Support Vector Machines

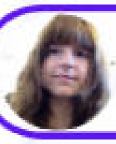
Introduction to SVM and SVR

Unique business applications

Math behind

“How to” in Python:

from SMART business question to model tuning



Julia Lenc

Analytics Journey

Business Analytics (BA)

1. Intro: Business and Revenue models. KPIs
2. Business models translated into analytics
3. Techniques: Descriptive, Diagnostic, Predictive, Prescriptive

Diagnostic Techniques

1. Inference: hypotheses testing
2. Unsupervised Learning: clustering, dimensionality reduction, anomalies

Predictive Techniques

1. Supervised learning: overview
2. Preparation: data pre-processing
3. Foundations: model choice and evaluation
4. Regression: linear and non-linear
5. Classification: logistic regression, Naive Bayes, k-NNs
6. Time series: ARIMA, SARIMA, Exponential Smoothing
7. Non-linear: a. Decision Trees, b. SVM (this presentation!), c. (G)ARCH
8. Ensemble: bagging, boosting, stacking
9. Neural Networks: FFNN, CNN, RNN, Transformers

Prescriptive Techniques

1. Optimization: Linear, Non-linear and Dynamic programming
2. Simulation: Monte Carlo, Discreet Events, System Dynamics
3. Probabilistic Sequence: Markov Chains, Markov Decision Processes
4. Reinforcement Learning: Q-Learning, Deep RL, Policy Gradient

Julia Lenc

Support Vector Machines
Support Vector Regression

Intro

1. Classification vs Regression
2. What is SVM and SVR?
3. Components: margin, support vectors, hyperplane, kernel, C parameter, gamma.
4. Business applications

Math

1. The essence: support vectors, margins and hyperplanes.
2. Addressing non-linearity: kernels
3. Hyperparameters: C and Gamma
4. Regularization (memento overfitting!)
5. SVM vs SVR

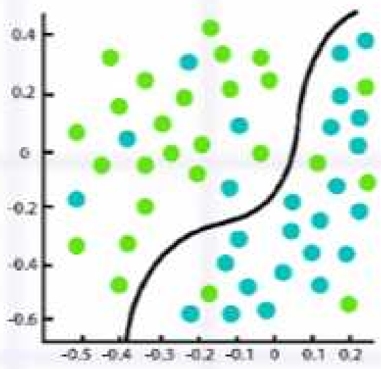
Modeling steps + Python script

1. SMART business question
2. Data preparation
3. EDA and feature selection
4. Training and evaluating initial model
5. Steps if initial model doesn't meet evaluation criteria:
   - Overfitting: lower C, try simpler kernel, reduce features.
   - Underfitting: increase C, try more complex kernel, add features.
   - Kernel tricks for non-linear problems.
   - Hyperparameter tuning: GridSearchCV.

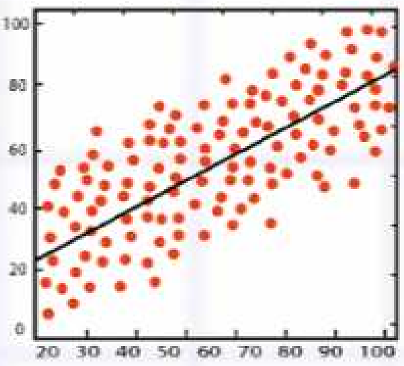
Supervised Learning Types

Classification
Assignment of a probability or a category to an observation.
Key questions: "**Which group**?", "How likely?", “Is it A or B?”
Popular applications: churn prediction, recommender systems

Regression
Prediction of a continuous value - number or quantity.
Key questions: "**How much**?", "What will the value be?"
Popular applications: sales forecast, budget changes, pricing



Classification



Regression

Julia Lenc

What is SVM and SVR?

Definition

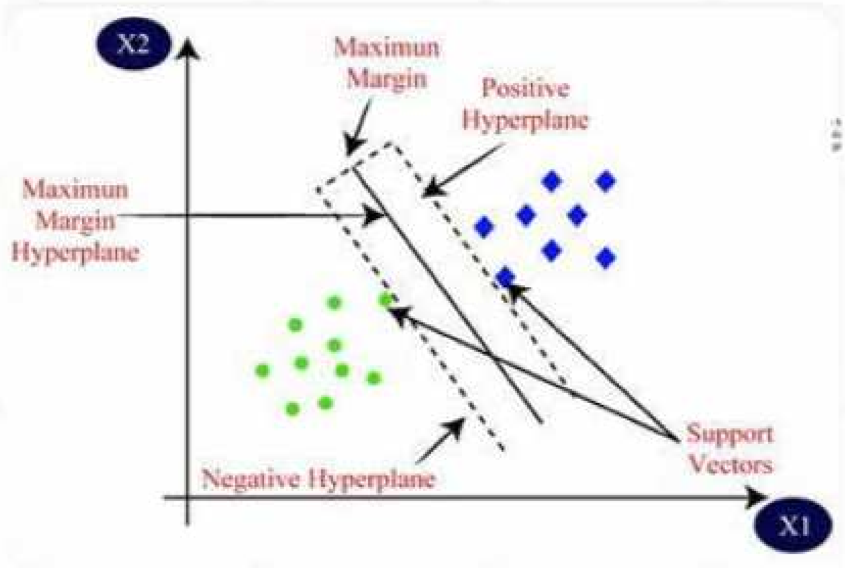
Support Vector Machine (SVM) is a predictive model that finds the most effective **boundary (hyperplane) to separate groups** in the data, maximizing the **margin** between them. SVM is used for classification; SVR (Support Vector Regression) applies the same principle for regression. SVM/SVR focus on the most critical data points: the **support vectors**. Thanks to **kernel tricks**, SVM can reveal non-linear patterns other models miss

Key applications:

1. Marketing: complex customer segmentation, targeting.
2. Operations & HR: predicting employee attrition, demand peaks.
3. Finance: credit risk scoring, transaction fraud detection.

Julia Lenc

SVM / SVR core components



Definitions:

Hyperplane: the boundary that best separates the classes.

Margins: the “buffer zones” on either side of the hyperplane.

Support vectors: the key points closest to the boundary.

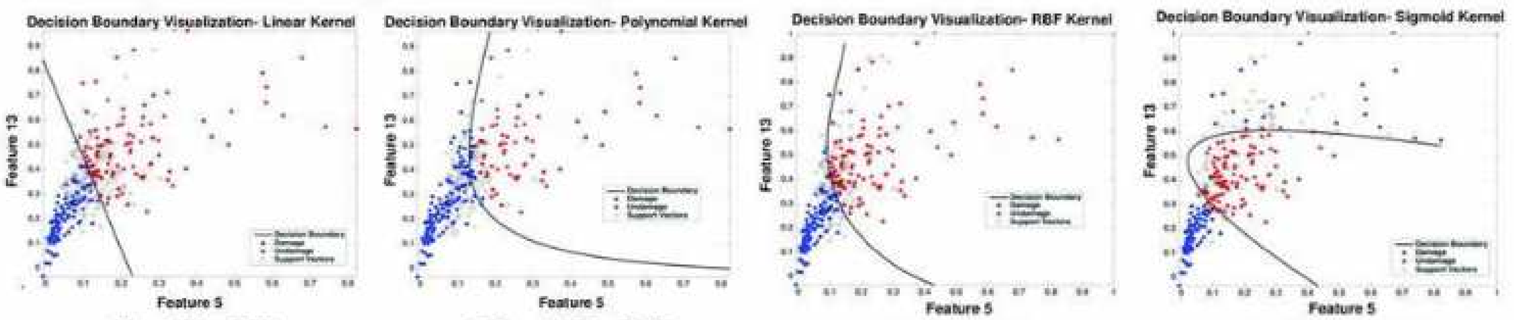
They define the margin, and thus, the model.

Business tip

Wider margins = better generalization, less likely to overfit to noise

Julia Lenc

Kernel trick



(a) Linear. DDA = 85.78% (b) Polynomial. DDA = 78.92% (c) RBF. DDA = 88.24% (d) Non-Linear. DDA = 79.90%

Explanation:

SVM can use different **kernels (linear, polynomial, RBF, sigmoid)** to create flexible decision boundaries. Kernel trick enables SVM to "see" **non-linear relationships** standard models miss.

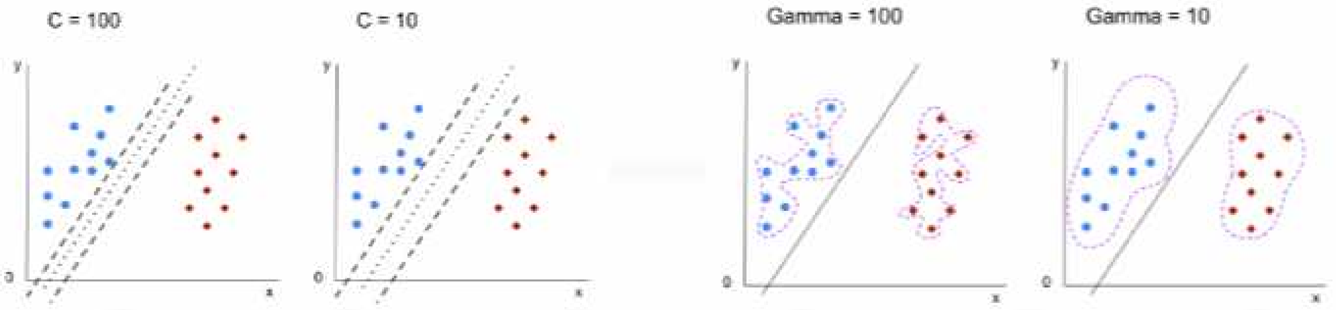
Widely used for text classification and image recognition.

Business tip

Kernel choice = SVM's secret superpower for tricky, non-linear data.

Julia Lenc

C and Gamma



Definitions:

C parameter: controls how much the model tries to avoid misclassifying training examples. High C = less margin, more focus on getting every point correct (risk: overfit). Low C = larger margin, allows some misclassification (better generalization).

Gamma: controls how far influence of a single point reaches. High gamma = tight fit around points (risk: overfit). Low gamma = smoother boundary.

Business tip

Tuning C and Gamma is like balancing “flexibility” vs “simplicity”. Critical for stable predictions.

