notebook

April 28, 2025

[192]:
```python
import pandas as pd
import re
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder
from typing import Tuple
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import time
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪GradientBoostingClassifier, ExtraTreesClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
```

[193]:
```python
data = pd.read_excel("./data/data.xlsx")
```

[194]:
```python
data.head()
```

[194]:
```
   explanation_quality  final_grade  study_hours absences group_work schedule  \
0                    5         9.46          4.3     <25%        oui    matin
1                    9         5.13         21.3     <25%        Non    matin
2                    7         9.67         18.9     <50%        Non    matin
3                    6        13.24          0.8     <25%        oui    matin
4                    3        12.28         48.0     <50%        oui    matin

  class_participation                              exam_content  \
0         un petit peu                                   mélange
1                moyen  fait parti du cours enseigner par le prof
2         un petit peu  fait parti du cours enseigner par le prof
3         un petit peu                                   mélange
4         un petit peu  fait parti du cours enseigner par le prof
```

```
                     result
0                non valide
1                non valide
2                    valide
3  valide apres rattrapage
4                    valide
```

[195]:
```python
data.describe()
```

[195]:
```
       explanation_quality  final_grade  study_hours
count          5000.000000  5000.000000  5000.000000
mean              5.343800    10.965118    23.969980
std               2.079631     2.536466     8.422406
min               1.000000     5.000000     0.000000
25%               4.000000     9.240000    19.400000
50%               5.500000    10.960000    24.600000
75%               7.000000    12.700000    29.700000
max              10.000000    20.000000    50.000000
```

[196]:
```python
data.isnull().sum()
```

[196]:
```
explanation_quality    0
final_grade            0
study_hours            0
absences               0
group_work             0
schedule               0
class_participation    0
exam_content           0
result                 0
dtype: int64
```

1 EDA

[197]:
```python
# How many rows and columns
data.shape
```

[197]:
```
(5000, 9)
```

[198]:
```python
# Quick look at the first few rows
data.head()
```

[198]:
```
   explanation_quality  final_grade  study_hours absences group_work schedule  \
0                    5         9.46          4.3     <25%        oui    matin
1                    9         5.13         21.3     <25%        Non    matin
2                    7         9.67         18.9     <50%        Non    matin
```

```
3                   6        13.24           0.8     <25%        oui    matin
4                   3        12.28          48.0     <50%        oui    matin

  class_participation                              exam_content  \
0        un petit peu                                   mélange
1               moyen  fait parti du cours enseigner par le prof
2        un petit peu  fait parti du cours enseigner par le prof
3        un petit peu                                   mélange
4        un petit peu  fait parti du cours enseigner par le prof

                    result
0               non valide
1               non valide
2                   valide
3  valide apres rattrapage
4                   valide
```

[199]:
```python
# Detailed info about types and missing values
data.info()
```

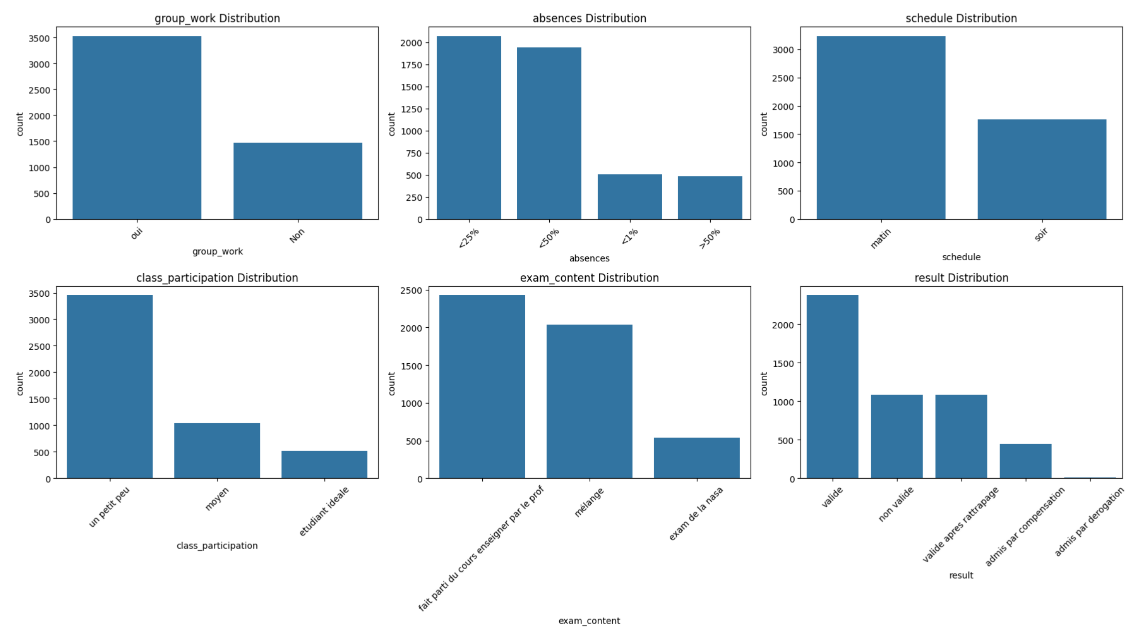
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   explanation_quality  5000 non-null   int64
 1   final_grade          5000 non-null   float64
 2   study_hours          5000 non-null   float64
 3   absences             5000 non-null   object
 4   group_work           5000 non-null   object
 5   schedule             5000 non-null   object
 6   class_participation  5000 non-null   object
 7   exam_content         5000 non-null   object
 8   result               5000 non-null   object
dtypes: float64(2), int64(1), object(6)
memory usage: 351.7+ KB
```

[200]:
```python
print(data['study_hours'].isna().sum())
```

```
0
```

[201]:
```python
# List of your categorical columns
categorical_cols = ['group_work', 'absences', 'schedule',␣
 ↪'class_participation', 'exam_content', 'result']

# Create subplots
plt.figure(figsize=(18, 10))  # Big figure for multiple plots
```

```python
for idx, col in enumerate(categorical_cols):
    plt.subplot(2, 3, idx + 1)  # 2 rows, 3 columns, plot number idx+1
    sns.countplot(x=col, data=data, order=data[col].value_counts().index)
    plt.title(f"{col} Distribution")
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



2 Interpretation of Categorical Feature Distributions

2.1 1. `group_work` Distribution

- "Non" (no group work) is more frequent than "oui" (yes group work).
- Students who did **not participate in group work** are slightly more common.
- This could mean that **group work is optional** or that **many students prefer to study individually**.
- Important later to check if group work participation impacts results.

2.2 2. `absences` Distribution

- `<1%` absence dominates, followed by `<25%`, `<50%`, and `>50%`.
- Most students **have very low absence rates** (good attendance).
- Attendance seems generally **good** across the dataset.
- Important: students with high absences (`>50%`) might have lower final results — to verify later.

2.3 3. `schedule` Distribution

- "matin" (morning classes) are much more common than "soir" (evening classes).
- Majority of the courses happen **in the morning**.
- Maybe evening students are special cases (working students?), could affect performance — needs checking.

2.4 4. `class_participation` Distribution

- "un petit peu" (a little participation) is the most common, followed by "moyen" (average participation), and fewer "étudiant idéal" (ideal students).
- Most students **only participate a little** in class.
- Very **few are model students** ("étudiant idéal").
- Important: more participation might correlate with better results — to check later.

