```python
import pandas as pd
import itertools
import numpy as np
import matplotlib.pyplot as plt
import re

pm = pd.read_csv("aap_air_quality_database_2018_v14.csv", skiprows=2, sep=";")
pd.options.display.max_columns = None

#Aufteilung in zwei unterschiedliche DataFrames
pm10 = pm.drop(columns=pm.iloc[:,8:11])
pm25 = pm.drop(columns=pm.iloc[:,5:8])

pm = pd.read_csv("aap_air_quality_database_2018_v14_pm10_latest.csv")
pm10_latest = pm.drop(columns = pm.iloc[:,8:11])

pm = pd.read_csv("aap_air_quality_database_2018_v14_pm25_latest.csv")
pm25_latest = pm.drop(columns = pm.iloc[:,5:8])

#data = {"PM10": pm10, "PM2.5": pm25}

#Feinstaub EU- & WHO-Grenzwert PM10 -> 40 Mikrorgamm pro Kubikmeter (Quelle: Umweltbundesamt)
#Feinstaub EU-Grenzwert PM2.5 -> 25 Mikrogramm pro Kubikmeter (Quelle: Umweltbundesamt)
#feststehende Grenzwerte
PMLimits={"PM10": [40, pm10_latest], "PM2.5": [25, pm25_latest]}
plt.style.use("ggplot")

def zwischenlinie(length: int):
    r"""


    Parameters
    ----------
    length : int
        Wie viele Zeichen beinhalten die längste Werte in den jeweiligen Spalten zusammengerechnet.

    Returns
    -------
    None.

    """

    #Der Wert 10 ergibt sich aus den Leerzeichen am Anfang und am Ende jeder Spalte (3*2) und den
    #Rauten zum Trennen sowie am Anfang und Ende (4*1)
    print("".join(c for c in itertools.repeat("#", length + 10)))

def einkommensVergleich(df:pd.DataFrame, limit:int, df_info:str):
    r"""


    Parameters
    ----------
    df : pd.DataFrame
        Datensatz, auf dem der Vergleich, wie viele Städte prozentual den Maximal-Wert
        der Luftverschmutzung übersteigen.
    limit : int
        Je nachdem, welche Partikelmasse betrachtet wird, wird ein unterschiedlicher Maximal-Wert vorausgesetzt.
    df_info : str
        Wird lediglich als Extra-Parameter übergeben, um diesen in den Titel zu übernehmen.

    Returns
    -------
    None.

    """
```

```
67
68
69          print(str("Prozentualer Anteil der Städte, die die \nWHO-Grenzwerte einhalten ("
            + df_info +")\n" ))
70
71          df["limit"] = df.annual_mean <= limit
72          HighIncome = [False,True]
73          continents = df.region.unique()
74          continents.sort()
75
76          #Länge der längsten Strings ermitteln, um Zwischenlinie zu skalieren
77          maxCon = max([len(x) for x in continents])
78          lenNoV = len("No values")
79          lenNum = 5
80          length = maxCon+lenNoV+lenNum
81          zwischenlinie(length)
82
83          #Überschriften ausgeben, format-Methode, um String-Formatter nach Variable
            auszurichten
84          format = "# %%%ds" % maxCon
85          print(format % "Continent", end = " ")
86          print("# %5s" % "LMIC", end = " ")
87          print("# %9s #" % "HIC")
88          zwischenlinie(length)
89
90          #Datenermittlung: Für jeden Kontinent in der Liste wird nach LMIC und HIC
            gefiltert,
91          #ausgewertet und gleichzeitig ausgegeben.
