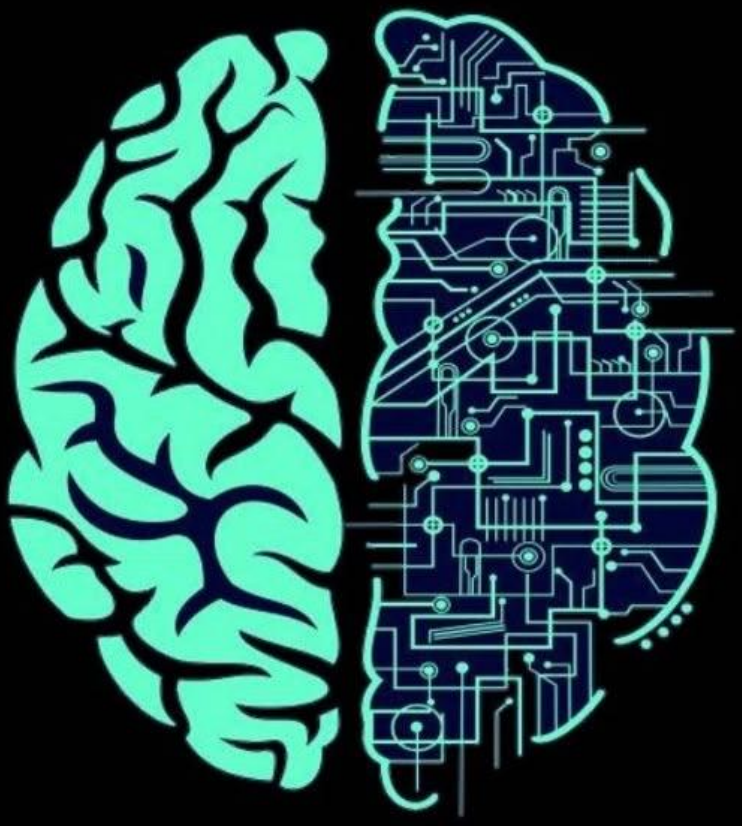
Artificial Intelligence

Machine Learning

E03-Classification



Name: Masoud Lotfizadeh Sepehri

Student ID: 810603134

Email: mlotfizadeh@ut.ac.ir

Course Instructor: Professor Shariatpanahi

فهرست مطالب

الف-۱ ..... ۳

الف-۲ ..... ۴

الف-۳ ..... ۵

الف-۴ ..... ۶

ب- ۱ ..... ۸

ب-۲ ..... ۸

ج-۱ ..... ۹

ج-۲ ..... ۹

ج-۳ ..... ۱۰

ج-۴ ..... ۱۰

ج-۵ ..... ۱۱

ج-۶ ..... ۱۵

ج-۷ ..... ۱۸

ج-۸ ..... ۲۰

د-۱ ..... ۲۱

د-۲ ..... ۲۲

د-۳ ..... ۲۴

د-۴ ..... ۲۵

بخش الف

الف-۱ بررسی اولیه داده‌ها

در ابتدا دیتاست مون رو میخونیم. مجموعه از ده هزار سطر و شش ستون تشکیل شده است

[18]:

	Air Temp (°C)	Process Temp (°C)	Rotational Speed (RPM)	Torque (Nm)	Tool Wear (Seconds)	Failure Types
0	29.021640	71.620737	1515.840689	50.223021	664.638000	No Failure
1	21.886075	69.896471	2083.417786	52.221351	6628.080758	No Failure
2	29.020744	74.731134	2455.801496	57.822145	3295.576818	No Failure
3	25.793868	70.715109	2112.654324	69.910072	7116.479752	No Failure
4	21.056760	71.025092	1642.485295	68.411333	1191.996403	No Failure
...	...	...	...	...	...	...
9995	26.348331	91.671449	440.302554	17.023823	29737.865306	Tool Wear Failure
9996	25.907144	87.530054	846.804899	27.393605	21222.615198	Tool Wear Failure
9997	20.622875	81.410821	361.552759	21.354210	22885.462945	Tool Wear Failure
9998	25.262251	96.950974	427.901464	32.502106	19724.207233	Tool Wear Failure
9999	23.223389	86.260566	548.189837	34.266373	30506.579732	Tool Wear Failure