Julia Lenc

Business Applications

By Business Model elements

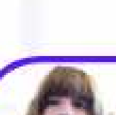
- **Value Creation**: segmenting customers with complex, high-dimensional data (SVM), churn prediction with noisy data (SVM)
- 💰 **Revenue Model**: detecting pricing anomalies and outliers (SVM), bank loan amount or insurance premium (SVR)
- 🔍 **Market Opportunity**: finding rare, high-value segments (SVM)
- 🛒 **Go-to-Market**: targeting audiences with fuzzy clusters (SVM), non-linear effects of campaigns - plateaus, saturation points, diminishing returns (SVR)
- 🏢 **Operations**: fraud and incident detection (SVM), forecasting sharp shifts, surges or costs (SVR)

Legend:

SVM - Support Vector Machines, SVR - Support Vector Regression

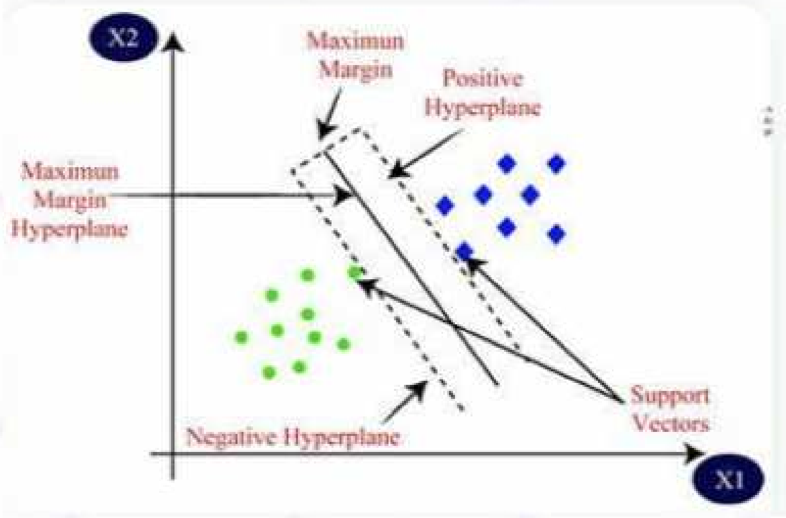
Julia Lenc

Math behind SVM and SVR



Julia Lenc

The essence of SVM



How it works:

1. The observations in the dataset have **features** (x1, x2, ... , xn) and labels (**class**). E.g., age = 29, daily social media time = 183min, class = “Prime prospect”. Typical SVM handles two classes, but can be more.

2. SVM finds observations from different classes, closest to each other based on their coordinates - **support vectors**. Let’s say we have A as support vector for positive hyperplane and D as support vector for negative hyperplane.

3. Support vectors define the **hyperplanes**.

   Positive hyperplane: $b + w_1 x_{1A} + w_2 x_{2A} + ... + w_n x_{nA} = +1$
   Negative hyperplane $b + w_1 x_{1D} + w_2 x_{2D} + ... + w_n x_{nD} = -1$

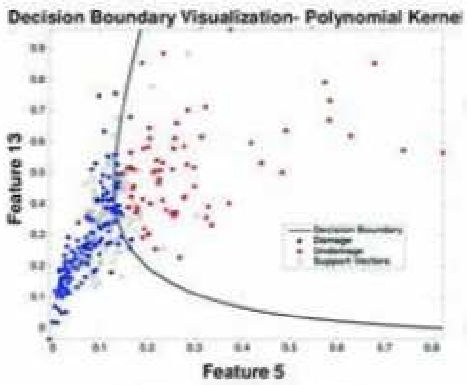
   x1, x2,..., xn: coordinates of the variable which defined support vector
   w1, w2,... wn (coefficients) and **b (intercept):** coefficients that SVM calculates.

4. The coefficients and intercept are used to calculate **maximum margin hyperplane**, which lies **exactly in between** the positive and negative hyperplanes.

   Maximum margin hyperplane: $b + w_1 x_1 + w_2 x_2 + ... + w_n x_n = 0$
   Maximum **margin width**: $\frac{2}{||w||}$

Kernel trick

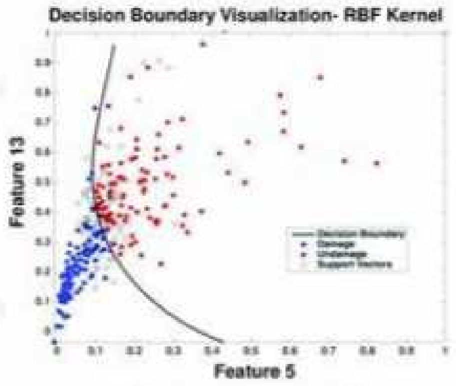
What if we can't separate classes with a straight line or flat hyperplane?



Polynomial kernel

$$(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b)^d = 0$$

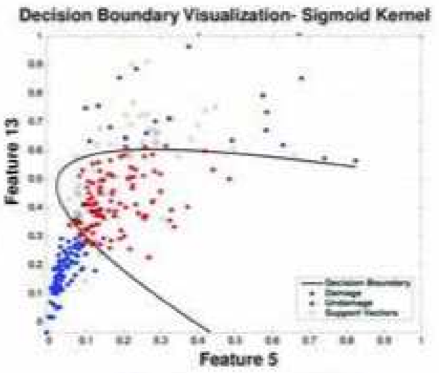
w: coefficient, importance of each feature, calculated by SVM
x: x1, x2, ..., xn , from dataset
b: intercept, calculated by SVM
d: degree of polynomial. d=1 (linear), d=2 (quadric), d>2 (complex, bewared of overfitting!)



Radial Basis Function (RBF) kernel

$$\sum_{i=1}^{N} \alpha_i y_i \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2\right) + b = 0$$

ai: weights for each support vector, calculated by SVM
yi: class label, either +1 or -1 (previous slide)
γ: gamma, controls the influence of support vectors (next slide
//x–xi//2: squared Euclidean distance between test point x and support vector xi
b: intercept, calculated by SVM



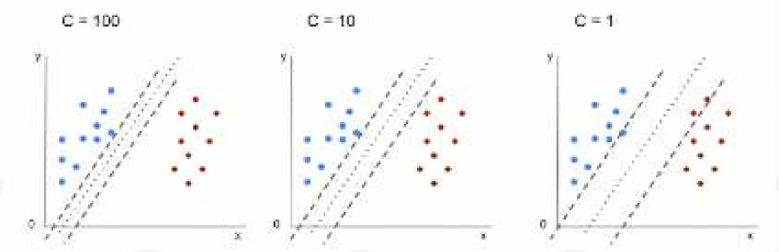
Sigmoid kernel

$$\sum_{i=1}^{N} \alpha_i y_i \tanh(\alpha \mathbf{x}^T \mathbf{x}_i + c) + b = 0$$

tanh: hyperbolic tangent, nonlinear (S-shaped) function, output between -1 and +1
α: kernel slope, controls the steepness of S-curve
c: kernel intercept, shifts the S-shaped curve left/right
ai, yi, x, xi, b: like for Polynomial and RBF kernels

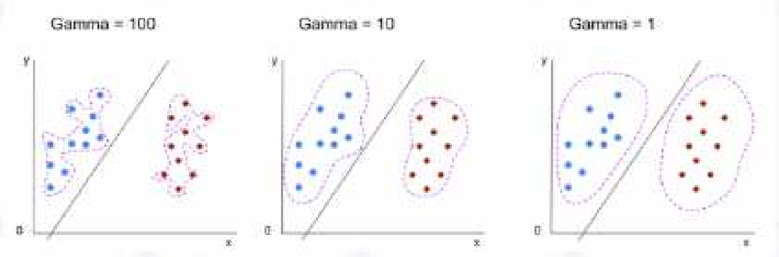
Hyperparameters C and Gamma

Hyperparameters: values set by data scientist before the training; they control how the learning process happens. **C is used in all kernels, Gamma in all except linear** kernel.



Hyperparameter C:

- Controls how wide the margin is = misclassification tolerance.
- **Lower C** → wider margin -> better **generalization**, but new cases may fall into the margin and become ambiguous (**underfitting**).
- **Higher C** → thinner margin → model fits the training data more tightly, more **accurate** classification but higher risk of **overfitting** to noise and outliers.



Hyperparameter Gamma:

- Controls how far the influence of a single training example reaches.
- **Lower Gamma** → each point has a large influence → the decision boundary is smoother → better **generalization**, but might miss small patterns (**underfitting**).
- **Higher Gamma** -> each point has a small influence → more complex decision boundaries, more **accurate** classification but risks **overfitting** to noise and outliers.

Regularization

Regularization: technique to prevent overfitting by discouraging complex models (keeps weights/slope small). Used aftermodel evaluation in case of overfitting or high variance.

Overfitting: model performs very well on training data but poorly on test or validation data because it "memorized" data instead of learning general patterns.

L1 regularization (Lasso): great for feature selection

$$\text{Loss}_{L1} = \text{Loss} + \lambda \sum_i |\beta_i|$$

- Adds sum of absolute values of the coefficients to the loss function.
- Encourages sparsity (drives some coefficients to exactly zero).

L2 regularization (Ridge): keeps all features but decreases magnitude of less important

$$\text{Loss}_{L2} = \text{Loss} + \lambda \sum_i \beta_i^2$$

- Adds sum of squares of the coefficients to the loss function.
- Encourages small but nonzero coefficients (shrinks them toward zero).

Loss: model prediction error (fit)

λ: regularization strength; larger λ = heavier penalty on large coefficients

βi: model coefficients, one per feature

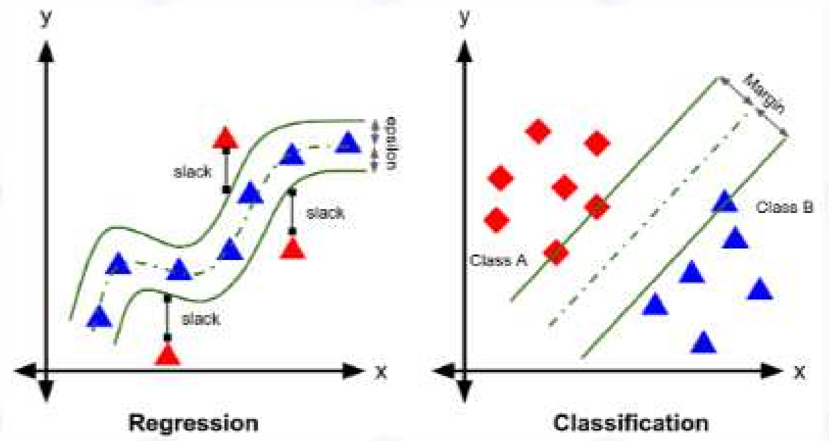
Support Vector Regression (SVR)

SVM: classification model, the class is predicted.

SVR: regression model, the value (e.g., amount, price, consumption) is predicted.

SVR its a line or a function through the data, allowing targets to fall within a tube (**epsilon**) around the prediction. Only points outside the tube matter for tweaking the prediction - this helps handle outliers and focus on "big errors."

Slack variables (arrows) show points that fall outside the "tube"; they add penalty but don't collapse the whole model.



Business applications:

- Loan amount given income, risk, credit
- House price estimation given its parameters
- Insurance premium estimation given demographics, health, history

How to set up SVM and SVR

Julia Lenc

The process: Stage 1

Understand and formulate the business question as:

Specific: Define the problem clearly

- Is this a classification (class, probability) or regression (number) problem?
- Which group/class do we want to predict? E.g., customer churn: yes/no, which product category will be chosen.
- What quantity are we predicting? E.g., sales, number of calls, temperature.

Measurable: Determine evaluation criteria. E.g., MSE or F1.

Achievable: Ensure data and resources match the modeling goals.

