هوش مصنوعی

تمرین سوم

مائده اسماعیل‌زاده

۸۱۰۶۰۲۱۶۱

استاد: دکتر شریعت‌پناهی

دانشکده مهندسی مکانیک

پردیس دانشکده‌های فنی دانشگاه تهران

فهرست مطالب

بررسی داده‌های خام

الف-۱) ساختار کلی داده‌ها

نتیجه مربوط به این کد نشان می‌دهد فایل به خوبی در محیط آپلود شده است. در مرحله بعد برای خواندن اطلاعات موجود در فایل مربوطه، کد نوشته شده است.

```
import pandas as pd
from google.colab import files

uploaded = files.upload()
milling = pd.read_csv('milling_machine.csv')

print("Sakhtare kolie dadeha:")
print(milling.info())

print("Kholase amarie dadeha:")
print(milling.describe())
```

```
Choose Files milling_machine.csv
• milling_machine.csv(text/csv) - 1067729 bytes, last modified: 4/14/2025 - 100% done
Saving milling_machine.csv to milling_machine (1).csv
Sakhtare kolie dadeha:
<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
None
Kholase amarie dadeha:
       Air Temp (°C)  Process Temp (°C)  Rotational Speed (RPM)   Torque (Nm)  \
count    9965.000000        9990.000000            10000.000000  10000.000000
mean       28.516926          80.812186             1401.909988     46.998845
std         7.719340          15.548350              968.446183     26.747646
min        20.001366          60.001876                0.047731      0.015920
25%        23.176455          68.090324              423.672240     18.091381
50%        26.212082          76.553203             1377.047835     54.983239
75%        29.377536          92.825894             2307.969925     67.258375
max        49.998008         119.971025             2999.953724     89.993221

       Tool Wear (Seconds)
count          9993.000000
mean          11393.143344
std            9023.336380
min               3.469877
25%            5023.027818
50%            8995.172952
75%           15024.825673
max           35999.566519
```

همانطور که مشاهده می‌شود در ابتدا فایل milling_machine.csv حاوی اطلاعات مربوط به پایش ابزار برش دستگاه فرز بارگذاری شد. داده‌ها شامل ۶ ویژگی در ۱۰۰۰۰ نمونه هستند. خروجی اجرای دستور info() و describe() به شرح زیر تحلیل می‌شوند:

نوع داده‌ها بصورت پنج ستون عددی با نوع داده float64 و یک ستون متنی با نوع داده object (ستون Failure Types) می‌باشد.

همانطور که مشاهده می‌شود ستون Air Temp (°C) دارای ۳۵ مقدار گمشده، ستون Process Temp (°C) دارای ۱۰ مقدار گمشده، ستون Tool Wear (Seconds) دارای ۷ مقدار گمشده و ستون Failure Types دارای ۹ مقدار گمشده می‌باشند.

همچنین ویژگی‌های آماری نشان می‌دهند که ویژگی‌ها دارای تنوع بالایی هستند و برخی ستون‌ها مانند Tool Wear، Torque و Rotational Speed دارای بازه تغییرات وسیعی‌اند که ممکن است در ادامه نیاز به استانداردسازی یا نرمال‌سازی در مراحل بعدی داشته باشند.

الف-۲) تعداد و نسبت مقادیر ناموجود

لازم به ذکر است در این بخش ستون Failure Types شامل مقادیر متنی می‌باشد که نشان‌دهنده وضعیت ابزار فرز در شرایط مختلف هستند. از آنجاییکه در این بخش هدف اصلی تشخیص این است که آیا ابزار سالم یا معیوب است، برای ساده‌سازی مسئله و مناسب‌سازی آن جهت استفاده از مدل‌هایی مانند Logistic Regression ، SVM، یاKNN ، تصمیم گرفته شد که این ستون به صورت باینری تبدیل شود. به این صورت که اگر مقدار اصلی ستون برابر No Failure باشد (ابزار سالم)، صفر و اگر مقدار ستون نشان‌دهنده هر نوع خرابی دیگری باشد (ابزار معیوب)، یک و اگر مقدار اصلی ناموجود باشد، NaN می‌شوند.

پس از ایجاد ستون باینری برای نشان دادن وضعیت سلامت ابزار(Failure Binary) ، بررسی مقادیر گمشده در داده‌ها انجام شد. در این مرحله مطابق کد زیر از دستور isnull().sum() برای شمارش داده‌های گمشده و mean() برای محاسبه نسبت آن‌ها استفاده شد. مشاهده می‌شود تعداد و نسبت داده‌های ناموجود در کل داده‌ها نسبتاً کم است (همه کمتر از ۰.۵٪).

```python
import numpy as np

milling['Failure Binary'] = np.where(milling['Failure Types'].isnull(), np.nan,
                                     (milling['Failure Types'] != 'No Failure').astype(int))


# Hazfe sutune failure types ghabl az binary shodan
milling = milling.drop(columns=['Failure Types'])

# Maghadire gomshode dar har sotoon
missing_count = milling.isnull().sum()
missing_ratio = (milling.isnull().mean() * 100).round(2)

missing_milling = pd.DataFrame({'Missing Count': missing_count,
                                'Missing Ratio (%)': missing_ratio})

print("Maghadire namojoud dar har sotoon:")
print(missing_milling)
```

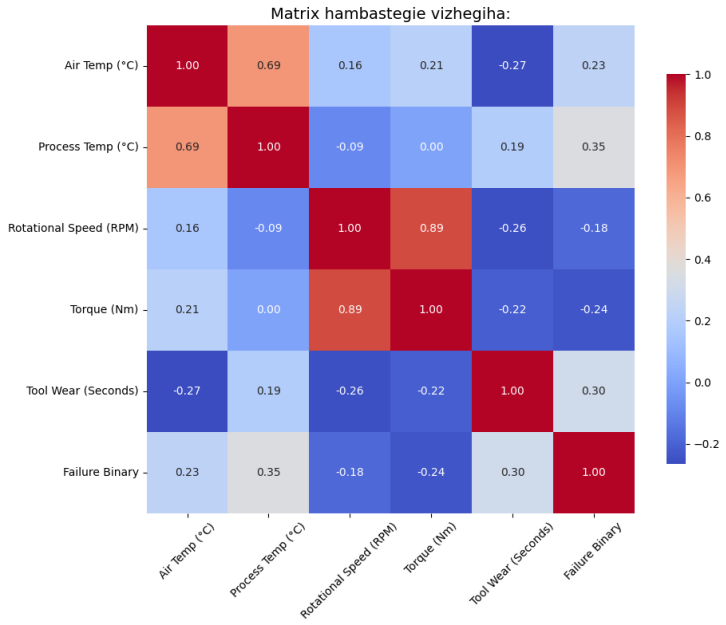
```
Maghadire namojoud dar har sotoon:
                         Missing Count  Missing Ratio (%)
Air Temp (°C)                       35               0.35
Process Temp (°C)                   10               0.10
Rotational Speed (RPM)               0               0.00
Torque (Nm)                          0               0.00
Tool Wear (Seconds)                  7               0.07
Failure Binary                       9               0.09
```

