

## ▼ PANDAS RECAP

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

## ▼ PANDAS BASICS

### ▼ Pandas Series

```
pandas.Series(data=None, index=None, ...)
```

Tek boyutlu veri yapılarıdır

```
labels = ['a', 'b', 'c']
my_list = [10, 20, 30]
array_ = np.array([10, 20, 30])
d      = {'a': 10, 'b': 20, 'c': 30}
```

```
pd.Series(my_list) # pd.Series(array_) # pd.Series(d)
```

```
0    10
1    20
2    30
dtype: int64
```

```
pd.Series(data = array_, index=labels)
```

```
a    10
b    20
c    30
dtype: int32
```

```
pd.Series(data=["element1","element2"])
```

```
0    element1
1    element2
dtype: object
```

```
pd.Series(data="element", index= range(5))
```

```
0    element
1    element
2    element
3    element
```

```

4    element
dtype: object

ser = pd.Series(["ahmet", 5 , True, 5.8])
ser.dtype

dtype('O')

```

## ▼ Pandas DataFrames

pandas.DataFrame() , series.to\_frame()

Tablo şeklinde çok sütunlu veri yapıları

```

df1 = pd.DataFrame(ser)    # pd.DataFrame(my_list) #  pd.DataFrame(array_) # pd.DataFrame(d
type(df1)

```

pandas.core.frame.DataFrame

```

df2 = ser.to_frame()
type(df2)

```

pandas.core.frame.DataFrame

```

list1 = [1, 3, "a", 7, 9]
array1 = np.arange(1,24,2).reshape(3,4)
dict1 = {"var1": [1,2,2], "var2": [3,4], "var3": [5,6]}

```

pd.DataFrame(list1)

	0
0	1
1	3
2	a
3	7
4	9

pd.DataFrame(array1)

	0	1	2	3
0	1	3	5	7
1	9	11	13	15
2	17	19	21	23

```
pd.DataFrame(dict1)
```

	<b>var1</b>	<b>var2</b>	<b>var3</b>
<b>0</b>	1.0	3	5
<b>1</b>	2.2	4	6

## ▼ Some attributes of Pandas DataFrame

```
df = pd.DataFrame({"var1": [1, "a", 4.5, 5, 6, 7, True], "var2": [1, 2.5, 2, 2.6, 10, 4.8, 7], "var3": [2.5, 3, 4, 5, 6, 7, 8], "var4": [21, 22, 23, 24, 25, 26, 27], "var5": [True, False, False, True, True, False, True], "var6": ["John", "Jack", "Melinda", "Sandra", "Leonardo", "Lena", "Arthur"]}, index = ["a", "b", 2, "d", 5.5, "6", True])
```

```
df
```

	<b>var1</b>	<b>var2</b>	<b>var3</b>	<b>var4</b>	<b>var5</b>	<b>var6</b>
<b>a</b>	1	1.0	2.5	21	True	John
<b>b</b>	a	2.5	3.2	22	False	Jack
<b>2</b>	4.5	2.0	4.8	23	False	Melinda
<b>d</b>	5	2.6	5.7	24	True	Sandra
<b>5.5</b>	6	10.0	7.6	25	True	Leonardo
<b>6</b>	7	4.8	8.3	26	False	Lena
<b>True</b>	True	7.0	9.5	27	True	Arthur

```
df.info()
# df.head()
# df.tail()
# df.sample(2)
# df.index
# df.columns
# df["var1"]
# df.columns = ["new1", "new2", "new3", "new4", "new5", "new6"]
# df.index = ["a", "b", "c", "d", "e", "f", "g"]
# df.rename(columns = {"new1": "aaa", "new2": "bbb"})
# df.rename(index = {1: "a", 2: "b"}, inplace=True)
# df.shape
# df.size
# df.ndim
# df.values
# df.dtypes
# df["var1"].value_counts()
# df["var1"].value_counts(normalize=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, a to True
```

```
Data columns (total 6 columns):
 #   Column Non-Null Count Dtype  
 --- 
 0   var1    7 non-null   object  
 1   var2    7 non-null   float64 
 2   var3    7 non-null   float64 
 3   var4    7 non-null   int64   
 4   var5    7 non-null   bool    
 5   var6    7 non-null   object  
dtypes: bool(1), float64(2), int64(1), object(2)
memory usage: 343.0+ bytes
```

## ▼ Indexing, Slicing & Selection

```
from numpy.random import randn
df = pd.DataFrame(randn(5, 4), index = 'A B C D E'.split(), columns = 'W X Y Z'.split())
df
```

	W	X	Y	Z
A	0.337080	-0.950286	-0.202146	-1.533382
B	0.557993	-0.060379	-0.667551	0.015456
C	-0.434050	-0.222327	-2.125566	-2.501308
D	0.482627	0.075776	2.684830	0.418298
E	-0.111065	-0.582703	0.626998	0.401063

```
# df["A"]                      # HATA  # .loc kullanmadan indexlerde bu şekilde çıktı alamıyor
# df["Y"]                      # dtype: Series    # Also: df.Y
# df[["Y"]]]                   # dtype: DataFrame
#df[["A", "C"]]]               # HATA  # .loc kullanmadan indexlerde bu şekilde çıktı alamıyoruz
df[["X", "Y"]]]                # dtype: DataFrame
#df[1:3][['W', 'Y']]          # dtype: DataFrame # Dikkat!: D dahil değil
# df["B":"D"][['W', 'Y']]      # dtype: DataFrame # Dikkat!: "D" dahil      # also: df[['W', 'Y']]
```

## ▼ Creating a New Column

```
df["new1"] = df["W"] * df["X"]
df["new2"] = np.arange(5)
df
```

W	X	Y	Z	new1	new2
---	---	---	---	------	------

## ▼ Removing Columns

```
# [ ] ↓ 33 cells hidden
```

```
df.drop("new2", axis=1) # inplace = True, / axis=0 default
```

	W	X	Y	Z	new1
A	-0.085447	0.889068	1.322141	-1.054706	-0.075968
B	-0.673976	-1.695707	0.317709	-0.293604	1.142866
C	-1.057357	0.245693	-0.282496	1.358318	-0.259785
D	-0.692295	-0.234705	0.593594	-0.269769	0.162485
E	0.249910	-0.020141	0.266774	0.148755	-0.005033

```
df.drop(["new1","new2"], axis=1) # inplace = True / axis=0 default
```

	W	X	Y	Z
A	-0.085447	0.889068	1.322141	-1.054706
B	-0.673976	-1.695707	0.317709	-0.293604
C	-1.057357	0.245693	-0.282496	1.358318
D	-0.692295	-0.234705	0.593594	-0.269769
E	0.249910	-0.020141	0.266774	0.148755

