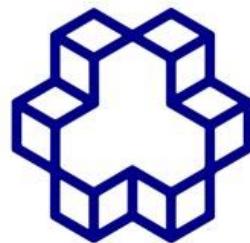


به نام خالق هستی بخش



دانشگاه صنعتی خواجه نصیرالدین طوسی

گزارش کار مینی پروژه ۱

درس: یادگیری ماشین

استاد درس : دکتر علیاری

نام و نام خانوادگی دانشجو:

علی مهرابی

شماره دانشجویی: ۴۰۲۲۳۸۵۴

۱۴۰۳ بهار

فهرست محتوا

شماره صفحه

عنوان

۱	لینک‌های مربوط به این گزارش
۲	سوال (۱)
۲	بخش (۱)
۳	بخش (۲)
۵	بخش (۳)
۸	بخش (۴)
۱۱	بخش (۵)
۱۶	سوال (۲)
۱۶	بخش (۱)
۱۹	بخش (۲)
۲۵	بخش (۳)
۲۸	بخش (۴)
۳۰	سوال (۳)
۳۰	بخش (۱)
۴۰	بخش (۲)
۴۵	بخش (۳)
۴۶	بخش (۴)
۴۷	مراجع

لينک‌های مربوط به اين گزارش

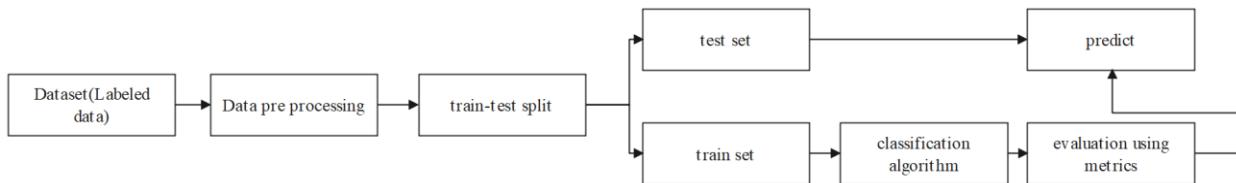
• لينک مربوط به گيت‌ها:

- https://github.com/dednomerios/ML_2024_Aliyari/tree/main/Mini%20Projects/Mini%20Project%201
 - لينک مربوط به گوگل كولب:
- <https://colab.research.google.com/drive/1s1gzqh1ekSHEWubZ6eHPxa8hsXtx4zLJ?usp=sharing>

سوال ۱)

بخش ۱)

روش‌های مختلفی برای بررسی الگوریتم‌های طبقه‌بندی وجود دارد. در این بخش به صورت زیر این الگوریتم نشان داده شده و در ادامه توضیحاتی بیان می‌شود.



در ابتدا دقت شود که ما کار Supervised learning انجام می‌دهیم (ورودی‌ها همگی برچسب دارند) و ما طبقه‌بندی خطی انجام می‌دهیم یعنی یک خط را بین دو یا چند خط را بین چند طبقه قرار می‌دهیم.

- در ابتدا یک دیتاست گرفته می‌شود که معمولاً دارای برچسبی است که target محسوب می‌شود. مثلاً این نمونه متعلق به گروه ۱ است و دیگری متعلق به گروه ۲ و این target feature گروه همان است.

- در مرحله بعد نیاز به انجام پیش پردازش دیتا داریم که شامل scaling و یا کار با missing value‌ها در داده است. در واقع هرچه داده تمیزتر و با کیفیت تری را وارد مدل کنیم، نتیجه بهتری خواهیم گرفت.
- مرحله بعدی باید کل داده‌ها را به دو بخش تقسیم کنیم. یک بخش test که از حالا تا انتهای الگوریتم کاری با آن نداریم و یک دسته train که تمام آموخته الگوریتم روی آن انجام خواهد شد. معمولاً این دسته از داده‌ها خود به یک دسته دیگر تقسیم می‌شود که شامل validation set می‌شود. معمولاً این بخش (train-test split) با نسبت ۸۰ به ۲۰ انجام می‌شود.

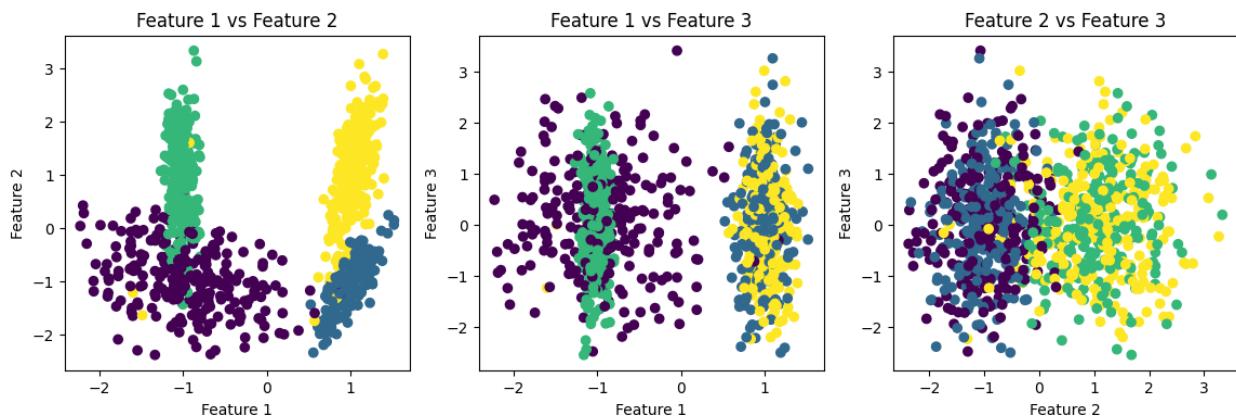
- حالا باید الگوریتم مورد نظر را از بین الگوریتم‌های موجود (LR, SVM, Perceptron...) انتخاب کنیم.

- سپس با fit کردن مدل روی این الگوریتم‌ها، می‌توانیم متريک‌های متدال مانند recall, accuracy ... را روی داده‌های train و test پیاده کنیم و میزان overfit یا underfit بودن مدل را بسنجیم.
- در نهایت کار predict را با دادن داده‌های test به مدلی که ساختیم انجام می‌دهیم و خطای، همان اختلاف بین مقدار پیش‌بینی شده و y_test است که جوابش را می‌دانستیم. در این مرحله می‌توانیم با کمک متريک‌هایی مانند confusion matrix میزان کارایی مدل را بهتر بسنجیم.

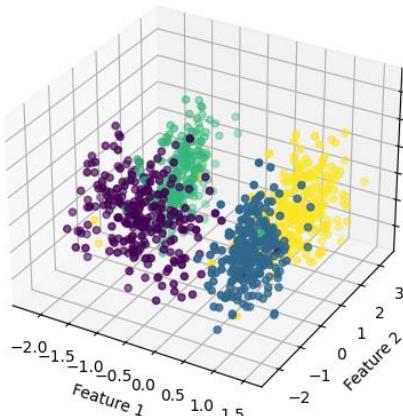
حالا فرض شود که داریم یک طبقه بندی چند کلاسه انجام می‌دهیم. در این حالت نیازمند عوض کردن چند مورد هستیم. یکی اینکه باید loss function را تغییر دهیم به طوری که به همه طبقه‌ها خدمت کند. مثلاً به جای binary - cross entropy در حالت دو کلاسه به soft max در چند کلاسه تغییر دهیم که احتمال تعلق هر instance را به هر کلاس به ما می‌دهد. پس در بخش الگوریتم و مشخصات مربو به آن باید تغییر ایجاد کنیم.

بخش ۲

به کمک کد نوشته شده و استفاده از `make_classification`، داده زیر را تولید می‌کنیم:



3D Scatter Plot of Data



به دو صورت ۲ بعدی (دو به دو) و ۳ بعدی (تمامی فیچرها در یک نمودار) داده‌ها و کلاس‌هارا رسم می‌کنیم.

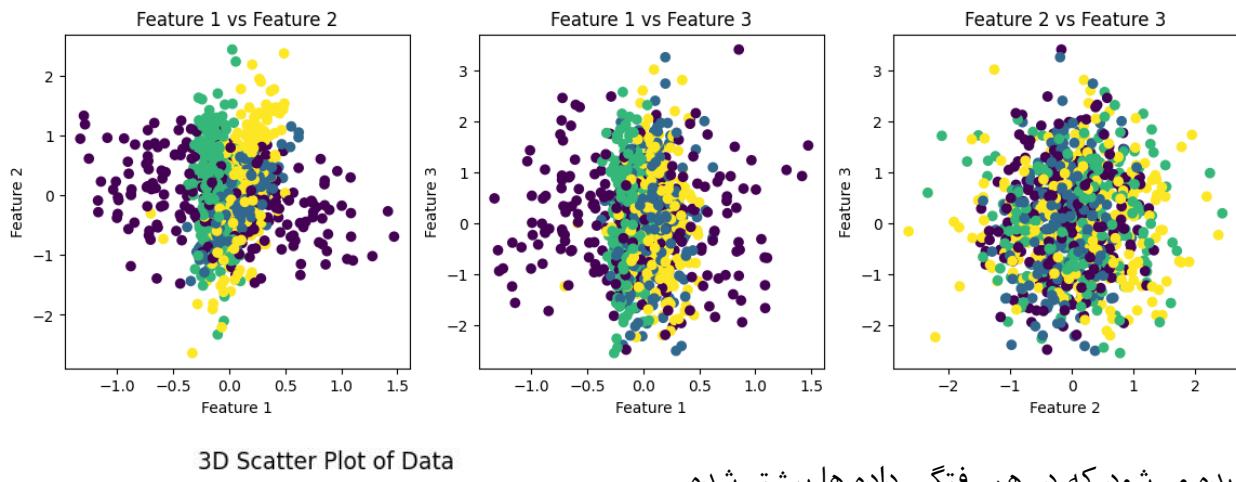
دیده می‌شود که به طور کلی ۱۰۰۰ سمپل یا نمونه داریم که در ۴ طبقه یا class قرار می‌گیرند. y در خروجی بیانگر این کلاس برای هر سمپل است. همینطور ۳ فیچر داریم که در نمودار ۳ بعدی روی هر محور قرار گرفته‌اند. فیچرها در X وجود دارند. در نهایت نمودار پراکندگی آن‌ها رسم شده و دیده می‌شود که داده‌ها پراکندگی قابل قبولی دارند. یعنی کار ما با این داده‌ها

برای طبقه بندی زیاد سخت نخواهد بود. البته فیچرهای ۲ و ۳ نزدیک به هم تغییر می‌کنند و ممکن است یکی از آن‌ها قابل حذف کردن باشد.

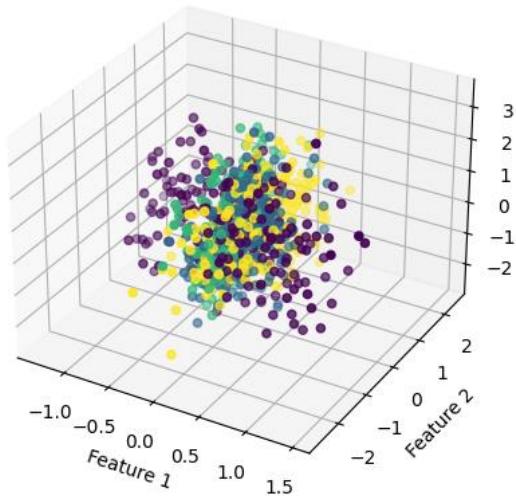
اما برای چالش برانگیزتر کردن عمل classification یا طبقه بنده می‌توان با ویژگی `class_sep` در ورودی تغییراتی را در دیتاست اعمال کرد به صورتی که جدا کردن آن‌ها سخت‌تر شود. با کم کردن این مقدار، داده‌ها داخل هم می‌روند و جدا کردن آن‌ها سخت‌تر خواهد شد. علاوه بر این $n_{clusters_per_class} = 1$ تنظیم شده تا توزیع داده‌ها در هر کلاس هم به یک شکل باشد. در صورتی که نوع بیشتری توزیع در هر کلاس داشته

باشیم می تواند کار را پیچیده تر هم بکند. البته باید درنظر داشت که شروط مربوط به خود متدهم رعایت شود و تمامی پارامتر هارا نمی توان به دلخواه هر عددی قرار داد.

برای مثال با قرار دادن $\text{class_sep} = 0.1$



دیده می شود که در هم رفتگی داده ها بیشتر شده.



بخش ۳

از دو طبقه بند خطی SGDClassifier و logistic regression استفاده می‌کنیم. ابتدا از train, test split استفاده می‌کنیم و دیتا را به صورت ۲۰-۸۰ جدا می‌کنیم. از رندوم استیت هم استفاده می‌کنیم تا داده‌ها قابل تکرار کردن باشند. توجه شود برای این قسمت class_sep = 0.5 تنظیم شده است.

سپس برای بهبود خروجی از standard scaler استفاده می‌کنیم تا داده‌ها scale شوند.

به صورت زیر مدل اول را می‌سازیم:

```
LogReg_model = LogisticRegression(solver = 'sag',max_iter = 200, random_state = 54)
LogReg_model.fit(x_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=200, random_state=54, solver='sag')
```

تعداد ایتریشن را روی ۲۰۰ تنظیم می‌کنیم. این روش بهینه سازی در واقع نوعی از Stochastic gradient descent است ولی از میانگین گرادیان‌های گام‌های گذشته برای گام‌های بعدی استفاده می‌کند و می‌تواند پایدار‌تر از روش‌های SGD باشد.

متريک‌های اين روش به شكل زير است:

```
[32] accuracy_logreg = accuracy_score(y_test,y_pred_logreg)
    print(f"for logistic regression the accuracy is {accuracy_logreg:.2f}%")
```

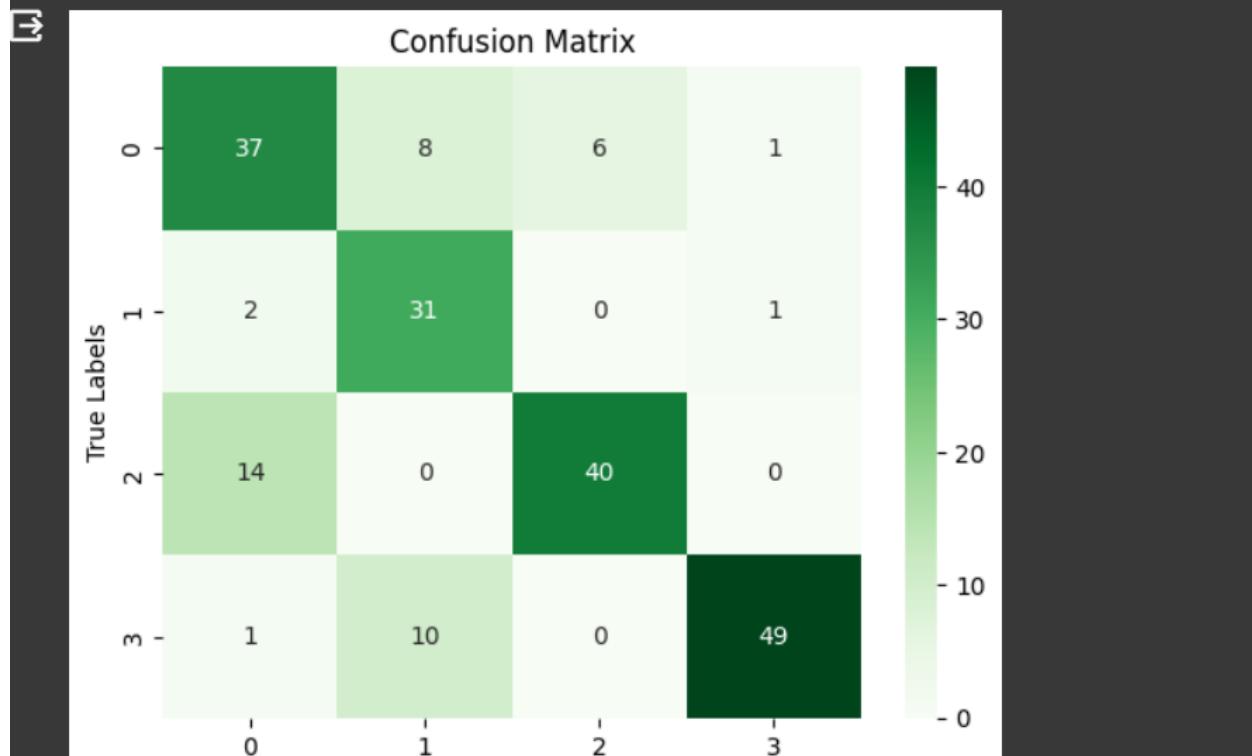
```
for logistic regression the accuracy is 0.79%
```

```
[33] classification_rep_logreg = classification_report(y_test,y_pred_logreg)
    print(classification_rep_logreg)
```

	precision	recall	f1-score	support
0	0.69	0.71	0.70	52
1	0.63	0.91	0.75	34
2	0.87	0.74	0.80	54
3	0.96	0.82	0.88	60
accuracy			0.79	200
macro avg	0.79	0.80	0.78	200
weighted avg	0.81	0.79	0.79	200

