Week 6

```
df_name.tail(x) # son x satır
df_name.columns # sütun isimleri
df_name.shape # Tupil Döndürür. >>(satır,sütun)   .shape[0] = (satır)
df_name.info() # df hakkında bilgi verir. Veri tiplerini ve eksik değerleri görmede kullanılır.
df_name.describe() # Verinin sayısal özet istatistiklerini döndürür(ortalama, medyan, min, max, std, vs.). describe().T ile verilerin transpoze edilir.
df_name.sütun_ismi | df_name.["sütun_ismi"] # Sütunu alır. !! ilk seçenek boşluk veya özel karakterleri anlamaz.
df_name.[["sütun1", "sütun2"]] # sütun1 ve sütun2 yi birlikte alır
df_name.["sütun"].str.len()  # stringlerin uzunluğunu verir.
df_name.[df_name.["sütun1"] == "x"] # sütundaki değeri x'e eşit olan tüm değerleri filtreler
df_name.[(df_name.["sütun1"] == "x") & (df_name.["sütun2"] > "y") ]  # x ve y'ye göre filtreler
df_name.["sütun1"].value_counts()  # sütundaki her bir değer kaç kez tekrar eder onu gösterir.   normalize = True olursa her değerin yüzdesini verir.
df_animal.vore.value_counts().index[0] #Vore sütunundaki her Unique value'nun sayısını hesaplayıp Seri olarak geri döndürür ve baştaki indexi verir.
df_name.groupby("x") # x değerine göre gruplar ama returnlemez, ek fonksiyon lazım.
df_name.groupby("sütun2").["sütun1"].count() # sütun2 değerlerine göre gruplayıp sütun1 deki sütun2 değerlerinin sayısını verir. | value.counts() ile ayn
df_name.groupby("sütun2").["sütun1"].size()  #üstteki fonksiyonun NaN değerleri eklenmiş hali
df_name["sütun1"].nunique(dropna = False) # sütun1 deki benzersiz değerleri sayar. default dropna = True'dur ve eksik girilmiş veri NaN değerlerini sayma
np.mean(df_name.sleep_total) | df_name["sleep_total"].mean() # Ortalama hesaplamak için 2 farklı yöntem
np.median(df_name.sleep_total) | df_name["sleep_total"].median() #Veri değerlerinin ortasını bulmak için kulllanılan 2 farklı yöntem

import statistics as stat
stat.mode(df_animal.vore) #en çok tekrar eden değeri (mod) hesaplar.

df_animal[df_animal.vore == "insecti"]["sleep_rem"].agg(["mean", "median"]) # .agg() metodu birden fazla işlemin yapılmasını sağlar
df_animal.loc[len(df_animal.index)]= ["New Insect", "", "insecti", "", "", 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] # database'e yeni element ekler
df_animal.drop([len(df_animal.index) - 1], inplace = True) # son satırı düşürür.
range_sleep_total=df_animal["sleep_total"].max()- df_animal["sleep_total"].min() # max-min
np.var(df_animal.sleep_total, ddof=1) #varyansını, yani dağılımını bulur. ddof = 0 Popülasyon, ddof = 1 Örneklem için kullanılır
np.std(df_animal.sleep_total, ddof=1) # standart sapmayı hesaplıyan kod

dists = df_animal.sleep_total - np.mean(df_animal.sleep_total)
np.mean(np.abs(dists)) # tüm değerlerin ortalamadan ne kadar saptığını dists'e kaydeder. np.abs() ile bu değerlerin mutlak değerini alır.

np.quantile(df_animal.sleep_total, 0.25) # verinin belirli bir yüzdesini hesaplamada kullanılır.
np.quantile(df_animal.sleep_total, [0, 0.25, 0.5, 0.75, 1])

import matplotlib.pyplot as plt
plt.boxplot(df_animal.sleep_total)# verinin kutu grafiği ile dağılımını, ortalamayı, çeyrekleri(Q1, Q3) ve aykırı değerleri(outliers) görselleştirir.
np.quantile(df_animal.sleep_total, np.linspace(0,1,5)) # linspace 0 dan 1 e kadar ki aralığı 5 e böler.

from scipy.stats import iqr
iqr(df_animal.sleep_total) #Çeyrekler arasını hesaplar. (Q3 - Q1).
np.quantile(df_animal.sleep_total, 0.75) - np.quantile(df_animal.sleep_total, 0.25) #alternatif

from scipy.stats import iqr
iqr = iqr(df_animal.bodywt)
lower_threshold = np.quantile(df_animal.bodywt, 0.25)- 1.5 * iqr
upper_threshold= np.quantile(df_animal.bodywt, 0.75) + 1.5 * iqr
df_animal[(df_animal.bodywt < lower_threshold) | (df_animal.bodywt > upper_threshold)] # Outlier bulmak için kullanılır.

import seaborn as sns
sns.boxplot(data=df_animal, y="bodywt") #seaborn boxplot modeli
import plotly.express as px
px.box(df_animal, x="bodywt") #plotly.express için boxplot

df_sales_users = df_sales.groupby("num_users")["amount"].agg(sum="sum") # her user için toplama yapıp toplamaları yeni bir sütun oluşturarak saklıyor.
df_sales_users.sample()  #rasgele örnek bir satırı seçmeyi sağlar

np.random.seed(42)
df_sales_users.sample() # seed belirleyerek seçilen rastgele örneği kaydetmeyi sağlar.
df_sales_users.sample(random_state = 42)

df_sales_users.sample(5, replace=True)  #Sayıyı tekrar yerine koyarak seçim yapar. (5 örnek alır ve bu örnekler aynı olabilir)

from scipy.stats import uniform
uniform.cdf(7, 0, 12) # Sayımız 7, başlangıcımız 0, uzunluğumuz 12 olmak üzere 0<=x<=12 için  0<=x<=7 olasılığını hesaplar.
1 - uniform.cdf(7,0,12) # 7< x <=12 olasılığını hesaplar

#P(wait time > 7)
1-uniform.cdf(7,0,12)

#P(4=<wait time <= 7)
uniform.cdf(7,0,12) - uniform.cdf(4,0,12)

#Uniform Dağılımına göre rastgele sayı üretme.
uniform.rvs(0, 5, size=10) # 0 ila 5 arasında 10 rasgele sayı üretilicek

#Binomda örneğin paranın bir tarafı daha ağır ve %25 yazı ihtimali %75 tura ihtimali var
binom.rvs(3, 0.25, size=10) # 3 kere para atılır. Kaç kere yazı geldiği listeye eklenir. Bu 10 kez tekrarlanır.

#7 sinin başarılı olma olasılığı
binom.pmf(7(başarı sayısı), 10(deneme sayısı) ,0.5 (başarı olasılığı))
# 7 veya 7den az yazı gelme olasılığı
binom.cdf(7, 10, 0.5)
#7 den fazla gelme olasılığı
1 - binom.cdf(7,10,0,5)
```

