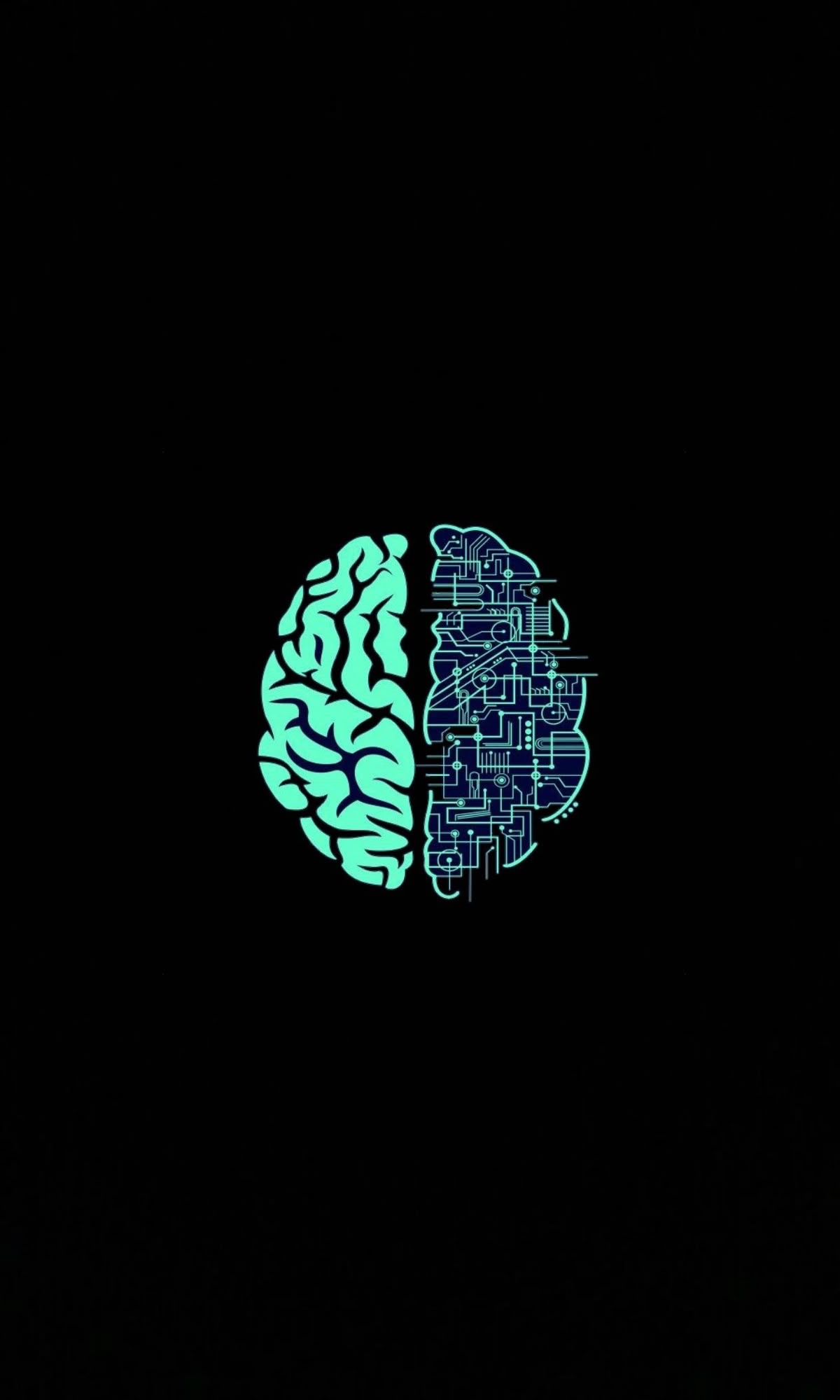
**Artificial Intelligence**

**Machine Learning**

**E03-Classification**

Name: Masoud Lotfizadeh Sepehri

Student ID: 810603134

Email: [mlotfizadeh@ut.ac.ir](mailto:mlotfizadeh@ut.ac.ir)

Course Instructor**:** Professor Shariatpanahi

**فهرست مطالب**

الف-۱ .................................................................................................................................................................................... ٣

الف-۲ ..................................................................................................................................................................................... ۴

الف-۳ .................................................................................................................................................................................... ۵

الف-۴ .......................................................................................................................................................................................... ۶

ب- ۱ ..................................................................................................................................................................................... ٨

ب-۲ ........................................................................................................................................................................................ ٨

ج-۱ ........................................................................................................................................................................................ ٩

ج-۲ ........................................................................................................................................................................................ ٩

ج-۳ ......................................................................................................................................................................................... ١٠

ج-۴.............................................................................................................................. .......................................................... ١٠

ج-۵ ...................................................................................................................................................................................... ١١

ج-۶ ....................................................................................................................................................... ................................ ١۵

ج-۷................................................................................................................................................................... .................... ١۸

ج-۸ ...................................................................................................................................................................................... ۲٠

د-۱ ....................................................................................................................................................................................... ۲١