الف-۳) ماتریس همبستگی

برای تحلیل ارتباط بین ویژگی‌های عددی موجود در داده‌ها، ابتدا ماتریس همبستگی بین ویژگی‌ها محاسبه شد. نتایج در قالب یک **heatmap** بصورت زیر نمایش داده شدند.

```
[ ] import seaborn as sns
    import matplotlib.pyplot as plt

    # Sotoonhaye adadi
    sutun_adadi = milling.select_dtypes(include=['float64', 'int64'])

    corr_matrix = sutun_adadi.corr()
    print(corr_matrix)

    # Vabastegie vizhegiha
    print("Hambastegie vizhegiha ba vaziate abzar (Binary shodeye sotoon failure types):")
    print(corr_matrix['Failure Binary'].drop('Failure Binary').sort_values(ascending=False))

    # Heatmap
    plt.figure(figsize=(10, 8))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True, cbar_kws={"shrink": .8})
    plt.title('Matrix hambastegie vizhegiha:', fontsize=14)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

```
                        Air Temp (°C)  Process Temp (°C)  \
Air Temp (°C)                1.000000           0.693634
Process Temp (°C)            0.693634           1.000000
Rotational Speed (RPM)       0.156958          -0.086616
Torque (Nm)                  0.213051           0.002808
Tool Wear (Seconds)         -0.266371           0.188924
Failure Binary               0.230165           0.353105

                        Rotational Speed (RPM)  Torque (Nm)  \
Air Temp (°C)                         0.156958     0.213051
Process Temp (°C)                    -0.086616     0.002808
Rotational Speed (RPM)                1.000000     0.888487
Torque (Nm)                           0.888487     1.000000
Tool Wear (Seconds)                  -0.256103    -0.216528
Failure Binary                       -0.183681    -0.241635

                        Tool Wear (Seconds)  Failure Binary
Air Temp (°C)                     -0.266371        0.230165
Process Temp (°C)                  0.188924        0.353105
Rotational Speed (RPM)            -0.256103       -0.183681
Torque (Nm)                       -0.216528       -0.241635
Tool Wear (Seconds)                1.000000        0.304649
Failure Binary                     0.304649        1.000000
Hambastegie vizhegiha ba vaziate abzar (Binary shodeye sotoon failure types):
Process Temp (°C)         0.353105
Tool Wear (Seconds)       0.304649
Air Temp (°C)             0.230165
Rotational Speed (RPM)   -0.183681
Torque (Nm)              -0.241635
Name: Failure Binary, dtype: float64
```



مشاهده می‌شود بیشترین همبستگی بین ویژگی‌های عددی، بین Rotational Speed (RPM) و Torque (Nm) با ضریب همبستگی حدود 0.89 می‌باشد. دمای هوا (Air Temp) و دمای فرآیند (Process Temp) نیز همبستگی نسبتاً بالایی برابر با 0.69 دارند. همچنین با بررسی همبستگی ستون باینری‌شده‌ی نوع خطا با سایر ویژگی‌ها، مشخص شد که Process Temp (°C)بیشترین ارتباط را با وضعیت ابزار دارد(0.35) . پس از آن، Tool Wear (Seconds) (0.30)وAir Temp (°C) (0.23)نیز همبستگی مثبتی نشان می‌دهند. در مقابل، Rotational Speed (RPM)وTorque (Nm)همبستگی منفی با وضعیت ابزار دارند.

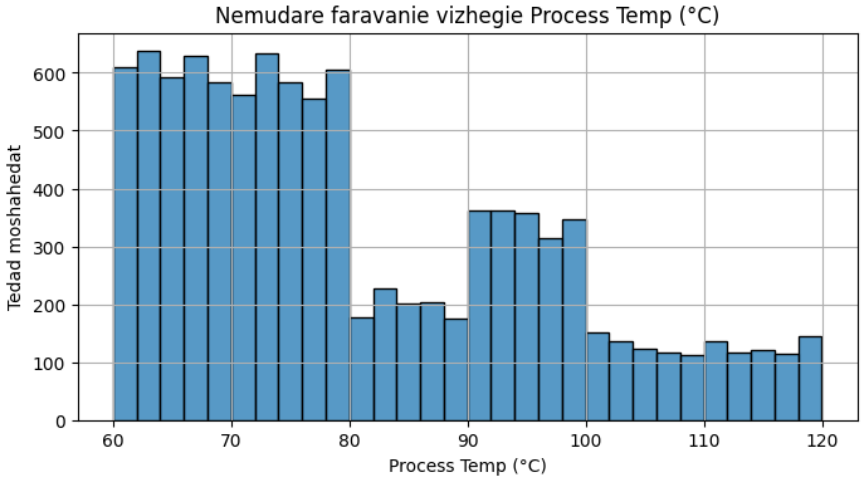
الف-۴) رسم نمودار تعداد مشاهدات هر مقدار

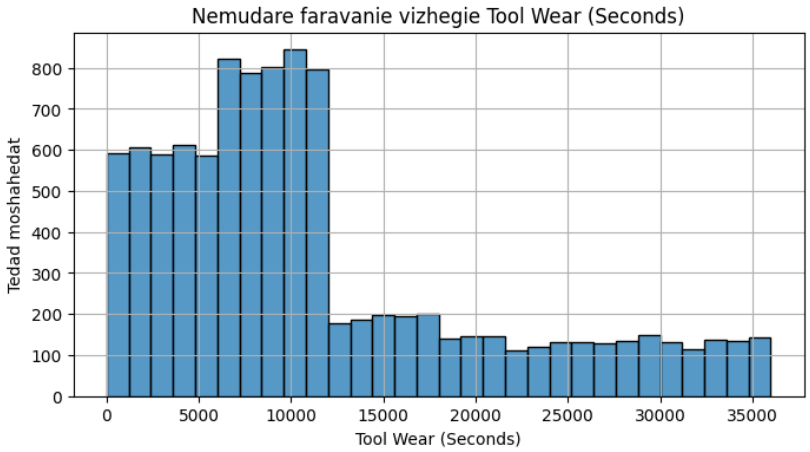
در ابتدا با استفاده از کد زیر سه ویژگی با همبستگی بالاتر به شرح زیر می‌باشند. سپس برای رسم نمودار خواسته شده کد زیر نوشته شده است.

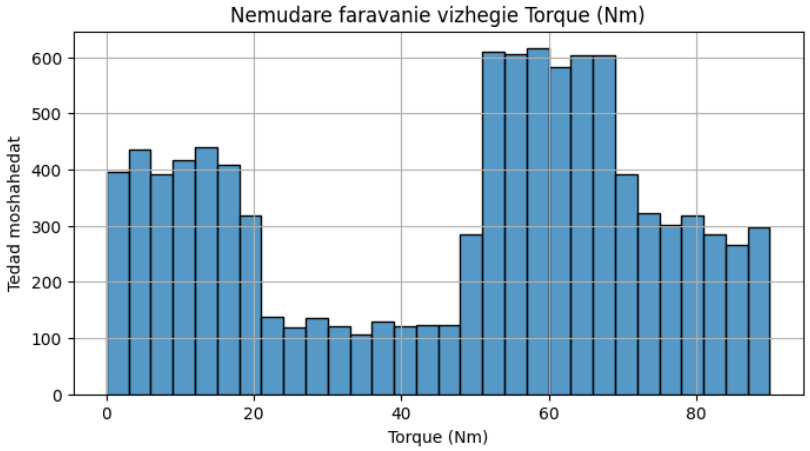
```
[ ]  # 3 vizhegi ba hambastegie bishtar
     correlations = corr_matrix['Failure Binary'].drop('Failure Binary').abs().sort_values(ascending=False)
     vizhegiha = correlations.head(3).index.tolist()
     print("3 vizhegi ba bishtarin hambastegi ba khuruji:")
     print(vizhegiha)
```

