T.C.

FIRAT ÜNİVERSİTESİ

TEKNOLOJİ FAKÜLTESİ

YAZILIM MÜHENDISLIĞI YMH 418 GÜNCEL KONULAR DERSİ PROJE DOSYASI

06.04.2020 – 10.04.2020

BÖLÜMÜ : YAZILIM MÜHENDİSLİĞİ

NUMARASI : 15542525

ADI ve SOYADI : ÇAĞDAŞ KARACA

II. Aşama - Verinin Görselleştirilmesi ve Sunumu:

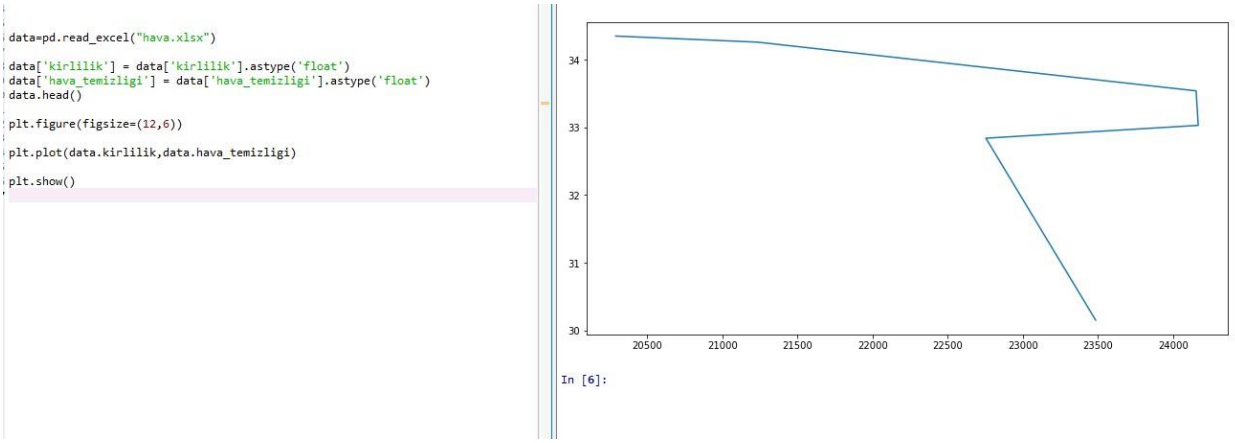
Veri Görselleştirme Nedir?
Karmaşık ve dağınık halde bulunan verileri düzenleyerek kolay anlaşılabilir, yorumlanabilir hale getirmektir. Projede kullancağımız hava kirliliği verilerinin görsel hale getirilmesi, somutlaştırılması ve bundan bir ön bilgi elde etmemizi sağlamaktadır. Veri görselleştirme için renklerin uyumlu olması, sade ve yalın olması, doğru grafik türünün seçilmesi, hangi verinin hangi grafikle görselleştirilecek olduğu önemlidir. Proje de veri madenciliği algoritmaları baz alınarak yazılan program üzerinde matplotlib kütüphanesi kullanılmıştır. Elde edilen hava verileri yazay düzlemde düzeltilerek csv dosyası haline getirilmiş, istatistiksel ve görsel grafikler oluşturulmuştur.

Matplotlib Nedir?
Matplotlib; veri görselleştirmede kullanılan temel python kütüphanelerinden biridir. 2boyutlu ve 3boyutlu olmak üzere grafiksel ve düzlemsel çizimler yapılmasını sağlar. Projede 2 boyutlu grafikler kullanılacağından Matplotlib ile çalışma yapılmıştır.