همانطور که دیده می‌شود، accuracy = 79%

```
▶ cm_logreg = confusion_matrix(y_test,y_pred_logreg)
sns.heatmap(cm_logreg, annot = True, fmt = 'd', cmap = 'Greens')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



```
[39] LogReg_model.score(x_train_scaled, y_train)
```

```
0.7725
```

```
▶ LogReg_model.score(x_test_scaled, y_test)
```

```
0.785
```

دیده می شود که overfit مشهود نیست.

برای روش بعدی از SGDClassifier استفاده می کنیم و از loss = 'log' استفاده میکنیم که برای multiclass است:

```
▶ SGD_model = SGDClassifier(loss = 'log', random_state = 54)
SGD_model.fit(x_train_scaled,y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:163: FutureWarning:
  SGDClassifier
```

و نتایج به صورت زیر است:

```

▶ accuracy_SGD = accuracy_score(y_test,y_pred_SGD)
print(f"for stochastic gradient decent classifier the accuracy is {accuracy_SGD:.2f}%")
for stochastic gradient decent classifier the accuracy is 0.73%
```



```

▶ classification_rep_SGD = classification_report(y_test,y_pred_SGD)
print(classification_rep_SGD)

      precision    recall   f1-score   support

          0       0.65     0.75     0.70      52
          1       0.66     0.56     0.60      34
          2       0.90     0.69     0.78      54
          3       0.74     0.87     0.80      60

    accuracy                           0.73      200
   macro avg       0.74     0.72     0.72      200
weighted avg       0.75     0.73     0.73      200

```



دیده می شود که نتایج logistic regression بهتری را به ما ارائه می دهد و برای بهینه سازی هم از الگوریتم SAG استفاده شد که توضیحاتی درباره آن داده شد. استفاده از تکنیک scaling به ما دقت قابل توجهی اضافه نکرد چون با گرفتن خروجی بدون این مرحله، برای مثال دقت روش اول به جای 78%، 79% بود.

حال اگر در داده های اولیه $\text{class_sep} = 1$ انتخاب شود، دقت روش ها به 94% افزایش خواهد یافت چون قابلیت جداسازی طبقه ها بهتر و ساده تر خواهد بود.

بخش ۴)

در این بخش ابتدا به سراغ استفاده از mlxtend رفتیم ولی چون تعداد ویژگی ها بیشتر از ۲ بود نتوانستیم از آن بهره ای بگیریم. پس به سراغ یک روش دیگر رفتیم.

در این روش از یک الگوریتم کاهش بعد استفاده می کنیم. در واقع به کمک PCA با استفاده از ۳ ویژگی، ۲ جز یا component تشکیل می شود که هر دو جز ترکیب خطی از ویژگی ها هستند. درواقع این یک تکنیک کاهش بعد است که از توضیحات بیشتر درباره آن در این قسمت خودداری می کنیم.

بعد از کاهش بعد مدل را فیت می کنیم و نواحی مرز را می کشیم:

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Assuming LogReg_model is your trained Logistic Regression model
# and x_test_scaled is your scaled test dataset

# Reduce dimensions with PCA
pca = PCA(n_components=2)
x_train_pca = pca.fit_transform(x_train_scaled)
x_test_pca = pca.transform(x_test_scaled)

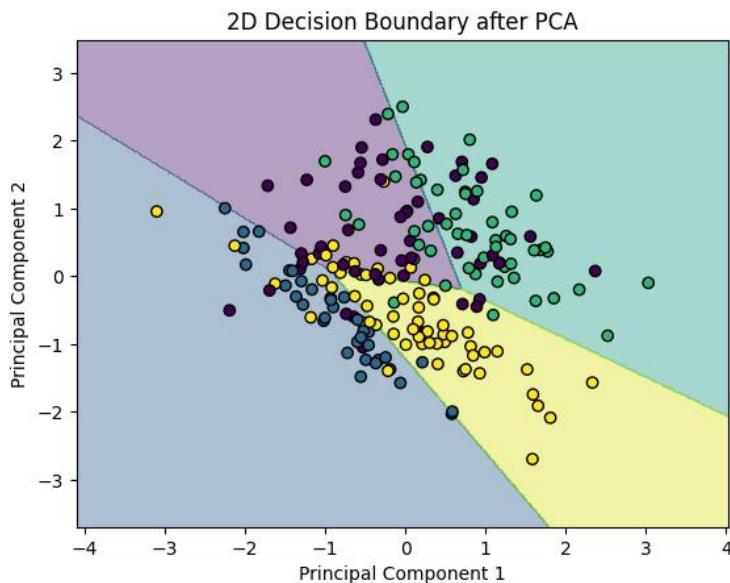
# Fit the model on the reduced data
LogReg_model.fit(x_train_pca, y_train)

# Create a mesh grid for the 2D plot
x_min, x_max = x_test_pca[:, 0].min() - 1, x_test_pca[:, 0].max() + 1
y_min, y_max = x_test_pca[:, 1].min() - 1, x_test_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                      np.arange(y_min, y_max, 0.02))

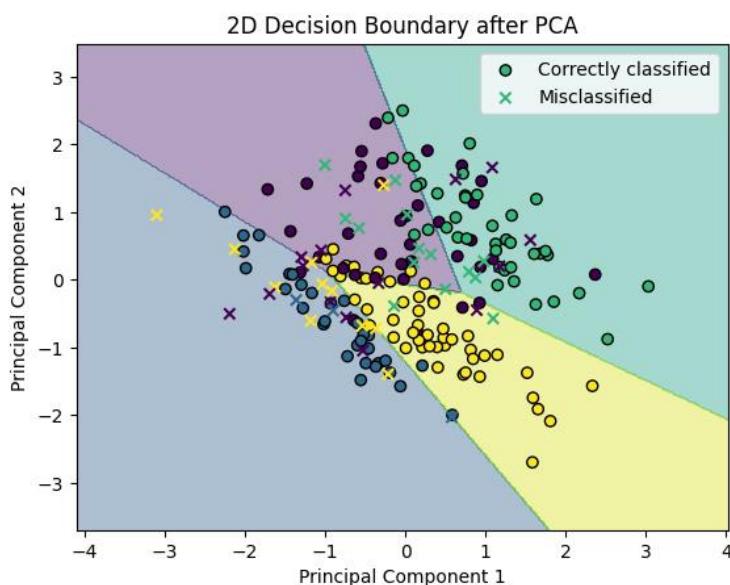
# Predict on the mesh grid
z = LogReg_model.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, z, alpha=0.4)
plt.scatter(x_test_pca[:, 0], x_test_pca[:, 1], c=y_test, edgecolors='k')
plt.title('2D Decision Boundary after PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

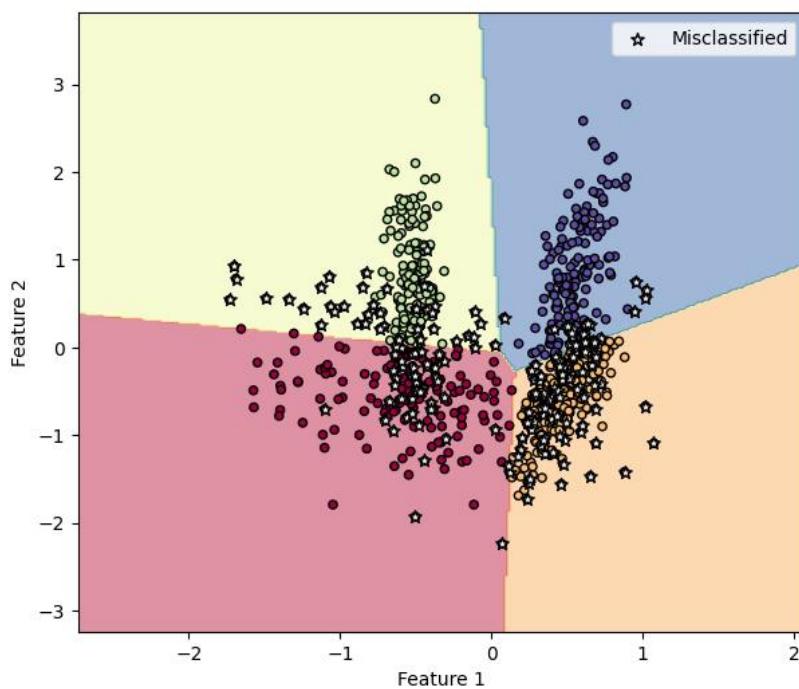
نواحی به صورت زیر خواهد بود:



و داده های missclass داده هایی هستند که مقدار پیش بینی آن ها و مقدار اصلی آن ها یکی نیست. پس همین منطق را در کد هم پیاده می کنیم و داریم:

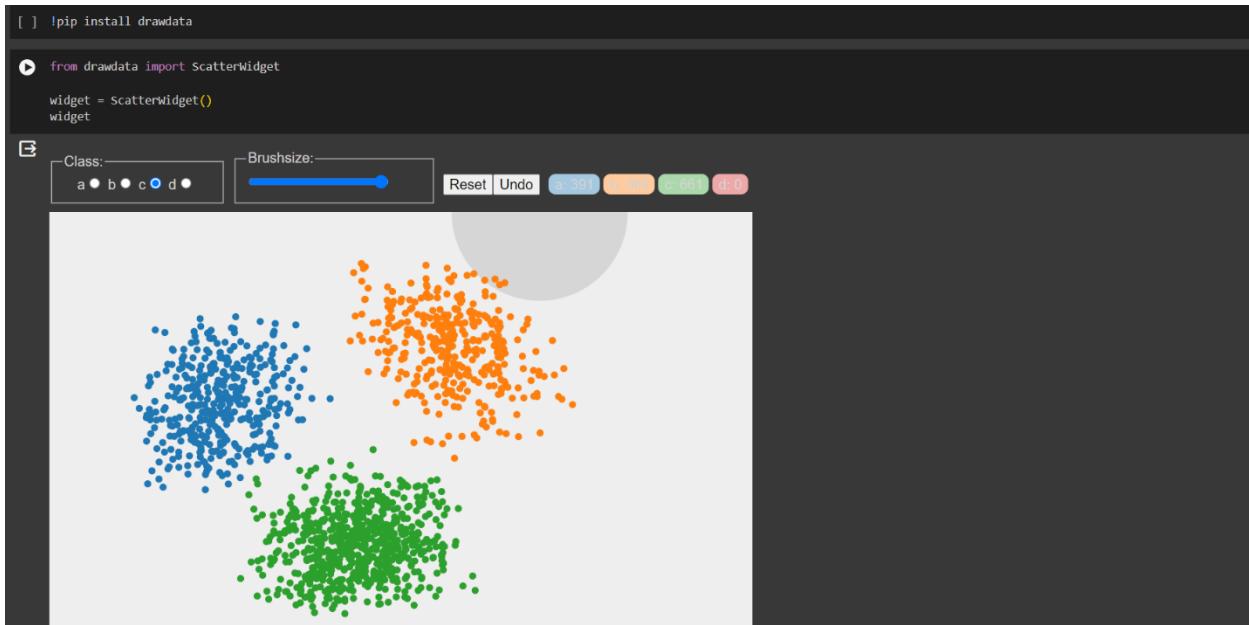


البته یک راه دیگر نیز وجود دارد و آن این است که تنها ۲ ویژگی را انتخاب کنیم و مرز نواحی مربوط به این دو ویژگی را به نمایش بگذاریم. در این صورت می‌توان به صورت زیر عملکرد طبقه‌بند را نشان داد:



بخش ۵

به صورت زیر با کمک drawdata داده می‌سازیم:



در ابتدا یک داده بسیار راحت برای عمل classification تولید می‌کنیم:

```
# df = pd.read_csv('iris.csv')
df = widget.data_as_pandas()
df
```

	x	y	color	label
0	131.045886	227.220730	#1f77b4	a
1	172.528524	262.936192	#1f77b4	a
2	166.424891	259.396766	#1f77b4	a
3	194.963622	238.256480	#1f77b4	a
4	115.347289	269.677400	#1f77b4	a
...
1416	418.402877	127.224556	#2ca02c	c
1417	400.942668	156.331144	#2ca02c	c
1418	384.980030	171.386526	#2ca02c	c
1419	434.162934	194.539604	#2ca02c	c
1420	395.326959	151.199836	#2ca02c	c

1421 rows × 4 columns

این دیتاست را به ما خواهد داد.

ستون color را جدا خواهیم کرد چون فقط به ها و label نیاز داریم. نیاز است تا a و b و c را shuffle کنیم.

```
[50] shuffled_df = df.sample(frac = 1).reset_index(drop = True)
shuffled_df
```

	x	y	label
0	390.298520	104.784087	c
1	453.597490	381.692875	b
2	300.125915	161.407298	c
3	359.659788	437.721503	b
4	555.022725	290.235566	b
...
1416	304.681253	123.588555	c
1417	407.640050	164.636682	c
1418	317.840603	188.655965	c
1419	470.032599	323.478052	b
1420	216.645327	289.554886	a

1421 rows × 3 columns

حالا دیتاست امده است تا عمل train_test_split را انجام دهیم.

```
[51] X = shuffled_df[["x", "y"]]
y = shuffled_df['label']
X.shape, y.shape
((1421, 2), (1421,))

[52] X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 54, test_size = 0.2)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
((1136, 2), (285, 2), (1136,), (285,))
```

حال به کمک ۳ مدل خطی کار classification را انجام می‌دهیم. طبقه بند perceptron و SGDClassifier را روی دیتاست فیت می‌کنیم.

:LR مدل ابتدا

```
[56] model1 = LogisticRegression(solver = 'sag', max_iter = 400, random_state = 54)
model1.fit(X_train, y_train)

[57] y_pred = model1.predict(X_test)

[58] accuracy_logreg = accuracy_score(y_test,y_pred)
print(f"for logistic regression the accuracy is {accuracy_logreg*100:.2f}%")
for logistic regression the accuracy is 92.63%
```

به دقت 92.63% رسیدیم. توضیحات هایپرپارامترها قبل از داده شده و از بازگویی آن خودداری می‌شود.

