VERİ HAKKINDA GENEL BİLGİ

Adisyon excel dosyasında:

TARIH = Tarih (saat bilgisi olmaksızın)

ADISYON_NO = Her bir sipariş için unique bir no (fatura no gibi), primary key

GUNLUK_ADISYON_NO = O gün içindeki sipariş nosu (faturadaki günlük no gibi)

ADISYON_TIPI = M (masa müşterisi, bu durumda kişi sayısı sütununda bilgi de olacak)

H (gel-al müşterilerinin siparişi)

P (adrese sipariş isteyen müşterilerinin siparişi, bu durumda müşteri no olması gerekir)

KISI_SAYISI = M tipi ADISYON için masadaki toplam müşteri sayısı

MUSTERI_NO = P tipi ADISYON için adrese sipariş veren müşterinin nosu

GIRIS_SAATI = Siparişin gün ve saat olarak sisteme girildiği zaman

CIKIS_SAATI = M tipi satış için masa müşterisinin restorandan hesap istediği zaman

ADISYON_TUTATI = Satış tutarı

değişkenleri bulunmaktadır.

Arsivsip excel dosyasında:

ADISYON_NO = Her bir sipariş için unique bir no ama sipariş sayısına göre tekrar ediyor

SIPARIS_SIRA_NO = Her bir siparişteki ayrı ürünler için sipariş sıra no

SIPARIS_ADEDI = Üründen kaç adet sipariş edildiği bilgisi

URUN_ADI = Sipariş edilen ürün adı

URUN_KODU = Sipariş edilen ürünün kodu

ANA_GRUP = Sipariş edilen ürünün ana grubu

ANA_GRUP_KODU = Sipariş edilen ürünün ana grubunun kodu

ALT_GRUP = Sipariş edilen ürünün alt grubu (örneğin içecekler ana grubundaki sıcak içecekler ve soğuk içecekler şeklinde alt grup)
ALT_GRUP_KODU = Sipariş edilen ürünün alt grubunun kodu

URUN_KODU_FULL = Sipariş edilen ürünün ana grup, alt grup ve ürün kodunun bileşimi şeklinde full kodu

değişkenleri bulunmaktadır.

In [1]:
```
import pandas as pd
import time
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
import datetime as dt
from datetime import datetime, timedelta
from sqlalchemy import create_engine
from lifetimes import BetaGeoFitter
from lifetimes import GammaGammaFitter
```

In [2]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', 500)
warnings.filterwarnings('ignore')
warnings.simplefilter(action='ignore', category=Warning)
```

In [3]:
```
data=pd.read_excel("Arsivsip_201701_202106.xlsx")
```

In [4]:

In [4]:
```
data.head()
```

Out[4]:

	ADISYON_NO	SIPARIS_SIRA_NO	SIPARIS_ADEDI	URUN_ADI	URUN_KODU	ANA_GRUP	ANA_GRUP_KODU	ALT_GRUP	ALT_GRUP_KODU
0	56729	1	1	ÜRÜN_150	150	PİZZALAR	16	2-KİŞİLİK PİZZALAR	24
1	56730	1	1	ÜRÜN_313	313	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
2	56730	2	1	ÜRÜN_319	319	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
3	56731	1	1	ÜRÜN_125	125	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
4	56731	2	1	ÜRÜN_188	188	İÇECEKLER	13	SOĞUK İÇECEKLER	17

In [5]:
```
data2=pd.read_excel("Adisyon_201701_202106.xlsx")
```

In [6]:
```
data2.head()
```

Out[6]:

	TARIH	ADISYON_NO	GUNLUK_ADISYON_NO	ADISYON_TIPI	KISI_SAYISI	MUSTERI_NO	GIRIS_SAATI	CIKIS_SAATI	ADISYON_TUTARI
0	2017-01-01 00:00:00.000	56729	1	M	1.0	NaN	2017-01-01 12:43:43.000	2017-01-01 12:50:11.000	29.5
1	2017-01-01 00:00:00.000	56730	2	P	NaN	8948.0	2017-01-01 13:17:02.000	NaN	36.0
2	2017-01-01 00:00:00.000	56731	3	P	NaN	9619.0	2017-01-01 13:17:47.000	NaN	21.0
3	2017-01-01 00:00:00.000	56732	4	P	NaN	9803.0	2017-01-01 13:18:18.000	NaN	33.5
4	2017-01-01 00:00:00.000	56733	5	P	NaN	10129.0	2017-01-01 14:48:30.000	NaN	20.0

In [7]:
```
data2["TARIH"]=data2["TARIH"].apply(pd.to_datetime)
```

In [8]:
```
data2["DAY_OF_WEEKDAY"]=data2.TARIH.dt.dayofweek
```

In [9]:
```
df=data.merge(data2, on='ADISYON_NO', how='left')
```

In [10]:
```
df.head()
```

Out[10]:

	ADISYON_NO	SIPARIS_SIRA_NO	SIPARIS_ADEDI	URUN_ADI	URUN_KODU	ANA_GRUP	ANA_GRUP_KODU	ALT_GRUP	ALT_GRUP_KODU
0	56729	1	1	ÜRÜN_150	150	PİZZALAR	16	2-KİŞİLİK PİZZALAR	24
1	56730	1	1	ÜRÜN_313	313	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
2	56730	2	1	ÜRÜN_319	319	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
3	56731	1	1	ÜRÜN_125	125	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
4	56731	2	1	ÜRÜN_188	188	İÇECEKLER	13	SOĞUK İÇECEKLER	17

In [11]:
```
df["TARIH"].max()
```

Out[11]:
```
Timestamp('2021-06-30 00:00:00')
```

In [12]:
```
df["TARIH"].max()-df["TARIH"].min()
```

Out[12]:
```
Timedelta('1641 days 00:00:00')
```

In [13]:
```
df["TARIH"].nunique()
```

Out[13]:
```
1554
```

In [14]:
```
1641-1554 #87 gün eksik
```

Out[14]:
```
87
```

In [15]:
```
df["TARIH"].head(20)#17-04-2020 24-06-2020 arası yok pandemi yasakları
```

Out[15]:
```
0    2017-01-01
1    2017-01-01
2    2017-01-01
3    2017-01-01
4    2017-01-01
5    2017-01-01
6    2017-01-01
7    2017-01-01
8    2017-01-01
9    2017-01-01
10   2017-01-01
11   2017-01-01
12   2017-01-01
13   2017-01-01
14   2017-01-01
15   2017-01-01
16   2017-01-01
17   2017-01-01
18   2017-01-01
19   2017-01-01
Name: TARIH, dtype: datetime64[ns]
```

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
ADISYON_NO               0
SIPARIS_SIRA_NO          0
SIPARIS_ADEDI            0
URUN_ADI                 0
URUN_KODU                0
ANA_GRUP                 0
ANA_GRUP_KODU            0
ALT_GRUP                 0
ALT_GRUP_KODU            0
URUN_KODU_FULL           0
TARIH                    0
GUNLUK_ADISYON_NO        0
ADISYON_TIPI             0
KISI_SAYISI         118319
MUSTERI_NO           90878
GIRIS_SAATI              0
CIKIS_SAATI         118319
ADISYON_TUTARI           0
DAY_OF_WEEKDAY           0
dtype: int64
```

