

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق

درس یادگیری ماشین

استاد دکتر علیاری

سید محمد رضا حسینی

شماره دانشجویی: 40204584

گرایش: سیستم های الکترونیک دیجیتال

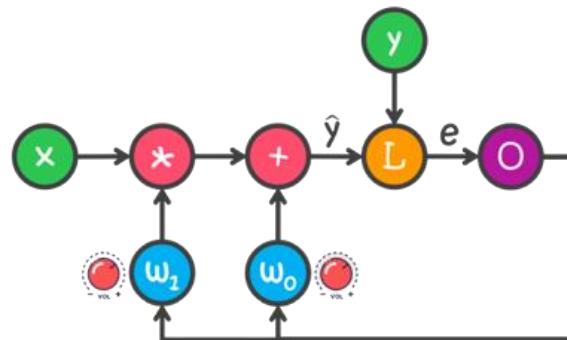
مینی پروژه شماره 1

[Google Colab](#)

[Github](#)

1. سوال اول

1.1. بخش اول



همانطور که مشاهده می شود، ویژگی ها (X) به عنوان ورودی داده می شوند. خروجی پیش‌بینی شده به صورت $wX + w_0$ ساخته می شود. حال این کلاس پیش‌بینی شده (\hat{y}) با تابع هزینه مناسب با کلاس واقعی (y) توسط یک تابع هزینه مناسب مقایسه می شود (e). به کمک الگوریتم های بهینه سازی (O) سعی می شود که w و w_0 به گونه ای تغییر داده شوند که میزان تابع هزینه کمتر شود. این w و w_0 ساخته شده مجددا به سیستم داده می شوند تا در مرحله بعدی از آنها استفاده می شود.

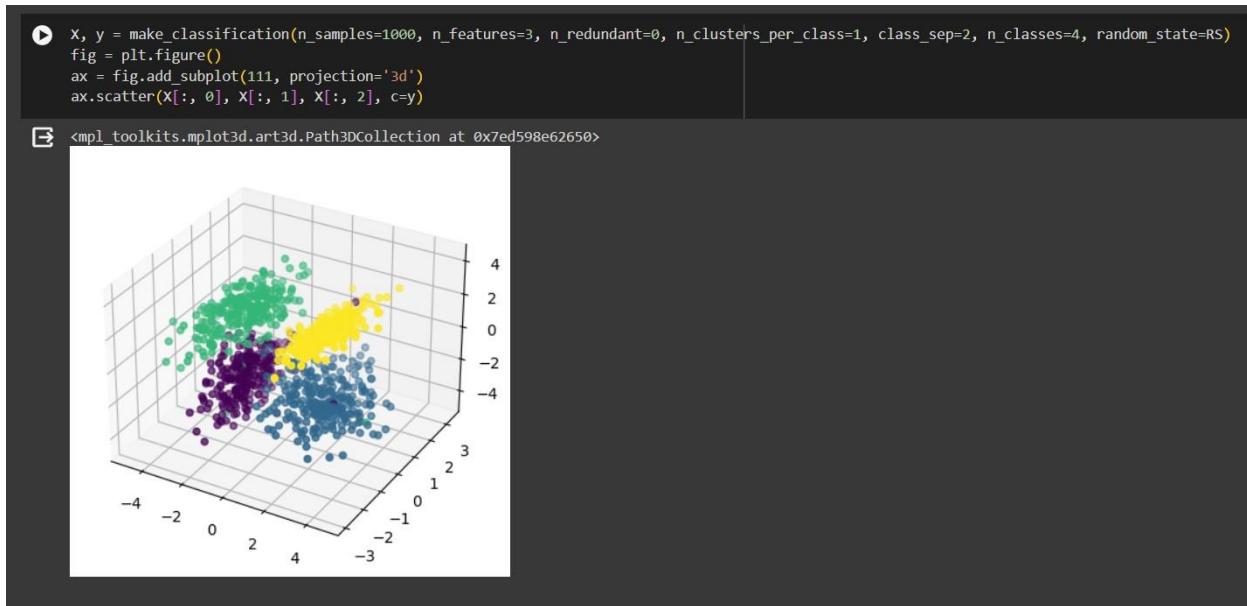
در حالت چند کلاسه، ما به جای یک طبقه بند خطی، چند طبقه بند داریم. به این شکل که یا باید تک تک کلاس ها را در مقابل بقیه کلاس ها قرار دهیم و چند طبقه بند دو کلاسه بدست آوریم یا باید کلاس ها را دو به دو با هم درنظر بگیریم و برای هر 2 کلاس متفاوت، یک طبقه بند قرار دهیم.

1.2. بخش دوم

در ابتدای کد import های مورد نیاز را انجام می دهیم.

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
```

در ادامه با توجه به خواسته سوال، دیتاست مورد نظر تولید شد. با توجه به اینکه در این سوال 3 ویژگی وجود دارد، دیتاست را در نمودار 3 بعدی نمایش دادیم.

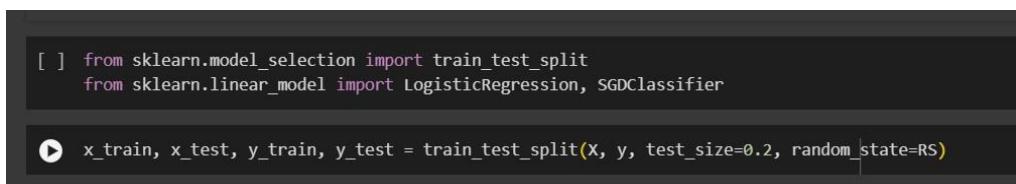


عدد `class_sep` نشان دهنده میزان چالش برانگیز بودن دیتاست است. هر چه این عدد کوچکتر باشد، کلاس‌ها بیشتر به داخل هم فرو رفته‌اند و در نتیجه کار جداسازی آن‌ها سخت‌تر و چالش برانگیزتر است. به مانند تمرین حل شده در کلاس حل تمرین، در این سوال هم ما عدد `class_sep` را 2 در نظر گرفتیم. با توجه به اینکه کلاس‌ها تقریباً به خوبی از هم جدا شده‌اند، این دیتاست چندان چالش برانگیز نیست. برای چالش برانگیزتر کردن آن می‌توان `class_sep` را عددی کوچکتر مانند 0.5 قرار داد.

1.3. بخش سوم

در ابتدای کد `import`‌های مورد نیاز را انجام می‌دهیم.

به کمک تابع `train_test_split`، دیتا را به دو قسمت `train` و `test` تقسیم می‌کنیم. 20 درصد برای تست و 80 درصد برای یادگیری قرار می‌دهیم.



در حالت اول از `LogisticRegression` استفاده کردیم. برای بهبود عملکرد پارامترهای مختلفی را تغییر دادیم. به طور مثال، `RandomState` را تغییر دادیم اما تغییر مثبتی حاصل نشد. همچنین با توجه به توضیحات صفحه [صفحه](#) در سایت `scikit-learn`، سایر پارامترها (مانند `C` و `solver` و ...) را تغییر دادیم اما تغییر مثبتی

حاصل نشد. همچنین تعداد تکرار یا همان `max_iter` را تا 20000 بالا بردیم اما بهبودی حاصل نشد. بنظر می‌رسد که طبقه بند همگرا شده است.

The screenshot shows a Jupyter Notebook cell titled "Logistic Regression". The code defines a logistic regression model with a maximum iteration of 200. The output shows the model's performance on training and test data, with scores of 0.98375 and 0.98 respectively.

```
[ ] model1 = LogisticRegression(solver='sag', max_iter=200, random_state=RS)
model1.fit(x_train, y_train)
print("Logistic Regression")
print(" Train data score:",format(model1.score(x_train, y_train)))
print(" Test data score:",format(model1.score(x_test, y_test)))
```

```
Logistic Regression
Train data score: 0.98375
Test data score: 0.98
```

در حالت دوم از `SGDClassifier` استفاده کردیم. در این حالت هم مانند حالت قبل پارامترهای مختلف را با توجه به توضیحات [صفحه SGDClassifier](#) در سایت `scikit-learn` تغییر دادیم و بهترین حالت را (که `RandomState` برابر با 80 دارد) در کد قرار دادیم.

The screenshot shows two Jupyter Notebook cells for `SGDClassifier`. The first cell uses a random state of 80, resulting in a train score of 0.9625 and a test score of 0.965. The second cell uses a random state of 200, resulting in a train score of 0.96375 and a test score of 0.965.

```
▶ model2 = SGDClassifier(loss='log_loss', random_state=RS)
model2.fit(x_train, y_train)
print("SGDClassifier")
print(" Train data score:",format(model2.score(x_train, y_train)))
print(" Test data score:",format(model2.score(x_test, y_test)))
```

```
SGDClassifier
Train data score: 0.9625
Test data score: 0.965
```



```
[ ] model2 = SGDClassifier(loss='log_loss', random_state=80)
model2.fit(x_train, y_train)
print("SGDClassifier")
print(" Train data score:",format(model2.score(x_train, y_train)))
print(" Test data score:",format(model2.score(x_test, y_test)))
```

```
SGDClassifier
Train data score: 0.96375
Test data score: 0.965
```

1.4. بخش چهارم

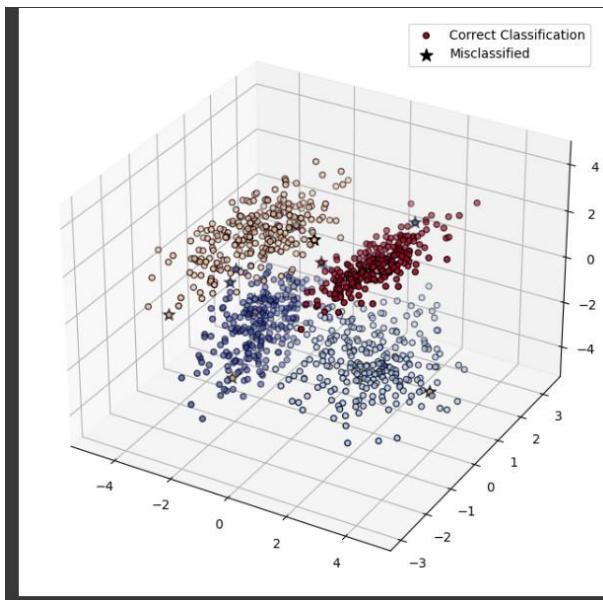
در این قسمت دیتاهای در یک نمودار سه بعدی با توجه به کلاس خود رسم می‌شوند. آن دسته از دیتاهایی که اشتباہ پیشینی شده اند با ستاره مشخص شده اند. به دلیل اینکه تعداد ویژگی‌ها 3 است، مرزهای تصمیم گیری 2 بعدی و به شکل صفحه هستند که به دلیل پیچیدگی، موفق به رسم این صفحه نشدیم.

ابتدا `Logistic Regression` انجام شد.

```
from mpl_toolkits.mplot3d import Axes3D
```

Logistic Regression

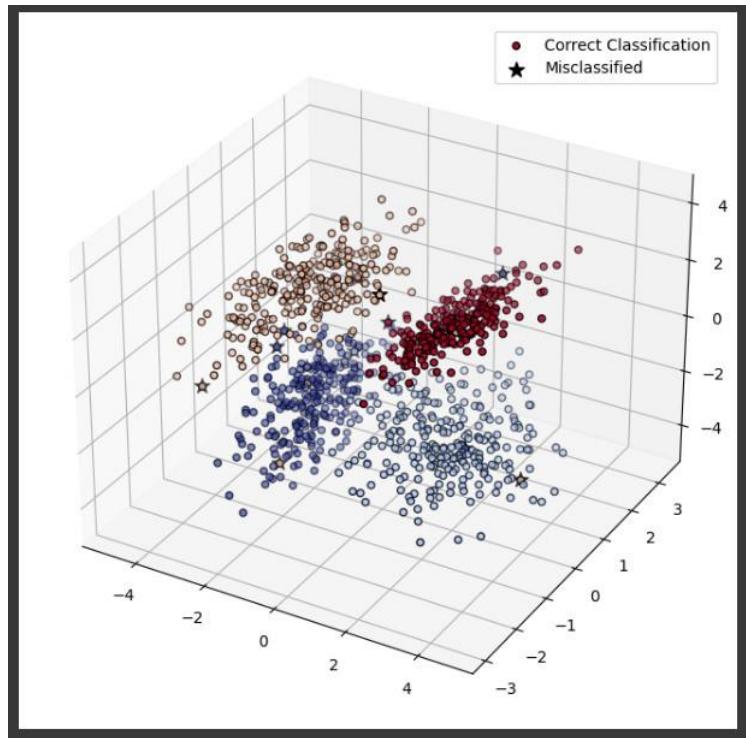
```
[ ] predicted = model1.predict(X)
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k', label='Correct Classification')
misclassified = X[y != predicted]
ax.scatter(misclassified[:, 0], misclassified[:, 1], misclassified[:, 2], c='black', marker='*', s=100, label='Misclassified')
plt.legend()
plt.show()
```



سپس SGDClassifier انجام شد.

```
SGDClassifier
```

```
[ ] predicted = model1.predict(X)
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k', label='Correct Classification')
misclassified = X[y != predicted]
ax.scatter(misclassified[:, 0], misclassified[:, 1], misclassified[:, 2], c='black', marker='*', s=100, label='Misclassified')
plt.legend()
plt.show()
```



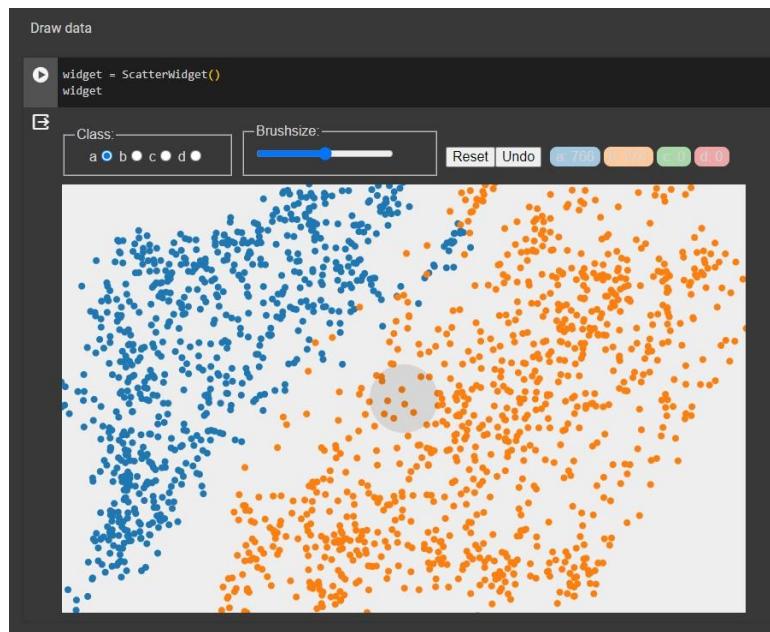
بخش پنجم 1.5

ابتدا موارد مورد نیاز را نصب می کنیم.