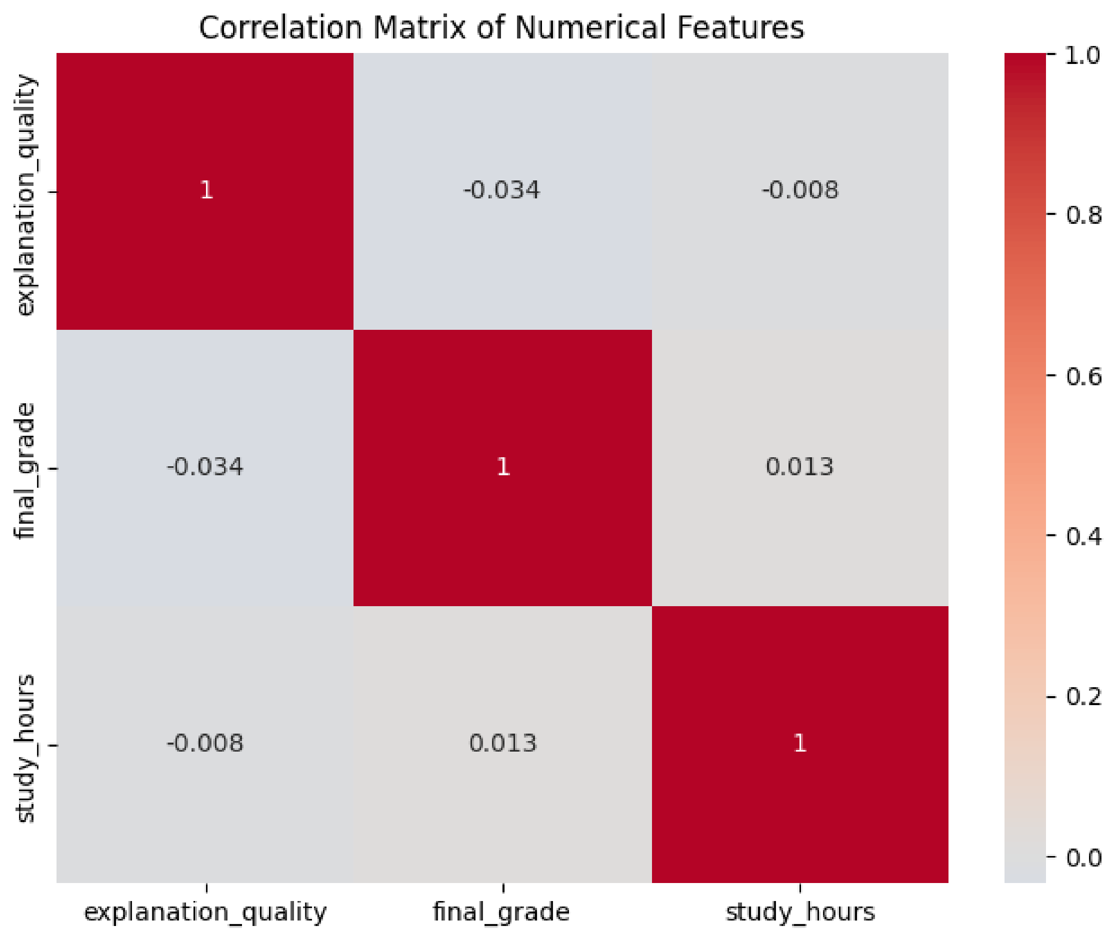
2.5 5. `exam_content` Distribution

- "fait parti du cours enseigner par le prof" dominates heavily, "mélange" and "exam de la nasa" are less frequent.
- Most exams **are directly based on course material taught**.
- Some exams ("mélange" or "exam de la nasa") may be harder or unexpected — worth checking if those students performed differently.

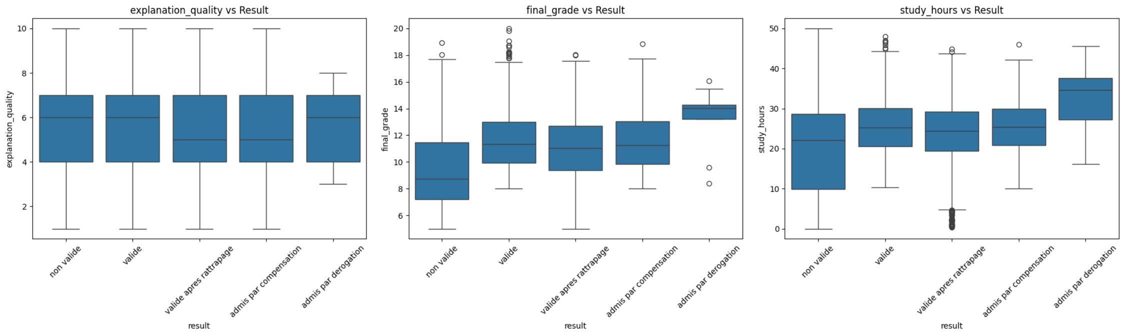
2.6 6. `result` (Target) Distribution

- "valide" is extremely dominant, while "valide après rattrapage", "admis par compensation" are fewer, and "non valide" and "admis par dérogation" are very rare.
- **Most students pass** their courses normally.
- Very few fail completely (`non valide`) or pass exceptionally (`dérogation`).
- The dataset is **imbalanced** toward positive results → class imbalance techniques might be necessary during modeling.

[202]:
```
# Select only numerical columns
numerical_cols = ['explanation_quality', 'final_grade', 'study_hours']

# Compute the correlation matrix
corr = data[numerical_cols].corr()

# Plot the heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

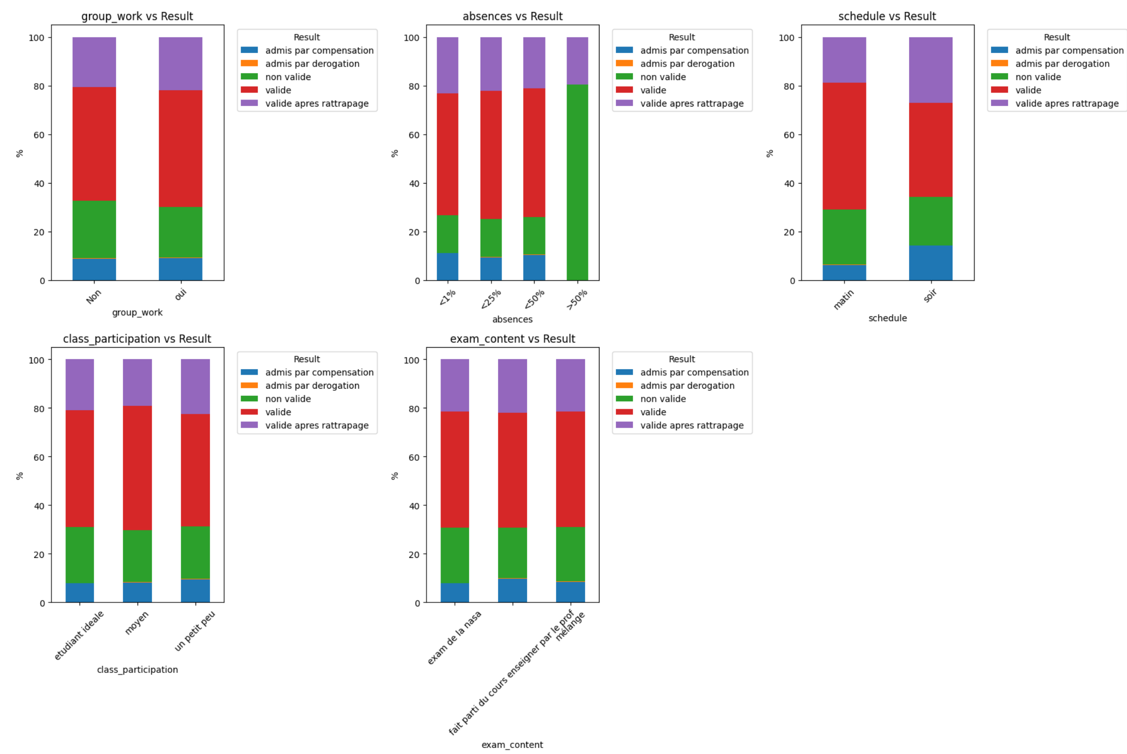


[203]:
```python
# List of numerical columns
numerical_cols = ['explanation_quality', 'final_grade', 'study_hours']