92          for con in continents:
93              data = df.loc[df.region.str.contains(con)]
94              format = "# %%%ds" % maxCon
95              print(format % con, end = " ")
96
97              for ein in HighIncome:
98                  dataTemp = data.loc[data.HIC==ein]
99
100                 #Prüfen, ob der DataFrame überhaupt Daten enthält (gibt es HIC-Städte,
101                 #im jeweiligen Kontinent im Datensatz)
102                 if dataTemp.region.count() > 0:
103                     p = round((dataTemp.limit ==
                        True).sum()/dataTemp.region.count()*100,2)
104
105                     #Die Raute am Ende muss nur bei HIC (letzte Spalte) eingefügt werden
106                     if ein == False:
107                         print("# %5.2f" % p, end = " ")
108                     else:
109                         format = "# %%%d.2f #" % lenNoV
110                         print(format % p)
111             else:
112                 #percentage.append("No values")
113                 print("# No values #")
114         zwischenlinie(length)
115     print("\n")
116
117
118
119 def stadtEntwicklung(stadt: str):
120     r"""
121
122
123     Parameters
124     ----------
125     stadt : str
126         Stadt, für die die Entwicklung über die letzten Jahre über die Datenpunkte
127         inkl. einer Regressionsgraden angezeigt werden soll.
128         Voraussetzung: Mehr als 2 Dateneinträge.
129
130     Returns
131     -------
132     None.
133
134     """
```

```
135
136         frames = {"PM10": pm10, "PM2.5": pm25}
137         color = ["blue", "red"]
138         for index, key in enumerate(frames):
139             df = frames[key]
140             data = df.loc[df["city"] == stadt]
141
142             #Die Regression soll nur durchgeführt werden, wenn mehr als zwei Datensätze
143             #vorhanden sind.
144             if data["year"].count() > 2:
145
146                 #Hundert gleichverteilte Werte zur Regressionsberechnung
147                 xp = np.linspace(min(data["year"]), max(data["year"]), 100)
148
149                 #Ermittlung der Luft-Verschmutzungswerte
150                 anMean = np.array(data.annual_mean)
151
152                 #Berechnung der Regressionsgraden
153                 p = np.poly1d(np.polyfit(data["year"], anMean, 1))
154
155                 #Visuelle Darstellung
156                 plt.plot(data["year"], anMean, "o", c=color[index])
157                 plt.plot(xp, p(xp), c = color[index], label=key)
158             elif data["year"].count() == 0:
159                 print("Die angegebene Stadt wurde nicht gefunden")
160                 break
161             else:
162                 print("Zu dieser Stadt gibt es nicht genug Datenpunkte")
163                 break
164         plt.title(stadt)
165         plt.xticks(np.arange(min(data.year), max(data.year)+1))
166         plt.legend(loc="best")
167         plt.xlabel("Jahr")
168         plt.ylabel("Partikelmasse µg/m³")
169         plt.show()
170
171     def stadtRanking(country: str, asc = True):
172         r"""
173
174
175         Parameters
176         ----------
177         country : str
178             Land, für welches das Städteranking durchgeführt werden soll.
179         asc : TYPE, optional
180             Ob der DataFrame auf- (True) bzw. absteigend (False) ausgegeben werden soll.
181             The default is True.
182
183         Returns
184         -------
185         None.
186
187         """
188
189         frames = {"PM10": pm10, "PM2.5": pm25}
190         for key in frames:
191             df = frames[key]
192             data = df.loc[df["year"] == 2016]
193             data = data.loc[data["country"] == country].sort_values("annual_mean",
                ascending=asc)
194             data = data.loc[:, ["city", "annual_mean"]]
195             plt.barh(data.head(10).city, data.head(10).annual_mean)
196             plt.title(str("Top 10 Ranking Cities in " +country + " " +key+" (Best):"))
                if asc == True else plt.title(str("Top 10 Ranking Cities in " +country+ " "
                +key+" (Worst):"))
197             plt.gca().invert_yaxis()
198             plt.xlabel("Partikelmasse µg/m³")
199             plt.show()
200
201     def GetStationCount(value: str)->int:
202         r"""
203
```

```
204
205         Parameters
206         ----------
207         value : str
208             Jeweiliger Pandas-Dateneintrag.
209             Funktion zum Bereinigen und Addieren der Messstationen einer Stadt.
210
211         Returns
212         -------
213         int
214             Anzahl der Messstationen in der Stadt.
215
216         """
217         count = 0
218         strvalue = str(value)
219         for match in re.findall(r'(\d+)\s+\D+', strvalue):
220             count += int(match)
221         return count
222
223     def uebersichtMessstationen():
224         r"""
225
226         In dieser Methode wird lediglich ein Donut-Diagramm
227         zur Verteilung der Anzahl an Messtationen geplottet.