Matplotlib kütüphanesi program üzerinde “ import matplotlib.pyplot as plt ” komutu ile koda entegre edilmektedir.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

data=pd.read_excel("hava.xlsx")

data['kirlilik'] = data['kirlilik'].astype('float')
data['hava_temizligi'] = data['hava_temizligi'].astype('float')
data.head()

plt.figure(figsize=(12,6))

plt.plot(data.kirlilik,data.hava_temizligi)

plt.show()
```

tarih	kirlilik	hava_temizligi
2019-11	23.485	30.15
2019-12	22.754	32.84
2019-12	24.168	33.03
2020-01	24.154	33.54
2020-02	21.234	34.26
2020-03	20.289	34.35



Kodu ile oluşturulan csv test verileri görselleştirme yapılarak incelenmiştir ve ilerlemeler devam edilerek grafikler üzerindeki gelişme sağlanmıştır.

Veri madenciliği algoritmaları kullanılarak yapılan araştırmalar ve çalışmalar sonucu Kmeans algoritması oluşturulmuştur. Oluşturulan algoritma üzerinde hava kirliliği verileri içerisinde örnek veriler alınarak K means algoritması doğru sonuç vermemiştir. Hava kirliliği raporları kullanılarak yapılan testlerden veri setindeki havadaki moleküllerin yapısı, havada bulunan partikül ve gazların arasındaki bağlantı ve hesaplamalar incelenmiştir. Sonuç olarak havadaki SO2, NO2 Rspm,Spm ve Co2 değerleri baz alınması kararlaştırılmıştır.

```
import numpy as np
from cycler import cycler
from matplotlib.lines import Line2D
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
import dataloader_1b, random

FIRST_K_POINTS = 1
UNIFORMLY_K_POINTS = 2
K_MEANS_PLUS_PLUS = 3
algo = 4

DATA_SET_FILE = "data1b/C2.txt"

CATEGORY10 = np.array([ [31, 119, 180], [255, 127, 14], [44, 160, 44], [214, 39, 40],
                        [148, 103, 189], [140, 86, 75], [227, 119, 194], [127, 127, 127],
                        [188, 189, 34], [23, 190, 207] ]) / 255.0


def find_nearest_point(points, p):

    minimal_distance = euclidean(p, points[0])
    minimal_distance_point_index = 0

    for i in range(1, len(points)):
        distance = euclidean(p, points[i])
        if distance < minimal_distance:
            minimal_distance = distance
            minimal_distance_point_index = i

    return minimal_distance_point_index, minimal_distance

def k_means_cost_function(points, k_centers, points_labels):
    cost_function = 0.0
    for i in range(len(points)):
        cost_function += euclidean(points[i], k_centers[points_labels[i]]) ** 2

    return cost_function


def k_means(points, k, initialization_method):
    if k <= 0 or len(points) <= k:
        return False

    k_centers = np.zeros((k, len(points[0])), dtype = np.float64)

    if initialization_method == FIRST_K_POINTS:
        k_centers = points[0:k]

    elif initialization_method == UNIFORMLY_K_POINTS:
        random_array = np.zeros(len(points), dtype = np.int)
        for i in range(random_array.size - 1):
            random_array[i + 1] =  random_array[i] + 1
     for i in range(random_array.size):
            j = random.randint(0, random_array.size - 1)
            e = random_array[i]
            random_array[i] = random_array[j]
            random_array[j] = e

