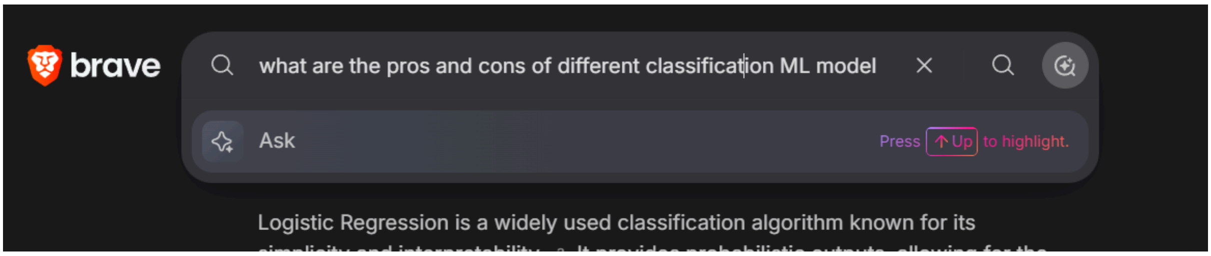
Original Kaggle DataSet
https://www.kaggle.com/datasets/saife245/english-premier-league/data?select=2021-2022.csv

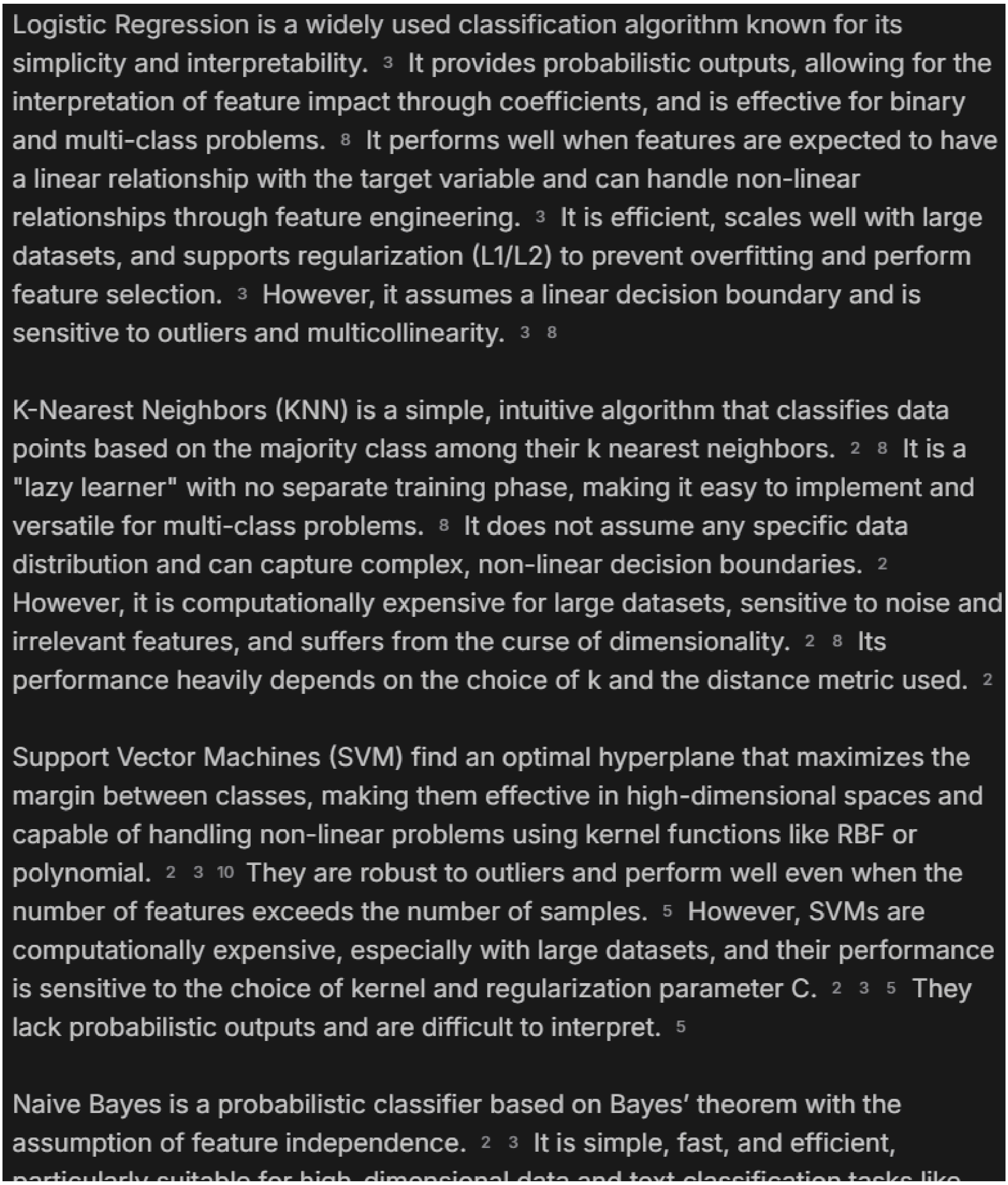
Example use of dataset from Kaggle
https://www.kaggle.com/code/saife245/football-match-prediction#Applying-the-Logistic-Regression

We used built-in browser AI responses from regular browser searches shown below as well as responses; we also used generative ai such as chat gpt to help with the notebook creation; We also used gemini and nano banana to generate some icons for our presentation



what are the pros and cons of different classification ML model

Logistic Regression is a widely used classification algorithm known for its

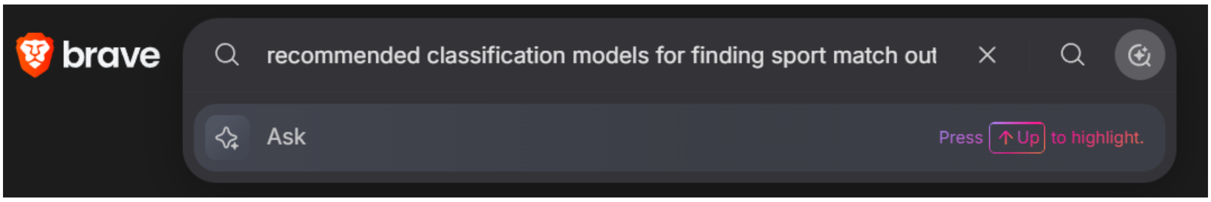


Logistic Regression is a widely used classification algorithm known for its simplicity and interpretability. [3] It provides probabilistic outputs, allowing for the interpretation of feature impact through coefficients, and is effective for binary and multi-class problems. [8] It performs well when features are expected to have a linear relationship with the target variable and can handle non-linear relationships through feature engineering. [3] It is efficient, scales well with large datasets, and supports regularization (L1/L2) to prevent overfitting and perform feature selection. [3] However, it assumes a linear decision boundary and is sensitive to outliers and multicollinearity. [3] [8]

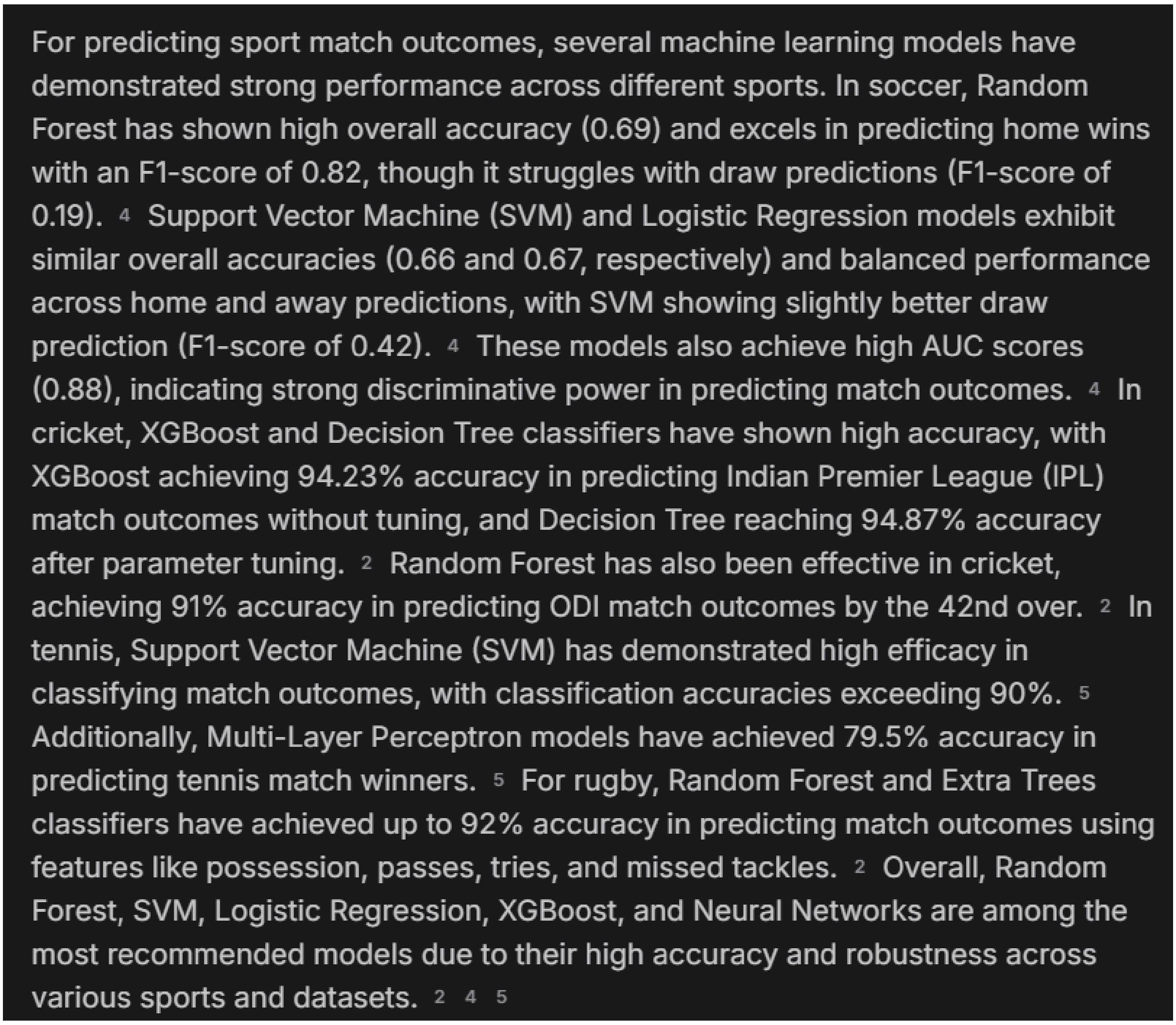
K-Nearest Neighbors (KNN) is a simple, intuitive algorithm that classifies data points based on the majority class among their k nearest neighbors. [2] [8] It is a "lazy learner" with no separate training phase, making it easy to implement and versatile for multi-class problems. [8] It does not assume any specific data distribution and can capture complex, non-linear decision boundaries. [2] However, it is computationally expensive for large datasets, sensitive to noise and irrelevant features, and suffers from the curse of dimensionality. [2] [8] Its performance heavily depends on the choice of k and the distance metric used. [2]

Support Vector Machines (SVM) find an optimal hyperplane that maximizes the margin between classes, making them effective in high-dimensional spaces and capable of handling non-linear problems using kernel functions like RBF or polynomial. [2] [3] [10] They are robust to outliers and perform well even when the number of features exceeds the number of samples. [5] However, SVMs are computationally expensive, especially with large datasets, and their performance is sensitive to the choice of kernel and regularization parameter C. [2] [3] [5] They lack probabilistic outputs and are difficult to interpret. [5]

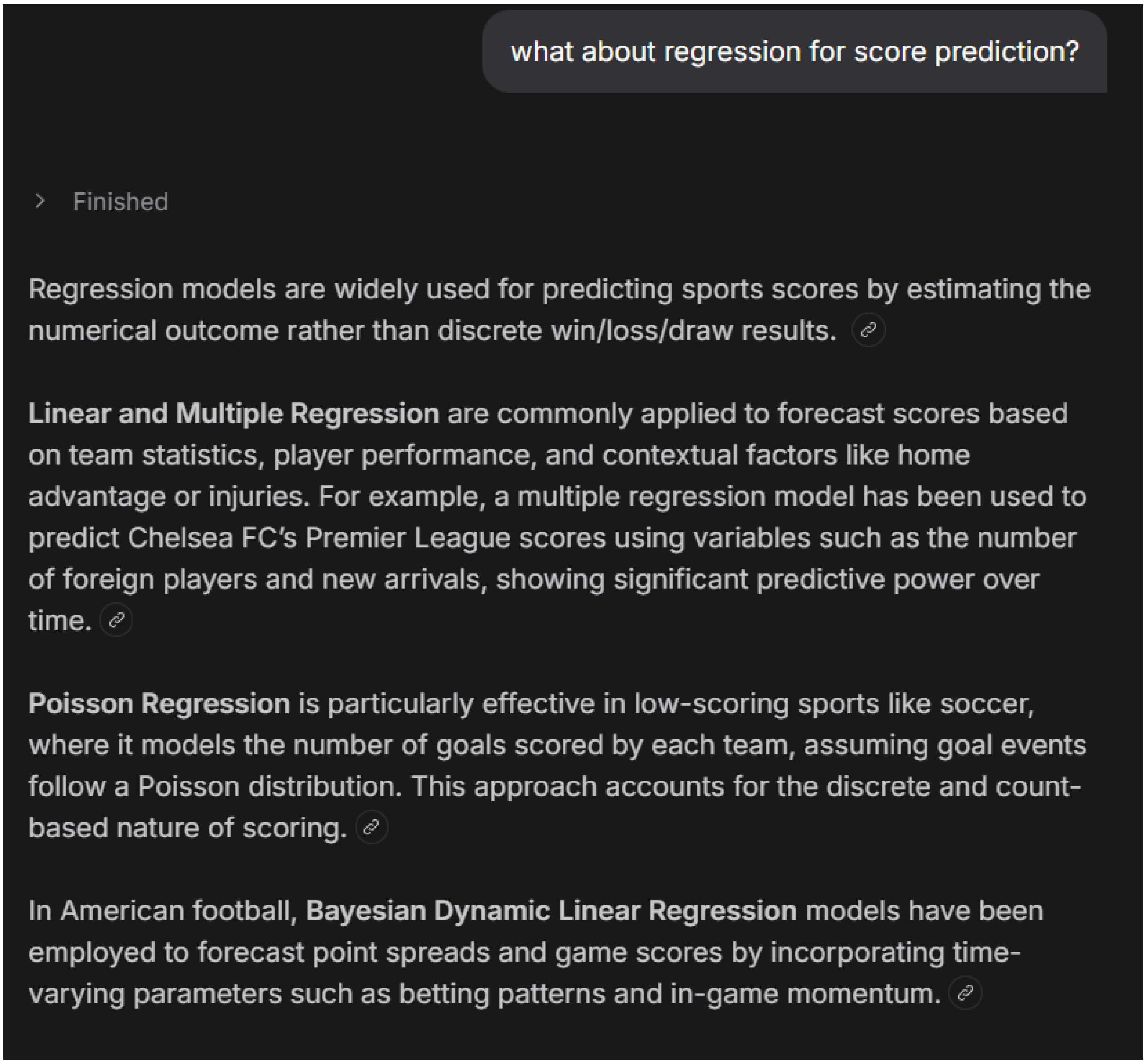
Naive Bayes is a probabilistic classifier based on Bayes’ theorem with the assumption of feature independence. [2] [3] It is simple, fast, and efficient, particularly suitable for high-dimensional data and text classification tasks like