## ▶ Removing Rows

```
[ ] ↓ 33 cells hidden
```

## ▶ Some Useful Functions

```
[ ] ↓ 36 cells hidden
```

## ▶ Apply

```
[ ] ↓ 32 cells hidden
```

## ▶ stack() vs unstack()

```
[ ] ↓ 10 cells hidden
```

## ▼ ÖZET / CHEAT SHEET

```

# PANDAS SERILER : Tek boyutlu veri yapıları
    # Seri oluşturma
        # pd.Series(data = ... ) -- data = list, array, dictionary, string vs olabilir
# PANDAS DATAFRAME : Tablo şeklinde çok sütunlu veri yapıları
    # DataFrame oluşturma
        # pd.DataFrame(data) # Veriyi Dataframe e çevirmek. # Data: list, array, dictionary
        # seri.to_frame() # Seriyi Dataframe e çevirmek
    # Some attributes of Pandas DataFrame
        # df.info(), df.head(), df.tail(), df.sample(2), df.index, df.columns, df.shape, d
        # df.columns = ["new1", "new2", "new3", "new4", "new5", "new6"] , df.index = ["a"
        # df.rename(columns = {"new1": "aaa", "new2": "bbb"}), # df.rename(index = {1:
        # df["var1"].value_counts() , # df["var1"].value_counts(normalize=True)
    # Indexing, Slicing & Selection
        # df['Y'], df[['Y']], df[['X', "Y"]], # NOT: B ve D index , X ve Y sütun
        # df[1:3][['W', 'Y']] # Dikkat!: D dahil değil , # NOT: W ve Y sü
        # df["B":"D"][['W', 'Y']] # Dikkat!: D dahil # NOT: B ve D in
    # New Column
        # df["new1"] = df["W"] * df["X"] , # df["new2"] = np.arange(5)
    # Remove columns
        # df.drop(["new1", "new2"], axis=1) # inplace = True / axis=0 default
    # Remove rows
        # df.drop(["B", "D"], axis=0) # inplace = True / axis=0 default
    # .loc[ ] and .iloc[ ] -
        # loc : 1."index/sütun isimleri", 2.indexlemede "dahil" eder
        # iloc: 1.integer değerler , 2.indexlemede "dahil" etmez
        # df.loc["D":"G", "var2":"var3"] # Dikkat! : G dahil
        # df.iloc[3:6,1:2] # Dikkat! : 6. indexdeki satır(Yani "G") dahil değil , 2.in
        # NOT: indexler nümerik ise .loc kullanırken indexler için nümerik değer kullanabi
            # df.reset_index(inplace=True)
            # df.loc[2,["var2","var3"]]
    # Conditional Selection
        # df["var1"]>10][["var2", "var3"]] # Koşulu sağlayanlara True sağlamayanlara Fa
        # df[df["var1"]>10][["var2", "var3"]]] # True değerlerin bulunduğu satırlara(indexl
    # Two or More Conditional Statements
        # df[ () & () ], df[ () | () ] # &: Kesişim(Ortak) , |: Birleşim(Toplam)
        # (df["var1"]>10) & (df["var2"]<20) # Koşulu sağlayanlara True sağlamayanlara
        # df[(df["var1"]>10) & (df["var2"]<20)] # True değerlerin bulunduğu satırlara(indexl
    # Conditional Selection Using .loc[ ]
        # df.loc[((df["var1"] < 10) | (df["var1"] > 30)), ['var2', 'var3']] # 2 condition
    # reset_index() & set_index()
        # df.reset_index(drop = True) # indexi 0,1,2,... şeklinde yeniden oluşturdu. #drop=
        # df.set_index("var4") # Sütunu indexe koydu
    # Multi-Index & Index Hierarchy
        # Multi index oluşturup(Liste) bunu hierarşik indexe çevirip dataframe e index ola
        # df.index.names = ["Group", "Num"]
        # df.index.levels[1] # dtype: numeric.Int64Index # df.index.leve
        # df.index.get_level_values(0) # dtype: base.Index
        # df.index.get_level_values("Group")
        # df.index.get_level_values(0)[6] --> M3

#####
# DataFrame.groupby()


```

```

# Aggregation fonksiyonları ile kullanılır(sum,count,mean,median,min,...)
# df.groupby('Company').mean()                                     # Groupby içine yazılan
# df.groupby(['Company','Department'])[['Age','Sales']].mean() # Groupby dışına yazılı
# DataFrame.agg() / DataFrame.aggregate()
# df2.agg(['sum','min'])                                         # also: df2.agg([sum, min])
# df2[['var1','var2']].agg([sum,min])
# DataFrame.groupby().agg()
# df2.groupby('groups').agg({'min','median','max'})
# df2.groupby('groups').agg({'var1':(min,'max'), 'var2':[ 'median',np.mean] })
#####
# DataFrame.filter()
# Sütun isimlerine ve index isimlerine göre filtreleme yapıyor
# Diğerlerinin yapamadığı ve filter'in yaptığı 2 şey, (1.regex ve 2.like)
# df2.filter(["var1","var2"])
# df2.filter(regex="^va", axis=1) # df2.filter(regex='r2$', axis=1)
# df2.filter(like="ou", axis=1)
# DataFrame.groupby().filter()
# def filter_func(x):
#     return x['var1'].mean() > 39
# df2.groupby('groups').filter(filter_func)
# df2.groupby("groups").filter(lambda x: x["var2"].sum() < 800)
#####
# DataFrame.transform()
# Transform bizim vereceğimiz fonksiyona göre bir dönüşüm sağlar.
# Dönüşümün sonunda df ile AYNI UZUNLUKTA bir df sonuç oluşturur.(seri ise seri)
# Transform aggregate sonucunu DataFrame.transform() da döndürmez
# df_num = df2.iloc[:,1:3]
# df_num.transform(lambda x : x+10)
# df_num.var1.transform([np.sqrt,np.exp])
# DataFrame.groupby().transform()
# df2.groupby("groups")['var1'].transform("mean")
#####
# Series.apply()
# Belirttiğimiz fonksiyon için for döngüsüyle işlem yaptırıyor
# df3['col2'].apply(lambda x : 'high' if x>500 else 'low')
# def squared(x):
#     return x**2
# df3['col1'].apply(squared)
# DataFrame.apply()
# Vereceğimiz axis(satıra ya da sütuna) e göre işlem yapacağız. Default : axis=0
# Aggreate fonksiyonlarının sonuçlarını bize döndürür.
# Transform aggregate sonucu verinin uzunluğunu bozmayacak şekilde döndürür((Yani tek
# df2.apply(np.sum)
# def squared(x):
#     return x**2
# df_num.apply(squared)
# df_num.apply(lambda x: x + 2)
# DataFrame.applymap()
# İçine yazılan fonksiyonu dataframe in her bir elemanına uygular.
# Element-wise dir(Apply dan en önemli farkı budur. Diğer farkı applymap() sadece data
# Sadece dataframe ile kullanılır. Serilerle kullanılmaz. Seriler ile "map" kullanılır
# df_num.applymap(lambda x: x*5)
# df_num.applymap(np.sum)
# Series.map()
# Vereceğimiz değerlere göre neyi neyle değiştirmemiz gerekiyorsa değiştirir(map eder)

