odore e H2SJ_odore ai quali è stata applicata una funione lambda che va ad aggiungere il valore True se il valore del sensore è superiore al range, False se è inferiore al range e infine None se è compreso nel range.

```
df_H2S = df_DatiPuliti[df_DatiPuliti['postazione'] == 'ATM05_01479'][['Data','H2S_ug_m3']].reset_index(drop = True)
df_H2SJ = df_DatiPuliti[df_DatiPuliti['postazione'] == 'ATM05_01479'][['Data','H2SJ_ug_m3']].reset_index(drop = True)
df_od = pd.concat([df_H2S,df_H2SJ]).reset_index().drop(['index'], axis = 1)
df_od['Data'] = pd.to_datetime(df_od['Data'].dt.strftime('%Y/%m/%d'))
df_od = df_od.groupby('Data').mean()
df_od.insert(1,'H2S_odore', df_od['H2S_ug_m3'].apply(lambda x: True if x >= puzza_max else
                                                     (False if x <= puzza_min else None )))
df_od.insert(3,'H2SJ_odore', df_od['H2SJ_ug_m3'].apply(lambda x: True if x >= puzza_max else
                                                     (False if x <= puzza_min else None)))
df_od
```

Queste operazioni sono state effettuate per tutte le stazioni