        for i in range(len(k_centers)):
            k_centers[i] = points[random_array[i]]
```

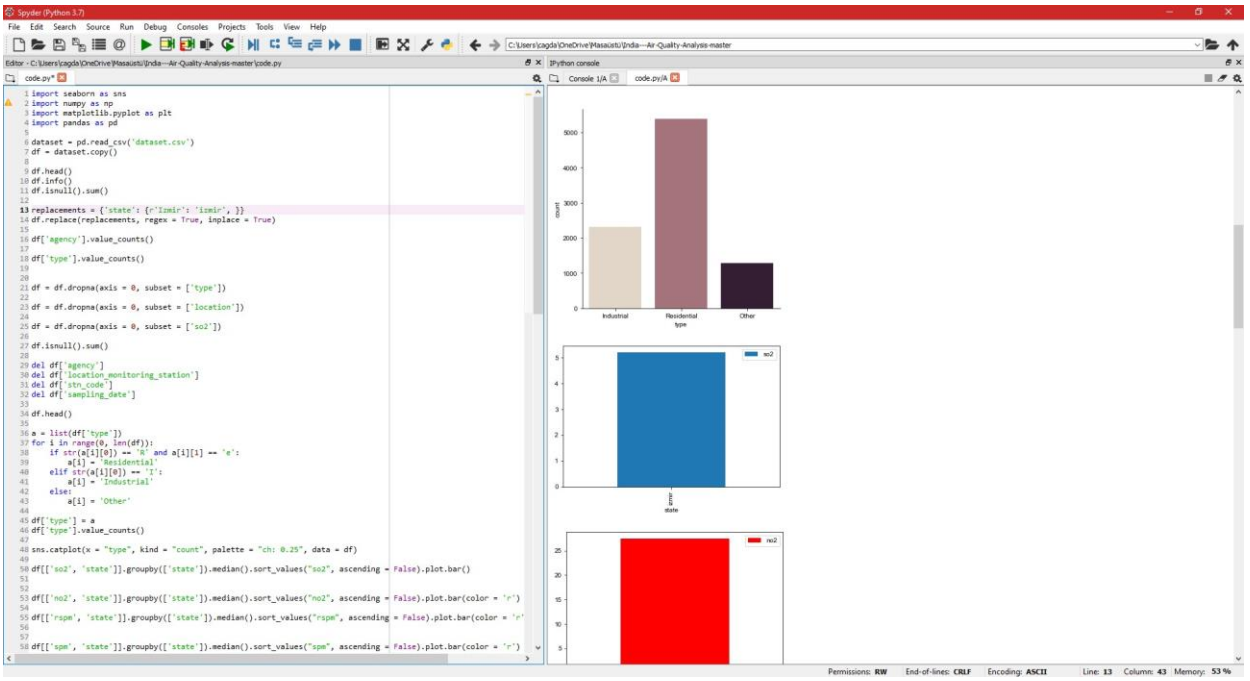
```
elif initialization_method == K_MEANS_PLUS_PLUS:
    c0_index = random.randint(0, len(points) - 1)
    k_centers[0] = points[c0_index]

    distribution = np.zeros(len(points), dtype = np.float64)

    for r in range(1, len(k_centers)):
        for i in range(len(points)):
            nearest_center_index, nearest_distance = find_nearest_point(k_centers[0: r], points[i])
            distribution[i] = nearest_distance ** 2

        sum_distances = np.sum(distribution)
        distribution /= sum_distances

        accumulate_distribution = np.zeros(len(distribution), dtype = np.float64)
        accumulate_distribution[0] = distribution[0]
        for i in range(1, len(distribution)):
            accumulate_distribution[i] = distribution[i] + accumulate_distribution[i - 1]

        random_number = random.random()
        for i in range(len(accumulate_distribution)):
            if random_number <= accumulate_distribution[i] and distribution[i] != 0:
                k_centers[r] = points[i]
                break

elif initialization_method == algo:
    k_centers[0] = points[0]

    for t in range(1, len(k_centers)):

        nearest_center_index, cost_function = find_nearest_point(k_centers[0: t], points[0])
        t_th_center_index = 0
        for i in range(1, len(points)):
            nearest_center_index, nearest_distance = find_nearest_point(k_centers[0: t], points[i])

            if nearest_distance > cost_function:
                t_th_center_index = i
                cost_function = nearest_distance

        k_centers[t] = points[t_th_center_index]

else:
    return False

points_labels = np.zeros(len(points), dtype = np.int)
k_means_cost_function_values = []
while True:
    for i in range(len(points)):
        nearest_center_index, nearest_distance = find_nearest_point(k_centers, points[i])
        points_labels[i] = nearest_center_index
    k_means_cost_function_values.append(k_means_cost_function(points, k_centers, points_labels))


    new_k_centers = np.zeros((len(k_centers), len(points[0])), dtype = np.float64)
    sums = np.zeros((len(k_centers), len(points[0])), dtype = np.float64)
    counts = np.zeros(len(k_centers), dtype = np.int)
    for i in range(len(points_labels)):
        sums[points_labels[i]] = np.add(sums[points_labels[i]], points[i])
        counts[points_labels[i]] += 1
    for i in range(len(new_k_centers)):
        new_k_centers[i] = sums[i] / counts[i]

    if np.linalg.norm(np.linalg.norm(new_k_centers - k_centers, axis = 1)) <= 10.0 ** (-10):
        k_centers = new_k_centers
        k_means_cost_function_values.append(k_means_cost_function(points, k_centers, points_labels))
```