Week 6

```
from scipy.stats import norm
norm.cdf(değer, ortalama, standart sapma)
norm.cdf ( 154, 161, 7) #154 cm ve aşağısı olanlar
1-norm.cdf(154,161,7)#154 cm den uzun olanlar
#norm.ppf kullanılarak yüzde hesaplanılabilir
norm.ppf(0.9, 161, 7)
#kadınların %90 ı 169.97cm den kısadır
#kadınların %90 ı şu boydan uzundur.
norm.ppf((1-0.9), 161,7)
norm.rvs(161, 7, size=10) #fonksiyonu, normal dağılımdan rastgele örnekler çeker.
die= pd.Series([1,2,3,4,5,6])
samp_5= die.sample(5, replace=True)
samp_5 #5 kere zar atar
sample_means = []
for i in range(10):
    samp_5 = die.sample(5, replace=True)
    sample_means.append(samp_5.mean())
plt.title("Örneklem Ortalamasının Örneklem Dağılımı")
plt.hist(sample_means)
sample_prp = []
for i in range(1000):
    samp_5 = sales_team.sample(10, replace=True)
    try:
        sample_prp.append(samp_5.value_counts()['Claire'] / 10)
    except:
        sample_prp.append(0)
plt.title("Örneklem Oran Örneklem Dağılımı")
plt.hist(sample_prp)
from scipy.stats import poisson
poisson.pmf(5,8) #ortalama 8 kere gerçekleşen olayın 5 kere gerçekleşme olasılığı
poisson.cdf(5,8) # 5 veya daha az gerçekleşme olasılığı
poisson.rvs(8, size=10) #ortalaması 8 olan seriden 10 tane sample alıyor

from scipy.stats import expon
expon.cdf(1, scale=2) # ortalama 2 birimlik sürede gerçekleşen olayın 1 birimlik sürede gerçekleşme olasılığı
sample_size = 5
pop_size = len(df_coffee)
interval = pop_size // sample_size
df_coffee.iloc[interval :: interval]
#########################################################################################################################################################
sample_data = df_election.copy()
sample_data["diff"]= sample_data["repub_percent_08"] - sample_data["repub_percent_12"]
x_bar_diff = sample_data["diff"].mean()

n_diff = len(sample_data)
s_diff = sample_data['diff'].std()
t_stat = (x_bar_diff - 0) / np.sqrt(s_diff**2 / n_diff)

from scipy.stats import t
degrees_of_freedom = n_diff - 1
p_value = t.cdf(t_stat, df=degrees_of_freedom)
pingouin.ttest(sample_data["diff"], y = 0, alternative = "less")
```