recommended classification models for finding sport match out



For predicting sport match outcomes, several machine learning models have demonstrated strong performance across different sports. In soccer, Random Forest has shown high overall accuracy (0.69) and excels in predicting home wins with an F1-score of 0.82, though it struggles with draw predictions (F1-score of 0.19). [4] Support Vector Machine (SVM) and Logistic Regression models exhibit similar overall accuracies (0.66 and 0.67, respectively) and balanced performance across home and away predictions, with SVM showing slightly better draw prediction (F1-score of 0.42). [4] These models also achieve high AUC scores (0.88), indicating strong discriminative power in predicting match outcomes. [4] In cricket, XGBoost and Decision Tree classifiers have shown high accuracy, with XGBoost achieving 94.23% accuracy in predicting Indian Premier League (IPL) match outcomes without tuning, and Decision Tree reaching 94.87% accuracy after parameter tuning. [2] Random Forest has also been effective in cricket, achieving 91% accuracy in predicting ODI match outcomes by the 42nd over. [2] In tennis, Support Vector Machine (SVM) has demonstrated high efficacy in classifying match outcomes, with classification accuracies exceeding 90%. [5] Additionally, Multi-Layer Perceptron models have achieved 79.5% accuracy in predicting tennis match winners. [5] For rugby, Random Forest and Extra Trees classifiers have achieved up to 92% accuracy in predicting match outcomes using features like possession, passes, tries, and missed tackles. [2] Overall, Random Forest, SVM, Logistic Regression, XGBoost, and Neural Networks are among the most recommended models due to their high accuracy and robustness across various sports and datasets. [2] [4] [5]



what about regression for score prediction?

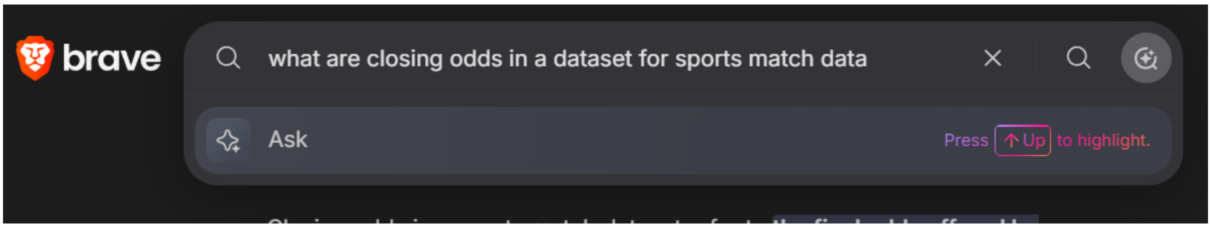
Finished

Regression models are widely used for predicting sports scores by estimating the numerical outcome rather than discrete win/loss/draw results.

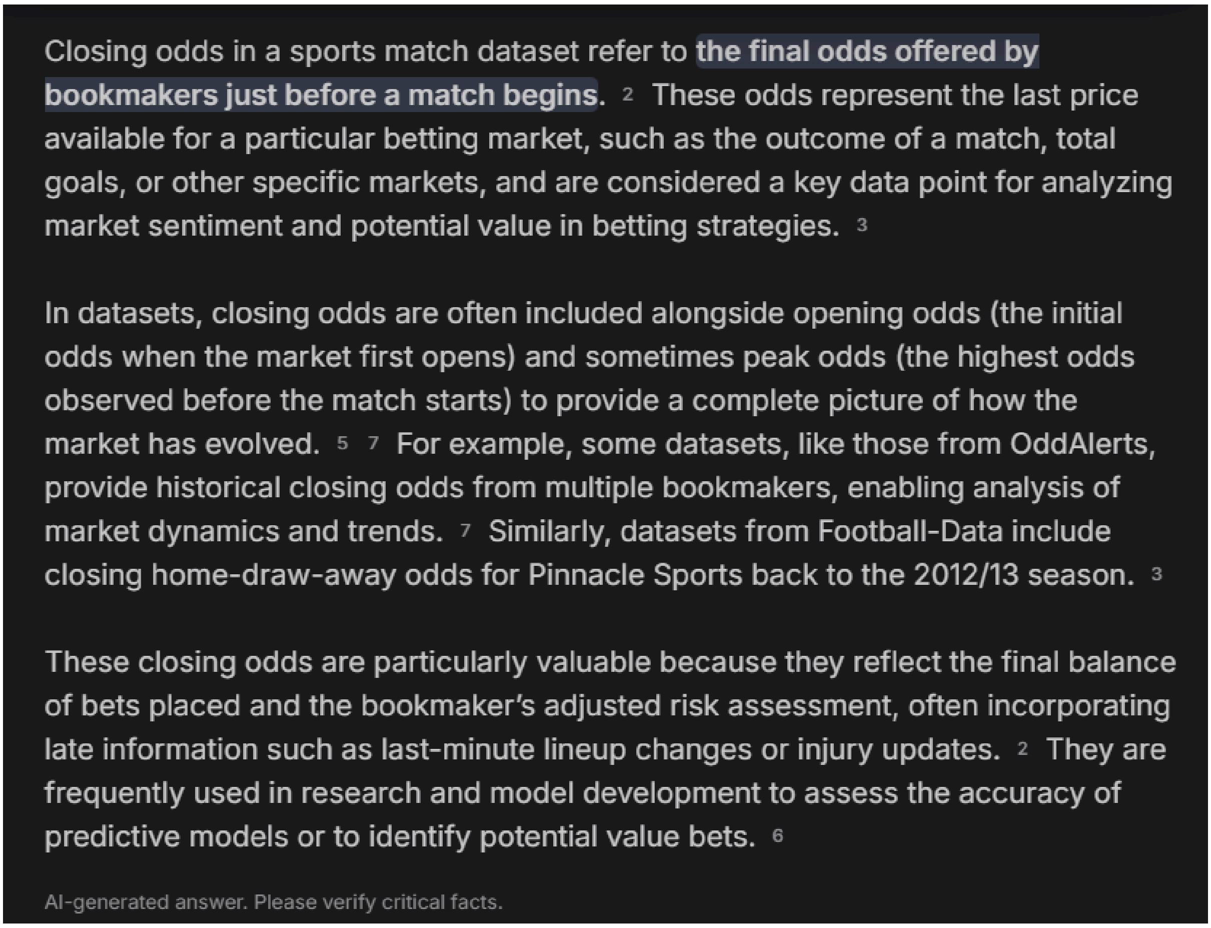
Linear and Multiple Regression are commonly applied to forecast scores based on team statistics, player performance, and contextual factors like home advantage or injuries. For example, a multiple regression model has been used to predict Chelsea FC's Premier League scores using variables such as the number of foreign players and new arrivals, showing significant predictive power over time.

Poisson Regression is particularly effective in low-scoring sports like soccer, where it models the number of goals scored by each team, assuming goal events follow a Poisson distribution. This approach accounts for the discrete and count-based nature of scoring.

In American football, **Bayesian Dynamic Linear Regression** models have been employed to forecast point spreads and game scores by incorporating time-varying parameters such as betting patterns and in-game momentum.



what are closing odds in a dataset for sports match data

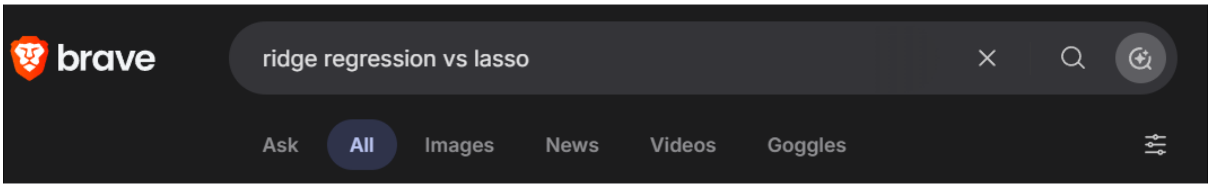


Closing odds in a sports match dataset refer to **the final odds offered by bookmakers just before a match begins**. [2] These odds represent the last price available for a particular betting market, such as the outcome of a match, total goals, or other specific markets, and are considered a key data point for analyzing market sentiment and potential value in betting strategies. [3]

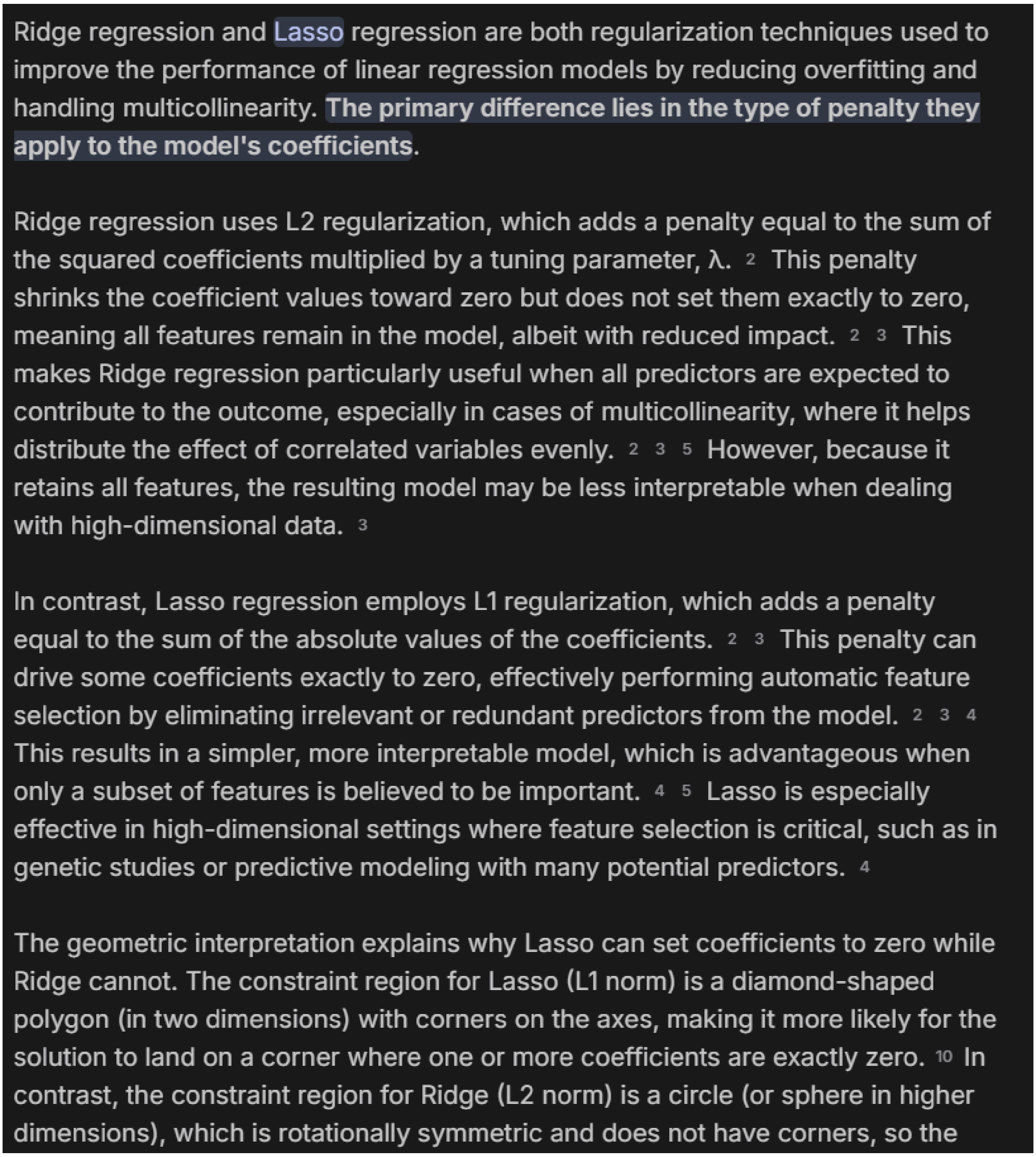
In datasets, closing odds are often included alongside opening odds (the initial odds when the market first opens) and sometimes peak odds (the highest odds observed before the match starts) to provide a complete picture of how the market has evolved. [5] [7] For example, some datasets, like those from OddAlerts, provide historical closing odds from multiple bookmakers, enabling analysis of market dynamics and trends. [7] Similarly, datasets from Football-Data include closing home-draw-away odds for Pinnacle Sports back to the 2012/13 season. [3]

These closing odds are particularly valuable because they reflect the final balance of bets placed and the bookmaker's adjusted risk assessment, often incorporating late information such as last-minute lineup changes or injury updates. [2] They are frequently used in research and model development to assess the accuracy of predictive models or to identify potential value bets. [6]

AI-generated answer. Please verify critical facts.



ridge regression vs lasso

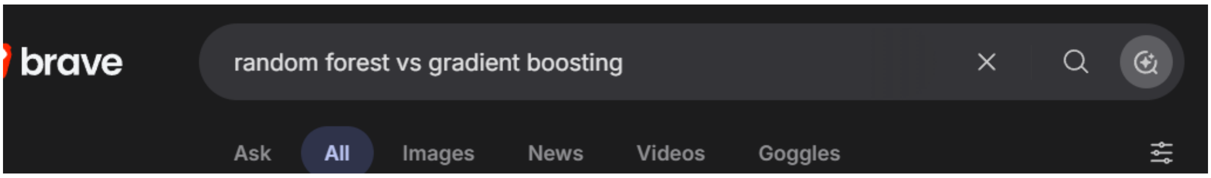


Ridge regression and Lasso regression are both regularization techniques used to improve the performance of linear regression models by reducing overfitting and handling multicollinearity. **The primary difference lies in the type of penalty they apply to the model's coefficients**.