```

classification_report(y_test,y_pred)
print(classification_report)

precision    recall   f1-score   support
a            0.94    0.97    0.95     87
b            0.87    0.81    0.84     67
c            0.94    0.96    0.95    131

accuracy          0.93    285
macro avg       0.92    0.91    0.91    285
weighted avg    0.93    0.93    0.93    285

```

دیده می شود که در کلاس a و c الگوریتم به خوبی عمل کرده و وضعیت در کلاس b به نسبت بدتر است ولی در کل کار طبقه بندی به خوبی انجام شده است.

:SGDClassifier

```

[62] model2 = SGDClassifier(loss = 'log', random_state = 54)
model2.fit(X_train,y_train)

SGDClassifier()
SGDClassifier(loss='log', random_state=54)

y_pred= model2.predict(X_test)

[64] accuracy = accuracy_score(y_test,y_pred)
print(f"for stochastic gradient decent classifier the accuracy is {accuracy*100:.2f}%")


for stochastic gradient decent classifier the accuracy is 90.88%

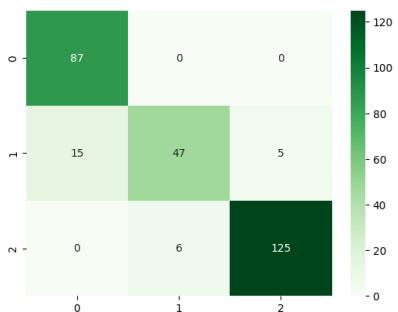
[65] classification_report = classification_report(y_test,y_pred)
print(classification_report)

precision    recall   f1-score   support
a            0.85    1.00    0.92     87
b            0.89    0.70    0.78     67
c            0.96    0.95    0.96    131

accuracy          0.90    285
macro avg       0.90    0.89    0.89    285
weighted avg    0.91    0.91    0.91    285

```

در این حالت به دقت 90.88% می رسمیم. با توجه به metric ها، کلاس c بهتر از بقیه کلاس ها طبقه بندی شده و طبقه بند در کلاس b بدتر عمل کرده است.



Confusion matrix هم به این صورت است. دیده می شود که در این حالت طبقه بند عملکرد خوبی دارد.

:Perceptron

```

from sklearn.linear_model import Perceptron
model = Perceptron(penalty='l2', alpha=0.0001, max_iter=1000, tol=1e-3, random_state=54)

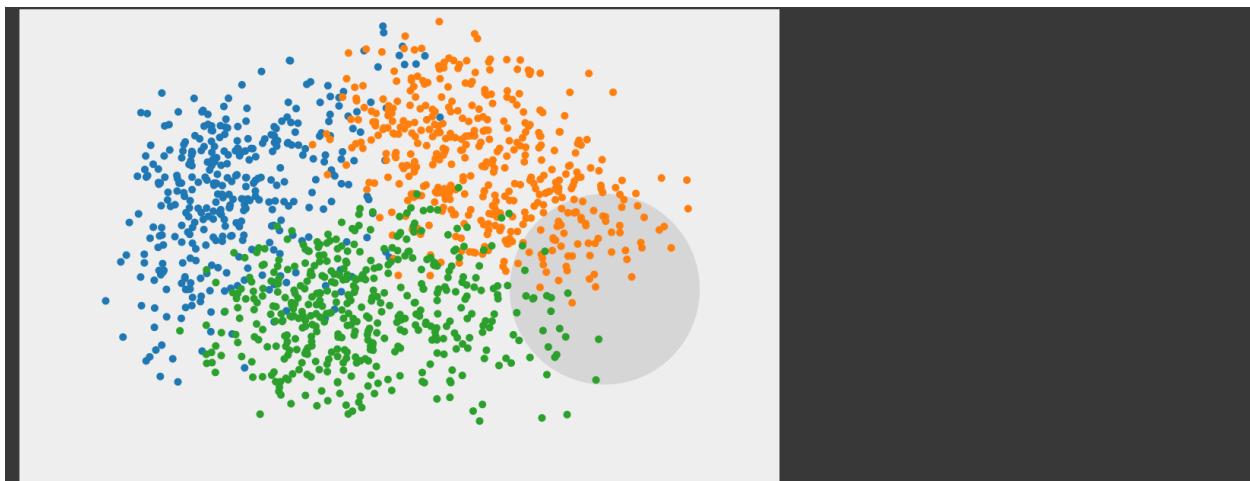
model.fit(X_train, y_train)
y_pred= model2.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
print(f"for Perceptron classifier the accuracy is {accuracy*100:.2f}%")


for Perceptron classifier the accuracy is 90.88%

```

در این حالت دقت 90.88% داریم، در واقع این روش مانند روش SGD است.

حال اگر داده ها را به صورت زیر تولید کنیم:



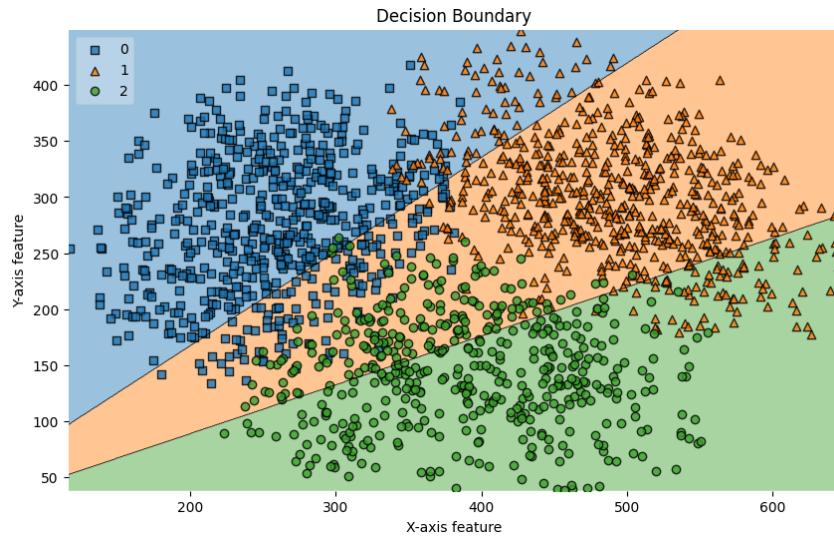
```
[78] accuracy_logreg = accuracy_score(y_test,y_pred)
print(f"for logistic regression the accuracy is {accuracy_logreg*100:.2f}%")
for logistic regression the accuracy is 61.94%
```

```
[83] accuracy = accuracy_score(y_test,y_pred)
print(f"for stochastic gradient decent classifier the accuracy is {accuracy*100:.2f}%")
for stochastic gradient decent classifier the accuracy is 55.87%
print(f"for Perceptron classifier the accuracy is {accuracy*100:.2f}%")
for Perceptron classifier the accuracy is 55.87%
```

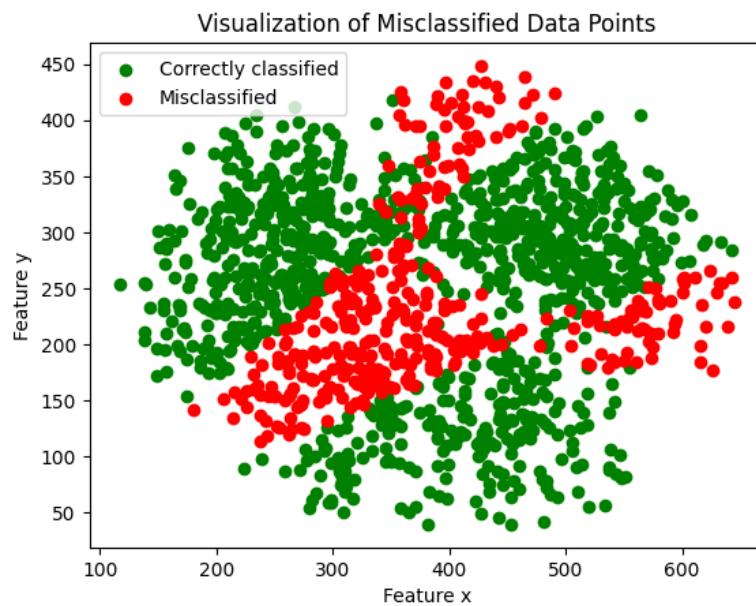
نتایج به صورت بالا است و همانطور که دیده می شود با سخت تر شدن کار طبقه بندی، عملکرد طبقه بند خطی بدتر می شود. همینطور دقت شود که کم بودن دیتا هم تاثیرگذار است.

برای نشان دادن مرز تصمیم گیری از mlxtend استفاده می کنیم ولی قبل از آن باید داده های categorical در متغیر وابسته را به داده ها عددی تبدیل کنیم. برای اینکار از mlxtend استفاده می کنیم:

در نهایت به شکل زیر می‌رسیم:



که در آن لیبل ها از a,b,c به ۰,۱,۲ تغییر یافته اند. برای نشان دادن داده های missclass به صورتی عمل میکنیم که داده ها را رسم می کنیم و آنها بی که y اشان با y_pred اشان با y اصلی برابر نیست با رنگ دیگر نشان می دهیم:



(سوال ۲)

(بخش ۱)

مجموعه داده CWRU ، ارائه شده توسط دانشگاه Case Western Reserve، یک مجموعه داده منبع باز و پراستفاده است که به راحتی قابل دسترسی است. مجموعه داده حامله CWRU به عنوان مرجع استاندارد و مبنایی برای اعتبارسنجی عملکرد انواع الگوریتم‌های یادگیری ماشین (ML) و یادگیری عمیق (DL) شناخته می‌شود. در این پایگاه داده، داده‌ها برای بلبرینگ نرمال، نقطه تک درایو (DE) و نقطه پایانی فن (FE) جمع‌آوری شده‌اند.

مجموعه داده بلبرینگ CWRU با استفاده از یک ترتیب تجهیزات آزمایشی تشکیل شده است که شامل یک ۲ اسپ بخار، یک ترانسیدیوسر گشتاور، یک دینامومتر و کنترل می‌باشد. این مجموعه داده شامل انواع مختلفی از عیوب است:

مجموعه داده شامل ۱۶۱ رکورد است که به چهار کلاس تقسیم شده است:

48k بیشینه طبیعی

48k عیب سمت درایو

12k عیب سمت درایو

12k عیب سمت فن

هر کلاس به زیر مجموعه‌هایی برای انواع مختلف عیوب تقسیم می‌شود:

عیب بلبرینگ (B)

عیب داخلی

عیوب خارجی، طبق مکان نسبت به ناحیه بار دهی دسته‌بندی می‌شوند:

'مرکزی' (عیب در موقعیت ساعت ۰۰:۰۰)

'عمودی' (موقعیت در ساعت ۰۰:۳۰)

'مخالف' (ساعت ۰۰:۱۲)

برای ایجاد این عیوب، ماشینکاری از نوع الکترو-تخریبی (ماشینکاری تخلیه الکتریکی) بر روی بلبرینگ‌های آزمایشی استفاده شد. عیوب با قطرهای خاصی مشخص شده‌اند، به عنوان مثال ۷ میل، ۱۴ میل، ۲۱ میل، ۲۸ میل و ۴۰ میل. (لازم به ذکر است که یک میل برابر با 0.00002π اینچ است).

جمع‌آوری داده‌ها شامل دو فرکانس نمونه‌گیری متفاوت بود: ۱۲ کیلوهرتز و ۴۸ کیلوهرتز. داده‌های ارتعاش برای بارهای موتور در محدوده ۰ تا ۳ اسب بخار، با سرعت‌های موتور بین ۱۷۲۰ و ۱۷۹۷ RPM ثبت شد.

برای نام فایل‌های داده، حرف اول موقعیت خط را نشان می‌دهد، سه عدد بعدی نشان دهنده قطر خط و آخرین عدد نشان دهنده بارهای تحمل کننده است. به عنوان مثال، فایل داده "B007_0" حاوی داده‌های خطای بلبرینگ با قطر خط ۷، ۰، ۰ اینچ است که تحت بار موتور ۰ اسب بخار کار می‌کند.

$$54\%4 = 2 \rightarrow X = 2$$

داده‌های مورد نظر دریافت شدند:

Name	Value
ans	1
X098_DE_time	483903x1 double
X098_FE_time	483903x1 double
X099_DE_time	485063x1 double
X099_FE_time	485063x1 double
X107_BA_time	122136x1 double
X107_DE_time	122136x1 double
X107_FE_time	122136x1 double
X107RPM	1748

حالا باید داده‌ها را از مطلب به پایتون بیاوریم. برای اینکار از کتابخانه `import scipy.io` استفاده می‌کنیم.

به صورت زیر، تمام داده‌های نرمال را در یک دیتا فرم میریزیم. بعده می‌توانیم هر کدام از انواع داده را در کلاس استفاده کنیم.

```
mat_data = scipy.io.loadmat('/content/99.mat')

normal_variables = ['X098_DE_time', 'X098_FE_time', 'X099_DE_time', 'X099_FE_time']

dfs = []
for i in normal_variables:
    data = mat_data[i]
    dfs.append(pd.DataFrame(data))
normal_df = pd.concat(dfs, axis=1, ignore_index=True)
normal_df.columns = normal_variables
print(normal_df)
print(f'null items: \n {normal_df.isnull().sum()}')


X098_DE_time X098_FE_time X099_DE_time X099_FE_time
0      0.046104      0.023216      0.064254      0.038625
1     -0.037134      0.081155      0.063002      0.096769
2     -0.089496      0.095331     -0.004381      0.127382
3     -0.084906      0.091016     -0.035882      0.144640
4     -0.038594      0.038625     -0.023991      0.086702
...
485058        ...        ...        ...
485059        NaN        NaN      0.023991     -0.046022
485059        NaN        NaN      0.034004     -0.027736
485060        NaN        NaN      0.005215      0.031640
485061        NaN        NaN     -0.065714      0.113000
485062        NaN        NaN     -0.122666      0.162925

[485063 rows x 4 columns]
null items:
X098_DE_time    1160
X098_FE_time    1160
```

برای داده عیب هم همینکار را تکرار می‌کنیم.

```
▶ mat_data = scipy.io.loadmat('/content/107.mat')

fault_variables = ['X107_BA_time', 'X107_DE_time', 'X107_FE_time']
# RPM = 1748
dfs = []
for i in fault_variables:
    data = mat_data[i]
    dfs.append(pd.DataFrame(data))
fault_df = pd.concat(dfs, axis=1, ignore_index=True)
fault_df.columns = fault_variables
print(fault_df)
print(f'null items: \n {fault_df.isnull().sum()}')


X107_BA_time X107_DE_time X107_FE_time
0      0.048808     -0.093238     -0.029175
1      0.052912      0.187288     -0.186347
2     -0.056815      0.217663     -0.341260
3     -0.108520      0.070172     -0.160871
4     -0.018228      0.100385      0.179978
...
122131      0.062609     -0.162273      0.093482
122132      0.139221      0.134171     -0.152858
122133      0.078704      0.065624      0.084853
122134     -0.010019     -0.184039      0.067184
122135     -0.002776     -0.040446     -0.094304

[122136 rows x 3 columns]
null items:
X107_BA_time    0
X107_FE_time    0
```

بخش (۲)

قسمت الف) برای کلاس نرمال از داده های "X099_FE_time" و برای کلاس عیب از داده های

$$\begin{cases} M = 150 \\ N = 300 \end{cases}$$

استفاده می کنیم. سپس باید دو ماتریس $M \times N$ تشکیل دهیم.

برای این منظور یک پنجره به طول ۳۰۰ در نظر می گیریم و ۱۵۰ بار از کلاس نمونه برداری می کنیم. در نهایت همه داده ها را در یک np.array ذخیره می کنیم:

```
▶ normal_data = np.array(normal_df['X099_FE_time'])
num_sample = 150
size_sample = 300
normal_matrix = np.zeros((num_sample, size_sample))

for i in range(num_sample):
    start_index = i * size_sample
    end_index = start_index + size_sample
    normal_matrix[i] = normal_data[start_index : end_index]
# normal_matrix = np.hstack((normal_matrix,label_normal))
print(normal_matrix)
normal_matrix.shape

[[[ 0.06425354  0.06300185 -0.00438092 ...  0.11494708  0.06696554
   0.00104308]
 [-0.02524246  0.00250338  0.04568677 ...  0.01668923  0.02107015
   0.04234892]
 [ 0.03775938  0.00688431 -0.03755077 ...  0.113904   0.08866154
   0.03984554]
 ...
 [-0.00312923  0.01856677  0.00792738 ...  0.11056615  0.08490646
   0.01126523]
 [-0.06633969 -0.08574092 -0.03108369 ...  0.03921969  0.01585477
   0.03755077]
 [ 0.05570031  0.02712   -0.010848   ...  0.05424   0.02065292
   -0.02795446]]
(150, 300)
```

```
▶ fault_data = np.array(fault_df['X107_DE_time'])
num_sample = 150
size_sample = 300
fault_matrix = np.zeros((num_sample, size_sample))

for i in range(num_sample):
    start_index = i * size_sample
    end_index = start_index + size_sample
    fault_matrix[i] = fault_data[start_index : end_index]
# fault_matrix = np.hstack((fault_matrix,label_fault))
print(fault_matrix)
fault_matrix.shape

[[[-0.09323776  0.1872877  0.21766307 ...  0.72169928 -0.24170347
   -0.56300016]
 [ 0.36076842  0.25323637 -0.46001629 ...  0.07959321 -0.38773265
   0.30082986]
 [ 0.75905936 -0.21392707 -0.31382467 ... -0.80843964  0.01266994
   0.10233413]
 ...
 [ 0.00341114  0.10964371  0.09372507 ... -0.0994103  0.0545782
   0.10736962]
 [-0.15447581 -0.06692327  0.15415094 ... -0.08381653  0.18550092
   0.16941984]
 [-0.0508422  -0.0841414  -0.11857764 ...  0.24608922  0.01510647
   -0.09811082]]
(150, 300)
```

این دو آرایه، ۱۵۰ سطر دارند که همان تعداد نمونه گیری ماست. ۳۰۰ ستون نیز دارند که اندازه نمونه گیری یا تعداد نمونه ها در هر دسته است. در واقع هر batch در یک سطر ذخیره شده.

در ادامه داده های این دو کلاس را به صورت عمودی زیر هم قرار می دهیم: (با کمک np.vstack())

```
[49] mix_class_data = np.vstack((normal_matrix, fault_matrix))
print(mix_class_data)
mix_class_data.shape

[[ 0.06425354  0.06300185 -0.00438092 ...  0.11494708  0.06696554
  0.00104308]
 [-0.02524246  0.00250338  0.04568677 ...  0.01668923  0.02107015
  0.04234892]
 [ 0.03775938  0.00688431 -0.03755077 ...  0.113904   0.08866154
  0.03984554]
 ...
 [ 0.00341114  0.10964371  0.09372507 ... -0.0994103   0.0545782
  0.10736962]
 [-0.15447581 -0.06692327  0.15415094 ... -0.08381653  0.18550092
  0.16941984]
 [-0.0508422  -0.0841414  -0.11857764 ...  0.24608922  0.01510647
 -0.09811082]]
(300, 300)
```

می بینیم که در حال حاضر یک آرایه 300×300 داریم که ۱۵۰ سطر اول آن داده های نرم و ۱۵۰ سطر بعدی داده های خطا هستند.

سپس با توجه به توضیحات بالا برای داده های نرمال برچسب^۱ و برای داده های خطا برچسب ۱ می زنیم.