```
Install drawdata

[ ] !pip install drawdata
!pip install pandas polars
from drawdata import ScatterWidget
```

سپس یک دیتاست دو کلاسه با دو ویژگی را به کمک `rawdata`, رسم می کنیم.

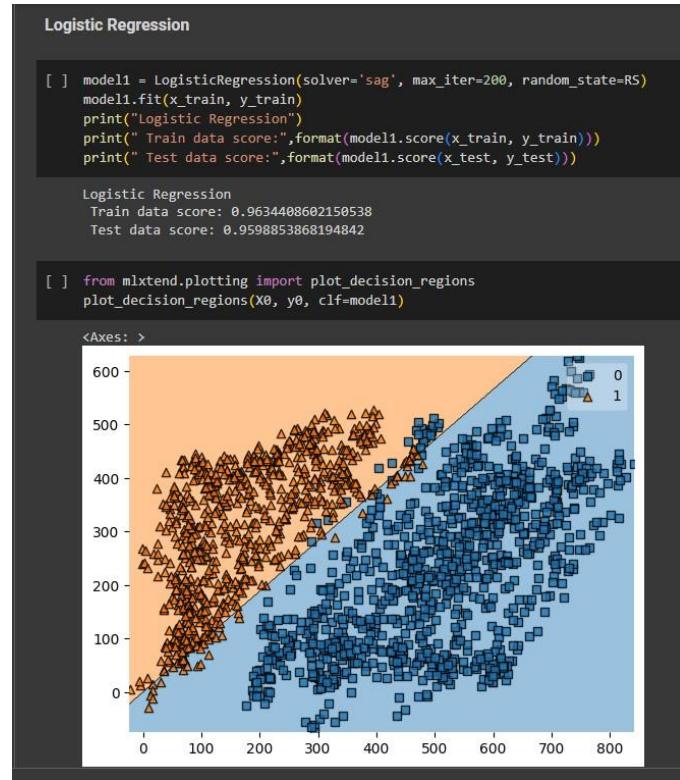


حال مقدار X و y را دریافت می کنیم و دیتا را به دو بخش آموزش و تست تقسیم می کنیم.

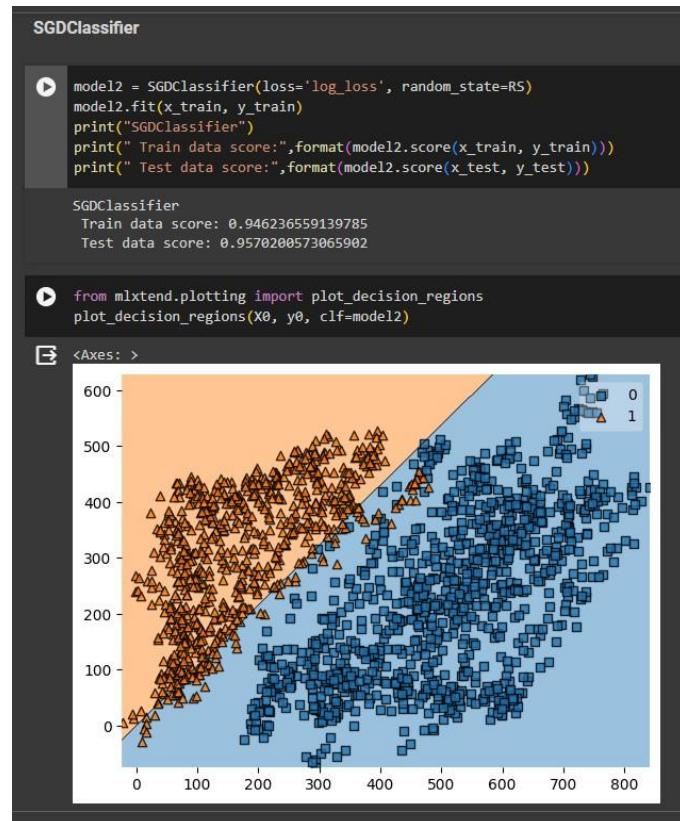
```
Making X and y
[ ] data = widget.data_as_pandas
x1 = data['x'].values
x2 = data['y'].values
X0 = np.concatenate([x1.reshape(-1, 1), x2.reshape(-1, 1)], axis=1)
o = data["label"].values
y0 = np.where(o == 'a', 1, 0)

Making train and test data
▶ x_train, x_test, y_train, y_test = train_test_split(X0, y0, test_size=0.2, random_state=RS)
```

در مرحله اول به کمک Logistic Regression دو کلاس را از یکدیگر جدا می کنیم. Score را نمایش می دهیم و دیتابست، مرز و کلاس ها را نمایش می دهیم.



در مرحله بعد نيز تمام کارهایي که با SGDClassifier انجام دادیم را با Logistic Regression انجام می دهیم.



2. سوال دوم

2.1. بخش اول

در دیتاست مربوط به حوزه CWRU Bearing، مجموعه‌ای از داده‌های ارتعاش بلبرینگ‌ها جمع‌آوری شده است، که از سه نوع بلبرینگ مختلف (بلبرینگ‌های معمولی، بلبرینگ‌های انتهایی درایو و بلبرینگ‌های انتهایی فن) و از حالت‌های عادی و عیوب‌دار آن‌ها، نمونه‌هایی جمع‌آوری شده است.

در این داده‌ها، متغیرهای مختلفی از جمله داده‌های ارتعاش انتهایی فن و انتهایی درایو، سرعت چرخش موتور، داده‌های شتابنگاری پایه، و داده‌های سری زمانی موجود است. فرمت فایل‌ها به صورت مطلب می‌باشد که این فرمت سازگاری و آسانی در تجزیه و تحلیل داده‌ها را فراهم می‌کند.

هدف اصلی این دیتاست، بهبود قابلیت اطمینان دستگاه‌های مکانیکی با شناسایی دقیق و طبقه‌بندی عیوب بلبرینگ است. با تحلیل داده‌ها و الگوهای ارتعاش مرتبط، متخصصان قادرند عیوب بلبرینگ را به دقت شناسایی و طبقه‌بندی کنند و از طریق این شناسایی، بهبود استراتژی‌های نظارت بر وضعیت، توسعه و اعتبارسنجی الگوریتم‌های تشخیص عیوب، بهبود مدل‌های نگهداری پیش‌بینی و بررسی اثرات عیوب مختلف بر ویژگی‌های ارتعاش را انجام دهند.

دیتاست CWRU Bearing به محققان امکان می‌دهد الگوریتم‌های تشخیصی را توسعه دهند و بهبود بخشنده، مدل‌های نگهداری را بهینه‌سازی کنند، و به دقت عیوب بلبرینگ را از طریق تجزیه و تحلیل دقیق الگوهای ارتعاش و ویژگی‌های مرتبط شناسایی کنند.

با توجه به شماره دانشجویی 40204584، برای داده نرمال از Normal_0 و برای داده خطا از IR007 استفاده شده است.

در ابتدای کد، import های مورد نیاز را انجام می‌دهیم. همچنین به کمک دستور gdown دیتاست‌ها را که قبلاً در Google Drive به صورت public قرار داده ایم، فرا می‌خوانیم.

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
!pip install scipy
import scipy.io as sio

Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dist-packages

[2]: !pip install --upgrade --no-cache-dir gdown
!gdown 1Qyn2eK3F28M4UTTpYneH-KcMdVqFCG
!gdown 1BKKcpTNN7cnLlqWjGp73F2TaoTQkCeNR

Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages
```

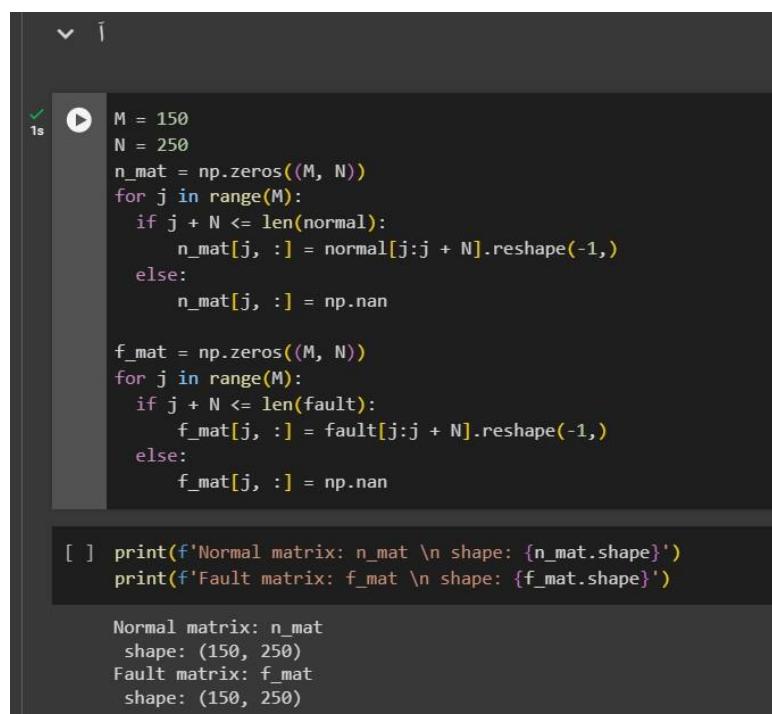
حال دیتای مورد نیاز که شامل X105_DE_time و X097_DE_time است را از دیتاست استخراج می‌کنیم.

```
[3]: data_norm = sio.loadmat('97.mat')
normal = data_norm['X097_DE_time']
data_fault = sio.loadmat('105.mat')
fault = data_fault['X105_DE_time']
```

2.2. بخش دوم

2.2.1.1 قسمت آ

یک ماتریس 150×250 برای داده نرمال (n_mat) و یک ماتریس 150×250 برای داده خطأ (f_mat) تشکیل می‌دهیم. (مقدار M را 150 و مقدار N را 250 در نظر گرفتیم). به این منظور برای تشکیل هر یک از ماتریس‌ها یک حلقه for ایجاد کردیم که N تا N تا دیتا برمی‌دارد و در ماتریس قرار می‌دهد.



```

M = 150
N = 250
n_mat = np.zeros((M, N))
for j in range(M):
    if j + N <= len(normal):
        n_mat[j, :] = normal[j:j + N].reshape(-1, )
    else:
        n_mat[j, :] = np.nan

f_mat = np.zeros((M, N))
for j in range(M):
    if j + N <= len(fault):
        f_mat[j, :] = fault[j:j + N].reshape(-1, )
    else:
        f_mat[j, :] = np.nan

[ ] print(f'Normal matrix: n_mat \n shape: {n_mat.shape}')
print(f'Fault matrix: f_mat \n shape: {f_mat.shape}')

Normal matrix: n_mat
shape: (150, 250)
Fault matrix: f_mat
shape: (150, 250)

```

2.2.1.2 قسمت ب

بیشترین اهمیت استخراج ویژگی‌ها در یادگیری ماشین به این دلیل است که:

افزایش دقیق مدل: با انتخاب ویژگی‌های مناسب و حذف ویژگی‌های غیرضروری، دقیق مدل‌های یادگیری ماشین افزایش می‌یابد.

کاهش ابعاد داده: با استخراج ویژگی‌های مهم و کاهش ابعاد داده، فرآیند یادگیری سریع‌تر و موثرتر می‌شود و همچنین حجم داده‌های پردازشی کاهش می‌یابد که این باعث بهینه‌سازی استفاده از منابع محاسباتی می‌شود.

کاهش تأثیر نویز: با حذف داده‌های تکراری و نویزها، مدل‌های یادگیری ماشین به دقیق بیشتری دست می‌یابند و توانایی تعمیم بهتری دارند.

افزایش تفہیم قابلیت‌های داده: با استخراج ویژگی‌ها، محلول‌های یادگیری ماشین قابلیت تفسیر و فهم داده‌ها را افزایش می‌دهند که این امر می‌تواند در تصمیم‌گیری‌های مربوط به کسب و کار مفید باشد.

افزایش توانایی تعمیم: با استفاده از ویژگی‌های مناسب، مدل‌های یادگیری ماشین توانایی تعمیم بهتری به داده‌های جدید را دارا می‌شوند و عملکرد بهتری در مواجهه با داده‌های ناشناخته ارائه می‌دهند.

در کل، استخراج ویژگی از داده‌های خام به مدل‌های یادگیری ماشین اجازه می‌دهد تا به طور کارآمد عمل کنند و پیش‌بینی‌های دقیقی ارائه دهند.

مانند قسمت قبل ابتدا، import مورد نیاز را انجام می‌دهیم. سپس یک کلاس به نام Features تعریف می‌کنیم. این کلاس 2 متده است:

1. متده `__init__` که برای راه اندازی و مقداردهی اولیه کلاس است و ماتریس ورودی را ذخیره می‌کند.
2. متده `extract` که ویژگی‌ها را از ماتریس استخراج می‌کند و در کلاس ذخیره می‌کند.

