DSML - All 25 Assignments (Commented Code + Cheatsheet)

1. Medical Diagnosis Using SVM

Dataset: **01_svm_skin_lesion.csv** — Description: SVM classifier on image-derived features (B/M)

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("01_svm_skin_lesion.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['diagnosis'].dtype == 'object':
    df['diagnosis'] = df['diagnosis'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['diagnosis'])
y = df['diagnosis']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

2. Loan Default Prediction with Decision Trees

Dataset: **02_decision_tree_loan.csv** — Description: Decision Tree for loan default

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("02_decision_tree_loan.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['default'].dtype == 'object':
    df['default'] = df['default'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['default'])
y = df['default']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

3. Disease Classification with KNN

Dataset: **03_knn_diabetes.csv** — Description: KNN for diabetes

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("03_knn_diabetes.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['diabetes'].dtype == 'object':
    df['diabetes'] = df['diabetes'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['diabetes'])
y = df['diabetes']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

4. Email Spam Detection Using Naive Bayes

Dataset: **04_nb_spam.csv** — Description: Naive Bayes for spam

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("04_nb_spam.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['spam'].dtype == 'object':
    df['spam'] = df['spam'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['spam'])
y = df['spam']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

5. Customer Sentiment Analysis with SVM

Dataset: **05_svm_sentiment.csv** — Description: SVM for sentiment

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("05_svm_sentiment.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['sentiment'].dtype == 'object':
    df['sentiment'] = df['sentiment'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['sentiment'])
y = df['sentiment']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

6. House Price Prediction Using Linear Regression

Dataset: **06_linear_house_price.csv** — Description: Linear Regression for house price

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("06_linear_house_price.csv")
print('Dataset shape:', df.shape)

X = df.drop(columns=['price'])
y = df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print('RMSE:', mean_squared_error(y_test, y_pred, squared=False))
print('R2:', r2_score(y_test, y_pred))
```

7. Churn Probability with Logistic Regression

Dataset: **07_logistic_churn.csv** — Description: Logistic Regression for churn

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("07_logistic_churn.csv")
print('Dataset shape:', df.shape)

X = df.drop(columns=['churn'])
y = df['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print('RMSE:', mean_squared_error(y_test, y_pred, squared=False))
print('R2:', r2_score(y_test, y_pred))
```

8. Stock Price Forecasting with Least Squares

Dataset: **08_least_squares_stock.csv** — Description: Linear Regression for stock

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("08_least_squares_stock.csv")
print('Dataset shape:', df.shape)

X = df.drop(columns=['closing_price'])
y = df['closing_price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print('RMSE:', mean_squared_error(y_test, y_pred, squared=False))
print('R2:', r2_score(y_test, y_pred))
```

9. Crop Yield Estimation Using Linear Regression

Dataset: **09_linear_crop_yield.csv** — Description: Linear Regression for crop yield

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("09_linear_crop_yield.csv")
print('Dataset shape:', df.shape)

X = df.drop(columns=['yield'])
y = df['yield']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print('RMSE:', mean_squared_error(y_test, y_pred, squared=False))
print('R2:', r2_score(y_test, y_pred))
```

10. Heart Disease Risk Prediction with Logistic Regression

Dataset: **10_logistic_heart.csv** — Description: Logistic Regression for heart disease

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("10_logistic_heart.csv")
print('Dataset shape:', df.shape)

X = df.drop(columns=['heart_disease'])
y = df['heart_disease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print('RMSE:', mean_squared_error(y_test, y_pred, squared=False))
print('R2:', r2_score(y_test, y_pred))
```

11. Fraud Detection in Financial Transactions

Dataset: **11_rf_fraud.csv** — Description: Random Forest for fraud

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("11_rf_fraud.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['fraud'].dtype == 'object':
    df['fraud'] = df['fraud'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['fraud'])
y = df['fraud']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

12. Employee Attrition Prediction

Dataset: **12_rf_attrition.csv** — Description: Random Forest for attrition

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("12_rf_attrition.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['attrition'].dtype == 'object':
    df['attrition'] = df['attrition'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['attrition'])
y = df['attrition']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

13. Air Quality Classification

Dataset: **13_rf_air_quality.csv** — Description: AQI classification

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("13_rf_air_quality.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['aqi_level'].dtype == 'object':
    df['aqi_level'] = df['aqi_level'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['aqi_level'])
y = df['aqi_level']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

14. Product Recommendation System

Dataset: **14_rf_recommendation.csv** — Description: Recommendation/purchase prediction

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("14_rf_recommendation.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['purchased'].dtype == 'object':
    df['purchased'] = df['purchased'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['purchased'])
y = df['purchased']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

15. Customer Segmentation for Marketing

Dataset: **15_kmeans_customers.csv** — Description: KMeans clustering for customers

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np

df = pd.read_csv("15_kmeans_customers.csv")
print('Dataset shape:', df.shape)

X = df.select_dtypes(include=[np.number]).dropna()
scaler = StandardScaler(); X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_scaled)
print('Inertia:', kmeans.inertia_)
print('Cluster counts:', pd.Series(labels).value_counts().to_dict())

pca = PCA(n_components=2, random_state=42); X2 = pca.fit_transform(X_scaled)
print('PCA shape:', X2.shape)
```