```

```
# df3.col2.map({444:"A", 666:"B"})  
#####  
# pivot() and pivot_table() : hierarşik index ve hierarşik sütunlardan oluşan veri yapıları  
# pivot      1. agg_func özelliği yok  
#           2. Bu yüzden duplicate değerler ile başa çıkamıyor.Ya hata verir ya da  
#           3. Duplicate değerler yok ise pivot_table ile aynı çıktıyı verir(values  
#           4. Values kısmına verdigimiz değişken kategorik ise çıktı verir  
#           5. Values Ayrıntı: "Values =" parametresinde yazdığımız değişkenler süt  
# pivot_table 1. agg_func özelliği var  
#           2. Bu yüzden duplicate değerler ile başa çıkabiliyor  
#           3. Values kısmına verdigimiz değişken kategorik ise çıktı vermez  
#           4. Ayrıntı: "Values =" parametresinde yazdığımız değişkenler sütuna geç  
#           5. Pivotun genelleştirilmiş halidir  
#####  
# DataFrame.stack(level= - 1, dropna=True)  
# dropna=False   : Tamamen NaN değer içeren satırları tutar  
# levelini belirttiğimiz sütunu, indexe geçirir  
# level: Dışta(En üstte) kalan sütun level=0 dır alta gittikçe level=1, level=2, ... $  
# DataFrame.unstack(level= - 1, fill_value=None)  
# levelini belirttiğimiz indexi, sütuna geçirir  
# level: Dışta(En solda) kalan index level=0 dır sağa gittikçe level=1, level=2, ... $  
# fill_value= "-" : Sonradan oluşan NaN değerlere tire(-) verir.Önceden zaten datafra
```

## ▼ PANDAS RECAP 2

### ► MISSING VALUES

```
[ ] ↴ 54 cells hidden
```

### ► OUTLIERS

```
[ ] ↴ 3 cells hidden
```

### ► 1.Detecting Outliers

```
[ ] ↴ 7 cells hidden
```

### ► 2.Handling Outliers

```
[ ] ↴ 23 cells hidden
```

## ▼ ÖZET / CHEAT SHEET

```

# MISSING VALUES

# Bazi Notlar
    # NaN : Not a Number
    # Steve Hoca: Eda dan sonra yapacağımız şey eksik değerlere çözüm bulmak
    # Steve Hoca: Çözüm için net bir metot yok(Veriden veriye değişiklik gösteriyor)

# Detecting Missing Values
    # df.isnull().sum()
    # df.notnull().sum()

# Handling Missing Values
    # 1.Drop
        # a.dropna(axis=0, how="any", thresh=None, inplace=False) # NaN değerleri drop etme
        # b.drop(axis=0)                                         # Satır ya da sütun
            # df.drop(["var1","var2"], axis=1)

    # 2. Filling Missing Values (Imputation)
        # 2.a.Filling with a specific value
            # df.fillna(0) # df.var1.fillna(0)
        # 2.b.Filling with any Proper Value # mean(), median(), where , interpolate()
            # df.fillna(df.mean())
            # df.fillna({"dept":"other","var1":df.var1.mean(), "var2":df.var2.median()})
            # df.where(cond = df.notna(), other=df.mean(), axis=1)      # Çok kullanılmıyor
            # df.interpolate()                                     # Not: "None" şeklinde yazan
                # 2 index arasında tek NaN varsa: Bir üst(örn: 4) ve bir alt(örn: 2) i
                # .. ortalamasını al ve ara indexe(4 ile 2 arası --> 3. indexe) yazdır
                # 2 index arasında 2 veya daha çok NaN varsa: linspace mantığı ile çal
                # .. Örneğin 2,3,4,5. indexlerdeki değerler 30,NaN,NaN,39 diyelim.Bunu
        # 2.c.Filling the Missing Values of Categorical Variables # mode, ffill, bfill
            # df.dept.fillna(df.dept.mode()[0])
            # df[["dept"]].fillna(method="ffill") # Yukardan aşağı dolduruyor
            # df[["dept"]].fillna(method="bfill") # Aşağıdan yukarı dolduruyor
        # 2.d.Filling by condition & by Group of the Categorical Variables
            # df[["trans_salary"]] = df.groupby(["status", "dept"]).salary.transform("mean")
            # df.salary.fillna(df.trans_salary, inplace=True)

    # 3. Keep missing values

# OUTLIERS

# Detecting Outliers
    # 1.Graphs                      : Box plot, Histogram, Scatter plot
        # sns.boxplot(x=df.table);
        # sns.histplot(x=df.table, bins=20)
    # 2.Tukey's Fences | Tukey's Rule : IQR method
        # IQR = Q3-Q1
        # Q1- 1.5 x IQR , Q3 + 1.5 x IQR
    # 3.Statistical Tests           : Grubbs' test, Chi-square test, Dixon's Q test
        # Bunu görmedik

# Handling with Outliers
    # 1.Remove the outliers         : 1. Drop etmek 2.TILDA(~) = "NOT"
        # a.Drop
            # drop_index = df.loc[((df.table < lower_lim) | (df.table > upper_lim))].index
            # df.drop(index=drop_index, axis = 0)
        # b.Tilda(~)
            # df_cleaned = df.loc[~((df.table < lower_lim) | (df.table > upper_lim))]

    # 2.Winsorize : Oran belirleyerek whiskerslara en yakın olan değere baskılama
        # from scipy.stats.mstats import winsorize
        # a1 = len(data.var1[data.var1<lower_lim1])/len(data) # lower_lim den küçük ol

```

```

# b1 = len(data.var1[data.var1>upper_lim1])/len(data) # upper_lim den büyük ol
# winsorize(data.var1, (a1,b1))
# NOT: Steve Hoca: Bana bu yöntem mantıklı gelmiyor.Eğer o outlier yerine bir
# 3.Data transformation : Verinin ölçüğünü değiştirerek outlierları azalt
# carat_log = np.log(df.carat)
# sns.boxplot(x=carat_log);
# Hala elimizde outlier var bunları 1.Drop ya da 2.Tilda kullanarak verimizden
# 4.Replacing the outliers : (mean, median, mode)
# 5.Using different analysis methods : İstatistiksel non-parametric testler uygula
# 6.Valuing the outliers : Aykırı değer yerine başka değer vermek