به کمک این کلاس ویژگی‌ها را از 2 ماتریس نرمال و خطأ استخراج می‌کنیم.

```
[6] from scipy import stats

class Features:
    def __init__(self, matrix):
        self.matrix = matrix

    def extract(self):
        self.features = {
            'standard deviation': stats.tstd(self.matrix, axis=1),
            'peak': np.max(np.abs(self.matrix), axis=1),
            'skewness': stats.skew(self.matrix, axis=1),
            'kurtosis': stats.kurtosis(self.matrix, axis=1),
            'crest factor': np.max(np.abs(self.matrix), axis=1) / np.sqrt(np.mean(np.square(self.matrix), axis=1)),
            'clearance factor': np.max(np.abs(self.matrix), axis=1) / np.square(np.mean(np.sqrt(np.abs(self.matrix)), axis=1)),
            'peak to peak': np.max(self.matrix, axis=1) - np.min(self.matrix, axis=1),
            'square mean root': np.square(np.mean(np.sqrt(np.abs(self.matrix))), axis=1),
            'mean': np.mean(self.matrix, axis=1),
            'absolute mean': np.mean(np.abs(self.matrix), axis=1),
            'root mean square': np.sqrt(np.mean(np.square(self.matrix), axis=1)),
            'impulse factor': np.max(np.abs(self.matrix), axis=1) / np.mean(np.abs(self.matrix), axis=1),
        }

n_features = Features(n_mat)
n_features.extract()
f_features = Features(f_mat)
f_features.extract()
```

حال که 12 ویژگی ساختیم، لازم است یک ستون به نام label جهت تشخیص کلاس دیتا اضافه کنیم. سپس این

دو دیتابست را به هم می‌چسبانیم تا دیتای نهایی ما بدست آید.

```
Add label to data. 0 for normal and 1 for fault data

n_df = pd.DataFrame(n_features.features)
f_df = pd.DataFrame(f_features.features)
n_df['label'] = np.zeros((n_df.shape[0],), dtype='int8')
f_df['label'] = np.ones((f_df.shape[0],), dtype='int8')
df = pd.concat([n_df, f_df], ignore_index=True)
df.shape
```

(300, 13)

2.2.1.3 قسمت ج

اهمیت فرایند بر زدن:

1. جلوگیری از تعصب: بر زدن داده‌ها کمک می‌کند تا جلوی تعصب در یادگیری گرفته شود و هر دسته یا تکرار آموزش شامل نمایندگی متنوعی از داده باشد.
2. بهینه‌سازی نزول گرادیان تصادفی (SGD): بر زدن داده‌ها به خصوص هنگام استفاده از الگوریتم‌های بهینه‌سازی تصادفی مانند SGD اهمیت دارد. این الگوریتم‌ها بر روی تصادفی بودن برای فرار از کمینه‌های محلی و پیدا کردن کمینه سراسری تابع از دست رفتگی وابسته‌اند.
3. تصادفی بودن داده‌های آموزشی: بر زدن داده‌ها اطمینان می‌دهد که ترتیب نمونه‌های داده بر روی فرآیند یادگیری تأثیر نگذارد و مدل به طور ناخودآگاه الگوهایی بر اساس ترتیب داده‌ها یاد نگیرد.
4. کاهش بیش‌پرازش: بر زدن داده‌ها می‌تواند با واریانس در فرآیند آموزش، به کاهش بیش‌پرازش کمک کند و مدل را مجبور به یادگیری نمایندگی‌های قوی‌تری از داده کند.
5. تعمیم بهتر: بر زدن به مدل کمک می‌کند تا به داده‌های ناشناخته بهتری تعمیم بدهد و از بیش‌پرازش جلوگیری کند. در نهایت، بر زدن داده‌ها یک مرحله پیش‌پردازش ضروری است که به بهبود کارایی و کارآیی مدل‌های یادگیری ماشین کمک می‌کند.

مانند قسمت‌های قبل در این قسمت نیز برای تقسیم دیتا از `train_test_split` استفاده شد و 20 درصد برای تست در نظر گرفته شد. با این فرق که این بار بر زدن را فعال کردیم.

```
[ ] from sklearn.model_selection import train_test_split  
[ ] x_train, x_test, y_train, y_test = train_test_split(df.drop('label', axis=1, inplace=False).values, df.label.values, test_size = 0.2, shuffle = True, random_state = RS)
```

2.2.1.4 قسمت د

نرمال‌سازی داده‌ها یک فرایند مهم در پیش‌پردازش داده‌های مورد استفاده در ماشین لرنینگ است. این فرایند به معنای تغییر مقیاس داده‌ها به گونه‌ای است که مقادیر آنها در محدوده خاصی قرار گیرند، معمولاً بین ۰ و ۱ یا با میانگین صفر و واریانس یک. این کار مزایای زیادی دارد که شامل موارد زیر می‌شود:

1. جلوگیری از تأثیر مقیاس متفاوت ویژگی‌ها: ویژگی‌های با مقیاس‌های مختلف ممکن است تأثیر متفاوتی در مدل داشته باشند. با نرمال‌سازی داده‌ها، تأثیر این اختلافات مقیاس کاهش می‌یابد و مدل می‌تواند بهتر و کارآمدتر با ویژگی‌ها برخورد کند.
2. کاهش آثار نوافه و داده‌های پرت: در برخی موارد، داده‌های پرت یا نوافه می‌توانند تأثیر منفی بر عملکرد مدل داشته باشند. نرمال‌سازی داده‌ها می‌تواند به کاهش این تأثیرات کمک کند و مدل را از پایداری و قابلیت عمومی بیشتری برخوردار سازد.

برای نرمال‌سازی داده‌ها، دو روش رایج عبارتند از:

StandardScaler

این روش داده‌ها را به گونه‌ای نرمال می‌کند که میانگین آنها صفر و واریانس آنها یک شود. فرمول استفاده شده برای نرمال‌سازی داده‌ها به صورت زیر است:

$$z = \frac{x - \mu}{\sigma}$$

که در آن:

- X مقدار اولیه داده

- μ میانگین داده‌ها

- σ انحراف معیار داده‌ها

- Z مقدار نرمال شده

است.

این روش به طور ویژه در الگوریتم‌هایی مانند SVM و نوع‌های خاصی از روش‌های خوشبندی که بر اساس فاصله کار می‌کنند، مفید است. اهمیت استفاده از این روش این است که با استفاده از مقیاس یکسان برای ویژگی‌ها، مدل می‌تواند بهتر و سریع‌تر به جستجوی الگوها و ارتباط‌های مهم در داده‌ها بپردازد.

Min-Max Scaler

این روش داده‌ها را به گونه‌ای نرمال می‌کند که مقادیر آنها بین بازه‌ای خاص (معمولاً ۰ تا ۱) قرار بگیرند. فرمول استفاده شده برای این روش به صورت زیر است:

$$z = \frac{x - \min(X)}{\max(X) - \min(X)}$$

که در آن:

- X مقدار اولیه داده

- $\min(X)$ حداقل مقدار در داده‌ها

- $\max(X)$ حداکثر مقدار در داده‌ها

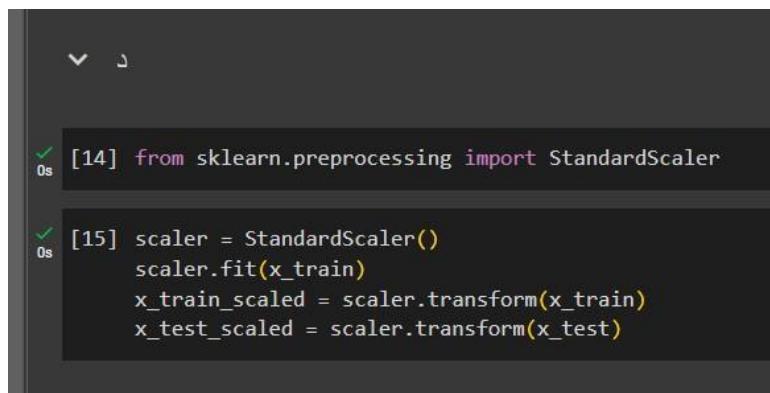
- Z مقدار نرمال شده

است.

این روش برای الگوریتم‌هایی مانند شبکه‌های عصبی که مقادیر ورودی باید در بازه مشخصی قرار داشته باشند، مناسب است.

در مورد استفاده از داده‌های `test` برای نرمال‌سازی، عموماً از آنها برای محاسبه مقادیر میانگین و واریانس استفاده نمی‌شود. دلیل این است که هدف از نرمال‌سازی داده‌ها ایجاد یک مقیاس یکسان برای آموزش مدل است، بنابراین اطلاعات `test` ممکن است نرمال شوند، اما مقادیر میانگین و واریانس بر اساس داده‌های آموزش محاسبه می‌شوند تا از انحراف مدل از داده‌های آموزش جلوگیری شود و عملکرد مدل روی داده‌های جدید بهبود یابد.

برای حل تمرین از روش `StandardScaler` و داده‌ی `train` استفاده شده است.



```
[14] from sklearn.preprocessing import StandardScaler  
[15] scaler = StandardScaler()  
      .fit(x_train)  
      .transform(x_train)  
      .transform(x_test)
```

2.3. بخش سوم

برای پیاده سازی یک مدل طبقه بند از طبقه بند `LogisticRegression` استفاده کردیم.

ابتدا یک کلاس به همین نام تعریف می‌کنیم. این کلاس شامل متدهای `__init__` (مقاداردهی اولیه)، `predict` (مقاداردهی اولیه ضرایب W یا همان وزن‌ها)، `update` (به روزرسانی وزن‌ها)، `_error` (محاسبه خطا)، (`piyeshbinii` خروجی بر حسب ورودی دلخواه) و `fit` (تعلیم ماشین) است.

از این کلاس برای طراحی طبقه بند دو طبقه دیتای خود استفاده کردیم. طبق نتست متوجه شدیم که به دلیل راحت بودن کار تفکیک، 10 تکرار کافی است.

بخش 3

```
✓ 0s  class LogisticRegression:

    def __init__(self, n_iter=50, learning_rate=0.005, random_state=None):
        self.n_iter = n_iter
        self.eta = learning_rate
        np.random.seed(random_state)
    def _weight_init(self, num_features):
        self.w = np.random.rand(num_features) * 0
        self.b = np.random.rand()

    def _update(self, x, E):
        dj = np.dot(x.T, E)
        self.w += self.eta * dj
        self.b += self.eta * np.sum(E)

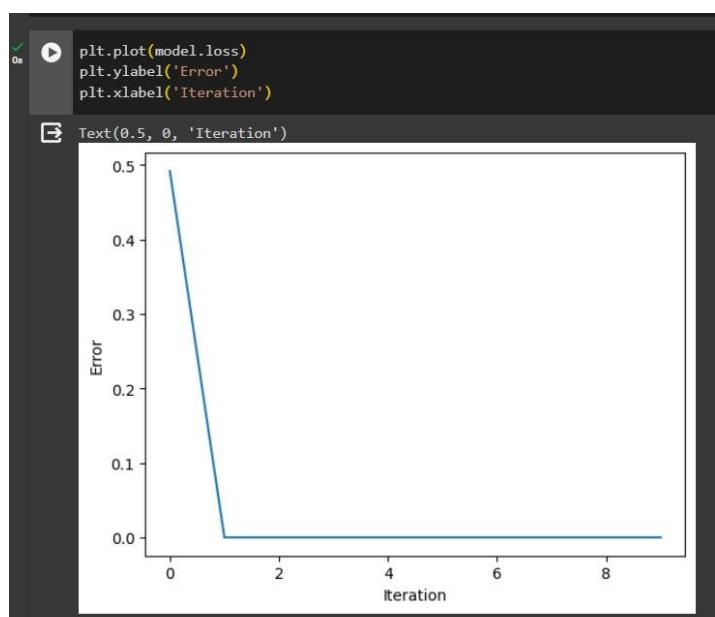
    def _error(self, predict, true):
        E = true - predict
        e = 1 / len(true) * np.dot(E, E)
        return E, e

    def predict(self, x):
        z = np.dot(x, self.w) + self.b
        a = 1 / (1 + np.e**(-z))
        y_hat = np.array([1 if hat > 0.5 else 0 for hat in a])
        return y_hat

    def fit(self, x, y):
        num_features = x.shape[1]
        self._weight_init(num_features)
        self.loss = []
        for iter in range(self.n_iter):
            y_hat = self.predict(x)
            E, e = self._error(y_hat, y)
            self.loss.append(e)
            self._update(x, E)

model = LogisticRegression(n_iter=10, learning_rate=0.005, random_state=RS)
model.fit(x_train_scaled, y_train)
```

نمودار خطای بر حسب تکرار



از دو معیار زیر برای ارزیابی عملکرد استفاده می‌کنیم. F1-score و Accuracy دو معیار ارزیابی متداول در مسائل دسته‌بندی ماشینی هستند که به ترتیب میزان دقیقی مدل و توازن بین دقت و بازخوانی را اندازه‌گیری می‌کنند.

:Accuracy .1

- دقیقی معیاری است که نشان می‌دهد چه تعداد از نمونه‌های دسته‌بندی شده به درستی تشخیص داده شده‌اند.
- به صورت ریاضی، دقیق برابر است با تعداد نمونه‌هایی که به درستی دسته‌بندی شده‌اند تقسیم بر کل تعداد نمونه‌ها.
- این معیار مفید است زمانی که کلاس‌ها در داده‌ها متوازن هستند، به این معنا که تعداد نمونه‌ها در هر کلاس به یکدیگر نزدیک است.

:F1-score .2

- F1-score یک معیار جامع است که هم دقیق (Precision) و هم بازخوانی (Recall) را در نظر می‌گیرد.
- دقیق نسبت تعداد نمونه‌هایی که به درستی دسته‌بندی شده‌اند به کل نمونه‌های دسته‌بندی شده است.
- بازخوانی نسبت تعداد نمونه‌هایی که به درستی دسته‌بندی شده‌اند به کل نمونه‌های واقعی یک کلاس است.
- F1-score از این دو معیار به صورت هندسی میانگین گرفته می‌شود و به عنوان معیاری برای ارزیابی توازن بین دقیق و بازخوانی مدل استفاده می‌شود.

- معمولاً زمانی که دقیق و بازخوانی هر دو مهم هستند و می‌خواهیم توازن مناسبی بین آنها داشته باشیم، از استفاده می‌کنیم، به خصوص زمانی که کلاس‌ها نامتوازن هستند و تعداد نمونه‌های هر کلاس متفاوت است.