In [17]:
```
df["CIKIS_SAATI"]=df["CIKIS_SAATI"].apply(pd.to_datetime)
```

In [18]:
```
df.describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	75%	max
ADISYON_NO	198646.0	9.920944e+04	22920.308242	56729.0	80324.0	99174.5	117927.75	141067.0
SIPARIS_SIRA_NO	198646.0	2.138201e+00	1.442062	1.0	1.0	2.0	3.00	20.0
SIPARIS_ADEDI	198646.0	1.142047e+00	0.550068	1.0	1.0	1.0	1.00	52.0
URUN_KODU	198646.0	2.895893e+02	138.511772	100.0	159.0	265.0	419.00	515.0
ANA_GRUP_KODU	198646.0	1.430998e+01	2.197847	10.0	13.0	14.0	16.00	19.0
ALT_GRUP_KODU	198646.0	2.003816e+01	5.539714	10.0	17.0	20.0	23.00	34.0

URUN_KODU_FULL	198646.0	1.451326e+06	225138.370206	1010194.0	1317369.0	1420349.0	1623312.00	1934494.0
GUNLUK_ADISYON_NO	198646.0	3.210228e+01	21.377169	1.0	15.0	29.0	46.00	125.0
KISI_SAYISI	80327.0	3.008254e+00	1.983887	0.0	2.0	3.0	4.00	60.0
MUSTERI_NO	107768.0	8.542853e+03	4379.970340	1236.0	4978.0	8049.0	12119.00	17044.0
ADISYON_TUTARI	198646.0	7.127966e+01	47.013546	1.9	42.0	61.5	85.00	1064.0
DAY_OF_WEEKDAY	198646.0	3.299170e+00	1.969430	0.0	2.0	4.0	5.00	6.0

In [19]:
```
df.nunique()
```

Out[19]:
```
ADISYON_NO           84339
SIPARIS_SIRA_NO         20
SIPARIS_ADEDI           22
URUN_ADI               416
URUN_KODU              416
ANA_GRUP                10
ANA_GRUP_KODU           10
ALT_GRUP                21
ALT_GRUP_KODU           25
URUN_KODU_FULL         463
TARIH                 1554
GUNLUK_ADISYON_NO      125
ADISYON_TIPI             3
KISI_SAYISI             23
MUSTERI_NO           15809
GIRIS_SAATI          84339
CIKIS_SAATI          23406
ADISYON_TUTARI        2017
DAY_OF_WEEKDAY           7
dtype: int64
```

In [20]:
```
df.columns
```

Out[20]:
```
Index(['ADISYON_NO', 'SIPARIS_SIRA_NO', 'SIPARIS_ADEDI', 'URUN_ADI', 'URUN_KODU', 'ANA_GRUP', 'ANA_GRUP_KODU', 'ALT_GRUP', 'ALT_GRUP_KODU', 'URUN_KODU_FULL', 'TARIH', 'GUNLUK_ADISYON_NO', 'ADISYON_TIPI', 'KISI_SAYISI', 'MUSTERI_NO', 'GIRIS_SAATI', 'CIKIS_SAATI', 'ADISYON_TUTARI', 'DAY_OF_WEEKDAY'], dtype='object')
```

In [21]:
```
df["URUN_KODU_FULL"].nunique()
```

Out[21]:
```
463
```

In [22]:
```
df['MUSTERI_NO'].nunique()
```

Out[22]:
```
15809
```

KEŞİFÇİ VERİ ANALİZİ

1) 4 yıl boyunca satıştan kazanılan para 2) Haftanın günlerine göre ortalama günlük satış sayısı(adisyon) 3) Haftanın günlerine göre ortalama günlük satış tutarı 4) Haftanın günlerine ve satış türüne (M, H, P) göre ortalama günlük satış tutarı 5) Haftanın günlerine göre ortalama günlük sipariş sayısı 6) Haftanın günlerine ve satış türüne (M, H, P) göre ortalama günlük sipariş sayısı 7) Haftanın günlerine göre ortalama günlük satış 8) Pizzaları tek kişilik (small ve 1-kişilik), iki kişilik ve dört kişilik olarak gruplandırıp bunların haftanın günlerine göre ortalama günlük sipariş sayıları

1) 4 yıl boyunca satıştan kazanılan para

In [23]:
```
df[["ADISYON_NO","SIPARIS_SIRA_NO","ADISYON_TUTARI"]].tail(10)
```

Out[23]:

	ADISYON_NO	SIPARIS_SIRA_NO	ADISYON_TUTARI
198636	141061	2	99.0
198637	141061	3	99.0
198638	141062	1	28.0

198639	141063	1	65.0
198640	141064	1	34.0
198641	141065	1	29.0
198642	141066	1	38.0
198643	141067	1	92.0
198644	141067	2	92.0
198645	141067	3	92.0

In [24]:
```
df.groupby(["ADISYON_NO"])[["SIPARIS_SIRA_NO","ADISYON_TUTARI"]].max().head()
```

Out[24]:

ADISYON_NO	SIPARIS_SIRA_NO	ADISYON_TUTARI
56729	1	29.5
56730	2	36.0
56731	2	21.0
56732	1	33.5
56733	1	20.0

In [25]:
```
df.groupby(["ADISYON_NO"])[["SIPARIS_SIRA_NO","ADISYON_TUTARI"]].max().sum()
```

Out[25]:
```
SIPARIS_SIRA_NO     199183.00
ADISYON_TUTARI     4938192.88
dtype: float64
```

In [26]:
```
df.groupby(["DAY_OF_WEEKDAY","ADISYON_NO"])[["ADISYON_TUTARI"]].max().head()
```

Out[26]:

DAY_OF_WEEKDAY	ADISYON_NO	ADISYON_TUTARI
0	**56749**	34.9
	56750	35.9
	56751	21.0
	56752	26.0
	56753	85.8

In [27]:
```
df.groupby(["TARIH","ADISYON_NO"])[["ADISYON_NO","ADISYON_TUTARI"]].agg({"ADISYON_NO":"count",
                                                                         "ADISYON_TUTARI":"mean"}).head()
```

Out[27]:

		ADISYON_NO	ADISYON_TUTARI
TARIH	ADISYON_NO		
2017-01-01	**56729**	1	29.5
	56730	2	36.0
	56731	2	21.0
	56732	1	33.5
	56733	1	20.0

In [28]:
```
df["DAY_OF_WEEKDAY"].unique()
```

Out[28]:
```
array([6, 0, 1, 2, 3, 4, 5], dtype=int64)
```

2) Haftanın günlerine göre ortalama günlük satış sayısı(adisyon)

In [29]:
```
data2.groupby(["TARIH","DAY_OF_WEEKDAY"])[["GUNLUK_ADISYON_NO"]].max().head()
```

Out[29]:

		GUNLUK_ADISYON_NO
TARIH	DAY_OF_WEEKDAY	
2017-01-01	6	20
2017-01-02	0	15
2017-01-03	1	22
2017-01-04	2	19
2017-01-05	3	13

In [30]:
```
df5=data2.groupby(["TARIH","DAY_OF_WEEKDAY"])[["GUNLUK_ADISYON_NO"]].max()
```

In [31]:
```
df5=df5.reset_index()
```

In [32]:
```
df5.groupby("DAY_OF_WEEKDAY")[["GUNLUK_ADISYON_NO"]].mean()
```

Out[32]:

	GUNLUK_ADISYON_NO
DAY_OF_WEEKDAY	
0	43.923077
1	47.122172
2	49.267857
3	51.035874
4	61.321267
5	64.345291
6	62.895928

3) Haftanın günlerine göre ortalama günlük satış tutarı

In [33]:
```
datatop=data2.groupby(["TARIH","DAY_OF_WEEKDAY"])["ADISYON_TUTARI"].sum()
```

In [34]:
```
datatop=datatop.reset_index()
```

In [35]:
```
datatop.groupby(['DAY_OF_WEEKDAY'])[["ADISYON_TUTARI"]].mean()
```

Out[35]:

	ADISYON_TUTARI
DAY_OF_WEEKDAY	
0	2468.050181
1	2740.816380
2	2840.138214
3	2956.089641
4	3689.262670
5	3822.622870
6	3727.884480

4) Haftanın günlerine ve satış türüne (M, H, P) göre ortalama günlük satış tutarı

In [36]:
```
df_sales=df.groupby(["TARIH","ADISYON_TIPI","ADISYON_NO"])["ADISYON_TUTARI",'DAY_OF_WEEKDAY'].max()
```

In [37]:

In [37]:
```
df_sales.head()
```

Out[37]:

TARIH	ADISYON_TIPI	ADISYON_NO	ADISYON_TUTARI	DAY_OF_WEEKDAY
2017-01-01	M	56729	29.5	6
		56746	20.0	6
	P	56730	36.0	6
		56731	21.0	6
		56732	33.5	6

In [38]:
```
df_sales=df_sales.reset_index()
```

In [39]:
```
df_sales.groupby(["TARIH","DAY_OF_WEEKDAY","ADISYON_TIPI"])["ADISYON_TUTARI"].sum().head()
```

Out[39]:
```
TARIH       DAY_OF_WEEKDAY  ADISYON_TIPI
2017-01-01  6               M               49.5
                            P              618.4
2017-01-02  0               M               33.5
                            P              498.2
2017-01-03  1               M               33.0
Name: ADISYON_TUTARI, dtype: float64
```

In [40]:
```
df_sales2=df_sales.groupby(["TARIH","ADISYON_TIPI","DAY_OF_WEEKDAY"])["ADISYON_TUTARI"].sum()
```

In [41]:
```
df_sales2=df_sales2.reset_index()
```

In [42]:
```
df_sales2.head()
```

Out[42]:

	TARIH	ADISYON_TIPI	DAY_OF_WEEKDAY	ADISYON_TUTARI
0	2017-01-01	M	6	49.5
1	2017-01-01	P	6	618.4
2	2017-01-02	M	0	33.5
3	2017-01-02	P	0	498.2
4	2017-01-03	M	1	33.0

In [43]:
```
df_sales2[df_sales2["ADISYON_TIPI"]=="P"].groupby(["DAY_OF_WEEKDAY"])["ADISYON_TUTARI"].mean()
```

Out[43]:
```
DAY_OF_WEEKDAY
0    1492.859864
1    1585.340452
2    1648.896607
3    1697.217668
4    2093.038643
5    2289.643184
6    2293.531041
Name: ADISYON_TUTARI, dtype: float64
```

In [44]:
```
df_sales2[df_sales2["ADISYON_TIPI"]=="M"].groupby(["DAY_OF_WEEKDAY"])["ADISYON_TUTARI"].mean()
```

Out[44]:
```
DAY_OF_WEEKDAY
0     865.506784
1    1041.968112
2    1079.674673
3    1154.663316
4    1401.406683
5    1455.234694
6    1379.044844
Name: ADISYON_TUTARI, dtype: float64
```

In [45]:
```
df_sales2[df_sales2["ADISYON_TIPI"]=="H"].groupby(["DAY_OF_WEEKDAY"])["ADISYON_TUTARI"].mean()
```

Out[45]:
```
DAY_OF_WEEKDAY
0    282.883725
1    343.184094
2    337.551039
3    355.649935
4    443.830318
5    356.153899
6    336.874194
Name: ADISYON_TUTARI, dtype: float64
```

0:pazartesi 1:salı 2:çarşamba 3:perşembe 4:cuma 5:cumartesi 6:pazar

5) Haftanın günlerine göre ortalama günlük sipariş sayısı

In [46]:
```
df.groupby(["TARIH","DAY_OF_WEEKDAY"])[["SIPARIS_ADEDI"]].sum().head()
```

Out[46]:

		SIPARIS_ADEDI
TARIH	**DAY_OF_WEEKDAY**	
2017-01-01	**6**	23
2017-01-02	**0**	20
2017-01-03	**1**	31
2017-01-04	**2**	40
2017-01-05	**3**	22

In [47]:
```
dfsip=df.groupby(["TARIH","DAY_OF_WEEKDAY"])[["SIPARIS_ADEDI"]].sum()
```

In [48]:
```
dfsip=dfsip.reset_index()
```

In [49]:
```
dfsip.groupby("DAY_OF_WEEKDAY")[["SIPARIS_ADEDI"]].mean()
```

Out[49]:

	SIPARIS_ADEDI
DAY_OF_WEEKDAY	
0	113.914027
1	126.153846
2	131.450893
3	137.636771
4	169.402715
5	173.757848
6	169.610860

toplam sipariş sayısı fatura sayısı değil,adisyon içindeki her bir ürün sipariş olarak kabul ediliyor.

6) Haftanın günlerine ve satış türüne (M, H, P) göre ortalama günlük sipariş sayısı

In [50]:
```
df.groupby(["TARIH","ADISYON_TIPI",'DAY_OF_WEEKDAY'])[["SIPARIS_ADEDI"]].sum().head()
```

Out[50]:

			SIPARIS_ADEDI
TARIH	**ADISYON_TIPI**	**DAY_OF_WEEKDAY**	
2017-01-01	**M**	**6**	2
	P	**6**	21
2017-01-02	**M**	**0**	1
	P	**0**	19
2017-01-03	**M**	**1**	1

In [51]:
```
df4=df.groupby(["TARIH","ADISYON_TIPI",'DAY_OF_WEEKDAY'])[["SIPARIS_ADEDI"]].sum()
```

In [52]:
```
df4=df4.reset_index()
```

In [53]:
```
df4.groupby(["ADISYON_TIPI","DAY_OF_WEEKDAY"]).mean().head()
```

Out[53]:

		SIPARIS_ADEDI
ADISYON_TIPI	DAY_OF_WEEKDAY	
H	0	9.267974
	1	10.899329
	2	10.532468
	3	10.960784
	4	13.382166

7) Haftanın günlerine göre ortalama günlük satış

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

Out[54]:

	ADISYON_NO	SIPARIS_SIRA_NO	SIPARIS_ADEDI	URUN_ADI	URUN_KODU	ANA_GRUP	ANA_GRUP_KODU	ALT_GRUP	ALT_GRUP_KODU
0	56729	1	1	ÜRÜN_150	150	PİZZALAR	16	2-KİŞİLİK PİZZALAR	24
1	56730	1	1	ÜRÜN_313	313	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
2	56730	2	1	ÜRÜN_319	319	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
3	56731	1	1	ÜRÜN_125	125	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
4	56731	2	1	ÜRÜN_188	188	İÇECEKLER	13	SOĞUK İÇECEKLER	17

In [55]:
```
df.groupby(['DAY_OF_WEEKDAY',"TARIH","ADISYON_NO","ANA_GRUP"])[["SIPARIS_SIRA_NO","ADISYON_TUTARI"]].max().head()
```

Out[55]:

				SIPARIS_SIRA_NO	ADISYON_TUTARI
DAY_OF_WEEKDAY	TARIH	ADISYON_NO	ANA_GRUP		
0	2017-01-02	56749	KAMPANYALAR	1	34.9
		56750	KAMPANYALAR	1	35.9
		56751	PİZZALAR	1	21.0
		56752	KAMPANYALAR	1	26.0
		56753	KAMPANYALAR	1	85.8

In [56]:
```
dataf=df.groupby(['DAY_OF_WEEKDAY',"TARIH","ADISYON_NO","ANA_GRUP"])[["SIPARIS_SIRA_NO","ADISYON_TUTARI"]].max()
```

In [57]:
```
dataf=dataf.reset_index()
```

In [58]:
```
dataf.groupby(["TARIH",'DAY_OF_WEEKDAY'])["ADISYON_NO"].count().groupby('DAY_OF_WEEKDAY').mean()
```

Out[58]:
```
DAY_OF_WEEKDAY
0     79.402715
1     86.407240
2     90.044643
3     94.304933
4    114.651584
5    119.026906
```