16. Document Clustering for News Articles

Dataset: **16_kmeans_documents.csv** — Description: KMeans for documents

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np

df = pd.read_csv("16_kmeans_documents.csv")
print('Dataset shape:', df.shape)

X = df.select_dtypes(include=[np.number]).dropna()
scaler = StandardScaler(); X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_scaled)
print('Inertia:', kmeans.inertia_)
print('Cluster counts:', pd.Series(labels).value_counts().to_dict())

pca = PCA(n_components=2, random_state=42); X2 = pca.fit_transform(X_scaled)
print('PCA shape:', X2.shape)
```

17. Image Compression Using K-Means

Dataset: **17_kmeans_image_compression.csv** — Description: KMeans for image colors

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np

df = pd.read_csv("17_kmeans_image_compression.csv")
print('Dataset shape:', df.shape)

X = df.select_dtypes(include=[np.number]).dropna()
scaler = StandardScaler(); X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_scaled)
print('Inertia:', kmeans.inertia_)
print('Cluster counts:', pd.Series(labels).value_counts().to_dict())

pca = PCA(n_components=2, random_state=42); X2 = pca.fit_transform(X_scaled)
print('PCA shape:', X2.shape)
```

18. Traffic Pattern Analysis

Dataset: **18_kmeans_traffic.csv** — Description: KMeans for traffic

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np

df = pd.read_csv("18_kmeans_traffic.csv")
print('Dataset shape:', df.shape)

X = df.select_dtypes(include=[np.number]).dropna()
scaler = StandardScaler(); X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_scaled)
print('Inertia:', kmeans.inertia_)
print('Cluster counts:', pd.Series(labels).value_counts().to_dict())

pca = PCA(n_components=2, random_state=42); X2 = pca.fit_transform(X_scaled)
print('PCA shape:', X2.shape)
```

19. Market Basket Analysis for Retail

Dataset: **19_apriori_retail.csv** — Description: Apriori for retail

```
import pandas as pd
try:
    from mlxtend.frequent_patterns import apriori, association_rules
except Exception as e:
    raise ImportError('mlxtend required') from e

df = pd.read_csv("19_apriori_retail.csv")
print('Dataset shape:', df.shape)
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
print('Frequent itemsets:\n', frequent_itemsets)
rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.5)
print('Rules:\n', rules[['antecedents','consequents','support','confidence','lift']].head())
```

20. Web Navigation Pattern Mining

Dataset: **20_apriori_web.csv** — Description: Apriori for web logs

```
import pandas as pd
try:
    from mlxtend.frequent_patterns import apriori, association_rules
except Exception as e:
    raise ImportError('mlxtend required') from e

df = pd.read_csv("20_apriori_web.csv")
print('Dataset shape:', df.shape)
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
print('Frequent itemsets:\n', frequent_itemsets)
rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.5)
print('Rules:\n', rules[['antecedents','consequents','support','confidence','lift']].head())
```

21. Medical Prescription Pattern Discovery

Dataset: **21_apriori_prescriptions.csv** — Description: Apriori for prescriptions

```
import pandas as pd
try:
    from mlxtend.frequent_patterns import apriori, association_rules
except Exception as e:
    raise ImportError('mlxtend required') from e

df = pd.read_csv("21_apriori_prescriptions.csv")
print('Dataset shape:', df.shape)
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
print('Frequent itemsets:\n', frequent_itemsets)
rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.5)
print('Rules:\n', rules[['antecedents','consequents','support','confidence','lift']].head())
```

22. Online Course Recommendation

Dataset: **22_apriori_courses.csv** — Description: Apriori for courses