```
            break
        else:
            k_centers = new_k_centers

    return k_centers, points_labels, k_means_cost_function_values

if __name__ == "__main__":
    points = dataloader_1b.load_data_1b(DATA_SET_FILE)
    points_x = [p[0] for p in points]
    points_y = [p[1] for p in points]

    for k in [3, 4, 5]:
        print ("k:", k)

        costs_different_initializations = {}

        print ("FIRST_K_POINTS")
        k_centers, points_labels, k_means_cost_function_values = k_means(points, k, FIRST_K_POINTS)
        costs_different_initializations["FIRST_K_POINTS"] = k_means_cost_function_values

        print ("Cost function", k_means_cost_function_values)

        k_centers_x = [c[0] for c in k_centers]
        k_centers_y = [c[1] for c in k_centers]

        fig1, (axes_clusters, axes_cost) = plt.subplots(1, 2)
        axes_clusters.scatter(points_x, points_y, c = [CATEGORY10[label] for label in points_labels], alpha = 0.8,
label = "clusters")
        axes_clusters.scatter(k_centers_x, k_centers_y, marker = "+", label = "centers")
        axes_clusters.set_ylim([min(points_x), max(points_y) + 5])
        axes_clusters.set_title("Clusters with first " + str(k) + " points initialization")
        axes_clusters.legend(loc = "upper right", fontsize = "medium")

        axes_cost.plot(k_means_cost_function_values)
        axes_cost.set_title("Cost function with first " + str(k) + " points initialization")
        axes_cost.set_xlabel("Number of iterations")
        axes_cost.set_ylabel("Cost function")

        print ("UNIFORMLY_K_POINTS")

        fig2, axeses = plt.subplots(3, 2)
        axeses = np.reshape(axeses, axeses.size)
        fig2.suptitle("Clusters and cost function with uniformly picked " + str(k) + " points initialization", fontsize =
"x-large")

        costs_different_initializations["UNIFORMLY_K_POINTS"] = []
        k_centers, points_labels, k_means_cost_function_values = None, None, None
        for i in range(5):
            print ("Run"), i + 1

            k_centers, points_labels, k_means_cost_function_values = k_means(points, k, UNIFORMLY_K_POINTS)
            costs_different_initializations["UNIFORMLY_K_POINTS"].append(k_means_cost_function_values)

            print ("Cost function"), k_means_cost_function_values

            k_centers_x = [c[0] for c in k_centers]
            k_centers_y = [c[1] for c in k_centers]
            axeses[i].scatter(points_x, points_y, c = [CATEGORY10[label] for label in points_labels], alpha = 0.8, label
= "clusters")
            axeses[i].scatter(k_centers_x, k_centers_y, marker = "+", label = "centers")
            axeses[i].set_ylim([min(points_x), max(points_y) + 5])
            axeses[i].set_title("Run " + str(i + 1), fontsize = "medium")
            axeses[i].legend(loc = "upper right", fontsize = "small")

        final_costs = np.array([costs_i[-1] for costs_i in costs_different_initializations["UNIFORMLY_K_POINTS"]])
        print( "Average:", np.average(final_costs), ", Standard deviation:", np.std(final_costs) )
```