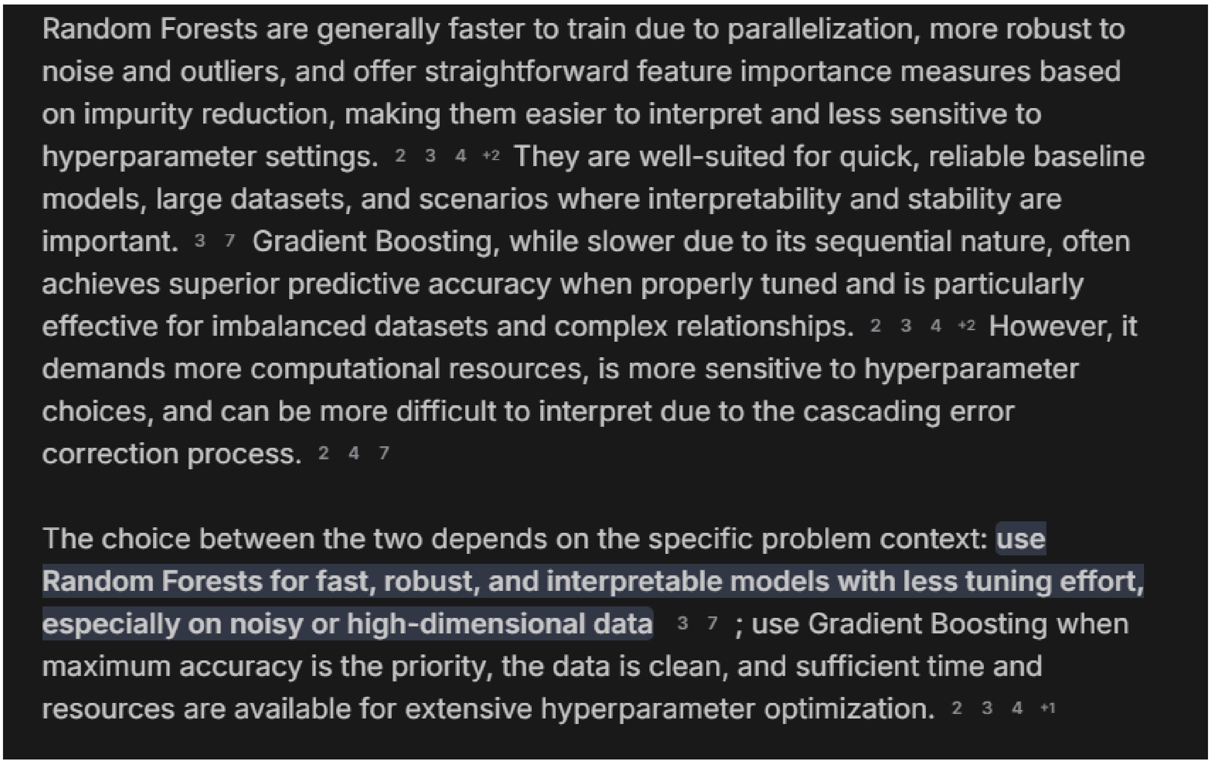
Ridge regression uses L2 regularization, which adds a penalty equal to the sum of the squared coefficients multiplied by a tuning parameter, λ. [2] This penalty shrinks the coefficient values toward zero but does not set them exactly to zero, meaning all features remain in the model, albeit with reduced impact. [2] [3] This makes Ridge regression particularly useful when all predictors are expected to contribute to the outcome, especially in cases of multicollinearity, where it helps distribute the effect of correlated variables evenly. [2] [3] [5] However, because it retains all features, the resulting model may be less interpretable when dealing with high-dimensional data. [3]

In contrast, Lasso regression employs L1 regularization, which adds a penalty equal to the sum of the absolute values of the coefficients. [2] [3] This penalty can drive some coefficients exactly to zero, effectively performing automatic feature selection by eliminating irrelevant or redundant predictors from the model. [2] [3] [4] This results in a simpler, more interpretable model, which is advantageous when only a subset of features is believed to be important. [4] [5] Lasso is especially effective in high-dimensional settings where feature selection is critical, such as in genetic studies or predictive modeling with many potential predictors. [4]

The geometric interpretation explains why Lasso can set coefficients to zero while Ridge cannot. The constraint region for Lasso (L1 norm) is a diamond-shaped polygon (in two dimensions) with corners on the axes, making it more likely for the solution to land on a corner where one or more coefficients are exactly zero. [10] In contrast, the constraint region for Ridge (L2 norm) is a circle (or sphere in higher dimensions), which is rotationally symmetric and does not have corners, so the

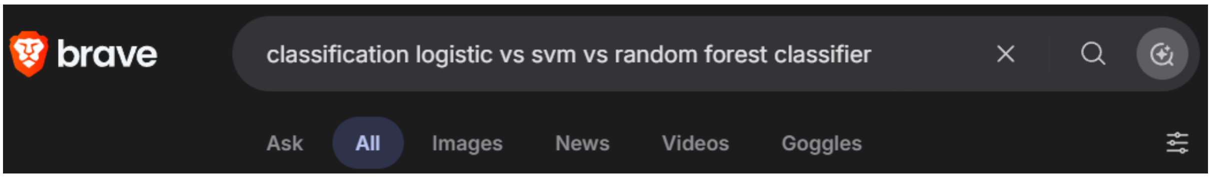


random forest vs gradient boosting

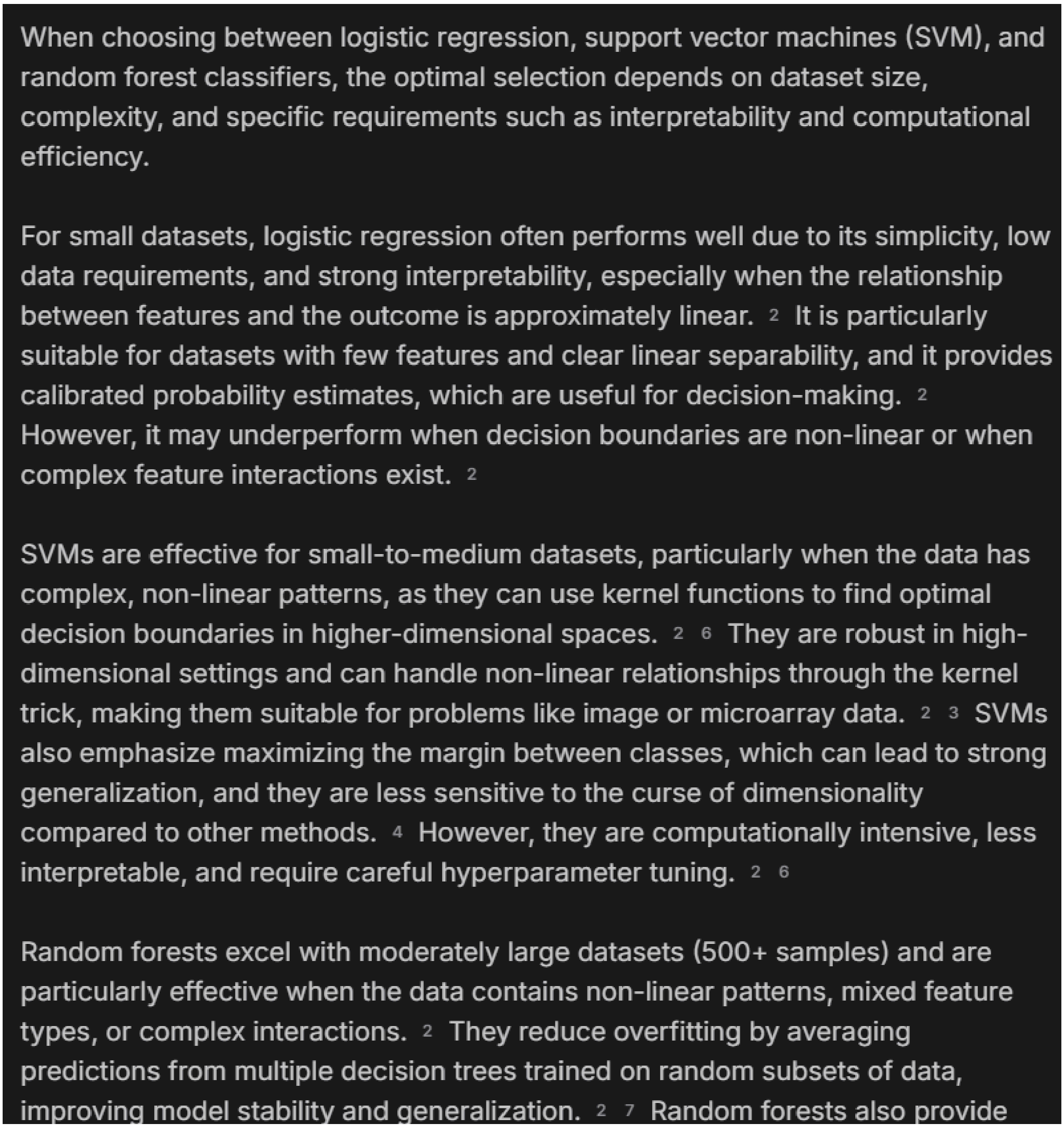


Random Forests are generally faster to train due to parallelization, more robust to noise and outliers, and offer straightforward feature importance measures based on impurity reduction, making them easier to interpret and less sensitive to hyperparameter settings. [2] [3] [4] [+2] They are well-suited for quick, reliable baseline models, large datasets, and scenarios where interpretability and stability are important. [3] [7] Gradient Boosting, while slower due to its sequential nature, often achieves superior predictive accuracy when properly tuned and is particularly effective for imbalanced datasets and complex relationships. [2] [3] [4] [+2] However, it demands more computational resources, is more sensitive to hyperparameter choices, and can be more difficult to interpret due to the cascading error correction process. [2] [4] [7]

The choice between the two depends on the specific problem context: **use Random Forests for fast, robust, and interpretable models with less tuning effort, especially on noisy or high-dimensional data** [3] [7] ; use Gradient Boosting when maximum accuracy is the priority, the data is clean, and sufficient time and resources are available for extensive hyperparameter optimization. [2] [3] [4] [+1]



classification logistic vs svm vs random forest classifier

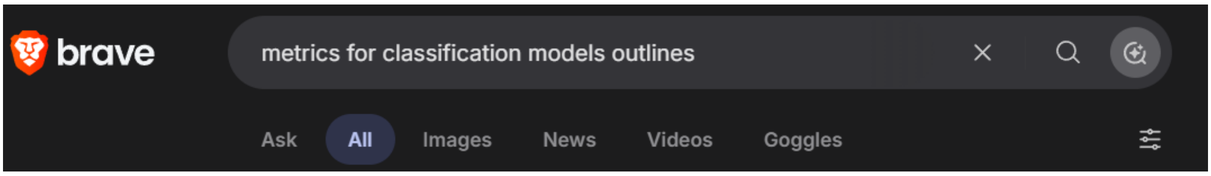


When choosing between logistic regression, support vector machines (SVM), and random forest classifiers, the optimal selection depends on dataset size, complexity, and specific requirements such as interpretability and computational efficiency.