```
6    117.000000
Name: ADISYON_NO, dtype: float64
```

8) Pizzaları tek kişilik (small ve 1-kişilik), iki kişilik ve dört kişilik olarak gruplandırıp bunların haftanın günlerine göre ortalama günlük sipariş sayıları

In [59]:
```
df.head()
```

Out[59]:

	ADISYON_NO	SIPARIS_SIRA_NO	SIPARIS_ADEDI	URUN_ADI	URUN_KODU	ANA_GRUP	ANA_GRUP_KODU	ALT_GRUP	ALT_GRUP_KODU
0	56729	1	1	ÜRÜN_150	150	PİZZALAR	16	2-KİŞİLİK PİZZALAR	24
1	56730	1	1	ÜRÜN_313	313	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
2	56730	2	1	ÜRÜN_319	319	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
3	56731	1	1	ÜRÜN_125	125	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
4	56731	2	1	ÜRÜN_188	188	İÇECEKLER	13	SOĞUK İÇECEKLER	17

In [60]:
```
df["ALT_GRUP"].value_counts()
```

Out[60]:
```
SOĞUK İÇECEKLER                   52029
1-KİŞİLİK PİZZALAR                42956
2-KİŞİLİK PİZZALAR                21459
APERATİFLER                       17975
4-KİŞİLİK PİZZALAR                12279
KAMPANYALAR                        8635
2-KİŞİLİK FIFTY-FIFTY PİZZALAR     7566
4-KİŞİLİK FIFTY-FIFTY PİZZALAR     6905
TATLILAR                           6160
1-KİŞİLİK PİZZA İLAVE MALZEMELERİ  5330
SMALL PİZZALAR                     3209
HAMBURGERLER                       3141
SALATALAR                          2684
SANDVİCHLER                        1918
WRAPLER                            1500
MAKARNALAR                         1222
2-KİŞİLİK PİZZA İLAVE MALZEMELERİ  1064
SOSLAR                              889
SICAK İÇECEKLER                     798
CALZONELAR (KAPALI PİZZALAR)        613
4-KİŞİLİK PİZZA İLAVE MALZEMELERİ   314
Name: ALT_GRUP, dtype: int64
```

In [61]:
```
df["ALT_GRUP_N"]=[i[:2] for i in df["ALT_GRUP"].values]
```

In [62]:
```
df["ALT_GRUP_N"]=["3" if i=="SM" else i for i in df["ALT_GRUP_N"].values]
```

In [63]:
```
df["ALT_GRUP_N"]=df["ALT_GRUP_N"].astype(str)
```

In [64]:
```
df["ALT_GRUP_N"]=[i[:1] for i in df["ALT_GRUP_N"].values]
```

In [65]:
```
df["ALT_GRUP_N"]=[0 if i not in ["1","2","3","4"] else i for i in df["ALT_GRUP_N"].values]
```

0:diğer 1:tek kişilik pizza 2: 2 kişilik pizza 3:small pizza 4:4 kişilik pizza

In [66]:
```
df.head()
```

Out[66]:

	ADISYON_NO	SIPARIS_SIRA_NO	SIPARIS_ADEDI	URUN_ADI	URUN_KODU	ANA_GRUP	ANA_GRUP_KODU	ALT_GRUP	ALT_GRUP_KODU
0	56729	1	1	ÜRÜN_150	150	PİZZALAR	16	2-KİŞİLİK PİZZALAR	24

1	56730	1	1	ÜRÜN_313	313	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
2	56730	2	1	ÜRÜN_319	319	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
3	56731	1	1	ÜRÜN_125	125	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
4	56731	2	1	ÜRÜN_188	188	İÇECEKLER	13	SOĞUK İÇECEKLER	17

In [67]:
```
df.groupby('DAY_OF_WEEKDAY')["ALT_GRUP_N"].value_counts()
```

Out[67]:
```
DAY_OF_WEEKDAY  ALT_GRUP_N
0               0             10832
                1              5698
                2              3214
                4              1911
                3               417
1               0             11933
                1              5985
                2              3701
                4              2323
                3               397
2               0             12575
                1              6604
                2              3761
                4              2286
                3               448
3               0             13260
                1              6583
                2              4056
                4              2468
                3               441
4               0             15931
                1              7832
                2              5045
                4              3233
                3               514
5               0             16760
                1              7801
                2              5320
                4              3759
                3               512
6               0             16273
                1              7783
                2              4992
                4              3518
                3               480
Name: ALT_GRUP_N, dtype: int64
```

In [68]:
```
df.describe().T
```

Out[68]:

	count	mean	std	min	25%	50%	75%	max
ADISYON_NO	198646.0	9.920944e+04	22920.308242	56729.0	80324.0	99174.5	117927.75	141067.0
SIPARIS_SIRA_NO	198646.0	2.138201e+00	1.442062	1.0	1.0	2.0	3.00	20.0
SIPARIS_ADEDI	198646.0	1.142047e+00	0.550068	1.0	1.0	1.0	1.00	52.0
URUN_KODU	198646.0	2.895893e+02	138.511772	100.0	159.0	265.0	419.00	515.0
ANA_GRUP_KODU	198646.0	1.430998e+01	2.197847	10.0	13.0	14.0	16.00	19.0
ALT_GRUP_KODU	198646.0	2.003816e+01	5.539714	10.0	17.0	20.0	23.00	34.0
URUN_KODU_FULL	198646.0	1.451326e+06	225138.370206	1010194.0	1317369.0	1420349.0	1623312.00	1934494.0
GUNLUK_ADISYON_NO	198646.0	3.210228e+01	21.377169	1.0	15.0	29.0	46.00	125.0
KISI_SAYISI	80327.0	3.008254e+00	1.983887	0.0	2.0	3.0	4.00	60.0
MUSTERI_NO	107768.0	8.542853e+03	4379.970340	1236.0	4978.0	8049.0	12119.00	17044.0
ADISYON_TUTARI	198646.0	7.127966e+01	47.013546	1.9	42.0	61.5	85.00	1064.0
DAY_OF_WEEKDAY	198646.0	3.299170e+00	1.969430	0.0	2.0	4.0	5.00	6.0

ÜRÜN VE MÜŞTERİLERE GÖRE YİYECEK TAVSİYE SİSTEMİ

ITEM BASED

In [69]:
```
df[df["ADISYON_TIPI"]=="P"].groupby(["MUSTERI_NO","URUN_KODU_FULL"])[["SIPARIS_ADEDI"]].sum().head()
```

Out[69]:

		SIPARIS_ADEDI
MUSTERI_NO	URUN_KODU_FULL	
1236.0	1623129	1
	1623307	1
1237.0	1010463	1
	1317188	1
	1420433	1

In [70]:
```
df_user=df[df["ADISYON_TIPI"]=="P"].groupby(["MUSTERI_NO","URUN_KODU_FULL"])[["SIPARIS_ADEDI"]].sum()
```

In [71]:
```
user_eat_df = df_user.pivot_table(index=["MUSTERI_NO"], columns=["URUN_KODU_FULL"], values="SIPARIS_ADEDI")
```

In [72]:
```
user_eat_df.T.sum().sort_values(ascending=False).head()
```

Out[72]:
```
MUSTERI_NO
14128.0    1844.0
12932.0     687.0
6983.0      341.0
3942.0      313.0
6848.0      303.0
dtype: float64
```

In [73]:
```
random_eat=pd.Series(df["URUN_KODU_FULL"]).sample(1, random_state=42).values[0]
```

In [74]:
```
random_eat
```

Out[74]:
```
1623319
```

In [75]:
```
eat_id=user_eat_df.get(random_eat).notnull()
```

In [76]:
```
user_eat_df.corrwith(eat_id).sort_values(ascending=False).head(15)
```

Out[76]:
```
URUN_KODU_FULL
1419264    1.000000
1419271    0.792118
1418200    0.676025
1112448    0.672910
1623316    0.643482
1317471    0.638009
1622299    0.600925
1832426    0.600088
1622289    0.582815
1832425    0.549134
1622298    0.518978
1624148    0.515094
1629283    0.478660
1622103    0.464814
1113448    0.462910
dtype: float64
```

In [77]:
```
random_eat
```

Out[77]:
```
1623319
```

ITEM BASED SONUCU URUN KODU 1419264 OLAN YİYECEK İLE RANDOM SEÇİLEN YİYECEK(1623319) ARASINDAKİ KORELASYON 1 (PİZZA/KOLA OLABİLİR.) FONKSİYON HALİNE GETİRELİM.

In [78]:
```
def cor_eat(urunkodu):
    df_user=df[df["ADISYON_TIPI"]=="P"].groupby(["MUSTERI_NO","URUN_KODU_FULL"])[["SIPARIS_ADEDI"]].sum()
    user_eat_df = df_user.pivot_table(index=["MUSTERI_NO"], columns=["URUN_KODU_FULL"], values="SIPARIS_ADEDI")
    eat_id=user_eat_df.get(urunkodu).notnull()
    return user_eat_df.corrwith(eat_id).sort_values(ascending=False).head(15)
```

In [79]:
```
#ürün kodu verilir verilen ürünün diğer ürünlerle ilişkisine bakılır.
# item based
cor_eat(1832426)
```

Out[79]:
```
URUN_KODU_FULL
1420321    0.959543
1622293    0.939842
1622103    0.907872
1622108    0.896489
1624148    0.873909
1622289    0.832253
1623318    0.758579
1111466    0.708550
1111467    0.705718
1111495    0.671370
1622298    0.640435
1622290    0.628725
1622288    0.616476
1111383    0.614110
1215510    0.606339
dtype: float64
```

In [80]:
```
cor_eat(1622293)
```

Out[80]:
```
URUN_KODU_FULL
1420321    0.673465
1215510    0.618718
1622289    0.570930
1625176    0.541005
1624160    0.540444
1622109    0.535976
1317457    0.527330
1111424    0.522756
1622296    0.512771
1112414    0.490566
1111467    0.485490
1623134    0.476556
1111466    0.466343
1629283    0.465246
1622101    0.461017
dtype: float64
```

USER BASED

In [81]:
```
import numpy as np
```

In [82]:
```
user_eat_df.index[:5]
```

Out[82]:
```
Float64Index([1236.0, 1237.0, 1238.0, 1239.0, 1240.0], dtype='float64', name='MUSTERI_NO')
```

In [83]:
```
random_user=14285.0
```

In [84]:
```
random_user_df=user_eat_df[user_eat_df.index==random_user]
```

In [85]:
```
random_user_df
```

Out[85]:

URUN_KODU_FULL	1010194	1010280	1010370	1010385	1010389	1010393	1010410	1010411	1010412	1010413	1010415	1010441	1010447
MUSTERI_NO													
14285.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [86]:
```
random_user_df.columns[random_user_df.notna().any()].tolist()
```

Out[86]:
```
[1111436, 1111467, 1317417, 1317460, 1622109, 1623127, 1623131, 1934435]
```

In [87]:
```
eatTo_eat=random_user_df.columns[random_user_df.notna().any()].tolist()
```

In [88]:
```
user_eat_df.loc[user_eat_df.index==random_user,user_eat_df.columns]
```

Out[88]:

URUN_KODU_FULL	1010194	1010280	1010370	1010385	1010389	1010393	1010410	1010411	1010412	1010413	1010415	1010441	1010447
MUSTERI_NO													
14285.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [89]:
```
user_eat_df.loc[user_eat_df.index == random_user, user_eat_df.columns == eatTo_eat[0]]
```

Out[89]:

URUN_KODU_FULL	1111436
MUSTERI_NO	
14285.0	1.0

In [90]:
```
eatTo_eat_df=user_eat_df[eatTo_eat]
```

In [91]:
```
eatTo_eat_df.head()
```

Out[91]:

URUN_KODU_FULL	1111436	1111467	1317417	1317460	1622109	1623127	1623131	1934435
MUSTERI_NO								
1236.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1237.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1238.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1239.0	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN
1240.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [92]:
```
user_eat_count=eatTo_eat_df.T.notnull().sum()
```

In [93]:
```
user_eat_count.head()
```

Out[93]:
```
MUSTERI_NO
1236.0    0
1237.0    0
1238.0    0
1239.0    1
1240.0    0
dtype: int64
```

In [94]:
```
user_eat_count=user_eat_count.reset_index()
```

In [95]:
```
user_eat_count.columns=["MUSTERI_NO","eattoeat"]
```

In [96]:
```
user_eat_count.head()
```

Out[96]:

	MUSTERI_NO	eattoeat
0	1236.0	0
1	1237.0	0
2	1238.0	0
3	1239.0	1
4	1240.0	0

In [97]:
```
user_eat_count.describe().T
```

Out[97]:

	count	mean	std	min	25%	50%	75%	max
MUSTERI_NO	15809.0	9140.000000	4563.809538	1236.0	5188.0	9140.0	13092.0	17044.0
eattoeat	15809.0	0.492378	0.697620	0.0	0.0	0.0	1.0	8.0

In [98]:
```
user_eat_count[user_eat_count["eattoeat"]>1]["MUSTERI_NO"].head()
```

Out[98]:
```
6     1242.0
8     1244.0
16    1252.0
22    1258.0
24    1260.0
Name: MUSTERI_NO, dtype: float64
```

In [99]:
```
users_same_eat=user_eat_count[user_eat_count["eattoeat"]>1]["MUSTERI_NO"]
```

In [100…
```
users_same_eat.head()
```

Out[100…
```
6     1242.0
8     1244.0
16    1252.0
22    1258.0
24    1260.0
Name: MUSTERI_NO, dtype: float64
```

In [101…
```
final=pd.concat([eatTo_eat_df[eatTo_eat_df.index.isin(users_same_eat)],random_user_df[eatTo_eat]])
```

In [102…
```
final.head()
```

Out[102…

URUN_KODU_FULL	1111436	1111467	1317417	1317460	1622109	1623127	1623131	1934435
MUSTERI_NO								
1242.0	NaN	NaN	1.0	NaN	NaN	1.0	NaN	NaN
1244.0	NaN	NaN	1.0	NaN	NaN	1.0	NaN	NaN
1252.0	NaN	NaN	NaN	1.0	NaN	1.0	NaN	1.0
1258.0	NaN	NaN	NaN	5.0	NaN	3.0	NaN	NaN
1260.0	NaN	NaN	3.0	NaN	NaN	2.0	NaN	NaN

In [103…
```
final.shape
```

Out[103…
```
(1205, 8)
```

In [104…
```
corr_df = final.T.corr().unstack().sort_values().drop_duplicates()
```

In [105…
```
corr_df = pd.DataFrame(corr_df, columns=["corr"])
```

In [106…]
```
corr_df .head()
```

Out[106…]

		corr
MUSTERI_NO	MUSTERI_NO	
13269.0	13035.0	-1.000000
4956.0	5570.0	-1.000000
4840.0	10589.0	-1.000000
10589.0	12932.0	-0.999887
	14128.0	-0.999877

In [107…]
```
corr_df.index.names = ['MUSTERI_NO_1', 'MUSTERI_NO_2']
corr_df = corr_df.reset_index()
```

In [108…]
```
random_user
```

Out[108…]
```
14285.0
```

In [109…]
```
corr_df.head()
```

Out[109…]

	MUSTERI_NO_1	MUSTERI_NO_2	corr
0	13269.0	13035.0	-1.000000
1	4956.0	5570.0	-1.000000
2	4840.0	10589.0	-1.000000
3	10589.0	12932.0	-0.999887
4	10589.0	14128.0	-0.999877

In [110…]
```
top_users = corr_df[(corr_df["MUSTERI_NO_1"] == random_user) & (corr_df["corr"] >= 0.1)][
    ["MUSTERI_NO_2", "corr"]].reset_index(drop=True)
```

In [111…]
```
top_users = top_users.sort_values(by='corr', ascending=False)
```

In [112…]
```
top_users.rename(columns={"MUSTERI_NO_2": "MUSTERI_NO"}, inplace=True)
```

In [113…]
```
top_users
```

Out[113…]

	MUSTERI_NO	corr
3	4494.0	0.870388
2	9349.0	0.577350
1	16492.0	0.174078
0	16492.0	0.174078

In [114…]
```
df_user=df_user.reset_index()
```

In [115…]
```
df_user.head()
```

Out[115…]

	MUSTERI_NO	URUN_KODU_FULL	SIPARIS_ADEDI
0	1236.0	1623129	1
1	1236.0	1623307	1
2	1237.0	1010463	1
3	1237.0	1317188	1
4	1237.0	1420433	1

In [116…
```
top_users_ratings = top_users.merge(df_user[["MUSTERI_NO", "URUN_KODU_FULL", "SIPARIS_ADEDI"]], on="MUSTERI_NO",h
```

In [117…
```
top_users_ratings["corr"].describe()
```

Out[117…
```
count    103.000000
mean       0.424967
std        0.273156
min        0.174078
25%        0.174078
50%        0.174078
75%        0.577350
max        0.870388
Name: corr, dtype: float64
```

In [118…
```
top_users_ratings['weighted_rating'] = top_users_ratings['corr'] * top_users_ratings['SIPARIS_ADEDI']
```

In [119…
```
top_users_ratings
```

Out[119…

	MUSTERI_NO	corr	URUN_KODU_FULL	SIPARIS_ADEDI	weighted_rating
0	4494.0	0.870388	1010410	1	0.870388
1	4494.0	0.870388	1111436	2	1.740777
2	4494.0	0.870388	1111467	1	0.870388
3	4494.0	0.870388	1317188	1	0.870388
4	4494.0	0.870388	1317456	1	0.870388
5	4494.0	0.870388	1420321	1	0.870388
6	4494.0	0.870388	1420349	1	0.870388
7	4494.0	0.870388	1623110	2	1.740777
8	4494.0	0.870388	1623112	1	0.870388
9	4494.0	0.870388	1623118	1	0.870388
10	4494.0	0.870388	1623124	1	0.870388
11	4494.0	0.870388	1623127	1	0.870388
12	4494.0	0.870388	1623305	1	0.870388
13	4494.0	0.870388	1624136	2	1.740777
14	4494.0	0.870388	1625183	1	0.870388
15	4494.0	0.870388	1934405	1	0.870388
16	4494.0	0.870388	1934435	3	2.611165
17	4494.0	0.870388	1934446	3	2.611165
18	9349.0	0.577350	1010410	1	0.577350
19	9349.0	0.577350	1010412	1	0.577350
20	9349.0	0.577350	1010447	1	0.577350
21	9349.0	0.577350	1111436	1	0.577350
22	9349.0	0.577350	1215504	1	0.577350
23	9349.0	0.577350	1215505	4	2.309401
24	9349.0	0.577350	1215509	1	0.577350
25	9349.0	0.577350	1317187	1	0.577350
26	9349.0	0.577350	1317188	2	1.154701
27	9349.0	0.577350	1317417	1	0.577350
28	9349.0	0.577350	1420391	1	0.577350
29	9349.0	0.577350	1622100	1	0.577350
30	9349.0	0.577350	1623112	7	4.041452
31	9349.0	0.577350	1623114	2	1.154701
32	9349.0	0.577350	1623115	1	0.577350
33	9349.0	0.577350	1623116	11	6.350853
34	9349.0	0.577350	1623118	1	0.577350
35	9349.0	0.577350	1623127	2	1.154701
36	9349.0	0.577350	1623128	1	0.577350

37	9349.0	0.577350	1623129	1	0.577350
38	9349.0	0.577350	1623130	2	1.154701
39	9349.0	0.577350	1623132	3	1.732051
40	9349.0	0.577350	1623312	1	0.577350
41	9349.0	0.577350	1624137	2	1.154701
42	9349.0	0.577350	1624143	1	0.577350
43	9349.0	0.577350	1624145	2	1.154701
44	9349.0	0.577350	1624147	1	0.577350
45	9349.0	0.577350	1624332	1	0.577350
46	9349.0	0.577350	1625165	1	0.577350
47	9349.0	0.577350	1832426	1	0.577350
48	9349.0	0.577350	1934435	2	1.154701
49	9349.0	0.577350	1934446	1	0.577350
50	9349.0	0.577350	1934494	4	2.309401
51	16492.0	0.174078	1010385	2	0.348155
52	16492.0	0.174078	1010447	6	1.044466
53	16492.0	0.174078	1010475	3	0.522233
54	16492.0	0.174078	1010503	1	0.174078
55	16492.0	0.174078	1111470	1	0.174078
56	16492.0	0.174078	1111476	2	0.348155
57	16492.0	0.174078	1215505	2	0.348155
58	16492.0	0.174078	1317188	2	0.348155
59	16492.0	0.174078	1317368	1	0.174078
60	16492.0	0.174078	1317456	2	0.348155
61	16492.0	0.174078	1317459	1	0.174078
62	16492.0	0.174078	1622100	1	0.174078
63	16492.0	0.174078	1622109	1	0.174078
64	16492.0	0.174078	1622282	1	0.174078
65	16492.0	0.174078	1622286	1	0.174078
66	16492.0	0.174078	1623112	1	0.174078
67	16492.0	0.174078	1623116	2	0.348155
68	16492.0	0.174078	1623118	1	0.174078
69	16492.0	0.174078	1623123	1	0.174078
70	16492.0	0.174078	1623127	3	0.522233
71	16492.0	0.174078	1623131	1	0.174078
72	16492.0	0.174078	1623319	1	0.174078
73	16492.0	0.174078	1626381	1	0.174078
74	16492.0	0.174078	1832430	1	0.174078
75	16492.0	0.174078	1934435	2	0.348155
76	16492.0	0.174078	1934446	1	0.174078
77	16492.0	0.174078	1010385	2	0.348155
78	16492.0	0.174078	1010447	6	1.044466
79	16492.0	0.174078	1010475	3	0.522233
80	16492.0	0.174078	1010503	1	0.174078
81	16492.0	0.174078	1111470	1	0.174078
82	16492.0	0.174078	1111476	2	0.348155
83	16492.0	0.174078	1215505	2	0.348155
84	16492.0	0.174078	1317188	2	0.348155
85	16492.0	0.174078	1317368	1	0.174078
86	16492.0	0.174078	1317456	2	0.348155
87	16492.0	0.174078	1317459	1	0.174078
88	16492.0	0.174078	1622100	1	0.174078
89	16492.0	0.174078	1622109	1	0.174078
90	16492.0	0.174078	1622282	1	0.174078
91	16492.0	0.174078	1622286	1	0.174078

92	16492.0	0.174078	1623112	1	0.174078
93	16492.0	0.174078	1623116	2	0.348155
94	16492.0	0.174078	1623118	1	0.174078
95	16492.0	0.174078	1623123	1	0.174078
96	16492.0	0.174078	1623127	3	0.522233
97	16492.0	0.174078	1623131	1	0.174078
98	16492.0	0.174078	1623319	1	0.174078
99	16492.0	0.174078	1626381	1	0.174078
100	16492.0	0.174078	1832430	1	0.174078
101	16492.0	0.174078	1934435	2	0.348155
102	16492.0	0.174078	1934446	1	0.174078

In [120…
```
top_users_ratings["MUSTERI_NO"].unique()
```

Out[120…
```
array([ 4494.,  9349., 16492.])
```

In [121…
```
top_users_ratings.groupby('URUN_KODU_FULL').agg({"weighted_rating": "mean"}).head()
```

Out[121…

URUN_KODU_FULL	weighted_rating
1010385	0.348155
1010410	0.723869
1010412	0.577350
1010447	0.888761
1010475	0.522233

In [122…
```
recommendation_df = top_users_ratings.groupby('URUN_KODU_FULL').agg({"weighted_rating": "mean"})
```

In [123…
```
recommendation_df = recommendation_df.reset_index()
```

In [124…
```
recommendation_df.head()
```

Out[124…

	URUN_KODU_FULL	weighted_rating
0	1010385	0.348155
1	1010410	0.723869
2	1010412	0.577350
3	1010447	0.888761
4	1010475	0.522233

In [125…
```
recommendation_df[recommendation_df["weighted_rating"] >1].sort_values(by="weighted_rating",ascending=False)
```

Out[125…

	URUN_KODU_FULL	weighted_rating
30	1623116	2.349055
57	1934494	2.309401
26	1623110	1.740777
43	1624136	1.740777
39	1623132	1.732051
27	1623112	1.314999
6	1111436	1.159063
28	1623114	1.154701
37	1623130	1.154701

44	1624137	1.154701
46	1624145	1.154701
55	1934435	1.115544
11	1215505	1.001904

#Belirlenen bir müşteriye göre önerilen yiyecekler yukarıdaki gibidir.

TIME SERIES

2022'nin ilk 3 ayının her bir gününde gelmesi beklenen sipariş sayıları zaman serileri yöntemiyle tahmin edilecektir.

In [126…
```
df=df.groupby("TARIH")[["SIPARIS_ADEDI"]].sum()
```

In [127…
```
df=df.reset_index()
```

In [128…
```
test=pd.DataFrame()
```

In [129…
```
c=[]
```

In [130…
```
a=pd.to_datetime("2022-01-01")
```

In [131…
```
c.append(a)
```

In [132…
```
for i in range(89):
    a=a+ timedelta(days=1)
    c.append(a)
```

In [133…
```
test["TARIH"]=c
```

In [134…
```
test.shape
```

Out[134… (90, 1)

In [135…
```
df = pd.concat([df, test], sort=False)
```

In [136…
```
df.head()
```

Out[136…

	TARIH	SIPARIS_ADEDI
0	2017-01-01	23.0
1	2017-01-02	20.0
2	2017-01-03	31.0
3	2017-01-04	40.0
4	2017-01-05	22.0

In [137…
```
df.tail()
```

Out[137…

	TARIH	SIPARIS_ADEDI
85	2022-03-27	NaN
86	2022-03-28	NaN
87	2022-03-29	NaN
88	2022-03-30	NaN
89	2022-03-31	NaN

In [138…