در کد زیر این دو فاکتور را برای دیتای train و test محاسبه کرده ایم:

```
[21] y_hat = model.predict(x_train_scaled)
    TP = np.sum(y_hat[y_train==1])
    TN = np.sum(np.abs(y_hat[y_train==0]-1))
    FP = np.sum(y_hat[y_train==0])
    FN = np.sum(np.abs(y_hat[y_train==1]-1))
    accuracy = (TP+TN)/len(y_train)
    print(f'Accuracy of train data: {accuracy*100:.1f}%')
    precision = TP / (TP+FP)
    recall = TP / (TP+FN)
    f1_score = 2 * (precision*recall) / (precision+recall)
    print(f'F1-score of train data: {f1_score*100:.1f}%')

    y_hat = model.predict(x_test_scaled)
    TP = np.sum(y_hat[y_test==1])
    TN = np.sum(np.abs(y_hat[y_test==0]-1))
    FP = np.sum(y_hat[y_test==0])
    FN = np.sum(np.abs(y_hat[y_test==1]-1))
    accuracy = (TP+TN)/len(y_test)
    print(f'Accuracy of test data: {accuracy*100:.1f}%')
    precision = TP / (TP+FP)
    recall = TP / (TP+FN)
    f1_score = 2 * (precision*recall) / (precision+recall)
    print(f'F1-score of test data: {f1_score*100:.1f}%')

Accuracy of train data: 100.0%
F1-score of train data: 100.0%
Accuracy of test data: 100.0%
F1-score of test data: 100.0%
```

نمی توان از روی نمودار تابع اتلاف و قبل از مرحله ارزیابی با قطعیت در مورد عملکرد مدل نظر داد. زیرا این نمودار حین آموزش و از طریق دیتای `train` ساخته می شود. ممکن است ماشین ما با این دیتا `overtrain` شده باشد و صرفا برای آن دیتا خیلی خوب کار کند و برای دیتای جدید عملکرد خوبی نداشته باشد. (مثلًا نویز این دیتای خاص را هم یاد گرفته باشد و برای دیتای جدید با نویز متفاوت اشتباه عمل کند). برای همین قسمتی از دیتا را تحت عنوان دیتای تست از دیتای آموزش جدا می کنند و با آن دیتای متفاوت نیز عملکرد دستگاه را ارزیابی می کنند.

2.4. بخش چهارم

از روش `SGDClassifier` برای طبقه بندی استفاده شد.

پس از `import` کردن موارد لازم، جهت رسم تابع هزینه یک کلاس به نام `newSGD` تعریف می کنیم که از کلاس `SGDClassifier` ارث بری می کند. دو متدهای `__init__` (مقداردهی اولیه) و `partial_fit` (مانند همان `SGDClassifier` است با این تفاوت که میانگین مربعات خطأ را ذخیره می کند). را در این کلاس می نویسیم. حال به کمک کلاس تعریف شده و برای ۱ تکرار، ماشین را آموزش داده و نمودار تابع اتلاف را رسم می کنیم.

بخش 4

```
[22] from sklearn.linear_model import SGDClassifier

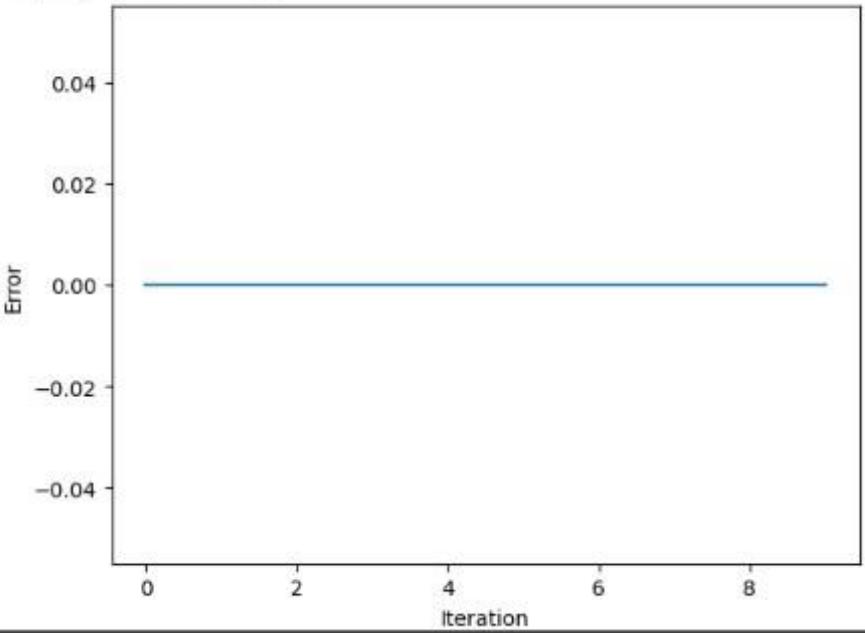
[23] class newSGD(SGDClassifier):
    def __init__(self):
        super().__init__()
        self.loss_history = []
    def partial_fit(self, X, y, classes=None, sample_weight=None):
        super().partial_fit(X, y, classes=classes, sample_weight=sample_weight)
        y_hat = self.predict(X)
        error = y_hat - y
        loss = 1/len(y) * np.dot(error, error)      #mean squared error (MSE)
        self.loss_history.append(loss)

model = newSGD()
N_iter = 10

for i in range(N_iter):
    model.partial_fit(x_train_scaled, y_train, classes=np.unique(y_train))

plt.plot(model.loss_history)
plt.ylabel('Error')
plt.xlabel('Iteration')

Text(0.5, 0, 'Iteration')
```



به دلیل سادگی و جدا بودن کلاس ها، این روش و روش قبلی هر دو عملکرد 100 درصدی و مناسبی دارند و نمی توان عملکرد این دو روش را با این دیتاست ساده، چندان مقایسه کرد.

2.5. بخش پنجم

Orange یک نرمافزار داده‌کاوی و تحلیل داده متن‌باز و قابل استفاده برای افراد بدون تخصص عمیق در علوم کامپیوتر و داده‌کاوی است. این نرمافزار با رابط کاربری گرافیکی (GUI) ساده و کاربرپسند امکانات متنوعی برای بارگیری، تحلیل، و بصری‌سازی داده‌ها ارائه می‌دهد. تعداد زیادی از الگوریتم‌های داده‌کاوی و یادگیری ماشین، ابزارهای پردازش داده، و ابزارهای بصری‌سازی اطلاعاتی در این نرمافزار گنجانده شده است.

بعضی از قابلیت‌های مهم Orange عبارتند از:

1. تحلیل داده‌های گوناگون: از جمله تحلیل داده‌های دسته‌ای، رگرسیون، خوشبندی، کاوش جامعه، و پیش‌بینی.
 2. تجزیه و تحلیل مرورگری: این ابزار به شما امکان می‌دهد تا با استفاده از محیط گرافیکی، اقدام به تجزیه و تحلیل داده‌ها کنید و نتایج خود را به راحتی بصری‌سازی کنید.
 3. پشتیبانی از الگوریتم‌های متنوع: الگوریتم‌های متنوعی از جمله درخت تصمیم، ماشین بردار پشتیبان، شبکه‌های عصبی، و الگوریتم‌های خوشبندی در Orange قابل دسترسی هستند.
 4. قابلیت ترکیب مأذول‌ها: کاربران می‌توانند مأذول‌های مختلف را با یکدیگر ترکیب کرده و فرایندهای تحلیلی پیچیده‌تر را ایجاد کنند.
 5. پشتیبانی از زبان برنامه‌نویسی Python: کاربران می‌توانند با استفاده از زبان برنامه‌نویسی Python، قابلیت‌های Orange را گسترش داده و سفارشی‌سازی‌های مورد نیاز خود را انجام دهند.
 6. پشتیبانی از ویژگی‌های بصری‌سازی: Orange امکانات گسترده‌ای برای بصری‌سازی داده‌ها از جمله نمودارهای مختلف، نمایش‌های چندبعدی، و نمایش‌های شبکه‌ای فراهم می‌کند.
- در کل، Orange یک ابزار قدرتمند و کارآمد برای انجام تحلیل داده‌ها به صورت گرافیکی و بدون نیاز به دانش عمیق در زمینه داده‌کاوی است.

3. سوال سوم

3.1 بخش اول

ابتدا موارد مورد نیاز را نصب می کنیم.

```
▼ 1 بخش 1

[26] import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt

[27] !pip install --upgrade --no-cache-dir gdown
      !gdown 1AbMidVqGDkuWLshJttuK0sN72tVYw26
```

سپس پس از نصب gdown، با استفاده از آن دیتاست CSV که دانلود کرده و در drive به صورت public قرار داده ایم را به Apparent Temperature، Temperature، humidity و google colab می آوریم و دیتای مربوط به Apparent Temperature را جدا می کنیم.

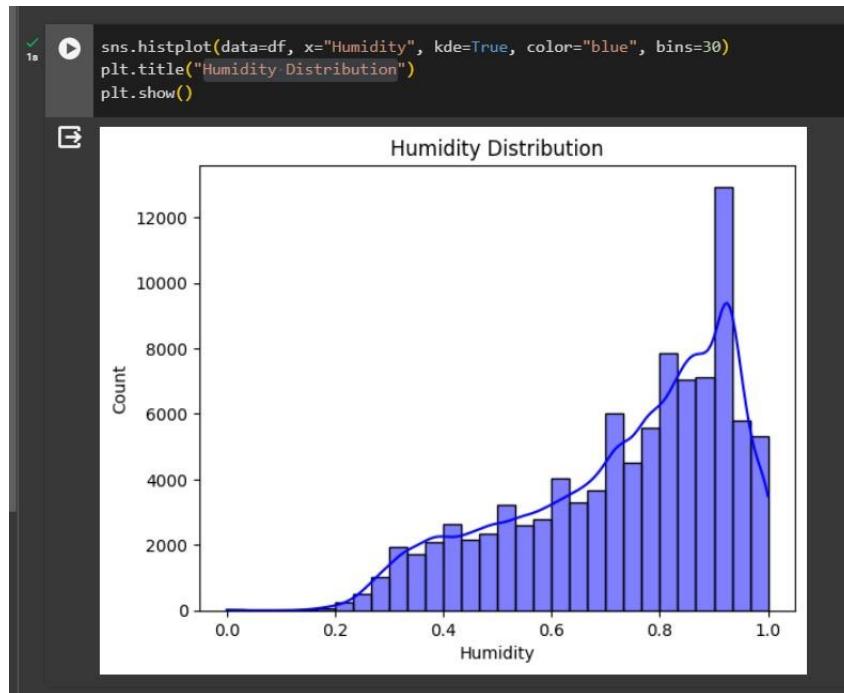
```
[28] csv_file_path = '/content/weatherHistory.csv'
      df = pd.read_csv(csv_file_path)
      humidity = df['Humidity'].values
      temp = df['Temperature (C)'].values
      app_temp = df['Apparent Temperature (C)'].values
```

هیئت مپ ماتریس همبستگی ویژگی ها

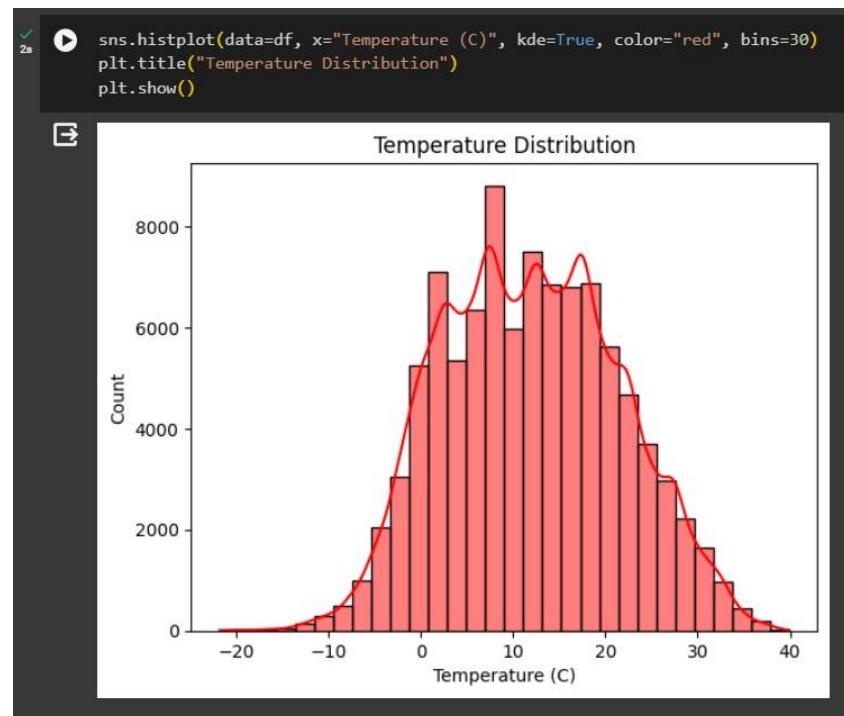


با توجه به این ماتریس مشخص می شود که کدام ویژگی ها بیشترین همبستگی را با یکدیگر دارند و می توانند برای پیش‌بینی مورد استفاده قرار گیرند. مثلا برای پیش‌بینی Apparent Temperature به ترتیب بهترین ویژگی، Visibility و Humidity، Temperature است.