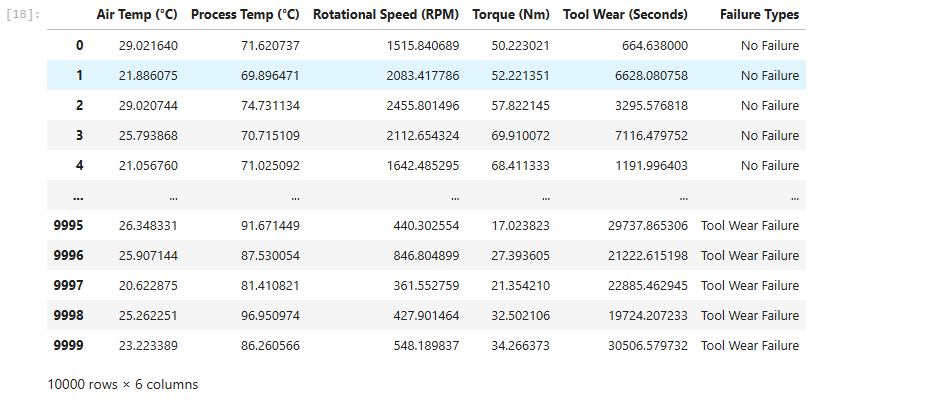
د-۲ ....................................................................................................................................................................................... ۲۲

د-۳ ....................................................................................................................................................................................... ۲۴

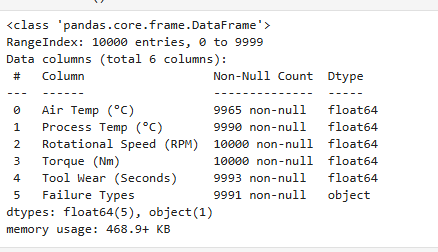
د-۴......................................................................................................................................................................................... ۲۵

بخش الف

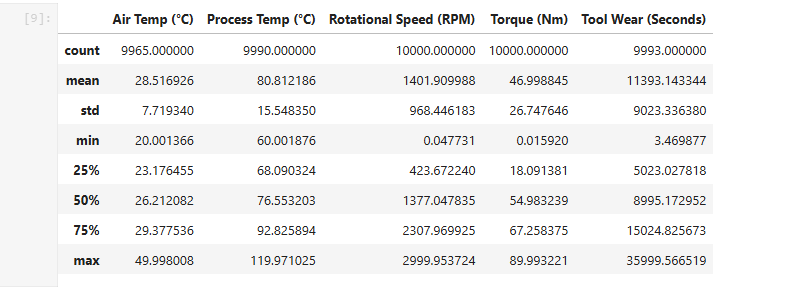
**الف-۱ بررسی اولیه داده‌ها**

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

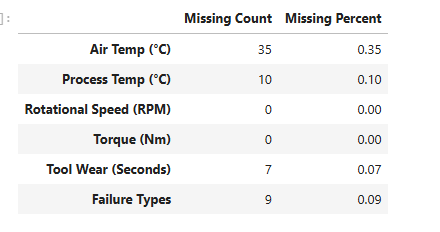
سپس ویژگی ها رو با دستور مربوطه بدست آوردیم که شامل تعداد سطر و ستون، هر ستون چند تا داده نال داره و میسینک ولیو هامون چطوریه . همچنین نوع داده ها (ستون Failure types از نوع object و بقیه float هستن ...) مشخص شدند.



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.

****

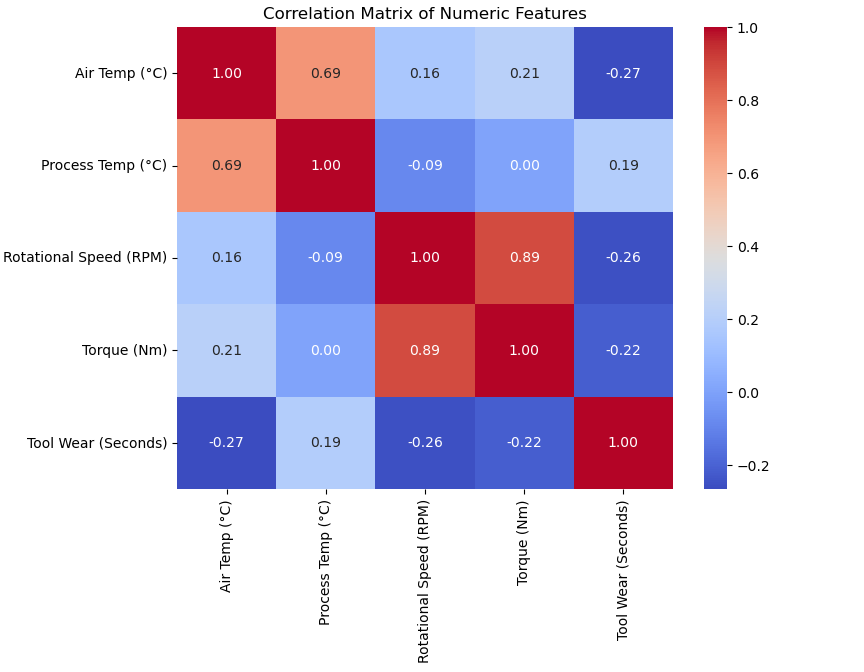
**الف-۲ بررسی مقادیر گم شده (Missing Values)**

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.