# Create boxplots for each numerical feature against 'result'
plt.figure(figsize=(20,6))

for idx, col in enumerate(numerical_cols):
    plt.subplot(1, 3, idx + 1)  # 1 row, 3 columns
    sns.boxplot(x='result', y=col, data=data)
    plt.title(f"{col} vs Result")
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



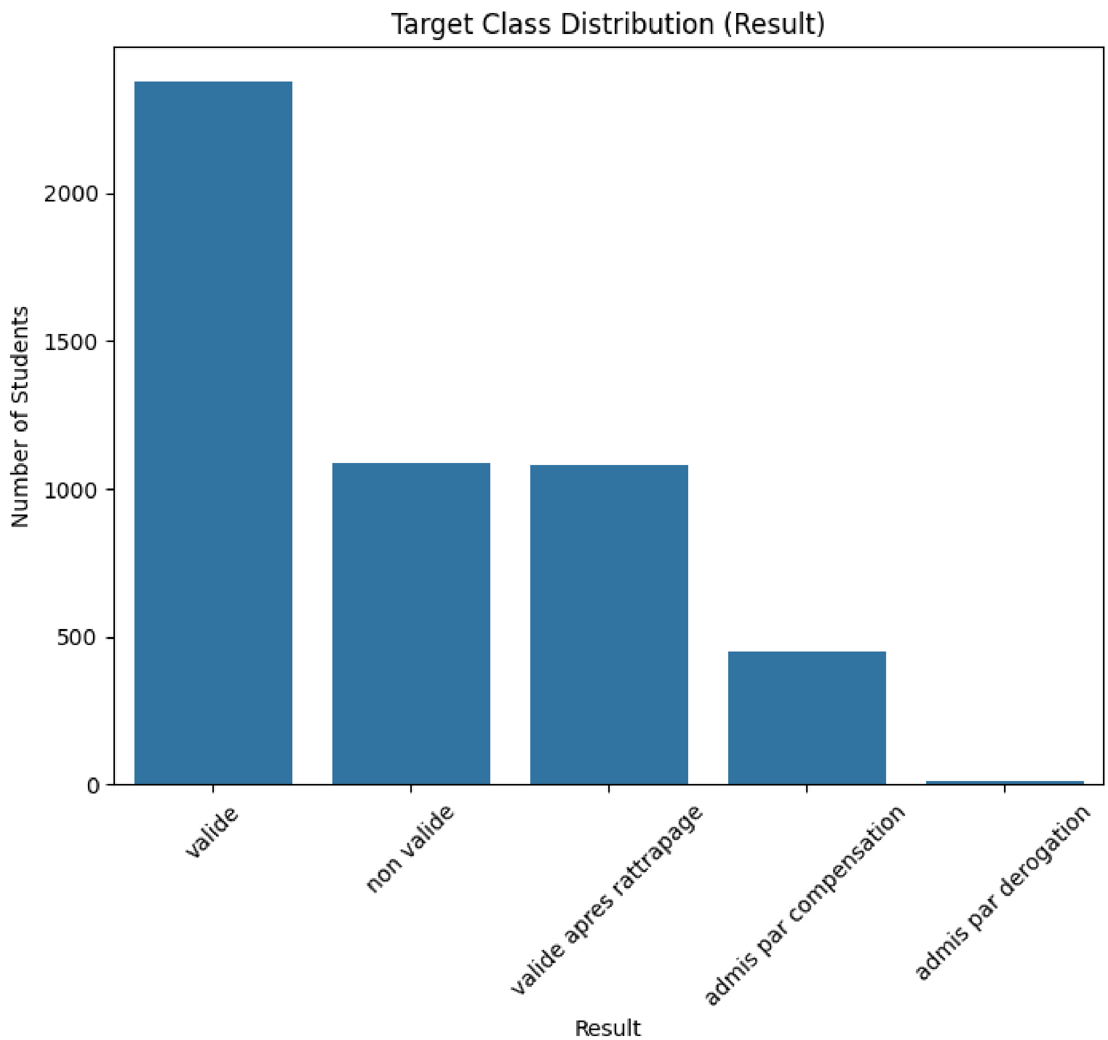
[204]:

```
# List of categorical features
categorical_cols = ['group_work', 'absences', 'schedule', 'class_participation', 'exam_content']

# Create subplots
plt.figure(figsize=(18, 12))

for idx, col in enumerate(categorical_cols):
    plt.subplot(2, 3, idx + 1)  # 2 rows, 3 columns
    crosstab = pd.crosstab(data[col], data['result'], normalize='index') * 100
    crosstab.plot(kind='bar', stacked=True, ax=plt.gca())
    plt.title(f"{col} vs Result")
    plt.ylabel('%')
    plt.xticks(rotation=45)
    plt.legend(title='Result', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()
```



[205]:

```python
# Countplot of result
plt.figure(figsize=(8,6))
sns.countplot(x='result', data=data, order=data['result'].value_counts().index)
plt.title('Target Class Distribution (Result)')
plt.xlabel('Result')
plt.ylabel('Number of Students')
plt.xticks(rotation=45)
plt.show()

# Print percentage distribution
print(data['result'].value_counts(normalize=True) * 100)
```




```
result
valide                     47.52
non valide                 21.68
valide apres rattrapage    21.66
admis par compensation      8.96
admis par derogation        0.18
Name: proportion, dtype: float64
```

```
[ ]:
```

3 cleaning the study hours column

[206]:
```
def clean_study_hours(value):
    if pd.isna(value):
        return None
    # Extract the first number found
    match = re.search(r'\d+', str(value))
    if match:
        return int(match.group())
    else:
        return None
```

[207]:
```
data['study_hours'] = data['study_hours'].apply(clean_study_hours)
```

[208]:
```
data.head(5000)
```

[208]:
```
      explanation_quality  final_grade  study_hours absences group_work  \
0                       5         9.46            4     <25%        oui
1                       9         5.13           21     <25%        Non
2                       7         9.67           18     <50%        Non
3                       6        13.24            0     <25%        oui
4                       3        12.28           48     <50%        oui
…                     …          …          …      …      …
4995                    8        16.37           30     <50%        Non
4996                    3        12.55           34     <50%        oui
4997                    5        10.69           30     >50%        oui
4998                    7        14.72           26     <25%        Non
4999                    6        15.33           22     <25%        oui

     schedule class_participation                               exam_content  \
0       matin        un petit peu                                    mélange
1       matin               moyen  fait parti du cours enseigner par le prof
2       matin        un petit peu  fait parti du cours enseigner par le prof
3       matin        un petit peu                                    mélange
4       matin        un petit peu  fait parti du cours enseigner par le prof
…        …                 …                                       …
4995     soir               moyen  fait parti du cours enseigner par le prof
4996    matin        un petit peu                                    mélange
4997    matin        un petit peu                          exam de la nasa
4998     soir        un petit peu                                    mélange
4999    matin        un petit peu                          exam de la nasa

                   result
0              non valide
1              non valide
2                  valide
3     valide apres rattrapage
```

```
4                        valide
…                          …
4995   admis par compensation
4996  valide apres rattrapage
4997               non valide
4998                   valide
4999                   valide

[5000 rows x 9 columns]
```