10000 rows × 6 columns

سپس ویژگی ها رو با دستور مربوطه بدست آوردیم که شامل تعداد سطر و ستون، هر ستون چند تا داده نال داره و میسینک ولیو هامون چطوریه . همچنین نوع داده ها (ستون Failure types از نوع object و بقیه float هستن ...) مشخص شدند.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Air Temp (°C)          9965 non-null   float64
 1   Process Temp (°C)      9990 non-null   float64
 2   Rotational Speed (RPM) 10000 non-null  float64
 3   Torque (Nm)            10000 non-null  float64
 4   Tool Wear (Seconds)    9993 non-null   float64
 5   Failure Types          9991 non-null   object
dtypes: float64(5), object(1)
memory usage: 468.9+ KB
```

از دستور بعدی استفاده شد تا یه سری اطلاعات آماری بدست بیاد که به عنوان مثال هر ویژگی که داریم چند تا نمونه توش هست، میانگینش، انحراف معیارش،حداقل،حداکثر و چارک هاش مشخص شدند.

[9]:

	Air Temp (°C)	Process Temp (°C)	Rotational Speed (RPM)	Torque (Nm)	Tool Wear (Seconds)
count	9965.000000	9990.000000	10000.000000	10000.000000	9993.000000
mean	28.516926	80.812186	1401.909988	46.998845	11393.143344
std	7.719340	15.548350	968.446183	26.747646	9023.336380
min	20.001366	60.001876	0.047731	0.015920	3.469877
25%	23.176455	68.090324	423.672240	18.091381	5023.027818
50%	26.212082	76.553203	1377.047835	54.983239	8995.172952
75%	29.377536	92.825894	2307.969925	67.258375	15024.825673
max	49.998008	119.971025	2999.953724	89.993221	35999.566519

الف-۲ بررسی مقادیر گم شده (Missing Values)

هدفمون شمارش تعداد و درصد مقادیر گمشده برای هر ستون هستش با کد مربوطه، جدولی می‌گیریم که ستون‌های دارای مقادیر گمشده رو به ما نشون می‌ده.

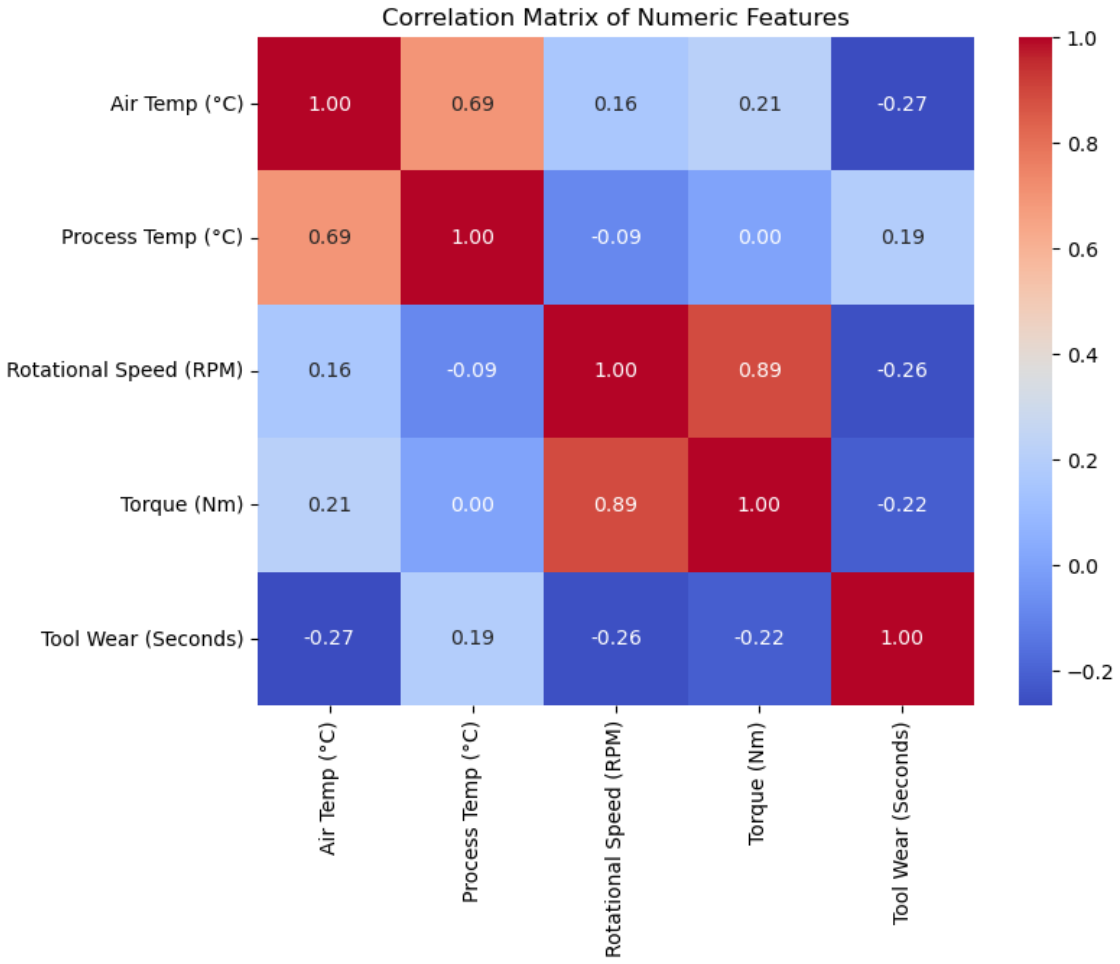
]:

	Missing Count	Missing Percent
Air Temp (°C)	35	0.35
Process Temp (°C)	10	0.10
Rotational Speed (RPM)	0	0.00
Torque (Nm)	0	0.00
Tool Wear (Seconds)	7	0.07
Failure Types	9	0.09

همونطور که مشاهده میکنیم مقادیر گم شده کم و قابل مدیریت هستند.

الف-۳ بررسی همبستکی ویژگی ها (correlation)

برای بررسی همبستگی بین ویژگی‌های عددی، ابتدا فقط ستون‌های عددی داده‌ها انتخاب شده و سپس ماتریس همبستگی آن‌ها محاسبه گردید. سپس با استفاده از کتابخانهSeaborn ، یک هیت مپ از این ماتریس ترسیم شد.

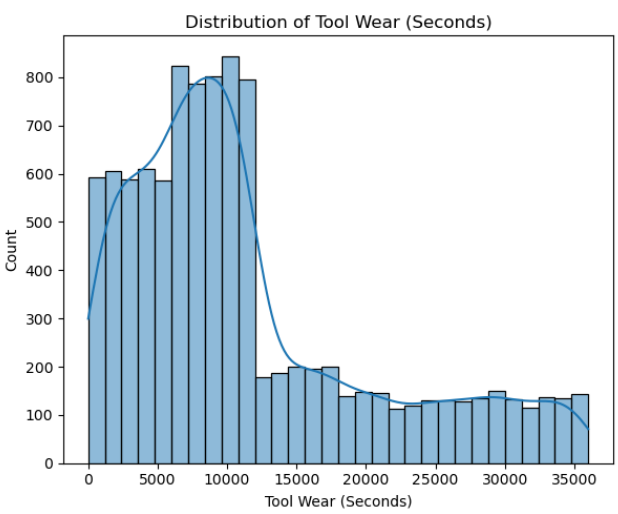


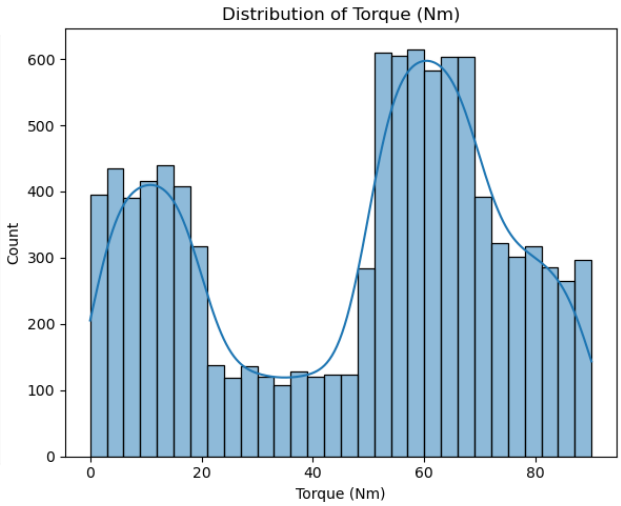
به طور کلی، اکثر ویژگی‌ها دارای همبستگی کم تا متوسط با یکدیگر بودند Process Temp و Air Temp بیشترین همبستگی مثبت را با هم داشتند (ضریب حدود 0.8)؛ سایر ویژگی‌ها مانند Tool Wear، Torque و Rotational Speed همبستگی‌های ضعیف‌تری با یکدیگر دارند، که نشان‌دهنده اطلاعات مستقل آن‌ها در پیش‌بینی خروجی است.

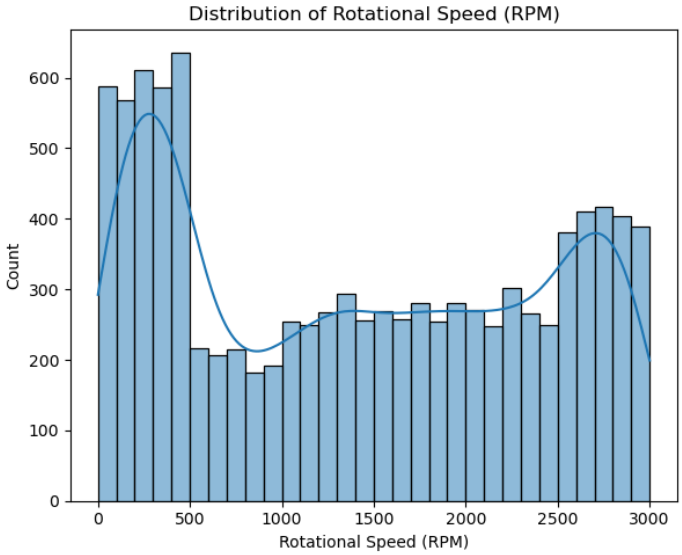
الف-۴ رسم نمودار توزیع مشاهدات برای سه ویژگی با بیشترین تأثیر

در این مرحله هدفمون بررسی نحوه توزیع داده‌ها برای ویژگی‌های عددی تأثیرگذار در دیتاستمون هستش. این کار به ما کمک می‌کنه تا شناخت بهتری نسبت به پراکندگی داده‌ها، نقاط پرت(outliers) ، و ارتباط احتمالی بین متغیرها و Failure Types داشته باشیم.

بر اساس بررسی همبستگی (در مرحله‌ی قبل)، سه ویژگی Tool Wear ، Torque و Rotational Speed به عنوان مهم‌ترین ویژگی‌های عددی انتخاب شدند؛ در ابتدا، توزیع هر یک از ویژگی‌های فوق به صورت جداگانه و بدون توجه به نوع خطا ترسیم شد:

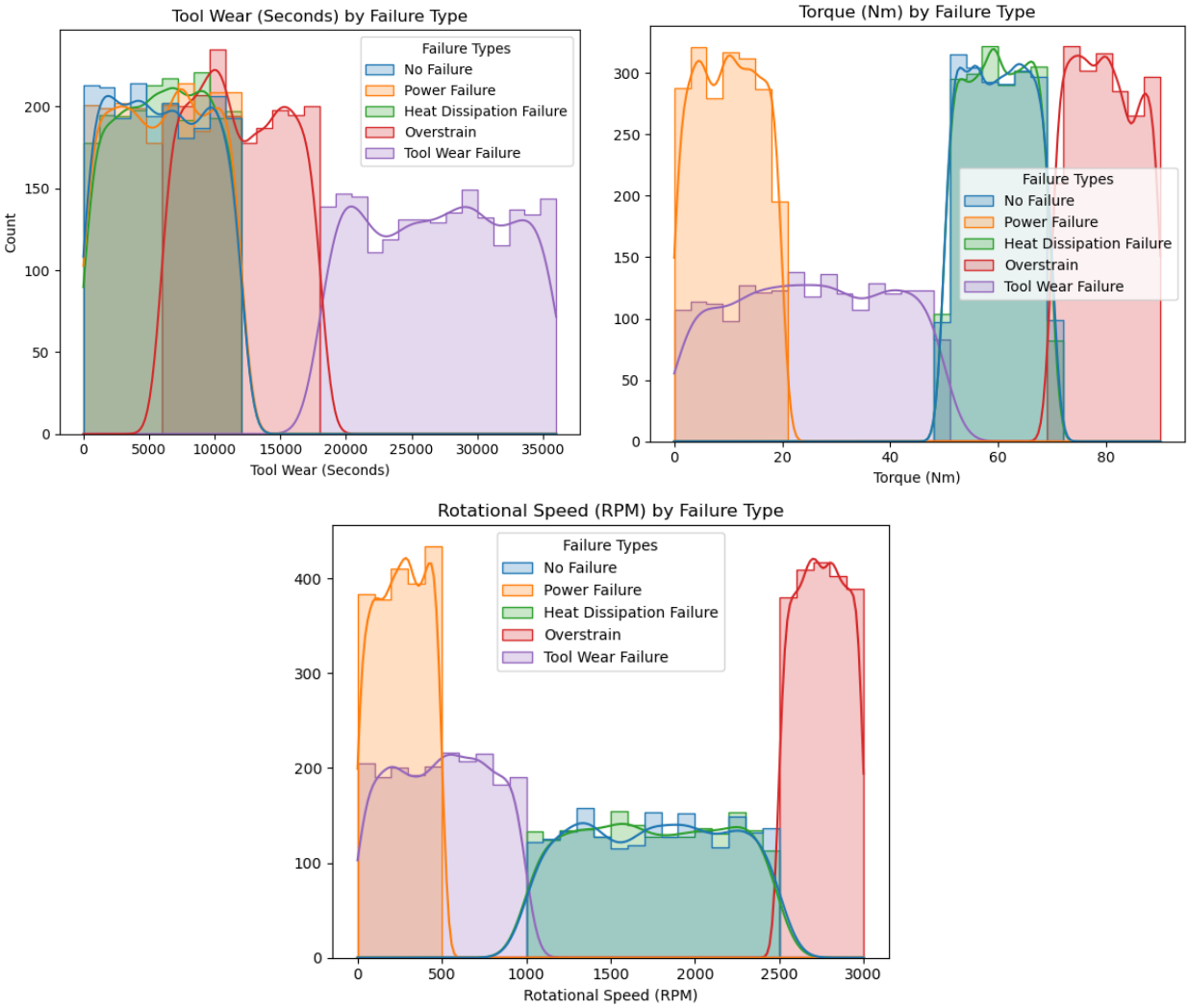






- توی Tool Wear بیشتر داده‌ها بین ۵ تا ۱۵ هزار ثانیه هستن، ولی یه سری مقدار خیلی بالا هم داشتیم که نشون میده بعضی ابزارها خیلی بیشتر از بقیه استفاده شده اند.
- در Torque ، داده‌ها توی بازه‌ی ۲۰ تا ۷۰ بیشتر جمع شده اند.
- Rotational Speed هم دو تا قله داره؛ انگار در دو بازه‌ی سرعت خاص، دستگاه بیشتر کار کرده.

در گام بعدی، همین نمودارها با در نظر گرفتن متغیر Failure Types ترسیم شدند:



این نمودارها بهمون نشون دادن که بین مقدار بعضی ویژگی‌ها و نوع خرابی رابطه هستش. یعنی بعضی ویژگی‌ها واقعاً قابلیت این رو دارن که به مدل کمک کنن نوع خرابی رو تشخیص بده.

بخش ب

ب- ۱ بررسی و حل مشکل مقادیر ناموجود (Missing Values)

در قسمت الف، بررسی شد که آیا داخل دیتاست مقدار گمشده وجود داره یا نه. با استفاده از دستور مربوطه و محاسبه درصد، مشخص شد که بعضی ستون‌ها مقدارهای ناقصی دارن.

چون تعداد این مقدارهای گمشده کم بود ، به جای حذف کل سطرها، فقط مقدارها اصلاح شد. برای ستون‌های عددی یعنی Air Temp، Process Temp وTool Wear ، از میانگین همان ستون‌ها استفاده شد تا مقدارهای خالی باهاشون جایگزین بشه. این روش برای داده‌های عددی مؤثره.

برای ستون Failure Types که از نوع متنیه (object هستش) و مقدار هدف محسوب می‌شه، به‌جای پر کردن، اون چند ردیف ناقص رو حذف کردم چون تعدادشون کم بود و تأثیر زیادی روی حجم کل داده‌ها نداشت.

در نهایت، دوباره بررسی کردم و مطمئن شدم که دیگه هیچ مقدار گمشده‌ای توی دیتاست وجود نداشته باشه.