Relevant: Validate that the forecast will drive action. E.g., demand forecasting   resource planning; probabilities   recommenders.

Time-bound: Identify stakeholders and key deadlines for delivery.

Prepare your data:

Read here about 8 stages of data preparation.

Scaling / normalization and Encoding are **critical** for SVM and SVR.

Julia Lenc

The process: Stage 3

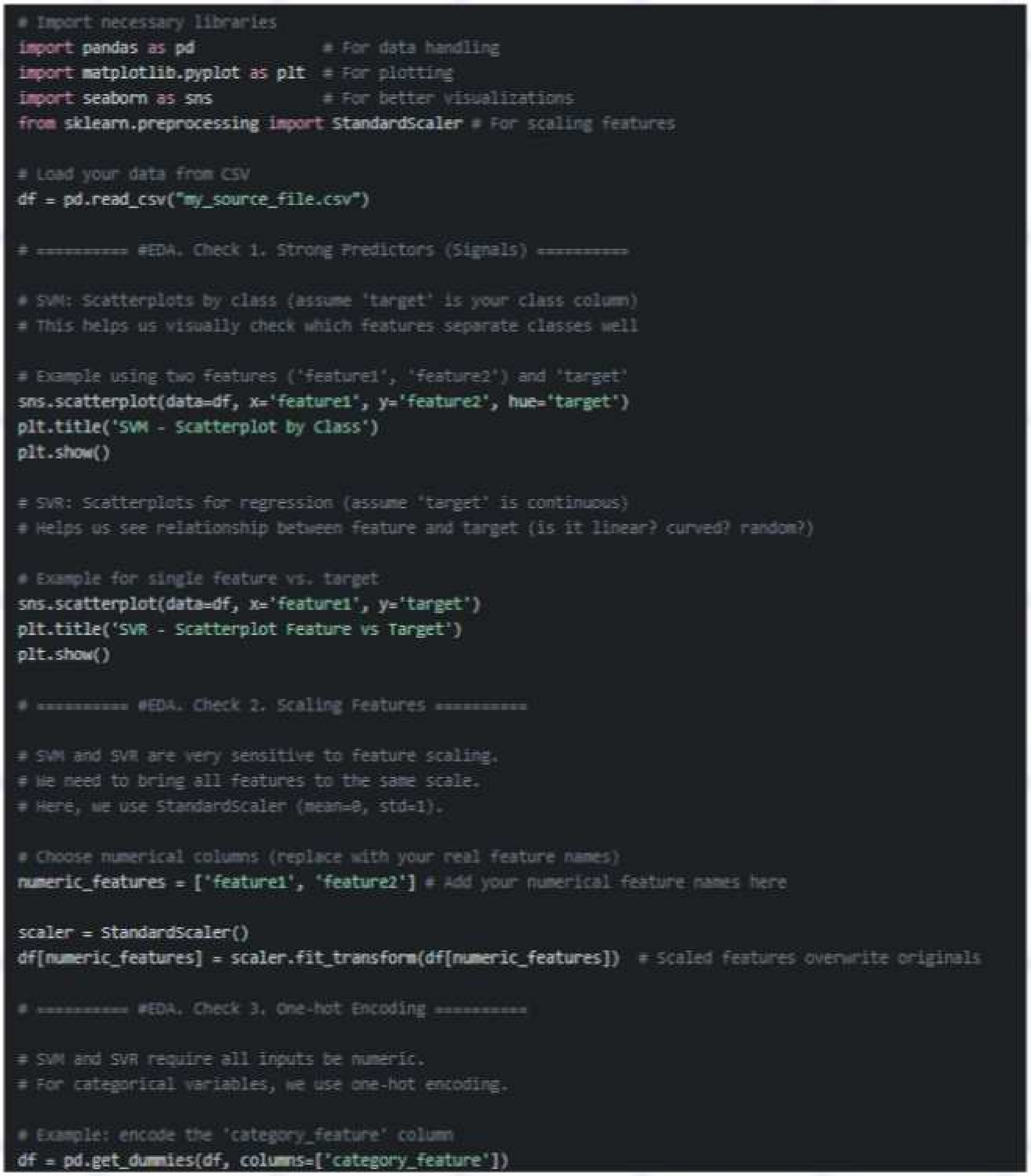
EDA and feature selection

What to check?

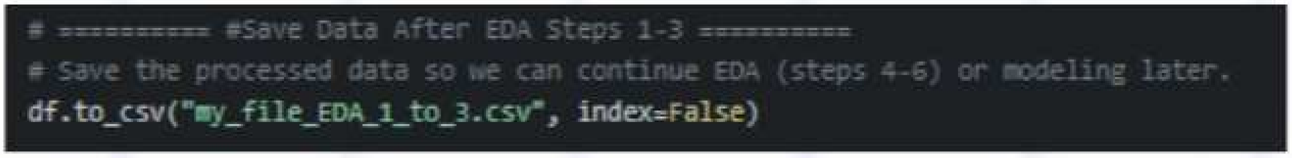
1. **Strong Predictors (Signal)**
   - Identify features with clear relationship to your target.
   - Visualize (for SVM): boxplots, violin plots, scatterplots by class.
   - Visualize (for SVR): scatterplots, line plots, correlation coefficients.
2. **Scaling of Features**
   - SVM/SVR are highly sensitive to feature scales. Always check the distribution/range of each variable before model training.
3. **Encoding of Categorical Features**
   - All inputs must be numeric! Use one-hot encoding or similar approaches.
4. **High-Cardinality Categorical Features**
   - Too many unique categories can create a flood of features (after encoding). Consider combining rare categories or selecting the most informative levels.
5. **Outliers**
   - SVM/SVR can be sensitive to outliers, especially with linear kernels. Visualize with boxplots or histograms; consider capping if needed.
6. **Remove or combine very similar features (strong correlations).**
7. **Dimensionality Reduction**
   - With many features, SVM/SVR performance can suffer.
   - Consider PCA or similar techniques to simplify your feature space.

Julia Lenc

The process: Stage 3.1 (EDA)

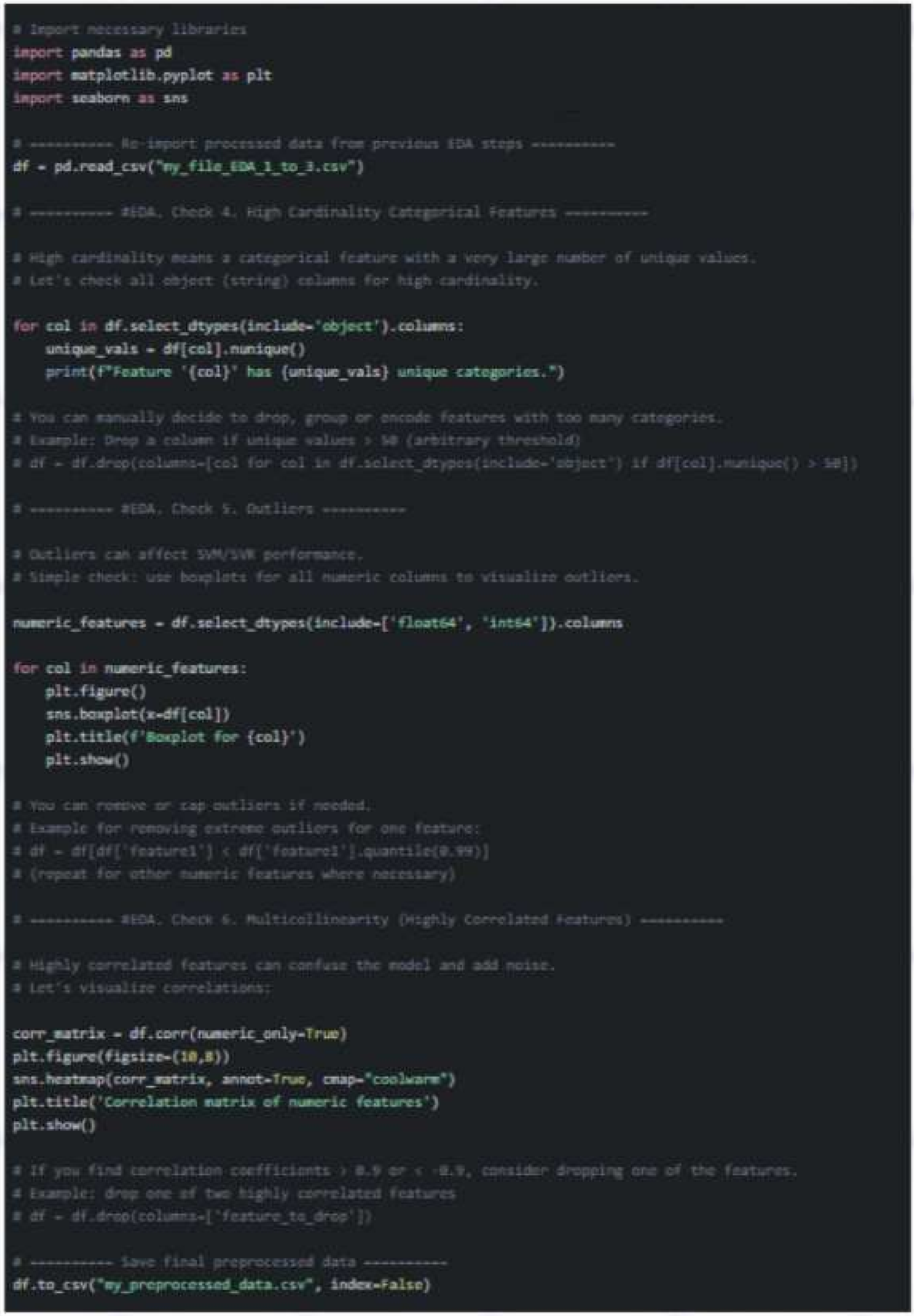


```python
# Import necessary libraries
import pandas as pd              # For data handling
import matplotlib.pyplot as plt  # For plotting
import seaborn as sns            # For better visualizations
from sklearn.preprocessing import StandardScaler # For scaling features

# Load your data from CSV
df = pd.read_csv("my_source_file.csv")

# ========== #EDA. Check 1. Strong Predictors (Signals) ==========

# SVM: Scatterplots by class (assume 'target' is your class column)
# This helps us visually check which features separate classes well

# Example using two features ('feature1', 'feature2') and 'target'
sns.scatterplot(data=df, x='feature1', y='feature2', hue='target')
plt.title('SVM - Scatterplot by Class')
plt.show()

# SVR: Scatterplots for regression (assume 'target' is continuous)
# Helps us see relationship between feature and target (is it linear? curved? random?)

# Example for single feature vs. target
sns.scatterplot(data=df, x='feature1', y='target')
plt.title('SVR - Scatterplot Feature vs Target')
plt.show()

# ========== #EDA. Check 2. Scaling Features ==========

# SVM and SVR are very sensitive to feature scaling.
# We need to bring all features to the same scale.
# Here, we use StandardScaler (mean=0, std=1).

# Choose numerical columns (replace with your real feature names)
numeric_features = ['feature1', 'feature2'] # Add your numerical feature names here

scaler = StandardScaler()
df[numeric_features] = scaler.fit_transform(df[numeric_features])  # Scaled features overwrite originals

# ========== #EDA. Check 3. One-hot Encoding ==========

# SVM and SVR require all inputs be numeric.
# For categorical variables, we use one-hot encoding.

# Example: encode the 'category_feature' column
df = pd.get_dummies(df, columns=['category_feature'])
```



```python
# ========== #Save Data After EDA Steps 1-3 ==========
# Save the processed data so we can continue EDA (steps 4-6) or modeling later.
df.to_csv("my_file_EDA_1_to_3.csv", index=False)
```