```
3 vizhegi ba bishtarin hambastegi ba khuruji:
['Process Temp (°C)', 'Tool Wear (Seconds)', 'Torque (Nm)']
```

```
[ ]  # Rasme nemudare faravani
     for feature in vizhegiha:
         plt.figure(figsize=(8, 4))
         sns.histplot(milling[feature], bins=30, kde=False)
         plt.title(f'Nemudare faravanie vizhegie {feature}')
         plt.xlabel(feature)
         plt.ylabel('Tedad moshahedat')
         plt.grid(True)
         plt.show()
```







Process Temp (°C) در بازه‌ای نسبتاً گسترده (حدود ۶۰ تا ۱۲۰ درجه سانتی‌گراد) توزیع شده است. یک تراکم زیاد مشاهدات بین بازه ۶۰ تا ۸۰ درجه مشاهده می‌شود که نشان می‌دهد در این محدوده بیشترین عملیات فرزکاری انجام شده است. از حدود ۸۰ تا ۱۰۰ درجه، کاهش شدید در تعداد مشاهدات وجود دارد که ممکن است نشان‌دهنده ناهنجاری یا شرایط خاص عملیاتی باشد. مجدداً در بازه‌های بالای ۹۰ درجه، تراکم کمی وجود دارد که می‌تواند مرتبط با رخداد خرابی یا شرایط غیرعادی باشد.

Tool Wear (Seconds) سایش ابزار توزیعی دو تکه‌ای دارد. در بازه‌ی پایین (۰ تا حدود ۱۰۰۰۰۰ ثانیه) تمرکز زیادی از مشاهدات دیده می‌شود. در مقادیر بالاتر از ۱۰۰۰۰۰ ثانیه، به‌مرور فراوانی کاهش می‌یابد اما همچنان مشاهدات پراکنده‌ای وجود دارد. این توزیع می‌تواند نشان‌دهنده‌ی تفاوت در دوره‌های استفاده از ابزار و احتمالاً سیاست‌های نگهداری و تعویض ابزار باشد.

Torque (Nm) گشتاور نیز دارای توزیعی چندتکه است. دو قله در بازه‌های حدود ۱۰–۲۰ و ۵۵–۷۰ نیوتن‌متر مشاهده می‌شود. نشان‌دهنده‌ی وجود دو حالت مختلف در دستگاه است: یکی با گشتاور پایین (شاید بدون بار یا بار سبک) و دیگری با گشتاور بالا (بار عملیاتی یا بار سنگین). احتمالاً تغییر در حالت کاری یا نوع عملیات فرزکاری (نظیر فرز سبک یا سنگین) باعث این الگو شده است.

بطور کلی توزیع‌های غیر یکنواخت در ویژگی‌های مهمی مانند دمای فرآیند، سایش ابزار و گشتاور، نشان‌دهنده‌ی وجود الگوهای رفتاری خاص در داده‌هاست. این الگوها می‌توانند نشانه‌ای از تغییر وضعیت ابزار (از سالم به معیوب) باشند و برای طراحی مدل‌های یادگیری ماشین بسیار مهم هستند.

پیش‌پردازش داده‌ها

ب-۱) جایگزینی مقادیر ناموجود

مطابق کد زیر در ابتدا با استفاده از توابع ()isnull و ()sum تعداد مقادیر گمشده برای هر ستون بررسی شد. مقادیر گمشده در هر ستون در بخش قبل آمده بود. ستون های Air temp، Process temp، Failure types(Binary) و Tool wear بیشترین مقادیر ناموجود را داشتند. برای ویژگی‌هایی با نوع داده‌ی عددی float64 یا int64، از میانگین (mean) برای جایگزینی مقادیر گمشده استفاده شد. زیرا میانگین به عنوان شاخص مرکزی داده‌ها در صورتی‌که توزیع نرمال یا نزدیک به نرمال باشد، گزینه‌ی مناسبی است. در این تمرین، بیشتر ویژگی‌های عددی مانند دما، سرعت، و فرسایش ابزار، پراکندگی شدیدی نداشتند. برای ستون‌هایی با نوع داده‌ی کیفی (object)، از مد (mode) برای جایگزینی استفاده شد. زیرا مد رایج‌ترین مقدار در داده است و برای داده‌های کیفی که مقدار غالبی دارند، مناسب است.

```python
# Jaygozinie maghadire gomshode ba miangin baraye sutunhaye adadi
sutun_adadi_gomshode = missing_milling.index[missing_milling['Missing Count'] > 0]

for col in sutun_adadi_gomshode:
    if milling[col].dtype in ['float64', 'int64']:
        milling[col] = milling[col].fillna(milling[col].mean())

# Jaygozinie maghadire gomshode ba mode baraye sutunhaye daste bandi
sutun_daste_gomshode = missing_milling.index[missing_milling['Missing Count'] > 0]

for col in sutun_daste_gomshode:
    if milling[col].dtype == 'object':
        milling[col] = milling[col].fillna(milling[col].mode()[0])

# Barresi mojadad
missing_count_after = milling.isnull().sum()
missing_ratio_after = (milling.isnull().mean() * 100).round(2)

missing_milling_after = pd.DataFrame({
    'Missing Count': missing_count_after,
    'Missing Ratio (%)': missing_ratio_after
})

print("Maghadire namojoud pas az jaygozini:")
print(missing_milling_after)
```

```
Maghadire namojoud pas az jaygozini:
                         Missing Count  Missing Ratio (%)
Air Temp (°C)                        0                0.0
Process Temp (°C)                    0                0.0
Rotational Speed (RPM)               0                0.0
Torque (Nm)                          0                0.0
Tool Wear (Seconds)                  0                0.0
Failure Binary                       0                0.0
```

مشاهده می‌شود هیچ مقدار گمشده‌ای موجود نیست.

ب-۲) استانداردسازی و نرمال‌سازی

فرآیندهای **استانداردسازی** و **نرمال‌سازی** از مراحل مهم پیش‌پردازش داده‌ها هستند که برای ویژگی‌های کمی به کار می‌روند تا داده‌ها برای مدل‌های یادگیری ماشین آماده شوند. در استانداردسازی، داده‌ها به گونه‌ای تغییر می‌یابند که میانگین برابر با صفر و انحراف معیار برابر با یک شود. این کار باعث می‌شود ویژگی‌ها در توزیع نرمال استاندارد قرار گیرند. در نرمال‌سازی، داده‌ها به گونه‌ای تغییر می‌یابند که در بازه‌ای مشخص (معمولاً $[0, 1]$) قرار گیرند. این کار به ویژه برای الگوریتم‌هایی که از فاصله‌ی بین نمونه‌ها استفاده می‌کنند اهمیت دارد. فرمول مربوط به این دو روش بصورت زیر می‌باشند:

$$\text{استاندارد سازی } z = \frac{x - \mu}{\sigma}$$

$$\text{نرمال سازی } x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

از آنجاییکه در این تمرین، هدف اصلی افزایش دقت و سرعت همگرایی مدل‌ها و بهبود عملکرد آن‌ها در حالت کلی است، نیاز به یکی از این دو روش می‌باشد. زیرا ویژگی‌ها در مقیاس‌های متفاوتی هستند (مثلاً Torque از ۰ تا ۹۰، ولی Tool Wear از ۰ تا ۳۵،۰۰۰). همچنین مدل‌هایی مانند KNN و SVM که بر اساس فاصله عمل می‌کنند، نسبت به مقیاس داده‌ها بسیار حساس‌اند. از طرفی نمودارها نشان دادند که بعضی ویژگی‌ها غیربهینه هستند، که استفاده از نرمال‌سازی یا استانداردسازی را مفید می‌کند. در استانداردسازی تبدیل داده‌ها به میانگین صفر و انحراف معیار یک اتفاق می‌افتد. در نرمال‌سازی فشرده‌سازی داده‌ها در بازه $[0, 1]$ اتفاق می‌افتد. با توجه به مدل‌هایی که در این تمرین استفاده شده‌اند، مدل‌های حساس به مقیاس مانندLogistic Regression، K-Nearest Neighbors (KNN)، وSupport Vector Machine (SVM) منطقی است که که تنها از استانداردسازی (StandardScaler)استفاده شود. این روش برای مدل‌های حساس مانندSVM ، KNNو Logistic Regression عملکرد مناسبی دارد و تأثیر منفی بر مدل‌هایی مانند درخت تصمیم یا جنگل تصادفی نخواهد داشت. بنابراین، داده‌های عددی با استفاده از کتابخانه scikit-learnو ابزار StandardScaler بصورت زیر مقیاس‌بندی شدند. نکته لازم به ذکر آن است که نمی‌توان از هر دو روش با هم استفاده نمود، زیرا اگر پشت سر هم اعمال شوند، ممکن است یک روش، نتیجه روش دیگر را در داده‌ها از بین ببرد.

0s