```
Air Temp (°C)            0
Process Temp (°C)        0
Rotational Speed (RPM)   0
Torque (Nm)              0
Tool Wear (Seconds)      0
Failure Types            0
dtype: int64
```

ب-۲ اجرای فرآیندهای استانداردسازی (Standardization) و نرمال‌سازی(Normalization)

بعد از پاک‌سازی داده‌ها، رسیدیم به مرحله‌ی پیش‌پردازش عددی؛ یعنی استانداردسازی (Standardization) و نرمال‌سازی (Normalization).

قبل از اعمال هر نوع تبدیل، دیتای ورودی و خروجی رو از هم جدا کردم، و با استفاده از train_test_split، داده‌ها رو به دو بخش آموزش و تست تقسیم کردم (۸۰ درصد آموزش و ۲۰ درصد تست). این کار باعث می‌شه که اطلاعات داده‌های تست توی فرآیند آموزش دخالت نکنن.

برای اینکه ویژگی‌ها مقیاس یکسانی داشته باشن، اول از StandardScaler استفاده کردم. این روش داده‌ها رو طوری تبدیل می‌کنه که میانگینش صفر و انحراف معیارش یک باشه. این مرحله بیشتر برای مدل‌هایی مثل SVM و Logistic Regression ضروریه.

بعد از استانداردسازی، همون ویژگی‌های عددی رو با MinMaxScaler هم نرمال‌سازی کردم. این کار داده‌ها رو بین بازه ۰ تا ۱ می‌بره. برای بعضی الگوریتم‌ها مثل KNN یا Neural Network که به مقیاس عددها حساسن، این مرحله خیلی تأثیر داره.

بنابراین تمام ویژگی‌های عددی توی یک مقیاس مشخص قرار گرفتن و آماده‌ی آموزش مدل هستن. این مرحله کمک می‌کنه که مدل‌ها عملکرد دقیق‌تری داشته باشن، مخصوصاً اون‌هایی که به فاصله بین داده‌ها وابسته‌ان.

بخش ج

ج-۱ ایجاد ستون برچسب دوگانه (Failure / No Failure)

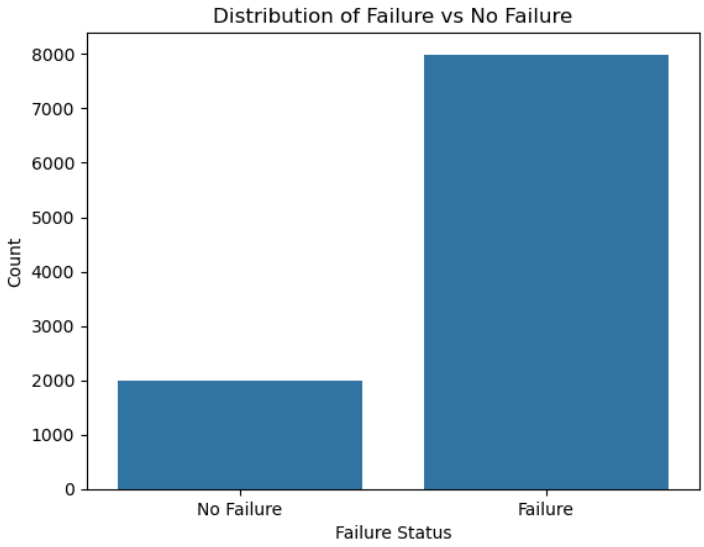
در این مرحله، اومدم یه ستون جدید درست کردم به اسم Failure_Binary تا نوع خرابی رو ساده‌تر کنم و فقط به دو حالت تقسیم بشه (یا Failure (برای هر نوع خرابی) و یا No Failure که یعنی سالم بوده.

هدف از این کار این بود که به‌جای دسته‌بندی چندکلاسه، یه دسته‌بندی دوتایی داشته باشیم که برای بعضی مدل‌ها و تحلیل‌ها ساده‌تره، مخصوصاً وقتی فقط بخوایم تشخیص بدیم یه قطعه خراب شده یا نه، بدون اینکه نوع خرابی برامون مهم باشه.

ج-۲ توزیع برچسب‌ها (Failure vs No Failure)

بعد از اینکه توی بخش قبل ستون جدید Failure_Binary رو ساختم، حالا اومدم بررسی کنم که داده‌ها بیشتر مربوط به خرابی هستن یا سالم بودن دستگاه. برای این کار، از countplot استفاده کردم تا تعداد

داده‌های هر کلاس رو به صورت نمودار ستونی نشون بدم. این نمودار نشون می‌ده که چندتا از نمونه‌ها مربوط به Failure هستن و چندتا No Failure.



مشاهده میکنیم که تعداد داده‌های کلاس Failure خیلی بیشتر از No Failure هستش؛ یعنی مجموعه داده نامتوازنه و این می‌تونه باعث بشه بعضی مدل‌ها تمایل داشته باشن بیشتر کلاس غالب که همون Failure هست رو حدس بزنن. به همین خاطر، توی مرحله بعد با SMOTE سعی می‌کنیم این عدم تعادل رو برطرف کنیم.

ج-۳ توضیح مشکل عدم توازن داده‌ها

وقتی داده‌های دسته‌بندی نامتوازن هستن (مثلاً یک کلاس خیلی بیشتر از دیگری است)، مدل ممکنه یاد بگیره همیشه کلاس اکثریت رو پیش‌بینی کنه چون این کار به‌ظاهر باعث افزایش دقت می‌شه. اما در واقع مدل هیچ‌چیزی یاد نگرفته و این دقت بالا گمراه‌کننده هستش.

ج-۴ متوازن سازی داده‌ها با SMOTE

بعد از اینکه توی بخش قبل فهمیدیم داده‌ها نامتوازن هستن یعنی تعداد نمونه‌های Failure خیلی بیشتر از No Failure هستش، برای رفع این مشکل از روش SMOTE استفاده میکنیم. SMOTE با ساختن داده‌های مصنوعی از کلاس اقلیت (که در اینجا No Failure هستش)، کمک می‌کنه که تعادل بین کلاس‌ها برقرار بشه.

اومدیم اول داده‌ها رو به ویژگی‌ها و برچسب تبدیل کردیم، سپس داده‌ها رو به آموزش و تست تقسیم کردیم، ولی با stratify تا نسبت کلاس‌ها توی آموزش حفظ بشه و در نهایت روی داده‌های آموزشی، SMOTE رو اعمال کردیم.