The process: Stage 3.2 (EDA)



```python
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# ---------- Re-import processed data from previous EDA steps ----------
df = pd.read_csv("my_file_EDA_1_to_3.csv")

# ---------- #EDA. Check 4. High Cardinality Categorical Features ----------

# High cardinality means a categorical feature with a very large number of unique values.
# Let's check all object (string) columns for high cardinality.

for col in df.select_dtypes(include='object').columns:
    unique_vals = df[col].nunique()
    print(f"Feature '{col}' has {unique_vals} unique categories.")

# You can manually decide to drop, group or encode features with too many categories.
# Example: Drop a column if unique values > 50 (arbitrary threshold)
# df = df.drop(columns=[col for col in df.select_dtypes(include='object') if df[col].nunique() > 50])

# ---------- #EDA. Check 5. Outliers ----------

# Outliers can affect SVM/SVR performance.
# Simple check: use boxplots for all numeric columns to visualize outliers.

numeric_features = df.select_dtypes(include=['float64', 'int64']).columns

for col in numeric_features:
    plt.figure()
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot for {col}')
    plt.show()

# You can remove or cap outliers if needed.
# Example for removing extreme outliers for one feature:
# df = df[df['feature1'] < df['feature1'].quantile(0.99)]
# (repeat for other numeric features where necessary)

# ---------- #EDA. Check 6. Multicollinearity (Highly Correlated Features) ----------

# Highly correlated features can confuse the model and add noise.
# Let's visualize correlations:

corr_matrix = df.corr(numeric_only=True)
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title('Correlation matrix of numeric features')
plt.show()

# If you find correlation coefficients > 0.9 or < -0.9, consider dropping one of the features.
# Example: drop one of two highly correlated features
# df = df.drop(columns=['feature_to_drop'])

# ---------- Save final preprocessed data ----------
df.to_csv("my_preprocessed_data.csv", index=False)
```

The process: Stage 4

Training. Evaluation (SVM)

Confusion matrix

	Predicted positives	Predicted negatives
Actual positives	True positives (TP)	False negatives (FN)
Actual negatives	False positives (FP)	True negatives (TN)

Core evaluations metrics

Accuracy: overall correctness. (TP + TN) / total predictions

Precision: % correctly predicted positives among all positives. TP / (TP + FP)
Critical if the cost of False Positive is high. Examples: insurance company pays false claim, bank gives a loan to a customer with bad records, cybersecurity (false alarms disrupt operations).

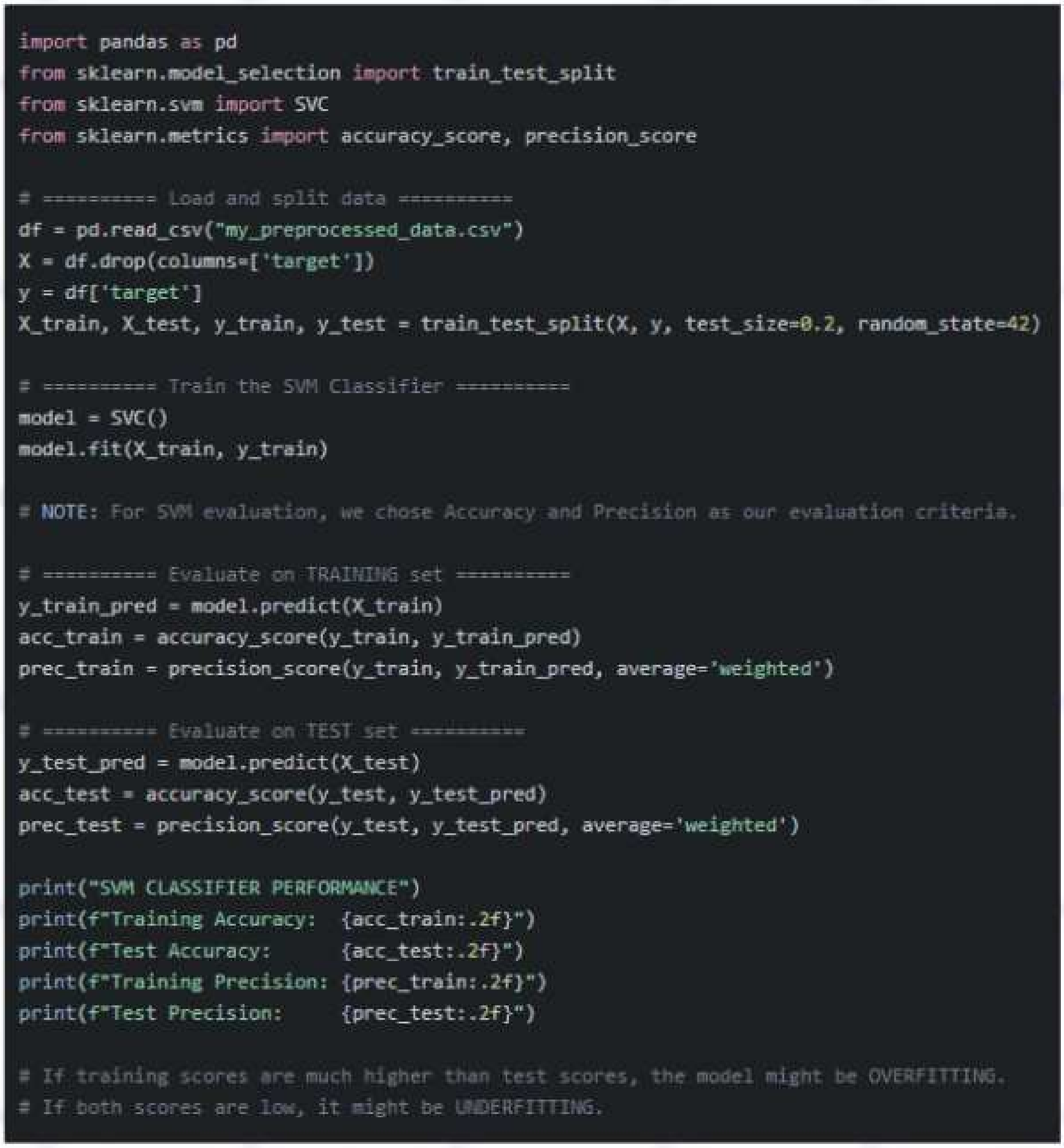
Recall: % correctly predicted positives among all corrects. TP / (TP + FN)
Critical if capturing Positives is more important than avoiding False Positives. Examples: fraudulent transactions detection, retail recommender system.

F1 score: model's performance. 2*(Precision * Recall) / (Precision + Recall)

Julia Lenc

The process: Stage 4

Training. Evaluation (SVM)



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score

# ========== Load and split data ==========
df = pd.read_csv("my_preprocessed_data.csv")
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)

# ========== Train the SVM Classifier ==========
model = SVC()
model.fit(X_train, y_train)

# NOTE: For SVM evaluation, we chose Accuracy and Precision as our evaluation criteria.

# ========== Evaluate on TRAINING set ==========
y_train_pred = model.predict(X_train)
acc_train = accuracy_score(y_train, y_train_pred)
prec_train = precision_score(y_train, y_train_pred, average='weighted')

# ========== Evaluate on TEST set ==========
y_test_pred = model.predict(X_test)
acc_test = accuracy_score(y_test, y_test_pred)
prec_test = precision_score(y_test, y_test_pred, average='weighted')

print("SVM CLASSIFIER PERFORMANCE")
print(f"Training Accuracy:  {acc_train:.2f}")
print(f"Test Accuracy:      {acc_test:.2f}")
print(f"Training Precision: {prec_train:.2f}")
print(f"Test Precision:     {prec_test:.2f}")