```
from sklearn.preprocessing import StandardScaler

target_cols = ['Failure Binary', 'Failure Types']

sutun_adadi = milling.select_dtypes(include=['float64', 'int64']).columns
sutun_adadi = [col for col in sutun_adadi if col not in target_cols]

# Shey'e standard
scaler = StandardScaler()

# Standardsazi sutunhaye adadi
milling[sutun_adadi] = scaler.fit_transform(milling[sutun_adadi])

print("Dadeha pas az standardsazi:")
print(milling[sutun_adadi].describe())
```

```
Dadeha pas az standardsazi:
       Air Temp (°C)  Process Temp (°C)  Rotational Speed (RPM)   Torque (Nm)  \
count   1.000000e+04       1.000000e+04            1.000000e+04  1.000000e+04
mean   -1.818989e-16       1.136868e-16            6.821210e-17  2.955858e-16
std     1.000050e+00       1.000050e+00            1.000050e+00  1.000050e+00
min    -1.105137e+00      -1.339162e+00           -1.447610e+00 -1.756613e+00
25%    -6.920985e-01      -8.181528e-01           -1.010161e+00 -1.080802e+00
50%    -2.954023e-01      -2.726269e-01           -2.567349e-02  2.985232e-01
75%     1.101337e-01       7.729670e-01            9.356279e-01  7.574701e-01
max     2.787784e+00       2.519907e+00            1.650194e+00  1.607488e+00

       Tool Wear (Seconds)
count         10000.000000
mean              0.000000
std               1.000050
min              -1.262752
25%              -0.705906
50%              -0.265293
75%               0.401393
max               2.728068
```

دسته‌بندی دوگانه

ج-۱) ایجاد برچسب

در این بخش یک ستون جدید به نام "Failure Label"به داده‌ها اضافه شده است؛ به این صورت که اگر مقدار "Failure Binary" برابر با صفر باشد برچسب "No Failure" و اگر مقدار "Failure Binary"برابر با یک باشد، برچسب "Failure"می‌گیرد. با توجه به نتیجه این کد ۸۰۰۲ داده معیوب و ۱۹۹۸ داده سالم می‌باشند.

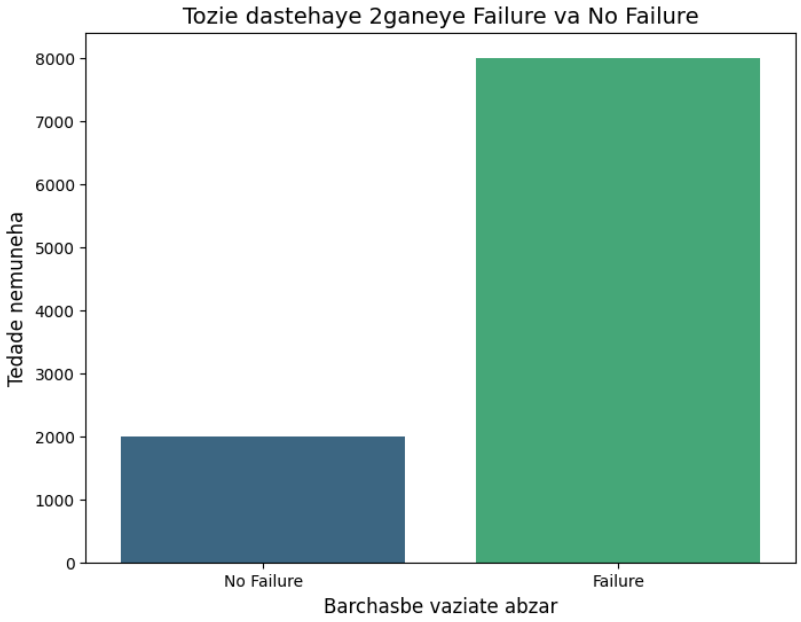
```
[ ]  # Sutune jadid ba barchasbe "no failure" va "failure"
     milling['Failure Label'] = np.where(milling['Failure Binary'] == 0, 'No Failure', 'Failure')

     # Tedad barchasbha
     failure_counts = milling['Failure Label'].value_counts()

     print("Tedad nemuneha baraye har barchasb:")
     print(failure_counts)
```

```
Tedad nemuneha baraye har barchasb:
Failure Label
Failure       8002
No Failure    1998
Name: count, dtype: int64
```

ج_۲و۳) نمودار میله‌ای

در این بخش از کد زیر استفاده شده است. نمودار میله‌ای زیر نشان می‌دهد که داده‌ها در دو کلاس Failure و No Failure به صورت نامتوازن توزیع شده‌اند. از ۱۰,۰۰۰ نمونه حدود ۸۰۰۰ نمونه مربوط به ابزارهای معیوب (Failure) و حدود ۲۰۰۰ نمونه به ابزارهای سالم (No Failure) اختصاص دارند. این عدم توازن می‌تواند باعث شود مدل تمایل بیشتری به پیش‌بینی کلاس Failure داشته باشد و کلاس No Failure را نادیده بگیرد. این عدم توازن می‌تواند منجر به عملکرد ضعیف مدل‌های یادگیری ماشین در شناسایی ابزار سالم شود. در چنین شرایطی، برای بهبود عملکرد مدل باید از تکنیک‌هایی مانند SMOTE استفاده کرد که در بخش بعد انجام شده است.