```
قبل SMOTE: {1: 6394, 0: 1598}
بعد SMOTE: {1: 6394, 0: 6394}
```

مشاهده میکنیم که قبل از اعمال، داده های Failure، ۶۳۹۴ تا بودن و داده های No Failure، ۱۵۹۸ تا بود؛ اما پس از اعمال SMOTE هر دو کلاس ۶۳۹۴ تا شدن. به این ترتیب، داده‌ها متعادل شدن و حالا مدل می‌تونه با دقت و انصاف بیشتری هر دو کلاس رو یاد بگیره.

ج-۵ آموزش سه مدل مختلف و ارزیابی آن‌ها روی داده‌های تست

بعد از متعادل‌سازی داده‌ها باSMOTE ، چهار تا مدل مختلف رو روی داده‌های آموزش‌شده تست کردیم تا ببینیم کدوم بهتر عمل می‌کنه. مدل‌ها عبارت بودند از:

1. Logistic Regression

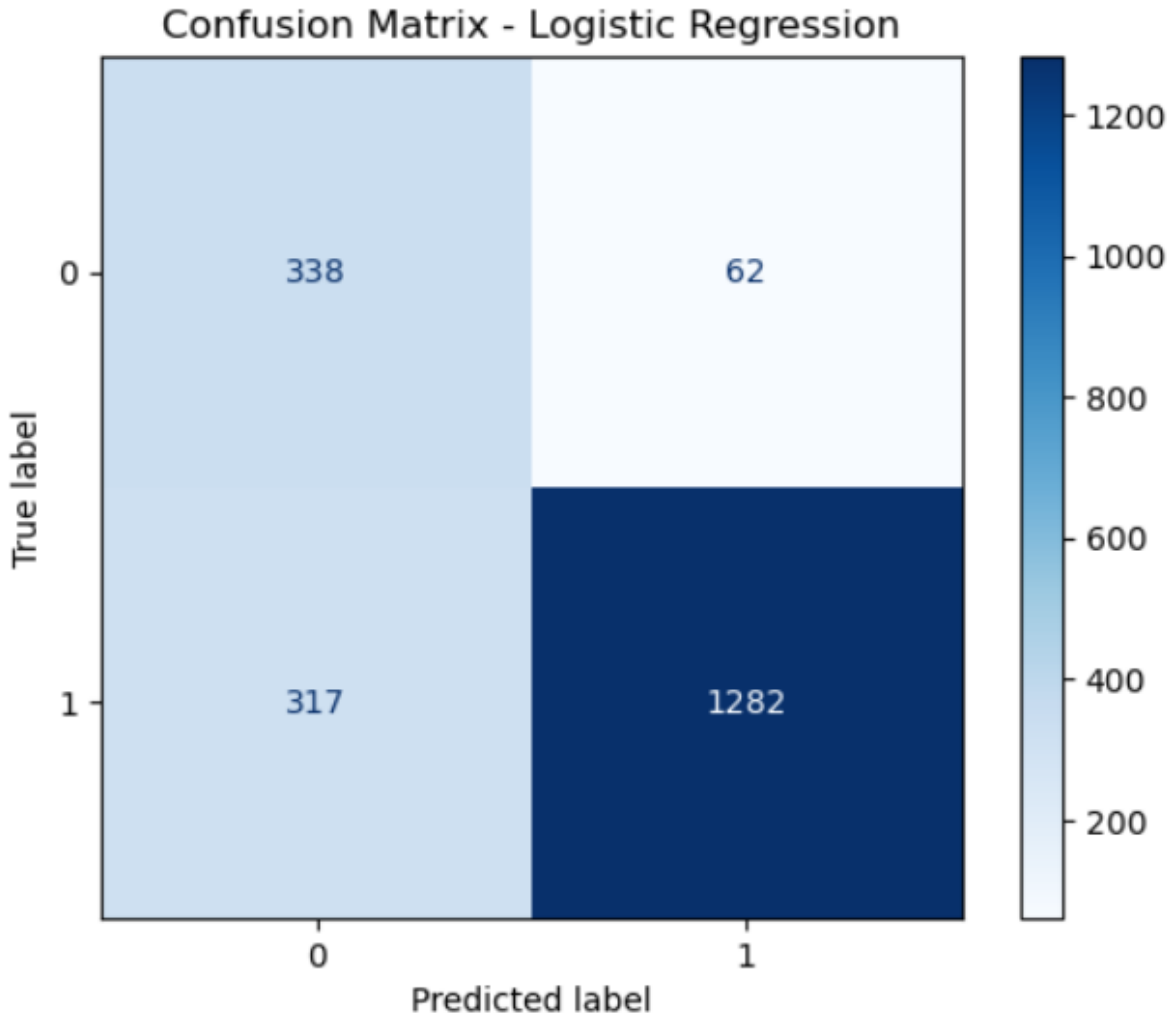
2. K-Nearest Neighbors (KNN)

3. LinearSVC

4. SVC با کرنل RBF (kernel) (غیرخطی)

برای هر مدل، هم آموزش انجام شد و هم پیش‌بینی روی داده‌های تست، و بعدش با استفاده از متریک‌هایی مثل دقت، precision، recall، F1-score و ماتریس آشفتگی عملکردشون ارزیابی شد.

- Logistic Regression

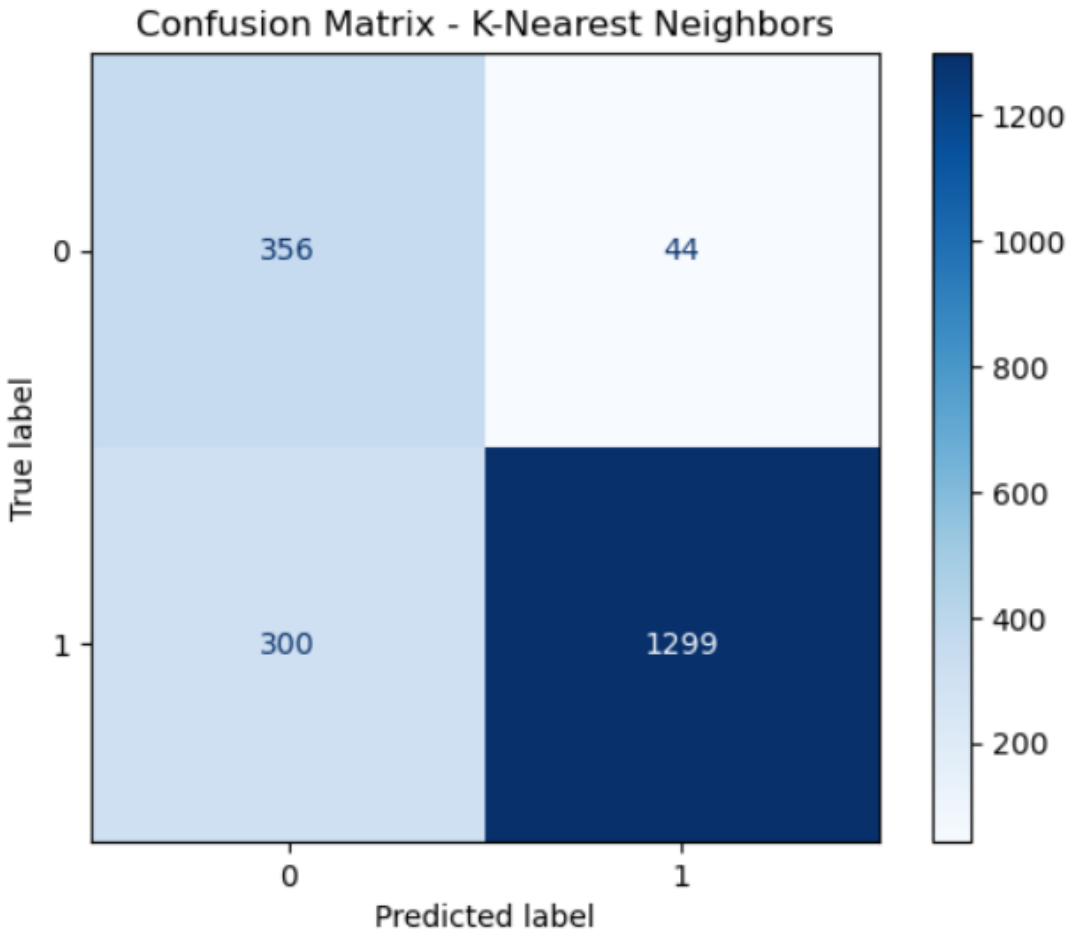


Classification Report:

	precision	recall	f1-score	support
0	0.52	0.84	0.64	400
1	0.95	0.80	0.87	1599
accuracy			0.81	1999
macro avg	0.73	0.82	0.76	1999
weighted avg	0.87	0.81	0.83	1999

Logistic Regression.1 : دقت حدود ۸۱ درصد، توی کلاس Failure خوب بود ولی No Failure رو کمتر درست تشخیص داد.

- K-Nearest Neighbors

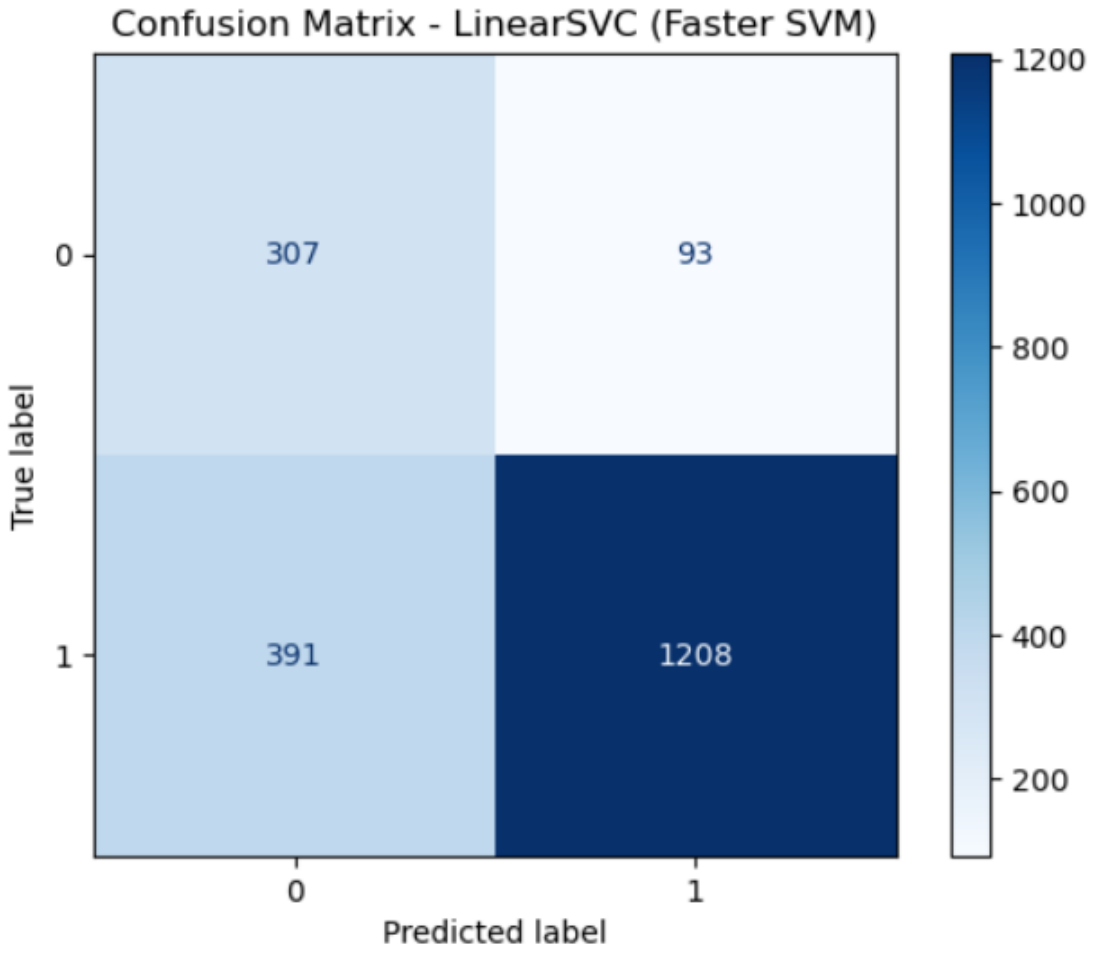


Classification Report:

	precision	recall	f1-score	support
0	0.54	0.89	0.67	400
1	0.97	0.81	0.88	1599
accuracy			0.83	1999
macro avg	0.75	0.85	0.78	1999
weighted avg	0.88	0.83	0.84	1999

KNN .2: دقت حدود ۸۳ درصد ، عملکرد متعادل‌تر بین دو کلاس، مخصوصاً توی بالا بردن recall برای کلاس No Failure.

LinearSVC (Faster SVM)



Classification Report:

	precision	recall	f1-score	support
0	0.44	0.77	0.56	400
1	0.93	0.76	0.83	1599
accuracy			0.76	1999
macro avg	0.68	0.76	0.70	1999
weighted avg	0.83	0.76	0.78	1999

3. LinearSVC: دقت کمتر، حدود ۷۶ درصد، توی تشخیص کلاس Failure خوب بود ولی کلاس No Failure رو کمتر شناسایی کرد.

SVM با هسته RBF:

	precision	recall	f1-score	support
0	0.41	1.00	0.58	400
1	1.00	0.64	0.78	1599
accuracy			0.71	1999
macro avg	0.71	0.82	0.68	1999
weighted avg	0.88	0.71	0.74	1999

Confusion Matrix:

```
[[ 400    0]
 [ 575 1024]]
```

4. SVC با کرنل RBF (kernel) (غیرخطی): این مدل غیرخطی بود و برخلاف انتظار، بیش‌برازش (overfitting)داشت. توی آموزش عالی عمل کرده بود، ولی روی تست دقتش فقط حدود ٪۷۱ بود. یعنی کلاس No Failure رو کاملاً درست تشخیص داد، ولی بخش زیادی از نمونه‌های Failure رو اشتباه گرفت.

در بین مدل‌های خطی و KNN، KNN عملکرد بهتری داشت. ولی SVC غیرخطی با کرنل RBF با اینکه از نظر تئوری قدرت بالایی داره، توی این داده‌ها خوب تعمیم نداد. احتمالاً به خاطر این بوده که بیش از حد روی داده‌های آموزشی فیت شده بود (Overfitted).

ج-۶ مقایسه عملکرد مدل‌ها با استفاده از معیارهای ارزیابی

در این بخش، مدل‌هایی که توی مرحله قبل آموزش داده با هم مقایسه شدن تا ببینیم کدوم بهتر عمل کرده. برای مقایسه از معیارهای Accuracy، Precision، Recall، F1-score و Confusion Matrix استفاده کردیم:

Model	Recall (avg)	Precision (avg)	Accuracy	F1 score (avg)
Logistic Regression	0.81	0.87	0.81	0.83
K-Nearest Neighbors	0.83	0.88	0.83	0.84
Linear SVC	0.76	0.83	0.76	0.78
RBF SVC	0.71	0.88	1.00 (train) / 0.71 (test)	0.74

همچنین برای هر مدل، ماتریس آشفتگی کشیده شد تا مشخص بشه کدوم کلاس‌ها بیشتر اشتباه پیش‌بینی شدن.

KNN متعادل‌ترین عملکرد رو داشت و تونست هر دو کلاس رو با دقت نسبتاً خوبی تشخیص بده.

Logistic Regression هم خوب بود ولی نسبت به KNN کمی ضعیف‌تر عمل کرد.

LinearSVC توی تشخیص کلاس No Failure ضعیف‌تر بود.

SVC با کرنل RBF دقت خیلی خوبی روی داده‌های آموزش داشت (تقریباً ۱۰۰٪) ولی روی داده‌های تست افت کرد، که نشون می‌ده مدل دچار بیش‌برازش شده.

در مجموع، از بین این چهار مدل، KNN انتخاب بهتری برای این مسئله به نظر می‌رسه، مخصوصاً وقتی که تعادل بین دقت در هر دو کلاس برامون مهم باشه.

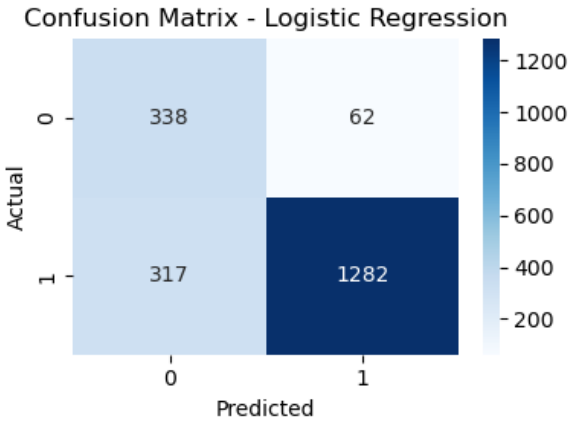
```
                 Model  Accuracy
0  Logistic Regression  0.810405
1  K-Nearest Neighbors  0.827914
2           Linear SVC  0.757879
3             RBF SVC  1.000000
```

- Logistic Regression