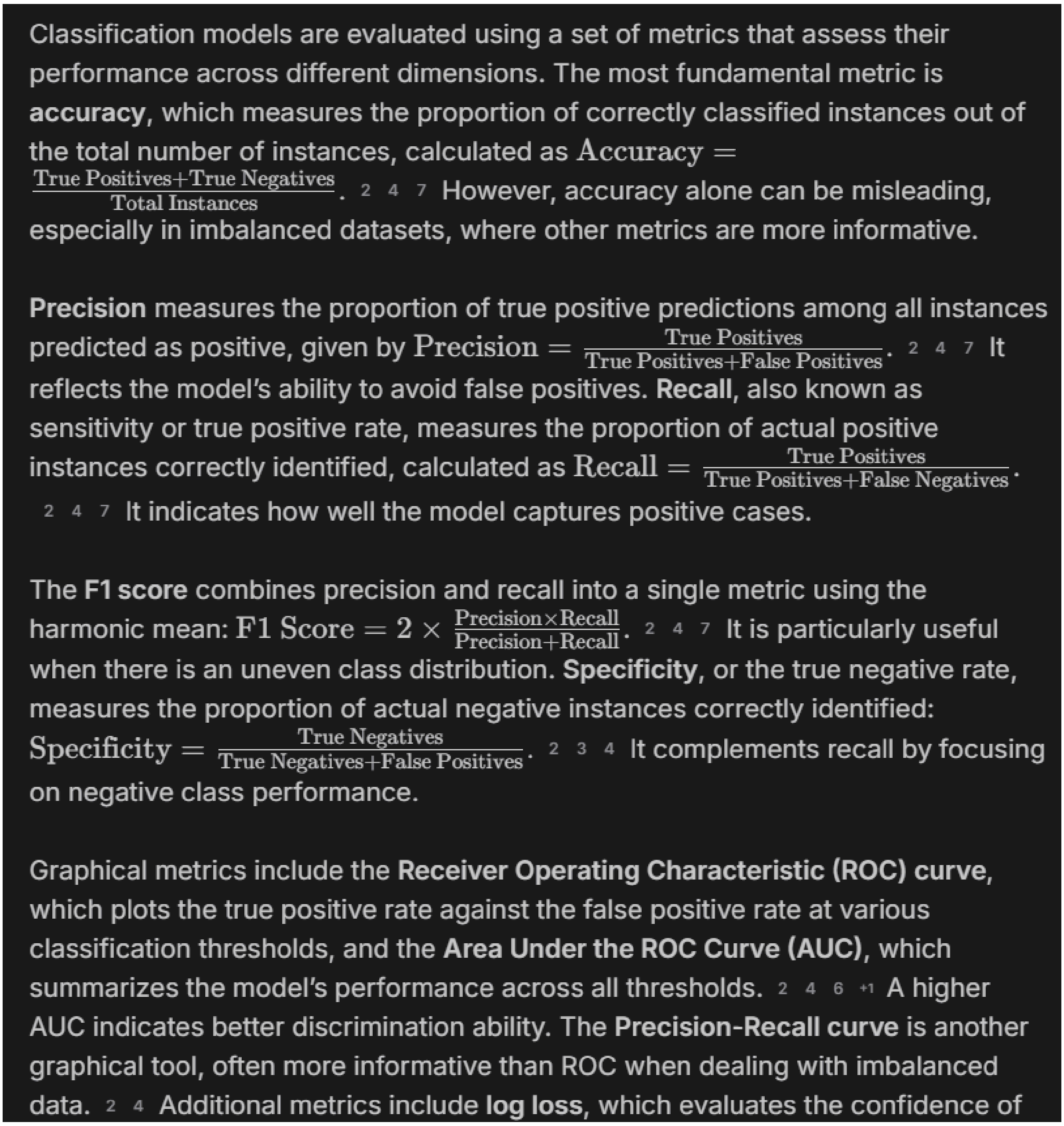
For small datasets, logistic regression often performs well due to its simplicity, low data requirements, and strong interpretability, especially when the relationship between features and the outcome is approximately linear. [2] It is particularly suitable for datasets with few features and clear linear separability, and it provides calibrated probability estimates, which are useful for decision-making. [2] However, it may underperform when decision boundaries are non-linear or when complex feature interactions exist. [2]

SVMs are effective for small-to-medium datasets, particularly when the data has complex, non-linear patterns, as they can use kernel functions to find optimal decision boundaries in higher-dimensional spaces. [2] [6] They are robust in high-dimensional settings and can handle non-linear relationships through the kernel trick, making them suitable for problems like image or microarray data. [2] [3] SVMs also emphasize maximizing the margin between classes, which can lead to strong generalization, and they are less sensitive to the curse of dimensionality compared to other methods. [4] However, they are computationally intensive, less interpretable, and require careful hyperparameter tuning. [2] [6]

Random forests excel with moderately large datasets (500+ samples) and are particularly effective when the data contains non-linear patterns, mixed feature types, or complex interactions. [2] They reduce overfitting by averaging predictions from multiple decision trees trained on random subsets of data, improving model stability and generalization. [2] [7] Random forests also provide



metrics for classification models outlines

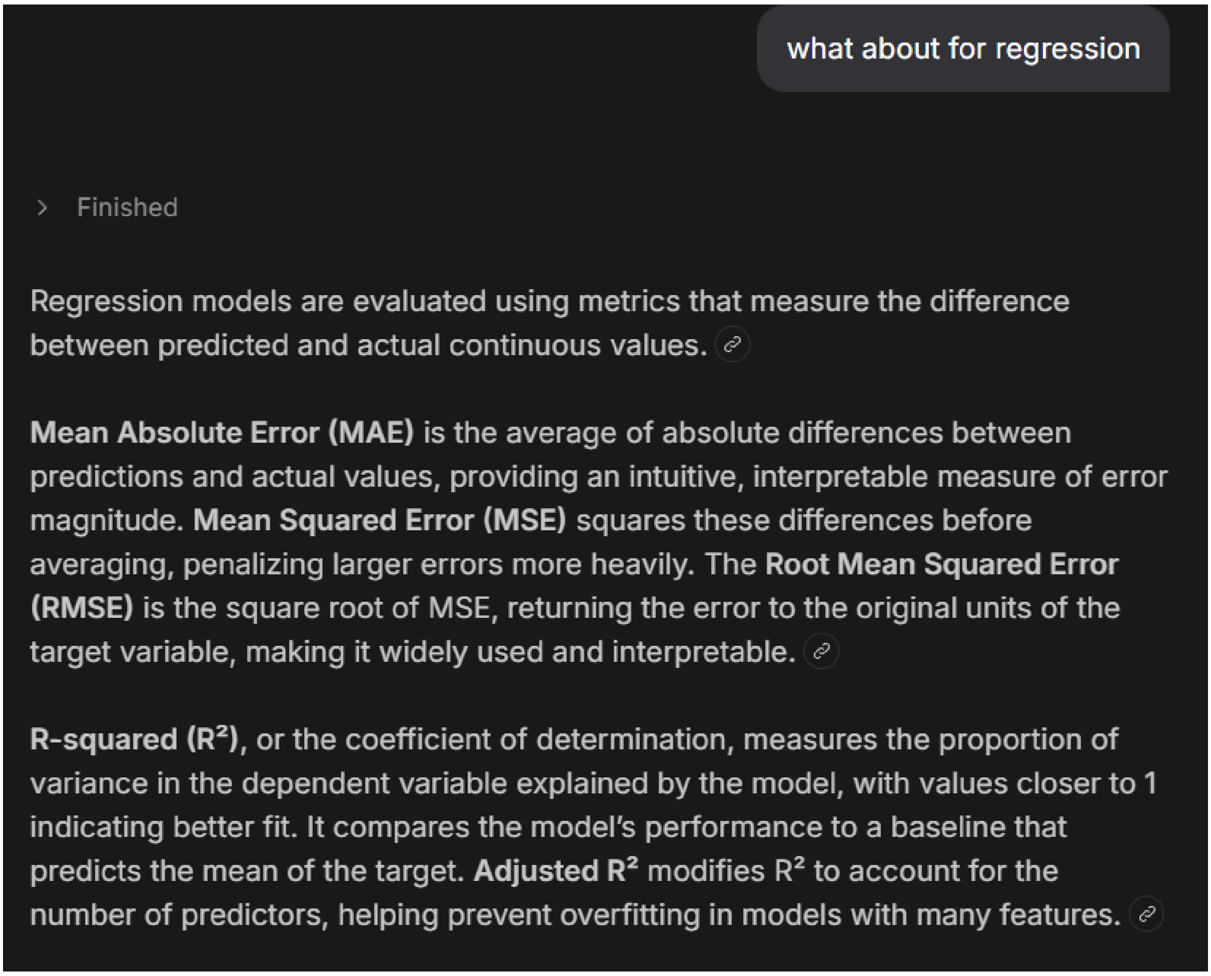


Classification models are evaluated using a set of metrics that assess their performance across different dimensions. The most fundamental metric is **accuracy**, which measures the proportion of correctly classified instances out of the total number of instances, calculated as $\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$. [2][4][7] However, accuracy alone can be misleading, especially in imbalanced datasets, where other metrics are more informative.

Precision measures the proportion of true positive predictions among all instances predicted as positive, given by $\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$. [2][4][7] It reflects the model's ability to avoid false positives. **Recall**, also known as sensitivity or true positive rate, measures the proportion of actual positive instances correctly identified, calculated as $\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$. [2][4][7] It indicates how well the model captures positive cases.

The **F1 score** combines precision and recall into a single metric using the harmonic mean: $\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$. [2][4][7] It is particularly useful when there is an uneven class distribution. **Specificity**, or the true negative rate, measures the proportion of actual negative instances correctly identified: $\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$. [2][3][4] It complements recall by focusing on negative class performance.

Graphical metrics include the **Receiver Operating Characteristic (ROC) curve**, which plots the true positive rate against the false positive rate at various classification thresholds, and the **Area Under the ROC Curve (AUC)**, which summarizes the model's performance across all thresholds. [2][4][6] +1 A higher AUC indicates better discrimination ability. The **Precision-Recall curve** is another graphical tool, often more informative than ROC when dealing with imbalanced data. [2][4] Additional metrics include **log loss**, which evaluates the confidence of

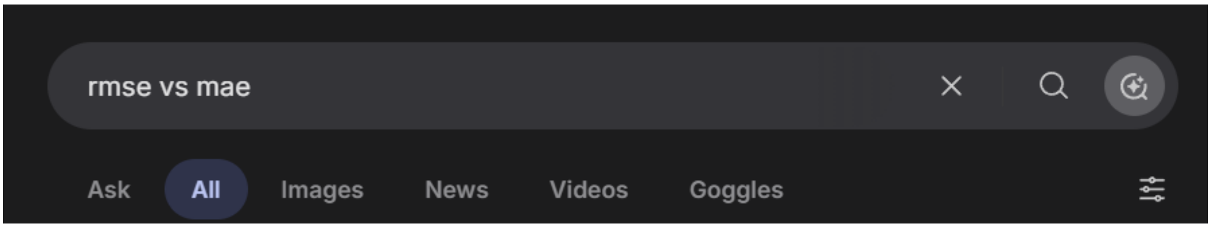


what about for regression

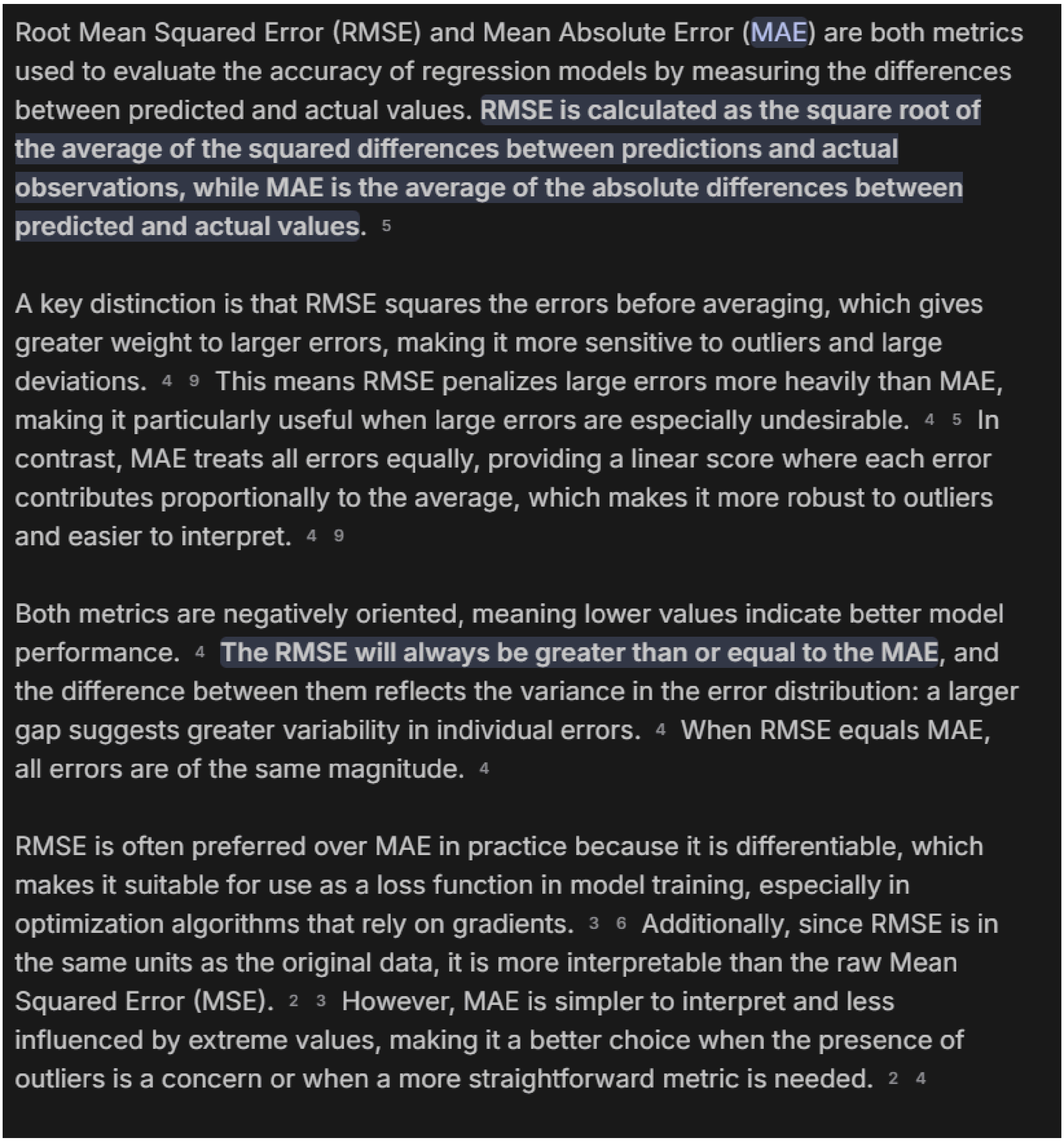
Regression models are evaluated using metrics that measure the difference between predicted and actual continuous values.

Mean Absolute Error (MAE) is the average of absolute differences between predictions and actual values, providing an intuitive, interpretable measure of error magnitude. **Mean Squared Error (MSE)** squares these differences before averaging, penalizing larger errors more heavily. The **Root Mean Squared Error (RMSE)** is the square root of MSE, returning the error to the original units of the target variable, making it widely used and interpretable.