228
229         Returns
230         -------
231         None.
232
233         """
234         regions = pm10.region.unique()
235         regions.sort()
236         data=[]
237         for element in regions :
238             data.append(pm10.loc[pm10.region == element].monitor_station_count.sum())
239         plt.pie(data, labels= regions,wedgeprops=dict(width=0.5))
240         plt.title("Verteilung Anzahl an Messstationen nach Kontinenten")
241         plt.show()
242
243     def uebersichtWertVerteilung():
244         r"""
245
246         In dieser Methode werden alle Datenpunkte (PM10 und PM25)
247         in einem Scatter Diagramm angezeigt.
248         Dies zeigt die Verteilung der Datenpunkte an nach Regionen.
249
250
251         Returns
252         -------
253         None.
254
255         """
256         regions = pm10.region.unique()
257         plt.style.use("dark_background")
258         plt.figure(figsize=(12, 12), dpi= 100)
259         for element in regions:
260             data10=pm10.loc[pm10.region == element]
261             data25=pm25.loc[pm25.region == element]
262             plt.scatter(data25.annual_mean,data10.annual_mean, label=element, s=3)
263         plt.title("DatenPunkte (PM2.5 , PM10) Jahresdurchschnitt nach Regionen")
264         plt.xlabel("PM2.5 (annual mean)")
265         plt.ylabel("PM10 (annual mean)")
266         plt.legend()
267         plt.show()
268
269     def aufbereitung():
270         r"""
271
272         In dieser Methode werden die Datensätze gleich bereinigt.
273         Es wird nicht berücksichtigt, ob die Werte berechnet oder gemessen wurden.
274         Der Wert "temporal coverage" wurde durch eine Klassifikation ersetzt.
275         Das Einkommen der Stadt sowie die Angabe, ob es sich um einen berechneten oder
```

```
        einen
276     gemessenen Wert handelt, wurden als binäre Werte in eine separate Spalte
        aufgenommen bzw.
277     umgewandelt.
278
279     Returns
280     -------
281     None.
282
283     """
284
285     cols =
        ['region','iso3','country','city','year','annual_mean','temp_coverage','measured',
        'monitor_station_count','reference','db','status','HIC']
286     frames = [pm10, pm25, pm10_latest, pm25_latest]
287     for df in frames:
288
289         #Neue binäre Spalte HIC
290         df["HIC"]=df.Region.str.contains('HIC')
291         df.columns=cols
292         #Bereinigen der Region Spalte
293         df.region= df.region.str.split('(').str[0]
294
295         #Measured zu einem logischen Attribut machen
296         df.measured=df.measured.str.contains('measured', case=False)
297
298         #Bereinigen von Annual mean --> nur noch Wert
299         df.annual_mean.replace(r'\D', '', regex = True, inplace = True)
300         df.annual_mean= df.annual_mean.astype(int)
301         #Bereinigen von temp coverage -->
302         df.temp_coverage = df.temp_coverage.fillna(0)
303         df.temp_coverage= df.temp_coverage.astype(str)
304         temp = list(df.temp_coverage.unique())
305         temp.sort()
306         for index, cover in enumerate(temp):
307             df.temp_coverage.replace(cover, index, inplace = True)
308         df.monitor_station_count= [GetStationCount(x) for x in
            df.monitor_station_count]
309
310 def main():
311     aufbereitung()
312     for key in PMLimits:
313         einkommensVergleich(PMLimits[key][1],PMLimits[key][0], key)
314     stadtRanking("India", False)
315     stadtEntwicklung("Beijing")
316     stadtEntwicklung("Pasakha")
317     uebersichtMessstationen()
318     uebersichtWertVerteilung()
319
320
321 if __name__ == "__main__":
322     main()
```