[209]:

```python
for column in data.columns:
    unique_values = data[column].unique()
    print(f"Column: {column}")
    print(f"Unique values ({len(unique_values)}): {unique_values}\n")
```

```
Column: explanation_quality
Unique values (10): [ 5  9  7  6  3  2  8  1  4 10]

Column: final_grade
Unique values (1106): [ 9.46  5.13  9.67 … 15.4   5.79  6.48]

Column: study_hours
Unique values (50): [ 4 21 18  0 48 29 44 20 27 37  2 26 15 30 35 25 23 32 17 22
31  3 34 24
 28 16 40 42 19 38 12 36 13 33 14  9  7  6 11  8 41 39  5  1 10 47 43 45
 50 46]

Column: absences
Unique values (4): ['<25%' '<50%' '<1%' '>50%']

Column: group_work
Unique values (2): ['oui' 'Non']

Column: schedule
Unique values (2): ['matin' 'soir']

Column: class_participation
Unique values (3): ['un petit peu' 'moyen' 'etudiant ideale']

Column: exam_content
Unique values (3): ['mélange' 'fait parti du cours enseigner par le prof' 'exam
de la nasa']

Column: result
Unique values (5): ['non valide' 'valide' 'valide apres rattrapage' 'admis par
compensation'
 'admis par derogation']
```

[210]:
```
rename_mapping = {
        'explanation_quality': 'Study Quality',
        'final_grade': 'Note',
        'result': 'Result',
        'study_hours': 'Autoformation',
        'absences': 'Absence',
        'group_work': 'Group Study',
        'exam_content': 'Alignement with Lecture',
        'schedule': 'Horaire',
        'class_participation': 'Class Engagement'
    }

    # Rename columns
structured_data = data.rename(columns=rename_mapping)
```

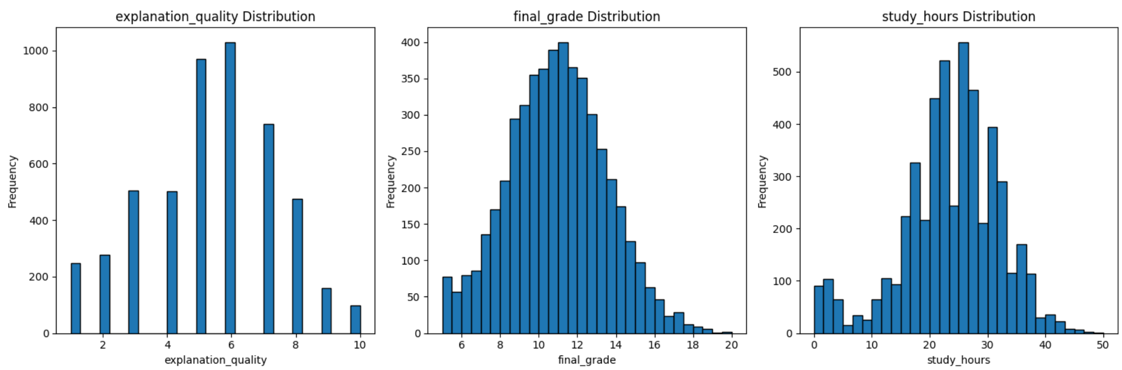
[211]:
```
structured_data.head()
```

[211]:
```
   Study Quality   Note  Autoformation Absence Group Study Horaire  \
0              5   9.46              4    <25%         oui   matin
1              9   5.13             21    <25%         Non   matin
2              7   9.67             18    <50%         Non   matin
3              6  13.24              0    <25%         oui   matin
4              3  12.28             48    <50%         oui   matin

  Class Engagement                    Alignement with Lecture  \
0     un petit peu                                    mélange
1            moyen  fait parti du cours enseigner par le prof
2     un petit peu  fait parti du cours enseigner par le prof
3     un petit peu                                    mélange
4     un petit peu  fait parti du cours enseigner par le prof

                  Result
0             non valide
1             non valide
2                 valide
3  valide apres rattrapage
4                 valide
```

4 plots

[212]:
```
# Select numerical columns
numerical_cols = ['explanation_quality', 'final_grade', 'study_hours']

#cast final grade column to float
data['final_grade'] = data['final_grade'].astype(float)
```

```python
# Create subplots
plt.figure(figsize=(15, 5))

for idx, col in enumerate(numerical_cols):
    plt.subplot(1, 3, idx + 1)  # 1 row, 3 columns, plot number idx+1
    plt.hist(data[col], bins=30, edgecolor='black')
    plt.title(f"{col} Distribution")
    plt.xlabel(col)
    plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```



5 feature engineering

why binary encoding ?

[213]:
```python
# 1. Encode group_work (yes/no --> 1/0)
def encode_group_work(df: pd.DataFrame) -> pd.DataFrame:
    df['Group Study'] = df['Group Study'].map({'oui': 1, 'Non': 0})
    return df
```

[214]:
```python
encode_group_work(structured_data)
```

[214]:
```
      Study Quality   Note  Autoformation Absence  Group Study Horaire  \
0                 5   9.46              4    <25%            1   matin
1                 9   5.13             21    <25%            0   matin
2                 7   9.67             18    <50%            0   matin
3                 6  13.24              0    <25%            1   matin
4                 3  12.28             48    <50%            1   matin
…               …     …             …       …             …      …
4995              8  16.37             30    <50%            0    soir
4996              3  12.55             34    <50%            1   matin
```

```
4997                5  10.69              30     >50%            1   matin
4998                7  14.72              26     <25%            0    soir
4999                6  15.33              22     <25%            1   matin

     Class Engagement                   Alignement with Lecture  \
0        un petit peu                                   mélange
1               moyen  fait parti du cours enseigner par le prof
2        un petit peu  fait parti du cours enseigner par le prof
3        un petit peu                                   mélange
4        un petit peu  fait parti du cours enseigner par le prof
…                 …                                       …
4995            moyen  fait parti du cours enseigner par le prof
4996     un petit peu                                   mélange
4997     un petit peu                           exam de la nasa
4998     un petit peu                                   mélange
4999     un petit peu                           exam de la nasa

                       Result
0                  non valide
1                  non valide
2                      valide
3     valide apres rattrapage
4                      valide
…                        …
4995   admis par compensation
4996  valide apres rattrapage
4997               non valide
4998                   valide
4999                   valide

[5000 rows x 9 columns]
```

why ordinal encoding

[215]:
```python
# 2. Encode absences (<1%, <25%, etc.) as ordered categories
def encode_absences(df: pd.DataFrame) -> pd.DataFrame:
    absence_order = ['<1%', '<25%', '<50%', '>50%']
    ord_encoder = OrdinalEncoder(categories=[absence_order])
    df['Absence'] = ord_encoder.fit_transform(df[['Absence']])
    return df
```

[216]:
```python
encode_absences(structured_data)
```

[216]:
```
      Study Quality   Note  Autoformation  Absence  Group Study Horaire  \
0                 5   9.46              4      1.0            1   matin
1                 9   5.13             21      1.0            0   matin
2                 7   9.67             18      2.0            0   matin
3                 6  13.24              0      1.0            1   matin
```

```
4                3  12.28            48      2.0            1   matin
…             …    …           …     …          …      …
4995             8  16.37            30      2.0            0    soir
4996             3  12.55            34      2.0            1   matin
4997             5  10.69            30      3.0            1   matin
4998             7  14.72            26      1.0            0    soir
4999             6  15.33            22      1.0            1   matin

     Class Engagement                    Alignement with Lecture  \
0        un petit peu                                     mélange
1               moyen  fait parti du cours enseigner par le prof
2        un petit peu  fait parti du cours enseigner par le prof
3        un petit peu                                     mélange
4        un petit peu  fait parti du cours enseigner par le prof
…                 …                                          …
4995            moyen  fait parti du cours enseigner par le prof
4996     un petit peu                                     mélange
4997     un petit peu                             exam de la nasa
4998     un petit peu                                     mélange
4999     un petit peu                             exam de la nasa

                        Result
0                   non valide
1                   non valide
2                       valide
3     valide apres rattrapage
4                       valide
…                          …
4995   admis par compensation
4996  valide apres rattrapage
4997               non valide
4998                   valide
4999                   valide