```

Thank you for your time

## ▼ PANDAS RECAP-3

```

import numpy as np
import pandas as pd

```

### COMBINING DATAFRAMES

#### ► 1.APPEND

[ ] ↓ 2 cells hidden

#### ► 2.CONCATENATION

[ ] ↓ 8 cells hidden

#### ► 3.MERGING

[ ] ↓ 28 cells hidden

#### ► 4.JOIN

[ ] ↓ 7 cells hidden

## ▼ ÖZET/CHEAT SHEET



```

# Sütunlarda BAZI DEĞERLER ÇOKLU ve bazı değerler aynı ise;
##### LEFT3-KEY: 'K0', 'K0', 'K1', 'K1' ##### RIGHT3-KEY: 'K0', 'K1',
# how="left" a göre; SOLU baz alarak çarprazlama kombinasyonlar
# pd.merge(left3,right3, how="left",on="key")
# Satır sayısı: 2x1 + 2x2 = 6
# how="right" a göre; SAĞI baz alarak çarprazlama kombinasyonlar
# pd.merge(left3,right3, how="right",on="key")
# Satır sayısı: 1x2 + 2x2 + 1 = 7
# how="inner" a göre; ORTAK olan çarprazlama kombinasyonlar
# pd.merge(left3,right3, how="inner",on="key")
# Satır sayısı: 2x1 + 2x2 = 6
# how="outer" a göre; OLASI Çarprazlama kombinasyonlar
# pd.merge(left3,right3, how="outer",on="key")
# Satır sayısı: 2x1(ya da 1x2) + 2x2(ya da 2x2) + 1(sağdan(K2) gel

# IKI KEY(Ortak sütun) VARSA; NOT:Aşağıda "değer" derken "key" olan sütunlarında
# Sütunlarda BAZI DEĞERLER ÇOKLU ve bazı değerler aynı ise; key1 ve key2 ler v
##### left-key1: 'K0', 'K0', 'K1', 'K2','K2' ##### right-key1: 'K0',
##### left-key2: 'K0', 'K1', 'K0','K1' ##### right-key2: 'K0',
# how="left" a göre; SOLU baz alarak çarprazlama kombinasyonlar
# soldaki df in key1-key2 kombinasyonuna göre
# pd.merge(left4, right4, how="left", on=["key1","key2"])
# how="right" a göre; SAĞI baz alarak çarprazlama kombinasyonlar
# sağdaki df in key1-key2 kombinasyonuna göre
# pd.merge(left4, right4, how="right", on=["key1","key2"])
# how="inner" a göre; ORTAK olan çarprazlama kombinasyonlar
# pd.merge(left4, right4, how="inner", on=["key1","key2"])
# İki df in de key1-key2 lerin ortak olan kombinasyonlarına göre
# how="outer" a göre; OLASI Çarprazlama kombinasyonlar
# pd.merge(left4, right4, how="outer", on=["key1","key2"])
# İki df in olası key1-key lerin olası tüm kombinasyonlarına göre
# NOT: satır sayısı mantıkları, üstte açıklananla aynı

# TEK KEY Sütunlarda BAZI DEĞERLER ÇOKLU ve bazı değerler aynı ise Ve satır sa
##### lkey': ['x', 'y', 'z', 'x', 'z'] ##### rkey': ['a', 'x', 'z', ' '
# how="left" a göre; SOLU baz alarak çarprazlama kombinasyonlar
# pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="left")
# soldaki df in lkey-rkey kombinasyonuna göre
# how="right" a göre; SAĞI baz alarak çarprazlama kombinasyonlar
# pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="right")
# sağdaki df in lkey-rkey kombinasyonuna göre
# how="inner" a göre; lkey-rkey ORTAK olan çarprazlama kombinasyonlar
# pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="inner")
# İki df in de lkey-rkey lerin ortak olan kombinasyonlarına göre
# how="outer" a göre; OLASI Çarprazlama kombinasyonlar
# pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="outer")
# İki df in olası lkey-rkey lerin olası tüm kombinasyonlarına göre
# NOT: satır sayısı mantıkları, üstte açıklananla aynı

# TEK KEY Sütunlarda değerler aynı değilse;
##### lkey': ['x', 'y', 'z', 'x'], ##### rkey': ['a', 'b', 'c', 'b']
# how="left" a göre;
# pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="left")
# Soldaki df gelir, sağdaki df değerleri NaN gelir
# how="right" a göre;
# pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="right")
# Sağdaki df gelir, soldaki df değerleri NaN gelir

```

```

# how="inner" a göre;
    # pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="inner")
    # Boş gelir
# how="outer" a göre;
    # pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="outer")
    # how="left" ile how="right" alt alta eklenmiş gibi gelir
    # NOT: pd.merge(left6, right6, left_index=True, right_index=True, how="outer")
#####
# 4.JOIN : Indexe ya da sütuna göre birleştirme yapar
# left.join(right) or right.join(left)
# index belirtilmemişse indexe göre birleştirme yapar
    # left7.join(right7)
# index belirtilmemişse sütunlarda birleştirme yapar. Eşleşmeyenler NaN gelir
# ANCAK key olarak kullanılacak sütun isimleri aynı olanları birleştiremez.
# Bunu önlemek için lsuffix="_left", rsuffix="_right"
    # left8.join(right8, lsuffix="_left", rsuffix="_right")
# "Key" leri index e taşıyabilirim.
    # left8.set_index("key").join(right8.set_index("key")) # left in "key" ini index
# left8.join(right8.set_index("key"), on="key")           # left in "key" ini sütuna

```

## # PANDAS RECAP-3

```

import numpy as np
import pandas as pd

```

## ### COMBINING DATAFRAMES

## ## 1.APPEND

Alt alta ekleme(append) yapar. Append, concatin(pd.concat([df1,df2],axis=0, join='outer')

Eklemeden(Append den) sonra Index numaraları aynı kalır. İhtiyaca göre, ignore\_index=True

```

# df1.append(df2, ignore=index=False) ---> default
# df1.append([df2,df3,df4,df5], ignore=index=False) ---> default --> 2 den fazla df i de

```

## ## 2.CONCATENATION

Satır veya sütunlara göre birleştirme yapar

```

# pd.concat([df1,df2],axis=0, join='outer') ---> default,
# pd.concat([df1,df2,df3,df4,df5],axis=0, join='outer') ---> default --> 2 den fazla df i

```

```

df4 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])

```

```

df5 = pd.DataFrame({'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7'],
                    'E': ['A4', 'A5', 'A6', 'A7']},
                    index=[0, 1, 2, 3])

```

```

index=[4, 1, 2, 3])

display(df4,df5)
# Indexlerinin biri farklı, Sütunlarının biri farklı

pd.concat([df4,df5], axis=0)    # ignore_index=True # default join='outer'
##### df4.append(df5)           # Aynı çıktıyı verir # ignore_index

pd.concat([df4,df5],axis=0, join="inner") # ignore_index=True
# Sütunlardan 3 ü ortak olduğu/kesiştiği için çıktı verdi
# NOT: # df4.append(df5) join aynı çıktıyı vermez çünkü join=inner gibi bir parametresi yok
pd.concat([df4,df5], axis=1) # ignore_index=True # default join='outer'

pd.concat([df4,df5], axis=1, join="inner")
# indexler aynı olduğu için yukarıdaki ile aynı çıktı verdi
# Sütunlar ve sıraları aynı kaldı

pd.concat([df4,df5.reset_index(drop=True)], join="inner", axis=1) # Her şey düzeldi.
# .. reset_index(drop=True) yapınca df5'in indexleri 0,1,2,3'e dönüştü

# ÖNEMLİ NOT: Burada reset_index yapınca indexler 0,1,2,3 olarak başlayacak o yüzden df5
# Ama df4'in indexleri 0,1,2,3 olduğu için böyle oldu. Eğer df4'in indexleri farklı olsaydı

```

### ## 3. MERGING

Yan yana birleştirme işlemi yapar(2 df için).AMA--> df4=pd.merge(pd.merge(df1,df2,on='Cour  
Yan yana ortak SÜTUNA göre birleştirme işlemi yapar Çıktı da solda gelecek dataframe ve sa

```

# pd.merge(left=df1, right=df2, how="inner", on='key') ---> default. Burada: on='key' : ik

# MERGING
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})

display(left, right)
# Ortak olan "key" sütunu değerleri aynı

pd.merge(left=left, right=right, how="inner", on="key")
# left e(Sola) left df ini, right a(Sağ) right df ini koyduk.
# Ortak olan key sütununun değerleri ortak, direk yan yana birleştirildi
# NOT: Değerlerin sırası farklı olsaydı soldakiniin değerlerini baz alır ona göre

left2 = pd.DataFrame({'key': ['K0', 'K1', 'K4', 'K5'],
                      'A': ['A0', 'A1', 'A2', 'A3'],
                      'B': ['B0', 'B1', 'B2', 'B3']})