```
df[df["SIPARIS_ADEDI"].isnull()].shape
```

Out[138… (90, 2)

In [139…
```
def create_date_features(df):
    df['month'] = df.TARIH.dt.month
    df['day_of_month'] = df.TARIH.dt.day
    df['day_of_year'] = df.TARIH.dt.dayofyear
    df['year'] = df.TARIH.dt.year
    df["is_wknd"] = df.TARIH.dt.weekday // 4
    df['is_month_start'] = df.TARIH.dt.is_month_start.astype(int)
    df['is_month_end'] = df.TARIH.dt.is_month_end.astype(int)
    df["DAY_OF_WEEKDAY"]=df.TARIH.dt.dayofweek
    return df
```

In [140…
```
df = create_date_features(df)
```

In [141…
```
df.head()
```

Out[141…

	TARIH	SIPARIS_ADEDI	month	day_of_month	day_of_year	year	is_wknd	is_month_start	is_month_end	DAY_OF_WEEKDAY
0	2017-01-01	23.0	1	1	1	2017	1	1	0	6
1	2017-01-02	20.0	1	2	2	2017	0	0	0	0
2	2017-01-03	31.0	1	3	3	2017	0	0	0	1
3	2017-01-04	40.0	1	4	4	2017	0	0	0	2
4	2017-01-05	22.0	1	5	5	2017	0	0	0	3

In [142…
```
def new_sorting(df,col,old,new):
    maps={}
    n=len(old)
    for i in range(n):
        maps.setdefault(old[i],new[i])
    df[f"new_"+col] = df[col].map(maps).astype(int)
```

In [143…
```
new_sorting(df,"month",[1,2,3,4,5,6,7,8,9,10,11,12],[1,1,2,2,2,3,3,3,4,4,4,1])
```

In [144…
```
df.tail()
```

Out[144…

	TARIH	SIPARIS_ADEDI	month	day_of_month	day_of_year	year	is_wknd	is_month_start	is_month_end	DAY_OF_WEEKDAY	new_month
85	2022-03-27	NaN	3	27	86	2022	1	0	0	6	2
86	2022-03-28	NaN	3	28	87	2022	0	0	0	0	2
87	2022-03-29	NaN	3	29	88	2022	0	0	0	1	2
88	2022-03-30	NaN	3	30	89	2022	0	0	0	2	2
89	2022-03-31	NaN	3	31	90	2022	0	0	1	3	2

In [145…
```
def random_noise(dataframe):
    return np.random.normal(scale=1.6, size=(len(dataframe),))
```

In [146…
```
def lag_features(dataframe, lags):
    for lag in lags:
        dataframe['sales_lag_' + str(lag)] = dataframe["SIPARIS_ADEDI"].transform(
            lambda x: x.shift(lag)) + random_noise(dataframe)
    return dataframe
```

In [147…
```
df = lag_features(df, [91, 98, 105, 112, 119, 126, 182, 364])
```

In [148…
```
def roll_mean_features(dataframe, windows):
    for window in windows:
```