```
    for i in range(5):
        axeses[-1].plot(costs_different_initializations["UNIFORMLY_K_POINTS"][i], label =
"UNIFORMLY_K_POINTS" + "_" + str(i + 1))
    axeses[-1].legend(loc = "upper right", fontsize = "small")
    axeses[-1].set_xlabel("Number of iterations")
    axeses[-1].set_ylabel("Cost function")

    print ("K_MEANS_PLUS_PLUS")
    fig3, axeses = plt.subplots(3, 2)
    fig3.suptitle("Clusters and cost function with k-means++, k= " + str(k), fontsize = "large")
    axeses = np.reshape(axeses, axeses.size)

    costs_different_initializations["K_MEANS_PLUS_PLUS"] = []
    k_centers, points_labels, k_means_cost_function_values = None, None, None
    for i in range(5):
        print("Run"), i + 1
        k_centers, points_labels, k_means_cost_function_values = k_means(points, k, K_MEANS_PLUS_PLUS)
        costs_different_initializations["K_MEANS_PLUS_PLUS"].append(k_means_cost_function_values)

        print ("Cost function"), k_means_cost_function_values

        k_centers_x = [c[0] for c in k_centers]
        k_centers_y = [c[1] for c in k_centers]

        axeses[i].scatter(points_x, points_y, c = [CATEGORY10[label] for label in points_labels], alpha = 0.8, label
= "clusters")
        axeses[i].scatter(k_centers_x, k_centers_y, marker = "+", label = "centers")
        axeses[i].set_ylim([min(points_x), max(points_y) + 5])
        axeses[i].set_title("Run " + str(i + 1), fontsize = "medium")
        axeses[i].legend(loc = "upper right", fontsize = "small")

    final_costs = np.array([costs_i[-1] for costs_i in costs_different_initializations["K_MEANS_PLUS_PLUS"]])
    print ("Average:", np.average(final_costs), ", Standard deviation:", np.std(final_costs))

    for i in range(5):
        axeses[-1].plot(costs_different_initializations["K_MEANS_PLUS_PLUS"][i], label =
"K_MEANS_PLUS_PLUS" + "_" + str(i + 1))
    axeses[-1].legend(loc = "upper right", fontsize = "small")
    axeses[-1].set_xlabel("Number of iterations")
    axeses[-1].set_ylabel("Cost function")

    k_centers, points_labels, k_means_cost_function_values = k_means(points, k, algo)
    costs_different_initializations[""] = k_means_cost_function_values

    print ("Cost function", k_means_cost_function_values)

    k_centers_x = [c[0] for c in k_centers]
    k_centers_y = [c[1] for c in k_centers]

    fig4, (axes_clusters, axes_cost) = plt.subplots(1, 2)
    axes_clusters.scatter(points_x, points_y, c = [CATEGORY10[label] for label in points_labels], alpha = 0.8,
label = "clusters")
    axes_clusters.scatter(k_centers_x, k_centers_y, marker = "+", label = "centers")
    axes_clusters.set_ylim([min(points_x), max(points_y) + 5])
    axes_clusters.set_title("k= " + str(k))
    axes_clusters.legend(loc = "upper right", fontsize = "medium")

    axes_cost.plot(k_means_cost_function_values)
    axes_cost.set_title("k= " + str(k))
    axes_cost.set_xlabel("Number of iterations")
    axes_cost.set_ylabel("Cost function")

    fig5, axes_costs = plt.subplots(1, 1)
    fig5.suptitle("Cost function with different initializations, k= " + str(k), fontsize = "large")
```

```
unfilled_markers = [m for m, func in Line2D.markers.iteritems()\
                    if func != 'nothing' and m not in Line2D.filled_markers]
axes_costs.set_prop_cycle(cycler('marker', unfilled_markers))

for key in costs_different_initializations:
    costs = costs_different_initializations[key]
    if all(isinstance(costs_i, list) for costs_i in costs):
        for i in range(len(costs)):
            axes_costs.plot(range(1, len(costs[i]) + 1), costs[i], label = key + "_" + str(i + 1))
    else:
        axes_costs.plot(range(1, len(costs) + 1), costs, label = key)

axes_costs.legend(loc = "upper right", fontsize = "medium")
axes_costs.set_xlabel("Number of iterations")
axes_costs.set_ylabel("Cost function")