Week 7

```
Week 7

p_hat = (df_stck["age_cat"] == "Under 30").mean()
p_0 = 0.5

pay = p_hat - p_0
payda=np.sqrt(p_0*(1-p_0)/n)
z_score= pay/payda
z_score
#########################################################################################################################################################################
n_hobbyist = np.array([812, 1021])
n_rows = np.array([812 + 238, 1021 + 190])
from statsmodels.stats.proportion import proportions_ztest

z_score, p_value = proportions_ztest(count=n_hobbyist, nobs=n_rows,
                                     alternative='two-sided')

(z_score, p_value)
#########################################################################################################################################################################
# 'Under 30' kategorisindeki başarı sayısını bulalım
n_under_30 = (df_stck["age_cat"] == "Under 30").sum()  # 'Under 30' kategorisindeki gözlem sayısı
n_total = len(df_stck)  # Toplam gözlem sayısı

# proportion z-test'i kullanarak z-skorunu hesaplayalım
from statsmodels.stats.proportion import proportions_ztest

# p_0 = 0.5 olduğuna göre, bu parametreyi doğrudan kullanacağız
z_score, p_value = proportions_ztest(count=n_under_30, nobs=n_total, value=0.5, alternative='two-sided')

(z_score, p_value)

#########################################################################################################################################################################
# Veriyi gruplara ayıralım
hobbyist_group = df_stck[df_stck['category'] == 'hobbyist']
non_hobbyist_group = df_stck[df_stck['category'] == 'non_hobbyist']

# Başarı sayısını (1 olanları) ve toplam gözlem sayısını hesaplayalım
n_hobbyist = hobbyist_group['success'].sum()  # hobbyist grubundaki başarı sayısı
n_non_hobbyist = non_hobbyist_group['success'].sum()  # non-hobbyist grubundaki başarı sayısı

# Toplam gözlem sayısını hesaplayalım
n_total_hobbyist = len(hobbyist_group)  # hobbyist grubundaki toplam gözlem sayısı
n_total_non_hobbyist = len(non_hobbyist_group)  # non-hobbyist grubundaki toplam gözlem sayısı

# proportions_ztest fonksiyonunu kullanarak testi yapalım
from statsmodels.stats.proportion import proportions_ztest

z_score, p_value = proportions_ztest(count=[n_hobbyist, n_non_hobbyist],
                                     nobs=[n_total_hobbyist, n_total_non_hobbyist],
                                     alternative='two-sided')

(z_score, p_value)

#########################################################################################################################################################################
```

```
import pingouin

expected, observed, stats = pingouin.chi2_independence(data=df_stck, x='hobbyist',
                                                       y='age_cat',
                                                       correction=False)
stats

props = df_stck.groupby('age_cat')['job_sat'].value_counts(normalize=True)
wide_props = props.unstack()
wide_props.plot(kind='bar', stacked=True)

purple_link_counts= df_stck["purple_link"].value_counts()

purple_link_counts = purple_link_counts.rename_axis('purple_link').\
                                        reset_index(name='n').\
                                        sort_values('purple_link')
hypothesized = pd.DataFrame({
    'purple_link' : ['Amused', 'Annoyed', 'Hello, old friend', 'Indifferent'],
    'prop' : [1/6, 1/6, 1/2, 1/6]
}
)
n_total = len(df_stck)
hypothesized["n"] = hypothesized["prop"]* n_total

from scipy.stats import chisquare
chisquare(f_obs=purple_link_counts['n'], f_exp=hypothesized['n'])
#########################################################################################################################################################################
pingouin.anova(data=df_stck, dv="converted_comp", between="job_sat")
pingouin.pairwise_tests(data=df_stck, dv = "converted_comp",between = "job_sat",
                        padjust = "bonf")
```

