Cheat Sheet — MLT End Semester

Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, mean_squared_error
```

Loading Data

```
train = pd.read_csv('/kaggle/input/your-dataset/train.csv')
test = pd.read_csv('/kaggle/input/your-dataset/test.csv')

# Will probably be given in the first block during the kaggle competition
```

Preprocessing

Description [Only for your inference of the dataset]

```
# 1. General info (structure, nulls, data types)
print("General Information:")
print(df.info())
print("\n")

# 2. Descriptive statistics (for numerical columns)
print("Descriptive Statistics (Numerical Columns):")
print(df.describe())
print("\n")

# 3. Descriptive statistics for categorical data
```

```
print("Descriptive Statistics (Categorical Columns):")
print(df.describe(include=['object', 'category']))
print("\n")

# 4. Checking for missing values
print("Missing Values in Each Column:")
print(df.isnull().sum())
print("\n")

# 5. Value counts for each categorical feature
# (useful for understanding the distribution)
print("Value Counts for Categorical Features:")
for column in df.select_dtypes(include=['object', 'category']).columns:
    print(f"{column} Value Counts:")
    print(df[column].value_counts())
    print("\n")

# 6. Correlation matrix (for numerical columns)
print("Correlation Matrix:")
print(df.corr())
```

Dropping columns

```
train.drop(['id'], axis=1, inplace=True)
test_ids = test['id']
test.drop(['id'], axis=1, inplace=True)
```

Encoding

```
le = LabelEncoder()
train['feature'] = le.fit_transform(train['feature'])

# could also be OneHotEncoder() or OrdinalEncoder()
```

Scaling

```
scaler = StandardScaler()
X_train = scaler.fit_transform(train.drop('target', axis=1))
y_train = train['target']
X_test = scaler.transform(test)

# could also be MinMaxScaler() or RobustScaler()
```

Outlier Detection

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)
print("Number of outliers:", outliers.sum())

# or

import numpy as np
import pandas as pd
from scipy.stats import zscore

z_scores = np.abs(zscore(df))
threshold = 3
outliers = (z_scores > threshold).any(axis=1)

print("Number of outliers:", outliers.sum())
df_outliers = df[outliers]
```

Model - Specific

Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

- Use → Regression tasks with linear relationships
- Metric → `mean_squared_error(y_test, y_pred)`

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)
pred = nb.predict(X_test)
```

- Use → Categorical features / text data
- Metric → Classification accuracy

Gaussian Mixture Model

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(X_train)
labels = gmm.predict(X_test)
```

- Use → Probabilistic clustering (when clusters overlap)
- Compare with K-Means on same data

K-Means Clustering

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(X_train)
labels = model.labels_
```

- Use → Unsupervised problems
- Metric → Silhouette Score

Support Vector Machine

```
from sklearn.svm import SVC
model = SVC(kernel='rbf')
 # Try 'linear' if dataset is linearly separable
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

- Use:
  - Classification (for binary or multi class)
  - One Vs Rest Classifier (for multiclass)
- Metric → Accuracy / Confusion matrix

Support Vector Regression

```
from sklearn.svm import SVR
model = SVR(kernel='rbf')
# Use 'poly' or 'linear' depending on trend
# poly →
# linear → linearly separable
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

- Use → Non-linear regression
- Metric → `mean_squared_error`

Single Layer Perceptron

```
weights = np.random.rand(X_train.shape[1])
bias = 0
lr = 0.01

for epoch in range(100):
    for i in range(len(X_train)):
        z = np.dot(X_train[i], weights) + bias
        pred = 1 if z > 0 else 0
        error = y_train[i] - pred
        weights += lr * error * X_train[i]
        bias += lr * error

# or

from sklearn.linear_model import Perceptron
model = Perceptron() # this is also allowed -- so learn this alone
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

- Use → Binary Classification [linearly separable]