[5000 rows x 9 columns]
```

why binary encoding

[217]:
```python
# 3. Encode schedule (matin/soir) as binary 0/1
def encode_schedule(df: pd.DataFrame) -> pd.DataFrame:
    df['Horaire'] = df['Horaire'].map({'matin': 0, 'soir': 1})
    return df
```

[218]:
```python
encode_schedule(structured_data)
```

[218]:
```
      Study Quality   Note  Autoformation  Absence  Group Study  Horaire  \
0                 5   9.46              4      1.0            1        0
1                 9   5.13             21      1.0            0        0
```

```
2                  7   9.67             18       2.0             0         0
3                  6  13.24              0       1.0             1         0
4                  3  12.28             48       2.0             1         0
…             …    …             …       …          …       …
4995               8  16.37             30       2.0             0         1
4996               3  12.55             34       2.0             1         0
4997               5  10.69             30       3.0             1         0
4998               7  14.72             26       1.0             0         1
4999               6  15.33             22       1.0             1         0

     Class Engagement                     Alignement with Lecture  \
0        un petit peu                                     mélange
1               moyen  fait parti du cours enseigner par le prof
2        un petit peu  fait parti du cours enseigner par le prof
3        un petit peu                                     mélange
4        un petit peu  fait parti du cours enseigner par le prof
…                 …                                         …
4995            moyen  fait parti du cours enseigner par le prof
4996     un petit peu                                     mélange
4997     un petit peu                               exam de la nasa
4998     un petit peu                                     mélange
4999     un petit peu                               exam de la nasa

                      Result
0                 non valide
1                 non valide
2                     valide
3     valide apres rattrapage
4                     valide
…                          …
4995   admis par compensation
4996  valide apres rattrapage
4997              non valide
4998                  valide
4999                  valide

[5000 rows x 9 columns]
```

why ordinal encoding

[219]:
```
# 4. Encode class participation (un petit peu < moyen < etudiant ideale)
def encode_class_participation(df: pd.DataFrame) -> pd.DataFrame:
    participation_order = ['un petit peu', 'moyen', 'etudiant ideale']
    ord_encoder = OrdinalEncoder(categories=[participation_order])
    df['Class Engagement'] = ord_encoder.fit_transform(df[['Class Engagement']])
    return df
```

[220]:
```
encode_class_participation(structured_data)
```

[220]:

```
      Study Quality   Note  Autoformation  Absence  Group Study  Horaire  \
0                 5   9.46              4      1.0            1        0
1                 9   5.13             21      1.0            0        0
2                 7   9.67             18      2.0            0        0
3                 6  13.24              0      1.0            1        0
4                 3  12.28             48      2.0            1        0
…               …    …          …    …       …       …
4995              8  16.37             30      2.0            0        1
4996              3  12.55             34      2.0            1        0
4997              5  10.69             30      3.0            1        0
4998              7  14.72             26      1.0            0        1
4999              6  15.33             22      1.0            1        0

      Class Engagement                     Alignement with Lecture  \
0                  0.0                                     mélange
1                  1.0  fait parti du cours enseigner par le prof
2                  0.0  fait parti du cours enseigner par le prof
3                  0.0                                     mélange
4                  0.0  fait parti du cours enseigner par le prof
…                 …                                         …
4995               1.0  fait parti du cours enseigner par le prof
4996               0.0                                     mélange
4997               0.0                             exam de la nasa
4998               0.0                                     mélange
4999               0.0                             exam de la nasa

                          Result
0                     non valide
1                     non valide
2                         valide
3      valide apres rattrapage
4                         valide
…                          …
4995   admis par compensation
4996  valide apres rattrapage
4997                  non valide
4998                      valide
4999                      valide

[5000 rows x 9 columns]
```

why ordinal encoding

[221]:

```python
def onehot_encode_exam_content(df: pd.DataFrame) -> pd.DataFrame:
    # Ensure clean text (remove extra spaces)
    if df['Alignement with Lecture'].dtype == 'object':
        df['Alignement with Lecture'] = df['Alignement with Lecture'].str.
 ↪strip()
```

```
    # Perform One-Hot Encoding
    onehot = pd.get_dummies(
        df['Alignement with Lecture'],
        prefix='Alignement',
        dtype=int,  # ensures 0/1 integers not booleans,
        drop_first=True  # to avoid dummy variable trap
    )

    # Drop original and merge one-hot columns
    df = df.drop(columns=['Alignement with Lecture'])
    df = pd.concat([df, onehot], axis=1)

    return df
```

[222]:
```
structured_data = onehot_encode_exam_content(structured_data)
```

[223]:
```
structured_data.head(5000)
```

[223]:
```
      Study Quality   Note  Autoformation  Absence  Group Study  Horaire  \
0                 5   9.46              4      1.0            1        0
1                 9   5.13             21      1.0            0        0
2                 7   9.67             18      2.0            0        0
3                 6  13.24              0      1.0            1        0
4                 3  12.28             48      2.0            1        0
…               …    …             …      …           …       …
4995              8  16.37             30      2.0            0        1
4996              3  12.55             34      2.0            1        0
4997              5  10.69             30      3.0            1        0
4998              7  14.72             26      1.0            0        1
4999              6  15.33             22      1.0            1        0

      Class Engagement                   Result  \
0                  0.0               non valide
1                  1.0               non valide
2                  0.0                   valide
3                  0.0  valide apres rattrapage
4                  0.0                   valide
…                  …                     …
4995               1.0   admis par compensation
4996               0.0  valide apres rattrapage
4997               0.0               non valide
4998               0.0                   valide
4999               0.0                   valide

      Alignement_fait parti du cours enseigner par le prof  Alignement_mélange
0                                                     0                   1
```

```
1                                                                   1                       0
2                                                                   1                       0
3                                                                   0                       1
4                                                                   1                       0
…                                                               …                       …
4995                                                                1                       0
4996                                                                0                       1
4997                                                                0                       0
4998                                                                0                       1
4999                                                                0                       0

[5000 rows x 10 columns]
```

[224]:
```python
def encode_target(df: pd.DataFrame) -> pd.DataFrame:
    label_encoder = LabelEncoder()
    df['Result'] = label_encoder.fit_transform(df['Result'])
    return df
```

[225]:
```python
encode_target(structured_data)
```

[225]:
```
      Study Quality   Note  Autoformation  Absence  Group Study  Horaire  \
0                 5   9.46              4      1.0            1        0
1                 9   5.13             21      1.0            0        0
2                 7   9.67             18      2.0            0        0
3                 6  13.24              0      1.0            1        0
4                 3  12.28             48      2.0            1        0
…               …    …               …      …          …        …
4995              8  16.37             30      2.0            0        1
4996              3  12.55             34      2.0            1        0
4997              5  10.69             30      3.0            1        0
4998              7  14.72             26      1.0            0        1
4999              6  15.33             22      1.0            1        0

      Class Engagement  Result  \
0                  0.0       2
1                  1.0       2
2                  0.0       3
3                  0.0       4
4                  0.0       3
…                  …     …
4995               1.0       0
4996               0.0       4
4997               0.0       2
4998               0.0       3
4999               0.0       3

      Alignement_fait parti du cours enseigner par le prof  Alignement_mélange
```

```
0                                                          0        1
1                                                          1        0
2                                                          1        0
3                                                          0        1
4                                                          1        0
…                                                      …        …
4995                                                       1        0
4996                                                       0        1
4997                                                       0        0
4998                                                       0        1
4999                                                       0        0

[5000 rows x 10 columns]
```

6 Separate features and target

[226]:
```
X = structured_data.drop(columns=['Result'])  # Features (everything except the␣
↪target)
y = structured_data['Result']  # Target variable
```

[227]:
```
print(y)
```