```
0s  # Nemudar mileyi
    plt.figure(figsize=(8, 6))
    sns.countplot(data=milling, x='Failure Label', hue='Failure Label', palette='viridis', legend=False)

    plt.title('Tozie dastehaye 2ganeye Failure va No Failure', fontsize=14)
    plt.xlabel('Barchasbe vaziate abzar', fontsize=12)
    plt.ylabel('Tedade nemuneha', fontsize=12)
    plt.show()
```



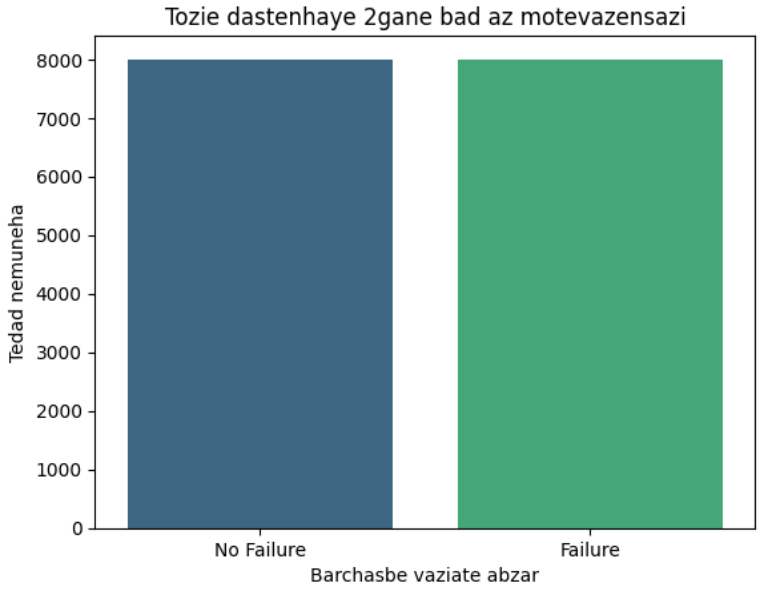
ج_۴) رفع عدم توازن داده‌ها

در این بخش کد بصورت زیر نوشته شده است. SMOTE با تولید نمونه‌های مصنوعی برای کلاس اقلیت، داده‌ها را متوازن می‌کند بدون آنکه نمونه‌های کلاس غالب را حذف کند. مراحل انجام این کد به این صورت است که ابتدا ویژگی‌ها (X) و برچسب هدف (y)جدا شدند. سپس از کتابخانه imbalanced-learn، کلاس SMOTE با random_state=42 فراخوانی شد. در نهایت داده‌ها با دستور fit_resample() بازنمونه‌گیری شدند. نمودار میلهٔ نهایی نشان می‌دهد که داده‌ها به شکل متوازن بین دو کلاس تقسیم شده‌اند، که این موضوع به افزایش دقت F1-score و تعادل عملکرد مدل در شناسایی هر دو کلاس کمک خواهد کرد.

```
[ ] !pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.11/dist-packages (0.13.0)
Requirement already satisfied: numpy<3,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (2.0.2)
Requirement already satisfied: scipy<2,>=1.10.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.15.3)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.6.1)
Requirement already satisfied: sklearn-compat<1,>=0.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (0.1.3)
Requirement already satisfied: joblib<2,>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.5.0)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (3.6.0)
```

```python
from imblearn.over_sampling import SMOTE
from collections import Counter

# Vizhegi
X = milling.drop(columns=['Failure Label', 'Failure Binary'])
# Hadaf
y = milling['Failure Label']

# Emale SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

print("Tedad nemuneha pas az motevazen sazi:", Counter(y_resampled))

# Tedad nemuneha dar har daste
sns.countplot(x=y_resampled, palette='viridis')
plt.title('Tozie dastenhaye 2gane bad az motevazensazi')
plt.xlabel('Barchasbe vaziate abzar')
plt.ylabel('Tedad nemuneha')
plt.show()
```



ج_۵ و ۶) آموزش مدل‌ها

در این مرحله، مطابق کد زیر داده‌های متوازن‌شده بخش قبل به دو بخش آموزش (۸۰٪) و آزمون (۲۰٪) تقسیم شدند و سپس چهار مدل طبقه‌بندی مختلف با استفاده از کتابخانه scikit-learn آموزش داده شدند.

```
[12] from sklearn.model_selection import train_test_split

     X_train, X_test, y_train, y_test = train_test_split(
          X_resampled, y_resampled, test_size=0.2, random_state=42,
          stratify=y_resampled)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

models = {'Logistic Regression': LogisticRegression(max_iter=1000),
          'KNN (k=5)': KNeighborsClassifier(n_neighbors=5),
          'SVM (Linear Kernel)': SVC(kernel='linear'),
          'SVM (RBF Kernel)': SVC(kernel='rbf')}

for name, model in models.items():
    print(f"Model: {name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(classification_report(y_test, y_pred))
    print("Matrix ashoftegi:")
    print(confusion_matrix(y_test, y_pred))
```

```
Model: Logistic Regression
              precision    recall  f1-score   support

     Failure       0.84      0.79      0.82      1601
  No Failure       0.80      0.85      0.83      1600

    accuracy                           0.82      3201
   macro avg       0.82      0.82      0.82      3201
weighted avg       0.82      0.82      0.82      3201

Matrix ashoftegi:
[[1272  329]
 [ 243 1357]]
Model: KNN (k=5)
              precision    recall  f1-score   support

     Failure       1.00      1.00      1.00      1601
  No Failure       1.00      1.00      1.00      1600

    accuracy                           1.00      3201
   macro avg       1.00      1.00      1.00      3201
weighted avg       1.00      1.00      1.00      3201

Matrix ashoftegi:
[[1601    0]
 [   0 1600]]
Model: SVM (Linear Kernel)
              precision    recall  f1-score   support

     Failure       0.88      0.77      0.82      1601
  No Failure       0.80      0.90      0.84      1600

    accuracy                           0.83      3201
   macro avg       0.84      0.83      0.83      3201
weighted avg       0.84      0.83      0.83      3201

Matrix ashoftegi:
[[1237  364]
 [ 168 1432]]
Model: SVM (RBF Kernel)
              precision    recall  f1-score   support

     Failure       1.00      1.00      1.00      1601
  No Failure       1.00      1.00      1.00      1600

    accuracy                           1.00      3201
   macro avg       1.00      1.00      1.00      3201
weighted avg       1.00      1.00      1.00      3201

Matrix ashoftegi:
[[1601    0]
 [   0 1600]]
```

با توجه به نتایج می‌توان گفت مدلLogistic Regression دقت ۸۲ درصد داشته و تقریباً متوازن بین دو کلاس Failure و No Failure بوده است. با استفاده از ماتریس آشفتگی می‌توان گفت ۱۲۷۲ نمونه از کلاس Failure به درستی، ۳۲۹ نمونه از Failure به اشتباه No Failure ، ۲۴۳ نمونه از No Failure به اشتباه Failure و ۱۳۵۷ نمونه از No Failure به درستی پیش‌بینی شده‌اند. بطور کلی می‌توان گفت نسبت به مدل‌های پیچیده‌تر دقت کمتری دارد و نمونه‌هایی را اشتباه طبقه‌بندی کرده است.

مدل KNN با K=5 دقت ۱۰۰ درصد داشته و بدون هیچ گونه پیش‌بینی اشتباه بوده است.لازم به ذکر است مدل به نظر بیش‌برازش (Overfitting)دارد، زیرا روی داده‌های آزمون عملکرد بی‌نقص دارد. این مورد باید در برابر داده‌های واقعی‌تر یا جدیدتر بررسی شود تا از پایداری آن اطمینان حاصل شود.

مدل SVM با هسته خطی دقت ۸۳ درصد داشته است. کلاس No Failure بهتر شناسایی شده است(Recall=90٪) . همچنین از روی ماتریس آشفتگی می‌توان گفت ۱۲۳۷ نمونه از Failure درست و ۳۶۴ نمونه اشتباه به No Failure نسبت داده شدند. بطور کلی مدل SVM با هسته خطی عملکرد بهتری نسبت به Logistic Regression داشته، ولی همچنان در شناسایی برخی از

نمونه‌های کلاس Failure ضعیف عمل کرده است.

مدل SVM با هسته غیرخطی نیز دقت ۱۰۰ درصد داشته و مشابه KNN کاملاً بدون خطا عمل کرده که نشان‌دهنده قابلیت بالای آن در تشخیص الگوهای غیرخطی در داده‌هاست. با این حال، احتمال بیش‌برازش نیز مطرح است و نیاز به ارزیابی روی داده‌های واقعی دارد.

بطور کلی می‌توان گفت مدل‌های غیرخطی مانند SVM-RBF و KNN توانستند به دقت کامل برسند، که نشان از توان بالای آن‌ها در یادگیری الگوهای پیچیده دارد. با این حال، دقت ۱۰۰٪ ممکن است نشانه‌ی بیش‌برازش باشد، بخصوص وقتی داده‌ها با SMOTE ساخته شده‌اند. همچنین مدل Logistic و SVM خطی نتایج قابل قبول‌تری ولی نه کامل دارند و در کاربردهای حساس‌تر ممکن است کمتر قابل اطمینان باشند.

برای نتایج دقیق‌تر از cross validation نیز استفاده شده است. کد و نتایج بصورت زیر قابل مشاهده هستند.