Multi Layer Perceptron

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300)
mlp.fit(X_train, y_train)
pred = mlp.predict(X_test)
```

- Use → Classification problems (binary/multiclass)
- Works well for non-linear boundaries

OvO and OvA

```
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier

# --- One-vs-Rest ---
ovr_model = OneVsRestClassifier(SVC(kernel='linear'))
ovr_model.fit(X_train, y_train)
ovr_pred = ovr_model.predict(X_test)

# --- One-vs-One ---
ovo_model = OneVsOneClassifier(SVC(kernel='linear'))
ovo_model.fit(X_train, y_train)
ovo_pred = ovo_model.predict(X_test)



# or
```

```
SVC(decision_function_shape='ovo')  # OvO mode
SVC(decision_function_shape='ovr')  # OvR mode
```

Model Evaluation [Inference — Just for Documentation]

Classification

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_

# Predict
y_pred = model.predict(X_test)

# Accuracy
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Classification Report
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

Inference:

```
Model: Support Vector Machine

Accuracy: 0.88

Confusion Matrix:
[[50  2]
 [ 6 42]]

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

```
           0       0.89      0.96      0.92        52
           1      0.95      0.88      0.91        48
    accuracy                          0.91       100

Observation:
- The model performs well with an overall accuracy of 91%.
- Class 0 has slightly better precision, while class 1 has better recall.
- Minor misclassification seen.
```

```
Model: Support Vector Machine (SVM)

Accuracy: 0.93

Confusion Matrix:
[[47  3]
 [ 4 46]]

Classification Report:
              precision    recall  f1-score   support
           0       0.92      0.94      0.93        50
           1      0.94      0.92      0.93        50

Observation:
- The model performs symmetrically across both classes.
- A few misclassifications occurred, but overall precision and recall are high, i
```

```
Model: Naive Bayes

Accuracy: 0.82

Classification Report:
              precision    recall  f1-score   support
           0       0.90      0.88      0.89        90
           1      0.52      0.58      0.55        10

Observation:
```

```
- While overall accuracy is decent, the model struggles with the minority class
- The precision and recall for class 1 are relatively low, indicating bias towards
```

```
Model: Logistic Regression

ROC-AUC Score: 0.87

Observation:
The high ROC-AUC indicates good separability between the classes, even in c
The model is reliable in ranking positive vs. negative examples.
```

Regression

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_sco
import numpy as np

y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R² Score:", r2)
```

Inference:

```
Model: Linear Regression

MAE: 3.52
MSE: 18.12
RMSE: 4.25
R² Score: 0.89
```

```
Observation:
- The R² score of 0.89 shows that the model explains 89% of the variance.
- The error metrics are low, indicating a good fit.
- Some residual errors remain, possibly due to non-linearity.
```

```
Model: Linear Regression

MAE: 2.14
RMSE: 3.10
R² Score: 0.94

Observation:
The model explains 94% of the variance, with minimal residuals.
This suggests that a linear assumption fits well and the features used are high
```

```
Model: Support Vector Regression (SVR - RBF Kernel)

MAE: 4.85
RMSE: 6.92
R² Score: 0.78

Observation:
The model captures the non-linear patterns in the data moderately well.
While R² is lower compared to linear models, it avoids underfitting and handle
```

```
Model: MLP Regressor (3 hidden layers)

MAE: 3.32
RMSE: 4.75
R² Score: 0.86

Observation:
The neural network adapts well to the underlying structure of the data.
Training convergence was stable after scaling the data, and the model genera
```

Clustering

```
from sklearn.metrics import silhouette_score, davies_bouldin_score

labels = model.fit_predict(X_scaled)

sil_score = silhouette_score(X_scaled, labels)
db_score = davies_bouldin_score(X_scaled, labels)

print("Silhouette Score:", sil_score)
print("Davies-Bouldin Index:", db_score)

# K Means Specific
print("Inertia (WCSS):", model.inertia_)
```

Inference:

```
Model: K-Means Clustering (k=3)