```
[50] label_normal = np.zeros((num_sample,1)) # 0 label for NORMAL data
label_fault = np.ones((num_sample,1)) # 1 label for fault data
mix_class_labels = np.vstack((label_normal, label_fault))
mix_class_labels.shape

(300, 1)
```

حال ما این آرایه را داریم و بعدا می توانیم آن را به آرایه کلی (شامل داده ها و ویژگی ها) اضافه کنیم.

قسمت ب) در مرحله بعد باید کار استخراج ویژگی ها را انجام دهیم. ویژگی های زیر از داده ها استخراج شدند:

Feature	Formula	Feature	Formula
Standard Deviation	$x_{std} = \sqrt{\frac{\sum_{i=1}^N (x(i) - \bar{x})^2}{N}}$	Shape Factor	$SF = \frac{x_{rms}}{\frac{1}{N} \sum_{i=1}^N x(i) }$
Peak	$x_p = \max x(i) $	Impact Factor	$IF1 = \frac{1}{N} \sum_{i=1}^N x(i) $
Skewness	$x_{ske} = \frac{\frac{1}{N} \sum_{i=1}^N (x(i) - \bar{x})^3}{x_{std}^3}$	Square Mean Root	$x_{smr} = \left(\frac{1}{N} \sum_{i=1}^N \sqrt{ x(i) } \right)^2$
Kurtosis	$x_{kur} = \frac{\frac{1}{N} \sum_{i=1}^N (x(i) - \bar{x})^4}{x_{std}^4}$	Mean	$\text{Mean} = \frac{1}{n} \sum_{i=1}^n x_i$
Crest Factor	$CF = \frac{x_p}{x_{rms}}$	Absolute Mean	$\text{Abs Mean} = \frac{1}{n} \sum_{i=1}^n x_i $
Clearance Factor	$CLF = \frac{x_p}{x_{smr}}$	Root Mean Square	$RMS = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}$
Peak to Peak	Maximum - Minimum	Impulse Factor	$IF2 = \frac{\text{Abs Max}}{\frac{1}{n} \sum_{i=1}^n x_i }$

¹ label

² feature

برای تعریف ویژگی ها از کتابخانه numpy استفاده شده و برای برخی از ویژگی ها مانند shape factor از سایر ویژگی هایی که قبلاً تعریف شد استفاده شده است.

هر ویژگی به طور جدا برای هر سطر (batch) استخراج می شود و در نهایت ما برای هر λ ویژگی داریم.
برای مثال `:std`

```
feature_std = np.std(mix_class_data, axis=1)
print(f"Standard deviation for each row: \n {feature_std}")
len(feature_std)
# x = np.hstack((mix_class_data, feature_std.reshape(-1,1)))
# print(x)

Standard deviation for each row:
[0.05840032 0.06137898 0.059578 0.05887425 0.06046178 0.0581098
 0.05611645 0.07120636 0.06989548 0.06390619 0.06540074 0.06163291
 0.06528472 0.05560958 0.0655745 0.06458427 0.06649353 0.05524422
 0.06355516 0.06735985 0.06371603 0.07330857 0.05518778 0.06200603
 0.05872442 0.06425147 0.07140902 0.06999041 0.05402161 0.0710969
 0.06214279 0.05870062 0.06118715 0.05832542 0.06492229 0.05786643
 0.06095264 0.0734294 0.06312474 0.05379645 0.06304174 0.06284055
 0.05963792 0.0663993 0.06446255 0.05735695 0.06814944 0.05935844
 0.06268549 0.0550223 0.05634947 0.06889787 0.05510165 0.06197917
 0.06738519 0.06016532 0.0627038 0.05661353 0.06709536 0.06423047
 0.05727389 0.06038576 0.07412215 0.06115466 0.07268951 0.06508967
 0.05834786 0.06370183 0.06203905 0.0638028 0.06722419 0.0589129
 0.06382726 0.07016517 0.063598 0.06201245 0.07464221 0.06080322
 0.06629185 0.06533225 0.06344281 0.07719892 0.06482696 0.05646983
 0.06320456 0.06483749 0.0641301 0.06416409 0.06039018 0.06897239
 0.06061096 0.06547172 0.07251514 0.05335621 0.0547361 0.07419405
 0.04703136 0.06257729 0.06585956 0.05923364 0.07103674 0.07269871
 0.06522263 0.06443529 0.05392825 0.06180017 0.06388172 0.05728464
 0.06195836 0.06962603 0.0568123 0.05764513 0.05957278 0.05942337]
```

دقت شود که در نهایت طول این آرایه ۳۰۰ است چون همانطور که گفتیم مجموع داده های نرم الerner و خطای داری ۳۰۰ سطر است.

در نهایت به صورت زیر تمامی ویژگی هارا در یک آرایه ذخیره می کنیم:

```
feature_list = [feature_std, feature_peak, feature_mean, feature_abs_mean, features_peak_to_peak, feature_rms, feature_crest_factor, feature_shape_factor]
len(feature_list)
8

[52] mix_class_data = np.hstack((mix_class_data, mix_class_labels))
mix_class_data.shape
(300, 301)

[56] feature_data = np.zeros((300, 1))

for feature in feature_list:
    # Horizontally stack the current feature array with X
    feature_data = np.hstack((feature_data, feature.reshape(-1, 1)))

    # Remove the first column of zeros
feature_data = feature_data[:, 1:]

print("Final feature data:")
print(feature_data)
feature_data.shape
```

```
[5] Final feature data:
[[5.84003212e-02 1.79826462e-01 1.54611815e-02 ... 6.04122972e-02
 2.97665326e+00 1.31090191e+00]
 [6.13789844e-02 1.39772308e-01 1.46830462e-02 ... 6.31107880e-02
 2.21471340e+00 1.20445202e+00]
 [5.95779972e-02 1.62302769e-01 1.50175262e-02 ... 6.14415481e-02
 2.64158007e+00 1.23984352e+00]
 ...
 [3.14808729e-01 1.30386679e+00 6.17524218e-03 ... 3.14869290e-01
 4.14097795e+00 1.43984594e+00]
 [2.95903356e-01 1.58520443e+00 6.31439494e-03 ... 2.95970721e-01
 5.35595017e+00 1.40658873e+00]
 [2.93675703e-01 1.15588838e+00 4.49728729e-03 ... 2.93710136e-01
 3.93547325e+00 1.38156819e+00]]
(300, 8)
```

دیده می شود که ابعاد این آرایه 8×300 است. در واقع برای هر سطر (batch) ۸ ویژگی تعریف شد که همگی در feature_data ذخیره شدند.

حالا برای کامل شدن داده ها، باید نمونه ها را در کنار برچسب ها و ویژگی هایشان به صورت عمودی قرار دهیم.

ابتدا برچسب هارا قرار می دهیم:

```
[52] mix_class_data = np.hstack((mix_class_data, mix_class_labels))
mix_class_data.shape
(300, 301)
```

سپس ویژگی هارا:

```
[59] data_with_label_features = np.hstack((mix_class_data, feature_data))
print(data_with_label_features)
data_with_label_features.shape
[[ 6.42535385e-02 6.30018462e-02 -4.38092308e-03 ... 6.04122972e-02
 2.97665326e+00 1.31090191e+00]
 [-2.52424615e-02 2.50338462e-03 4.56867692e-02 ... 6.31107880e-02
 2.21471340e+00 1.20445202e+00]
 [ 3.77593846e-02 6.88430769e-03 -3.75507692e-02 ... 6.14415481e-02
 2.64158007e+00 1.23984352e+00]
 ...
 [ 3.41113772e-03 1.09643713e-01 9.37250699e-02 ... 3.14869290e-01
 4.14097795e+00 1.43984594e+00]
 [-1.54475808e-01 -6.69232735e-02 1.54150938e-01 ... 2.95970721e-01
 5.35595017e+00 1.40658873e+00]
 [-5.08421956e-02 -8.41413972e-02 -1.18577645e-01 ... 2.93710136e-01
 3.93547325e+00 1.38156819e+00]]
(300, 309)
```

حالا ارایه data_with_label_features شامل داده ها و برچسب و ویژگی ها است.

قسمت ج) در این قسمت باید `train test split` انجام دهیم و کار `data shuffling` انجام دهیم.

برهم زدن داده ها یک تکنیک پیش پردازش است که اغلب برای بهبود یادگیری مدل استفاده می شود. به هم زدن داده ها برای مقابله با مسائل احتمالی ناشی از الگوهای در ترتیب متواالی نمونه های آموزشی طراحی شده است که می تواند منجر به بیش برآذش^۱ شود. علاوه بر این، می تواند به کاهش اثرات عدم تعادل یا اختلاف قابل توجه در توزیع انواع مختلف اشیا یا تصاویر بین مجموعه های آموزشی و اعتبار سنجی کمک کند.^[۱]

برای اینکار از دستور `shuffle` در `scikitlearn` استفاده میکنیم:

```
[64] from sklearn.utils import shuffle
shuffled_data = shuffle(data_with_label_features, random_state=54)

df = pd.DataFrame(shuffled_data)
df
```

4	5	6	7	8	9	...	299	300	301	302	303	304	305	306	307	308
54	0.015020	-0.034630	-0.055492	-0.044435	0.009805	...	-0.186294	0.0	0.061979	0.146865	0.016910	0.052018	0.333159	0.064244	2.286036	1.235038
54	0.009596	-0.012517	0.000626	0.015020	0.007302	...	-0.046730	0.0	0.060611	0.174611	0.007827	0.048446	0.346927	0.061114	2.857124	1.261491
30	0.019610	0.049233	0.042975	0.018150	0.003755	...	0.023365	0.0	0.063443	0.149369	0.013057	0.053683	0.319390	0.064773	2.306049	1.206574
71	1.178142	-0.873901	-0.741029	0.917109	0.041746	...	0.023391	1.0	0.314882	1.178142	0.004646	0.225178	2.344914	0.314916	3.741132	1.398521
19	0.046781	-0.133197	-0.211166	0.018355	0.071959	...	-0.364667	1.0	0.302780	1.400191	0.004017	0.219109	2.390395	0.302806	4.624047	1.381988
...
21	0.044644	-0.003546	-0.007510	0.036925	0.100553	...	-0.049650	0.0	0.055244	0.125795	0.013894	0.048061	0.254302	0.056965	2.208304	1.185260
26	0.006173	0.294657	0.178354	-0.085603	-0.090314	...	-0.216364	1.0	0.304438	1.243603	0.005632	0.214939	2.430030	0.304490	4.084212	1.416636
32	-0.014294	0.013351	0.048925	0.055422	0.005124	...	0.014294	0.0	0.002906	0.140815	0.002912	0.002535	0.002472	0.002062	1.214559	

برای `train-test-split` و راحت تر خواندن دیتا، آن را به `pd.DataFrame` تبدیل می کنیم.

حالا زمان جدا کردن متغیر های مستقل (X) (ویژگی ها) و وابسته (y) یا همان لیبل است.

```
[80] x = df.iloc[:, 301:309]
y = df.iloc[:, 300]
x.columns = ['feature_std', 'feature_peak', 'feature_mean', 'feature_abs_mean', 'features_peak_to_peak', 'feature_rms', 'feature_crest_factor', 'feature_shape_factor']

[81] x
```

	feature_std	feature_peak	feature_mean	feature_abs_mean	features_peak_to_peak	feature_rms	feature_crest_factor	feature_shape_factor
0	0.061979	0.146865	0.016910	0.052018	0.333159	0.064244	2.286036	1.235038
1	0.060611	0.174611	0.007827	0.048446	0.346927	0.061114	2.857124	1.261491
2	0.063443	0.149369	0.013057	0.053683	0.319390	0.064773	2.306049	1.206574
3	0.314882	1.178142	0.004646	0.225178	2.344914	0.314916	3.741132	1.398521
4	0.302780	1.400191	0.004017	0.219109	2.390395	0.302806	4.624047	1.381988
...
295	0.055244	0.125795	0.013894	0.048061	0.254302	0.056965	2.208304	1.185260
296	0.304438	1.243603	0.005632	0.214939	2.430030	0.304490	4.084212	1.416636
297	0.062006	0.140815	0.012084	0.052013	0.303535	0.063172	2.229062	1.214559

¹ overfitting

² validation

```
[82] y
0      0.0
1      0.0
2      0.0
3      1.0
4      1.0
...
295    0.0
296    1.0
297    0.0
298    1.0
299    0.0
Name: 300, Length: 300, dtype: float64
```

حال به ارزیابی و اموزش تبدیل می‌کنیم:

```
[84] 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 = 54)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
((240, 8), (60, 8), (240,), (60,))
```

به نسبت ۸۰ به ۲۰

قسمت د) دو روش وجود دارد که بسیار استفاده می‌شود:

- StandardScaler
- MinMaxScaler

نرمال سازی استاندارد داده‌ها را با میانگین ۰ و انحراف استاندارد ۱ تغییر مقیاس می‌دهد. این تضمین می‌کند که داده‌ها پس از تبدیل در مقیاسی مشابه با محوریت ۰ قرار خواهند گرفت. فرمول آن به صورت زیر است:

$$\frac{X - \text{mean}}{\text{std}}$$

نرمال سازی min-max برای مقیاس بندی هر ویژگی به یک محدوده مشخص، معمولاً [۰، ۱]، به روشنی استفاده می‌شود که فاصله نسبی بین نقاط داده را حفظ کند. این فرآیند شامل تفریق مقدار حداقل و تقسیم بر دامنه (حداکثر - حداقل) برای رسیدن به نرمال سازی است. فرمول آن به صورت زیر است:

$$\frac{X - \text{min}}{\text{max} - \text{min}}$$

در این قسمت از standardscaler استفاده می‌کنیم و داده‌های آموزش و ارزیابی را scale می‌کنیم.