```
        dataframe['sales_roll_mean_' + str(window)] = dataframe["SIPARIS_ADEDI"]. \
                                                      transform(
            lambda x: x.shift(1).rolling(window=window, min_periods=10, win_type="triang").mean()) + random_noise
            dataframe)
    return dataframe
```

In [149…
```
df = roll_mean_features(df, [365, 540])
```

In [150…
```
def ewm_features(dataframe, alphas, lags):
    for alpha in alphas:
        for lag in lags:
            dataframe['sales_ewm_alpha_' + str(alpha).replace(".", "") + "_lag_" + str(lag)] = \
                dataframe["SIPARIS_ADEDI"].transform(lambda x: x.shift(lag).ewm(alpha=alpha).mean())
    return dataframe
```

In [151…
```
alphas = [0.95, 0.9, 0.8, 0.7, 0.5]
lags = [91, 98, 105, 112, 180, 270, 365]
```

In [152…
```
df = ewm_features(df, alphas, lags)
```

In [153…
```
df = pd.get_dummies(df, columns=["DAY_OF_WEEKDAY", 'month',"new_month"])
```

In [154…
```
df["SIPARIS_ADEDI"] = np.log1p(df["SIPARIS_ADEDI"].values)
```

In [155…
```
def smape(preds, target):
    n = len(preds)
    masked_arr = ~((preds == 0) & (target == 0))
    preds, target = preds[masked_arr], target[masked_arr]
    num = np.abs(preds - target)
    denom = np.abs(preds) + np.abs(target)
    smape_val = (200 * np.sum(num / denom)) / n
    return smape_val

def lgbm_smape(preds, train_data):
    labels = train_data.get_label()
    smape_val = smape(np.expm1(preds), np.expm1(labels))
    return 'SMAPE', smape_val, False
```

Modelin hata oranının kontrol edilmesi amacıyla bağımlı değişkeni olan geçmişteki değerlerin bağımlı değişkeni silinerek tahmin yapılır ve gerçek değerler ile arasındaki hata oranına bakılır.

In [156…
```
train = df.loc[(df["TARIH"] < "2019-01-01"), :]
```

In [157…
```
val = df.loc[(df["TARIH"] >= "2019-01-01") & (df["TARIH"] < "2019-04-01"), :]
```

In [158…
```
import lightgbm as lgb
```

In [159…
```
cols = [col for col in train.columns if col not in ["TARIH","SIPARIS_ADEDI","year"]]

Y_train = train["SIPARIS_ADEDI"]
X_train = train[cols]
Y_val = val["SIPARIS_ADEDI"]
X_val = val[cols]
lgb_params = {'metric': {'mae'},
              'num_leaves': 10,
              'learning_rate': 0.02,
              'feature_fraction': 0.8,
              'max_depth': 5,
              'verbose': 0,
              'num_boost_round':500,
              'early_stopping_rounds': 200,
              'nthread': -1}

lgbtrain = lgb.Dataset(data=X_train, label=Y_train, feature_name=cols)
lgbval = lgb.Dataset(data=X_val, label=Y_val, reference=lgbtrain, feature_name=cols)
```

In [160…
```
model = lgb.train(lgb_params, lgbtrain,
                  valid_sets=[lgbtrain, lgbval],
                  num_boost_round=lgb_params['num_boost_round'],
                  early_stopping_rounds=lgb_params['early_stopping_rounds'],
                  feval=lgbm_smape,
                  verbose_eval=100)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Early stopping, best iteration is:
[286]   training's l1: 0.132386 training's SMAPE: 13.376        valid_1's l1: 0.145686  valid_1's SMAPE: 14.5648
```

In [161…

```
y_pred_val = model.predict(X_val, num_iteration=model.best_iteration)
```

In [162…

```
smape(np.expm1(y_pred_val), np.expm1(Y_val))
```

Out[162… 14.56475363884621

smape:Hata oranını bulan bir fonksiyon, başka hata metriği yöntemleri de kullanılabilir.Modelin tahmin ettiği değerler ile gerçek değerler arasındaki farkı yani hatayı %'lik oranda verir.2022 ilk 3 ayın tahmini

In [163…

```
train = df.loc[(df["TARIH"] <= "2020-03-31"), :]
```

In [164…

```
train.shape
```

Out[164… (1171, 76)

In [165…

```
Y_train = train["SIPARIS_ADEDI"]
```

In [166…

```
X_train = train[cols]
```

In [167…

```
test = df.loc[df.SIPARIS_ADEDI.isna()]
```

In [168…

```
X_test = test[cols]
```

In [169…

```
test.shape
```

Out[169… (90, 76)

In [170…

```
lgb_params = {'metric': {'mae'},
              'num_leaves': 10,
              'learning_rate': 0.02,
              'feature_fraction': 0.8,
              'max_depth': 5,
              'verbose': 0,
              'nthread': -1,
              "num_boost_round": model.best_iteration}
```

In [171…

```
lgbtrain_all = lgb.Dataset(data=X_train, label=Y_train, feature_name=cols)
```

In [172…

```
model = lgb.train(lgb_params, lgbtrain_all, num_boost_round=model.best_iteration)
```

```
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001379 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

In [173…

```
test_preds = model.predict(X_test, num_iteration=model.best_iteration)
```

In [174…

```
test_preds #logaritmik dönüşüm yapılmış hali
```

Out[174…

```
array([5.38991302, 5.393503  , 5.01516168, 5.16990232, 5.18719946,
       5.13169218, 5.28370222, 5.29557632, 5.34404383, 4.8695588 ,
       4.97790452, 5.15291467, 5.14841277, 5.17721479, 5.29398817,
       5.26041541, 4.86063529, 5.00775318, 4.93250271, 5.04271385,
       5.24481817, 5.28268209, 5.25652891, 4.89100149, 4.92324233,
       4.87366885, 4.87917392, 4.99890536, 5.23647681, 5.17792168,
       4.76723105, 4.96088769, 4.97106447, 4.8962049 , 5.21858768,
       5.12501677, 5.26669161, 4.75171761, 4.92209714, 5.10932412,
       4.88921354, 5.19007732, 5.21445518, 5.38180263, 4.86594764,
       4.99337714, 4.97732841, 5.02178033, 5.12435318, 5.22539213,
       5.33485724, 4.77607503, 5.04166869, 5.05907454, 5.05943088,
       5.33541814, 5.32064944, 5.34317371, 4.89722676, 5.11014781,
       5.10466087, 5.09252222, 5.26970817, 5.27932526, 5.43983227,
       4.8849449 , 5.11945563, 5.10817718, 5.12260086, 5.38912689,
       5.35581705, 5.45202498, 4.88546841, 5.08710731, 4.98202982,
       4.90846712, 5.28639313, 5.21876655, 5.29065962, 4.84399408,
       5.06373604, 4.90153112, 4.98768189, 5.20302855, 5.23595003,
       5.25880446, 4.86226532, 4.84673921, 4.91558186, 4.89356103])
```

In [175…

```
recommendation=pd.DataFrame()
```

In [176…

```
recommendation["SIPARIS_ADEDI"] = np.expm1(test_preds)
```

In [177…

```
#2022'nin ilk 90 gününde beklenen sipariş sayısı
recommendation["SIPARIS_ADEDI"]
```

Out[177…

```
0     218.184320
1     218.972602
2     149.680496
3     174.897655
4     177.966647
5     168.303367
6     196.098227
7     198.452542
8     208.357608
9     129.263432
10    144.169862
11    171.934804
12    171.158019
13    176.188615
14    198.136032
15    191.561467
16    128.106196
17    148.568306
```

```
18    137.726269
19    153.889793
20    188.581340
21    195.897264
22    190.814529
23    132.086793
24    136.447542
25    129.799923
26    130.521972
27    147.250789
28    187.006552
29    176.313913
30    116.593181
31    141.720432
32    143.180282
33    132.781103
34    183.673182
35    167.176961
36    192.773821
37    114.782984
38    136.290229
39    164.558420
40    131.849053
41    178.482430
42    182.911595
43    216.413840
44    128.793879
45    146.433487
46    144.086252
47    150.681106
48    167.065398
49    184.934067
50    206.443133
51    117.637785
52    153.727993
53    156.444739
54    156.500853
55    206.559520
56    203.516660
57    208.175520
58    132.917878
59    164.694844
60    163.788177
61    161.799963
62    193.359234
63    195.237422
64    229.403534
65    131.283176
66    166.244302
67    164.368643
68    166.771151
69    218.012081
70    210.836988
71    232.229975
72    131.352446
73    160.920798
74    144.769968
75    134.431654
76    196.629315
77    183.706217
78    197.474299
79    125.975491
80    157.180382
81    133.495551
82    145.596203
83    180.822066
84    186.907538
85    191.251510
86    128.316814
87    126.324533
88    135.398651
89    132.427869
Name: SIPARIS_ADEDI, dtype: float64
```