```
0       2
1       2
2       3
3       4
4       3
       ..
4995    0
4996    4
4997    2
4998    3
4999    3
Name: Result, Length: 5000, dtype: int64
```

7 Split X and y into training and testing sets

[228]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,       # 30% test, 70% train
    random_state=42,     # random state for reproducibility
    stratify=y           # keep the same distribution of result classes
)

# Let's check the shapes
print("Training set:", X_train.shape, y_train.shape)
```

```
print("Testing set:", X_test.shape, y_test.shape)
```

```
Training set: (3500, 9) (3500,)
Testing set: (1500, 9) (1500,)
```

[229]:
```
try:
    results
except NameError:
    results = []

print("="*50)
print("Training: Logistic Regression")
print("="*50)

start_train = time.time()
log_reg = LogisticRegression(max_iter=1000)
param_grid_log_reg = {
    'C': [0.01, 0.1, 1, 10],
    'solver': ['lbfgs', 'liblinear']
}
grid_log = GridSearchCV(log_reg, param_grid_log_reg, cv=3, n_jobs=-1,␣
 ↪scoring='accuracy')
grid_log.fit(X_train, y_train)
best_log = grid_log.best_estimator_
train_duration = round(time.time() - start_train, 2)

start_pred = time.time()
y_pred_log = best_log.predict(X_test)
pred_duration = round(time.time() - start_pred, 2)

acc_log = accuracy_score(y_test, y_pred_log)
print(f"Best Params: {grid_log.best_params_}")
print(f"Accuracy: {acc_log:.4f}")
print(classification_report(y_test, y_pred_log, zero_division=0))

results.append({
    'Model': 'Logistic Regression',
    'Best Params': grid_log.best_params_,
    'Accuracy': acc_log,
    'Training Time (s)': train_duration,
    'Prediction Time (s)': pred_duration
})
```

```
==================================================
Training: Logistic Regression
==================================================
Best Params: {'C': 0.01, 'solver': 'lbfgs'}
Accuracy: 0.5727
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.59      0.62      0.61       325
           3       0.57      0.92      0.70       713
           4       0.00      0.00      0.00       325

    accuracy                           0.57      1500
   macro avg       0.23      0.31      0.26      1500
weighted avg       0.40      0.57      0.47      1500
```

```
c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:465: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[230]:

```
print("="*50)
print("Training: KNN")
print("="*50)

start_train = time.time()
knn = KNeighborsClassifier()
param_grid_knn = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance']
}
grid_knn = GridSearchCV(knn, param_grid_knn, cv=3, n_jobs=-1,
  ↪scoring='accuracy')
grid_knn.fit(X_train, y_train)
best_knn = grid_knn.best_estimator_
train_duration = round(time.time() - start_train, 2)

start_pred = time.time()
y_pred_knn = best_knn.predict(X_test)
pred_duration = round(time.time() - start_pred, 2)

acc_knn = accuracy_score(y_test, y_pred_knn)
print(f"Best Params: {grid_knn.best_params_}")
print(f"Accuracy: {acc_knn:.4f}")
```

```
print(classification_report(y_test, y_pred_knn, zero_division=0))

results.append({
    'Model': 'KNN',
    'Best Params': grid_knn.best_params_,
    'Accuracy': acc_knn,
    'Training Time (s)': train_duration,
    'Prediction Time (s)': pred_duration
})
```

```
==================================================
Training: KNN
==================================================
Best Params: {'n_neighbors': 7, 'weights': 'uniform'}
Accuracy: 0.6093
              precision    recall  f1-score   support

           0       0.07      0.02      0.03       134
           1       0.00      0.00      0.00         3
           2       0.76      0.77      0.76       325
           3       0.62      0.88      0.73       713
           4       0.29      0.10      0.15       325

    accuracy                           0.61      1500
   macro avg       0.35      0.35      0.33      1500
weighted avg       0.53      0.61      0.55      1500
```

[231]:
```
print("="*50)
print("Training: SVM")
print("="*50)

start_train = time.time()
svm = SVC()
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],  # Added 'poly' for polynomial kernel
    'degree': [2, 3, 4]  # Degree of the polynomial kernel
}
grid_svm = GridSearchCV(svm, param_grid_svm, cv=3, n_jobs=-1,␣
 ↪scoring='accuracy')
grid_svm.fit(X_train, y_train)
best_svm = grid_svm.best_estimator_
train_duration = round(time.time() - start_train, 2)

start_pred = time.time()
y_pred_svm = best_svm.predict(X_test)
```

```
pred_duration = round(time.time() - start_pred, 2)

acc_svm = accuracy_score(y_test, y_pred_svm)
print(f"Best Params: {grid_svm.best_params_}")
print(f"Accuracy: {acc_svm:.4f}")
print(classification_report(y_test, y_pred_svm, zero_division=0))

results.append({
    'Model': 'SVM',
    'Best Params': grid_svm.best_params_,
    'Accuracy': acc_svm,
    'Training Time (s)': train_duration,
    'Prediction Time (s)': pred_duration
})
```

```
==================================================
Training: SVM
==================================================
Best Params: {'C': 10, 'degree': 2, 'kernel': 'rbf'}
Accuracy: 0.6320
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.68      0.83      0.75       325
           3       0.62      0.95      0.75       713
           4       0.00      0.00      0.00       325

    accuracy                           0.63      1500
   macro avg       0.26      0.36      0.30      1500
weighted avg       0.44      0.63      0.52      1500
```

[232]:
```
print("="*50)
print("Training: Decision Tree")
print("="*50)

# 1. Training + Grid Search
start_train = time.time()
tree = DecisionTreeClassifier()
param_grid_tree = {
    'max_depth': [1,2,3,4,5, 10, 20,50],
    'criterion': ['gini', 'entropy']
}
grid_tree = GridSearchCV(tree, param_grid_tree, cv=3, n_jobs=-1,␣
 ↪scoring='accuracy')
grid_tree.fit(X_train, y_train)
```

```python
best_tree = grid_tree.best_estimator_
train_duration = round(time.time() - start_train, 2)

# 2. Prediction
start_pred = time.time()
y_pred_tree = best_tree.predict(X_test)
pred_duration = round(time.time() - start_pred, 2)

# 3. Evaluation
acc_tree = accuracy_score(y_test, y_pred_tree)
print(f"Best Params: {grid_tree.best_params_}")
print(f"Accuracy: {acc_tree:.4f}")
print(classification_report(y_test, y_pred_tree, zero_division=0))

# 4. Save results
results.append({
    'Model': 'Decision Tree',
    'Best Params': grid_tree.best_params_,
    'Accuracy': acc_tree,
    'Training Time (s)': train_duration,
    'Prediction Time (s)': pred_duration
})
```

```
==================================================
Training: Decision Tree
==================================================
Best Params: {'criterion': 'gini', 'max_depth': 3}
Accuracy: 0.6853
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.80      0.97      0.88       325
           3       0.65      1.00      0.78       713
           4       0.00      0.00      0.00       325

    accuracy                           0.69      1500
   macro avg       0.29      0.39      0.33      1500
weighted avg       0.48      0.69      0.56      1500
```

[233]:
```python
print("="*50)
print("Training: Random Forest")
print("="*50)

# 1. Training + Grid Search
start_train = time.time()
```

```
rf = RandomForestClassifier()
param_grid_rf = {
    'n_estimators': [50,100, 200,300],
    'max_depth': [1,2,3,4,5,20,50,100],
}
grid_rf = GridSearchCV(rf, param_grid_rf, cv=3, n_jobs=-1, scoring='accuracy')
grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_
train_duration = round(time.time() - start_train, 2)

# 2. Prediction
start_pred = time.time()
y_pred_rf = best_rf.predict(X_test)
pred_duration = round(time.time() - start_pred, 2)

# 3. Evaluation
acc_rf = accuracy_score(y_test, y_pred_rf)
print(f"Best Params: {grid_rf.best_params_}")
print(f"Accuracy: {acc_rf:.4f}")
print(classification_report(y_test, y_pred_rf, zero_division=0))