plt.show()
```

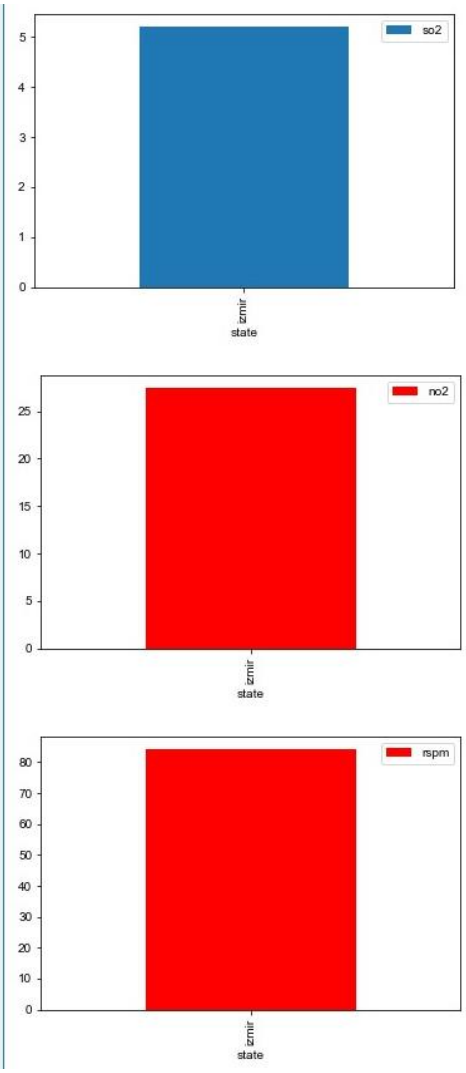
Veri setindeki istenilen maddelerin ve bu maddeler arasındaki bağıntı,hesaplamalar baz alınarak yapılan araştırmalar sonucu bu veriler üzerinde yapılan projeler incelenmiş, alınan veri seti üzerinde değişiklikler yapılıp, normalizasyon işlemleri uygulandıktan sonra proje de kullanılacak veri setlerinden çalıştırılan kod sayesinde aşağıdaki veriler elde edilmiştir.



Görsel 1. Semtlere Göre Verilerin Görselleştirilmesi

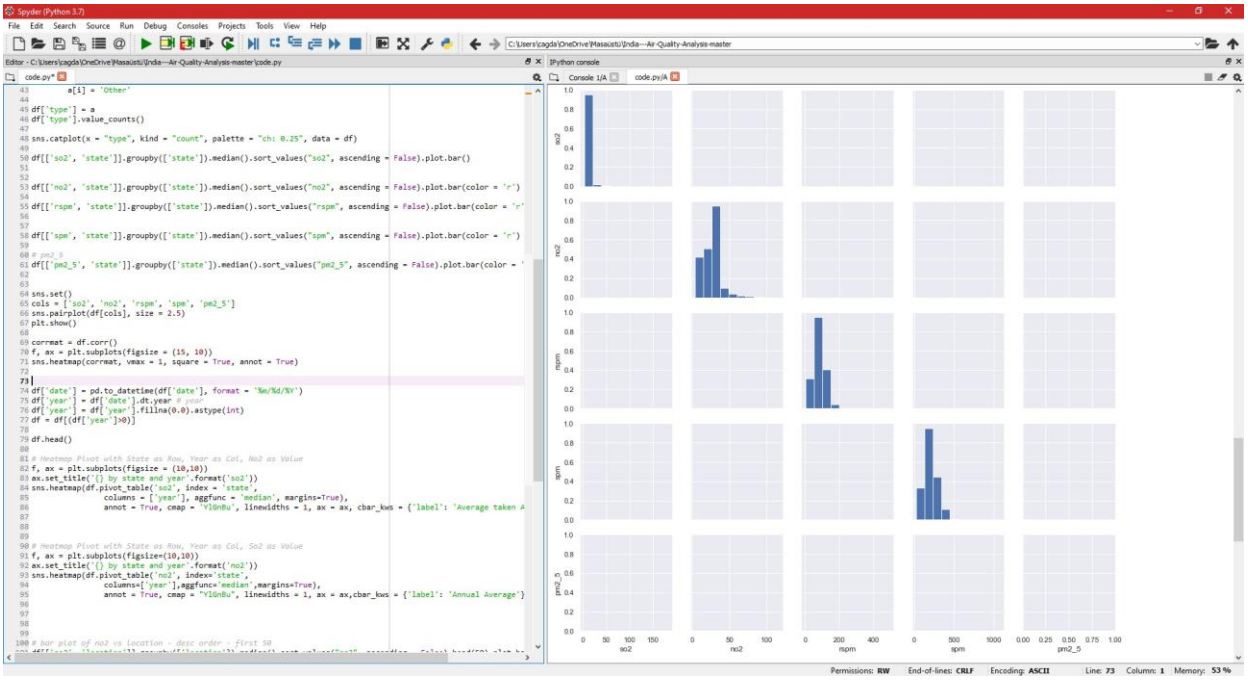
```
df = df.dropna(axis = 0, subset = ['type'])