CLTV MÜŞTERİ YAŞAM BOYU DEĞERİ

In [178…

```
df=data.merge(data2, on='ADISYON_NO', how='left')
```

In [179…
```
df.head()
```

Out[179…

	ADISYON_NO	SIPARIS_SIRA_NO	SIPARIS_ADEDI	URUN_ADI	URUN_KODU	ANA_GRUP	ANA_GRUP_KODU	ALT_GRUP	ALT_GRUP_KODU
0	56729	1	1	ÜRÜN_150	150	PİZZALAR	16	2-KİŞİLİK PİZZALAR	24
1	56730	1	1	ÜRÜN_313	313	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
2	56730	2	1	ÜRÜN_319	319	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
3	56731	1	1	ÜRÜN_125	125	PİZZALAR	16	1-KİŞİLİK PİZZALAR	23
4	56731	2	1	ÜRÜN_188	188	İÇECEKLER	13	SOĞUK İÇECEKLER	17

In [180…
```
dff=df[df["ADISYON_TIPI"]=="P"].groupby(["MUSTERI_NO","TARIH","ADISYON_NO"])[["ADISYON_TUTARI"]].max()
```

In [181…
```
dff=dff.reset_index()
```

In [182…
```
dff.head()
```

Out[182…

	MUSTERI_NO	TARIH	ADISYON_NO	ADISYON_TUTARI
0	1236.0	2019-07-19	101258	59.0
1	1237.0	2019-07-16	101094	45.8
2	1238.0	2019-07-16	101085	69.0
3	1239.0	2019-07-16	101069	28.0
4	1239.0	2019-10-15	106229	31.5

In [183…
```
dff["TARIH"].max()
```

Out[183… `Timestamp('2021-06-30 00:00:00')`

In [184…
```
today_date=dff["TARIH"].max()+ timedelta(days=2)
```

In [185…
```
cltv_df = dff.groupby("MUSTERI_NO").agg({'TARIH': [lambda date: (date.max() - date.min()).days,
                                                   lambda date: (today_date - date.min()).days],
                                         'ADISYON_NO': lambda num: num.nunique(),
                                         'ADISYON_TUTARI': lambda TotalPrice: TotalPrice.sum()})
```

In [186…
```
cltv_df.head()
```

Out[186…

	TARIH		ADISYON_NO	ADISYON_TUTARI
	<lambda_0>	<lambda_1>	<lambda>	<lambda>
MUSTERI_NO				
1236.0	0	714	1	59.0
1237.0	0	717	1	45.8
1238.0	0	717	1	69.0
1239.0	570	717	4	151.5
1240.0	218	717	4	316.0

In [187…
```
cltv_df.columns = cltv_df.columns.droplevel(0)
cltv_df.columns = ['recency', 'T', 'frequency', 'monetary']
```

In [188…
```
cltv_df.head()
```

Out[188…

	recency	T	frequency	monetary

MUSTERI_NO				
1236.0	0	714	1	59.0
1237.0	0	717	1	45.8
1238.0	0	717	1	69.0
1239.0	570	717	4	151.5
1240.0	218	717	4	316.0

In [189…
```
cltv_df["monetary"] = cltv_df["monetary"] / cltv_df["frequency"]
cltv_df = cltv_df[cltv_df["monetary"] > 0]
```

In [190…
```
cltv_df = cltv_df[(cltv_df['frequency'] > 1)]
```

In [191…
```
#haftalık olarak değerlendirilmesini istiyoruz
cltv_df["recency"] = cltv_df["recency"] / 7
cltv_df["T"] = cltv_df["T"] / 7
```

In [192…
```
bgf = BetaGeoFitter(penalizer_coef=0.001)
```

In [193…
```
bgf.fit(cltv_df['frequency'],
        cltv_df['recency'],
        cltv_df['T'])
```

Out[193… `<lifetimes.BetaGeoFitter: fitted with 7758 subjects, a: 0.97, alpha: 7.46, b: 4.83, r: 1.18>`

In [194…
```
ggf = GammaGammaFitter(penalizer_coef=0.01)
ggf.fit(cltv_df['frequency'], cltv_df['monetary'])
```

Out[194… `<lifetimes.GammaGammaFitter: fitted with 7758 subjects, p: 4.44, q: 0.79, v: 4.32>`

In [195…
```
ggf.conditional_expected_average_profit(cltv_df['frequency'],
                                        cltv_df['monetary']).head(10)
```

Out[195…
```
MUSTERI_NO
1239.0     39.430688
1240.0     81.057033
1241.0     80.944542
1242.0     54.489645
1244.0     36.118158
1247.0    100.584439
1252.0     40.176446
1254.0     93.610975
1255.0     58.218991
1258.0     95.934984
dtype: float64
```

In [196…
```
ggf.conditional_expected_average_profit(cltv_df['frequency'],
                                        cltv_df['monetary']).sort_values(ascending=False).head(10)
```

Out[196…
```
MUSTERI_NO
13234.0    586.898715
1587.0     279.391521
14985.0    249.277424
14407.0    240.638699
10022.0    222.521647
14165.0    221.855603
12024.0    217.398235
5490.0     203.513788
12317.0    202.187900
13598.0    200.281695
dtype: float64
```

In [197…

```
cltv_df["expected_average_profit"] = ggf.conditional_expected_average_profit(cltv_df['frequency'],
                                                                             cltv_df['monetary'])
```

In [198…
```
cltv_df.sort_values("expected_average_profit", ascending=False).head(20)
```

Out[198…

MUSTERI_NO	recency	T	frequency	monetary	expected_average_profit
13234.0	2.142857	30.571429	2	570.600000	586.898715
1587.0	13.285714	110.000000	2	270.500000	279.391521
14985.0	7.000000	39.428571	3	243.833333	249.277424
14407.0	12.571429	42.142857	3	235.333333	240.638699
10022.0	0.857143	121.285714	2	215.000000	222.521647
14165.0	19.000000	39.000000	2	214.350000	221.855603
12024.0	5.714286	18.714286	2	210.000000	217.398235
5490.0	12.428571	170.571429	2	196.450000	203.513788
12317.0	8.142857	21.285714	3	197.500000	202.187900
13598.0	31.571429	34.285714	7	198.285714	200.281695
2110.0	3.285714	7.000000	2	187.500000	194.342880
11429.0	6.142857	12.857143	2	181.450000	188.143552
11775.0	14.714286	16.571429	4	184.250000	187.590110
13231.0	3.285714	30.571429	2	180.000000	186.657762
15334.0	2.285714	65.285714	2	176.750000	183.327544
9786.0	3.857143	4.714286	2	174.000000	180.509668
1511.0	94.857143	107.857143	2	173.000000	179.484985
15485.0	46.571429	69.428571	3	170.000000	174.239082
15840.0	65.714286	78.428571	2	166.500000	172.824550
14526.0	4.571429	8.142857	2	166.000000	172.312208

In [199…
```
#müşterilerin 6 aylık beklenen ortalama kârları
cltv = ggf.customer_lifetime_value(bgf,
                                   cltv_df['frequency'],
                                   cltv_df['recency'],
                                   cltv_df['T'],
                                   cltv_df['monetary'],
                                   time=6,  # 6 aylık
                                   freq="W",  # T'nin frekans bilgisi.
                                   discount_rate=0.01)
```

In [200…
```
cltv = cltv.reset_index()
```

In [201…
```
cltv.sort_values(by="clv", ascending=False).head(50)
```

Out[201…

	MUSTERI_NO	clv
6390	14128.0	15850.624495
5925	12932.0	9801.251825
6440	14222.0	1164.950759
5477	11487.0	940.002294
2176	5225.0	874.664739
7647	16780.0	849.895245
1427	3942.0	833.194284
6531	14431.0	806.545697
6101	13436.0	794.760330
6174	13598.0	712.915506
5283	10589.0	712.200858
679	2571.0	710.170286
6905	15247.0	684.240823
3104	6848.0	681.543958
6663	14734.0	673.663646

4976	9786.0	671.286327
406	2059.0	668.752424
7570	16626.0	667.903128
3180	6983.0	664.697001
5566	11775.0	662.215839
5644	12035.0	659.149399
5613	11922.0	655.965981
5368	11005.0	654.046818
5239	10187.0	644.896334
5301	10688.0	637.474322
5919	12902.0	616.216037
5254	10408.0	613.356622
5480	11493.0	612.910421
5285	10600.0	604.474292
5493	11527.0	591.288748
5676	12146.0	574.072064
5333	10843.0	565.116864
5435	11296.0	563.804641
410	2066.0	562.502247
5974	13099.0	561.101429
5638	12020.0	548.579517
6631	14674.0	547.585464
5267	10508.0	542.190247
433	2110.0	539.836354
5698	12201.0	537.846052
5553	11742.0	532.664564
5257	10427.0	532.408119
4758	9491.0	514.622496
7218	15918.0	507.827295
5245	10325.0	507.423240
5459	11432.0	501.550682
5703	12220.0	496.808409
7149	15757.0	496.338978
5286	10603.0	495.226047
28	1298.0	491.272825

In [202…
```
cltv_final = cltv_df.merge(cltv, on="MUSTERI_NO", how="left")
```

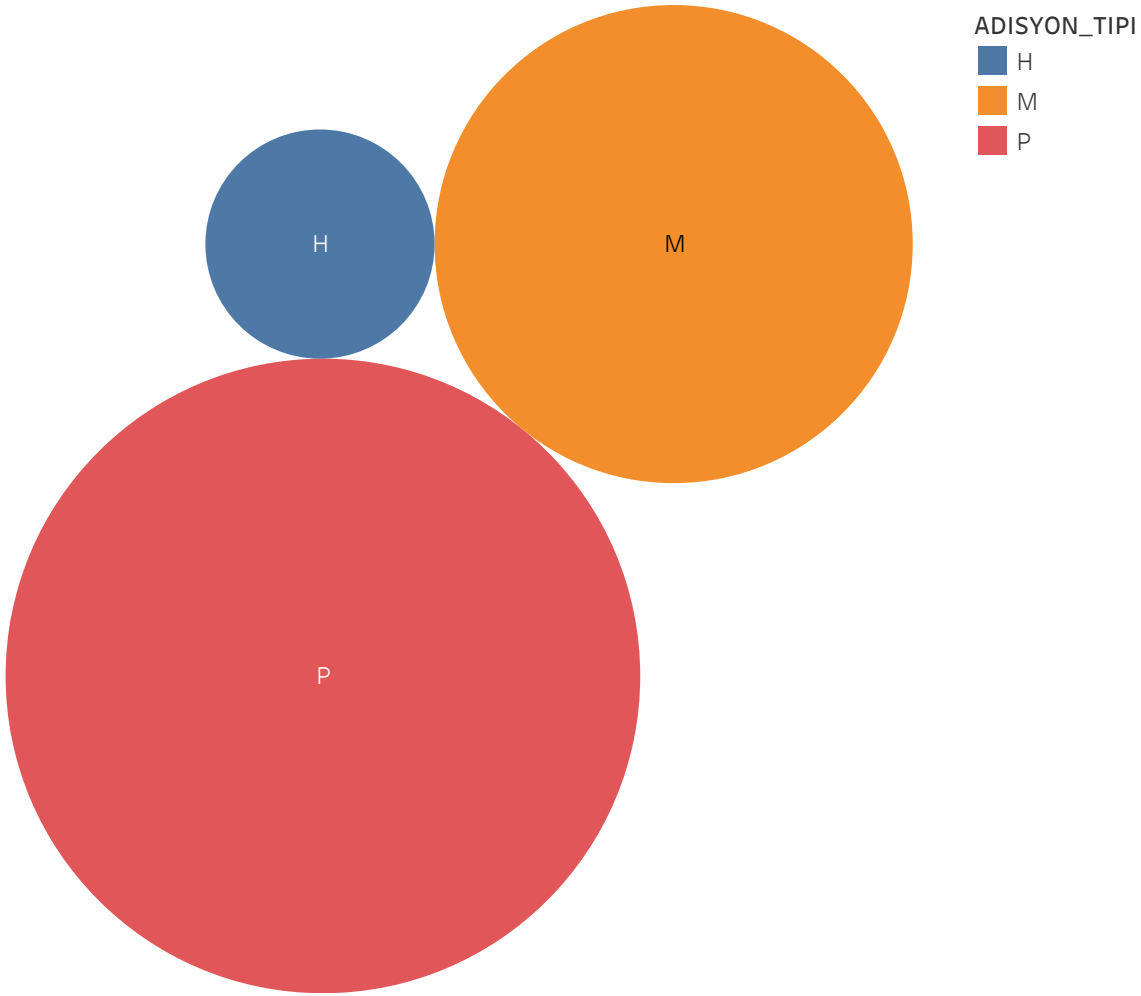
In [203…
```
cltv_final.sort_values(by="clv", ascending=False).head(10)
```

Out[203…

	MUSTERI_NO	recency	T	frequency	monetary	expected_average_profit	clv
6390	14128.0	35.142857	35.428571	906	54.957395	54.965090	15850.624495
5925	12932.0	27.714286	28.000000	348	54.428161	54.448121	9801.251825
6440	14222.0	38.142857	39.857143	46	63.152174	63.312462	1164.950759
5477	11487.0	13.000000	13.714286	11	99.590909	100.423740	940.002294
2176	5225.0	166.142857	166.571429	78	82.541026	82.647497	874.664739
7647	16780.0	2.714286	3.000000	4	123.500000	126.099521	849.895245
1427	3942.0	147.857143	148.571429	160	34.867500	34.905030	833.194284
6531	14431.0	41.142857	42.571429	33	60.603030	60.822828	806.545697
6101	13436.0	31.428571	32.285714	34	47.323529	47.518011	794.760330
6174	13598.0	31.571429	34.285714	7	198.285714	200.281695	712.915506

Müşterilerin yaşam boyu değerlerine yani şirkete olan getirilerine göre sıralamaları yukarıdaki gibidir.