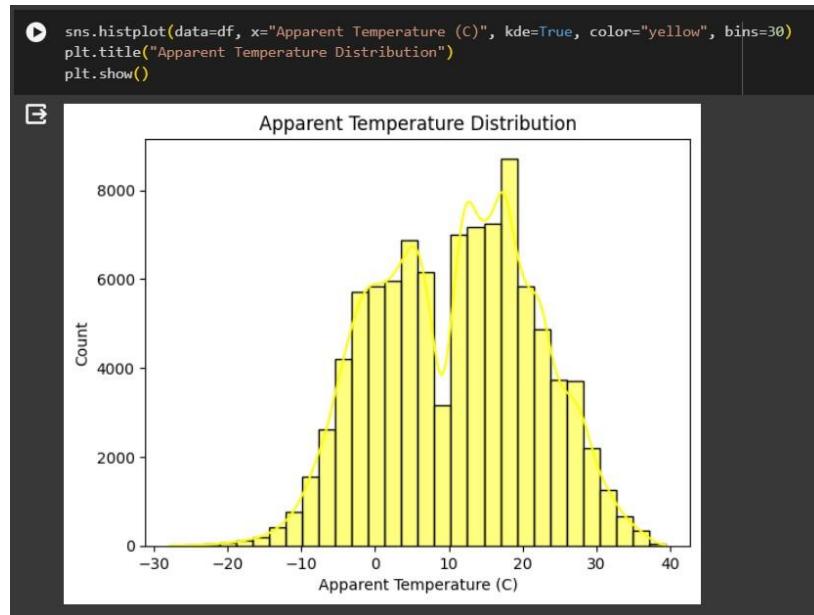
نمودار توزیع رطوبت



نمودار توزیع دما

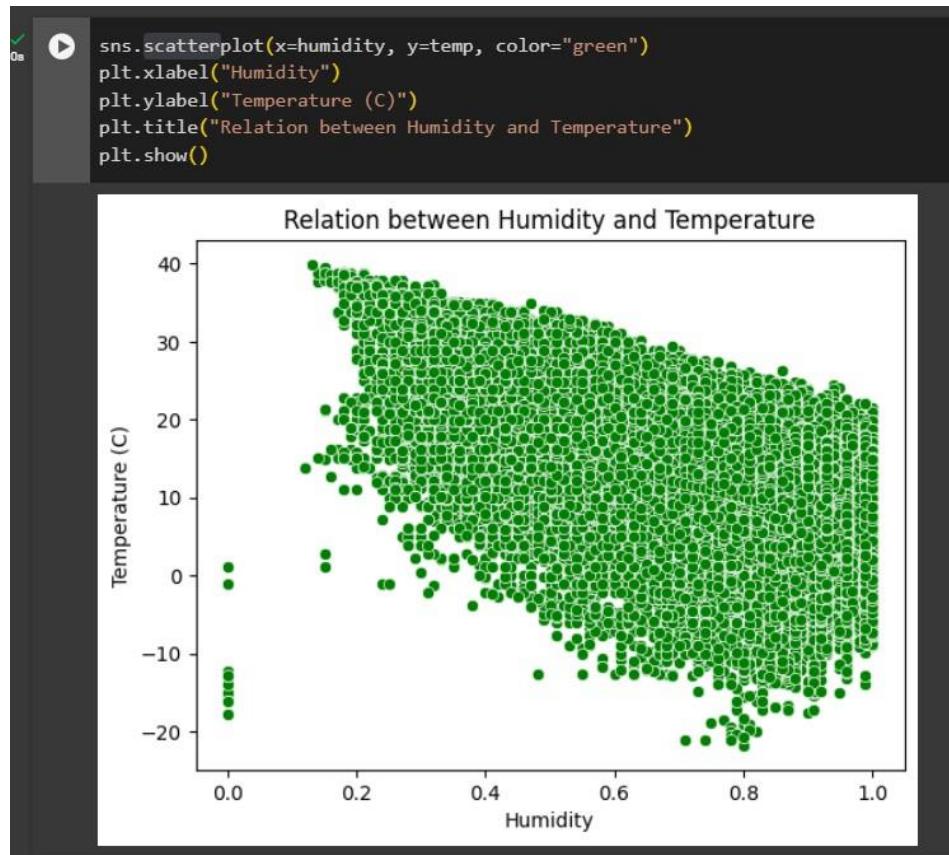


نمودار توزیع دمای ظاهری



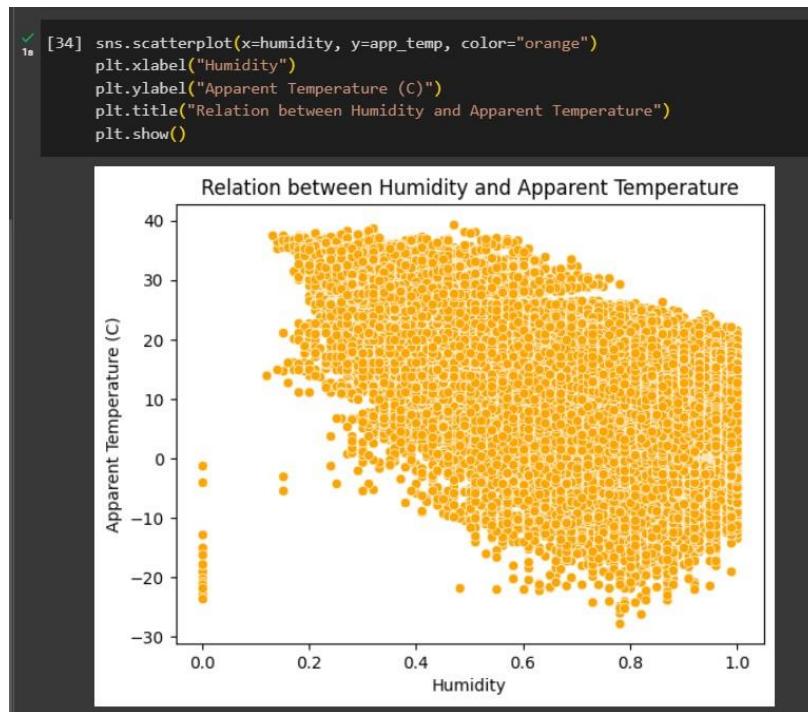
توزیع سه ویژگی در نمودارهای بالا نمایش داده شده است.

نمودار پراکندگی دما بر حسب رطوبت

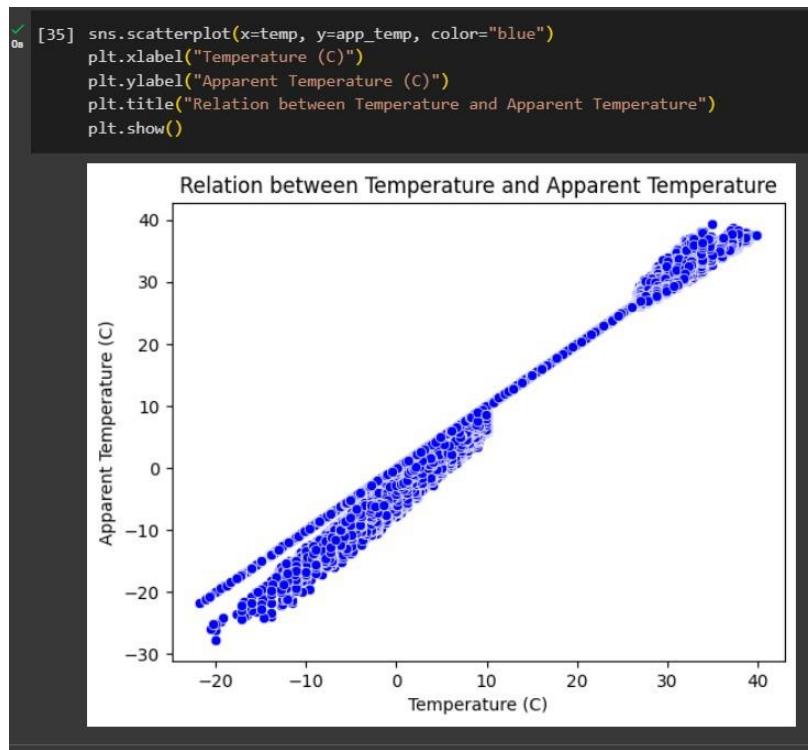


ارتباط خطی تنگاتنگی بین این دو ویژگی مشاهده نمی شود.

نمودار پراکندگی دمای ظاهری بر حسب رطوبت

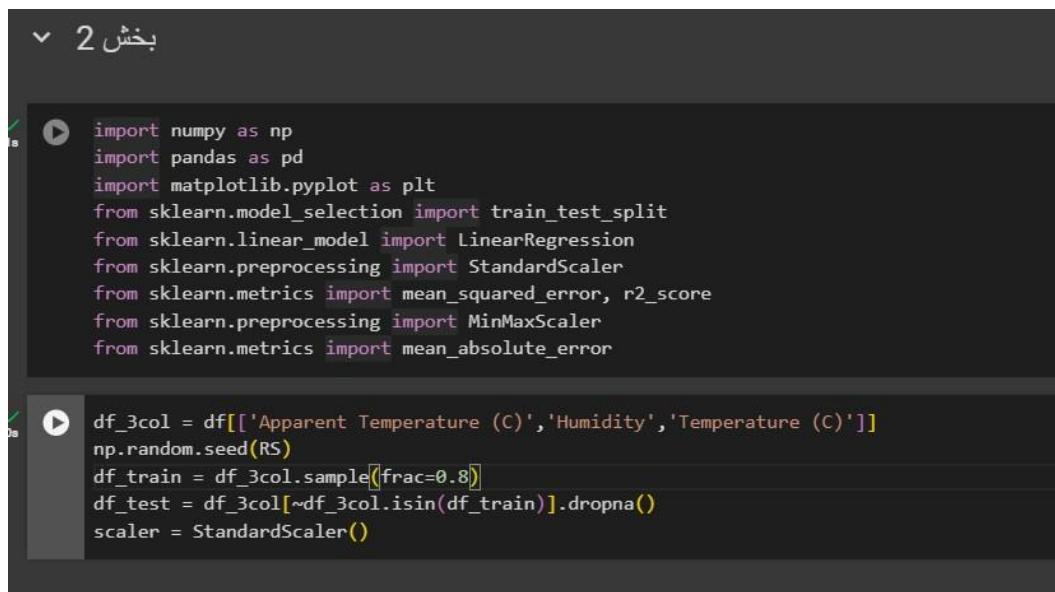


نمودار پراکندگی دمای ظاهری بر حسب دما



3.2. بخش دوم

ابتدا موارد مورد نیاز را `import` می کنیم. دیتای دمای ظاهری، دما و رطوبت را از بقیه جدا می کنیم. به مانند سوالات قبل، 20 درصد دیتا را برای تست و 80 درصد را برای آموزش جدا می کنیم.

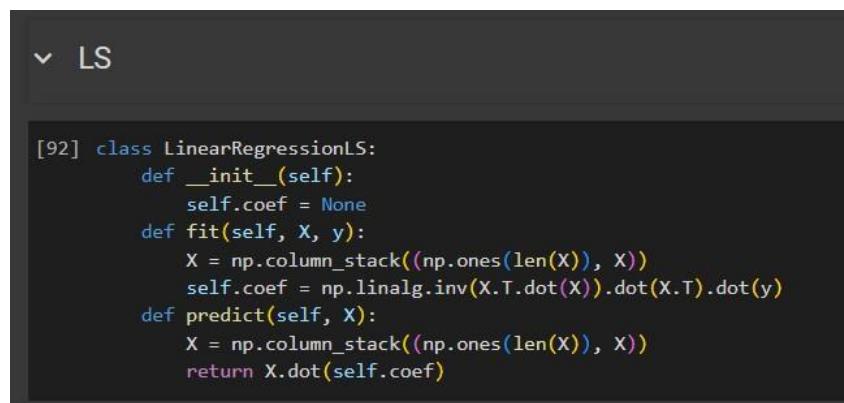


```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
```

```
df_3col = df[['Apparent Temperature (C)', 'Humidity', 'Temperature (C)']]
np.random.seed(42)
df_train = df_3col.sample(frac=0.8)
df_test = df_3col[~df_3col.isin(df_train)].dropna()
scaler = StandardScaler()
```

LS

حال کلاس `LinearRegressionLS` را برای تخمین `LS` می نویسیم. این کلاس حاوی سه متده است: `__init__` (مقداردهی اولیه)، `fit` (آموزش ماشین و آپدیت ضرایب) و `predict` (پیش‌بینی خروجی بر حسب ورودی وارد شده و ضرایب آموزش یافته) است.



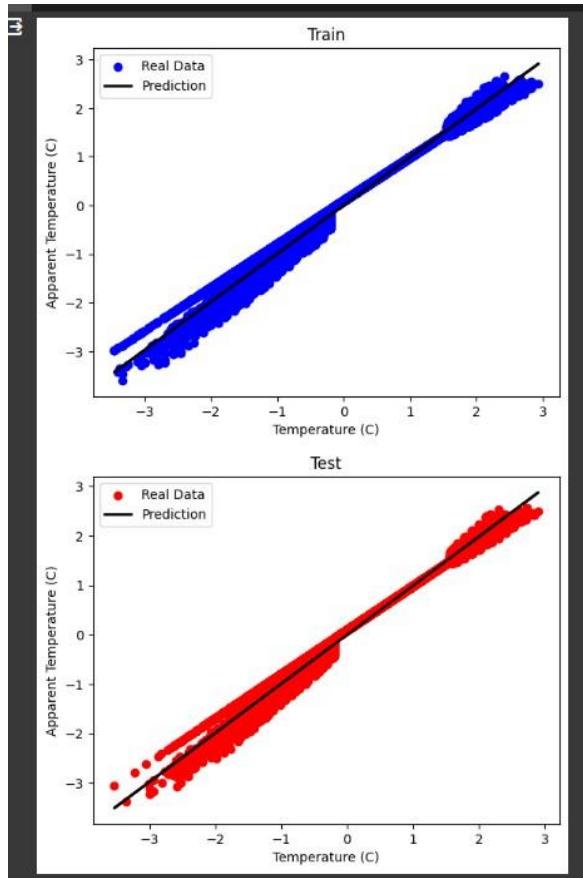
```
[92] class LinearRegressionLS:
    def __init__(self):
        self.coef = None
    def fit(self, X, y):
        X = np.column_stack((np.ones(len(X)), X))
        self.coef = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
    def predict(self, X):
        X = np.column_stack((np.ones(len(X)), X))
        return X.dot(self.coef)
```