# 4. Save results
results.append({
    'Model': 'Random Forest',
    'Best Params': grid_rf.best_params_,
    'Accuracy': acc_rf,
    'Training Time (s)': train_duration,
    'Prediction Time (s)': pred_duration
})
```

```
==================================================
Training: Random Forest
==================================================
Best Params: {'max_depth': 5, 'n_estimators': 50}
Accuracy: 0.6853
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.80      0.97      0.88       325
           3       0.65      1.00      0.78       713
           4       0.00      0.00      0.00       325

    accuracy                           0.69      1500
   macro avg       0.29      0.39      0.33      1500
weighted avg       0.48      0.69      0.56      1500
```

[234]:
```python
print("="*50)
print("Training: LightGBM")
print("="*50)

# 1. Training + Grid Search
start_train = time.time()
lgbm = LGBMClassifier()
param_grid_lgbm = {
    'n_estimators': [50,100, 200,300],
    'learning_rate': [0.05, 0.1]
}
grid_lgbm = GridSearchCV(lgbm, param_grid_lgbm, cv=3, n_jobs=-1,␣
 ↪scoring='accuracy')
grid_lgbm.fit(X_train, y_train)
best_lgbm = grid_lgbm.best_estimator_
train_duration = round(time.time() - start_train, 2)

# 2. Prediction
start_pred = time.time()
y_pred_lgbm = best_lgbm.predict(X_test)
pred_duration = round(time.time() - start_pred, 2)

# 3. Evaluation
acc_lgbm = accuracy_score(y_test, y_pred_lgbm)
print(f"Best Params: {grid_lgbm.best_params_}")
print(f"Accuracy: {acc_lgbm:.4f}")
print(classification_report(y_test, y_pred_lgbm, zero_division=0))

# 4. Save results
results.append({
    'Model': 'LightGBM',
    'Best Params': grid_lgbm.best_params_,
    'Accuracy': acc_lgbm,
    'Training Time (s)': train_duration,
    'Prediction Time (s)': pred_duration
})
```

```
==================================================
Training: LightGBM
==================================================

c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_tags.py:354: FutureWarning: The LGBMClassifier or
classes from which it inherits use `_get_tags` and `_more_tags`. Please define
the `__sklearn_tags__` method, or inherit from `sklearn.base.BaseEstimator`
and/or other appropriate mixins such as `sklearn.base.TransformerMixin`,
`sklearn.base.ClassifierMixin`, `sklearn.base.RegressorMixin`, and
`sklearn.base.OutlierMixin`. From scikit-learn 1.7, not defining
```

```
`__sklearn_tags__` will raise an error.
  warnings.warn(
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000243 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 328
[LightGBM] [Info] Number of data points in the train set: 3500, number of used
features: 9
[LightGBM] [Info] Start training from score -2.411125
[LightGBM] [Info] Start training from score -6.368759
[LightGBM] [Info] Start training from score -1.528516
[LightGBM] [Info] Start training from score -0.744140
[LightGBM] [Info] Start training from score -1.529835
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Best Params: {'learning_rate': 0.05, 'n_estimators': 50}
Accuracy: 0.6840
              precision    recall  f1-score   support

           0       0.25      0.01      0.01       134
           1       0.00      0.00      0.00         3
           2       0.80      0.97      0.88       325
           3       0.65      0.98      0.78       713
           4       0.35      0.03      0.05       325

    accuracy                           0.68      1500
   macro avg       0.41      0.40      0.35      1500
weighted avg       0.58      0.68      0.57      1500
```

[235]:
```python
# Create a DataFrame from the collected results
results_df = pd.DataFrame(results)

# Sort the models by their Accuracy (optional)
results_df = results_df.sort_values(by='Accuracy', ascending=False)

# Display the table
print("\n\nFinal Model Performance Summary:\n")
print(results_df.to_string(index=False))
```

```
Final Model Performance Summary:

              Model                                  Best Params  Accuracy
Training Time (s)  Prediction Time (s)
      Random Forest         {'max_depth': 3, 'n_estimators': 50}  0.686000
```

```
26.51                 0.01
      Random Forest         {'max_depth': 5, 'n_estimators': 50}  0.685333
25.28                 0.01
      Random Forest         {'max_depth': 5, 'n_estimators': 50}  0.685333
20.39                 0.02
      Decision Tree        {'criterion': 'gini', 'max_depth': 3}  0.685333
0.47                 0.00
      Decision Tree        {'criterion': 'gini', 'max_depth': 3}  0.685333
0.45                 0.00
      Random Forest       {'max_depth': 10, 'n_estimators': 200}  0.684000
6.24                 0.05
           LightGBM  {'learning_rate': 0.05, 'n_estimators': 50}  0.684000
14.71                 0.02
           LightGBM  {'learning_rate': 0.05, 'n_estimators': 50}  0.684000
22.15                 0.02
      Random Forest       {'max_depth': 10, 'n_estimators': 300}  0.683333
18.33                 0.11
      Random Forest       {'max_depth': 10, 'n_estimators': 200}  0.683333
5.72                 0.05
      Random Forest       {'max_depth': 10, 'n_estimators': 200}  0.683333
5.65                 0.05
      Decision Tree     {'criterion': 'entropy', 'max_depth': 5}  0.682000
0.29                 0.00
      Decision Tree     {'criterion': 'entropy', 'max_depth': 5}  0.682000
0.27                 0.00
      Decision Tree     {'criterion': 'entropy', 'max_depth': 5}  0.682000
0.30                 0.00
      Decision Tree     {'criterion': 'entropy', 'max_depth': 5}  0.682000
0.26                 0.00
           LightGBM {'learning_rate': 0.05, 'n_estimators': 100}  0.668000
17.00                 0.04
           LightGBM {'learning_rate': 0.05, 'n_estimators': 100}  0.668000
10.76                 0.04
           LightGBM {'learning_rate': 0.05, 'n_estimators': 100}  0.668000
28.19                 0.04
      Random Forest       {'max_depth': 20, 'n_estimators': 300}  0.663333
21.81                 0.15
                SVM                   {'C': 10, 'kernel': 'rbf'}  0.632000
201.91                 0.52
                SVM                   {'C': 10, 'kernel': 'rbf'}  0.632000
196.73                 0.56
                SVM     {'C': 10, 'degree': 2, 'kernel': 'rbf'}  0.632000
386.40                 0.71
                SVM                   {'C': 10, 'kernel': 'rbf'}  0.632000
140.96                 0.43
                SVM                   {'C': 10, 'kernel': 'rbf'}  0.632000
142.60                 0.48
                SVM     {'C': 10, 'degree': 2, 'kernel': 'rbf'}  0.632000
```

```
394.87                  0.50
                KNN     {'n_neighbors': 7, 'weights': 'uniform'}  0.609333
0.68                  0.17
                KNN     {'n_neighbors': 7, 'weights': 'uniform'}  0.609333
1.08                  0.14
                KNN     {'n_neighbors': 7, 'weights': 'uniform'}  0.609333
1.50                  0.27
                KNN     {'n_neighbors': 7, 'weights': 'uniform'}  0.609333
0.69                  0.56
                KNN     {'n_neighbors': 7, 'weights': 'uniform'}  0.609333
1.00                  0.15
                KNN     {'n_neighbors': 7, 'weights': 'uniform'}  0.609333
1.27                  0.16
                KNN     {'n_neighbors': 7, 'weights': 'uniform'}  0.609333
0.71                  0.08
Logistic Regression               {'C': 0.01, 'solver': 'lbfgs'}  0.572667
14.34                  0.00
Logistic Regression               {'C': 0.01, 'solver': 'lbfgs'}  0.572667
5.38                  0.00
Logistic Regression               {'C': 0.01, 'solver': 'lbfgs'}  0.572667
9.41                  0.00
Logistic Regression               {'C': 0.01, 'solver': 'lbfgs'}  0.572667
9.76                  0.00
Logistic Regression               {'C': 0.01, 'solver': 'lbfgs'}  0.572667
6.75                  0.00
Logistic Regression               {'C': 0.01, 'solver': 'lbfgs'}  0.572667
12.32                  0.01
Logistic Regression               {'C': 0.01, 'solver': 'lbfgs'}  0.572667
7.71                  0.01
```

[238]:

```python
import time
import pandas as pd
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, classification_report

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier

# Initialize results
results = []
```

```python
# Define models and parameters
models_and_params = {
    'Logistic Regression': (LogisticRegression(max_iter=1000), {
        'C': [0.01, 0.1, 1, 10, 50],
        'solver': ['lbfgs', 'liblinear']
    }),
    'KNN': (KNeighborsClassifier(), {
        'n_neighbors': list(range(3, 11)),
        'weights': ['uniform', 'distance']
    }),
    'SVM': (SVC(), {
        'C': [0.1, 1, 10, 100],
        'kernel': ['linear', 'rbf', 'poly'],
        'degree': [2, 3, 4]
    }),
    'Decision Tree': (DecisionTreeClassifier(), {
        'max_depth': [3, 5, 10, 20],
        'criterion': ['gini', 'entropy']
    }),
    'Random Forest': (RandomForestClassifier(), {
        'n_estimators': [50, 100, 200, 300],
        'max_depth': [5, 10, 20, 30]
    }),
    'LightGBM': (LGBMClassifier(), {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.05, 0.1]
    }),


}

# Train models
for model_name, (model, param_dist) in models_and_params.items():
    print("="*50)
    print(f"Training: {model_name}")
    print("="*50)

    # 1. Train
    start_train = time.time()
    random_search = RandomizedSearchCV(
        model,
        param_distributions=param_dist,
        n_iter=20,           # Try 10 random combinations
        cv=3,
        n_jobs=-1,
        scoring='accuracy',
        random_state=42
```

```
    )
    random_search.fit(X_train, y_train)
    best_model = random_search.best_estimator_
    train_duration = round(time.time() - start_train, 2)

    # 2. Predict
    start_pred = time.time()
    y_pred = best_model.predict(X_test)
    pred_duration = round(time.time() - start_pred, 2)

    # 3. Evaluate
    acc = accuracy_score(y_test, y_pred)
    print(f"Best Params: {random_search.best_params_}")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred, zero_division=0))

    # 4. Save results
    results.append({
        'Model': model_name,
        'Best Params': random_search.best_params_,
        'Accuracy': acc,
        'Training Time (s)': train_duration,
        'Prediction Time (s)': pred_duration
    })

# Final Table
results_df = pd.DataFrame(results)
print("\n\nFinal Model Performance Summary:")
print(results_df.sort_values(by='Accuracy', ascending=False))
```

```
==================================================
Training: Logistic Regression
==================================================

c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:317: UserWarning: The total space of
parameters 10 is smaller than n_iter=20. Running 10 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:465: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:317: UserWarning: The total space of
parameters 16 is smaller than n_iter=20. Running 16 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(

Best Params: {'solver': 'lbfgs', 'C': 0.01}
Accuracy: 0.5727
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.59      0.62      0.61       325
           3       0.57      0.92      0.70       713
           4       0.00      0.00      0.00       325

    accuracy                           0.57      1500
   macro avg       0.23      0.31      0.26      1500
weighted avg       0.40      0.57      0.47      1500

==================================================
Training: KNN
==================================================
Best Params: {'weights': 'uniform', 'n_neighbors': 10}
Accuracy: 0.6113
              precision    recall  f1-score   support

           0       0.03      0.01      0.01       134
           1       0.00      0.00      0.00         3
           2       0.78      0.74      0.76       325
           3       0.60      0.91      0.73       713
           4       0.28      0.07      0.11       325

    accuracy                           0.61      1500
   macro avg       0.34      0.35      0.32      1500
weighted avg       0.52      0.61      0.54      1500

==================================================
Training: SVM
==================================================
Best Params: {'kernel': 'rbf', 'degree': 3, 'C': 100}
Accuracy: 0.6553
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
```

```
           2       0.72      0.90      0.80       325
           3       0.63      0.97      0.77       713
           4       0.00      0.00      0.00       325

    accuracy                           0.66      1500
   macro avg       0.27      0.37      0.31      1500
weighted avg       0.46      0.66      0.54      1500

==================================================
Training: Decision Tree
==================================================
Best Params: {'max_depth': 3, 'criterion': 'gini'}
Accuracy: 0.6853
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.80      0.97      0.88       325
           3       0.65      1.00      0.78       713
           4       0.00      0.00      0.00       325

    accuracy                           0.69      1500
   macro avg       0.29      0.39      0.33      1500
weighted avg       0.48      0.69      0.56      1500

==================================================
Training: Random Forest
==================================================

c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:317: UserWarning: The total space of
parameters 8 is smaller than n_iter=20. Running 8 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:317: UserWarning: The total space of
parameters 16 is smaller than n_iter=20. Running 16 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(

Best Params: {'n_estimators': 50, 'max_depth': 5}
Accuracy: 0.6853
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.80      0.97      0.88       325
           3       0.65      1.00      0.78       713
           4       0.00      0.00      0.00       325
```

```
    accuracy                           0.69      1500
   macro avg       0.29      0.39      0.33      1500
weighted avg       0.48      0.69      0.56      1500

==================================================
Training: LightGBM
==================================================

c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_tags.py:354: FutureWarning: The LGBMClassifier or
classes from which it inherits use `_get_tags` and `_more_tags`. Please define
the `__sklearn_tags__` method, or inherit from `sklearn.base.BaseEstimator`
and/or other appropriate mixins such as `sklearn.base.TransformerMixin`,
`sklearn.base.ClassifierMixin`, `sklearn.base.RegressorMixin`, and
`sklearn.base.OutlierMixin`. From scikit-learn 1.7, not defining
`__sklearn_tags__` will raise an error.
  warnings.warn(
c:\Users\admin\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:317: UserWarning: The total space of
parameters 9 is smaller than n_iter=20. Running 9 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(

[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000217 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 328
[LightGBM] [Info] Number of data points in the train set: 3500, number of used
features: 9
[LightGBM] [Info] Start training from score -2.411125
[LightGBM] [Info] Start training from score -6.368759
[LightGBM] [Info] Start training from score -1.528516
[LightGBM] [Info] Start training from score -0.744140
[LightGBM] [Info] Start training from score -1.529835
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Best Params: {'n_estimators': 50, 'learning_rate': 0.01}
Accuracy: 0.6853
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       134
           1       0.00      0.00      0.00         3
           2       0.80      0.97      0.88       325
           3       0.64      1.00      0.78       713
           4       0.00      0.00      0.00       325

    accuracy                           0.69      1500
```

```
   macro avg       0.29      0.39      0.33      1500
weighted avg       0.48      0.69      0.56      1500


Final Model Performance Summary:
                 Model                                  Best Params  Accuracy  \
4        Random Forest        {'n_estimators': 50, 'max_depth': 5}  0.685333
3        Decision Tree       {'max_depth': 3, 'criterion': 'gini'}  0.685333
5             LightGBM  {'n_estimators': 50, 'learning_rate': 0.01}  0.685333
2                  SVM    {'kernel': 'rbf', 'degree': 3, 'C': 100}  0.655333
1                  KNN   {'weights': 'uniform', 'n_neighbors': 10}  0.611333
0  Logistic Regression              {'solver': 'lbfgs', 'C': 0.01}  0.572667

   Training Time (s)  Prediction Time (s)
4              16.75                 0.01
3               0.19                 0.00
5              13.22                 0.04
2             440.98                 0.68
1               2.44                 0.26
0               4.60                 0.00
```