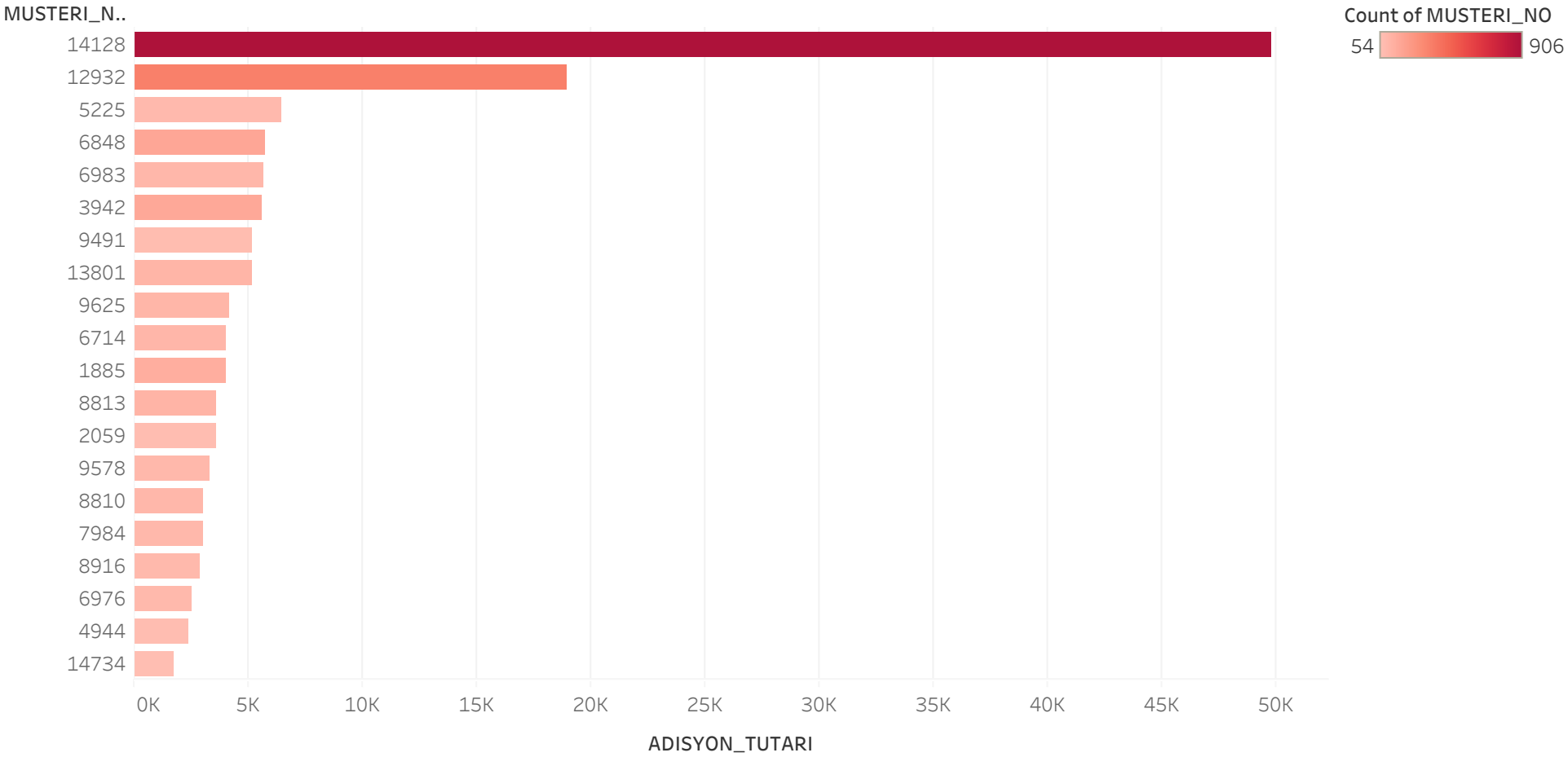
ADİSYON TİPLERİNE GÖRE TOPLAM ADİSYON TUTARI



ADISYON_TIPI
H
M
P

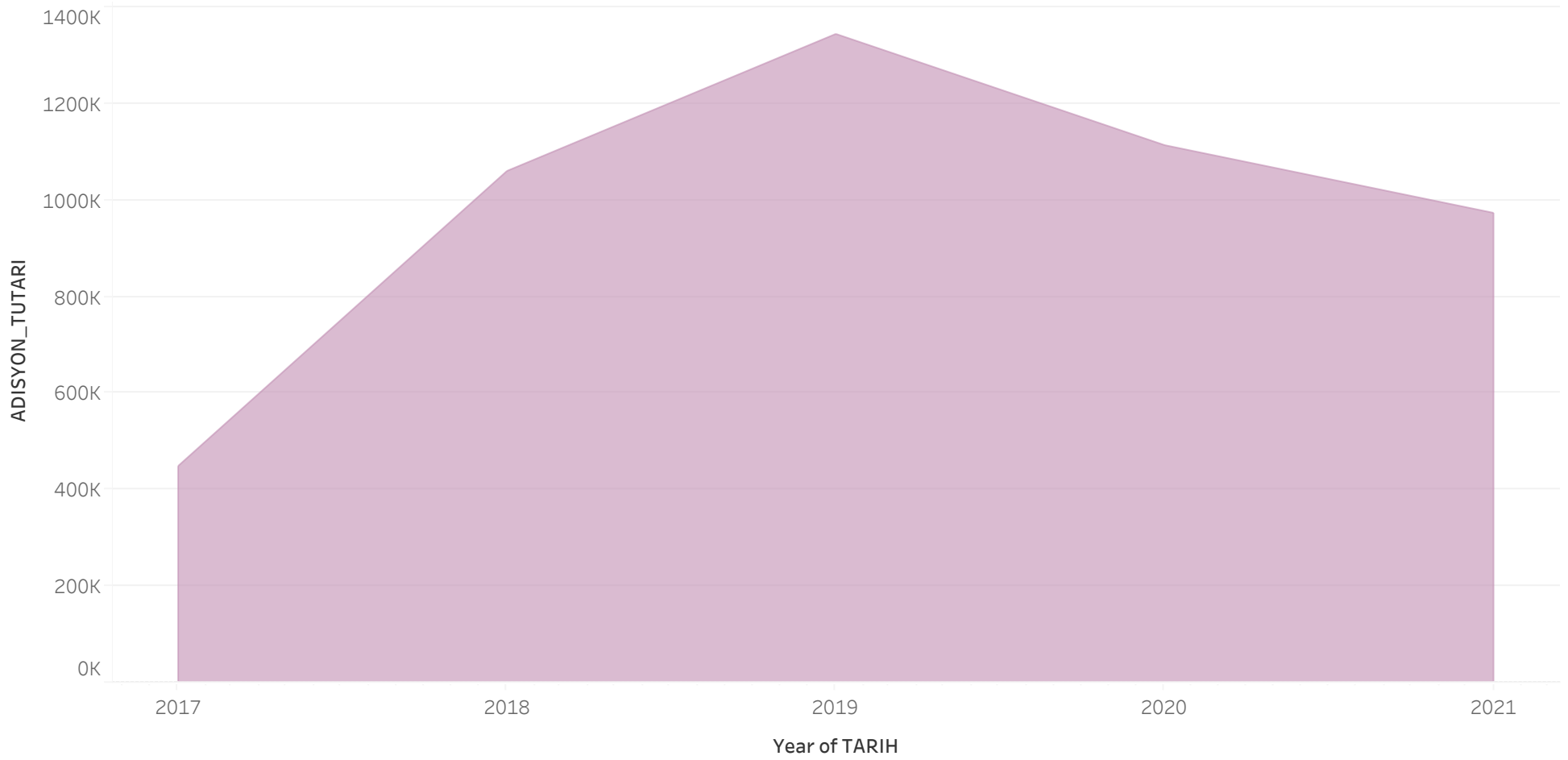
ADISYON_TIPI. Color shows details about ADISYON_TIPI. Size shows sum of ADISYON_TUTARI. The marks are labeled by ADISYON_TIPI. The data is filtered on Action (MUSTERI_NO), which keeps 15.810 members.

EN ÇOK GETİRİ SAĞLAYAN MÜŞTERİLER



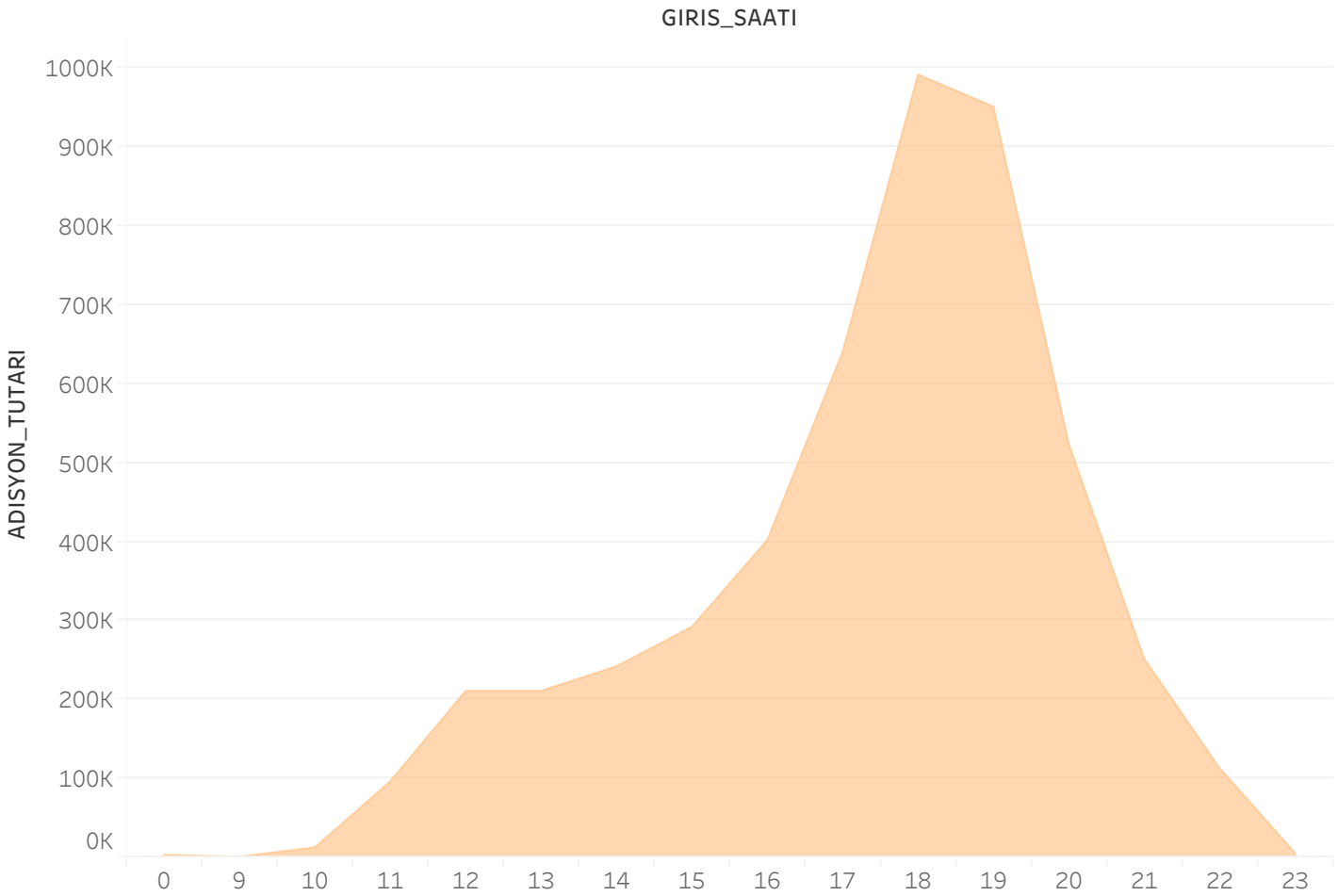
Sum of ADISYON_TUTARI for each MUSTERI_NO. Color shows count of MUSTERI_NO. The view is filtered on MUSTERI_NO, which keeps 20 of 15.810 members.