در این قسمت دما را به عنوان ورودی و دمای ظاهری را به عنوان خروجی در نظر می گیریم. ابتدا دیتا را با `StandardScaler` می کنیم. سپس به کمک کلاسی که تعریف کردیم، یک مدل یادگیری ماشین آموزش می دهیم. سپس نمودار پراکندگی و خط پیش‌بینی را رسم می کنیم. این کار را برای دیتای آموزش و تست انجام می دهیم.

```

input_train = df_train['Temperature (C)'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Temperature (C)'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionLS()
ls.fit(input_train_scaled, output_train_scaled)
plt.figure()
plt.scatter(input_train_scaled, output_train_scaled, c='blue', label='Real Data')
hat_train = ls.predict(input_train_scaled)
plt.plot(input_train_scaled, hat_train, c='black', linewidth=2, label='Prediction')
plt.title('Train')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Temperature (C)')
plt.legend()
plt.figure()
plt.scatter(input_test_scaled, output_test_scaled, c='red', label='Real Data')
hat_test = ls.predict(input_test_scaled)
plt.plot(input_test_scaled, hat_test, c='black', linewidth=2, label='Prediction')
plt.title('Test')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Temperature (C)')
plt.legend()
plt.show()

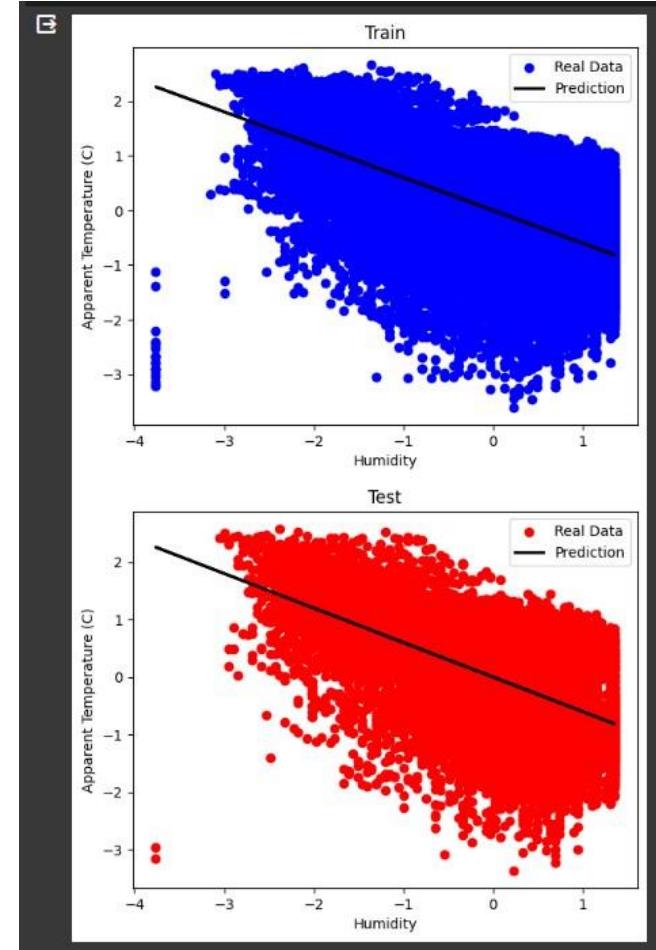
```



مطابق انتظار به علت وجود همبستگی بالا بین دو ویژگی، خط پیش‌بینی تطابق خوبی روی دیتای واقعی دارد.

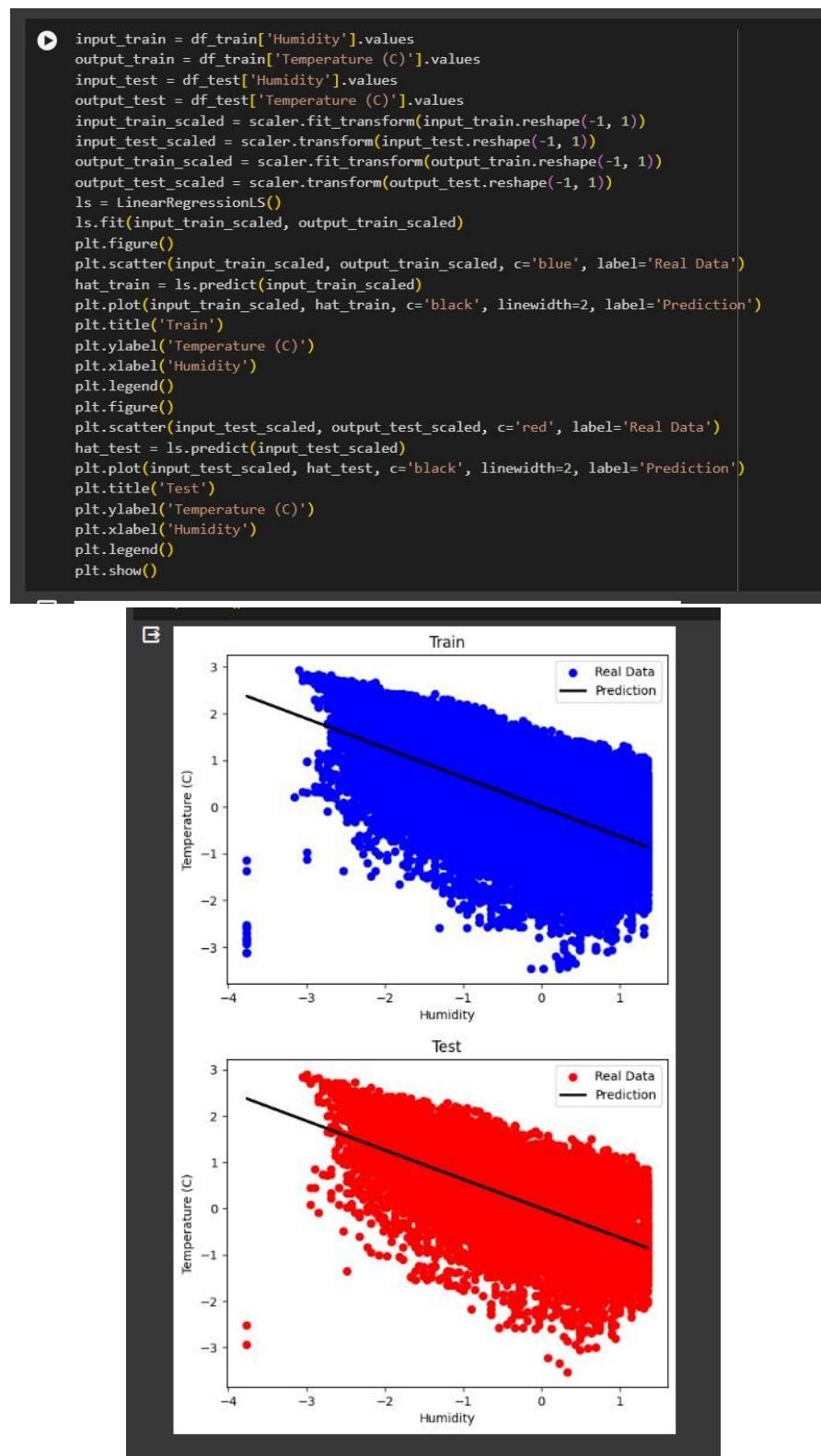
این قسمت شبیه حالت قبل است با این تفاوت که رطوبت را ورودی و دمای ظاهری را به عنوان خروجی درنظر می‌گیریم.

```
input_train = df_train['Humidity'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionLS()
ls.fit(input_train_scaled, output_train_scaled)
plt.figure()
plt.scatter(input_train_scaled, output_train_scaled, c='blue', label='Real Data')
hat_train = ls.predict(input_train_scaled)
plt.plot(input_train_scaled, hat_train, c='black', linewidth=2, label='Prediction')
plt.title('Train')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Humidity')
plt.legend()
plt.figure()
plt.scatter(input_test_scaled, output_test_scaled, c='red', label='Real Data')
hat_test = ls.predict(input_test_scaled)
plt.plot(input_test_scaled, hat_test, c='black', linewidth=2, label='Prediction')
plt.title('Test')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Humidity')
plt.legend()
plt.show()
```



مطابق انتظار به علت عدم وجود همبستگی بالا بین دو ویژگی، خط پیشینی تطابق خوبی روی دیتای واقعی ندارد.

این قسمت شبیه حالت قبل است با این تفاوت که رطوبت را ورودی و دما را به عنوان خروجی درنظر می‌گیریم.



مطابق انتظار به علت عدم وجود همبستگی بالا بین دو ویژگی، خط پیشینی تطابق خوبی روی دیتای واقعی ندارد.

برای هر سه حالت بالا، خطای mean absolute error و mean squared error را برای دیتای train و test محاسبه می کنیم. مطابق انتظار کمترین خطای برای حالت اول (پیش‌بینی دمای ظاهری به کمک دمای واقعی) است.

```

 input_train = df_train['Temperature (C)'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Temperature (C)'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionLS()
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Apparent Temperature from Temperature:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')

 Apparent Temperature from Temperature:
MSE Train: 0.01470743071208607, MSE Test: 0.014607000799199714, MAE Train: 0.09292441321964257, MAE Test: 0.0923393782683564

 input_train = df_train['Humidity'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionLS()
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Apparent Temperature from Humidity:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')

 Apparent Temperature from Humidity:
MSE Train: 0.6395939382037892, MSE Test: 0.6259186937602784, MAE Train: 0.6527070386075671, MAE Test: 0.6448773999248725

 [98] input_train = df_train['Humidity'].values
output_train = df_train['Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionLS()
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

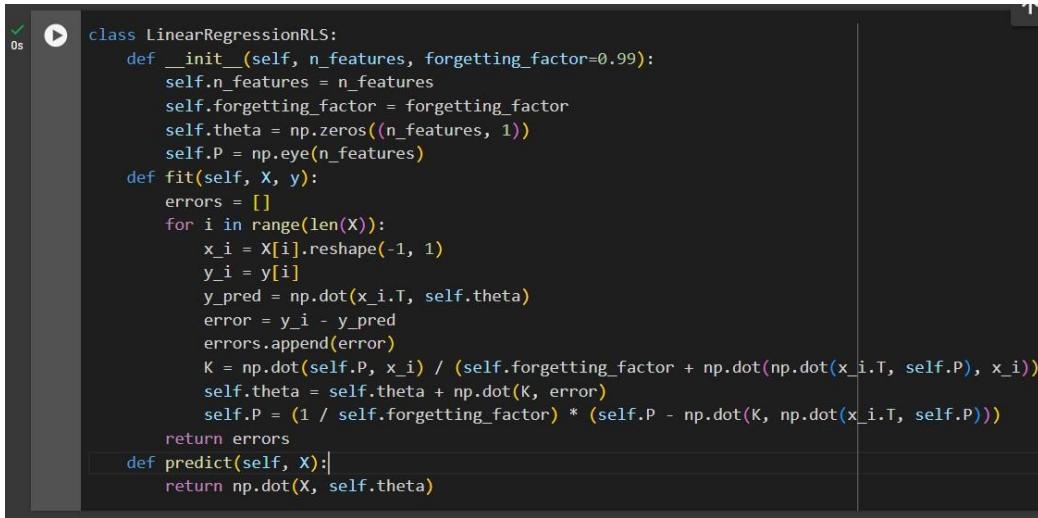
mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Temperature from Humidity:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')

 Temperature from Humidity:
MSE Train: 0.6028936006469036, MSE Test: 0.5912566869437793, MAE Train: 0.633513900402128, MAE Test: 0.6266027991686135