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 کتابخانه Seaborn، یک هیت مپ از این ماتریس ترسیم شد.

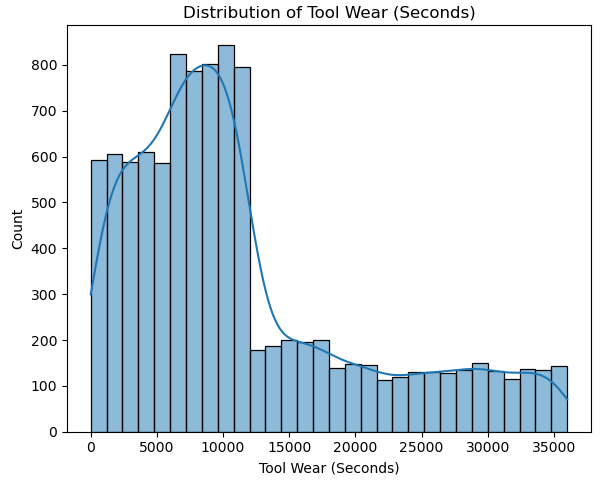


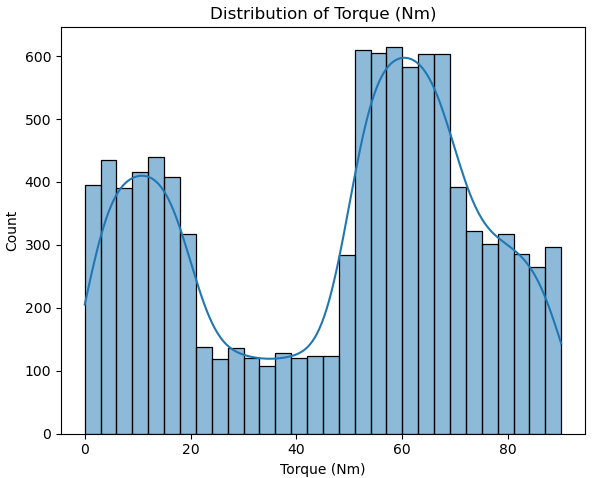
به طور کلی، اکثر ویژگی‌ها دارای همبستگی کم تا متوسط با یکدیگر بودند Process Temp و Air Temp بیشترین همبستگی مثبت را با هم داشتند (ضریب حدود 0.8)؛ سایر ویژگی‌ها مانند Tool Wear، Torque و Rotational Speed همبستگی‌های ضعیف‌تری با یکدیگر دارند، که نشان‌دهنده اطلاعات مستقل آن‌ها در پیش‌بینی خروجی است.

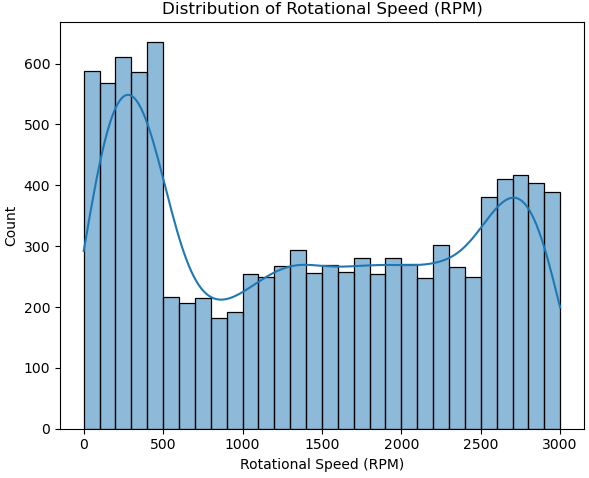
**الف-۴ رسم نمودار توزیع مشاهدات برای سه ویژگی با بیشترین تأثیر**

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.

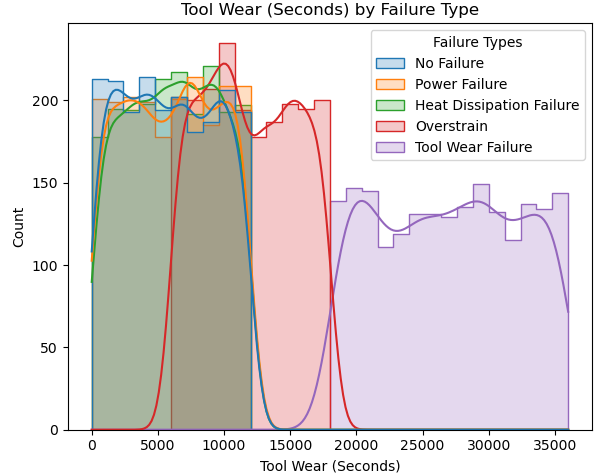
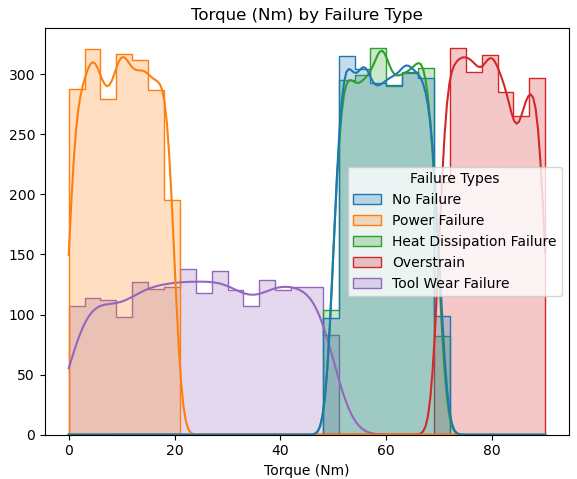
بر اساس بررسی همبستگی (در مرحله‌ی قبل)، سه ویژگی Tool Wear ، Torque و Rotational Speed به عنوان مهم‌ترین ویژگی‌های عددی انتخاب شدند؛ در ابتدا، توزیع هر یک از ویژگی‌های فوق به صورت جداگانه و بدون توجه به نوع خطا ترسیم شد:

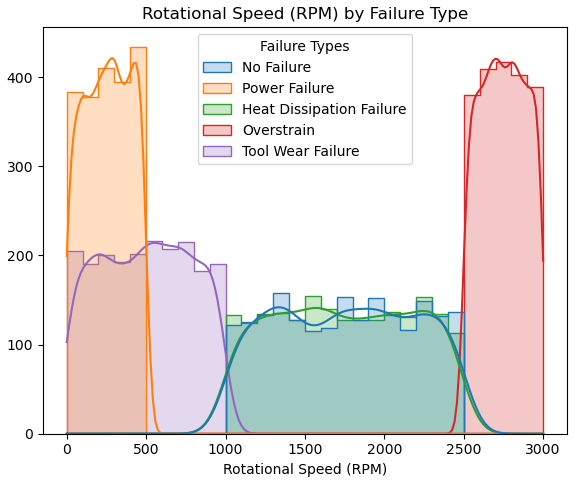






* توی Tool Wear بیشتر داده‌ها بین ۵ تا ۱۵ هزار ثانیه هستن، ولی یه سری مقدار خیلی بالا هم داشتیم که نشون میده بعضی ابزارها خیلی بیشتر از بقیه استفاده شده اند.
* در Torque ، داده‌ها توی بازه‌ی ۲۰ تا ۷۰ بیشتر جمع شده اند.
* Rotational Speed هم دو تا قله داره؛ انگار در دو بازه‌ی سرعت خاص، دستگاه بیشتر کار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

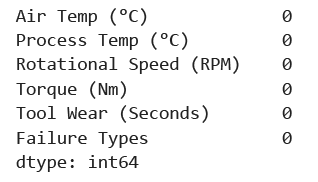
**ب- ۱ بررسی و حل مشکل مقادیر ناموجود (Missing Values)**

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 و Tool 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.

****

**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 باMinMaxScaler 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

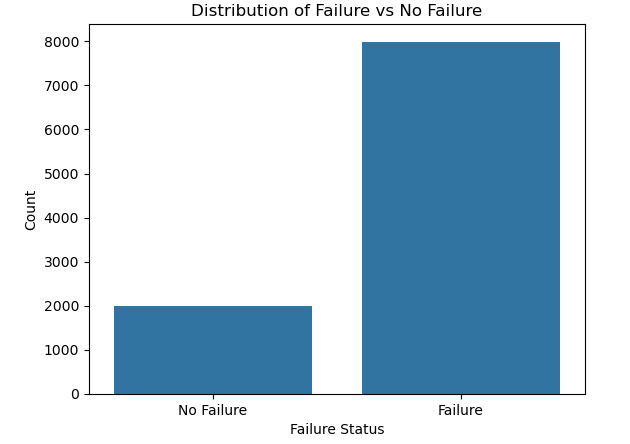
**ج-۱ ایجاد ستون برچسب دوگانه (Failure / No Failure)**

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 Failureکه 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 کلاسFailure خیلی بیشتر ازNoFailure 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.

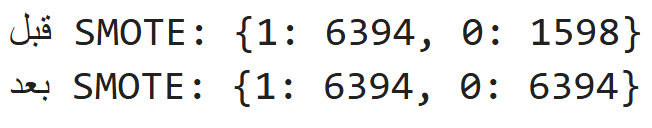
**ج-۳ توضیح مشکل عدم توازن داده‌ها**

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 مدلهیچ‌چیزی 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.