```
Accuracy: 0.8104052026013007
Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.84      0.64       400
           1       0.95      0.80      0.87      1599

    accuracy                           0.81      1999
   macro avg       0.73      0.82      0.76      1999
weighted avg       0.87      0.81      0.83      1999
```

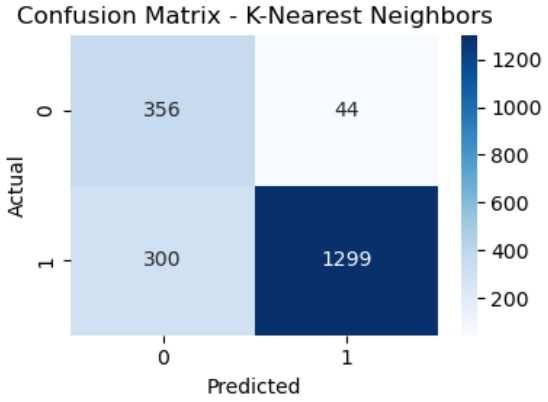


- K-Nearest Neighbors

```
Accuracy: 0.8279139569784892
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.89      0.67       400
           1       0.97      0.81      0.88      1599

    accuracy                           0.83      1999
   macro avg       0.75      0.85      0.78      1999
weighted avg       0.88      0.83      0.84      1999
```

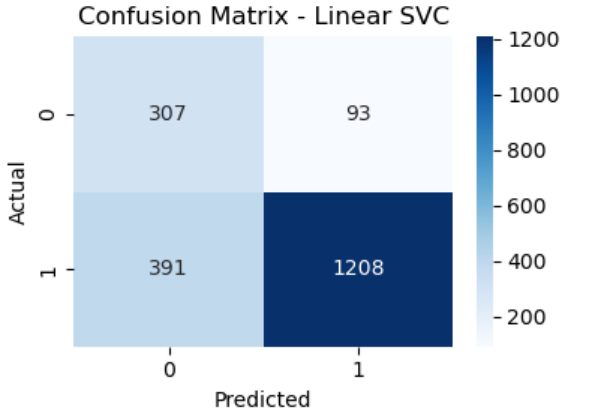


- Linear SVC

```
Accuracy: 0.7578789394697348
Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.77      0.56       400
           1       0.93      0.76      0.83      1599

    accuracy                           0.76      1999
   macro avg       0.68      0.76      0.70      1999
weighted avg       0.83      0.76      0.78      1999
```

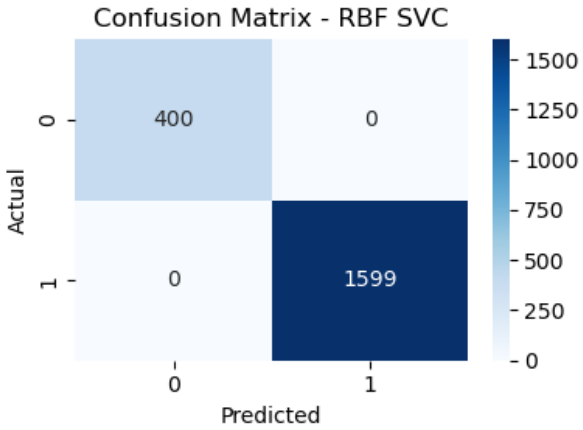


- RBF SVC

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   suppor

           0       1.00      1.00      1.00       400
           1       1.00      1.00      1.00      1599

    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999
```



ج-۷ تنظیم هایپرپارامترها با GridSearchCV

توی این بخش، اومدیم برای مدل‌هایی که قبلاً استفاده کرده بودیم، به‌جای اینکه با تنظیمات پیش‌فرض آموزش بدیم، از GridSearchCV استفاده کردیم تا بهترین ترکیب‌های ممکن برای هایپرپارامترها رو پیدا کنیم.

برای هر مدل، یه مجموعه از مقادیر مختلف برای هایپرپارامترها تعریف کردیم و با استفاده از crossvalidation بررسی کردیم که کدوم ترکیب بهترین عملکرد رو داره.

1. برای Logistic Regression پارامترهایی که تست کردیم:
   - C: مقدار تنظیم‌کننده (از ۰.۰۱ تا ۱۰)
   - solver: الگوریتم بهینه‌سازی liblinear و lbfgs

نتیجه:

- بهترین C برابر ۰.۱
- بهترین solver: lbfgs
- دقت (cross-val) : حدود 83.4٪

```
Best parameters (Logistic Regression): {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
Best accuracy (train, cross-val): 0.8341417250412257
```

2. برای K-Nearest Neighbors پارامترهایی که تست کردیم:
   - تعداد همسایه‌ها (n_neighbors) از ۳ تا ۱۵
   - وزن‌دهی (weights): uniform یا distance

نتیجه:

- بهترین حالت n_neighbors=3 :و weights='distance'
- دقت (cross-val) : حدود 91.5٪

```
Best parameters (KNN): {'n_neighbors': 3, 'weights': 'distance'}
Best accuracy (train, cross-val): 0.9147637156643997
```

3. برای LinearSVC پارامترهایی که تست کردیم:
   - C از ۰.۰۱ تا ۱۰
   - max_iter تا ۵۰۰۰

نتیجه:

- بهترین C عدد ۱۰
- دقت : حدود ٪۷۸.۸

```
Best parameters (LinearSVC): {'C': 10, 'max_iter': 1000}
Best accuracy (train, cross-val): 0.788472980241273
```

4. برای SVC با کرنلRBF پارامترهایی که تست کردیم:
   - C از ۰.۱ تا ۱۰
   - gamma: scale، ۰.۰۱، ۰.۱، ۱

نتیجه:

- بهترین حالت C=10 و 'gamma='scale
- دقت (cross-val) : حدود 86.8٪

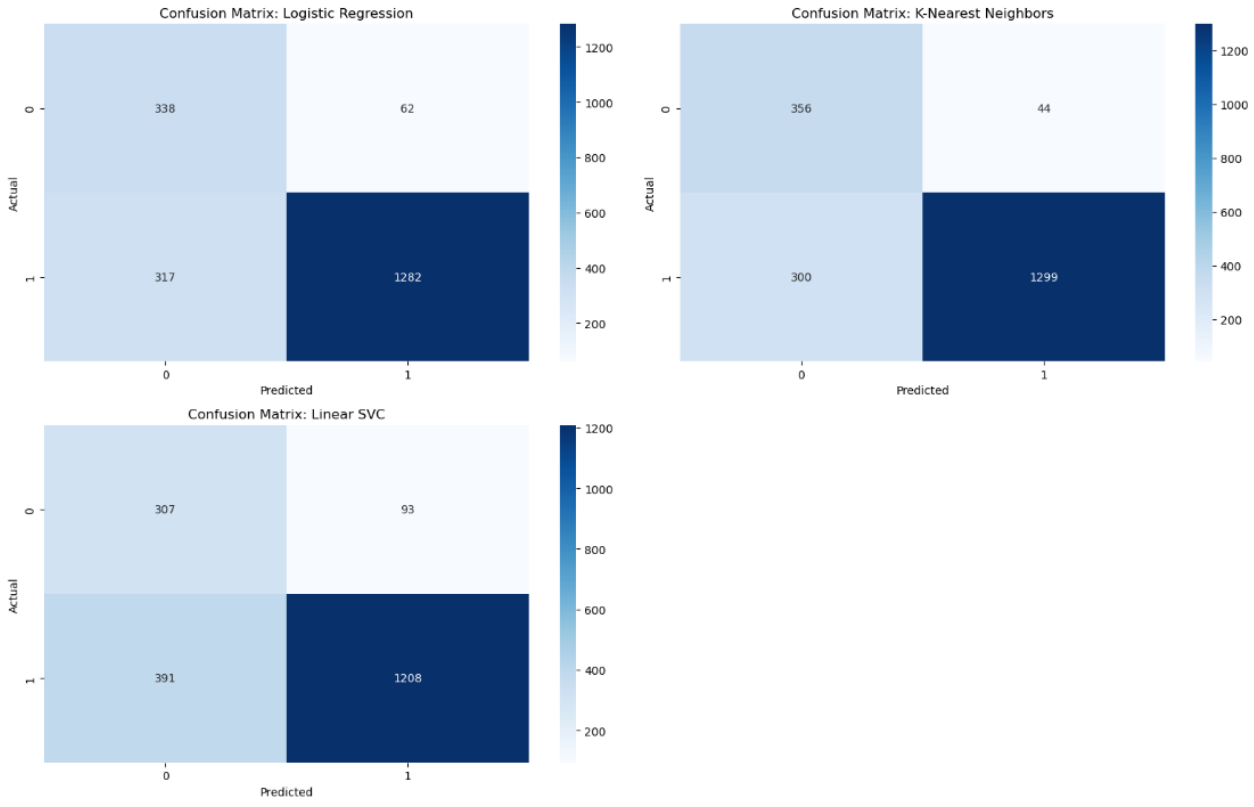
```
Best parameters (RBF SVC): {'C': 10, 'gamma': 'scale'}
Best accuracy (train, cross-val): 0.8681572576835332
```

در نهایت، با استفاده از بهترین مدل‌هایی که از GridSearch به دست اومدن، دوباره روی داده‌های تست ارزیابی انجام دادم تا ببینم چقدر بهبود حاصل شده.

```
Logistic Regression Test Accuracy: 0.8104
KNN Test Accuracy: 0.8599
Linear SVC Test Accuracy: 0.7579
RBF SVC Test Accuracy: 0.8009
```

ج-۸ بررسی و مقایسه عملکرد مدل‌های بهینه‌شده

توی این قسمت، پس از اینکه توی قسمت قبل اومدیم با استفاده از GridSearchCV بهترین پارامترها برای مدل‌ها رو پیدا کردیم، حالا اومدیم همون مدل‌های بهینه‌شده رو روی داده‌های تست اجرا کردیم تا ببینیم در عمل چقدر خوب کار می‌کنن. (LinearSVC و RBF SVC هم قبلاً بررسی شده بودن و چون توی دقت تست ضعیف‌تر بودن، تمرکز اصلی روی این سه مدل گذاشته شد)



جدول مقایسه عملکرد مدل‌ها:

	Model	Accuracy	Precision (avg)	Recall (avg)	F1-score (avg)
0	Logistic Regression	0.810405	0.866258	0.810405	0.825104
1	K-Nearest Neighbors	0.827914	0.882284	0.827914	0.841286
2	Linear SVC	0.757879	0.830730	0.757879	0.778295

KNN بهترین عملکرد رو داشت، هم از نظر دقت کلی، هم میانگین F1-score. Logistic Regressionساده‌تره ولی کمی ضعیف‌تر عمل کرد. Linear SVC پایین‌ترین نتایج رو داشت و نتونست به خوبی کلاس‌ها رو تفکیک کنه. در نتیجه در کل، KNN گزینه‌ی مناسب‌تریه توی این دیتاست.

بخش د

د-۱ تقسیم داده‌های پالایش‌شده و آماده‌سازی برای مدل‌سازی

توی این مرحله، اومدیم از داده‌هایی که قبلاً پاک‌سازی شده بودن (یعنی مقادیر missing values حذف یا اصلاح شده بودن)، برای آموزش مدل‌های جدید استفاده کردیم. هدفمون این بود که یه دسته‌بندی چندکلاسه انجام بدیم، یعنی مدل بتونه نوع خرابی رو تشخیص بده، نه فقط اینکه خرابی وجود داره یا نه.

در مرخله اول اومدیم و ویژگی‌های ورودی (Features) و خروجی یا برچسب (Target) رو انتخاب کردیم. سپس چون برای دسته‌بندی چندکلاسه نیاز به داده‌ی کامل داشتیم، ردیف‌هایی که مقدار Failure Types نداشتن رو حذف کردیم.

با استفاده از تابع train_test_split داده‌ها رو با نسبت ۸۰ به ۲۰ بین train و test تقسیم کردیم و پارامتر stratify هم گذاشتیم تا نسبت کلاس‌ها توی هر دو بخش تقریباً یکسان باشه.

در نتیجه حالا یه دیتاست کامل و آماده داریم که شامل تمام کلاس‌های مختلف خرابی هست و می‌تونیم باهاش مدل‌های چندکلاسه مون که مربوط به قسمت بعدی گزارش میشه رو تربیت کنیم.

د-۲ تربیت مدل‌های مختلف روی داده‌های چندکلاسه

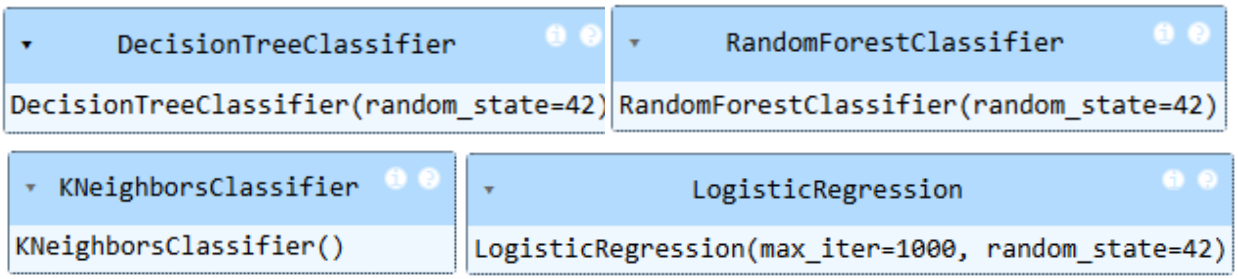
توی این بخش، بعد از آماده‌سازی داده‌ها توی مرحله‌ی قبل، چند مدل مختلف رو روی داده‌های چندکلاسه آموزش دادیم تا ببینم کدوم یکی بهتر می‌تونه نوع خرابی رو تشخیص بده؛ هر چهار مدل رو روی داده‌های آموزش آموزش فیت کردیم سپس با داده‌های تست عملکردشون رو بررسی کردیم. مدل هامون:

K-Nearest Neighbors (KNN)