Silhouette Score: 0.64
Davies-Bouldin Index: 0.78
Inertia: 204.5

Observation:
- A silhouette score of 0.64 suggests good clustering.
- The DB index < 1 indicates that the clusters are compact and well-separated
- Inertia value supports convergence at k=3.
```

```
Model: KMeans (k=4)

Silhouette Score: 0.69
Inertia: 154.2

Observation:
The clusters formed are compact and well-separated.
A silhouette score close to 0.7 confirms that points are closer to their own clus
The elbow method also supported k=4.
```

```
Model: Gaussian Mixture Model (GMM)

Silhouette Score: 0.41
Log-Likelihood: -302.17

Observation:
The GMM performed reasonably well, but the lower silhouette score suggests
This may indicate that some features are not well-separated or the number of
```

```
Model: KMeans (k=3)

Silhouette Score: 0.22
Davies-Bouldin Index: 1.76

Observation:
Low silhouette and high DBI imply poor separation and compactness.
Feature scaling or PCA might improve clustering.
The current feature space may not be ideal for cluster-based grouping.
```

Increasing Accuracy

Cross Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("CV Accuracy:", scores.mean())
```

Feature Selection

Recursive Feature Elimination

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
```

```
rfe = RFE(model, n_features_to_select=5)
rfe.fit(X, y)
X_selected = rfe.transform(X)
```

Select K-Best

```
from sklearn.feature_selection import SelectKBest, f_classif

selector = SelectKBest(score_func=f_classif, k=10)
X_new = selector.fit_transform(X, y)
```

Plotting

Scatter Plot

```
plt.scatter(x, y, color='blue', marker='o')
plt.title('Scatter Plot')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.grid(True)
plt.show()

# or

sns.scatterplot(x='feature1', y='feature2', data=data)
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns

cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Scatter Plot — Regression Line [Optional]

```
import seaborn as sns
import pandas as pd

# Assuming X and y are 1D
df = pd.DataFrame({'X': X.flatten(), 'y': y})
sns.regplot(x='X', y='y', data=df, ci=None)
plt.title("Regression Line")
plt.show()
```

Scatter Plot — Cluster

```
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis')
plt.title("Clustered Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Decision Boundary Plot [Optional]

```
import numpy as np

# Create meshgrid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap='coolwarm')
plt.title("Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Actual vs Predicted Plot — Regression [Optional]

```
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
# y = x line
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted")
plt.grid(True)
plt.show()
```

Submission

```
submission_pred = mlp.predict(test_1)

df = pd.DataFrame({"PassengerId":test['PassengerId'],
                   "Survived":submission_pred})
# Column names should be same as the ones in the train data
df.to_csv("/kaggle/working/submission.csv", index=False)

# Just for checking
output = pd.read_csv('/kaggle/working/submission.csv')
output
```

Notes

Ways to check Linearity:

- Use scatter plots (`sns.pairplot` or `plt.scatter`)
- Try fitting a Linear Regression and plot predicted vs actual
- Use PCA (2D projection) — see if classes look linearly separated

- Try SVM with linear kernel → low performance means it's non-linear
- Try logistic regression or perceptron — if accuracy is poor, try non-linear models

Documentation — Inference:

- Start with model name and key metric.
- Mention if the result was expected (good/bad).
- Comment on whether the dataset is linear, balanced, noisy, etc.
- Say what could improve results (scaling, feature selection, different model).
- Keep it factual but clear.

Key Points:

- If `Naive Bayes` fails → maybe features are not independent
- If `Linear Regression` fails → maybe data has non-linearity or outliers
- If SVM takes too long → reduce dataset size or use `LinearSVC`
- For **imbalanced data** → use `class_weight='balanced'` or `SMOTE`
- For **non-vectorial data** (like text, images) → use embeddings or CNNs
- **Always scale before**: SVM, KMeans, Neural Nets