Week 8

```
Week 8
```

```
from scipy.stats import rankdata
df_small['rank_abs_diff'] = rankdata(df_small['abs_diff'])

#Parametrik olmayan testler -> Normal dağılım yok, Örneklem küçük en az 30
pingouin.wilcoxon(x=df_small['repub_percent_08'],   #Eşleştirilmiş veri
                  y=df_small['repub_percent_12'],
                  alternative='less')

age_vs_comp = df_stck[['converted_comp', 'age_first_code_cut']]
age_vs_comp_wide = age_vs_comp.pivot(columns='age_first_code_cut',
                                     values='converted_comp')

pingouin.mwu(x=age_vs_comp_wide['child'], #Eşleştirilmemiş veri
             y=age_vs_comp_wide['adult'],
             alternative='greater')
#########################################################################################################################################################################
from sklearn.neighbors import KNeighborsClassifier
df_churn = pd.read_csv("data/telecom_churn_clean.csv")
X = df_churn[['total_day_charge', 'total_eve_charge']].values
y= df_churn['churn'].values
(X.shape, y.shape)
#Aynı row sayısı olmak zorunda

model_knn = KNeighborsClassifier(n_neighbors=15)
model_knn.fit(X,y)
#########################################################################################################################################################################
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)

#########################################################################################################################################################################
train_accuracies = {}
test_accuracies = {}
neighbours = np.arange(1, 26)
for neighbour in neighbours:
    knn = KNeighborsClassifier(n_neighbors=neighbour)
    knn.fit(X_train, y_train)
    train_accuracies[neighbour] = knn.score(X_train, y_train)
    test_accuracies[neighbour] = knn.score(X_test, y_test)

plt.figure(figsize=(8, 6))
plt.title('KNN : Değişen Komşu Sayılarında')
plt.plot(neighbours, train_accuracies.values(), label='Eğitim Doğruluğu')
plt.plot(neighbours, test_accuracies.values(), label='Test Doğruluğu')

#########################################################################################################################################################################
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df_diabets = pd.read_csv('data/diabetes_clean.csv')
df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) | (df_diabets['glucose'] == 0)]
df_diabets.drop(df_filtered.index, inplace=True)
X = df_diabets.drop('glucose', axis=1).values
y = df_diabets['glucose'].values

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)
model_reg = LinearRegression()
model_reg.fit(X_train, y_train)

y_pred = model_reg.predict(X_test)
model_reg.score(X_test, y_test)
model_reg.intercept_ #Kesişim noktası
model_reg.coef_ #Eğimi verir
```

```
from sklearn.metrics import root_mean_squared_error, r2_score, accuracy_score
root_mean_squared_error(y_test, y_pred)
#########################################################################################################################################################################
# Generate random data for the scatter plot
np.random.seed(42)
X = np.linspace(0, 10, 20)
Y = 3 * X + 5 + np.random.normal(0, 3, size=len(X))

# Fit a line (y = mx + c) manually
m = 3  # slope
c = 5  # intercept
Y_fit = m * X + c
# Calculate residuals (vertical distances between points and the line)
residuals = Y - Y_fit

# Add vertical lines to represent residuals
for i in range(len(X)):
    plt.plot([X[i], X[i]], [Y[i], Y_fit[i]], color='gray', linestyle='dotted')
    if i == 4:  # Add a label to one residual line
        plt.text(X[i], (Y[i] + Y_fit[i]) / 2, 'Residual', color='black', fontsize=10, ha='right')
```