```

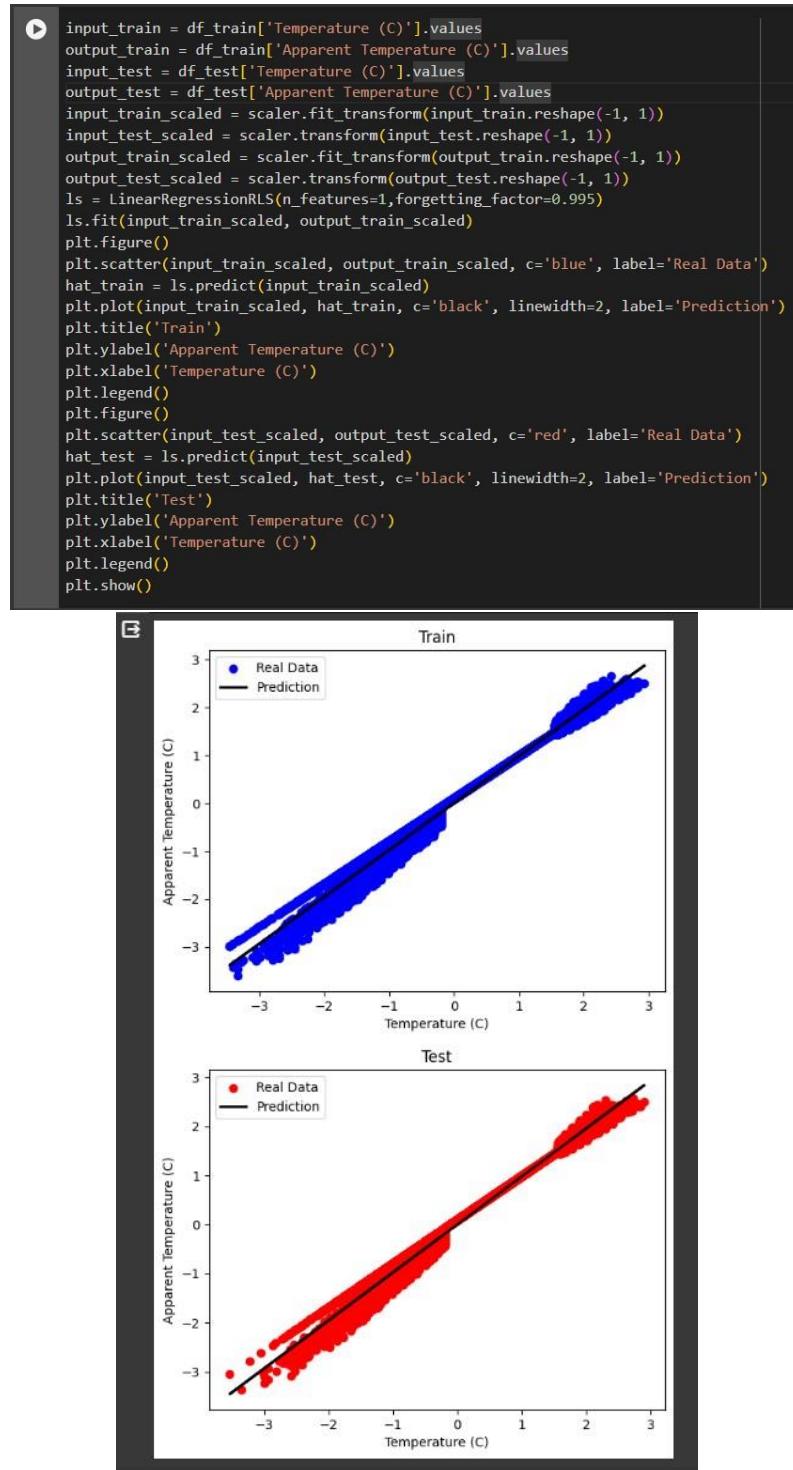
RLS

حال کلاس RLS را برای تخمین LinearRegression می نویسیم. این کلاس حاوی سه متد `__init__` (مقداردهی اولیه)، `fit` (آموزش ماشین و آپدیت ضرایب) و `predict` (پیش‌بینی خروجی بر حسب ورودی وارد شده و ضرایب آموزش یافته) است.



```
0s class LinearRegressionRLS:
    def __init__(self, n_features, forgetting_factor=0.99):
        self.n_features = n_features
        self.forgetting_factor = forgetting_factor
        self.theta = np.zeros((n_features, 1))
        self.P = np.eye(n_features)
    def fit(self, X, y):
        errors = []
        for i in range(len(X)):
            x_i = X[i].reshape(-1, 1)
            y_i = y[i]
            y_pred = np.dot(x_i.T, self.theta)
            error = y_i - y_pred
            errors.append(error)
            K = np.dot(self.P, x_i) / (self.forgetting_factor + np.dot(np.dot(x_i.T, self.P), x_i))
            self.theta = self.theta + np.dot(K, error)
            self.P = (1 / self.forgetting_factor) * (self.P - np.dot(K, np.dot(x_i.T, self.P)))
        return errors
    def predict(self, X):
        return np.dot(X, self.theta)
```

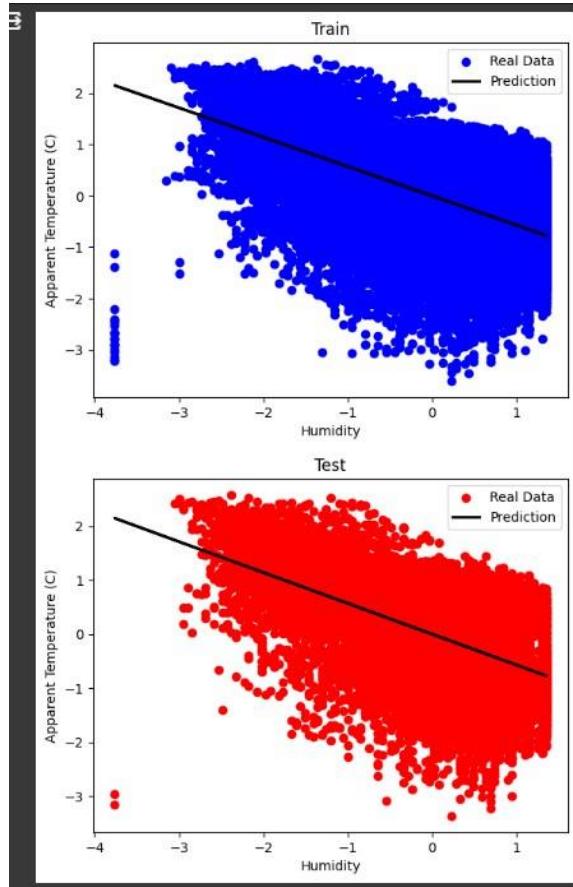
در این قسمت دما را به عنوان ورودی و دمای ظاهری را به عنوان خروجی در نظر می‌گیریم. ابتدا دیتا را با `StandardScaler` می‌کنیم. سپس به کمک کلاسی `LinearRegression` یک مدل یادگیری ماشین آموزش می‌دهیم. سپس نمودار پراکندگی و خط پیش‌بینی را رسم می‌کنیم. این کار را برای دیتای آموزش و تست انجام می‌دهیم.



مطابق انتظار به علت وجود همبستگی بالا بین دو ویژگی، خط پیش‌بینی تطابق خوبی روی دیتای واقعی دارد.

این قسمت شبیه حالت قبل است با این تفاوت که رطوبت را ورودی و دمای ظاهری را به عنوان خروجی درنظر می‌گیریم.

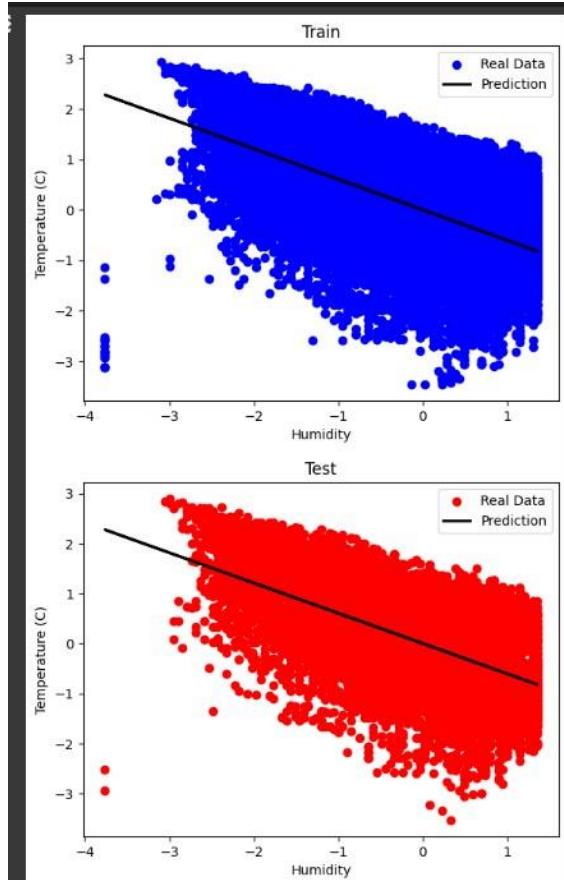
```
input_train = df_train['Humidity'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionRLS(n_features=1, forgetting_factor=0.995)
ls.fit(input_train_scaled, output_train_scaled)
plt.figure()
plt.scatter(input_train_scaled, output_train_scaled, c='blue', label='Real Data')
hat_train = ls.predict(input_train_scaled)
plt.plot(input_train_scaled, hat_train, c='black', linewidth=2, label='Prediction')
plt.title('Train')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Humidity')
plt.legend()
plt.figure()
plt.scatter(input_test_scaled, output_test_scaled, c='red', label='Real Data')
hat_test = ls.predict(input_test_scaled)
plt.plot(input_test_scaled, hat_test, c='black', linewidth=2, label='Prediction')
plt.title('Test')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Humidity')
plt.legend()
plt.show()
```



مطابق انتظار به علت عدم وجود همبستگی بالا بین دو ویژگی، خط پیش‌بینی تطابق خوبی روی دیتای واقعی ندارد.

این قسمت شبیه حالت قبل است با این تفاوت که رطوبت را ورودی و دما را به عنوان خروجی درنظر می‌گیریم.

```
▶ input_train = df_train['Humidity'].values
   output_train = df_train['Temperature (C)'].values
   input_test = df_test['Humidity'].values
   output_test = df_test['Temperature (C)'].values
   input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
   input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
   output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
   output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
   ls = LinearRegressionRLS(n_features=1, forgetting_factor=0.995)
   ls.fit(input_train_scaled, output_train_scaled)
   plt.figure()
   plt.scatter(input_train_scaled, output_train_scaled, c='blue', label='Real Data')
   hat_train = ls.predict(input_train_scaled)
   plt.plot(input_train_scaled, hat_train, c='black', linewidth=2, label='Prediction')
   plt.title('Train')
   plt.ylabel('Temperature (C)')
   plt.xlabel('Humidity')
   plt.legend()
   plt.figure()
   plt.scatter(input_test_scaled, output_test_scaled, c='red', label='Real Data')
   hat_test = ls.predict(input_test_scaled)
   plt.plot(input_test_scaled, hat_test, c='black', linewidth=2, label='Prediction')
   plt.title('Test')
   plt.ylabel('Temperature (C)')
   plt.xlabel('Humidity')
   plt.legend()
   plt.show()
```



مطابق انتظار به علت عدم وجود همبستگی بالا بین دو ویژگی، خط پیشینی تطابق خوبی روی دیتای واقعی ندارد.

برای هر سه حالت بالا، خطای mean absolute error و mean squared error را برای دیتای train و test محاسبه می کنیم. مطابق انتظار کمترین خطای برای حالت اول (پیش‌بینی دمای ظاهری به کمک دمای واقعی) است.

```

 input_train = df_train['Temperature (C)'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Temperature (C)'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionRLS(n_features=1, forgetting_factor=0.995)
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Apparent Temperature from Temperature:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')

 Apparent Temperature from Temperature:
MSE Train: 0.014895853281471944, MSE Test: 0.014756551328078377, MAE Train: 0.09421693127065481, MAE Test: 0.09359417344000796

```

```

 0s  input_train = df_train['Humidity'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
LinearRegressionRLS(n_features=1, forgetting_factor=0.995)
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Apparent Temperature from Humidity:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')

 Apparent Temperature from Humidity:
MSE Train: 0.6404728852594911, MSE Test: 0.6273736015248265, MAE Train: 0.6558645623681925, MAE Test: 0.6483250637155668

```

```

 0s  input_train = df_train['Humidity'].values
output_train = df_train['Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionRLS(n_features=1, forgetting_factor=0.995)
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Temperature from Humidity:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')

 Temperature from Humidity:
MSE Train: 0.603540145134373, MSE Test: 0.5924046927406619, MAE Train: 0.6355872153224793, MAE Test: 0.6288312150100959

```

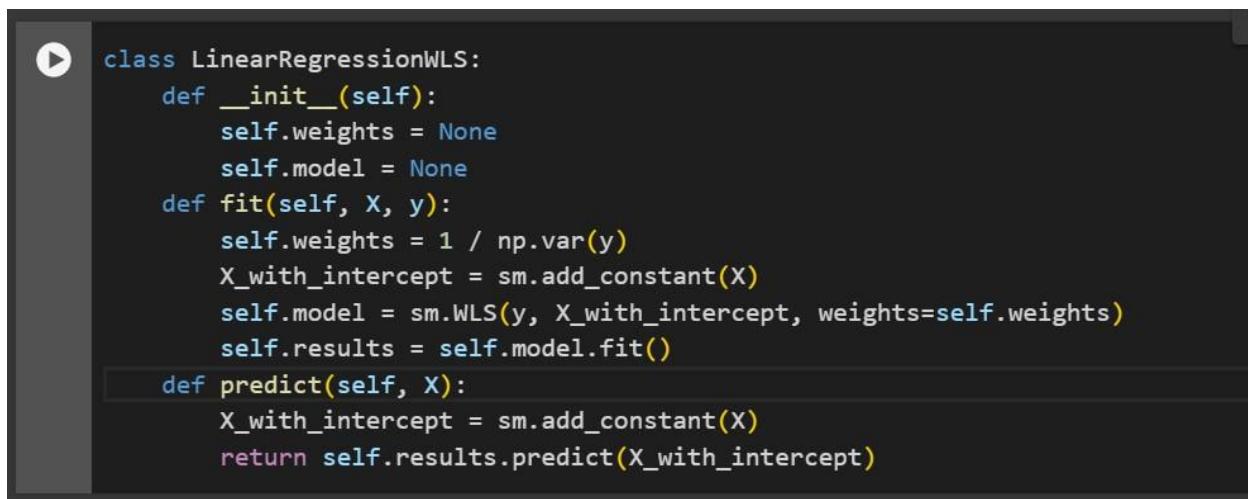
همانطور که مشاهده می شود، برای این دیتاست عملکرد LS و RLS تفاوت چندانی ندارند.

3.3. بخش سوم

روش رگرسیون خطی Weighted Least Squares (WLS) یک روش است که در آن برای تخمین پارامترهای مدل خطی، از یک ماتریس وزنی برای نمونه‌ها استفاده می‌شود. این ماتریس وزنی نشان دهنده اهمیت نمونه‌ها در مدل‌سازی است، به این معنی که نمونه‌های با وزن بیشتر در محاسبه تخمین‌ها واردتر هستند و وزن کمتری به نمونه‌های کم‌اهمیت تخصیص داده می‌شود.

بر خلاف روش عادی رگرسیون خطی (LS)، در WLS وزن مخصوص به هر نقطه در مدل‌سازی استفاده می‌شود. این وزن‌ها معمولاً بر اساس نوع مشکل و داده‌ها تعیین می‌شوند. به عنوان مثال، اگر نمونه‌ها با ویژگی‌های کیفیت بالاتر دارای وزن بیشتر باشند (به عنوان مثال، نمونه‌هایی که دقیق‌تر اندازه گرفته شده‌اند)، آن‌ها به طور معمول وزن بیشتری در مدل‌سازی خواهند داشت. فرق اصلی بین روش WLS و LS در استفاده از ماتریس وزنی است. در روش LS، همه نمونه‌ها به یکسان در مدل‌سازی استفاده می‌شوند، در حالی که در WLS، وزن‌ها برای هر نمونه ممکن است متفاوت باشند و به نمونه‌های با اهمیت بیشتر وزن بیشتری تخصیص داده شود. این امر به مدل کمک می‌کند تا بهتر به داده‌های با کیفیت بالا و حساس به نویز پاسخ دهد و از اغتشاشات ناشی از داده‌های با کیفیت پایین کاسته شود.