```
[ ] from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)

▶ print(X_train_sc)
X_train_sc.shape

[[ 1.05378972  1.41921483 -0.93073731 ...  1.05416505  1.65098505
  0.90522577]
 [ 0.73798892  0.51833625 -0.99368771 ...  0.73629923  0.32131202
  0.41073357]
 [ 1.01291361  0.93710995 -0.71030672 ...  1.01318388  0.73501665
  1.23715419]
 ...
 [ 0.97612032  0.8637791  -0.84121018 ...  0.97605753  0.64843759
  0.87698522]
 [-0.95824466 -0.95746141  1.48879751 ... -0.95412379 -0.82561268
 -1.2918482]
 [ 1.09179772  0.590152  -0.77409089 ...  1.09252192 -0.10676383
  0.7122775]
 (240, 8)
```

داده‌ها با میانگین صفر و std ۱ نرمال شدند.

از اطلاعات بخش ارزیابی در نرمال سازی استفاده کردیم چون معمولاً توصیه می‌شود همان مراحل پیش‌پردازش برای داده‌های آزمایشی اعمال شود که برای داده‌های آموزشی انجام شده. این بحث شامل نرمال سازی هم می‌شود.

با عادی سازی داده‌ها از تقسیم، یکپارچگی مجموعه آزمایشی را به عنوان یک معیار ارزیابی بی طرفانه حفظ می‌کنیم. این رویکرد سناریویی را شبیه سازی می‌کند که در آن مدل در حین استقرار با داده‌های دیده نشده مواجه می‌شود.

توجه شود که فرآیند نرمال سازی شامل محاسبه پارامترهای آماری (به عنوان مثال، حداقل، حداکثر، میانگین، انحراف استاندارد) در مجموعه آموزشی و سپس اعمال آن پارامترها برای نرمال کردن مجموعه تست است. این تضمین می‌کند که مجموعه تست کاملاً جدا از فرآیند آموزش باقی می‌ماند.[2]

(بخش ۳)

از الگوریتم LR استفاده می‌شود. توابع به صورت زیر هستند:

```
[66] # import the function needed
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def logistic_regression(x, w):
    y_hat = sigmoid(x @ w)
    return y_hat
def bce(y, y_hat):
    loss = -(y*np.log(y_hat) + (1-y)*np.log(1-y_hat))
    return loss
def gradient(x, y, y_hat):
    grads = (x.T @ (y_hat - y)) / len(y)
    return grads
def gradient_descent(w, eta, grads):
    w -= eta*grads
    return w
def accuracy(y, y_hat):
    acc = np.sum(y == np.round(y_hat)) / len(y)
    return acc
```

دوتابع اول مربوط به خود الگوریتم هستند.تابع اتلاف همان `bce` است و الگوریتم بهینه ساز هم گرادیان کاهشی است.

به صورت جز به جز الگوریتم را جلو می برمیم به این صورت که ابتدا y_{hat} را به دست می آوریم. بعد از آن خط را محاسبه می کنیم و در نهایت به کمک گرادیان کاهشی، ضرایب مدل را بهبود می بخسیم.

```
[70] bce(y_train.values.reshape(-1,1), y_hat)
0.24694922540656955
```

```
[72] eta = 0.01
n_epochs = 15000

❶ error_hist = []

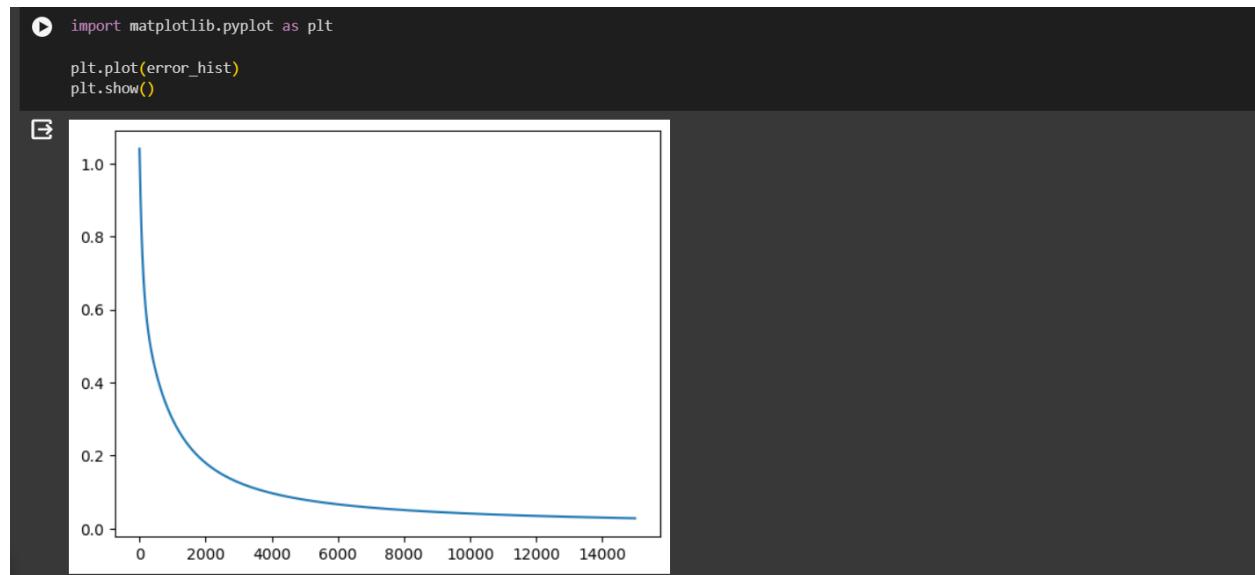
for epoch in range(n_epochs):
    # predictions
    y_hat = logistic_regression(X_train_sc, w)

    # loss
    e = bce(y_train.values.reshape(-1,1), y_hat)
    error_hist.append(e)

    # gradients
    grads = gradient(X_train_sc, y_train.values.reshape(-1,1), y_hat)

    # gradient descent
    w = gradient_descent(w, eta, grads)

    if (epoch+1) % 1 == 0:
        print(f'Epoch={epoch}, \t E={e:.4}, \t w={w.T[0]}')
```



تابع اتلاف به صورت بالا است که می بینیم با گذشتن ایپاک ها، کم و کمتر می شود. این مقدار از کمی بیشتر از ۱ شروع می شود که مقدار دقیق آن گزارش شد. به مرور زمان و با $\eta = 0.01$ که به نوعی بیانگر کنترل ما روی الگوریتم بهینه سازی، خطای کمتر می شود. دیده می شود که به epoch های بیشتر می توانستیم به عملکرد بهتری هم برسیم ولی شبکه نمودار به مرور کمتر خواهد شد.

حال اگر به مسئله یک x_0 یا بایاس اضافه کنیم، به صورت زیر خواهد شد:

```

14s  ● X_train_b = np.hstack((np.ones((len(X_train_sc),1)), X_train_sc))
    w = np.random.randn(X_train_b.shape[1],1)
    error_hist = []

    for epoch in range(n_epochs):
        # predictions
        y_hat = logistic_regression(X_train_b, w)

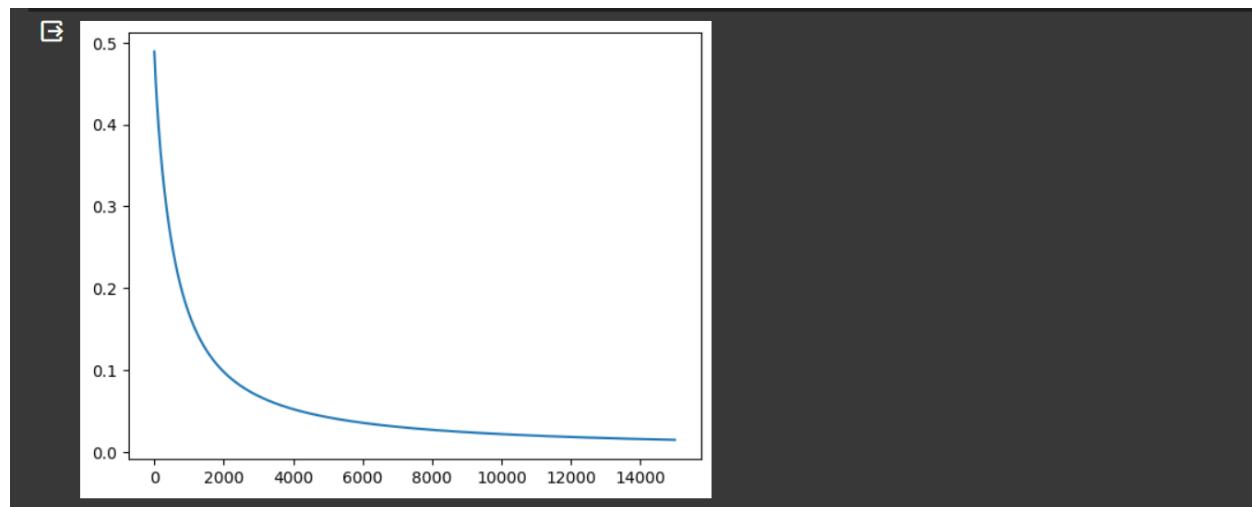
        # loss
        e = bce(y_train.values.reshape(-1,1), y_hat)
        error_hist.append(e)

        # gradients
        grads = gradient(X_train_b, y_train.values.reshape(-1,1), y_hat)

        # gradient descent
        w = gradient_descent(w, eta, grads)

        if (epoch+1) % 1 == 0:
            print(f'Epoch={epoch}, \t E={e:.4}, \t w={w.T[0]}')

```



میبینیم که خطا از مقدار کمتری شروع می‌شود و باز هم به کمک الگوریتم بهینه ساز، مقدراً تابع اتلاف کمتر می‌شود.

مقدار دقت هم به صورت زیر بدست می‌آید: (دقت train)

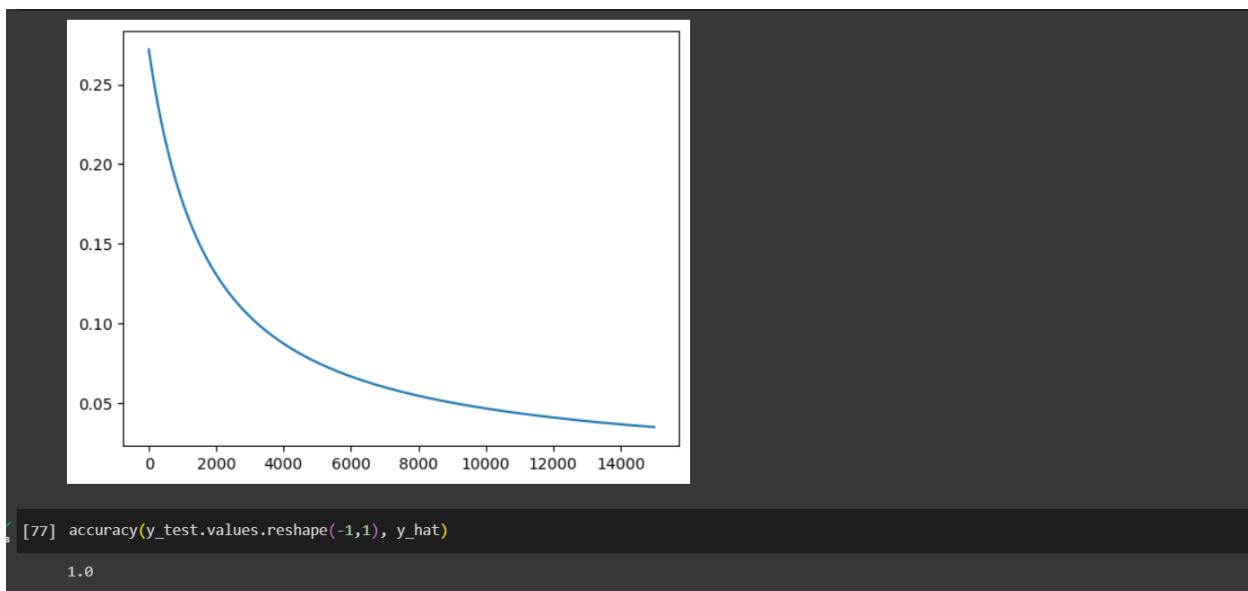
```

1s [73] accuracy(y_train.values.reshape(-1,1), y_hat)
      1.0

```

حال برای داده تست هم همینکار را تکرار می کنیم و نتایج به صورت زیر است:

تابع اتلاف و دقت:



دیده می شود که میزان خطای کمتر است و دقت هم مشابه حالت قبل است.

به طور کلی می توان به کمک تابع اتلاف و نمودار مربوط به آن درباره عملکرد مدل نظر داد. اما این نظر دادن باید به دقت صورت پذیرد. ارزیابی در این مرحله عموما در قالب ارزیابی نسبی شناخته می شود و به عنوان یک شاخص اولیه درنظر گرفته می شود.

در مسائل با پیچیدگی بالاتر ممکن است نمودار اتلاف شاخص مناسبی برای ارزیابی مدل نباشد. راه حل دیگر می تواند استفاده از شاخص های دیگر مانند دقت، confusion matrix و ... باشد.

(بخش ۴)

از LR در سایکیت لرن استفاده می کنیم:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model = LogisticRegression()
model.fit(X_train_sc, y_train)
y_pred = model.predict(X_test_sc)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Accuracy: 1.0

bce و دیده می شود که دقت مشابه حالت قبلی بدست می آید. توجه شود در این حالت نیز تابع اتلاف همان است.

برای کشیدن تابع اتلاف مطابق زیر عمل می‌کنیم:

```

import matplotlib.pyplot as plt
from sklearn.metrics import log_loss

model = LogisticRegression()
model.fit(X_train_sc, y_train)

y_pred_probs = model.predict_proba(X_test_sc) Loading...

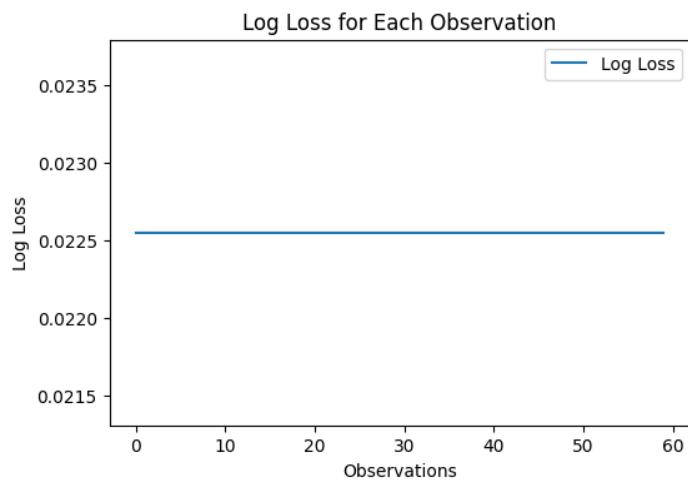
loss = log_loss(y_test, y_pred_probs)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

plt.figure(figsize=(6,4))
plt.plot([loss] * len(y_test), label='Log Loss')
plt.xlabel('Observations')
plt.ylabel('Log Loss')
plt.title('Log Loss for Each Observation')
plt.legend()
plt.show()

```

ابتدا مدل را فیت می‌کنیم، سپس احتمالات پیش بینی شده از مدل را استخراج می‌کنیم. در نهایت $\log \text{loss}$ را محاسبه و آن را رسم می‌کنیم:



سوال (۳)

بخش (۱)

بنظر می‌رسد که در این بخش از ما خواسته شده تا روی دیتا EDA انجام دهیم. در ادامه همین کار را انجام می‌دهیم:

دیتاستی که داریم شامل موارد زیر است:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.

این دیتاست وضعیت آب و هوا در یکی از شهرهای مجارستان بین سال‌های ۲۰۰۶ تا ۲۰۱۶ است. فیچرها در تصویر بالا مواردی از قبیل ساعت و تاریخ میلادی، خلاصه از وضعیت آب و هوا، دما، رطوبت، سرعت باد و ... هستند. برای اطلاعات بیشتر از این مجموعه دیتا:

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96453 entries, 0 to 96452
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Formatted Date    96453 non-null   object 
 1   Summary          96453 non-null   object 
 2   Precip Type      95936 non-null   object 
 3   Temperature (C)  96453 non-null   float64
 4   Apparent Temperature (C) 96453 non-null   float64
 5   Humidity         96453 non-null   float64
 6   Wind Speed (km/h) 96453 non-null   float64
 7   Wind Bearing (degrees) 96453 non-null   float64
 8   Visibility (km)  96453 non-null   float64
 9   Loud Cover       96453 non-null   float64
 10  Pressure (millibars) 96453 non-null   float64
 11  Daily Summary    96453 non-null   object 
dtypes: float64(8), object(4)
memory usage: 8.8+ MB
```

میبینیم که ۱۲ ستون داریم و نوع هر ستون در مقابل آن نوشته شده که از بین این موارد، فیچر های دما تا فشار هوایی هستند. پس هم عددی داریم هم غیرعددی.

df.describe()									
	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	
count	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.0	96453.000000	
mean	11.932678	10.855029	0.734899	10.810640	187.509232	10.347325	0.0	1003.235956	
std	9.551546	10.696847	0.195473	6.913571	107.383428	4.192123	0.0	116.969906	
min	-21.822222	-27.716667	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	
25%	4.688889	2.311111	0.600000	5.828200	116.000000	8.339800	0.0	1011.900000	
50%	12.000000	12.000000	0.780000	9.965900	180.000000	10.046400	0.0	1016.450000	
75%	18.838889	18.838889	0.890000	14.135800	290.000000	14.812000	0.0	1021.090000	
max	39.905556	39.344444	1.000000	63.852600	359.000000	16.100000	0.0	1046.380000	

تعداد داده ها، میانگین، انحراف معیار و پراکندگی را می توان در جدول بالا مشاهده کرد.