Week 9

```
Week 9

from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
X = df_diabets.drop("glucose", axis = 1).values
y =df_diabets["glucose"].values

kf= KFold(n_splits = 6, shuffle = True, random_state = 42) #Eşit parçalara böl testing olarak dene
model_reg = LinearRegression()
cv_results = cross_val_score(model_reg, X, y, cv=kf)
np.quantile(cv_results, [0.025, 0.975])
(np.mean(cv_results), np.std(cv_results))

#########################################################################################################################################################
#  Alfa arttıkça performansın kötüleştiğini görüyoruz.

from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
X = df_diabets.drop('glucose', axis=1).values
y = df_diabets['glucose'].values
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2,
                                                   random_state=42)
scores = []
for alpha in [0.1, 1.0, 10.0, 100.0, 1000.0]:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
    scores.append(ridge.score(X_test, y_test))

from sklearn.linear_model import Lasso
scores = [] #  Alfa 20'nin üzerine çıktıkça performans önemli ölçüde düşer!
for alpha in [0.1, 1.0, 10.0, 20.0, 50.0]:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    y_pred = lasso.predict(X_test)
    scores.append(lasso.score(X_test, y_test))

&&&&&&&&&&
scores=[]
names = df_diabets.drop('glucose', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
plt.bar(names, lasso_coef)
plt.xticks(rotation=45)
#########################################################################################################################################################
from sklearn.metrics import classification_report, confusion_matrix
df_churn = pd.read_csv("data/telecom_churn_clean.csv")
X = df_churn[['total_day_charge', 'total_eve_charge']].values
y= df_churn['churn'].values
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

confusion_matrix(y_test, y_pred)
classification_report(y_test, y_pred)

&&&&&&&&&&&
model_log = LogisticRegression()
model_log.fit(X_train, y_train)

y_predict = model_log.predict(X_test) #0 veya 1 sonucunu verir
y_pred_probs = model_log.predict_proba(X_test) # 0'a ait olma, 1'e ait olma olasılığı
y_pred_probs = y_pred_probs[:,1] #Sadece 1 ile alakalı değerleri al
```

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_pred_probs)
#########################################################################################################################################################
```

Week 9

```
#########################################################################################################################################################################
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
X = df_diabets.drop('glucose', axis=1).values
y = df_diabets['glucose'].values
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)

kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {
    'alpha': np.arange(0.0001, 1, 10),
    'solver':['sag', 'lsqr']
}

ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf)
ridge_cv.fit(X_train, y_train)
from sklearn.model_selection import RandomizedSearchCV

#  n_iter iki olarak ayarlandığında
# beş katlı çapraz doğrulama 10 fit() gerçekleştirir.
ridge_cv = RandomizedSearchCV(ridge, param_grid, cv=kf, n_iter=2)
ridge_cv.fit(X_train, y_train)
(ridge_cv.best_params_, ridge_cv.best_score_)
#########################################################################################################################################################################
music_dummies = pd.get_dummies(df_music['music_genre'], drop_first=True, dtype='int')
music_dummies = pd.concat([df_music, music_dummies], axis=1)
music_dummies = music_dummies.drop(['music_genre', 'instance_id'], axis=1)

from sklearn.model_selection import cross_val_score, KFold, train_test_split
X = music_dummies.drop('popularity', axis=1).values
y = music_dummies['popularity'].values
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)
kf = KFold(n_splits=5, shuffle=True, random_state=42)

model_reg = LinearRegression()
model_reg_cv = cross_val_score(model_reg, X_train, y_train, cv=kf,
                               scoring='neg_mean_squared_error')
#cross_val_score negatif değer döndürür
np.sqrt(-model_reg_cv)
```

Week 10