```
from sklearn.model_selection import cross_val_score

print("Natayeje Cross Validation:")
for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    print(f"{name}: دقت میانگین = {scores.mean():.4f}, انحراف معیار = {scores.std():.4f}")
```

```
Natayeje Cross Validation:
Logistic Regression: 0.0041 = انحراف معیار ,0.8332 = دقت میانگین
KNN (k=5): 0.0002 = انحراف معیار ,0.9997 = دقت میانگین
SVM (Linear Kernel): 0.0052 = انحراف معیار ,0.8438 = دقت میانگین
SVM (RBF Kernel): 0.0004 = انحراف معیار ,0.9995 = دقت میانگین
```

با توجه به نتایج مدل‌هایی مثل KNN و SVM-RBF دقت بسیار بالا و انحراف معیار بسیار پایین دارند. این ترکیب نشان‌دهنده آن است که مدل‌ها روی داده‌ی آموزشی خیلی خوب عمل کرده‌اند. اما اگر داده واقعی‌تر، نویزدارتر یا متفاوت‌تری باشد، احتمال دارد دقت‌شان به شدت افت کند. بنابراین می‌توان گفت ممکن است Overfitting اتفاق افتاده باشد. مدل‌های Logistic Regression و SVM خطی دقتی در حدود ۸۳–۸۵٪ با انحراف معیار کم دارند که واقع‌بینانه‌تر و پایدارتر است.

ج-۷) مقدار بهینه پارامترها

کد مربوط به این بخش بصورت زیر نوشته شده است.

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

param_knn = {'n_neighbors': [3, 5, 7, 9, 11]}
grid_knn = GridSearchCV(KNeighborsClassifier(), param_knn, cv=5, scoring='accuracy')
grid_knn.fit(X_train, y_train)

print("Behtarin Parameterha baraye KNN:", grid_knn.best_params_)
print("Behtarin deghat:", grid_knn.best_score_)
```

```
Behtarin Parameterha baraye KNN: {'n_neighbors': 3}
Behtarin deghat: 0.9996875915169856
```

بهترین پارامتر و دقت برای مدل KNN در نتیجه کد قابل مشاهده می‌باشد. دقت همچنان بسیار بالا باقی مانده است. می‌توان گفت احتمال باقی‌ماندن Overfiitting همچنان وجود دارد . علت این اتفاق در این تمرین به دلیل سادگی مدل و نزدیکی زیاد نمونه‌ها در داده SMOTE شده می‌باشد.

برای مدل Logistic Regression مقدار کوچکتر C به معنی regularization بیشتر است و باعث کاهش پیچیدگی مدل و جلوگیری از Overfitting می‌شود. نسبت به قبل عملکرد کمی بهبود یافته است.

```
from sklearn.linear_model import LogisticRegression

param_logreg = {'C': [0.01, 0.1, 1, 10],'penalty': ['l2'],
    'solver': ['lbfgs']}

grid_logreg = GridSearchCV(LogisticRegression(max_iter=1000), param_logreg, cv=5, scoring='accuracy')
grid_logreg.fit(X_train, y_train)

print("Behtarin Parameterha baraye Logistic Regression:", grid_logreg.best_params_)
print("Behtarin deghat:", grid_logreg.best_score_)
```

```
Behtarin Parameterha baraye Logistic Regression: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
Behtarin deghat: 0.8395689855037094
```

برای مدل SVM هسته خطی نیز کاهش C منجر به حاشیه تصمیم (margin) بزرگ‌تر و مدل با تعمیم بهتر شده است. دقت نسبت به مقدار پیش‌فرض (C=1) بهتر شده است.

```
[18] from sklearn.svm import SVC

    param_svm_linear = {'C': [0.1, 1, 10],'kernel': ['linear']}

    grid_svm_linear = GridSearchCV(SVC(), param_svm_linear, cv=5, scoring='accuracy')
    grid_svm_linear.fit(X_train, y_train)

    print("Behtarin Parameterha baraye SVM (Khati):", grid_svm_linear.best_params_)
    print("Behtarin deghat:", grid_svm_linear.best_score_)
```

```
Behtarin Parameterha baraye SVM (Khati): {'C': 0.1, 'kernel': 'linear'}
Behtarin deghat: 0.84667704876025
```

برای مدل SVM با هسته غیرخطی با شرایط جدید، مدل هنوز دقتی بسیار بالا دارد و همچنان مشکوک به بیش‌برازش است. مقادیر بزرگ gamma می‌توانند مدل را بیش از حد پیچیده کنند و عملکرد ضعیفی در داده واقعی داشته باشند.

بطور کلی می‌توان گفت در مدل‌هایی مثل Logistic Regression و SVM خطی، بهبود دقت کاملاً مشهود است و ریسک بیش‌برازش کاهش یافته است. اما در مدل‌های KNN و SVM-RBF ، اگرچه دقت بسیار بالاست، اما هنوز احتمال بیش‌برازش باقی است و بهتر است این مدل‌ها روی داده‌ی واقعی‌تر یا داده‌هایی با نویز تست شوند.

ج_۸) مقایسه مدل‌ها

معیارهای ارزیابی زیادی برای مدل مناسب وجود دارند. یکی از معیارها درستی (Accuracy) یعنی نسبت پیش‌بینی‌های صحیح به کل نمونه‌ها می‌باشد. معیار بعدی دقت (Precision) به معنی نسبت نمونه‌های مثبتِ درست پیش‌بینی‌شده به کل نمونه‌هایی که مثبت پیش‌بینی شده‌اند می‌باشد. مورد بعدی Recall به معنی نسبت نمونه‌های مثبتِ درست پیش‌بینی‌شده به کل نمونه‌های مثبت واقعی می‌باشد. F1-score (شاخصی متعادل)، ماتریس آشفتگی برای بررسی جزئیات خطا در هر کلاس، Cross-validation accuracy برای سنجش پایداری مدل و ریسک Overfitting (اختلاف زیاد بین train و test یا دقت بسیار بالا غیرعادی می‌باشد) از دیگر معیارهای مقایسه مدل‌ها می‌باشد.