# If training scores are much higher than test scores, the model might be OVERFITTING.
# If both scores are low, it might be UNDERFITTING.
```

Julia Lenc

The process: Stage 4

Training. Evaluation (SVR)

n = number of data points

MAE (Mean Absolute Error): $\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_{\mathrm{true},i} - y_{\mathrm{pred},i}|$ Average absolute difference between the predicted and actual values. The most interpretable metric. Use when errors are acceptable as long as they cancel each other. Example: house prices.

MSE (Mean Squared Error): $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{\mathrm{true},i} - y_{\mathrm{pred},i})^2$ Average of the squared difference between the actual and predicted values. Penalizes for large errors. Use when relationship is deterministic, precision is critical, larger errors significantly impact conclusions. Example: R&D, engineering.

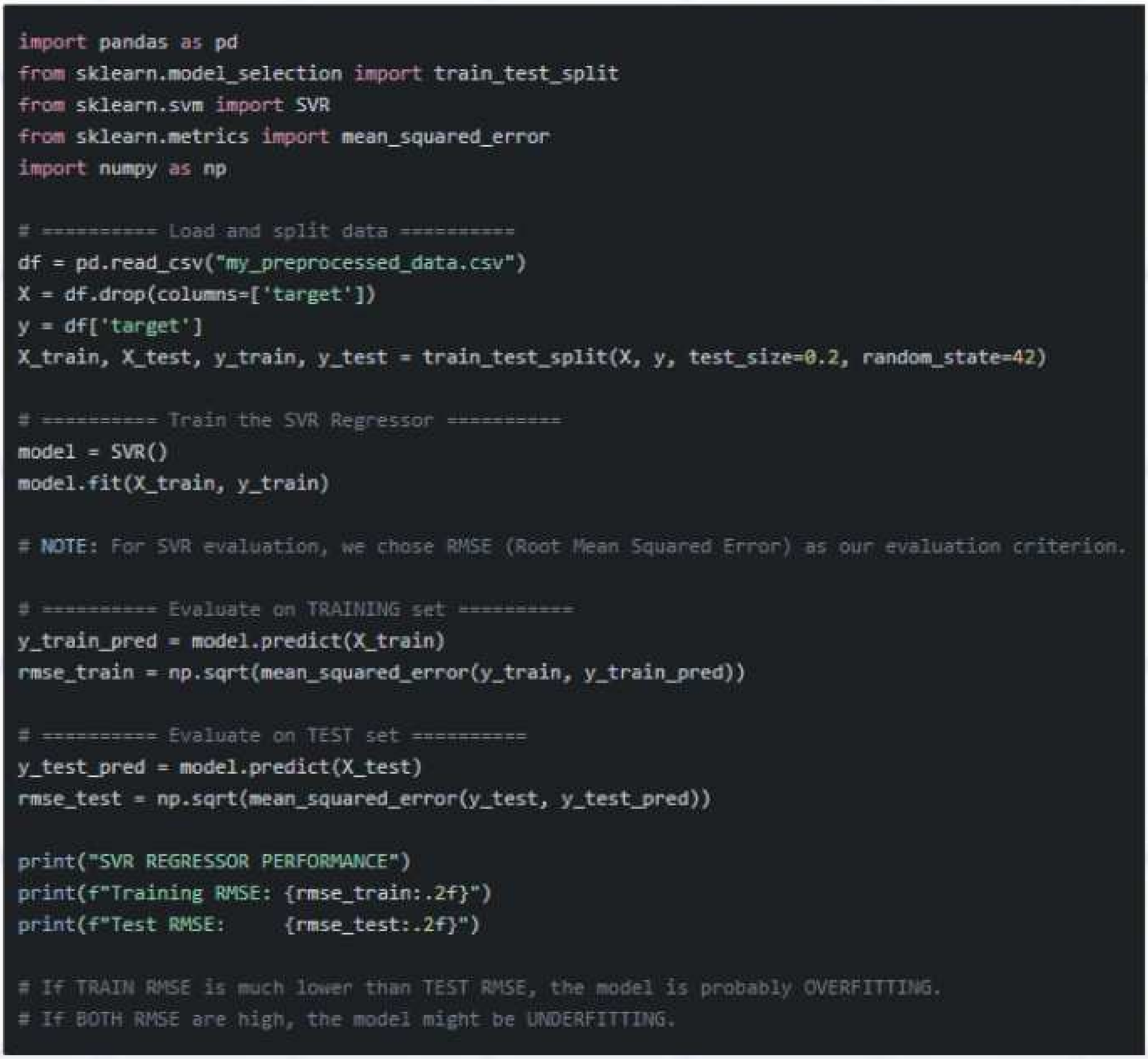
RMSE (Root Mean Squared Error): $\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{\mathrm{true},i} - y_{\mathrm{pred},i})^2}$ like MSE, but expressed in the same units as the target variable. Use when larger errors must be penalized because they lead to system collapse. Example: supply chain, electricity demand forecasting.

DO NOT USE MAPE (Mean Average Percentage Error)!

If your data contains zeros or very small values (consider WMAPE or other alternatives).

The process: Stage 4

Training. Evaluation (SVR)



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np

# ========== Load and split data ==========
df = pd.read_csv("my_preprocessed_data.csv")
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)

# ========== Train the SVR Regressor ==========
model = SVR()
model.fit(X_train, y_train)

# NOTE: For SVR evaluation, we chose RMSE (Root Mean Squared Error) as our evaluation criterion.

# ========== Evaluate on TRAINING set ==========
y_train_pred = model.predict(X_train)
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))

# ========== Evaluate on TEST set ==========
y_test_pred = model.predict(X_test)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("SVR REGRESSOR PERFORMANCE")
print(f"Training RMSE: {rmse_train:.2f}")
print(f"Test RMSE:     {rmse_test:.2f}")

# If TRAIN RMSE is much lower than TEST RMSE, the model is probably OVERFITTING.
# If BOTH RMSE are high, the model might be UNDERFITTING.
```

Julia Lenc

The process: Stage 5

If evaluation results are poor (both SVM and SVR)

Overfitting remedies: if high training and low test set performance

1. Lower C (more regularization)
2. Try a simpler kernel (e.g., use 'linear' instead of 'rbf')
3. Reduce the number of input features (feature selection)
4. Reduce model complexity (for SVR, could also decrease epsilon)
5. Increase regularization (gamma for some kernels)

Undefitting remedies: if low training and test set performance

1. Increase C (less regularization)
2. Try a more complex kernel (e.g., 'rbf' or 'poly')
3. Add features (feature engineering)
4. Reduce regularization (gamma or epsilon for SVR)

Hyperparameters tuning and kernel tricks

1. Use GridSearchCV or RandomizedSearchCV to find optimal parameters:

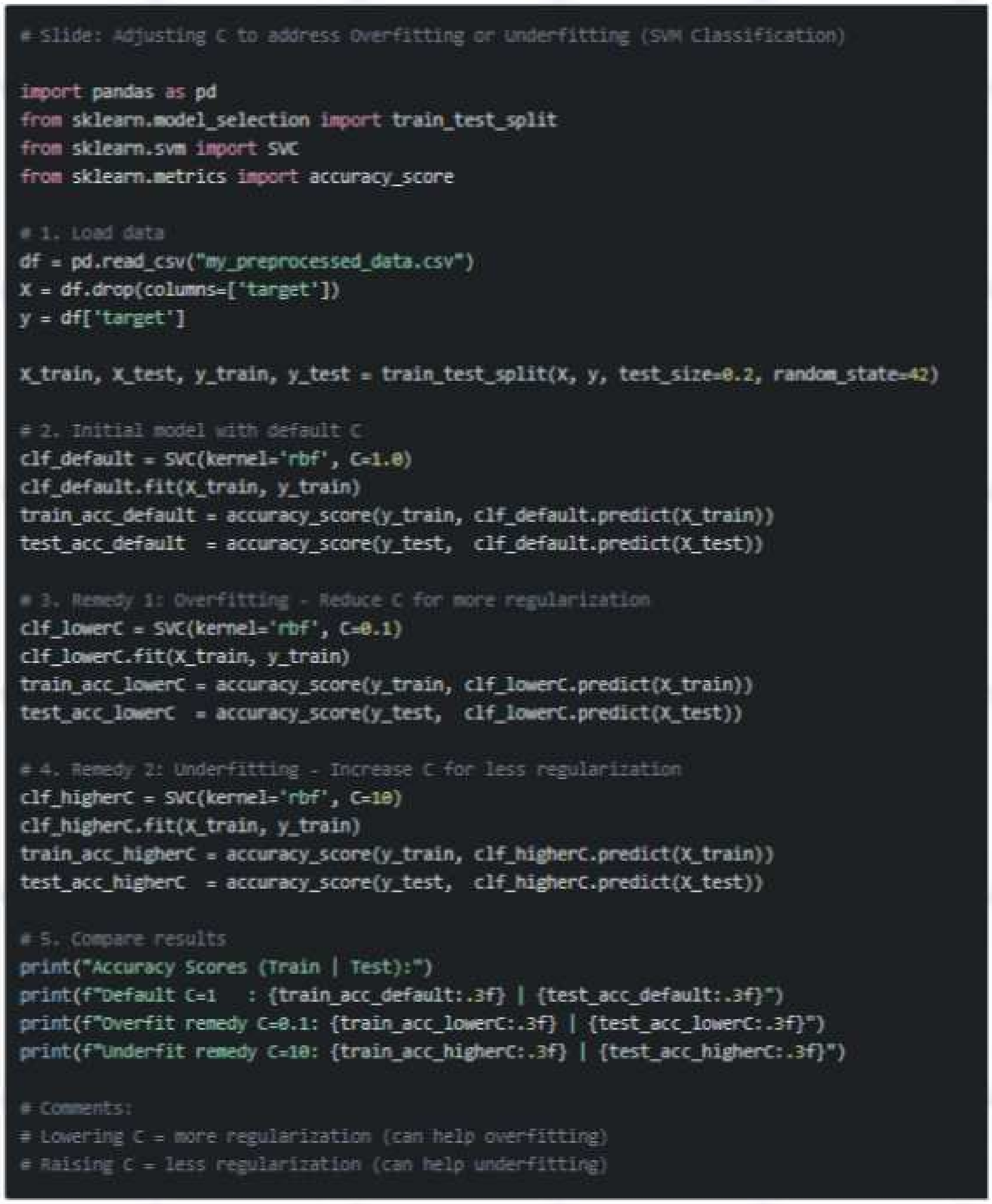
   C, kernel, gamma (and epsilon for SVR)

2. Use cross-validation to avoid overfitting on test set.
3. Re-traing the model with different kernel (if EDA shows it's applicable)

Julia Lenc

The process: Stage 5.1

Adjusting C up or down (SVM)

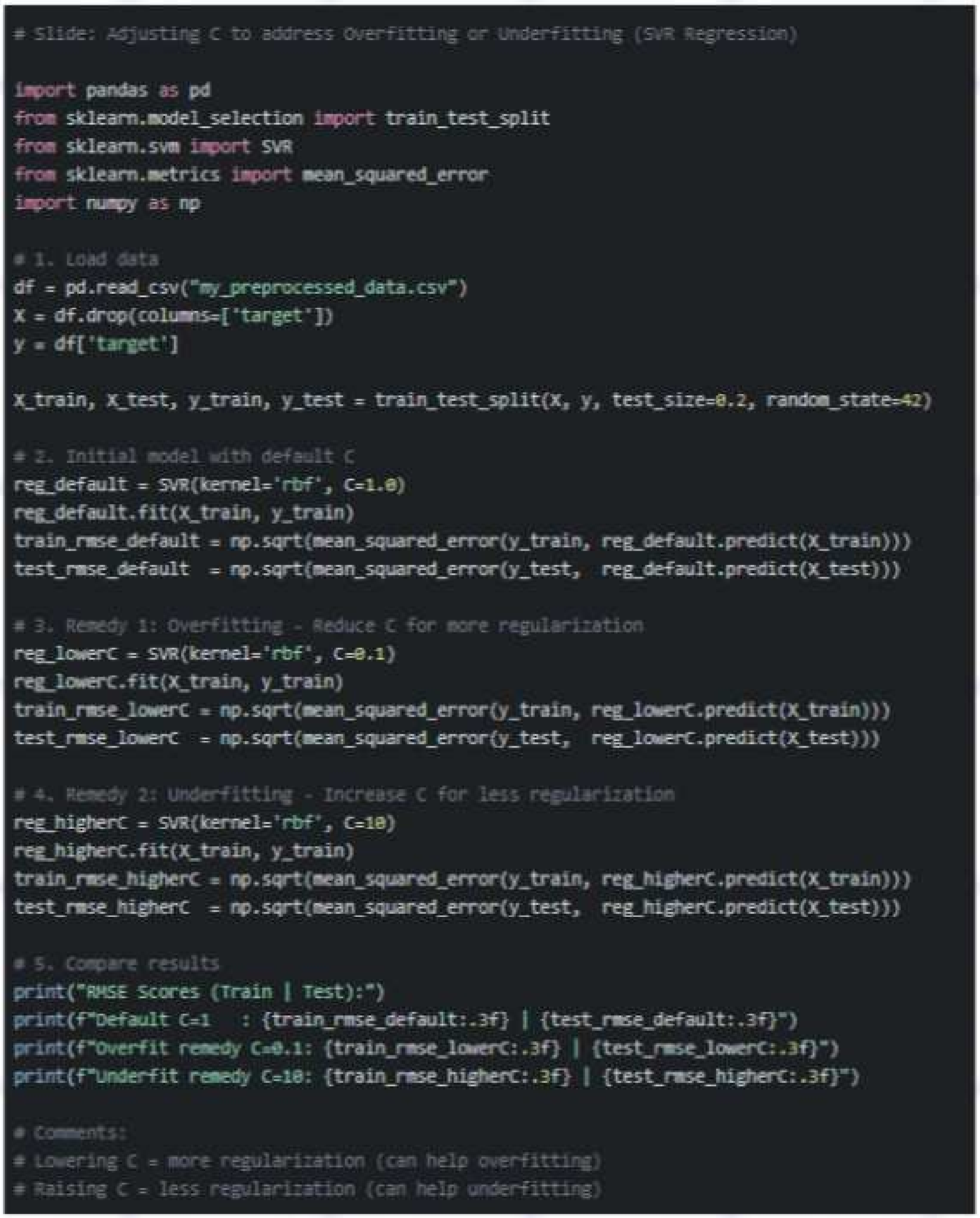


```python
# Slide: Adjusting C to address Overfitting or Underfitting (SVM Classification)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# 1. Load data
df = pd.read_csv("my_preprocessed_data.csv")
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)

# 2. Initial model with default C
clf_default = SVC(kernel='rbf', C=1.0)
clf_default.fit(X_train, y_train)
train_acc_default = accuracy_score(y_train, clf_default.predict(X_train))
test_acc_default  = accuracy_score(y_test,  clf_default.predict(X_test))

# 3. Remedy 1: Overfitting - Reduce C for more regularization
clf_lowerC = SVC(kernel='rbf', C=0.1)
clf_lowerC.fit(X_train, y_train)
train_acc_lowerC = accuracy_score(y_train, clf_lowerC.predict(X_train))
test_acc_lowerC  = accuracy_score(y_test,  clf_lowerC.predict(X_test))

# 4. Remedy 2: Underfitting - Increase C for less regularization
clf_higherC = SVC(kernel='rbf', C=10)
clf_higherC.fit(X_train, y_train)
train_acc_higherC = accuracy_score(y_train, clf_higherC.predict(X_train))
test_acc_higherC  = accuracy_score(y_test,  clf_higherC.predict(X_test))

# 5. Compare results
print("Accuracy Scores (Train | Test):")
print(f"Default C=1    : {train_acc_default:.3f} | {test_acc_default:.3f}")
print(f"Overfit remedy C=0.1: {train_acc_lowerC:.3f} | {test_acc_lowerC:.3f}")
print(f"Underfit remedy C=10: {train_acc_higherC:.3f} | {test_acc_higherC:.3f}")

# Comments:
# Lowering C = more regularization (can help overfitting)
# Raising C = less regularization (can help underfitting)
```