```

```

right2 = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
display(left2, right2)
# Ortak olan key sütunun değerleri farklı

pd.merge(left=left2,right=right2, how="inner",on="key")
# how="inner" # Ortak olan değerlere göre birleştirme yaptı

pd.merge(left=left2,right=right2, how="outer",on="key") # how="outer" her iki df dede tüm
# .. üstteki aynı mantıkta yaptı ama eşleşmeyen değerlere NaN verdi

pd.merge(left2,right2, how="right",on="key")      # how="right" :right2 nin key değerlerine

left3 = pd.DataFrame({'key': ['K0', 'K0', 'K1', 'K1'],
                      'A': ['A0', 'A1', 'A2', 'A3'],
                      'B': ['B0', 'B1', 'B2', 'B3']})

right3 = pd.DataFrame({'key': ['K0', 'K1', 'K1', 'K2'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})

display(left3, right3)
# Ortak olan key sütununun değerleri farklı ve left3-key de olan her değer right3-key de var

```

```

pd.merge(left3,right3, how="inner",on="key") # how="inner" ortak olanları alıyordu # 0. indexde K0 olduğu için onu aldı A ve B yi normal yazdı, C ve D yi 0. indexi normal yazdı 1. inde # .. 2. ve 3. index aynı mantıkta(3. indexteki C ve D sütununu çökladı)

```

```

pd.merge(left3,right3, how="left",on="key") # Aynı çıktıyı verdi. Çünkü;
# Ortak olan key sütunu için; left3-key de olan her değer right3-key de var

```

```

pd.merge(left3,right3, how="outer",on="key") # how="outer" 2 df dede tüm indexlere karşılı

```

"" OLASI KOMBINASYONLAR

K0 - K0 A0 B0 C0 D0 \*

K0 - K1 A0 B0 C1 D1

K0 - K1 A0 B0 C2 D2

K0 - K2 A0 B0 C3 D3

K0 - K0 A1 B1 C0 D0 \*

K0 - K1 A1 B1 C1 D1

K0 - K1 A1 B1 C2 D2

K0 - K2 A1 B1 C3 D3

K1 - K0 A2 B2 C0 D0

K1 - K1 A2 B2 C1 D1 \*

K1 - K1 A2 B2 C2 D2 \*

K1 - K2 A2 B2 C3 D3

K1 - K0 A3 B3 C0 D0

K1 - K1 A3 B3 C1 D1 \*

K1 - K1 A3 B3 C2 D2 \*

K1 - K2 A3 B3 C3 D3 \*

""

```

left4 = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2','K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1','K1'],
                     'A': ['A0', 'A1', 'A2', 'A3','A4'],
                     'B': ['B0', 'B1', 'B2', 'B3','B4']})

right4 = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2','K3'],
                      'key2': ['K0', 'K0', 'K0', 'K0','K1'],
                      'C': ['C0', 'C1', 'C2', 'C3','C4'],
                      'D': ['D0', 'D1', 'D2', 'D3','D4']})

display(left4, right4) # Burada primary key ler 2 tane

pd.merge(left4, right4, how="inner", on=["key1","key2"]) # how="inner". key1 ve key2 si 2

pd.merge(left4, right4, how="outer", on=["key1","key2"]) # how="outer" 2 df de de key1 ve

pd.merge(left4, right4, how="left", on=["key1","key2"])

left5 = pd.DataFrame({'lkey': ['x', 'y', 'z', 'x', 'z'],
                      'lvalue': [2, 3, 5, 7, 0]})

right5 = pd.DataFrame({'rkey': ['a', 'x', 'z', 'b'],
                      'rvalue': [7, 8, 9, 10]})

display(left5, right5)

pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="inner") # how="inner" ortak

pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="outer")

pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="left")

pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="right")

left6 = pd.DataFrame({'lkey': ['x', 'y', 'z', 'x'],
                      'lvalue': [2, 3, 5, 7]})

right6 = pd.DataFrame({'rkey': ['a', 'b', 'c', 'b'],
                      'rvalue': [7, 8, 9, 10]})

display(left6, right6) # Burada da indexler üzerinden yola çıkarak birleştirme yapacağız

pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="inner") # how="inner" ilgi

pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="outer") # Tüm lkey ve rkey

pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="left")

pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="right")

pd.merge(left6, right6, left_index=True, right_index=True, how="outer") # left_index=True,
## 4.JOIN

```

```

left7 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                      'B': ['B0', 'B1', 'B2']},
                     index = ['K0', 'K1', 'K2'])

right7 = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                       'D': ['D0', 'D2', 'D3']},
                      index = ['K0', 'K2', 'K3'])

display(left7, right7)

left7.join(right7) # left7 nin yanına right7 yi koydu, indexleri ortak olanların değerleri

# .. Ortak olmayan "K1" indexinin C ve D değerleri NaN geldi
# Not: Burada how="left" default. O yüzden soldaki df in(left7 nin) indexlerini getirdi
right7.join(left7) # Üsttekinin tam tersi

left8 = pd.DataFrame({'key1': ['K0', 'K2', 'K3', 'K4', 'K5', 'K6'],
                      'X': ['X0', 'X2', 'X3', 'X4', 'X5', 'X6']})

right8 = pd.DataFrame({'key': ['K0', 'K2', 'K3', 'K7'],
                       'Y': ['Y0', 'Y2', 'Y3', 'Y4']})

display(left8, right8)

# left8.join(right8) # HATA. Aynı isimdeki column(key) olunca birleştirmiyor "join". Bunu
# .. Bunu engellemek için "lsuffix" ve "rsuffix" kullanabiliriz
left8.join(right8, lsuffix="_left", rsuffix="_right") # Ortak isimli sütunların sonuna "_l

# If you want to join using the key columns, you need to set key to be the index in both d
# The joined DataFrame will have key as its index.
# Öyle birleştirelim "key_left", "key_right" çıktısı gelmesin dersem; # "Key" leri index e
left8.set_index("key").join(right8.set_index("key")) # merge ün yaptığı bir işlem için bur

# left8.join(right8, on="key") # Hata. on="key" dediğimizde sadece left8 e göre "key" i al
left8.join(right8.set_index("key"), on="key") # left in "key" ini sütuna yazar, right "k

# ÖZET/CHEAT SHEET

# COMBINING DATA FRAMES
#####
# 1.APPEND : Alt alta eklemeye(append) yapar. Append, concat(pd.concat([df1,df2],axis=0, join='outer'))
# df1.append(df2, ignore=index=False) ---> default
# df1.append([df2,df3,df4,df5], ignore=index=False) ---> default --> 2 den fazla df i
# Eklemeden(Append den) sonra Index numaraları aynı kalır. İhtiyaca göre, ignore_index=True
# NOT: "join" parametresi yoktur
#####
# 2.CONCATENATION : Satır veya sütunlara göre birleştirme yapar
# pd.concat([df1,df2],axis=0, join='outer') ---> default
# pd.concat([df1,df2,df3,df4,df5],axis=0, join='outer') ---> default --> 2 den fazla df i
# NOT: join= "inner"(KESİŞİM/ORTAK), join="outer"(BİRLEŞİM/OLASI TÜM COMBINASYON) vardır
# JOIN = "inner" : Ortak olanları alır, eşleşmeyenler değerler gelmez.
# join="inner" --> Yan yana(axis=1) concat yaparken indexler farklı ise indexler değişir
# pd.concat([df4,df6], axis=1, join="inner")
# join="inner" --> Alt alta(axis=0) concat yaparken sütunlar farklı ise sütunlar değişir
# pd.concat([df4,df5],axis=0, join="inner")
# Eğer birden fazla df concat olursa join="inner" ile ,mantık aynı