با توجه به موارد ذکر شده، مدل‌های Logistic Regression و SVM با هسته خطی دقت خوبی دارند و رفتارشان متعادل است. گزینه‌های مناسب برای مدل‌هایی هستند که باید قابل‌تعمیم و قابل‌تفسیر باشند. مدل‌های KNN و SVM-RBF مدل‌هایی با دقت بسیار بالا هستند، اما به دلیل داده‌های تولید شده با SMOTE و فاصله زیاد بین کلاس‌ها، احتمال Overfitting بالاست. باتوجه به نتایج Cross-validation مدل‌هایی با دقت نزدیک به ۰.۸۵ و عدم دقت بیش از حد روی داده‌های آموزشی، و مناسب‌ترین انتخاب‌ها برای دنیای واقعی هستند.

دسته‌بندی چندگانه

د_۱ و ۲) آموزش مدل‌ها

در ابتدا از همان داده‌های بخش‌های قبل استفاده شده است که کد و نتایج در زیر قابل مشاهده می‌باشند. یکی از علت‌های این که دقت تمام مدل‌ها ۱۰۰ درصد شده است آن است که ستون هدف در بخش‌های قبل بصورت باینری بوده است که عملا برای دسته‌بندی چندگانه کاربردی نمی‌باشد. بنابراین پس از این کد یک کد جدید نوشته شده است که داده‌ها از ابتدا خوانده شده، سپس ستون آخر به همان صورت اولیه باقی مانده و باینری نشده است.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from sklearn.metrics import classification_report, accuracy_score

X = milling.drop(columns=['Failure Label'])
y = milling['Failure Label']  # Barchasbe hadaf
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# KNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
print("Deghate KNN:", accuracy_score(y_test, knn_pred))
print(classification_report(y_test, knn_pred))

# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)
print("Deghate Decision Tree:", accuracy_score(y_test, dt_pred))
print(classification report(y test, dt pred))
```

```
# Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
print("Deghate Random Forest:", accuracy_score(y_test, rf_pred))
print(classification_report(y_test, rf_pred))

# SVM (One-vs-Rest)
svm = OneVsRestClassifier(SVC(kernel='linear', random_state=42))
svm.fit(X_train, y_train)
svm_pred = svm.predict(X_test)
print("Deghate SVM (One-vs-Rest):", accuracy_score(y_test, svm_pred))
print(classification_report(y_test, svm_pred))

# SVM (One-vs-One)
svm_ovo = OneVsOneClassifier(SVC(kernel='linear', random_state=42))
svm_ovo.fit(X_train, y_train)
svm_ovo_pred = svm_ovo.predict(X_test)
print("Deghate SVM (One-vs-One):", accuracy_score(y_test, svm_ovo_pred))
print(classification_report(y_test, svm_ovo_pred))
```

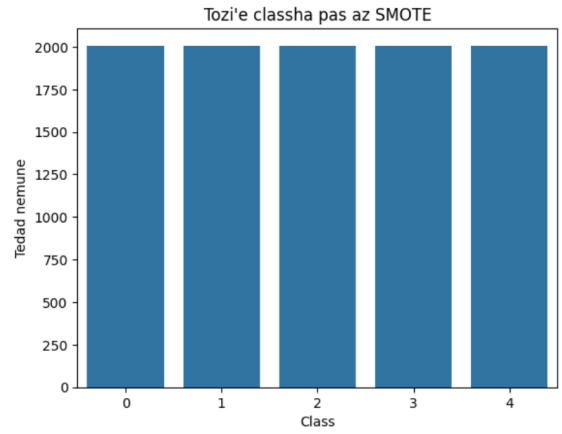
```
Deghate KNN: 1.0
              precision    recall  f1-score   support

     Failure       1.00      1.00      1.00      1590
  No Failure       1.00      1.00      1.00       410

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

Deghate Decision Tree: 1.0
              precision    recall  f1-score   support

     Failure       1.00      1.00      1.00      1590
  No Failure       1.00      1.00      1.00       410

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

Deghate Random Forest: 1.0
              precision    recall  f1-score   support

     Failure       1.00      1.00      1.00      1590
  No Failure       1.00      1.00      1.00       410

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

Deghate SVM (One-vs-Rest): 1.0
              precision    recall  f1-score   support

     Failure       1.00      1.00      1.00      1590
  No Failure       1.00      1.00      1.00       410

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

Deghate SVM (One-vs-One): 1.0
              precision    recall  f1-score   support

     Failure       1.00      1.00      1.00      1590
  No Failure       1.00      1.00      1.00       410

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

در حالت جدید، مراحل نوشته شده در کد زیر به این شرح می‌باشد: از ابتدا فایل داده‌ها خوانده شده و در این حالت ستون Failure Types در حالت باینری نمی‌باشد تا دسته‌بندی چندگانه به درستی انجام شود. در این کد داده‌ها مانند بخش قبل پالایش‌شده (با پر کردن مقادیر گمشده و استانداردسازی) سپس برای تعادل کلاس‌ها SMOTE اعمال شده و به دو مجموعه‌ی آموزش (۸۰٪) و آزمون (۲۰٪) تقسیم شدند. در مرحله بعد آموزش مدل‌ها انجام شده است؛ مدل‌ها با تنظیمات پیش فرض تعریف شده‌اند. سپس ماتریس آشفتگی، دقت (accuracy) و Classification Report هر مدل روی مجموعه‌ی آزمون به‌دست آمد که در بخش نتایج این کد قابل مشاهده می‌باشد.

```python
milling = pd.read_csv("milling_machine.csv")

# Barchasbgozarie chandgane
milling['Failure Types'] = milling['Failure Types'].fillna("No Failure")

# Pishpardazesh
numeric_features = milling.select_dtypes(include=['float64', 'int64']).columns
milling[numeric_features] = milling[numeric_features].fillna(milling[numeric_features].mean())

# Standardsazi
scaler = StandardScaler()
milling[numeric_features] = scaler.fit_transform(milling[numeric_features])

# Ramzgozarie barchasbha
le = LabelEncoder()
milling['Failure_Label_Multi'] = le.fit_transform(milling['Failure Types'])

# Namayeshe barchasbha
label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print("Mapping LabelEncoder:")
print(label_mapping)


X = milling.drop(columns=['Failure Types', 'Failure_Label_Multi'])
y = milling['Failure_Label_Multi']
```

```python
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

sns.countplot(x=y_resampled)
plt.title("Tozi'e classha pas az SMOTE")
plt.xlabel("Class")
plt.ylabel("Tedad nemune")
plt.show()

X_train, X_test, y_train, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.2, stratify=y_resampled, random_state=42)

# Modelha
models = {
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'SVM (OvR)': OneVsRestClassifier(SVC(kernel='rbf', random_state=42)),
    'SVM (OvO)': OneVsOneClassifier(SVC(kernel='rbf', random_state=42)),}
results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = (y_pred == y_test).mean()
    print(f"Model: {name}")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred, target_names=le.classes_))
    results[name] = acc
```

```
Mapping LabelEncoder:
{'Heat Dissipation Failure': np.int64(0), 'No Failure': np.int64(1), 'Overstrain': np.int64(2), 'Power Failure': np.int64(3), 'Tool Wear Failure': np.int64(4)}
```



```
Model: KNN
[[402   0   0   0   0]
 [  0 401   0   0   0]
 [  0   0 401   0   0]
 [  0   0   0 402   0]
 [  0   0   0   0 401]]
                         precision    recall  f1-score   support

Heat Dissipation Failure       1.00      1.00      1.00       402
              No Failure       1.00      1.00      1.00       401
              Overstrain       1.00      1.00      1.00       401
           Power Failure       1.00      1.00      1.00       402
       Tool Wear Failure       1.00      1.00      1.00       401

                accuracy                           1.00      2007
               macro avg       1.00      1.00      1.00      2007
            weighted avg       1.00      1.00      1.00      2007

Model: Decision Tree
[[400   2   0   0   0]
 [  1 400   0   0   0]
 [  0   0 401   0   0]
 [  0   0   0 402   0]
 [  0   0   0   0 401]]
                         precision    recall  f1-score   support

Heat Dissipation Failure       1.00      1.00      1.00       402
              No Failure       1.00      1.00      1.00       401
              Overstrain       1.00      1.00      1.00       401
           Power Failure       1.00      1.00      1.00       402
       Tool Wear Failure       1.00      1.00      1.00       401

                accuracy                           1.00      2007
               macro avg       1.00      1.00      1.00      2007
            weighted avg       1.00      1.00      1.00      2007

Model: Random Forest
[[402   0   0   0   0]
 [  0 401   0   0   0]
 [  0   0 401   0   0]
 [  0   0   0 402   0]
 [  0   0   0   0 401]]
                         precision    recall  f1-score   support

Heat Dissipation Failure       1.00      1.00      1.00       402
              No Failure       1.00      1.00      1.00       401
              Overstrain       1.00      1.00      1.00       401
           Power Failure       1.00      1.00      1.00       402
       Tool Wear Failure       1.00      1.00      1.00       401

                accuracy                           1.00      2007
               macro avg       1.00      1.00      1.00      2007
            weighted avg       1.00      1.00      1.00      2007

Model: SVM (OvR)
[[402   0   0   0   0]
 [  0 401   0   0   0]
 [  0   0 401   0   0]
 [  0   0   0 402   0]
 [  0   0   0   0 401]]
                         precision    recall  f1-score   support

Heat Dissipation Failure       1.00      1.00      1.00       402
              No Failure       1.00      1.00      1.00       401
              Overstrain       1.00      1.00      1.00       401
           Power Failure       1.00      1.00      1.00       402
       Tool Wear Failure       1.00      1.00      1.00       401

                accuracy                           1.00      2007
               macro avg       1.00      1.00      1.00      2007
            weighted avg       1.00      1.00      1.00      2007

Model: SVM (OvO)
[[402   0   0   0   0]
 [  0 401   0   0   0]
 [  0   0 401   0   0]
 [  0   0   0 402   0]
 [  0   0   0   0 401]]
                         precision    recall  f1-score   support

Heat Dissipation Failure       1.00      1.00      1.00       402
              No Failure       1.00      1.00      1.00       401
              Overstrain       1.00      1.00      1.00       401
           Power Failure       1.00      1.00      1.00       402
       Tool Wear Failure       1.00      1.00      1.00       401

                accuracy                           1.00      2007
               macro avg       1.00      1.00      1.00      2007
            weighted avg       1.00      1.00      1.00      2007
```