مشاهده میکنیم که قیل 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 با SMOTE، چهار تا مدل مختلف رو روی داده‌های آموزش‌شده تست کردیم تا ببینیم کدوم بهتر عمل می‌کنه. مدل‌ها عبارت بودند از:

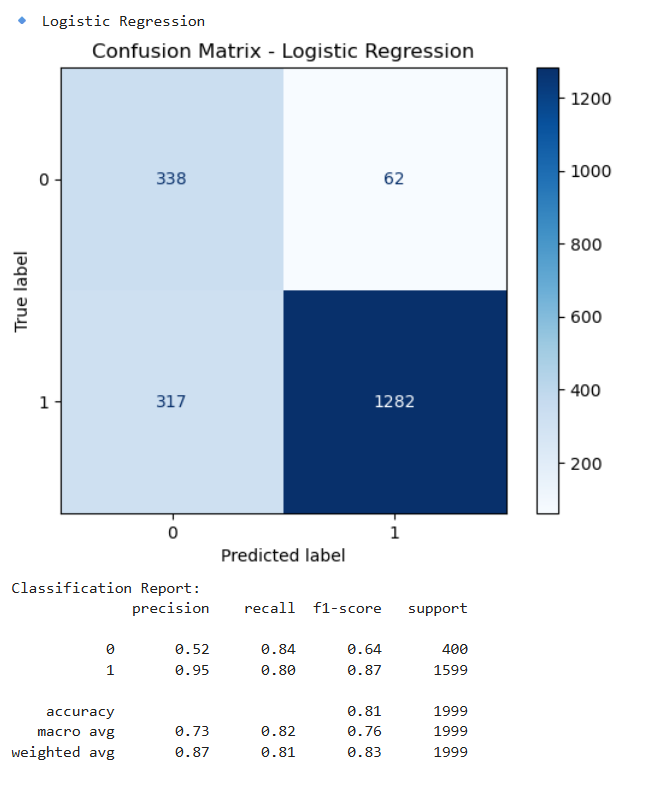
**1. Logistic Regression**

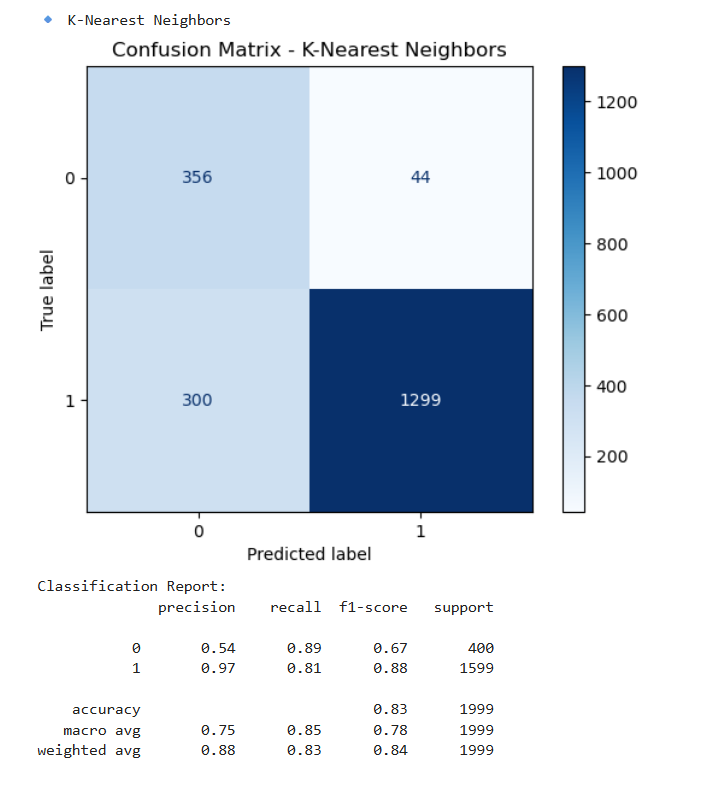
**2. K-Nearest Neighbors (KNN)**

**3. LinearSVC**

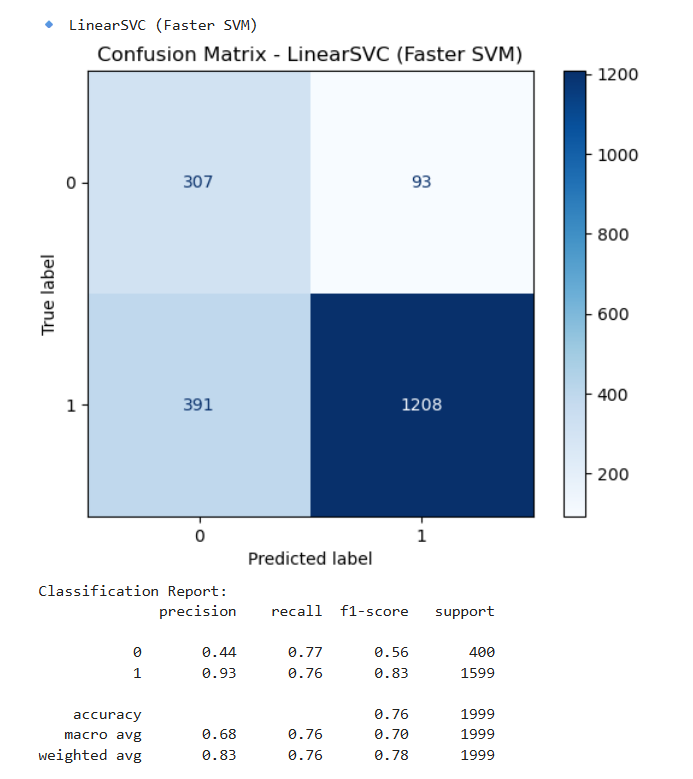
**SVC با کرنل(kernel) RBF (غیرخطی)4.**

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.