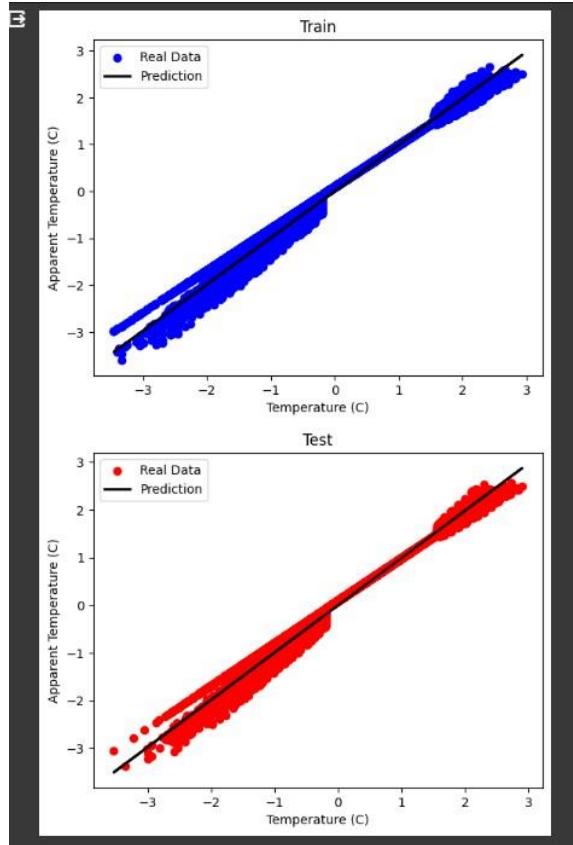
حال کلاس LinearRegressionWLS را برای تخمین WLS می‌نویسیم. این کلاس حاوی سه متدهای `__init__` (مقداردهی اولیه)، `fit` (آموزش ماشین و آپدیت ضرایب) و `predict` (پیش‌بینی خروجی بر حسب ورودی وارد شده و ضرایب آموزش یافته) است.



```
class LinearRegressionWLS:
    def __init__(self):
        self.weights = None
        self.model = None
    def fit(self, X, y):
        self.weights = 1 / np.var(y)
        X_with_intercept = sm.add_constant(X)
        self.model = sm.WLS(y, X_with_intercept, weights=self.weights)
        self.results = self.model.fit()
    def predict(self, X):
        X_with_intercept = sm.add_constant(X)
        return self.results.predict(X_with_intercept)
```

در این قسمت دما را به عنوان ورودی و دمای ظاهری را به عنوان خروجی در نظر می‌گیریم. ابتدا دیتا را با `StandardScaler` می‌کنیم. سپس به کمک کلاسی که تعریف کردیم، یک مدل یادگیری ماشین آموزش می‌دهیم. سپس نمودار پراکندگی و خط پیش‌بینی را رسم می‌کنیم. این کار را برای دیتای آموزش و تست انجام می‌دهیم.

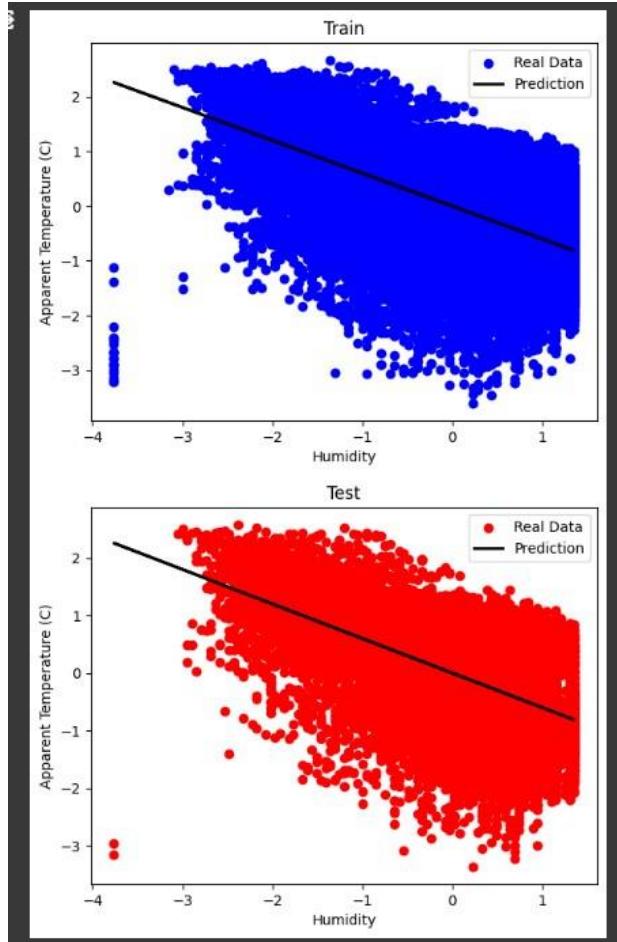
```
input_train = df_train['Temperature (C)'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Temperature (C)'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionWLS()
ls.fit(input_train_scaled, output_train_scaled)
plt.figure()
plt.scatter(input_train_scaled, output_train_scaled, c='blue', label='Real Data')
hat_train = ls.predict(input_train_scaled)
plt.plot(input_train_scaled, hat_train, c='black', linewidth=2, label='Prediction')
plt.title('Train')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Temperature (C)')
plt.legend()
plt.figure()
plt.scatter(input_test_scaled, output_test_scaled, c='red', label='Real Data')
hat_test = ls.predict(input_test_scaled)
plt.plot(input_test_scaled, hat_test, c='black', linewidth=2, label='Prediction')
plt.title('Test')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Temperature (C)')
plt.legend()
plt.show()
```



مطابق انتظار به علت وجود همبستگی بالا بین دو ویژگی، خط پیش‌بینی تطبیق خوبی روی دیتای واقعی دارد.

این قسمت شبیه حالت قبل است با این تفاوت که رطوبت را ورودی و دمای ظاهری را به عنوان خروجی درنظر می‌گیریم.

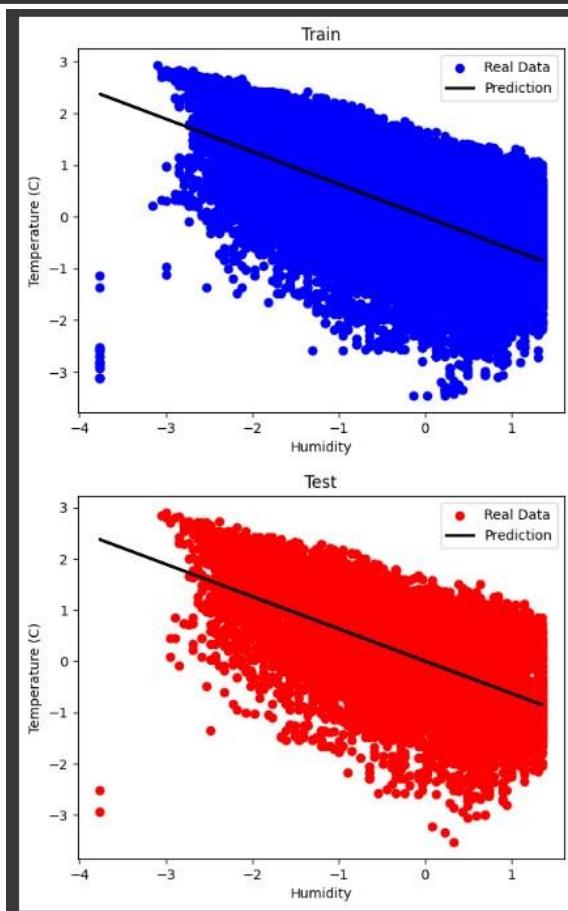
```
input_train = df_train['Humidity'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionNLS()
ls.fit(input_train_scaled, output_train_scaled)
plt.figure()
plt.scatter(input_train_scaled, output_train_scaled, c='blue', label='Real Data')
hat_train = ls.predict(input_train_scaled)
plt.plot(input_train_scaled, hat_train, c='black', linewidth=2, label='Prediction')
plt.title('Train')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Humidity')
plt.legend()
plt.figure()
plt.scatter(input_test_scaled, output_test_scaled, c='red', label='Real Data')
hat_test = ls.predict(input_test_scaled)
plt.plot(input_test_scaled, hat_test, c='black', linewidth=2, label='Prediction')
plt.title('Test')
plt.ylabel('Apparent Temperature (C)')
plt.xlabel('Humidity')
plt.legend()
plt.show()
```



مطابق انتظار به علت عدم وجود همبستگی بالا بین دو ویژگی، خط پیشینی تطابق خوبی روی دیتای واقعی ندارد.

این قسمت شبیه حالت قبل است با این تفاوت که رطوبت را ورودی و دما را به عنوان خروجی درنظر می‌گیریم.

```
▶ input_train = df_train['Humidity'].values
  output_train = df_train['Temperature (C)'].values
  input_test = df_test['Humidity'].values
  output_test = df_test['Temperature (C)'].values
  input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
  input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
  output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
  output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
  ls = LinearRegressionWLS()
  ls.fit(input_train_scaled, output_train_scaled)
  plt.figure()
  plt.scatter(input_train_scaled, output_train_scaled, c='blue', label='Real Data')
  hat_train = ls.predict(input_train_scaled)
  plt.plot(input_train_scaled, hat_train, c='black', linewidth=2, label='Prediction')
  plt.title('Train')
  plt.ylabel('Temperature (C)')
  plt.xlabel('Humidity')
  plt.legend()
  plt.figure()
  plt.scatter(input_test_scaled, output_test_scaled, c='red', label='Real Data')
  hat_test = ls.predict(input_test_scaled)
  plt.plot(input_test_scaled, hat_test, c='black', linewidth=2, label='Prediction')
  plt.title('Test')
  plt.ylabel('Temperature (C)')
  plt.xlabel('Humidity')
  plt.legend()
  plt.show()
```



مطابق انتظار به علت عدم وجود همبستگی بالا بین دو ویژگی، خط پیشینی تطابق خوبی روی دیتای واقعی ندارد.

برای هر سه حالت بالا، خطای mean absolute error و mean squared error را برای دیتای train و test محاسبه می کنیم. مطابق انتظار کمترین خطای برای حالت اول (پیش‌بینی دمای ظاهری به کمک دمای واقعی) است.

```
▶ input_train = df_train['Temperature (C)'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Temperature (C)'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionWLS()
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Apparent Temperature from Temperature:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')
```

✉ Apparent Temperature from Temperature:
MSE Train: 0.01470743071208607, MSE Test: 0.014607000799199714, MAE Train: 0.09292441321964251, MAE Test: 0.09233937826835636

```
▶ input_train = df_train['Humidity'].values
output_train = df_train['Apparent Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Apparent Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionWLS()
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Apparent Temperature from Humidity:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')
```

✉ Apparent Temperature from Humidity:
MSE Train: 0.6395939382037891, MSE Test: 0.6259186937602784, MAE Train: 0.6527070386075671, MAE Test: 0.6448773999248725

```
▶ input_train = df_train['Humidity'].values
output_train = df_train['Temperature (C)'].values
input_test = df_test['Humidity'].values
output_test = df_test['Temperature (C)'].values
input_train_scaled = scaler.fit_transform(input_train.reshape(-1, 1))
input_test_scaled = scaler.transform(input_test.reshape(-1, 1))
output_train_scaled = scaler.fit_transform(output_train.reshape(-1, 1))
output_test_scaled = scaler.transform(output_test.reshape(-1, 1))
ls = LinearRegressionWLS()
ls.fit(input_train_scaled, output_train_scaled)

hat_train = ls.predict(input_train_scaled)
hat_test = ls.predict(input_test_scaled)

mse_train = mean_squared_error(output_train_scaled, hat_train)
mse_test = mean_squared_error(output_test_scaled, hat_test)
mae_train = mean_absolute_error(output_train_scaled, hat_train)
mae_test = mean_absolute_error(output_test_scaled, hat_test)
print("Temperature from Humidity:")
print(f'MSE Train: {mse_train}, MSE Test: {mse_test}, MAE Train: {mae_train}, MAE Test: {mae_test}')
```

✉ Temperature from Humidity:
MSE Train: 0.6028936006469037, MSE Test: 0.5912566869437793, MAE Train: 0.633513900402128, MAE Test: 0.6266027991686135

3.4. بخش چهارم

الگوریتم (RLS) QR-Decomposition-Based Recursive Least Squares یک روش برای حل مسائل رگرسیون خطی است که بر پایه تجزیه QR ماتریس طراحی شده است. این الگوریتم مختص به مسائلی است که تعداد متغیرهای مستقل بسیار زیاد است و انتظار می‌رود که مدل به طور پویا با تغییرات در داده‌ها بروزرسانی شود.

در الگوریتم QR-Decomposition-Based RLS، ماتریس مربعی X از ویژگی‌ها را به دو ماتریس Q و R تجزیه می‌کنیم، به طوری که $X = QR$ باشد. سپس از این تجزیه برای به روزرسانی پارامترهای مدل استفاده می‌شود.

با استفاده از این تجزیه، مسئله رگرسیون خطی به دو مرحله تقسیم می‌شود:

۱. مرحله قدرت: ماتریس R به دست آمده از تجزیه QR می‌تواند به طور مستقیم با استفاده از روش‌های متعارف مسائل رگرسیون خطی حل شود، بدون نیاز به بازگشت به عقب.

۲. مرحله بازگشت به عقب: پس از به روزرسانی پارامترهای مدل با استفاده از ماتریس R ، ماتریس Q به روزرسانی می‌شود تا اطلاعات جدید وارد به مدل شود.

تفاوت اصلی بین الگوریتم RLS و QR-Decomposition-Based RLS ساده در استفاده از تجزیه QR ماتریس است. در الگوریتم ساده RLS، محاسبات به روزرسانی پارامترهای مدل براساس ماتریس غیرقابل تجزیه $X^T X$ انجام می‌شود. این ماتریس می‌تواند به دلیل بزرگ بودن تعداد ویژگی‌ها و نمونه‌ها باعث محاسبات سنگین شود. اما با استفاده از تجزیه QR، این مشکل کاهش می‌یابد زیرا ماتریس R دارای ابعاد کوچکتری است که محاسبات را سریع‌تر واقعی می‌کند و همچنین باعث از بین بردن محاسبات غیرضروری می‌شود.