تعداد missing value ها:

df.isnull().sum()	
Formatted Date	0
Summary	0
Precip Type	517
Temperature (C)	0
Apparent Temperature (C)	0
Humidity	0
Wind Speed (km/h)	0
Wind Bearing (degrees)	0
Visibility (km)	0
Loud Cover	0
Pressure (millibars)	0
Daily Summary	0
dtype: int64	

در فیچر precip type ۵۱۷ داده ثبت نشده داریم.

با استفاده از کد زیر، می توان به نام ستون ها حالت استانداردتری داد تا در آینده مشکل ساز نشوند:

df.columns = df.columns.str.replace(' ', '_')											
df.head()											
0	Formatted_Date	Summary	Precip_Type	Temperature_(c)	Apparent_Temperature_(C)	Humidity	Wind_Speed_(km/h)	Wind_Bearing_(degrees)	Visibility_(km)	Loud_Cover	Pressure_(millibars)
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1

در دیتا ست تمامی داده های فیچر date صفر هستند. علاوه بر این با فیچر loud_cover کاری نداریم پس آن هارا حذف می کنیم و دیتابست را خلوت تر می کنیم

علاوه بر این برای هندل کردن missing value ها می توانیم رکوردهایی که شامل missing value هستند را حذف کنیم چون تعداد خیلی کمی را شامل می شوند. (۵۰۰ سمپل از بین تقریبا ۱۰۰۰۰۰ سمپل)

بعد از حذف آن ها می بینیم که missing value نداریم:

```
df.dropna(axis = 0, inplace = True)
df.isnull().sum()

Summary          0
Precip_Type      0
Temperature_(C) 0
Apparent_Temperature_(C) 0
Humidity         0
Wind_Speed_(km/h) 0
Wind_Bearing_(degrees) 0
Visibility_(km)   0
Pressure_(millibars) 0
Daily_Summary    0
dtype: int64
```

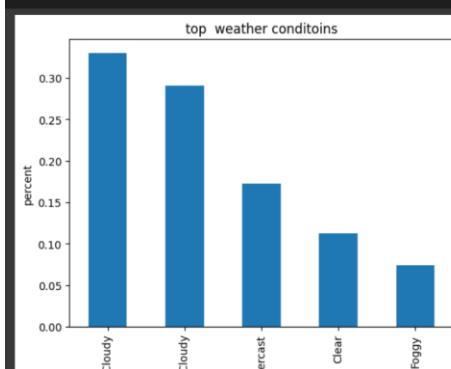
حالا به سراغ فیچر summary می رویم تا کمی درباره آن بحث کنیم:

```
df["Summary"].value_counts()

Partly Cloudy        31635
Mostly Cloudy        27914
Overcast              16516
Clear                 10763
Foggy                 7117
Breezy and Overcast  528
Breezy and Mostly Cloudy 516
Breezy and Partly Cloudy 386
Dry and Partly Cloudy 86
Windy and Partly Cloudy 67
Light Rain             63
Breezy                 54
Windy and Overcast    45
Humid and Mostly Cloudy 40
Drizzle                39
Breezy and Foggy       35
Windy and Mostly Cloudy 35
Dry                     34
Humid and Partly Cloudy 17
Dry and Mostly Cloudy 14
Rain                    10
Windy                  8
Humid and Overcast     7
Windy and Foggy        4
Windy and Dry           1
Dangerously Windy and Partly Cloudy 1
Breezy and Dry          1
```

میبینیم که مقادیر مختلف این فیچر اصلا توزیع نرمالی ندارند و به صورت چشمی، ۵ مقدار اول آن بسیار غالب تر هستند. به منظور نمایش: (توزیع نرمال)

```
plt1 = df["Summary"].value_counts(normalize = True).head(5).plot(kind = 'bar', title = 'top weather conditons')
plt1.set_xlabel('weather summary')
plt1.set_ylabel('percent')
plt1.show()
```

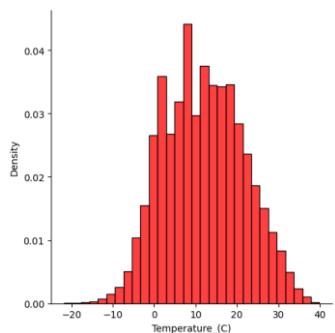


به صورت ریاضی:

```
print(f"percentage of the data with having only top 5 summary situation: {((df['Summary'].value_counts().head(5).sum() / len(df)).round(2))}%")
```

percentage of the data with having only top 5 summary situation: 0.98 %

۹۸ درصد از موقع، یکی از ۵ مورد اول رخ داده است. فعلا این مسئله را اینجا رها می کنیم و بعدا دوباره به آن برミگردیم.

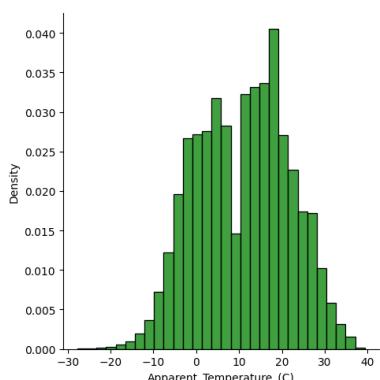


هیستوگرام دما:

این نمودار نرمالایز شده یعنی مساحت زیر آن ۱ است.

دیده می شود که محدوده دمایی از -۲۰ درجه سلسیوس تا ۴۰ درجه است.

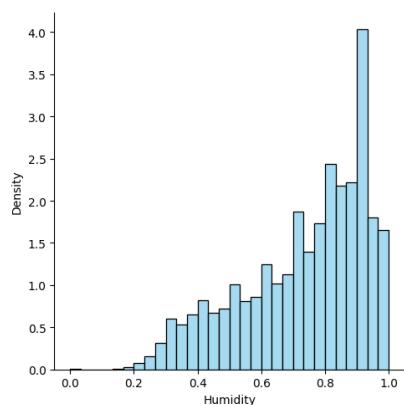
دما پراکندگی نرمال دارد.



هیستوگرام دمای ظاهری:

دیده می شود که پراکندگی دمای ظاهر کمی بیشتر از دمای واقعی است.

این نمودار نیز پراکندگی با الگوی نرمال دارد.

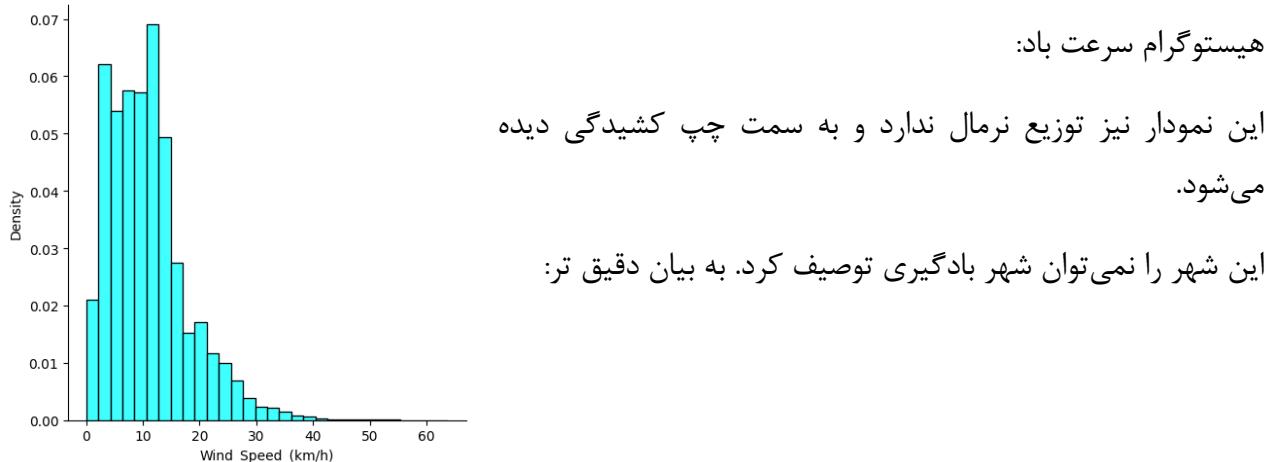


هیستوگرام رطوبت:

این توزیع، نرمال نیست و به صورت چشمی می توان گفت Szeged یک شهر مرطوب است. به طور دقیق تر و به بیان ریاضی:

```
print(f"percentage of days that Humidity is above 70% is: {((df[df.Humidity>0.7].value_counts().sum())/len(df)).round(2)} %")
percentage of days that Humidity is above 70% is: 0.63 %
```

در ۶۳ درصد از مواقع، رطوبت هوا بیشتر از ۷۰ درصد بوده است.



```
print(f"percentage of days that windspeed is between 3 and 17 km/h is: {((df[df['Wind_Speed_(km/h)'].between(3,17)].value_counts().sum())/len(df)).round(2)} %")
percentage of days that windspeed is between 3 and 17 km/h is: 0.78 %
```

در ۷۸ درصد از مواقع، سرعت باد بین ۳ تا ۱۷ کیلومتر بر ساعت است.

حال به سراغ نمودارهای پراکندگی می‌رویم (برداشت من از صورت سوال این بود که باید هیستوگرام و scatter plot را برای ویژگی‌ها بررسی کنیم).

دو نکته:

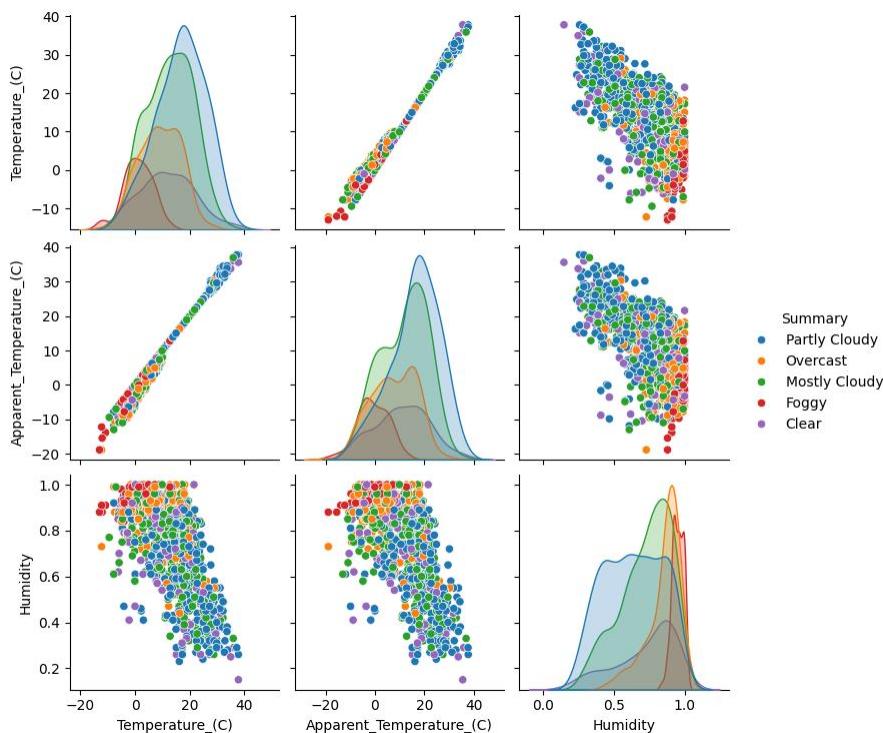
- در این قسمت دو دیتاست را بررسی می‌کنیم. یکی دیتاست اصلی و دیگری، دیتاست خلوت شده با معیار ۵ وضعیت اب و هوایی غالب که بالاتر استخراج کردیم و همینطور فیچرهای مهم در تعیین وضعیت اب و هوایی.

- در این قسمت از pairplot استفاده شده و پراکندگی برای ۱۰۰۰ نمونه رندوم از دیتاست رسم شده است.

۱- دیتا ست برای وضعیت اب و هوایی غالب (۵ مورد اول) :

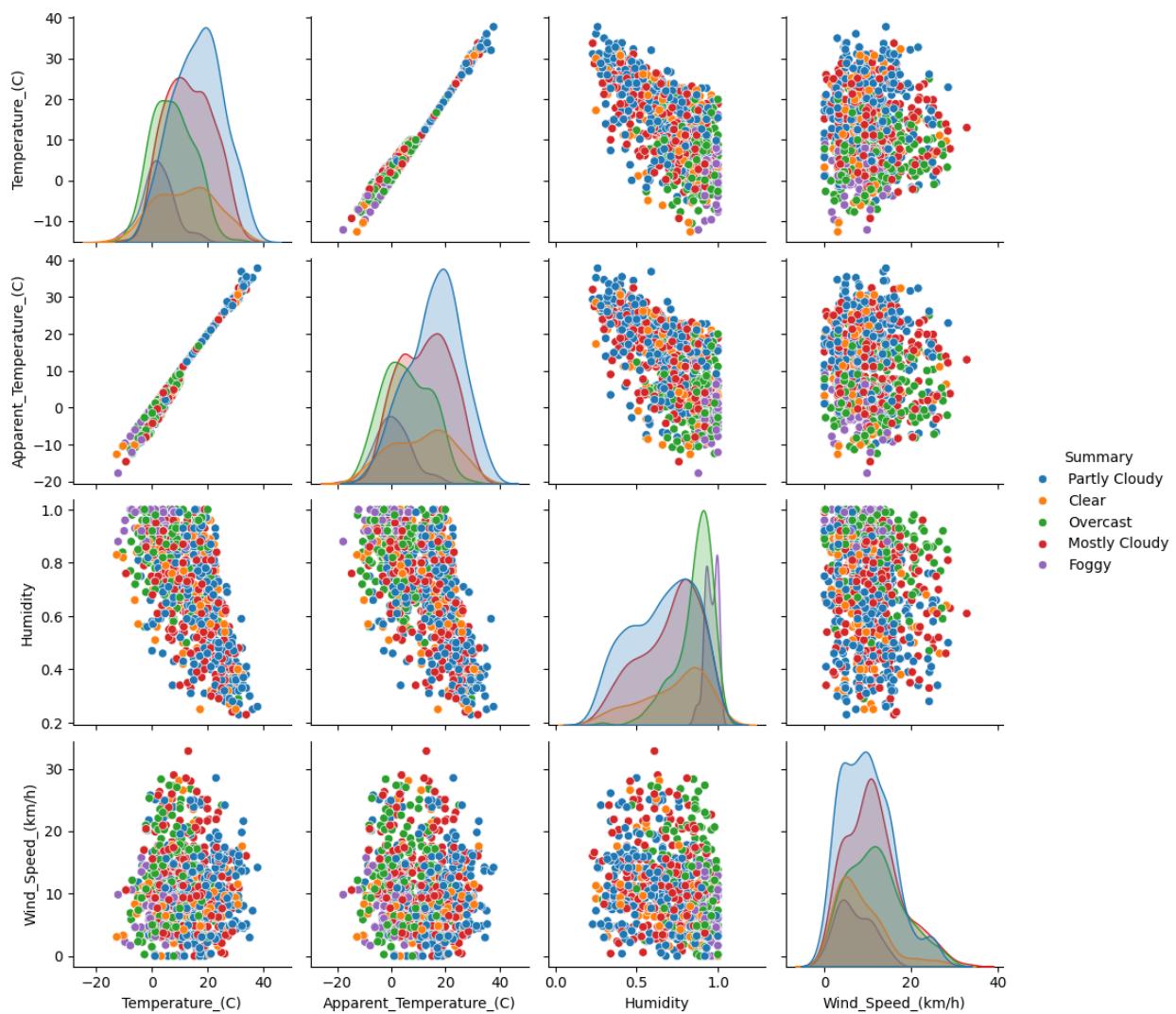
	Summary	Temperature_(C)	Apparent_Temperature_(C)	Humidity	Wind_Speed_(km/h)	Pressure_(millibars)
27153	Mostly Cloudy	22.777778	22.777778	0.66	8.0500	1016.10
88671	Partly Cloudy	25.044444	25.044444	0.57	12.3487	1015.04
64841	Mostly Cloudy	7.311111	5.261111	0.92	10.8997	1014.57
78569	Partly Cloudy	11.427778	11.427778	0.75	10.8675	1017.42

نمودار پراکندگی:



با توجه به legend ها خواندن این نمودار ها کار چالش برانگیز نیست و به راحتی می توان اینکار را انجام داد. دما و دمای ظاهری دو ویژگی هستند که تقریباً رابطه خطی دارند. در روز های نیمه ابری میانگین دما بالاتر است و روزهای مه آلود این میانگین در پایین تر حالت خود قرار دارد. همینطور در دماهای بالا، میزان رطوبت بالا (روز های نیمه ابری و اکثراً ابری) و در دمای پایین رطوبت بیشتر (روزهای مه آلود و ابری) است. در روز های ابری، رطوبت نیز به بیشترین مقدار خود می رسد. البته ما می توانستیم این اطلاعات از حالت بالامثلی یا پایین مثلثی این نمودار هم استخراج کنیم و درواقع این شکل یک ماتریس متقارن است.