```

```

# NOT : Eşleşmeyenler gelsin diye "ignore_index" kullanılabilir
    # pd.concat([df4,df5.reset_index(drop=True)], join="inner", axis=1)
# JOIN ="outer" : Bütün olası sonuçları alır, eşleşmeyen değerler NaN gelir
    # join="outer" --> Yan yana(axis=1) concat yaparken olası tüm indexler gelir.
        # pd.concat([df4,df5], axis=1, join="outer")
    # join="outer" --> Alt alta(axis=0) concat yaparken olası tüm sütunlar gelir.
        # pd.concat([df4,df5], axis=0, join="outer")
    # Eğer birden fazla df concat olursa join="inner" ile ,mantık aynı
#####
# 3.MERGE : Yan yana ortak SÜTUNA göre birleştirme işlemi yapar Çıktı da solda gelecek dat
#####
# AŞAĞININ ÖZETİ:
    # how="inner" : ortak olan keyleri / kombinasyonlarını alıp birleştirir. Eşleşmeye
    # how="outer" : Olası tüm keyleri / kombinasyonlarını alıp birleştirir. Eşleşmeyen
    # how="left"   : soldaki df in keylerini alıp birleştirir. Eşleşmeyen değerler NaN
    # how="right"  : sağdaki df in keylerini alıp birleştirir. Eşleşmeyen değerler NaN
#####
# pd.merge(left=df1, right=df2, how="inner", on='key') --> default on='key' : iki da
# NOT: 2 den fazla df birleşim--> df4=pd.merge(pd.merge(df1,df2, on='Courses'),df3, on='
# NOT: how= "inner"(KESİŞİM/ORTAK), how="outer"(BİRLEŞİM/OLASI TÜM COMBINASYON), how=
    # TEK KEY(Ortak sütun) VARSA; NOT:Aşağıda "değer" derken "key" olan sütunun değerl
        # Sütunlarda her DEĞERDEN BİR TANE, DEĞERLER AYNI ve SIRASI DA AYNI ise; how="
        ##### left-key: 'K0', 'K1', 'K2', 'K3' ##### right-key: 'K0', 'K1',
            # pd.merge(left=left, right=right, how="inner", on="key")
        # Sütunlarda her DEĞERDEN BİR TANE, DEĞERLER AYNI ve left1-key in SIRASI FARKL
        ##### left-key: 'K1', 'K0', 'K2', 'K3' ##### right-key: 'K0', 'K1',
            # Üstteki örnekte left-key i K1,K0,K2,K3 olacak şekilde değiştirip sonuç g
        # Sütunlarda her DEĞERDEN BİR TANE, değerlerinin BAZILARI aynı ise ;
        ##### left-key: 'K0', 'K1', 'K4', 'K5' ##### left-key: 'K0', 'K1',
            # how="inner" da ortak olan değerleri alır. Eşleşmeyen değerler gelmez
                # pd.merge(left=left2,right=right2, how="inner",on="key")
        # how="outer" da tüm olası değerleri alır. Eşleşmeyen değerler NaN gelir
            # pd.merge(left=left2,right=right2, how="outer",on="key")
        # how="left" soldaki keyleri alarak sonuç gelir. Eşleşmeyenler NaN gelir
            # pd.merge(left=left2,right=right2, how="left",on="key")
        # how="right" sağdaki keyleri alarak sonuç gelir. Eşleşmeyenler NaN gelir
            # pd.merge(left=left2,right=right2, how="right",on="key")
    # Sütunlarda BAZI DEĞERLER ÇOKLU ve bazı değerler aynı ise;
    ##### LEFT3-KEY:'K0', 'K0', 'K1', 'K1' ##### RIGHT3-KEY: 'K0', 'K1',
        # how="left" a göre; SOLU baz alarak çarprazlama kombinasyonlar
            # pd.merge(left3,right3, how="left",on="key")
            # Satır sayısı: 2x1 + 2x2 = 6
        # how="right" a göre; SAĞI baz alarak çarprazlama kombinasyonlar
            # pd.merge(left3,right3, how="right",on="key")
            # Satır sayısı: 1x2 + 2x2 + 1 = 7
        # how="inner" a göre; ORTAK olan çarprazlama kombinasyonlar
            # pd.merge(left3,right3, how="inner",on="key")
            # Satır sayısı: 2x1 + 2x2 = 6
        # how="outer" a göre; OLASI Çarprazlama kombinasyonlar
            # pd.merge(left3,right3, how="outer",on="key")
            # Satır sayısı: 2x1(ya da 1x2) + 2x2(ya da 2x2) + 1(sağdan(K2) gel
    # IKI KEY(Ortak sütun) VARSA; NOT:Aşağıda "değer" derken "key" olan sütunlarında
        # Sütunlarda BAZI DEĞERLER ÇOKLU ve bazı değerler aynı ise; key1 ve key2 ler v
        ##### left-key1: 'K0', 'K0', 'K1', 'K2','K2' ##### right-key1: 'K0',
        ##### left-key2: 'K0', 'K1', 'K0', 'K1', 'K1' ##### right-key2: 'K0',

```

```

# how="left" a göre; SOLU baz alarak çarprazlama kombinasyonlar
    # soldaki df in key1-key2 kombinasyonuna göre
    # pd.merge(left4, right4, how="left", on=["key1","key2"])
# how="right" a göre; SAĞI baz alarak çarprazlama kombinasyonlar
    # sağdaki df in key1-key2 kombinasyonuna göre
    # pd.merge(left4, right4, how="right", on=["key1","key2"])
# how="inner" a göre; ORTAK olan çarprazlama kombinasyonlar
    # pd.merge(left4, right4, how="inner", on=["key1","key2"])
    # İki df in de key1-key2 lerin ortak olan kombinasyonlarına göre
# how="outer" a göre; OLASI Çarprazlama kombinasyonlar
    # pd.merge(left4, right4, how="outer", on=["key1","key2"])
    # İki df in olası key1-key lerin olası tüm kombinasyonlarına göre
# NOT: satır sayısı mantıkları, üstte açıklananla aynı

# TEK KEY Sütunlarda BAZI DEĞERLER ÇOKLU ve bazı değerler aynı ise Ve satır sa
##### lkey': ['x', 'y', 'z', 'x', 'z'] #####'rkey': ['a', 'x', 'z', ' '
    # how="left" a göre; SOLU baz alarak çarprazlama kombinasyonlar
        # pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="left")
        # soldaki df in lkey-rkey kombinasyonuna göre
    # how="right" a göre; SAĞI baz alarak çarprazlama kombinasyonlar
        # pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="right")
        # sağdaki df in lkey-rkey kombinasyonuna göre
    # how="inner" a göre; lkey-rkey ORTAK olan çarprazlama kombinasyonlar
        # pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="inner")
        # İki df in de lkey-rkey lerin ortak olan kombinasyonlarına göre
    # how="outer" a göre; OLASI Çarprazlama kombinasyonlar
        # pd.merge(left5, right5, left_on="lkey", right_on="rkey", how="outer")
        # İki df in olası lkey-rkey lerin olası tüm kombinasyonlarına göre
    # NOT: satır sayısı mantıkları, üstte açıklananla aynı

# TEK KEY Sütunlarda değerler aynı değilse;
#####'lkey': ['x', 'y', 'z', 'x'], #####'rkey': ['a', 'b', 'c', 'b']
    # how="left" a göre;
        # pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="left")
        # Soldaki df gelir, sağdaki df değerleri NaN gelir
    # how="right" a göre;
        # pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="right")
        # Sağdaki df gelir, soldaki df değerleri NaN gelir
    # how="inner" a göre;
        # pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="inner")
        # Boş gelir
    # how="outer" a göre;
        # pd.merge(left6, right6, left_on="lkey", right_on="rkey", how="outer")
        # how="left" ile how="right" alt alta eklenmiş gibi gelir
    # NOT: pd.merge(left6, right6, left_index=True, right_index=True, how="oute

#####
# 4.JOIN : Indexe ya da sütuna göre birleştirme yapar
    # left.join(right) or right.join(left)
    # index belirtilmişse indexe göre birleştirme yapar
        # left7.join(right7)
    # index belirtilmemişse sütunlarda birleştirme yapar. Eşleşmeyenler NaN gelir
    # ANCAK key olarak kullanılacak sütun isimleri aynı olanları birleştiremez.
    # Bunu önlemek için lsuffix="_left", rsuffix="_right"
        # left8.join(right8, lsuffix="_left", rsuffix="_right")
    # "Key" leri index e taşıyabilirim.