R-squared (R^2), or the coefficient of determination, measures the proportion of variance in the dependent variable explained by the model, with values closer to 1 indicating better fit. It compares the model's performance to a baseline that predicts the mean of the target. **Adjusted R^2** modifies R^2 to account for the number of predictors, helping prevent overfitting in models with many features.



rmse vs mae

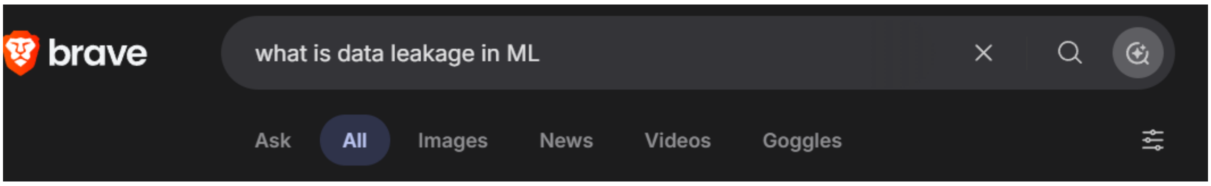


Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are both metrics used to evaluate the accuracy of regression models by measuring the differences between predicted and actual values. **RMSE is calculated as the square root of the average of the squared differences between predictions and actual observations, while MAE is the average of the absolute differences between predicted and actual values**. [5]

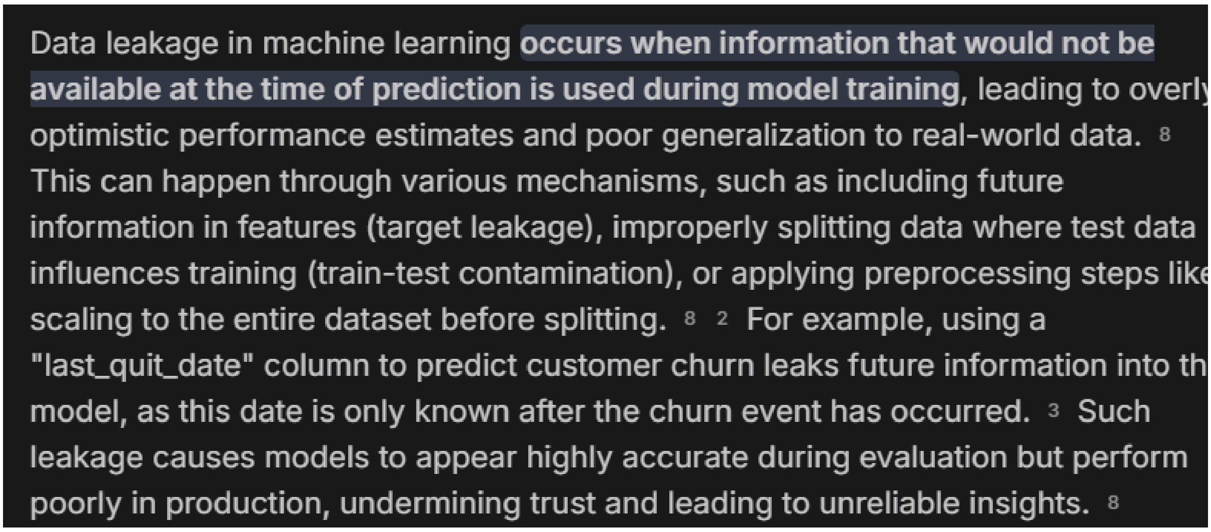
A key distinction is that RMSE squares the errors before averaging, which gives greater weight to larger errors, making it more sensitive to outliers and large deviations. [4] [9] This means RMSE penalizes large errors more heavily than MAE, making it particularly useful when large errors are especially undesirable. [4] [5] In contrast, MAE treats all errors equally, providing a linear score where each error contributes proportionally to the average, which makes it more robust to outliers and easier to interpret. [4] [9]

Both metrics are negatively oriented, meaning lower values indicate better model performance. [4] **The RMSE will always be greater than or equal to the MAE**, and the difference between them reflects the variance in the error distribution: a larger gap suggests greater variability in individual errors. [4] When RMSE equals MAE, all errors are of the same magnitude. [4]

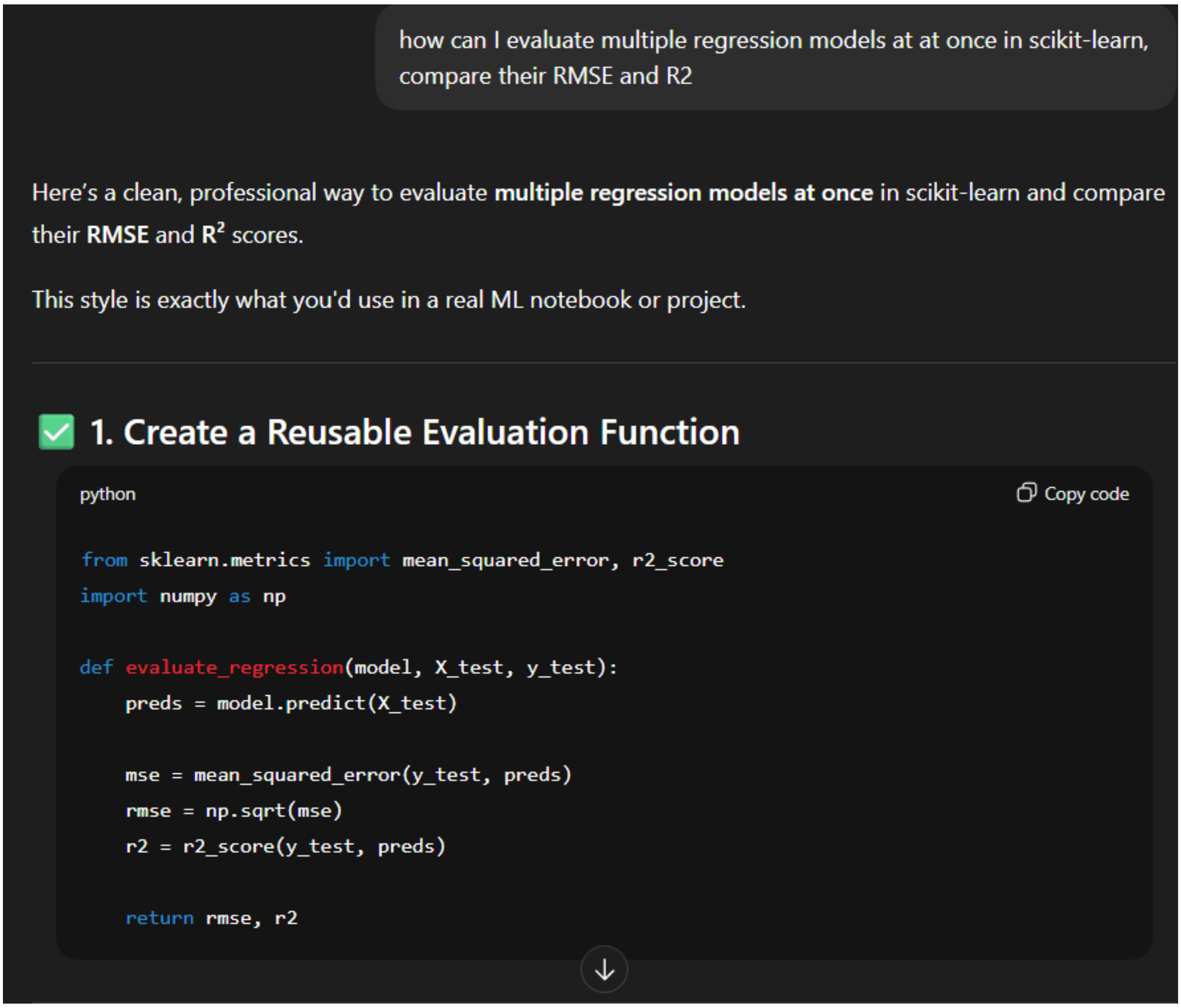
RMSE is often preferred over MAE in practice because it is differentiable, which makes it suitable for use as a loss function in model training, especially in optimization algorithms that rely on gradients. [3] [6] Additionally, since RMSE is in the same units as the original data, it is more interpretable than the raw Mean Squared Error (MSE). [2] [3] However, MAE is simpler to interpret and less influenced by extreme values, making it a better choice when the presence of outliers is a concern or when a more straightforward metric is needed. [2] [4]



what is data leakage in ML



Data leakage in machine learning **occurs when information that would not be available at the time of prediction is used during model training**, leading to overly optimistic performance estimates and poor generalization to real-world data. [8] This can happen through various mechanisms, such as including future information in features (target leakage), improperly splitting data where test data influences training (train-test contamination), or applying preprocessing steps lik scaling to the entire dataset before splitting. [8] [2] For example, using a "last_quit_date" column to predict customer churn leaks future information into th model, as this date is only known after the churn event has occurred. [3] Such leakage causes models to appear highly accurate during evaluation but perform poorly in production, undermining trust and leading to unreliable insights. [8]



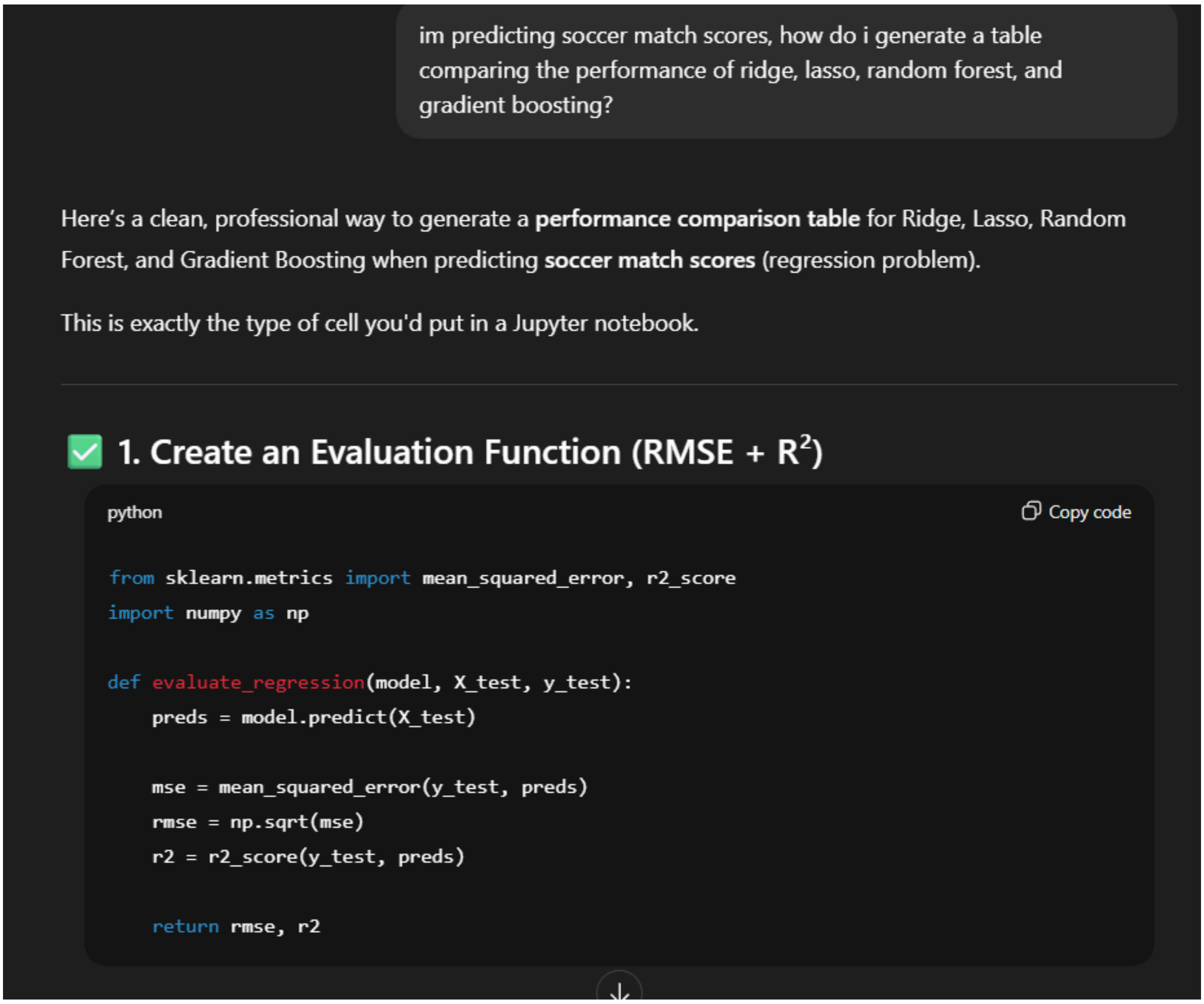
how can I evaluate multiple regression models at at once in scikit-learn, compare their RMSE and R2

Here's a clean, professional way to evaluate **multiple regression models at once** in scikit-learn and compare their **RMSE** and **R²** scores.

This style is exactly what you'd use in a real ML notebook or project.