```
import pandas as pd
try:
    from mlxtend.frequent_patterns import apriori, association_rules
except Exception as e:
    raise ImportError('mlxtend required') from e

df = pd.read_csv("22_apriori_courses.csv")
print('Dataset shape:', df.shape)
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
print('Frequent itemsets:\n', frequent_itemsets)
rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.5)
print('Rules:\n', rules[['antecedents','consequents','support','confidence','lift']].head())
```

23. Retail Loss Prevention

Dataset: **23_apriori_loss_prevention.csv** — Description: Apriori for loss prevention

```
import pandas as pd
try:
    from mlxtend.frequent_patterns import apriori, association_rules
except Exception as e:
    raise ImportError('mlxtend required') from e

df = pd.read_csv("23_apriori_loss_prevention.csv")
print('Dataset shape:', df.shape)
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
print('Frequent itemsets:\n', frequent_itemsets)
rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.5)
print('Rules:\n', rules[['antecedents','consequents','support','confidence','lift']].head())
```

24. Predictive Maintenance Using Classification and Regression

Dataset: **24_predictive_maintenance.csv** — Description: Classification + Regression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("24_predictive_maintenance.csv")
print('Dataset shape:', df.shape)

X = df.drop(columns=['failure/time_to_fail'])
y = df['failure/time_to_fail']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print('RMSE:', mean_squared_error(y_test, y_pred, squared=False))
print('R2:', r2_score(y_test, y_pred))
```

25. Healthcare Personalization Using Clustering and Association Rules

Dataset: **25_healthcare_clusters_rules.csv** — Description: Clustering + Apriori for healthcare

```
# Load libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Model import - choose appropriate
from sklearn.svm import SVC
model = SVC(kernel='linear', probability=True)

# Read dataset CSV
df = pd.read_csv("25_healthcare_clusters_rules.csv")  # ensure file is in same folder

# Print shape for quick check
print('Dataset shape:', df.shape)

# If target is categorical (B/M), convert to numeric 0/1
if df['TARGET'].dtype == 'object':
    df['TARGET'] = df['TARGET'].map(lambda x: 0 if str(x).strip().upper() in ['B','0','NO'] else 1)

# Separate features and target
X = df.drop(columns=['TARGET'])
y = df['TARGET']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y if len(y.u

# Scale numeric features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model on training data
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate performance
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))

# If model supports probabilities, compute AUC
if hasattr(model, 'predict_proba'):
    from sklearn.metrics import roc_curve, auc
    y_prob = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    print('AUC:', auc(fpr, tpr))
```

One-Page Cheatsheet

```
DSML Practical One-Page Cheatsheet
--------------------------------
Steps to use in exam:
1) import pandas as pd
2) df = pd.read_csv('FILE.csv')
3) X = df.drop('TARGET', axis=1); y = df['TARGET']
4) from sklearn.model_selection import train_test_split
   X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
5) from sklearn.preprocessing import StandardScaler
   sc = StandardScaler(); X_train = sc.fit_transform(X_train); X_test = sc.transform(X_test)
6) model = <model>; model.fit(X_train,y_train); y_pred = model.predict(X_test)
7) For classification: accuracy_score, confusion_matrix, classification_report
   For regression: mean_squared_error, r2_score
Pro tips: ensure both classes present; use stratify=y for classification
```

Memory-Friendly Template

```
Memory-Friendly Template (change FILE and TARGET only)
-----------------------------------------------------
FILE = '01_svm_skin_lesion.csv'
TARGET = 'diagnosis'

# Choose one model (uncomment)
# from sklearn.svm import SVC; model = SVC(kernel='linear', probability=True)
# from sklearn.neighbors import KNeighborsClassifier; model = KNeighborsClassifier(3)
# from sklearn.tree import DecisionTreeClassifier; model = DecisionTreeClassifier()
# from sklearn.naive_bayes import GaussianNB; model = GaussianNB()
# from sklearn.linear_model import LogisticRegression; model = LogisticRegression(max_iter=500)
# from sklearn.ensemble import RandomForestClassifier; model = RandomForestClassifier(n_estimators=50)
# from sklearn.linear_model import LinearRegression; model = LinearRegression()
# from sklearn.cluster import KMeans; model = KMeans(n_clusters=3)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv(FILE)
X = df.drop(columns=[TARGET]); y = df[TARGET]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
sc = StandardScaler(); X_train = sc.fit_transform(X_train); X_test = sc.transform(X_test)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))
```