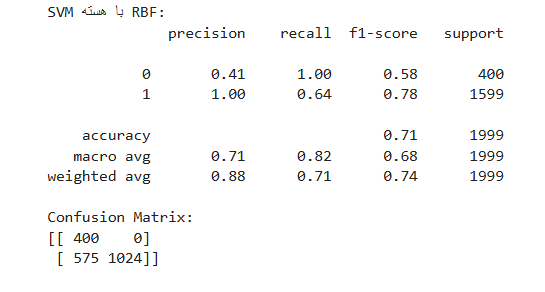
**1. Logistic Regression:** دقت حدود ۸۱ درصد، توی کلاس Failure خوب بود ولی No Failure رو کمتر درست تشخیص داد.



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



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



**4. SVC با کرنل(kernel) RBF (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:

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

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

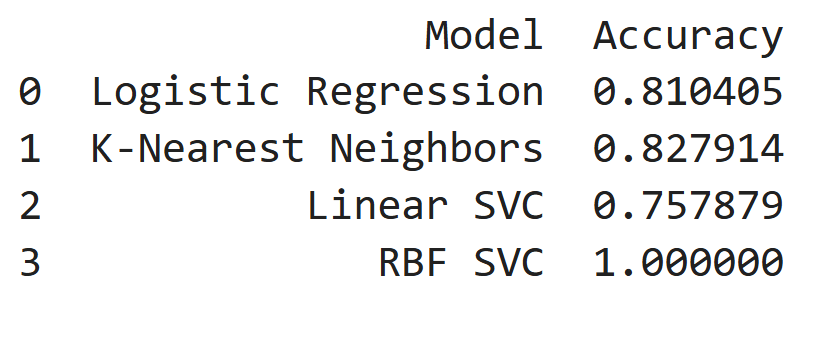
KNN متعادل‌ترین عملکرد رو داشت و تونست هر دو کلاس رو با دقت نسبتاً خوبی تشخیص بده.

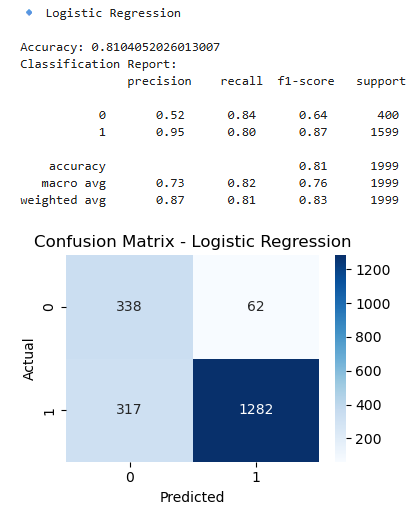
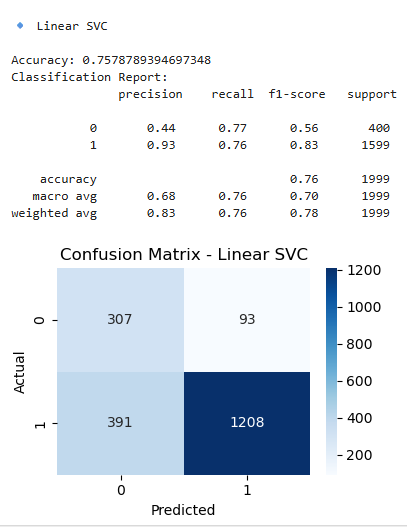
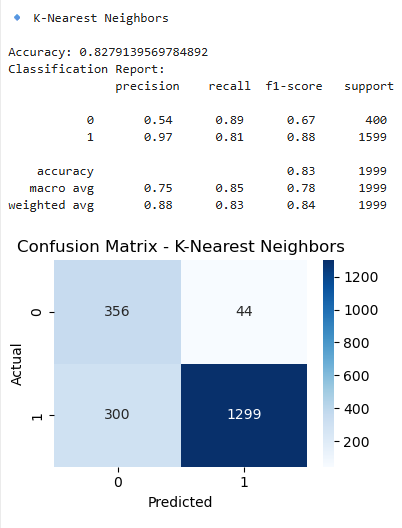
Logistic Regression هم خوب بود ولی نسبت به KNN کمی ضعیف‌تر عمل کرد.

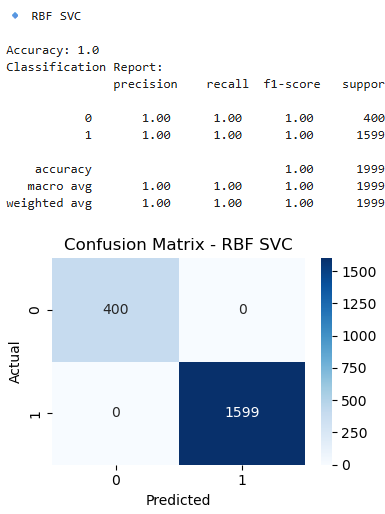
LinearSVC توی تشخیص کلاس No Failure ضعیف‌تر بود.