Julia Lenc

The process: Stage 5.1

Adjusting C up or down (SVR)

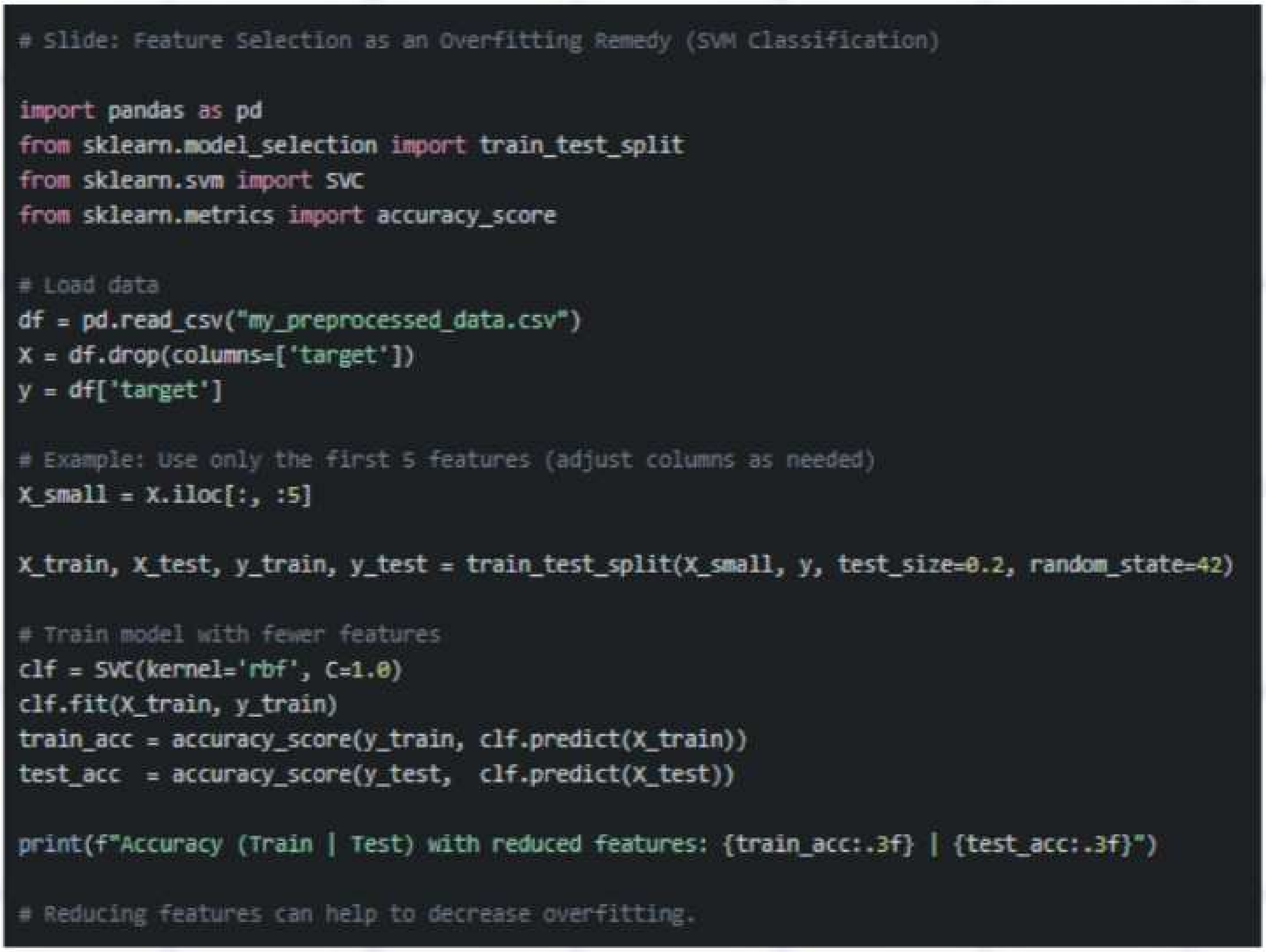


```
# Slide: Adjusting C to address Overfitting or Underfitting (SVR Regression)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np

# 1. Load data
df = pd.read_csv("my_preprocessed_data.csv")
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)

# 2. Initial model with default C
reg_default = SVR(kernel='rbf', C=1.0)
reg_default.fit(X_train, y_train)
train_rmse_default = np.sqrt(mean_squared_error(y_train, reg_default.predict(X_train)))
test_rmse_default  = np.sqrt(mean_squared_error(y_test,  reg_default.predict(X_test)))

# 3. Remedy 1: Overfitting - Reduce C for more regularization
reg_lowerC = SVR(kernel='rbf', C=0.1)
reg_lowerC.fit(X_train, y_train)
train_rmse_lowerC = np.sqrt(mean_squared_error(y_train, reg_lowerC.predict(X_train)))
test_rmse_lowerC  = np.sqrt(mean_squared_error(y_test,  reg_lowerC.predict(X_test)))

# 4. Remedy 2: Underfitting - Increase C for less regularization
reg_higherC = SVR(kernel='rbf', C=10)
reg_higherC.fit(X_train, y_train)
train_rmse_higherC = np.sqrt(mean_squared_error(y_train, reg_higherC.predict(X_train)))
test_rmse_higherC  = np.sqrt(mean_squared_error(y_test,  reg_higherC.predict(X_test)))

# 5. Compare results
print("RMSE Scores (Train | Test):")
print(f"Default C=1    : {train_rmse_default:.3f} | {test_rmse_default:.3f}")
print(f"Overfit remedy C=0.1: {train_rmse_lowerC:.3f} | {test_rmse_lowerC:.3f}")
print(f"Underfit remedy C=10: {train_rmse_higherC:.3f} | {test_rmse_higherC:.3f}")

# Comments:
# Lowering C = more regularization (can help overfitting)
# Raising C = less regularization (can help underfitting)
```

Julia Lenc

The process: Stage 5.2

Reducing features (SVM)

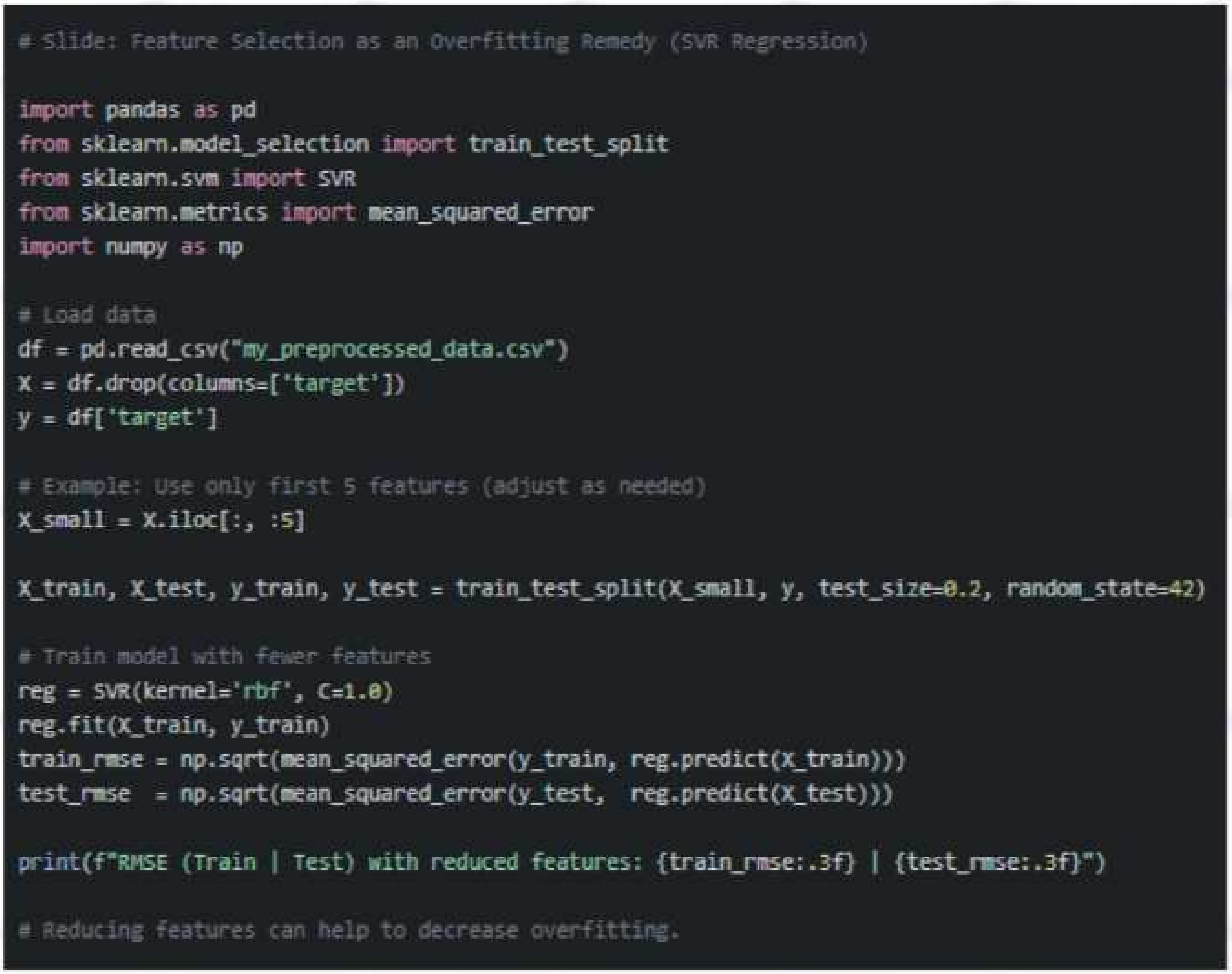


```python
# Slide: Feature Selection as an Overfitting Remedy (SVM Classification)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load data
df = pd.read_csv("my_preprocessed_data.csv")
X = df.drop(columns=['target'])
y = df['target']

# Example: Use only the first 5 features (adjust columns as needed)
X_small = X.iloc[:, :5]

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

# Train model with fewer features
clf = SVC(kernel='rbf', C=1.0)
clf.fit(X_train, y_train)
train_acc = accuracy_score(y_train, clf.predict(X_train))
test_acc  = accuracy_score(y_test,  clf.predict(X_test))

print(f"Accuracy (Train | Test) with reduced features: {train_acc:.3f} | {test_acc:.3f}")

# Reducing features can help to decrease overfitting.
```