مشاهده می‌شود تمام ماتریس‌های آشفتگی یکسان بوده و پیش‌بینی‌ها کاملاً دقیق (تمام نمونه‌ها روی قطر اصلی ماتریس) هستند که یا نشانه‌ی Overfitting شدید روی داده‌های SMOTE شده است و یا اینکه داده‌ها از ابتدا ساده و قابل تفکیک بوده‌اند. با بررسی داده‌ها می‌توان گفت داده‌ها ساده و قابل تفکیک هستند، به همین دلیل دقت تمام مدل‌ها ۱۰۰ درصد بدست آمده است. راه‌هایی برای این که جلوی این اتفاق گرفته شود وجود دارد؛ ارزیابی نهایی روی داده‌های واقعی یا بدونSMOTE ، قرار دادن SMOTE

درون چرخه‌یCross-Validation ، بررسی معیارهای مقاوم‌تر و یا افزودن نویز یا کاهش ویژگی‌های تفکیک‌کننده.

د-۳) مقدار بهینه پارامترها

برای این بخش کد زیر نوشته شده است. برای هر مدل، مهم‌ترین هایپرپارامترها در محدوده‌های زیر که در نتایج قابل مشاهده می‌باشد، بهینه شدند.

```
# KNN
param_grid_knn = {'n_neighbors': [3, 5, 7, 9]}
grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='accuracy')
grid_knn.fit(X_train, y_train)

# Decision Tree
param_grid_tree = {'max_depth': [5, 10, 15],'min_samples_split': [2, 5, 10]}
grid_tree = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid_tree, cv=5, scoring='accuracy')
grid_tree.fit(X_train, y_train)

# Random Forest
param_grid_rf = {'n_estimators': [50, 100],'max_depth': [5, 10, None]}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5, scoring='accuracy')
grid_rf.fit(X_train, y_train)

# SVM OvR
param_grid_svm_ovr = {'estimator__C': [0.1, 1, 10],'estimator__gamma': [0.01, 0.1, 1]}
grid_svm_ovr = GridSearchCV(OneVsRestClassifier(SVC(kernel='rbf')), param_grid_svm_ovr, cv=5, scoring='accuracy')
grid_svm_ovr.fit(X_train, y_train)

# SVM OvO
grid_svm_ovo = GridSearchCV(OneVsOneClassifier(SVC(kernel='rbf')), param_grid_svm_ovr, cv=5, scoring='accuracy')
grid_svm_ovo.fit(X_train, y_train)

print("Behtarin parameterha:")
```

```
print("KNN:", grid_knn.best_params_)
print("Decision Tree:", grid_tree.best_params_)
print("Random Forest:", grid_rf.best_params_)
print("SVM (OvR):", grid_svm_ovr.best_params_)
print("SVM (OvO):", grid_svm_ovo.best_params_)
```

```
Behtarin parameterha:
KNN: {'n_neighbors': 3}
Decision Tree: {'max_depth': 5, 'min_samples_split': 5}
Random Forest: {'max_depth': 5, 'n_estimators': 50}
SVM (OvR): {'estimator__C': 1, 'estimator__gamma': 1}
SVM (OvO): {'estimator__C': 1, 'estimator__gamma': 1}
```

د-۴) مقایسه مدل‌ها

با استفاده از معیارهای درس مانندAccuracy ، Precision، Recall، F1-Score و ارزیابی Cross-Validation می‌توان گفت در این حالت همه مدل‌ها دقت و معیارهای ۱۰۰٪ نشان می‌دهند و نتایج Cross-Validation نیز نشان می‌دهد میانگین دقت بسیار بالا (۰.۹۹۹۷) و انحراف معیار اندک است. علت این اتفاق ساده‌سازی بیش از حد داده‌ها، تفکیک‌پذیری طبیعی کلاس‌ها و SMOTE بوده که باعث شده مدل‌های فاصله‌محور (KNN, SVM-RBF) و درختی توانایی "حفظ" کامل داده‌ها را پیدا کنند.

برای نتایج بهتر می‌توان ارزیابی نهایی را روی داده‌های واقعی یا بدونSMOTE قرار داد(در این حالت نیز دقت‌ها بالاست) و یا SMOTE را درون چرخه‌ی Cross-Validation قرار داد، معیارهای مقاوم‌تر بررسی نمود و یا افزودن نویز یا کاهش ویژگی‌های تفکیک‌کننده را انجام داد. در حالت کلی با داده‌های ساده این تمرین، این دقت‌ها منطقی می‌باشد.

****لازم به ذکر است گزارش و کد در گیت هاب نیز ارائه شده است. لینک آن در زیر آمده است****

https://github.com/maedehesmz8010/HW3_810602161