```
Week 10

df_house = pd.read_csv("data/AmesHousing.csv")
df_house.info()
df_house.isna().sum().sort_values(ascending = False)
na_series = df_house.isna().sum()
data_len = len(df_house) * 0.05
na_series[(na_series < data_len) & (na_series != 0)]
col_names = list(na_series[(na_series <= data_len) & (na_series != 0)].keys())
df_house = df_house.dropna(subset=col_names)
object_cols = list(df_house.select_dtypes(include='object').columns)
#^Kategorik verileri alır
X_cat = df_house[object_cols]
X_nums = df_house.drop(object_cols, axis=1)
y = X_nums['SalePrice'].values.reshape(-1, 1)
X_nums.drop('SalePrice', inplace=True, axis=1)

from sklearn.model_selection import train_test_split
X_train_cat, X_test_cat, y_train_cat, y_test_cat =train_test_split(X_cat, y, test_size=0.2, random_state=42)
X_train_nums, X_test_nums, y_train_nums, y_test_nums =train_test_split(X_nums, y, test_size=0.2, random_state=42)

from sklearn.impute import SimpleImputer
imp_cat = SimpleImputer(strategy = "most_frequent")
X_train_cat = imp_cat.fit_transform(X_train_cat)
X_test_cat = imp_cat.fit_transform(X_test_cat)

imp_num = SimpleImputer()
X_train_nums = imp_num.fit_transform(X_train_nums)
X_test_nums = imp_num.fit_transform(X_test_nums)

X_train = np.append(X_train_nums, X_train_cat, axis=1)
X_test = np.append(X_test_nums, X_test_cat, axis=1)
#########################################################################################################################################################
from sklearn.pipeline import Pipeline

df_music = df_music.dropna(subset=['genre', 'popularity', 'loudness', 'liveness', 'tempo'])
X = df_music.drop('genre', axis=1).values
y = df_music['genre'].values
# Bir ardışık düzende, son adım hariç her adımın bir dönüştürücü olması gerektiği unutulmamalıdır
from sklearn.linear_model import LogisticRegression
steps = [
    ('imputation', SimpleImputer()),
    ('logistic_regression', LogisticRegression())
]

pipeline = Pipeline(steps=steps)
X_train, X_test, y_train, y_test =train_test_split(X, y,
                                                   test_size=0.2, random_state=42)
pipeline.fit(X_train, y_train)
pipeline.score(X_test, y_test)
#########################################################################################################################################################
```

```
from sklearn.preprocessing import StandardScaler
df_music = pd.read_csv('data/music_genre.csv')
X = df_music.drop('music_genre', axis=1).values
y = df_music['music_genre'].values
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

print(np.mean(X), np.std(X))
print(np.mean(X_train_scaled), np.std(X_train_scaled))
&&&&&&&&&&&&&&&&&&
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
steps = [
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier(n_neighbors=6))
]
pipeline = Pipeline(steps)

knn_scaled = pipeline.fit(X_train, y_train)
y_pred = knn_scaled.predict(X_test)

knn_scaled.score(X_test, y_test)
knn_unscaled = KNeighborsClassifier(n_neighbors=6).fit(X_train, y_train)
knn_unscaled.score(X_test, y_test)
&&&&&&&&&&&&&&&&&&&&&&&&
```

Week 10

```
&&&&&&&&&&&&&&&&&&&&&&&&
from sklearn.model_selection import GridSearchCV
import numpy as np
steps = [
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
]
pipeline = Pipeline(steps)
parameters = {"knn__n_neighbors" : np.arange(1, 50)} #Bu satıra dikkat et

cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(X_train, y_train)

(cv.best_score_, cv.best_params_)
#########################################################################################################################################################################
# Sınıflandırma modellerinin değerlendirilmesi
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

df_music = pd.read_csv("data/music_clean.csv")

X = df_music.drop('genre', axis=1).values
y = df_music['genre'].values

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

models = {
    "Logistic Regression" : LogisticRegression(),
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier()
}
results = []
for model in models.values():
    kf = KFold(n_splits=6, random_state=True, shuffle=True)
    cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kf)
    results.append(cv_results)

plt.boxplot(results, labels=models.keys())

# test performance
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    test_score = model.score(X_test_scaled, y_test)
    print(f"{name} Test Set Accuracy : {test_score} ")
```