df = df.dropna(axis = 0, subset = ['location'])

df = df.dropna(axis = 0, subset = ['so2'])

df.isnull().sum()

del df['agency']
del df['location_monitoring_station']
del df['stn_code']
del df['sampling_date']

df.head()

a = list(df['type'])
for i in range(0, len(df)):
    if str(a[i][0]) == 'R' and a[i][1] == 'e':
        a[i] = 'Residential'
    elif str(a[i][0]) == 'I':
        a[i] = 'Industrial'
    else:
        a[i] = 'Other'

df['type'] = a
df['type'].value_counts()

sns.catplot(x = "type", kind = "count", palette = "ch: 0.25", data = df)

df[['so2', 'state']].groupby(['state']).median().sort_values("so2", ascending = False).plot.bar()


df[['no2', 'state']].groupby(['state']).median().sort_values("no2", ascending = False).plot.bar(color = 'r')

df[['rspm', 'state']].groupby(['state']).median().sort_values("rspm", ascending = False).plot.bar(color = 'r'


df[['spm', 'state']].groupby(['state']).median().sort_values("spm", ascending = False).plot.bar(color = 'r')

# pm2_5
df[['pm2_5', 'state']].groupby(['state']).median().sort_values("pm2_5", ascending = False).plot.bar(color = '


#Scatter plots of all columns
sns.set()
cols = ['so2', 'no2', 'rspm', 'spm', 'pm2_5']
sns.pairplot(df[cols], size = 2.5)
plt.show()

#Correlation matrix
corrmat = df.corr()
f, ax = plt.subplots(figsize = (15, 10))
sns.heatmap(corrmat, vmax = 1, square = True, annot = True)