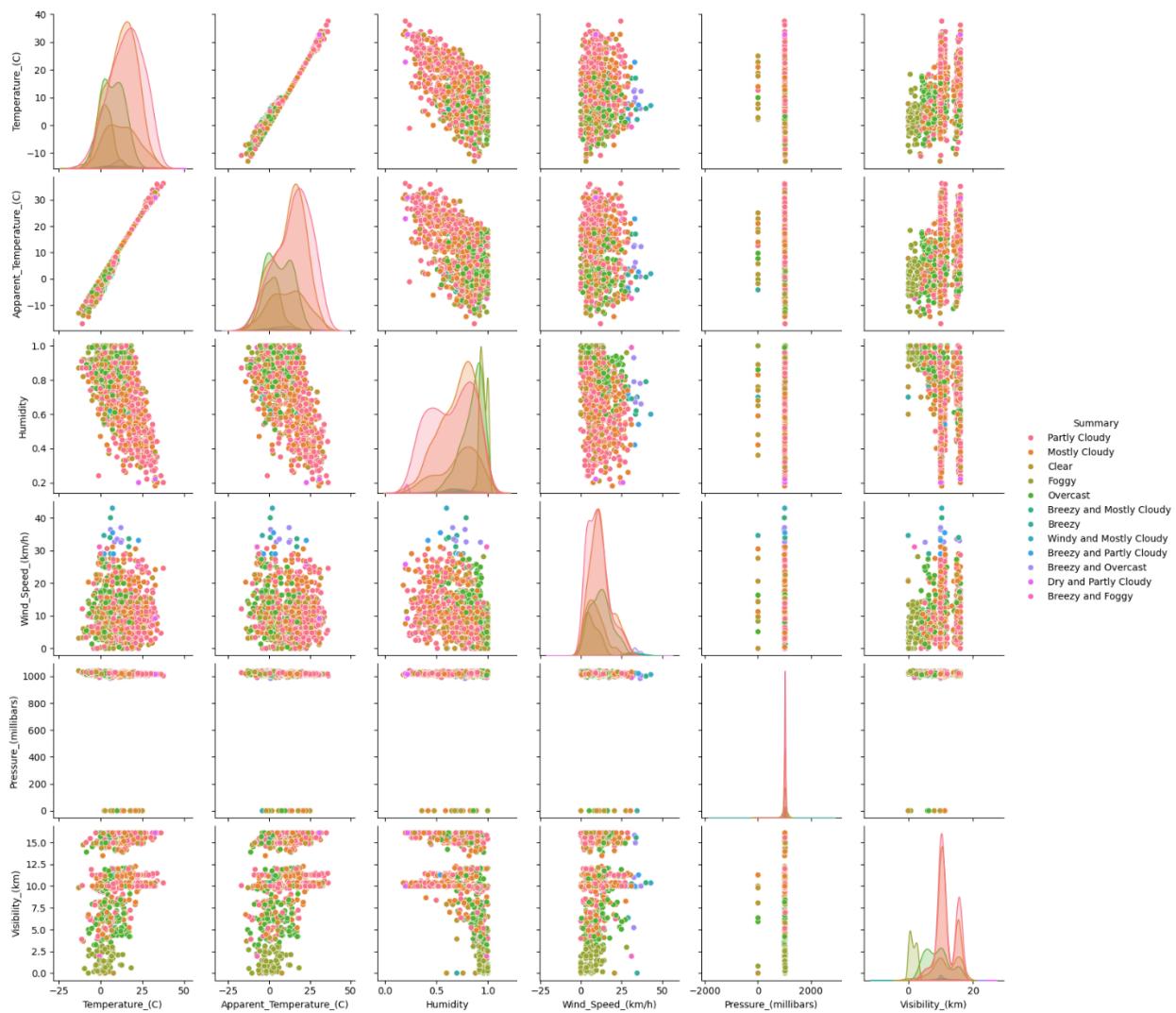
۲- دیتاست شامل ویژگی‌های بیشتر (سرعت باد):



دیده می‌شود در روزهای بخشی ابری سرعت باد بیشتر است و در روزهای مهآلود کمترین سرعت‌ها ثبت شده است. همینطور در روزهای مهآلود بیشترین رطوبت و کمترین سرعت باد را داریم.

درباره رابطه دما و سرعت باد؛ در تمامی دما‌ها، تمامی سرعت‌های داریم و کشیدگی به سمت خاصی دیده نمی‌شود. برای دمای ظاهری هم به همین صورت است.

نمودار کامل به صورت زیر است:



با توجه به زیاد بودن رنگ‌ها، خواندن این نمودار سخت‌تر از حالت‌های قبل است ولی همان روند‌ها دیده می‌شود

چون همانطور که گفته شد، ۹۸ درصد از سمپل‌ها، در حالت‌های قبلی وجود داشتند.

حال سراغ ماتریس همبستگی می‌رویم:

Correlation Matrix							
Temperature_(C)	Apparent_Temperature_(C)	Humidity	Wind_Speed_(km/h)	Wind_Bearing_(degrees)	Visibility_(km)	Pressure_(millibars)	
1.00	0.99	-0.63	0.01	0.03	0.39	-0.01	
0.99	1.00	-0.60	-0.06	0.03	0.38	-0.00	
		1.00	-0.22	0.00	-0.37	0.01	
			1.00	0.10	0.10	-0.05	
				1.00	0.05	-0.01	
					1.00	0.06	
						1.00	
Temperature_(C)	Apparent_Temperature_(C)	Humidity	Wind_Speed_(km/h)	Wind_Bearing_(degrees)	Visibility_(km)	Pressure_(millibars)	

هرچه اعداد به ۱ نزدیک تر باشند، یعنی رابطه آن دو ویژگی در یک جهت حرکت می‌کنند و رابطه قوی تری دارند.
 (مثلا با زیاد شدن یکی، دیگری هم به طور خطی زیاد می‌شود)

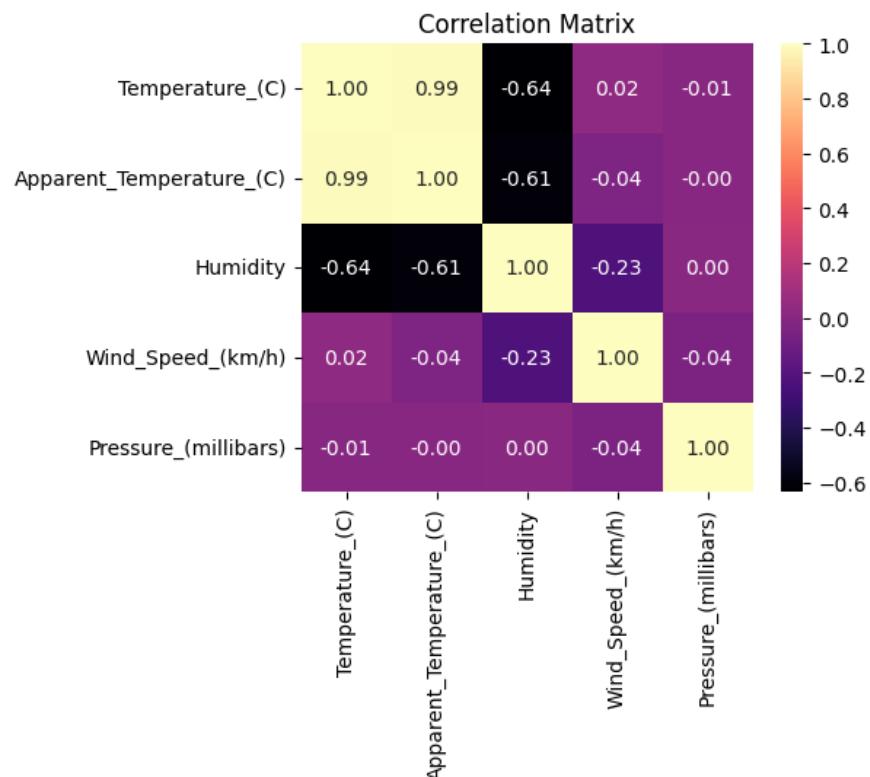
هرچه به ۱- نزدیکتر باشند، در جهت مخالف هم حرکت می‌کنند (معکوس همدیگر)

نزدیکی به صفر یعنی این دو متغیر رابطه خاصی با هم ندارند یا نسبت به هم به صورت تصادفی تغییر می‌کنند.

برای اینکه مسئله پیش بینی انجام دهیم، می‌توانیم ویژگی هایی را انتخاب کنیم که بیشتری همبستگی را با متغیر وابسته دارند. مثلا برای پیش بینی دمای ظاهری از دما و دید استفاده کنیم که همبستگی بالایی با دمای ظاهری دارند. ولی مثلا جهت باد و یا فشار هوا، کمکی به ما نمی‌کنند چون همبستگی با متغیر وابسته ندارند. استفاده از رطوبت هم می‌تواند خوب باشد چون همبستگی بالایی در جهت مخالف دارد یعنی با زیاد شدن دما رطوبت احتمالا کاهش می‌ابد.

البته برای گرفتن نتایج متمایز و کاهش بعد، می‌توان ویژگی هایی را انتخاب کرد که با هم همبستگی کمتری دارد.

ماتریس همبستگی برای دیتاست کوچک شده:



همانطور که دیده می شود ماتریس کوچیک تر است ولی همان نتایج قبلی را به ما می دهد.

بخش ۲

ابتدا به سراغ LS می‌رویم و از کدی که توسط حل تمرین در اختیارمان قرار داده شد استفاده می‌کنیم.

کتابخانه‌های زیر را import کردیم و چون داده‌های دارای missing value را حذف کردیم مشکلی از این بابت نداریم:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

df.isnull().sum()

Summary          0
Precip_Type      0
Temperature_(C)  0
Apparent_Temperature_(C) 0
Humidity         0
Wind_Speed_(km/h) 0
Wind_Bearing_(degrees) 0
Visibility_(km)   0
Pressure_(millibars) 0
Daily_Summary    0
dtype: int64
```

دقت شود که روش‌های زیادی برای پرکردن missing value ها وجود دارد ولی چون این داده‌ها درصد کمی از کل داده‌ها بودند، با حذف‌شان مشکلی پیش نمی‌آید.

ورودی‌های ماشین، دما و دمای ظاهری هستند و خروجی یا همان متغیر وابسته رطوبت است. (صورت سوال درباره مشخص کردن این رابطه کمی نامفهوم بود پس من با این فرض جلورفتم)

```
[34] x = df[['Apparent_Temperature_(C)', 'Temperature_(C)']]
y = df['Humidity']

[35] x.shape
(95936, 2)

[36] y.shape
(95936,)

[37] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 54)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
((76748, 2), (76748,), (19188, 2), (19188,))
```

از standard train_test_split استفاده می‌کنیم و ۲۰ درصد از داده‌ها را به تست اختصاص می‌دهیم. سپس از scaler استفاده می‌کنیم و در نهایت مدل را فیت می‌کنیم.

```

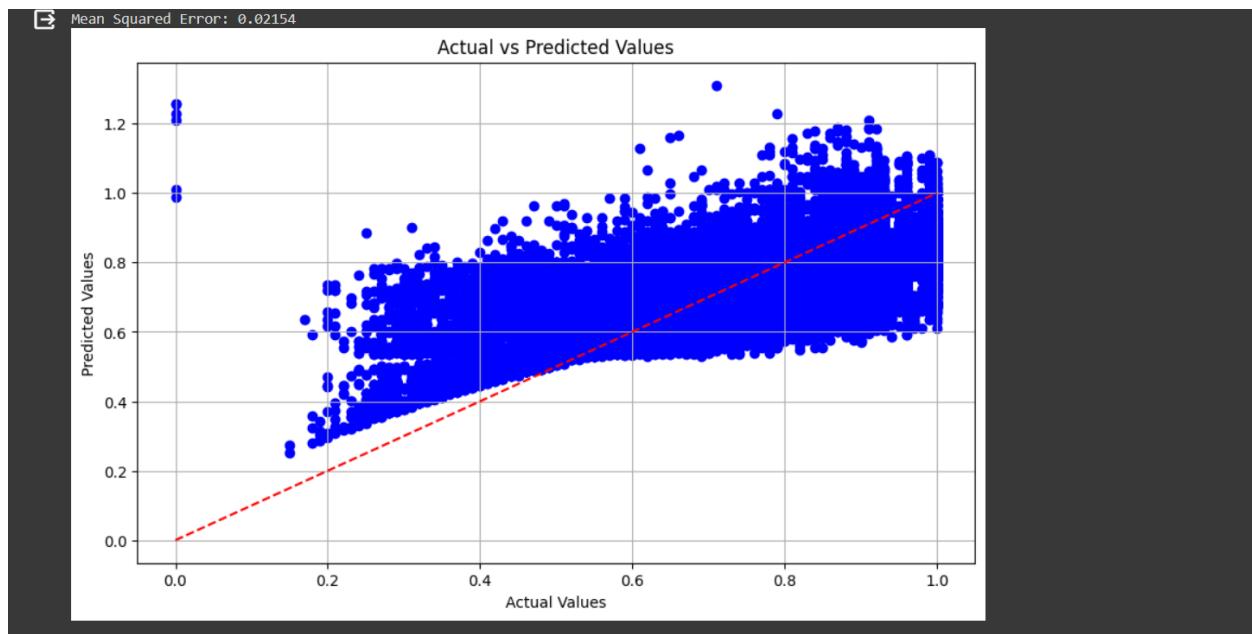
0s  scaler = StandardScaler()
      x_train_scaled = scaler.fit_transform(x_train)
      x_test_scaled = scaler.transform(x_test)

0s  [41] regLS_model = LinearRegressionLS()
      regLS_model.fit(x_train_scaled,y_train)
      y_pred = regLS_model.predict(x_test_scaled)

1s  [44] mse = mean_squared_error(y_test, y_pred)
      print("Mean Squared Error: ", mse.round(5))
      plt.figure(figsize=(10, 6))
      plt.scatter(y_test, y_pred, color='blue')
      plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--')
      plt.xlabel('Actual Values')
      plt.ylabel('Predicted Values')
      plt.title('Actual vs Predicted Values')
      plt.grid(True)
      plt.show()

```

و خروجی به صورت زیر است: MSE



همانطور که دیده می شود، یک خط به داده ها فیت و عمل linear regression انجام شده است. این خط به نوعی فیت شده که مربع اختلاف پیش بینی و مشاهده، برای هر نمونه کمترین مقدار باشد.

در این حالت $MSE = 0.02154$ گزارش می شود.

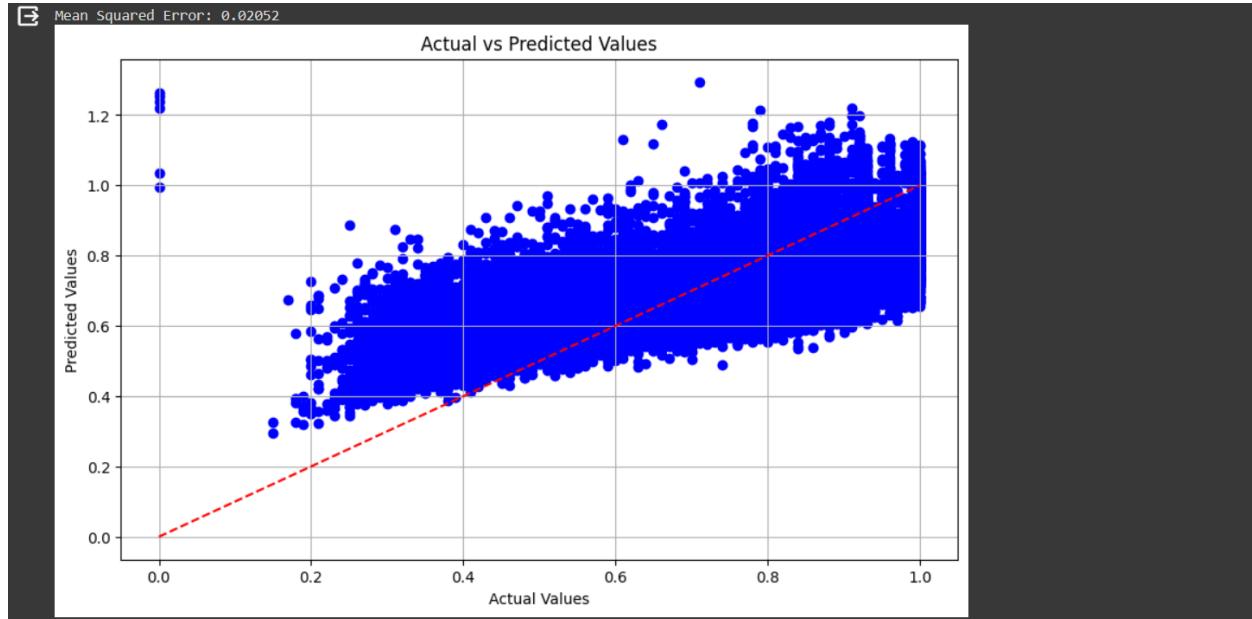
قبل از اینکه سراغ بخش بعدی برویم، یک تغییر کوچک در مدل اعمال می کنیم. اگر ماتریس همبستگی را با یاد بیاوریم برای ویژگی Humidity علاوه بر دما و دمای ظاهری، ویژگی های میدان دید و سرعت باد هم همبستگی خوبی با متغیر وابسته ما داشتند. پس این دو ویژگی هم وارد فرآیند آموزش و ارزیابی می کنیم تا نتایج را بررسی کنیم:

```
[61] x = df[['Apparent_Temperature_(C)', 'Temperature_(C)', 'Wind_Speed_(km/h)', 'visibility_(km)']]
y = df["Humidity"]

[62] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 54)
x_train.shape, y_train.shape, x_test.shape, y_test.shape

((76748, 4), (76748,), (19188, 4), (19188,))
```

نتایج به صورت زیر خواهد بود:



می‌بینیم که MSE در این حالت بهتر شده و در کل می‌توان گفت مدل عملکرد بهبود یافته تری نسبت به حالت قبل دارد.

پس به این صورت با کمک ماتریس correlation می‌توانیم به عملکرد بهتری دست پیدا کنیم و ویژگی هایی که در فرایند پیش بینی به ما کمک می‌کنند را شناسایی کنیم.

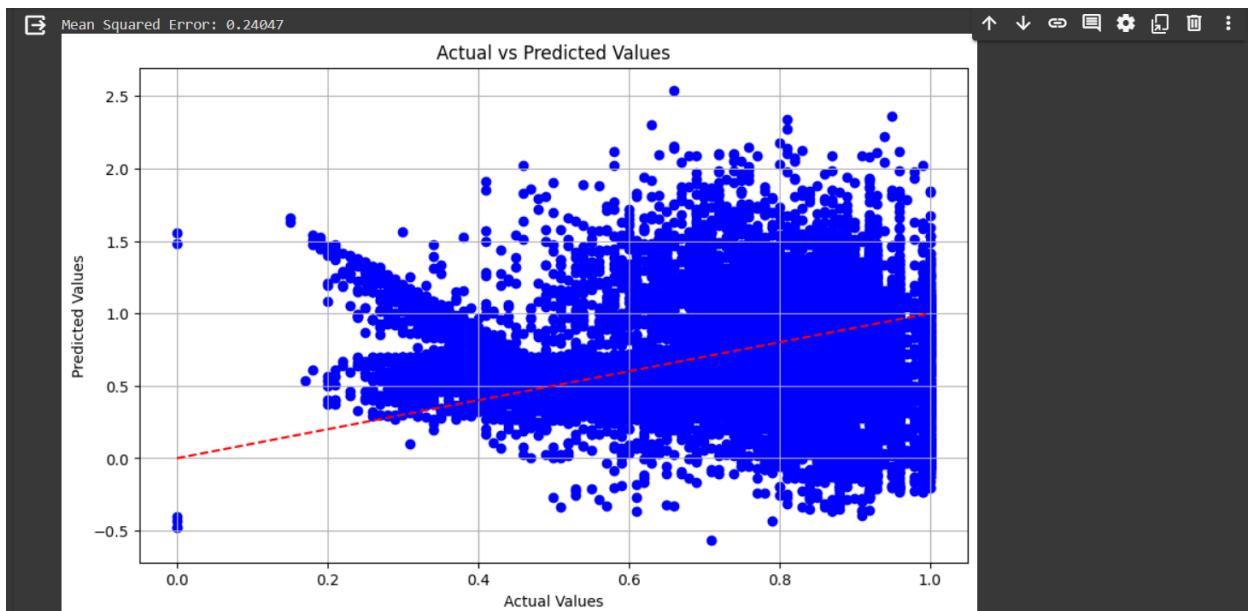
بعد از این به سراغ RLS می‌رویم که معمولاً برای دیتاستی استفاده می‌شود که به مرور به روز می‌شود و یک الگوریتم adaptive با یک حلقه for است که به اندازه طول دیتاست ایتریت می‌کند.

از الگوریتمی که توسط حل تمرین در اختیارمان قرار داده شد استفاده می‌کنیم و مقادیر فیچرها را (np.array) به الگوریتم می‌دهیم:

دقت شود که ما در این الگوریتم یک فاکتور فراموشی داریم که هرچه بزرگتر باشد، نرخ فراموشی سریع‌تر است و الگوریتم حافظه کمتری دارد.

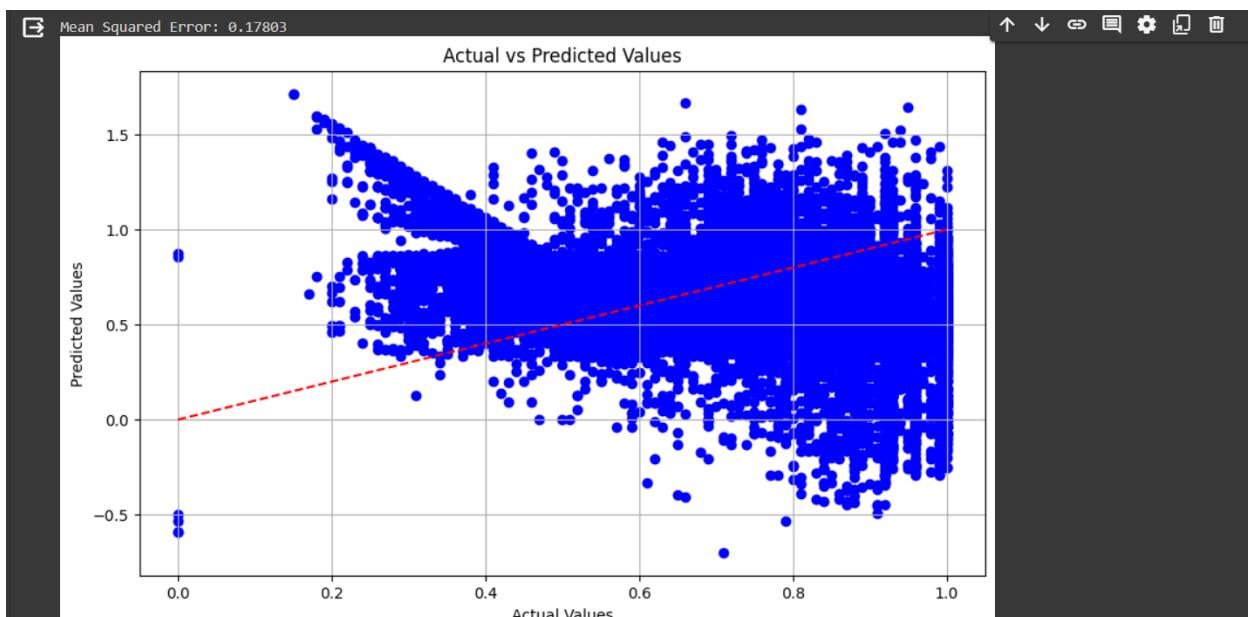
```
[55]: x = df[['Apparent_Temperature_(C)', 'Temperature_(C)']].values
y = df["Humidity"].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 54)
```

اگر این فاکتور را روی ۰,۷ قرار دهیم:



می‌بینیم که $MSE = 0.24047$ خواهد بود.

حالا اگر این فاکتور را روی ۰,۹۹ بگذاریم:

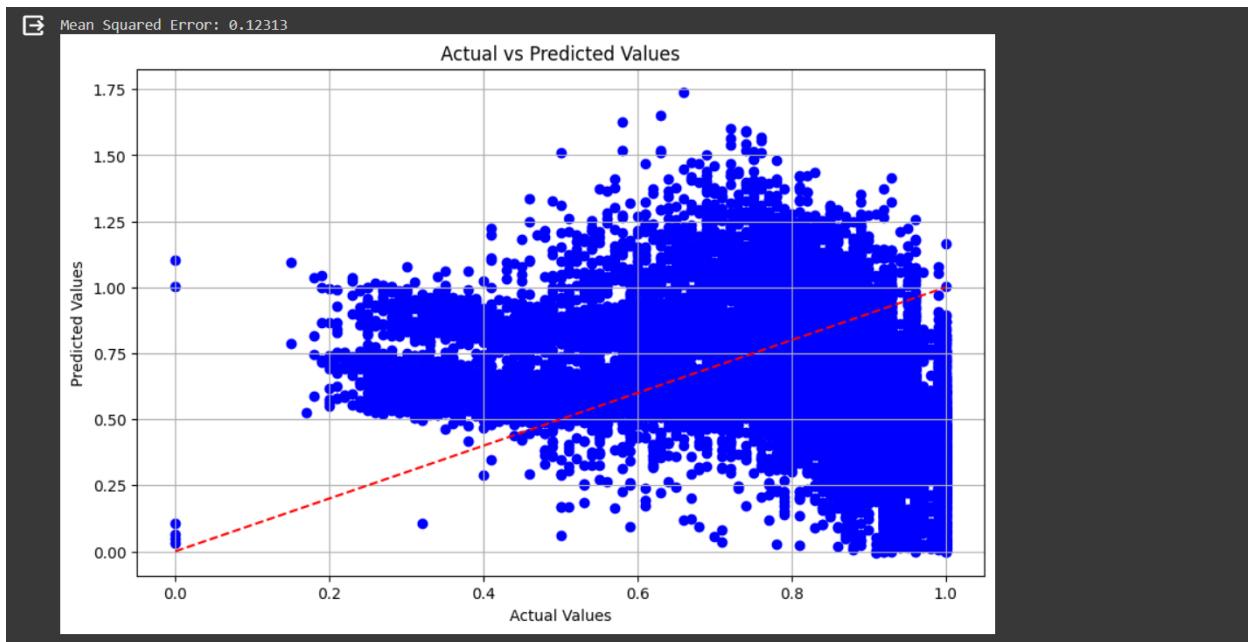


که $MSE = 0.17803$ خواهد بود.

می‌بینیم که وقتی ضریب فراموشی بیشتر باشد در این حالت، به MSE کمتری می‌رسیم.

در این حالت نیز مانند حالت LS از ۴ ویژگی معرفی شده استفاده می‌کنیم تا تعریف آن را ببینیم:

```
[68] x = df[['Apparent_Temperature_(C)', 'Temperature_(C)', 'Wind_Speed_(km/h)', 'Visibility_(km)']].values
y = df["Humidity"].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 54)
```



می‌بینیم که به کمک ویژگی‌های بیشتر، پیش‌بینی مدل بهتر شده است و معیار عملکرد بهتری دارد.

در کل این یک روش است که از روش قبلی پیچیده‌تر است و برای دیتاست‌های بزرگ جواب بهتری می‌گیرد. در اینجا ما از روش LS خطای کمتری گرفتیم چون یک روش ساده و پایداری است و کار regression ما هم مطابق چنین کاربردی است.

الگوریتم‌های مختلف نسبت به هم برتری مطلق ندارند و ممکن است الگوریتم‌های مختلف روی دیتاست‌های مختلف جواب‌های متفاوت بگیرند و باید دید با توجه به کاربرد خاص، کدام الگوریتم بهتر است.

بخش ۳

WLS در واقع نسخه ای پیشرفته تر از LS است که به هر داده (نقطه) یک وزن اختصاص می دهد. این الگوریتم وزن دهی برای زمانی استفاده می شود که برخی از داده ها به اندازه کافی قابل اتکا (یا بالهمیت) نیستند یا به هر دلیلی نیاز است تا برخی داده ها در عمل regression مهم تر واقع شوند. مثلا ممکن است برخی مشاهدات دارای ضریب اطمینان بالاتری نسبت به سایر مشاهدات داشته باشند. در این صورت این مشاهدات باید اهمیت بیشتری در تخمین داشته باشند.

در این الگوریتم نیز هدف کم کردن مربع باقی مانده است منتهی وزن هر باقی مانده، معکوس واریانس آن مشاهده است. وقتی واریانس نویز بیشتر باشد، وزن آن داده کمتر خواهد شد. در عمل، این واریانس ها معمولاً ناشناخته هستند و باید به نحوی تخمین زده شوند.

$$\sum_{t=1}^n w_t \varepsilon_t^2 = \sum_{t=1}^n w_t (y_t - y_t^M)^2 \Rightarrow \min$$

معادله نرمال مربوط به این الگوریتم:

$$X^T W X b = X^T W y$$

که در آن، W همان ماتریس قطری وزن ها و b ، ماتریس ضرایب تخمین مربوطه است.

```

0s [59] X = df[['Apparent_Temperature_(C)', 'Temperature_(C)']]
      y = df["Humidity"]

0s [60] error_variance = 2 # Modify this value based on your estimation
      # calculate weights based on the estimated variance
      weights = 1 / error_variance

      # Fit the weighted least squares model
      X_with_intercept = sm.add_constant(X) # Add intercept term
      model = sm.WLS(y, X_with_intercept, weights=weights)
      result = model.fit()

      # Print the model summary
      print(result.summary())

```

نتیجه زیر مربوط به پیاده سازی این الگوریتم بر دیتاست آب و هوای مربوط به سوال است:

WLS Regression Results						
Dep. Variable:	Humidity	R-squared:	0.443			
Model:	WLS	Adj. R-squared:	0.443			
Method:	Least Squares	F-statistic:	3.808e+04			
Date:	Thu, 04 Apr 2024	Prob (F-statistic):	0.00			
Time:	17:29:53	Log-Likelihood:	48382.			
No. Observations:	95936	AIC:	-9.676e+04			
Df Residuals:	95933	BIC:	-9.673e+04			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	p> t	[0.025	0.975]
const	0.9646	0.001	834.178	0.000	0.962	0.967
Apparent_Temperature_(C)	0.0312	0.000	85.977	0.000	0.031	0.032
Temperature_(C)	-0.0477	0.000	-117.108	0.000	-0.048	-0.047
Omnibus:	4400.820	Durbin-Watson:	0.151			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	5495.510			
Skew:	-0.487	Prob(JB):	0.00			
Kurtosis:	3.654	Cond. No.	56.7			

متغیر وابسته ما همان رطوبت است که قصد تخمین آن را داریم.

R_squared بیانگر فیت شدن مدل است که نشان می‌دهد ۴۴.۳٪ واریانس در متغیر وابسته، توسط متغیر های مستقل توصیف می‌شود.

تخمین ضرایب درواقع ضرایب ماشین هستند که رابطه بین ورودی ها و خروجی را توصیف می‌کند. برای مثال ضریب دما 0.0312 تخمین زده شده است و یک بایاس هم داریم که 0.9646 تعريف شده است. همینطور t-values و p-values هم تعريف شده اند که متريک هایی برای اهمیت ضرایب هستند در اين حالت $p < 0.05$ و همگی مهم و قابل اتكا هستند.

بقيه متريک ها نيز اورده شده (مانند AIC یا F-statistics) که برای ارزیابی مدل استفاده می‌شوند.

(۴) بخش

این روش از زیرمجموعه های RLS است که به صورت recursive ضرایب را به روز می‌کند و از تجزیه QR برای تخمین استفاده می‌کند. QR یک روش تجزیه ماتریس است که ماتریس را به صورت حاصل ضرب یک ماتریس قطری Q و ماتریس بالامثلی R تجزیه می‌کند.

مراجع

[1] <https://deci.ai/course/datasplitting/#:~:text=Data%20shuffling%20is%20an%20integral,which%20can%20lead%20to%20overfitting>.

[2] <https://medium.com/@spinjosovsky/normalize-data-before-or-after-split-of-training-and-testing-data-7b8005f81e26>