✅ 1. Create a Reusable Evaluation Function

```python
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

def evaluate_regression(model, X_test, y_test):
    preds = model.predict(X_test)

    mse = mean_squared_error(y_test, preds)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, preds)

    return rmse, r2
```



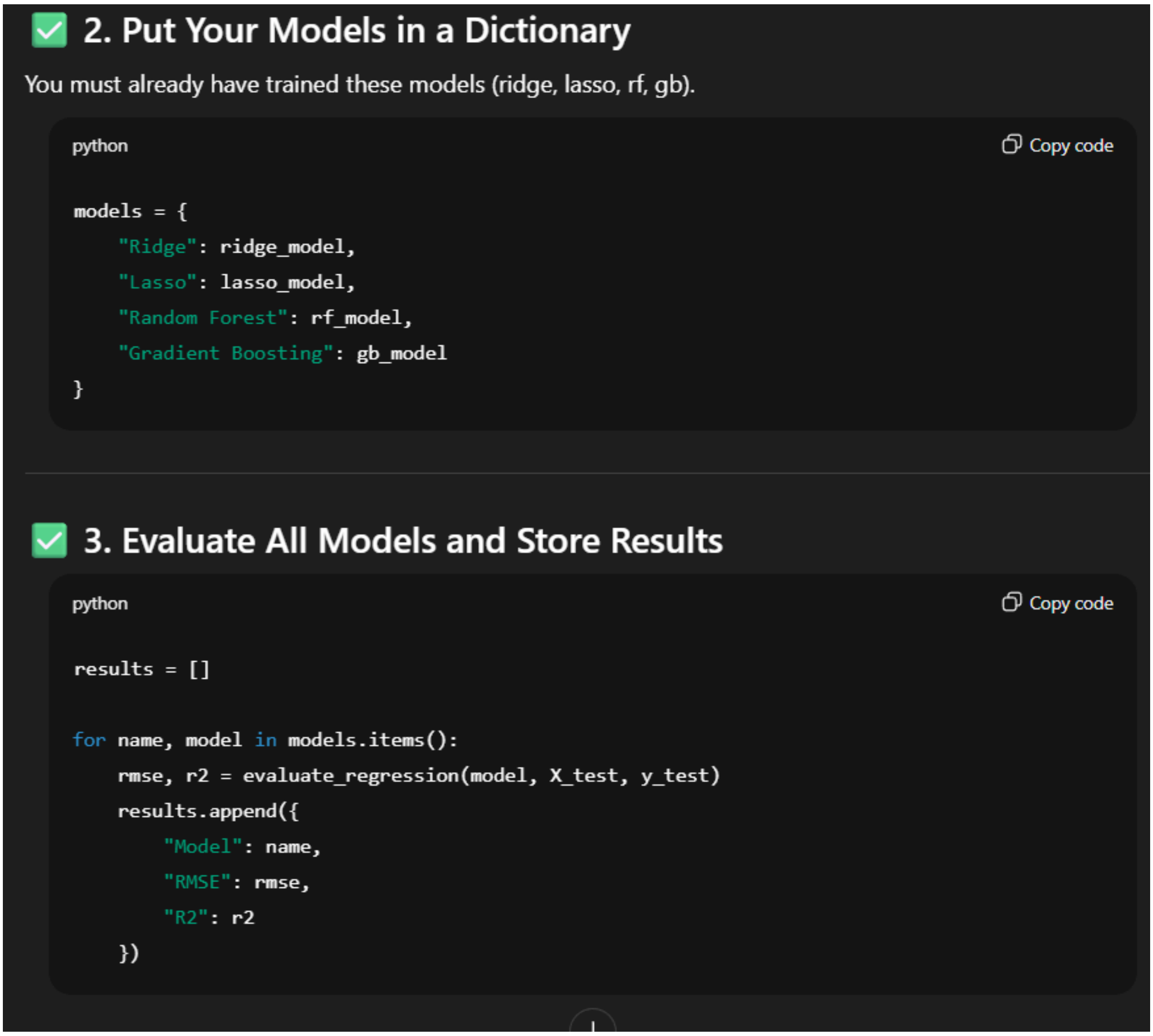
im predicting soccer match scores, how do i generate a table comparing the performance of ridge, lasso, random forest, and gradient boosting?

Here's a clean, professional way to generate a **performance comparison table** for Ridge, Lasso, Random Forest, and Gradient Boosting when predicting **soccer match scores** (regression problem).

This is exactly the type of cell you'd put in a Jupyter notebook.

✅ 1. Create an Evaluation Function (RMSE + R^2)

```python
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

def evaluate_regression(model, X_test, y_test):
    preds = model.predict(X_test)

    mse = mean_squared_error(y_test, preds)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, preds)

    return rmse, r2
```



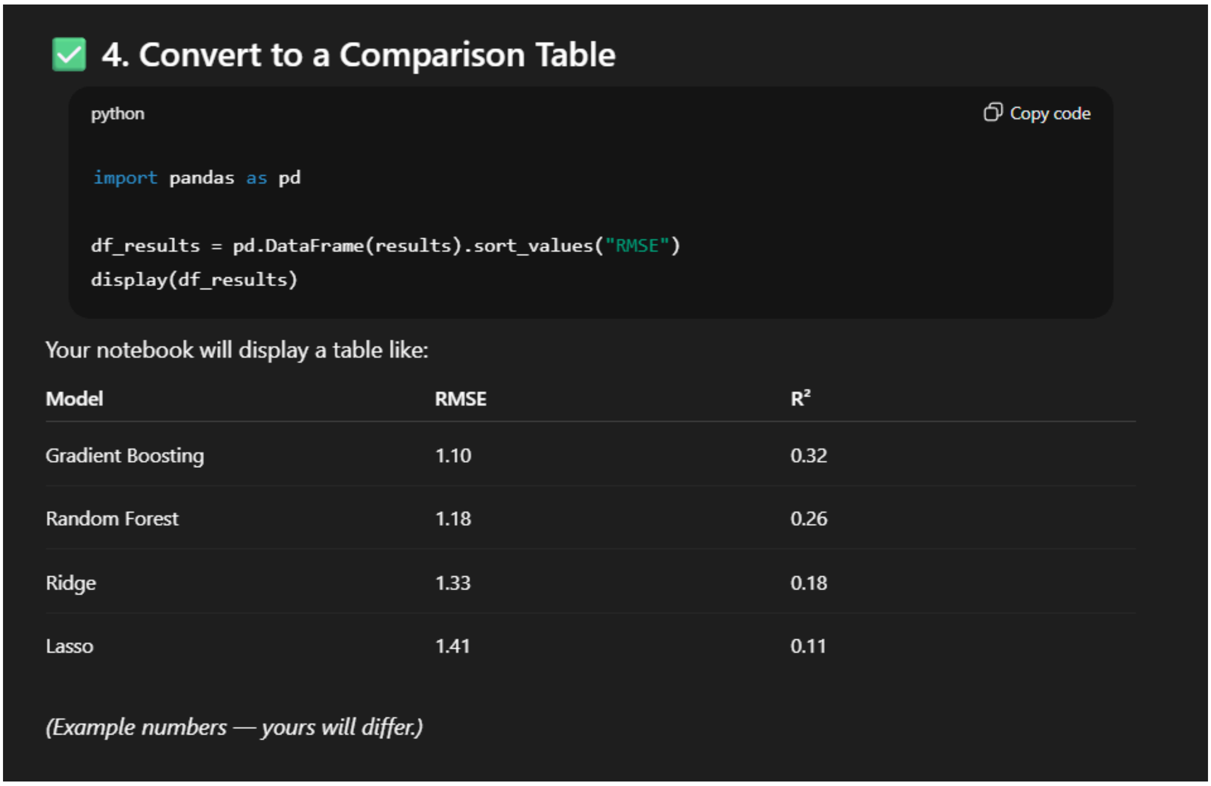
✅ 2. Put Your Models in a Dictionary

You must already have trained these models (ridge, lasso, rf, gb).

```python
models = {
    "Ridge": ridge_model,
    "Lasso": lasso_model,
    "Random Forest": rf_model,
    "Gradient Boosting": gb_model
}
```

✅ 3. Evaluate All Models and Store Results

```python
results = []

for name, model in models.items():
    rmse, r2 = evaluate_regression(model, X_test, y_test)
    results.append({
        "Model": name,
        "RMSE": rmse,
        "R2": r2
    })
```



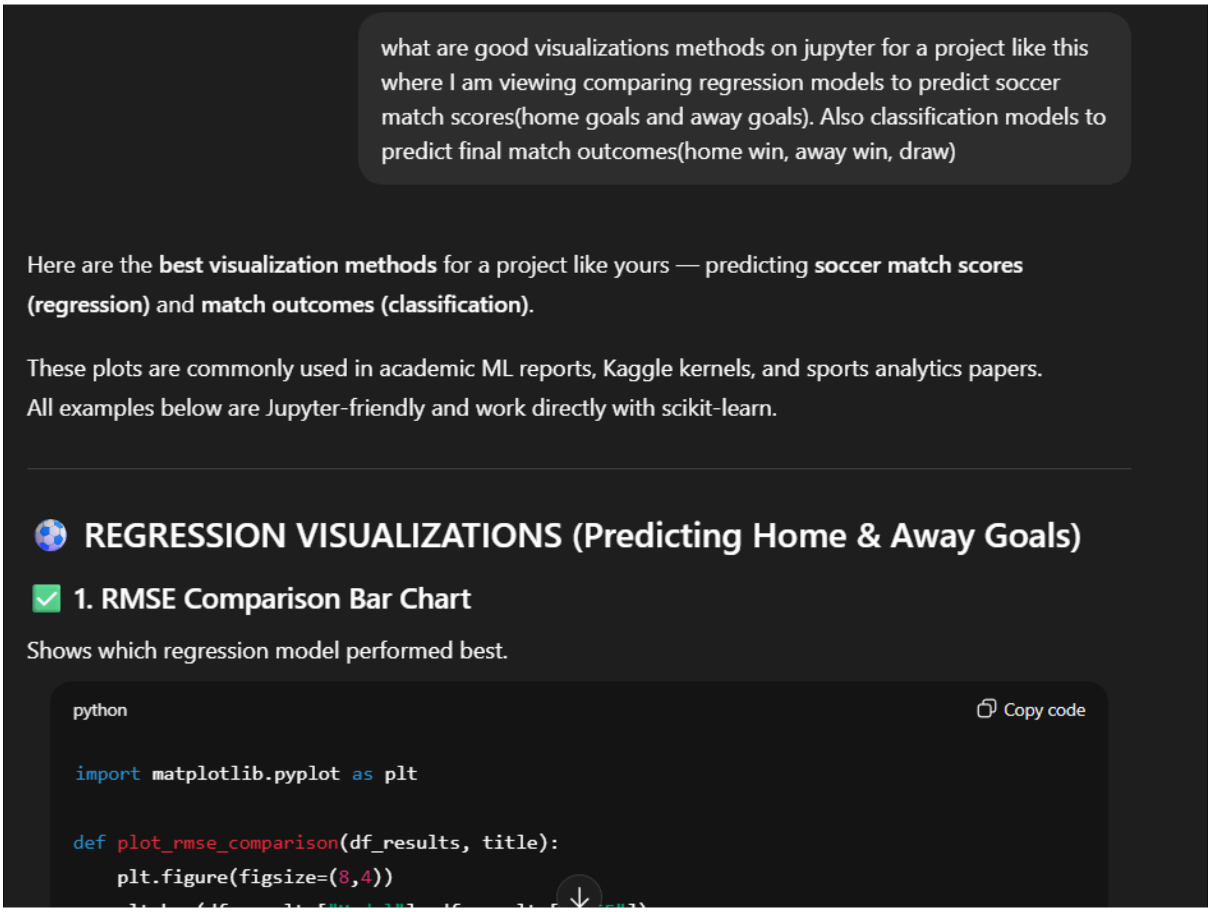
✅ 4. Convert to a Comparison Table

```python
import pandas as pd

df_results = pd.DataFrame(results).sort_values("RMSE")
display(df_results)
```

Your notebook will display a table like:

Model	RMSE	R²
Gradient Boosting	1.10	0.32
Random Forest	1.18	0.26
Ridge	1.33	0.18
Lasso	1.41	0.11

(Example numbers — yours will differ.)



what are good visualizations methods on jupyter for a project like this where I am viewing comparing regression models to predict soccer match scores(home goals and away goals). Also classification models to predict final match outcomes(home win, away win, draw)

Here are the **best visualization methods** for a project like yours — predicting **soccer match scores (regression)** and **match outcomes (classification)**.

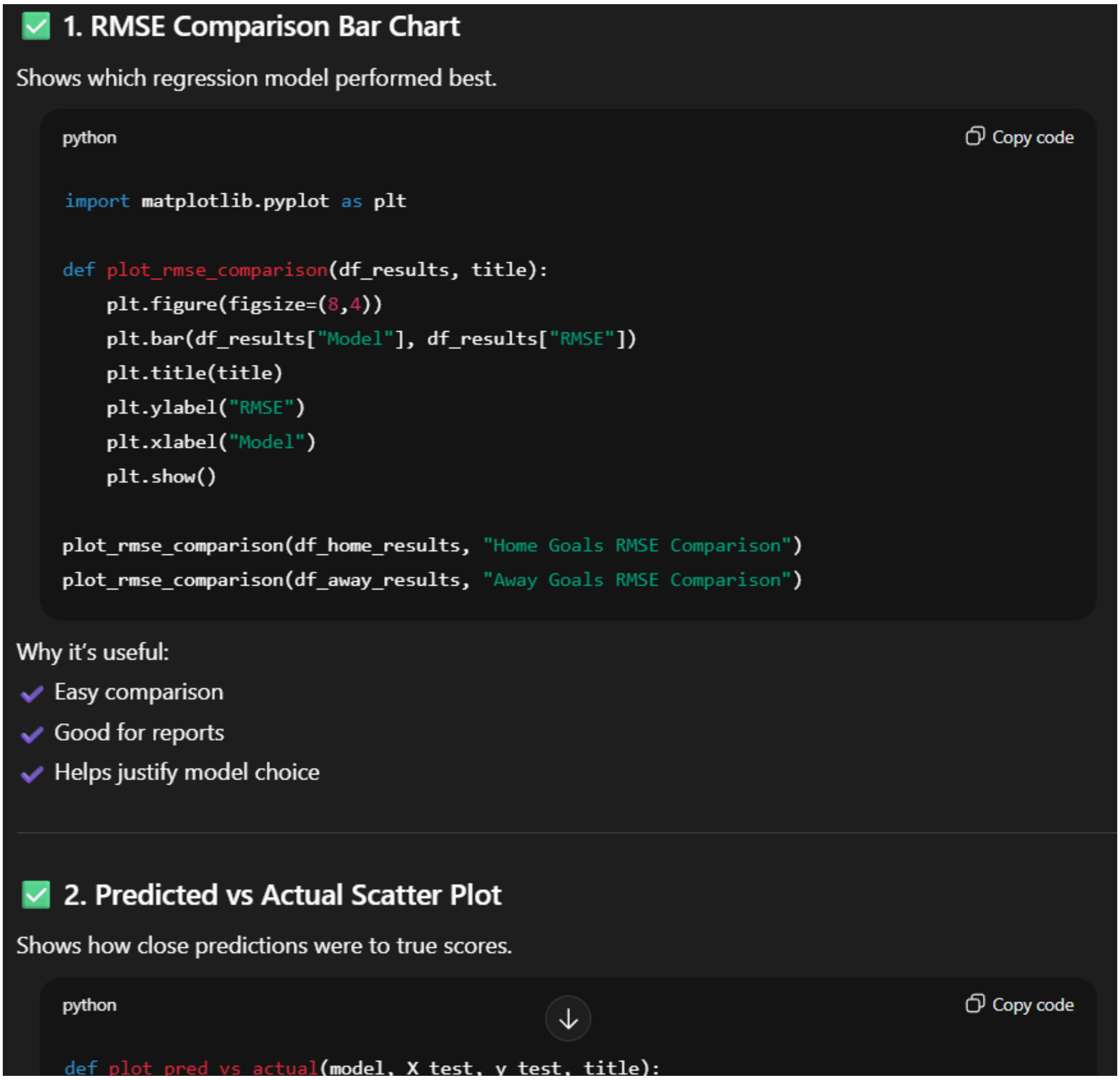
These plots are commonly used in academic ML reports, Kaggle kernels, and sports analytics papers.
All examples below are Jupyter-friendly and work directly with scikit-learn.