YILLARA GÖRE TOPLAM ADİSYON TUTARI



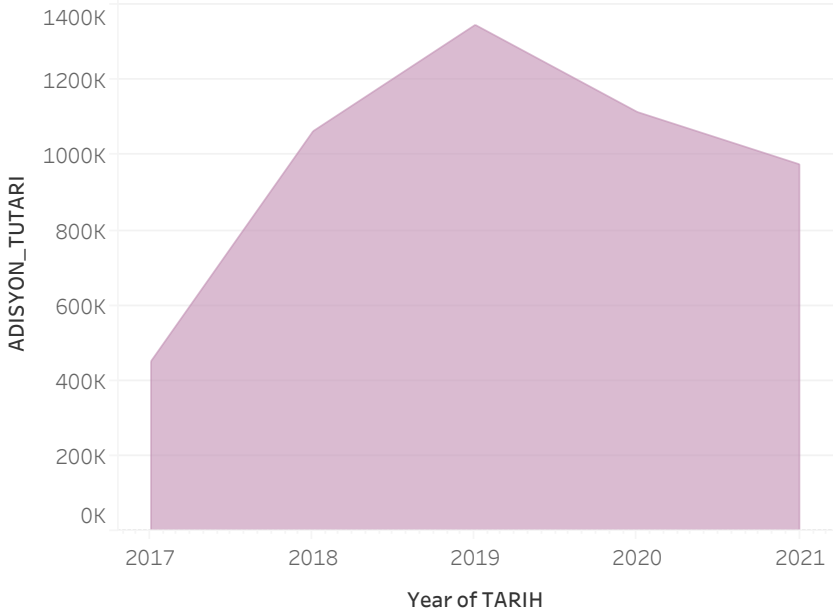
The plot of sum of ADISYON_TUTARI for TARIH Year. The data is filtered on Action (MUSTERI_NO), which keeps 15.810 members.

SİPARİŞLERİN VERİLİŞ SAATİNE GÖRE TOPLAM ADİSYON TUTARI

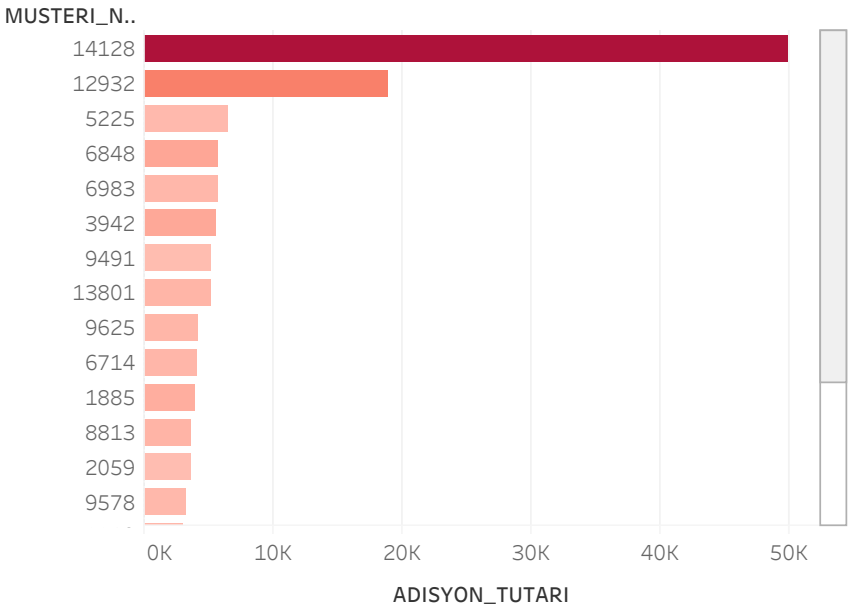


Sum of ADISYON_TUTARI for each GIRIS_SAATI Hour. The data is filtered on Action (MUSTERI_NO), which keeps 15.810 members.

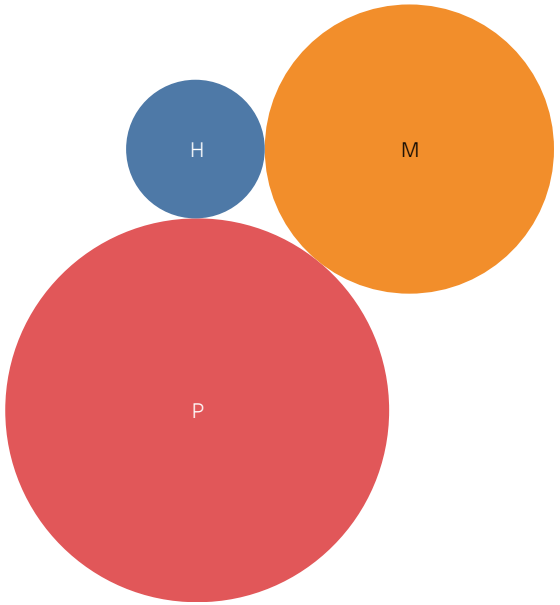
YILLARA GÖRE TOPLAM ADİSYON TUTARI



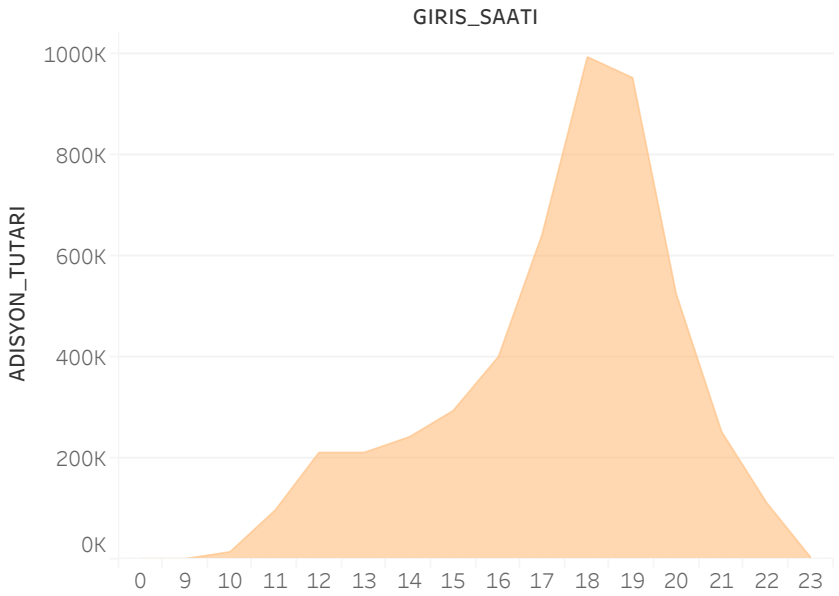
EN ÇOK GETİRİ SAĞLAYAN MÜŞTERİLER



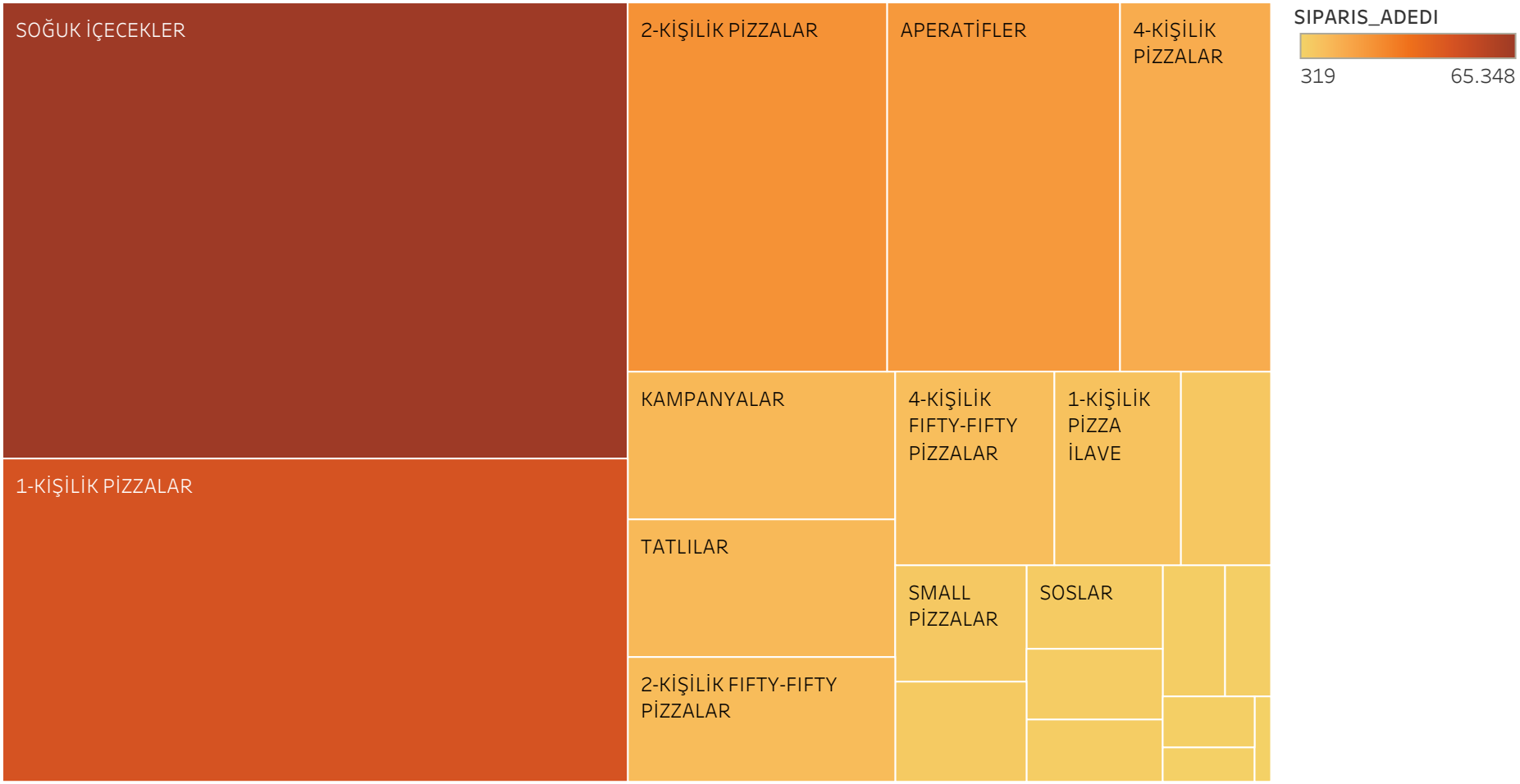
ADİSYON TİPLERİNE GÖRE TOPLAM ADİSYON TUTARI



SİPARİŞLERİN VERİLİŞ SAATİNE GÖRE TOPLAM ADİSYON TUTARI



SİPARİŞ SAYILARINA GÖRE ÜRÜN GRUPLARI



ALT_GRUP. Color shows sum of SIPARIS_ADEDI. Size shows sum of SIPARIS_ADEDI. The marks are labeled by ALT_GRUP.