Decision Tree

Random Forest

Logistic Regression



اول مقدارهای گمشده رو با میانگین پر کردیم (فقط برای اطمینان). سپس داده‌ها رو با StandardScaler نرمال‌سازی کردیم تا همه‌ی ویژگی‌ها در یه مقیاس قرار بگیرن.

```
Model: KNN
Confusion Matrix:
[[400   0   0   0   0   0]
 [  0 399   0   0   0   0]
 [  0   0 399   0   0   0]
 [  0   0   0 400   0   0]
 [  0   0   0   0 400   0]
 [  0   0   2   0   0   0]]
Classification Report:
              precision    recall  f1-score   support
0              1.000000  1.000000  1.000000   400.000
1              1.000000  1.000000  1.000000   399.000
2              0.995012  1.000000  0.997500   399.000
3              1.000000  1.000000  1.000000   400.000
4              1.000000  1.000000  1.000000   400.000
5              0.000000  0.000000  0.000000     2.000
accuracy       0.999000  0.999000  0.999000     0.999
macro avg      0.832502  0.833333  0.832917  2000.000
weighted avg   0.998005  0.999000  0.998501  2000.000
```

```
   Model  Accuracy
0    KNN    0.9990
1  Decision Tree    0.9975
2  Random Forest    0.9990
3  Logistic Regression    0.9985
```

```
Model: Random Forest
Confusion Matrix:
[[400   0   0   0   0   0]
 [  0 399   0   0   0   0]
 [  0   0 399   0   0   0]
 [  0   0   0 400   0   0]
 [  0   0   0   0 400   0]
 [  0   0   2   0   0   0]]
Classification Report:
              precision    recall  f1-score   support
0              1.000000  1.000000  1.000000   400.000
1              1.000000  1.000000  1.000000   399.000
2              0.995012  1.000000  0.997500   399.000
3              1.000000  1.000000  1.000000   400.000
4              1.000000  1.000000  1.000000   400.000
5              0.000000  0.000000  0.000000     2.000
accuracy       0.999000  0.999000  0.999000     0.999
macro avg      0.832502  0.833333  0.832917  2000.000
weighted avg   0.998005  0.999000  0.998501  2000.000
```

```
Model: Logistic Regression
Confusion Matrix:
[[400   0   0   0   0   0]
 [  0 398   1   0   0   0]
 [  0   0 399   0   0   0]
 [  0   0   0 400   0   0]
 [  0   0   0   0 400   0]
 [  0   0   2   0   0   0]]
Classification Report:
              precision    recall  f1-score   support
0              1.000000  1.000000  1.000000   400.0000
1              1.000000  0.997494  0.998745   399.0000
2              0.992537  1.000000  0.996255   399.0000
3              1.000000  1.000000  1.000000   400.0000
4              1.000000  1.000000  1.000000   400.0000
5              0.000000  0.000000  0.000000     2.0000
accuracy       0.998500  0.998500  0.998500     0.9985
macro avg      0.832090  0.832916  0.832500  2000.0000
weighted avg   0.997511  0.998500  0.998002  2000.0000
```

مشخصه که دقت همه‌ی مدل‌ها خیلی بالاست. این می‌تونه دلایل مختلفی داشته باشه. توی مدل‌هایی مثل Random Forest و KNN که روی داده‌های تمیز اجرا بشن، احتمال داره بیش‌برازش (overfitting) رخ داده باشه. یعنی مدل به خوبی داده‌های آموزش رو یاد گرفته ولی ممکنه توی داده‌های واقعی دنیای بیرون دقت پایین‌تری بده.

مثلا در مدل KNN اکثر کلاس‌ها مثل 0 تا 4 کاملاً درست پیش‌بینی شدن. ولی یه کلاس نادر (مثلاً کلاس ۵) که فقط ۲ نمونه داشت، یا اصلاً درست تشخیص داده نشد، یا فقط یکی از دو مورد درست بود. با این حال چون

تأثیر این کلاس خیلی کمه، دقت کلی مدل هنوز نزدیک به ۱ دیده می‌شه. شایدم داده ها واقعی نیستن ! با این حال میریم قسمت بعدی تمرین ...

د-۳ بهینه‌سازی هایپرپارامترهای مدل‌ها باGridSearchCV

پس از آموزش اولیه‌ی مدل‌ها، حالا توی این بخش میریم سراغ تنظیم دقیق‌ترشون با استفاده از **GridSearchCV و** این کار کمک می‌کنه تا بهترین ترکیب از پارامترهای قابل تنظیم برای هر مدل پیدا بشه و بتونیم عملکرد مدل‌ها رو بهتر کنیم. (هرچند که نتایج قسمت قبل منطقی نیست زیاد ...)

1. برای Logistic Regression پارامترهایی که تست کردیم:
   - برای c مقدارهای 0.01، 0.1، 1 و 10
   - solver: ‘liblinear’ و‘lbfgs’

نتیجه:

- بهترین C عدد 10
- بهترین solver: lbfgs
- دقتCross-Validation : حدود 99.89٪

```
Logistic Regression best params: {'C': 10, 'solver': 'lbfgs'}
Logistic Regression best accuracy: 0.998875
```

2. برای KNN پارامترهایی که تست کردیم:
   - n_neighbors : از ۱ تا ۲۰

نتیجه:

- بهترین مقدارn_neighbors=3

- دقت: حدود 99.88٪

```
KNN best params: {'n_neighbors': 3}
KNN best accuracy: 0.99875
```

3. برای Random Forestپارامترهایی که تست کردیم:

   - n_estimators: 50، 100، 200
   - max_depth: None، 10، 20، 30

نتیجه:

- بهترین ترکیب: n_estimators=50، max_depth=None
- دقت: حدود 99.90٪

```
Random Forest best params: {'max_depth': None, 'n_estimators': 50}
Random Forest best accuracy: 0.999
```

با اینکه دقت همه‌ی مدل‌ها خیلی بالا بود و نتایج قابل اعتمادی نیست ولی با GridSearch نشون دادیم حتی با دقت اولیه‌ی بالا هم می‌شه بهینه‌سازی انجام داد و مقادیرکمی بهتر شد. مخصوصاً توی مدل‌های KNN و Random Forest، انتخاب درست k روی دقت تأثیر می‌ذاره.

د-۴ ارزیابی مدل‌های بهینه‌شده روی داده‌های تست

بعد از اینکه در قسمت قبل مدل‌ها رو با GridSearchCV بهینه کردیم، حالا اومدیم همون مدل‌های تنظیم‌شده رو روی داده‌های تست اجرا کردیم تا ببینم توی عمل چقدر خوب کار می‌کنن. مدل‌هایی که تست شدن:

- Logistic Regression (بهینه‌شده با C=10 و solver=lbfgs)
- K-Nearest Neighbors (با k=3)
- Random Forest (با 50 درخت و عمق نامحدود)

```
=== Logistic Regression ===
Confusion Matrix:
[[400   0   0   0   0   0]
 [  0 399   0   0   0   0]
 [  0   0 399   0   0   0]
 [  0   0   0 400   0   0]
 [  0   0   0   0 400   0]
 [  0   0   2   0   0   0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       400
           1       1.00      1.00      1.00       399
           2       1.00      1.00      1.00       399
           3       1.00      1.00      1.00       400
           4       1.00      1.00      1.00       400
           5       0.00      0.00      0.00         2

    accuracy                           1.00      2000
   macro avg       0.83      0.83      0.83      2000
weighted avg       1.00      1.00      1.00      2000

=== Random Forest ===
Confusion Matrix:
[[400   0   0   0   0   0]
 [  0 399   0   0   0   0]
 [  0   0 399   0   0   0]
 [  0   0   0 400   0   0]
 [  0   0   0   0 400   0]
 [  0   0   2   0   0   0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       400
           1       1.00      1.00      1.00       399
           2       1.00      1.00      1.00       399
           3       1.00      1.00      1.00       400
           4       1.00      1.00      1.00       400
           5       0.00      0.00      0.00         2

    accuracy                           1.00      2000
   macro avg       0.83      0.83      0.83      2000
weighted avg       1.00      1.00      1.00      2000
```

```
=== KNN ===
Confusion Matrix:
[[400   0   0   0   0   0]
 [  0 399   0   0   0   0]
 [  0   0 399   0   0   0]
 [  0   0   0 400   0   0]
 [  0   0   0   0 400   0]
 [  0   0   2   0   0   0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       400
           1       1.00      1.00      1.00       399
           2       1.00      1.00      1.00       399
           3       1.00      1.00      1.00       400
           4       1.00      1.00      1.00       400
           5       0.00      0.00      0.00         2

    accuracy                           1.00      2000
   macro avg       0.83      0.83      0.83      2000
weighted avg       1.00      1.00      1.00      2000
```

همه‌ی مدل‌ها عملکرد بالایی داشتن. Random Forest کمی بهتر از بقیه بوده، مخصوصاً توی شناسایی کلاس‌هایی که نمونه‌های کمتری داشتن.

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	0.999	0.998005	0.999	0.998501
1	KNN	0.999	0.998005	0.999	0.998501
2	Random Forest	0.999	0.998005	0.999	0.998501

The End