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، KNNانتخاب بهتری برای این مسئله به نظر می‌رسه، مخصوصاً وقتی که تعادل بین دقت در هر دو کلاس برامون مهم باشه.







**ج-۷ تنظیم هایپرپارامترها با GridSearchCV**

توی این بخش، اومدیم برای مدل‌هایی که قبلاً استفاده کرده بودیم، به‌جای اینکه با تنظیمات پیش‌فرض آموزش بدیم، ازGridSearchCV 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٪



1. 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٪



1. 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٪



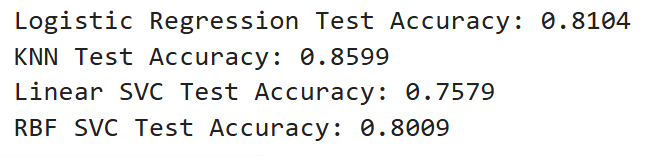
1. برایSVCبا کرنلRBF پارامترهایی که تست کردیم:
   * C از ۰.۱ تا ۱۰
   * gamma: scale، ۰.۰۱، ۰.۱، ۱

نتیجه:

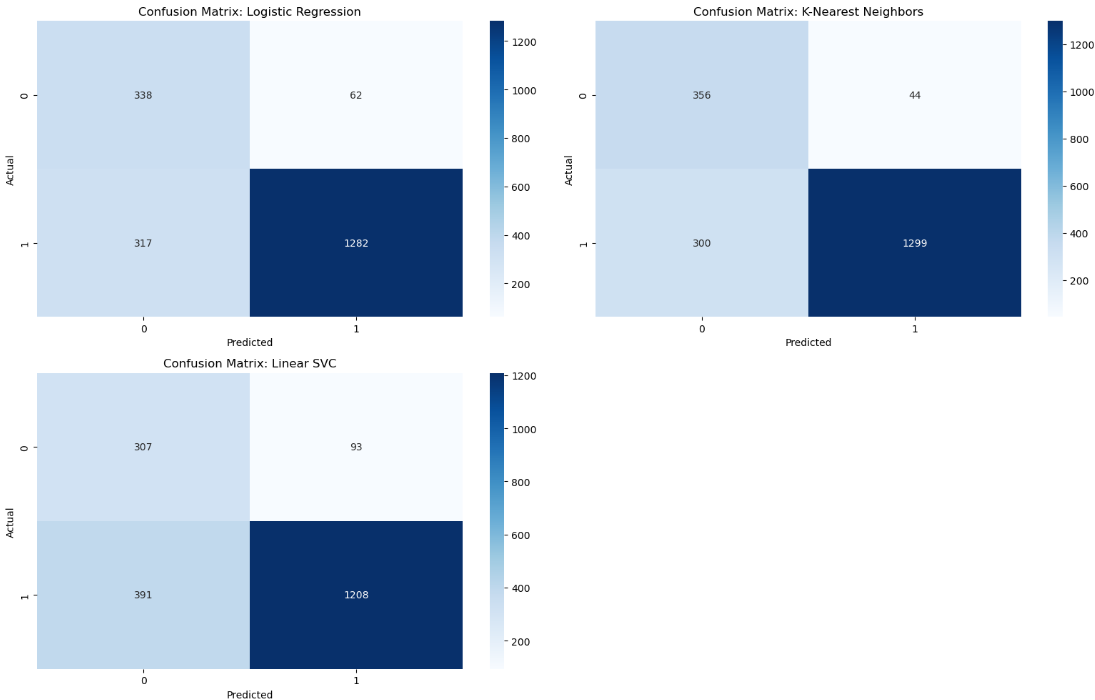
* بهترین حالت C=10 و gamma='scale'
* دقت (cross-val) : حدود 86.8٪

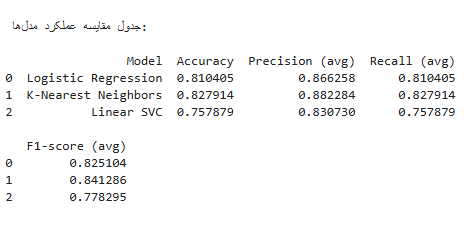


در نهایت، با استفاده از بهترین مدل‌هایی که از GridSearch به دست اومدن، دوباره روی داده‌های تست ارزیابی انجام دادم تا ببینم چقدر بهبود حاصل شده.



**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)



**KNN** بهترین عملکرد رو داشت، هم از نظر دقت کلی، هم میانگین Logistic .F1-score 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**

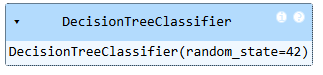
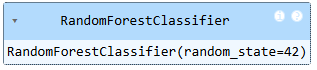
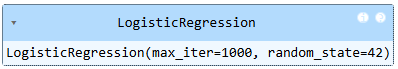
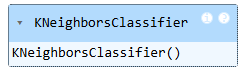
توی این بخش، بعد از آماده‌سازی داده‌ها توی مرحله‌ی قبل، چند مدل مختلف رو روی داده‌های چندکلاسه آموزش دادبم تا ببینم کدوم یکی بهتر می‌تونه نوع خرابی رو تشخیص بده؛ هر چهار مدل رو روی داده‌های آموزش آموزش فیت کردیم سپس با داده‌های تست عملکردشون رو بررسی کردیم. مدل هامون:

**K-Nearest Neighbors (KNN)**

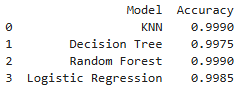
**Decision Tree**

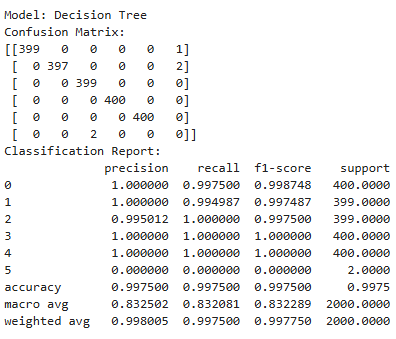
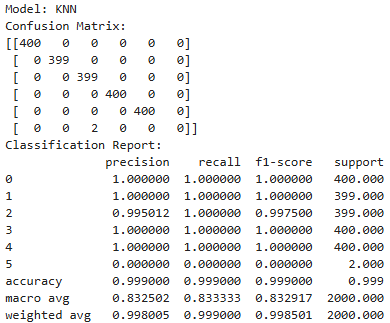
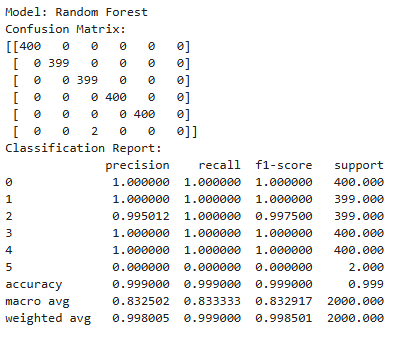
**Random Forest**

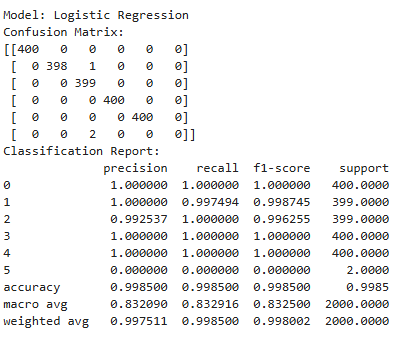
**Logistic Regression**



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.







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 با GridSearchCV**

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
* دقت Cross-Validation: حدود 99.89٪



2. برای KNN پارامترهایی که تست کردیم:

* n\_neighbors : از ۱ تا ۲۰

نتیجه:

* بهترین مقدار n\_neighbors=3
* دقت: حدود 99.88٪

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٪

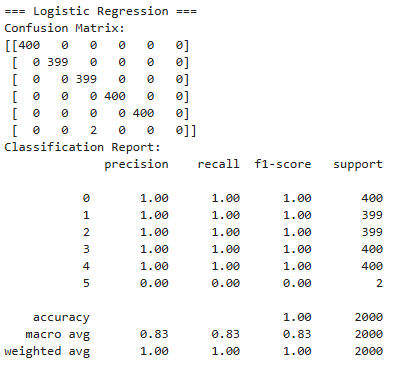


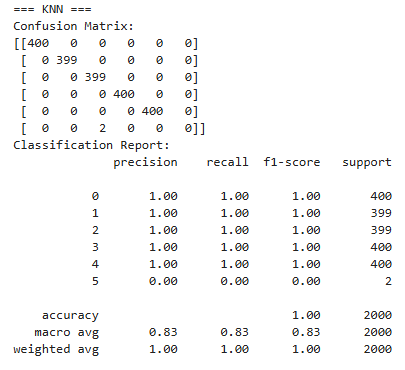
با اینکه دقت همه‌ی مدل‌ها خیلی بالا بود و نتایج قابل اعتمادی نیست ولی با GridSearch نشون دادیم حتی با دقت اولیه‌ی بالا هم می‌شه بهینه‌سازی انجام داد و مقادیرکمی بهتر شد. مخصوصاً توی مدل‌های KNN و Random Forest، انتخاب درست k روی دقت تأثیر می‌ذاره.

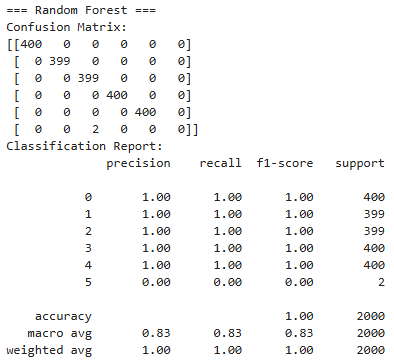
**د-۴ ارزیابی مدل‌های بهینه‌شده روی داده‌های تست**

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:

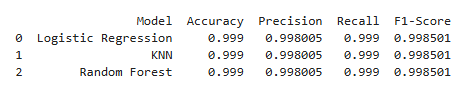
* Logistic Regression (بهینه‌شده با C=10 و solver=lbfgs)
* K-Nearest Neighbors (با k=3)
* Random Forest (با 50 درخت و عمق نامحدود)







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



The End