Julia Lenc

The process: Stage 5.2

Reducing features (SVR)

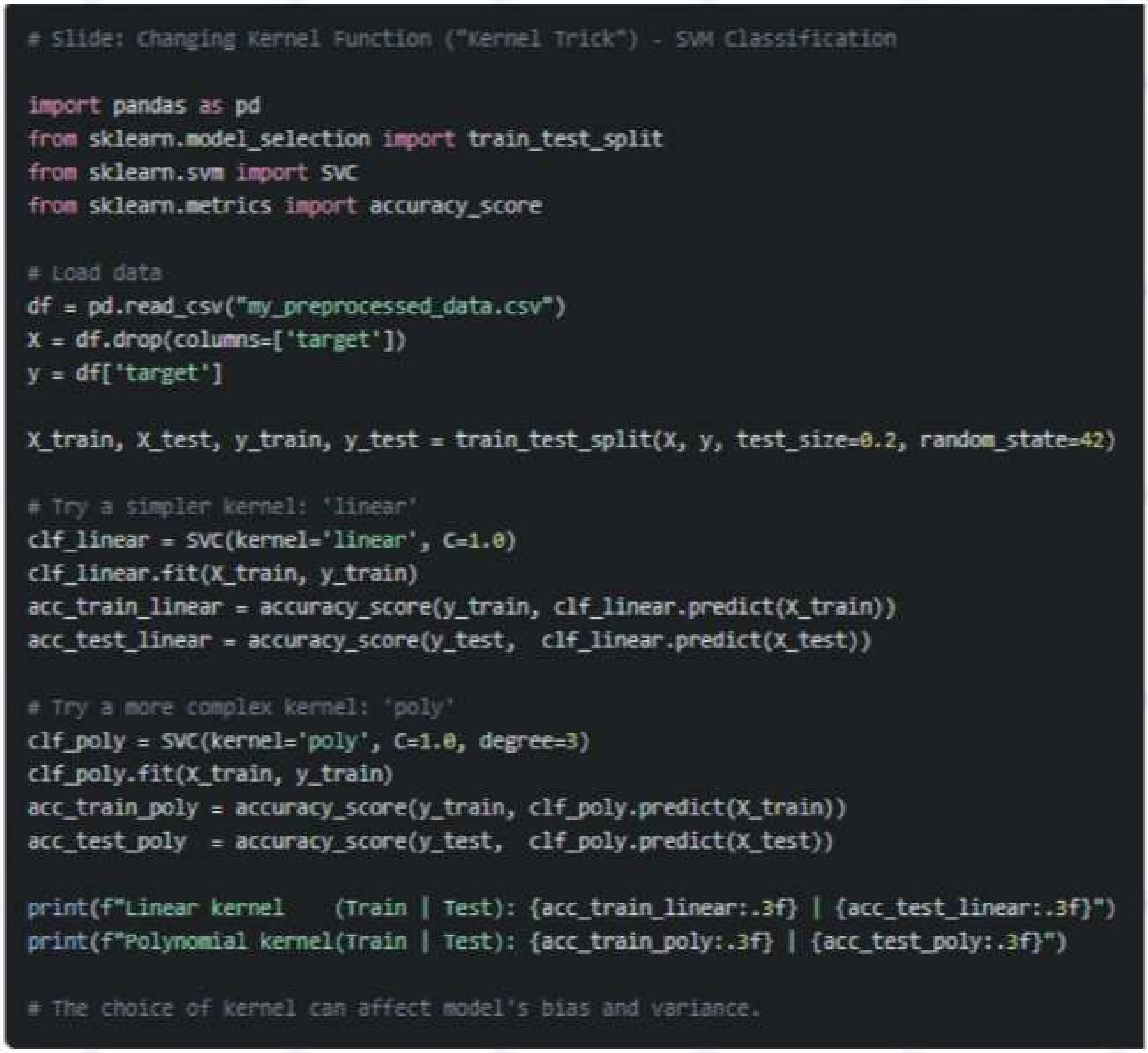


```python
# Slide: Feature Selection as an Overfitting Remedy (SVR Regression)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np

# Load data
df = pd.read_csv("my_preprocessed_data.csv")
X = df.drop(columns=['target'])
y = df['target']

# Example: Use only first 5 features (adjust as needed)
X_small = X.iloc[:, :5]

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

# Train model with fewer features
reg = SVR(kernel='rbf', C=1.0)
reg.fit(X_train, y_train)
train_rmse = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
test_rmse  = np.sqrt(mean_squared_error(y_test,  reg.predict(X_test)))

print(f"RMSE (Train | Test) with reduced features: {train_rmse:.3f} | {test_rmse:.3f}")

# Reducing features can help to decrease overfitting.
```

Julia Lenc

The process: Stage 5.3

Changing kernel function (SVM)

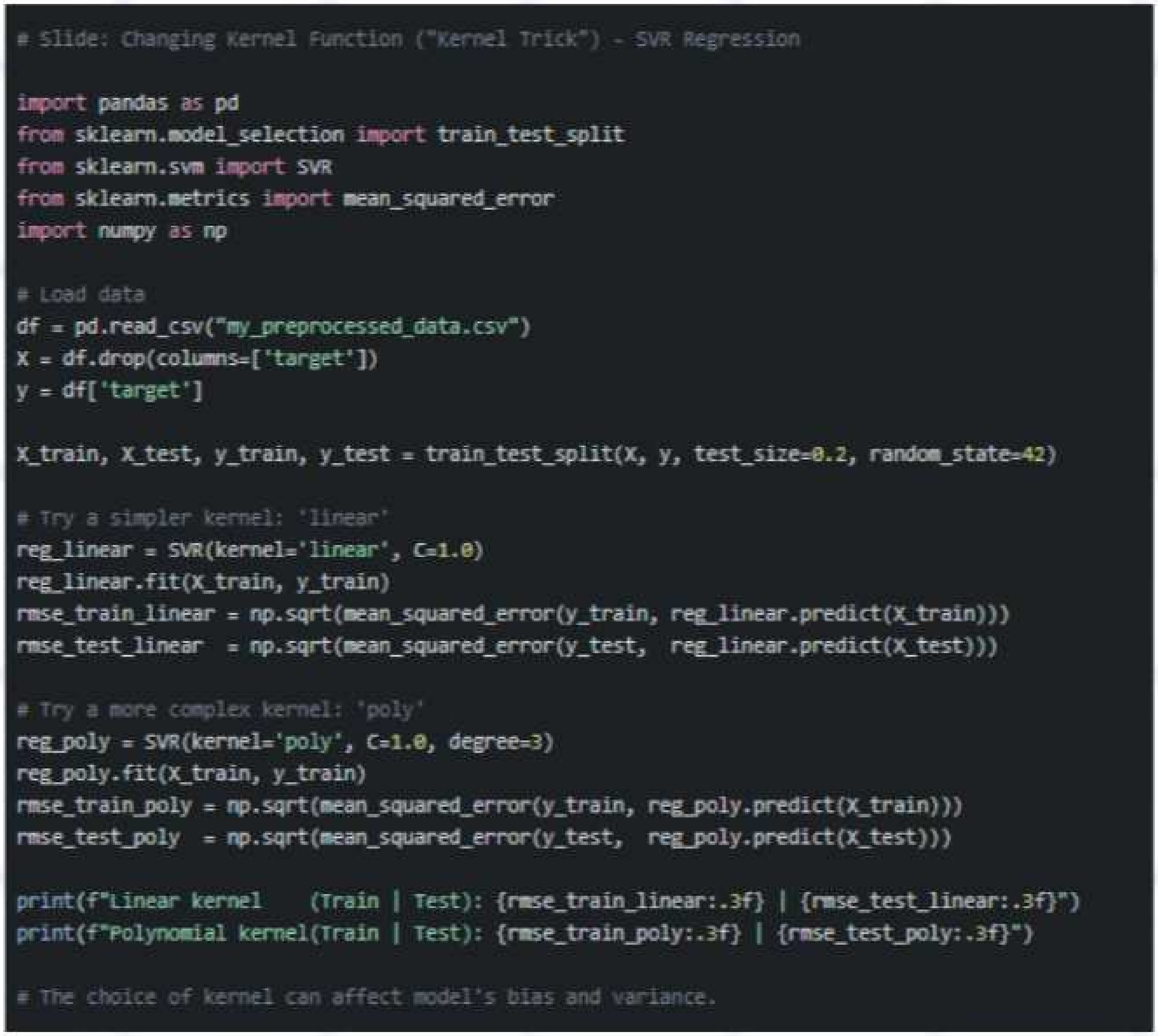


```python
# Slide: Changing Kernel Function ("Kernel Trick") - SVM Classification

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load data
df = pd.read_csv("my_preprocessed_data.csv")
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)

# Try a simpler kernel: 'linear'
clf_linear = SVC(kernel='linear', C=1.0)
clf_linear.fit(X_train, y_train)
acc_train_linear = accuracy_score(y_train, clf_linear.predict(X_train))
acc_test_linear = accuracy_score(y_test,  clf_linear.predict(X_test))

# Try a more complex kernel: 'poly'
clf_poly = SVC(kernel='poly', C=1.0, degree=3)
clf_poly.fit(X_train, y_train)
acc_train_poly = accuracy_score(y_train, clf_poly.predict(X_train))
acc_test_poly  = accuracy_score(y_test,  clf_poly.predict(X_test))

print(f"Linear kernel    (Train | Test): {acc_train_linear:.3f} | {acc_test_linear:.3f}")
print(f"Polynomial kernel(Train | Test): {acc_train_poly:.3f} | {acc_test_poly:.3f}")

# The choice of kernel can affect model's bias and variance.
```