# Creating an year column
df['date'] = pd.to_datetime(df['date'], format = '%m/%d/%Y')
```

Görsel 2. Semtlere Göre Verilerin İncelenmesi İşlemini Sağlayan Kod ve Çıktısı



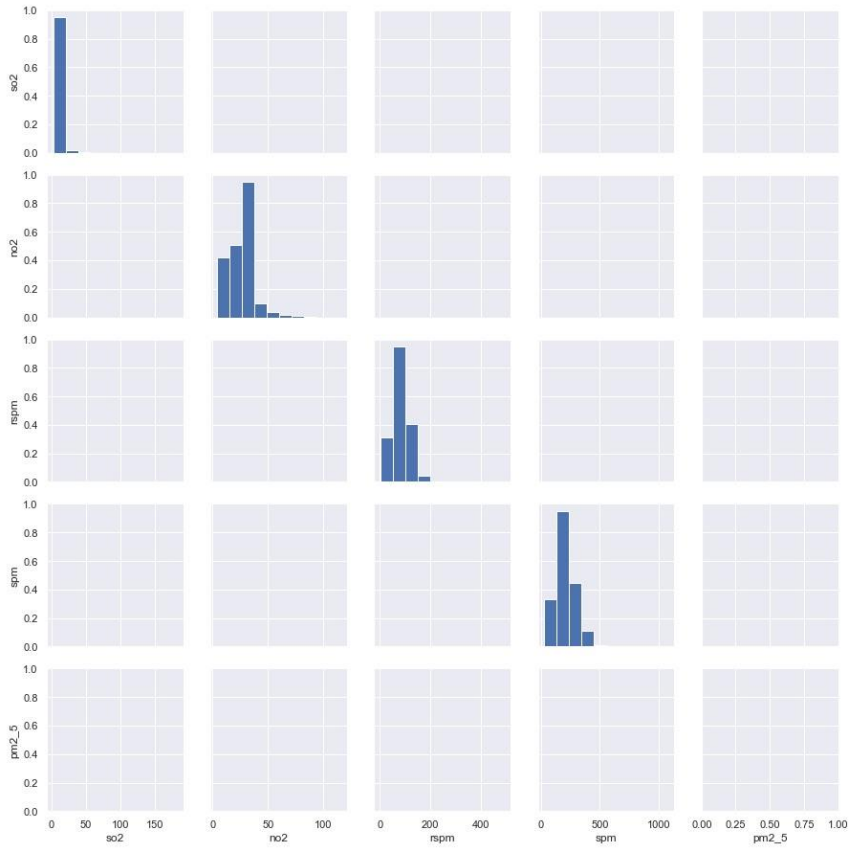
Görsel 3. Semtlere Göre Verilerin Görselleştirilmesinde Çalıştırılan Kodun Çıktısı

Yukarıda verilen Görsel 1 , Görsel 2 ve Görsel 3 veri seti okunup işlenerek location değerine göre İzmir de havada ölçülen maddelerin semtlere göre dağılımı verilmektedir.



Görsel 4. Verilerin Havadaki Maddelere Göre Aylık Dağılımları

Çalıştırılan kod üzerinde yapılan işlemde okunan veri setindeki havadaki SO2, NO2 Rspm,Spm ve Co2 değerlerinin aylık raporlara göre her grafik içerisinde İzmir de ki semtlerin verileri bulunmaktadır.



Görsel 5. Verilerin Havadaki Maddelere Göre Aylık Dağılımlarını Gösteren Grafiğin Yakın Görüntüsü

A	B	C	D	E	F	G	H	I	J	K	L	M	N
stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm				
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,8							
351	03/04/2020	izmir	izmir Bayraklı		Endustriyel alan	4,8							
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,3							
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,4							
351	03/04/2020	izmir	izmir Bayraklı		Endustriyel alan	6,1							
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	6,9							
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	8,2							
351	03/04/2020	izmir	izmir Bayraklı		Endustriyel alan	6,5							
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,1							
351	03/04/2020	izmir	izmir Bayraklı		Endustriyel alan	4,7							
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,7			133				
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,7			82				
351	03/04/2020	izmir	izmir Bayraklı		Endustriyel alan	5,2			111				
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,6			118				
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,1			135				
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,2			80				
351	03/04/2020	izmir	izmir Bayraklı		Endustriyel alan	5,9			179				
351	03/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	6,5			58				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	6,0			99				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,9			270				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,6			97				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	4,6			167				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,8			145				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,9			75				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	4,8			212				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,9			93				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,0			61				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	5,3			255				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,1			197				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	5,2			148				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	4,8			77				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	5,7			125				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	7,2			330				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	8,0			93				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	11,0			287				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	10,8			241				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	13,7			85				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	9,4							
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	8,8			283				
351	02/04/2020	izmir	izmir Bayraklı		Konut, Kırsal Alan ve Diger Alanlar	9,2			108				
351	02/04/2020	izmir	izmir Bayraklı		Endustriyel alan	9,3			234				

Görsel 6. Projede Kullanılan Veri Setinin Bir Kısmı

Saygılarımla,
15542525 Çağdaş Karaca