```

```
# left8.set_index("key").join(right8.set_index("key")) # left in "key" ini index
# left8.join(right8.set_index("key"), on="key")           # left in "key" ini sütuna
```

## ▼ PANDAS RECAP-4

```
import numpy as np
import pandas as pd
import seaborn as sns
```

### ► WORKING WITH TEXT DATA

```
[ ] ↓ 39 cells hidden
```

### ▼ Working with Time Data

```
# Pandas
# 1) pd.to_datetime() -- > string , list_like vs vs gibi ifadeyi time tipine çevirmeyi sağlar
# 2) Series.dt.          -- > datetime a çevrilmiş data içinden(date, year, quarter, month, ...
#                           # ...dayofweek, hour, minute, second, microsecond, day_name() vs almaya yarıyor

# pandas kendi içerisinde modüllerle yukarıdakileri yapabiliyor ama
# Ayrıca Datetime module ü var
# class datetime date
# class datetime time
# class datetime datetime    # mikrosaniyeye kadar bilgi gösterir
# class datetime timedelta  # parantez içine bilgiler yazıp bunu matematiksel işlemler içi
# class datetime tzinfo
# class datetime timezone

# Biz sadece
# 1) class datetime datetime
# 2) class datetime timedelta i kullanacağız
# -----
# 3) .strftime()      -- > date/datetime/timedelta object i string type a dönüştürür
#                           # datetime object sonuna .strftime() yazarsam içindeki formattaki ifadeye çeviri
# 4) .strptime()      -- > Sadece string type i datetime object e dönüştürür

df = pd.read_csv("time_exercise.csv")
df.head()
```

	<b>id_product</b>	<b>order_date</b>	<b>product_quantity</b>	<b>product_price</b>	<b>entry_date</b>
0	401	2021-01-23	1.0	541.487603	2018-12-04
1	416	2020-04-02	1.0	131.181818	2018-12-04
2	717	2018-09-10	1.0	2005.400500	2018-10-01

```
df.info() # order_date ve entry_date'i datetime'a çevirmeliyiz
# Steve Hoca: Siz de çalışmalarınızda bunları datetime'a çevirip öyle çalışmalısınız
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 911 entries, 0 to 910
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id_product      911 non-null    int64  
 1   order_date      911 non-null    object 
 2   product_quantity 911 non-null   float64 
 3   product_price   911 non-null   float64 
 4   entry_date      911 non-null   object 
dtypes: float64(2), int64(1), object(2)
memory usage: 35.7+ KB
```

## ► 1. pd.to\_datetime()

[ ] ↓ 9 cells hidden

## ► 2.Series.dt()

[ ] ↓ 3 cells hidden

# Datetime Module

## ► 1.class datetime.datetime

[ ] ↓ 4 cells hidden

## ► 2.class datetime.timedelta

[ ] ↓ 4 cells hidden

## ► 3. strftime()

Converting from date/datetime/timedelta object to string type

datetime object sonuna .strftime() yazarsam içindeki formattaki ifadeye çevir dersem ifadeyi, o formata çevirir