⚽ REGRESSION VISUALIZATIONS (Predicting Home & Away Goals)

✅ 1. RMSE Comparison Bar Chart

Shows which regression model performed best.

```python
import matplotlib.pyplot as plt

def plot_rmse_comparison(df_results, title):
    plt.figure(figsize=(8,4))
```



✅ 1. RMSE Comparison Bar Chart

Shows which regression model performed best.

```python
import matplotlib.pyplot as plt

def plot_rmse_comparison(df_results, title):
    plt.figure(figsize=(8,4))
    plt.bar(df_results["Model"], df_results["RMSE"])
    plt.title(title)
    plt.ylabel("RMSE")
    plt.xlabel("Model")
    plt.show()

plot_rmse_comparison(df_home_results, "Home Goals RMSE Comparison")
plot_rmse_comparison(df_away_results, "Away Goals RMSE Comparison")
```

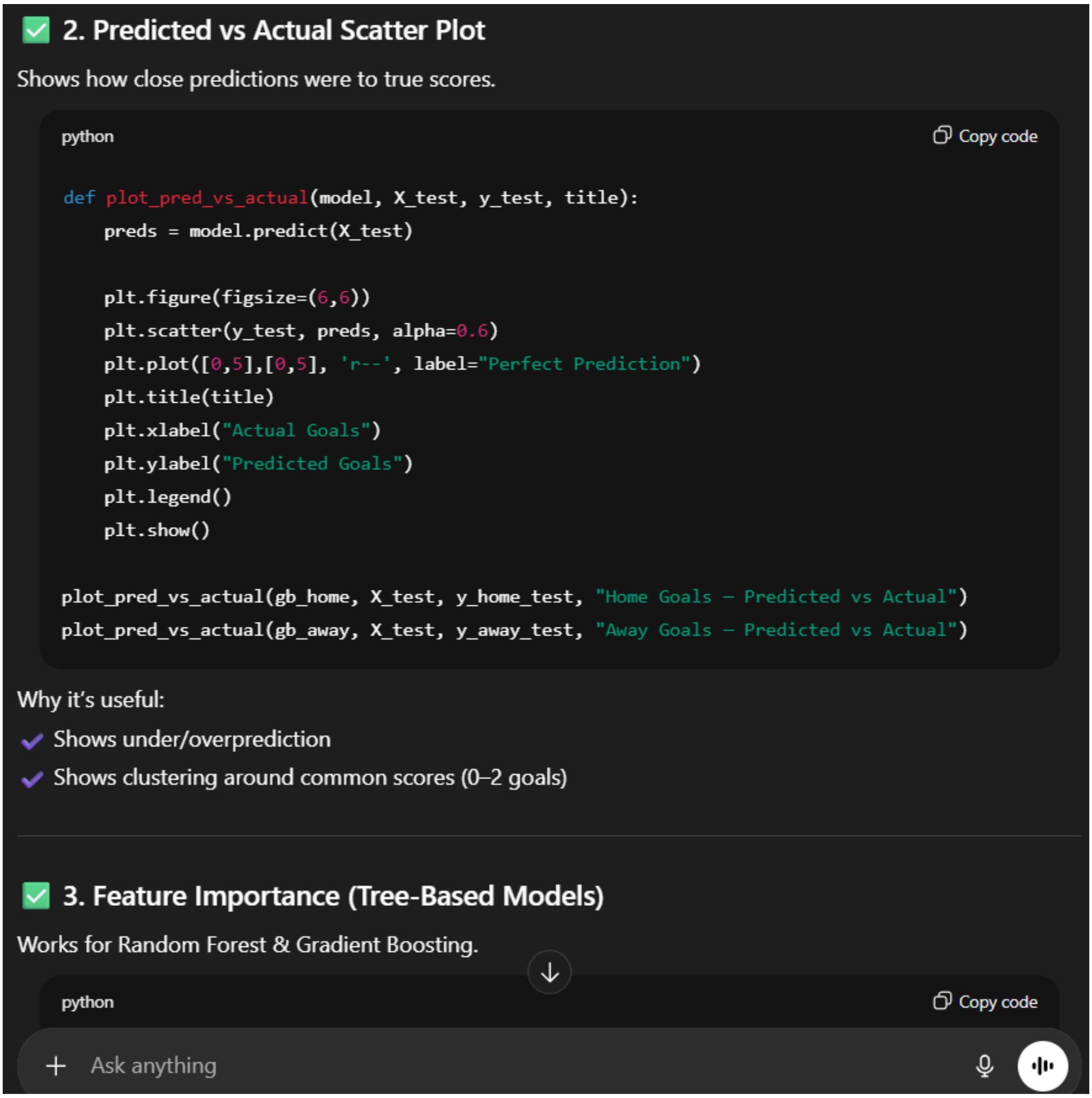
Why it's useful:

- ✔ Easy comparison
- ✔ Good for reports
- ✔ Helps justify model choice

✅ 2. Predicted vs Actual Scatter Plot

Shows how close predictions were to true scores.

```python
def plot_pred_vs_actual(model, X_test, y_test, title):
```



✅ 2. Predicted vs Actual Scatter Plot

Shows how close predictions were to true scores.

```python
def plot_pred_vs_actual(model, X_test, y_test, title):
    preds = model.predict(X_test)

    plt.figure(figsize=(6,6))
    plt.scatter(y_test, preds, alpha=0.6)
    plt.plot([0,5],[0,5], 'r--', label="Perfect Prediction")
    plt.title(title)
    plt.xlabel("Actual Goals")
    plt.ylabel("Predicted Goals")
    plt.legend()
    plt.show()

plot_pred_vs_actual(gb_home, X_test, y_home_test, "Home Goals – Predicted vs Actual")
plot_pred_vs_actual(gb_away, X_test, y_away_test, "Away Goals – Predicted vs Actual")
```

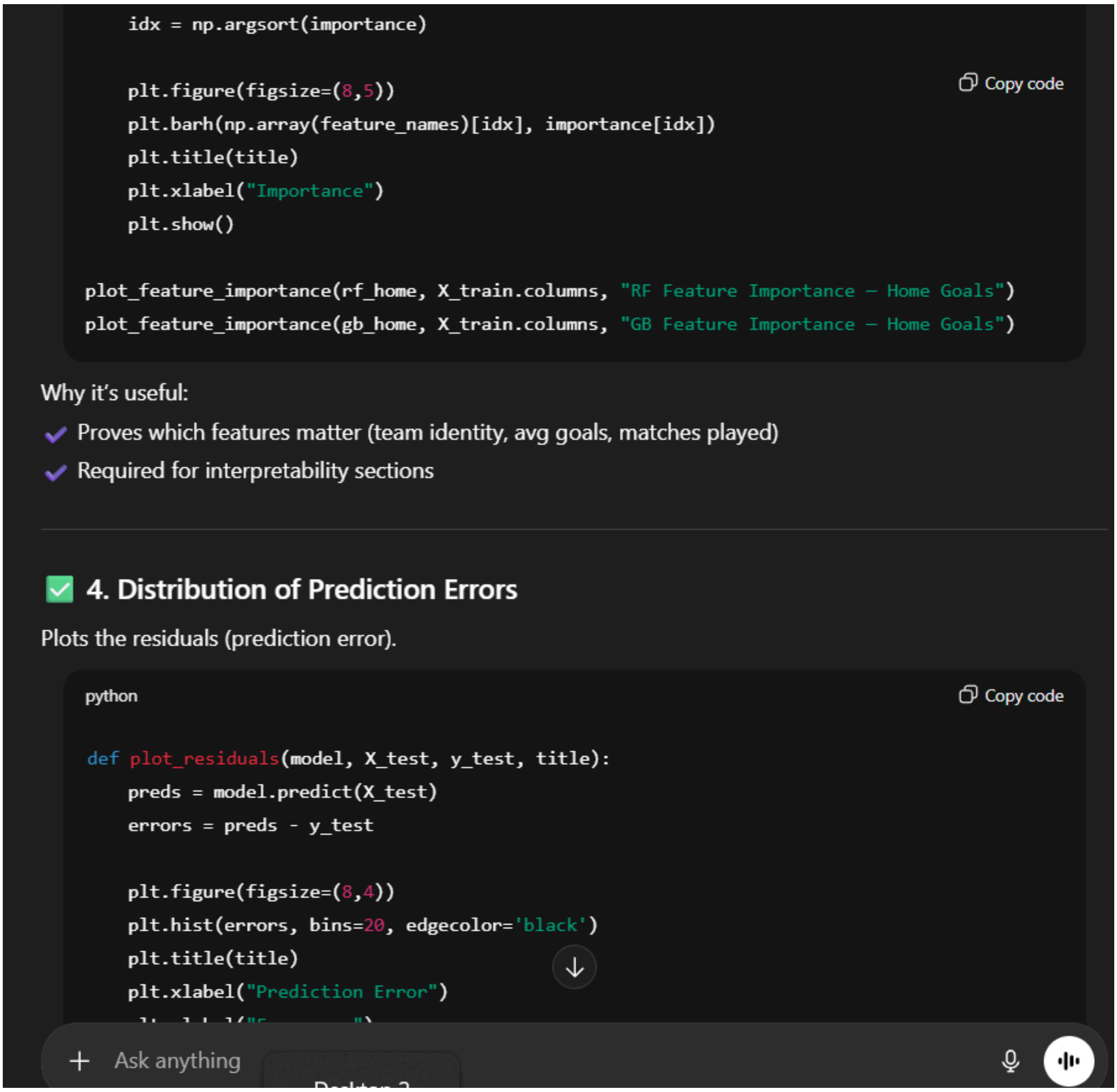
Why it's useful:

✓ Shows under/overprediction

✓ Shows clustering around common scores (0–2 goals)

✅ 3. Feature Importance (Tree-Based Models)

Works for Random Forest & Gradient Boosting.



```python
    idx = np.argsort(importance)

    plt.figure(figsize=(8,5))
    plt.barh(np.array(feature_names)[idx], importance[idx])
    plt.title(title)
    plt.xlabel("Importance")
    plt.show()

plot_feature_importance(rf_home, X_train.columns, "RF Feature Importance – Home Goals")
plot_feature_importance(gb_home, X_train.columns, "GB Feature Importance – Home Goals")
```

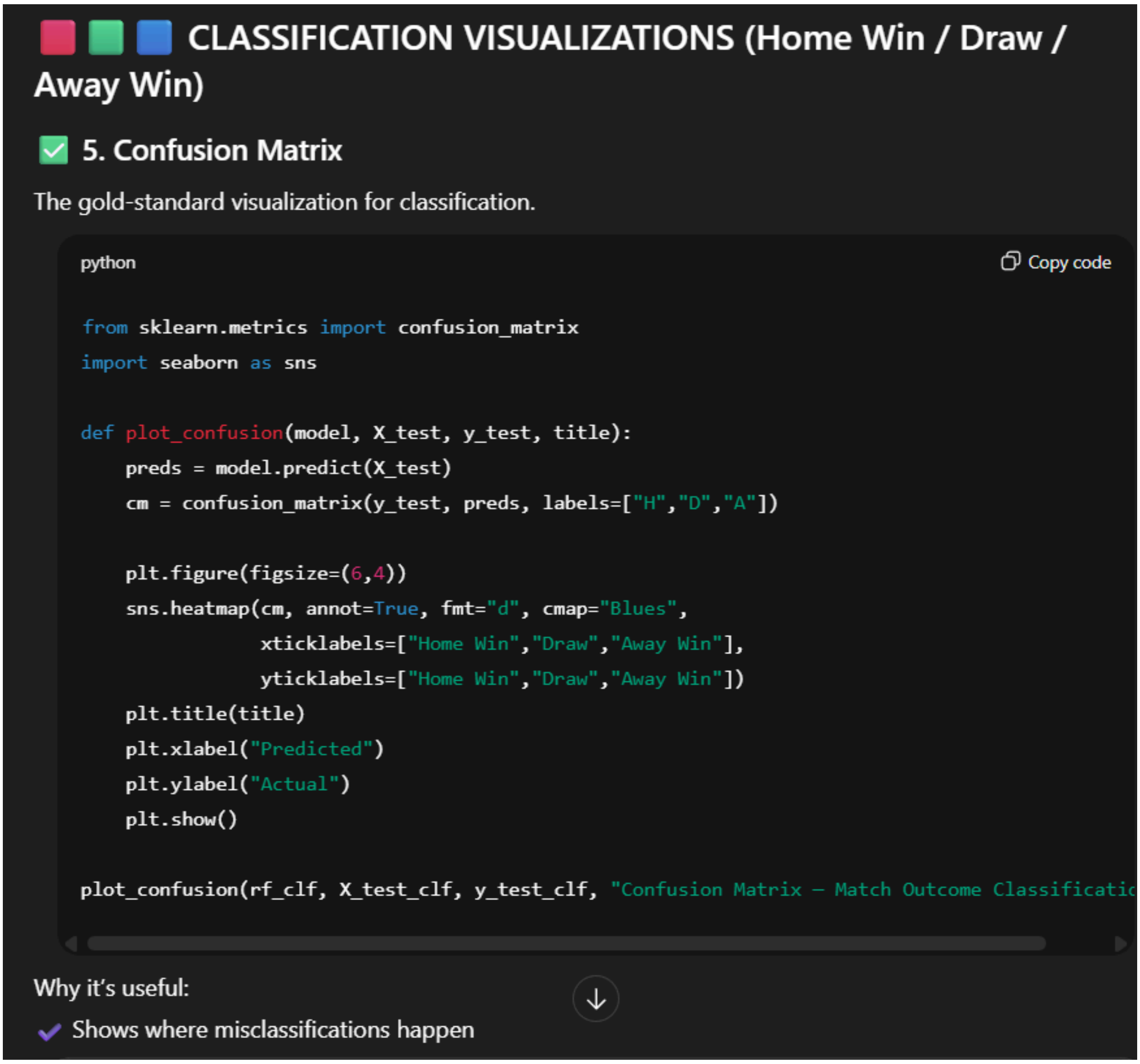
Why it's useful:

- ✓ Proves which features matter (team identity, avg goals, matches played)
- ✓ Required for interpretability sections

✅ 4. Distribution of Prediction Errors

Plots the residuals (prediction error).

```python
def plot_residuals(model, X_test, y_test, title):
    preds = model.predict(X_test)
    errors = preds - y_test

    plt.figure(figsize=(8,4))
    plt.hist(errors, bins=20, edgecolor='black')
    plt.title(title)
    plt.xlabel("Prediction Error")
```



🟥 🟩 🟦 CLASSIFICATION VISUALIZATIONS (Home Win / Draw / Away Win)

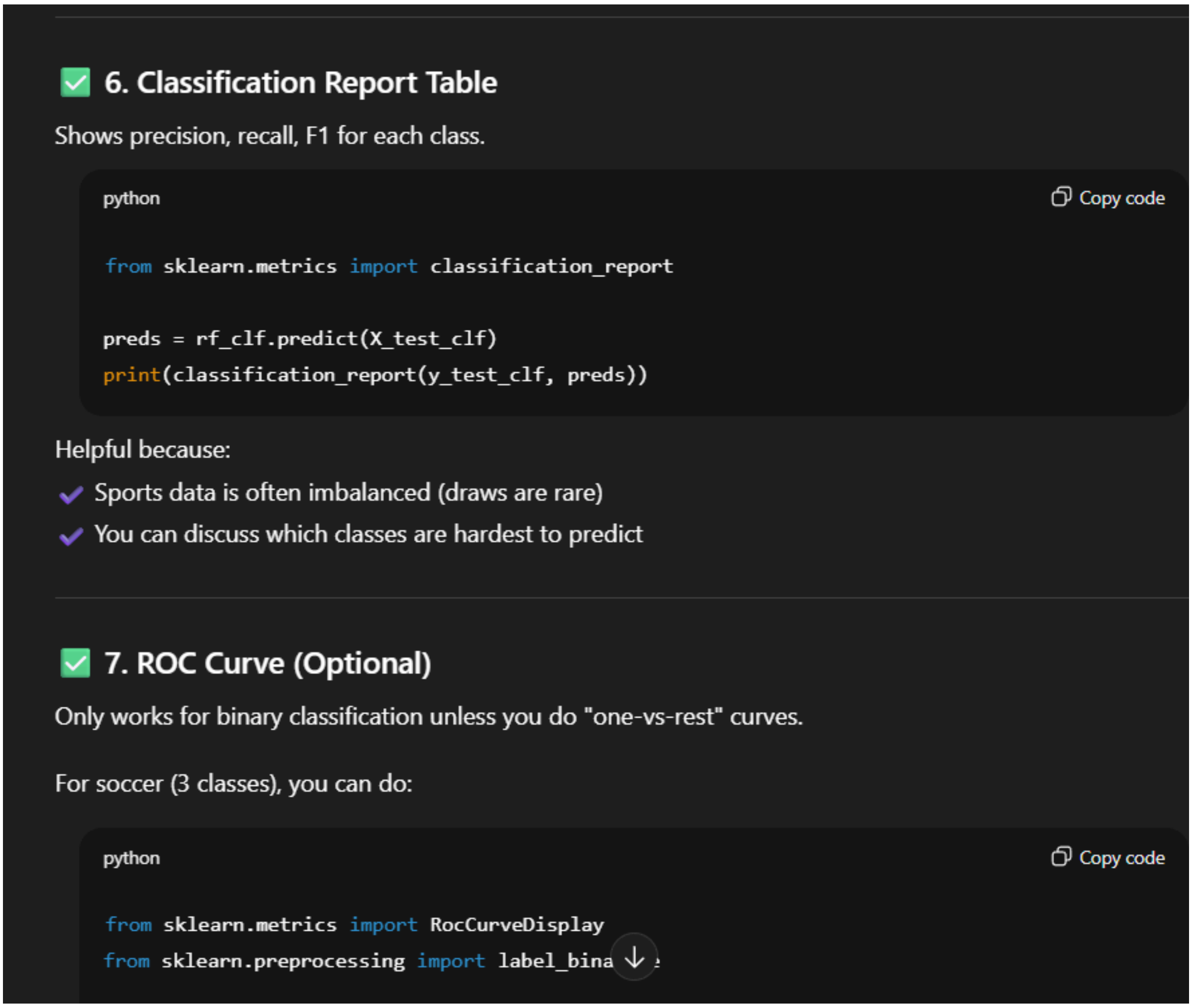
✅ 5. Confusion Matrix

The gold-standard visualization for classification.

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

def plot_confusion(model, X_test, y_test, title):
    preds = model.predict(X_test)
    cm = confusion_matrix(y_test, preds, labels=["H","D","A"])

    plt.figure(figsize=(6,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Home Win","Draw","Away Win"],
                yticklabels=["Home Win","Draw","Away Win"])
    plt.title(title)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

plot_confusion(rf_clf, X_test_clf, y_test_clf, "Confusion Matrix — Match Outcome Classificati
```

Why it's useful:

- ✔ Shows where misclassifications happen



✅ 6. Classification Report Table

Shows precision, recall, F1 for each class.

```python
from sklearn.metrics import classification_report

preds = rf_clf.predict(X_test_clf)
print(classification_report(y_test_clf, preds))
```

Helpful because:

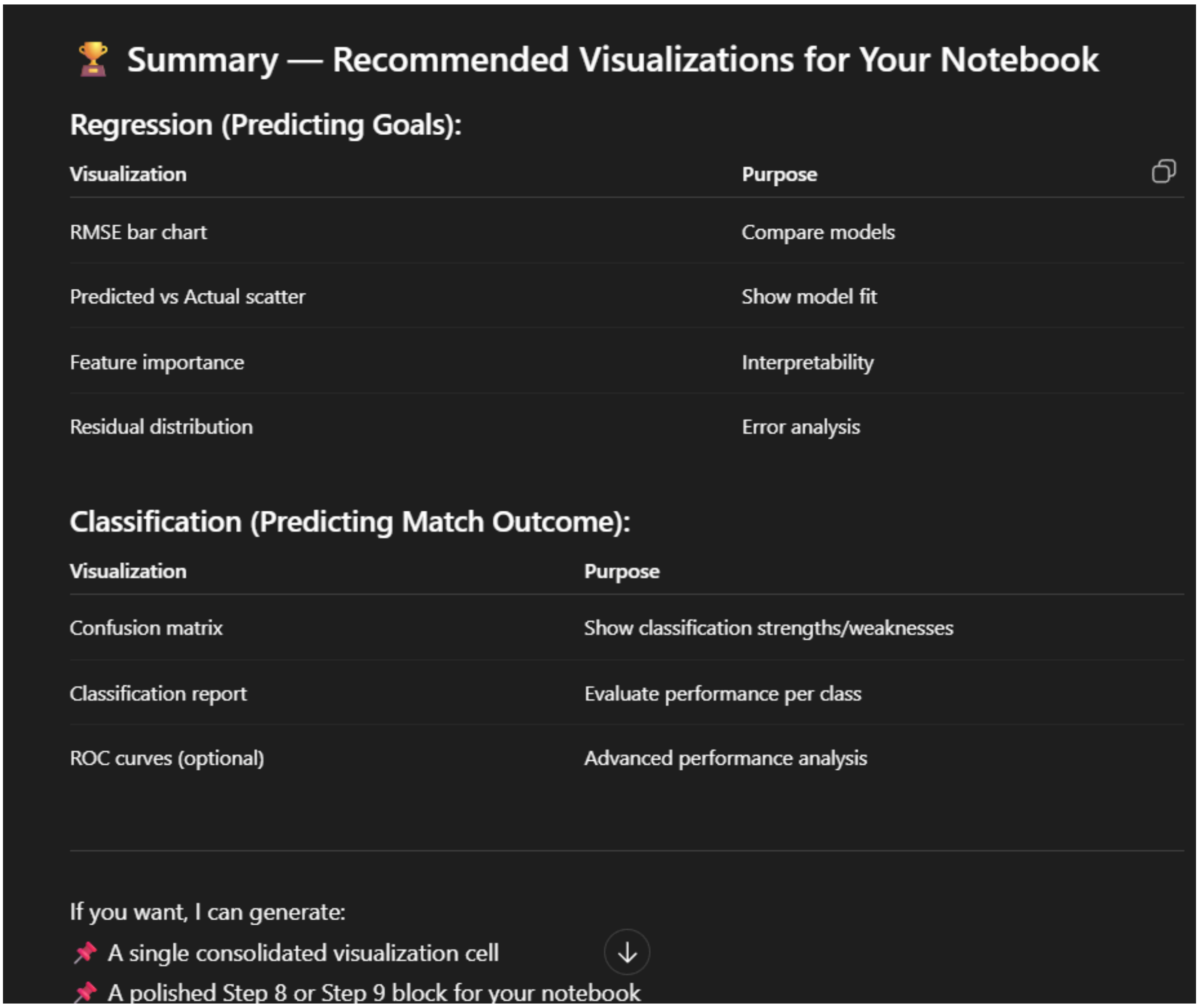
✔ Sports data is often imbalanced (draws are rare)
✔ You can discuss which classes are hardest to predict

✅ 7. ROC Curve (Optional)

Only works for binary classification unless you do "one-vs-rest" curves.

For soccer (3 classes), you can do:

```python
from sklearn.metrics import RocCurveDisplay
from sklearn.preprocessing import label_bina
```



🏆 Summary — Recommended Visualizations for Your Notebook

Regression (Predicting Goals):

Visualization	Purpose
RMSE bar chart	Compare models
Predicted vs Actual scatter	Show model fit
Feature importance	Interpretability
Residual distribution	Error analysis

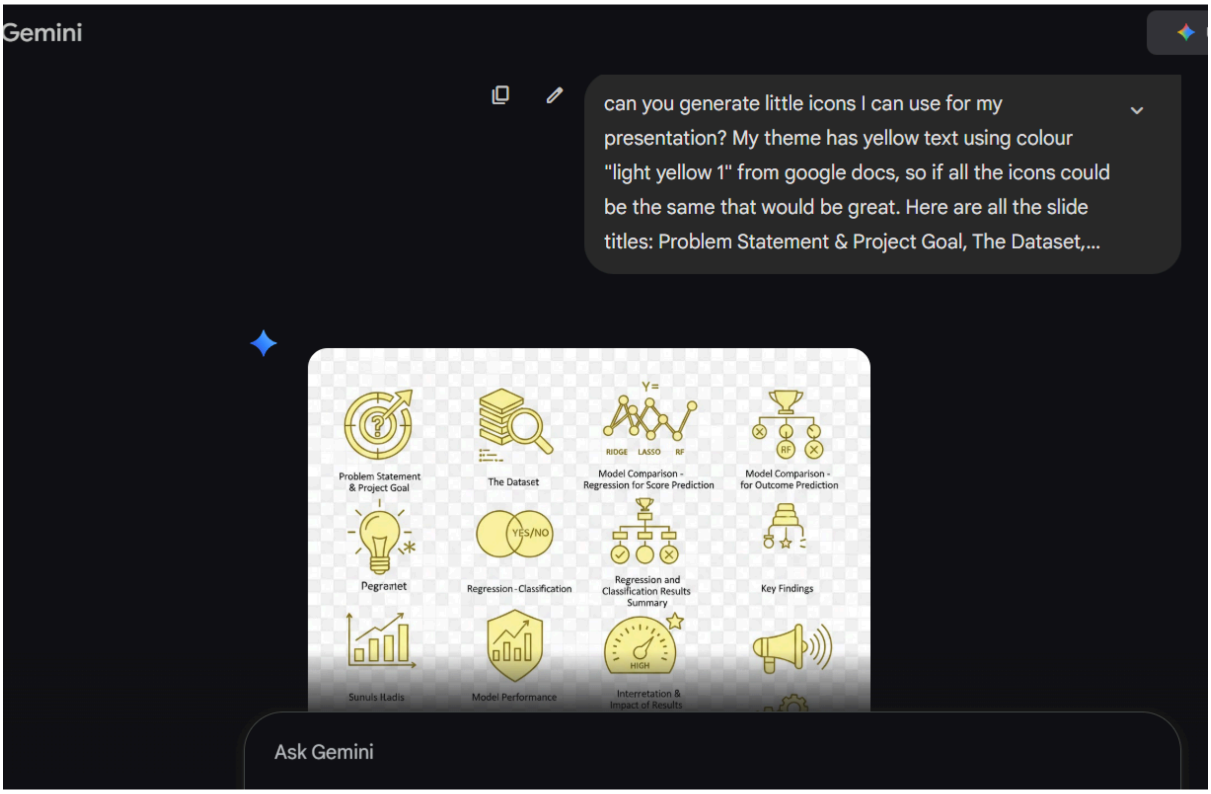
Classification (Predicting Match Outcome):

Visualization	Purpose
Confusion matrix	Show classification strengths/weaknesses
Classification report	Evaluate performance per class
ROC curves (optional)	Advanced performance analysis

If you want, I can generate:

📌 A single consolidated visualization cell

📌 A polished Step 8 or Step 9 block for your notebook



can you generate little icons I can use for my presentation? My theme has yellow text using colour "light yellow 1" from google docs, so if all the icons could be the same that would be great. Here are all the slide titles: Problem Statement & Project Goal, The Dataset,...

Y=
RIDGE
LASSO
RF
RF
YES/NO
HIGH
Problem Statement & Project Goal
The Dataset
Model Comparison - Regression for Score Prediction
Model Comparison - for Outcome Prediction
Pegramet
Regression - Classification
Regression and Classification Results Summary
Key Findings
Sunuls Iladis
Model Performance
Interretation & Impact of Results
Ask Gemini