Julia Lenc

The process: Stage 5.3

Changing kernel function (SVR)

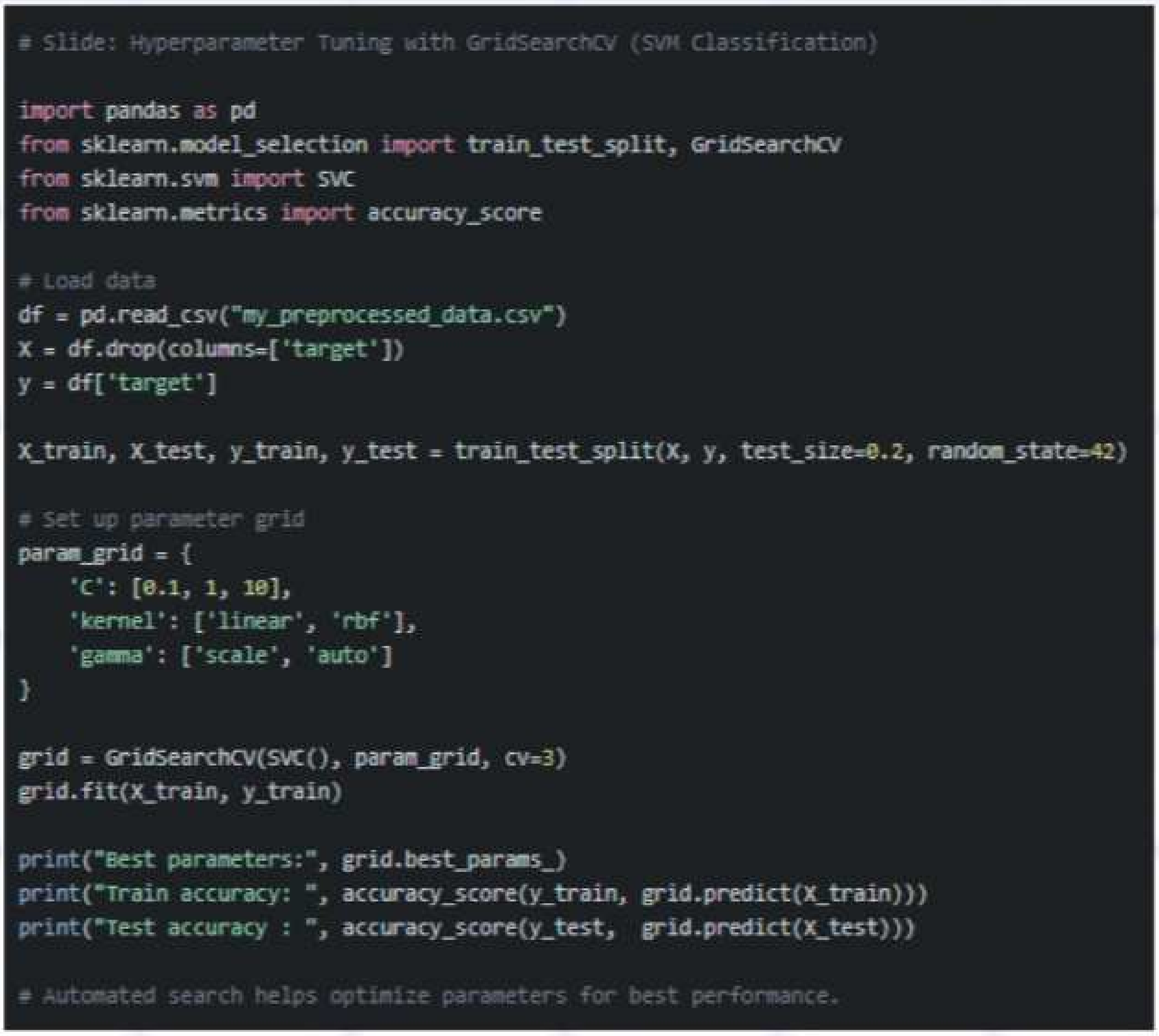


```python
# Slide: Changing Kernel Function ("Kernel Trick") - SVR Regression

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np

# Load data
df = pd.read_csv("my_preprocessed_data.csv")
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)

# Try a simpler kernel: 'linear'
reg_linear = SVR(kernel='linear', C=1.0)
reg_linear.fit(X_train, y_train)
rmse_train_linear = np.sqrt(mean_squared_error(y_train, reg_linear.predict(X_train)))
rmse_test_linear  = np.sqrt(mean_squared_error(y_test,  reg_linear.predict(X_test)))

# Try a more complex kernel: 'poly'
reg_poly = SVR(kernel='poly', C=1.0, degree=3)
reg_poly.fit(X_train, y_train)
rmse_train_poly = np.sqrt(mean_squared_error(y_train, reg_poly.predict(X_train)))
rmse_test_poly  = np.sqrt(mean_squared_error(y_test,  reg_poly.predict(X_test)))

print(f"Linear kernel    (Train | Test): {rmse_train_linear:.3f} | {rmse_test_linear:.3f}")
print(f"Polynomial kernel(Train | Test): {rmse_train_poly:.3f} | {rmse_test_poly:.3f}")

# The choice of kernel can affect model's bias and variance.
```

Julia Lenc

The process: Stage 5.4

Automated Hyperparameter Tuning with GridSearchCV (SVM)

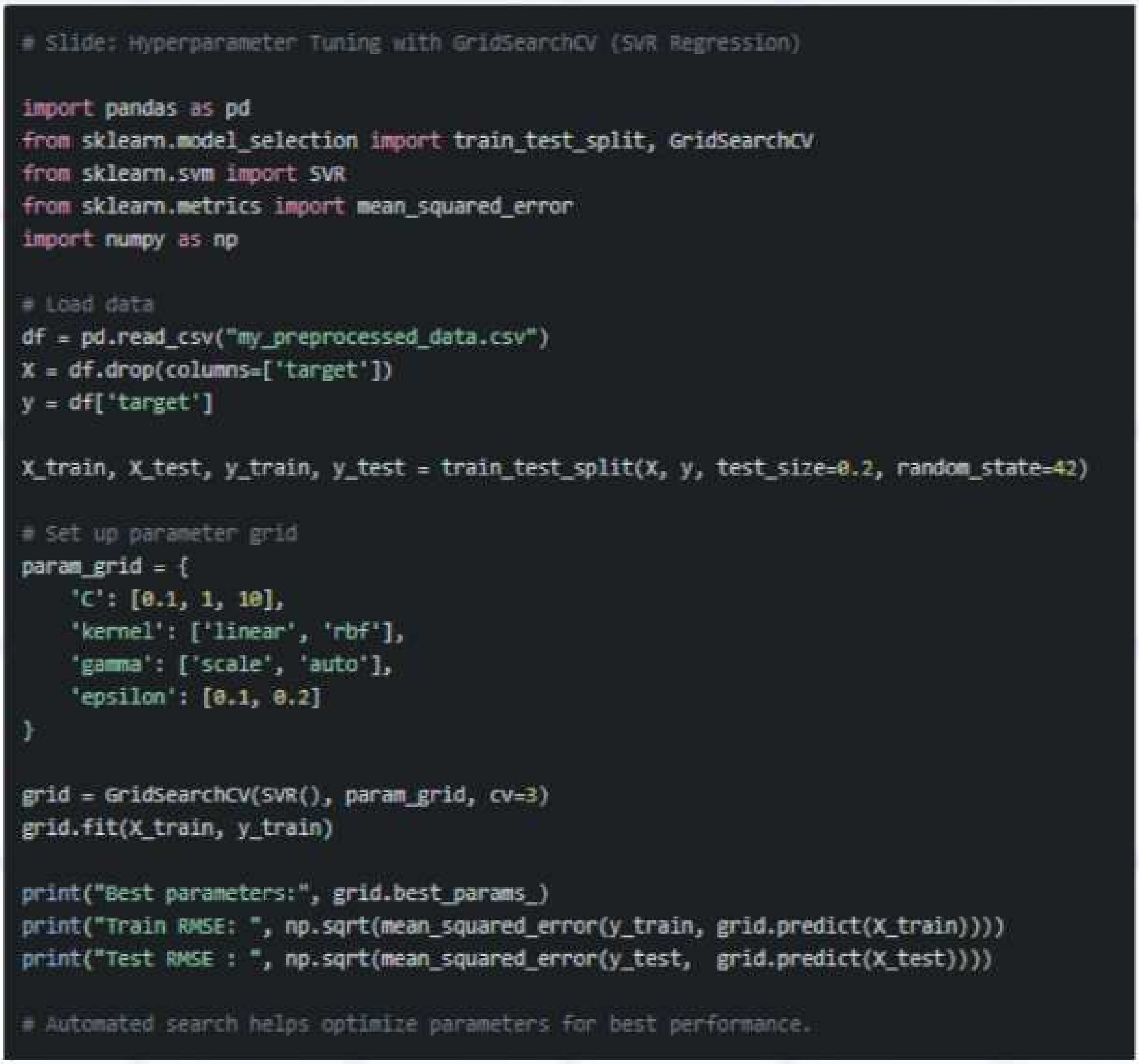


```python
# Slide: Hyperparameter Tuning with GridSearchCV (SVM Classification)

import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load data
df = pd.read_csv("my_preprocessed_data.csv")
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)

# Set up parameter grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid = GridSearchCV(SVC(), param_grid, cv=3)
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
print("Train accuracy: ", accuracy_score(y_train, grid.predict(X_train)))
print("Test accuracy : ", accuracy_score(y_test,  grid.predict(X_test)))

# Automated search helps optimize parameters for best performance.
```

Julia Lenc

The process: Stage 5.4

Automated Hyperparameter Tuning with GridSearchCV (SVR)



```python
# Slide: Hyperparameter Tuning with GridSearchCV (SVR Regression)

import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np

# Load data
df = pd.read_csv("my_preprocessed_data.csv")
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)

# Set up parameter grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto'],
    'epsilon': [0.1, 0.2]
}

grid = GridSearchCV(SVR(), param_grid, cv=3)
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
print("Train RMSE: ", np.sqrt(mean_squared_error(y_train, grid.predict(X_train))))
print("Test RMSE : ", np.sqrt(mean_squared_error(y_test,  grid.predict(X_test))))

# Automated search helps optimize parameters for best performance.
```

Julia Lenc

Did you find it useful?

Save
Share
Follow

Analytics, Market Research, Machine Learning and AI

Julia Lenc