[ ] ↓ 6 cells hidden

## ► 4. strftime()

Converting from string type to datetime object

[ ] ↓ 8 cells hidden

# ▼ DATA INPUT&OUTPUT

## ► 1.CSV INPUT AND OUTPUT

[ ] ↓ 8 cells hidden

## ► 2.EXCEL INPUT AND OUTPUT

[ ] ↓ 6 cells hidden

## ► 3.HTML INPUT-OUTPUT

[ ] ↓ 5 cells hidden

# ▼ ÖZET/CHEAT SHEET

```
#####
# WORKING WITH TEXT DATA
# String Methods()
    # 1. Convert string - # str.lower(), str.upper(), str.capitalize(), str.title(), s
        # df.staff.str.lower()          # Tüm harfleri küçük yaptı
        # df.staff.str.upper()         # Tüm harfleri büyük yaptı
        # df.staff.str.capitalize()   # Sadece ilk harfi büyük yaptı
        # df.staff.str.title()        # Kelimelerin baş harflerini büyük diğer yerleri
        # df.staff.str.swapcase()     # Büyük gördüğünü küçük, küçük gördüğünü büyük ya
    # 2. Return boolean - # str.isalpha(), str.isnumeric(), str.isalnum(), str.startswith(),
        # df.job.str.isalpha()         # Alfabetik değer mevcut olduğunda True döndürür(
        # df.age.str.isnumeric()       # Nümerik mi değil mi # NOT: sütun dtype object i
            # Bunu düzeltmek için age in dtype ini değiştirmem gereklili "string olarak"
        # df.salary.str.isalnum() # output: index 3,7 TRUE, diğerleri FALSE. Sadece ha
        # df.job.str.startswith("data") # output: index 2,6,7 TRUE, diğerleri FALSE #
        # df.job.str.endswith("per")   # output: index 4,5 TRUE, diğerleri FALSE #
        # df.job.str.contains("data")  # output: index 2,6,7 TRUE, diğerleri FALSE #
            # NOT: Eğer bu boolean sonuçları df içine alırsak dataframe sonuçlar alabı
                # örn: df[df.job.str.endswith("per")]
                # örn: df.loc[df.job.str.contains("data"), "department"] ="DS"
```

```

# örn: df.loc[df.job.str.contains("data"), "department"] ="IT"
# NOT: regex kullanılabılır contains ile(replace ile de)
    # df.salary.str.contains("[a-z]+") # output: index 3,7 TRUE, diğe
    # ... içinde "a" dan "z" ye kadar herhangi bir tane harf geçenler
# 3. Applying function - # str.strip(), str.replace(), str.split(), str.find(), st
    # df.salary.str.strip("\").str.rstrip("dolar").str.lstrip("$") # strip: Çıkar
        # df.salary.str.strip("\\"dolar$") # 2. yol
        # df.salary.str.strip("\\"dolar$").str.replace(",","") # replace: Değ
        # df.salary.str.strip("\\"dolar$").str.replace(",","").astype("int") # Son
# df["age"] = df.age.replace("-", np.nan) # NOT: # Bunu str ile yapacak olsay
    # .. string olarak isteyecekti # df.age.str.replace("-", "np.nan") # Ha
# df.staff.str.title().str.split()## isimlerimi düzeltmesi için title uyguladı
    # df["first_name"] = df.staff.str.title().str.split().str[0]
    # df["last_name"] = df.staff.str.title().str.split().str[1]
    # df.drop("staff", axis=1, inplace=True)
# df.job.str.find("developer") # -1 olanlar "developer" geçmeyenler
# df.job.str.findall("developer") # "developer" varsa o satırda o ifadeden ka
# df.job.str.findall("developer").apply(len) # 0 ifadeden kaç adet olduğunu g
# str.join(): ÖRNEK için ilgili bölüme bakınız

# NOT1 : Steve Hoca: find ve findall çok kullanılmıyor
# NOT2 : Series.str.replace() vs Series.replace
    # Purpose: Use str.replace for substring replacements on a single string column, and r
    # Usage: str.replace can replace one thing at a time. replace lets you perform multipl
    # Default behavior: str.replace enables regex replacement by default. replace only per

#####
# DUMMY OPERATIONS - get_dummies()
    # Syntax1: pd.get_dummies(data, prefix=None, prefix_sep="_") ----- drop_first=True
    # Syntax2: df["col_name"].str.get_dummies(sep = ",") ----- drop_first=True
    # Bir datasetini modele sokmadan önce değerlerimizin hepsi nümerik olmalı
    # Bu yüzden label encoding ve one-hot encoding yaparak kategorik sütunları nümeric süt
    # pd.get_dummies(arraylike) pd.get_dummies(Series) pd.get_dummies(DataFrames)
    # Steve Hoca: Modelin biraz daha hızlı çalışması için drop_first yapmalıyız
    # Steve Hoca: O sütunun bilgi kaybı olmuyor yani sütunlardan hangisi düşerse düşün(dr
        # Seriye dummy
            # pd.get_dummies(df.age, drop_first=True) # .add_prefix("ageeeeeee_")
        # df imdeki sütunlardaki sorunlar hallolduysa; df e dummy
            # pd.get_dummies(df_final, drop_first= True)
        # Bizim df de Skills ve skills de sorun vardı. Örneğin alttaki gibi onları atıp du
            # df.drop(["Skills","skills"],axis=1, inplace=True)
        # pd.get_dummies(df) # add_prefix("Skills_")

#####
# WORKING WITH TIME DATA
# Steve Hoca: Çalışmalarınızda datetime a uygun ancak tipi datetime olmayan sütunları date
# PANDAS
    # 1) pd.to_datetime() -- > string , list_like vs vs gibi ifadeyi time tipine çevirmey
        # pd.to_datetime(df["entry_date"])
            # df.entry_date.min()
            # df.entry_date.max()
            # df.entry_date.max() - df.entry_date.min()
        # pd.to_datetime(a,format="%d-%m-%Y")
    # 2) Series.dt. -- > datetime a çevrilmiş data içinden(date, year, quarter, mon

```

```

# df.entry_date.dt.year.tail(7)
# df.entry_date.dt.quarter
# df.entry_date.dt.dayofweek
# df.entry_date.dt.day_name()
# it can be date, year, quarter, month, week, day, weekday, dayofweek, hour, minut

# DATETIME
# 1) class datetime datetime --- from datetime import datetime
#     # datetime.now
#     # current_datetime.date()           # type(current_datetime.date()) : datetime.date
#     # current_datetime.weekday()        # output: 2   # Haftanın 2. günü # Monday e 0
#     # current_datetime.isoweekday()    # output: 3   # Haftanın 3. günü # Monday e 1
# 2) class datetime timedelta --- from datetime import timedelta
#     # timedelta(days=2) # Docstring e bakarak parametreleri girebiliriz. # Burada sad
#     # current_datetime - current_datetime.weekday() # HATA. Çünkü type(current_datetime
#             # DIKKAT : # type(current_datetime.year) : int
#     # current_datetime - timedelta(days=2) #Bugünden 2 gün çıkardım
#     # current_datetime + timedelta(weeks=2, days=3, hours=4, minutes=10) # 2 hafta 3 g
#     # datetime.now()-pd.to_datetime("21.07.1980") # Steve hocanın yaşadığı gün sayısı
#     # Steve Hoca: Analiz firmasında çalışırsanız böyle şeyler kullanabilirsiniz Mesela
# 3) .strftime() --> date/datetime/timedelta object i string type a dönüştürür
#     # datetime object sonuna .strftime() yazarsam içindeki formattaki ifadeye çevi
#     # year = current_datetime.strftime("%Y")
#     # print("year:", year)
#     # month = current_datetime.strftime("%m")
#     # print("month:", month)
#     # day = current_datetime.strftime("%d")
#     # print("day:", day)
#     # time = current_datetime.strftime("%H:%M:%S")
#     # print("time:", time)
#     # date_time = current_datetime.strftime("%m/%d/%Y, %H:%M:%S")
#     # print("date and time:", date_time)
# 4) .strptime() --> Sadece string type i datetime object e dönüştürür
#     # dt_string1 = "21 June, 2018"
#     # dt_string1 # Output : '21 June, 2018'
#     # datetime.strptime(dt_string1,"%d %B, %Y")
#     # dt_string2 = "12/11/2018 ,&sadsadas/ 09:15:32"
#     # datetime.strptime(dt_string2, "%d/%m/%Y ,&sadsadas/ %H:%M:%S") # dt_string = 11/
#     # print("date_object1 =", dt_string2)

#####
# DATA INPUT & OUTPUT
# 1.CSV INPUT AND OUTPUT
# a.Csv input
#     # pd.read_csv('example.csv')
#     # pd.read_csv("ornekcsv.csv", sep=';', index_col=0)          # index_col : Kaçı
#     # pd.read_csv("ornekcsv.csv", sep = ";", usecols=['a', 'b'], nrows=10) # KOŞUL
#     # pd.read_csv("https://www.stats.govt.nz/assets/Uploads/Annual-enterprise-surve
#     """ import csv
#         reader = csv.DictReader(open('example.csv'))"""
#     # b.Csv input
#         # df.to_csv('example_1.csv')
# 2.EXCEL INPUT AND OUTPUT
# a.Excel Input
#     # df1 = pd.read_excel('Excel_Sample.xlsx')
#     # df2 = pd.read_excel('Excel_Sample.xlsx', sheet_name = "Sheet2")

```

```
# df3 = pd.read_excel('Excel_Sample.xlsx', sheet_name = None)      # sheet_name
# b.Excel Output
    """with pd.ExcelWriter('combined_df.xlsx') as writer:           # For döngüsü
        df1.to_excel(writer, sheet_name='sample1', index=False)
        df2.to_excel(writer, sheet_name='sample2', index=True)"""
# 3.HTML INPUT-OUTPUT
# a.HTML Input
# !pip install lxml
# df = pd.read_html('https://www.bbc.com/news/world-51235105') # read_html: In
# b.HTML output
# df2[0].to_html('simple.html',index=False) # simple html diye yeni html dosyası o

File "C:\Users\cansi\AppData\Local\Temp\ipykernel_18408/3641927388.py", line 59
    """ import csv
 ^
IndentationError: unexpected indent
```

SEARCH STACK OVERFLOW

