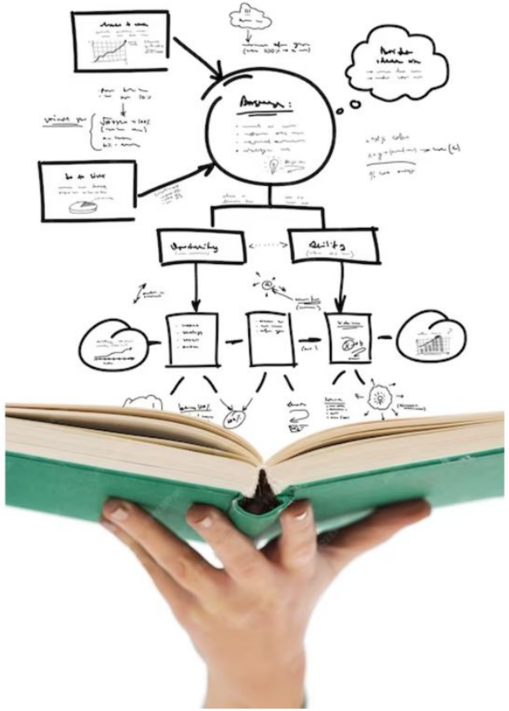
SESSION - SUPERVISED LEARNING
Recap and Code

Presented by: Khadijah S. Mohammed

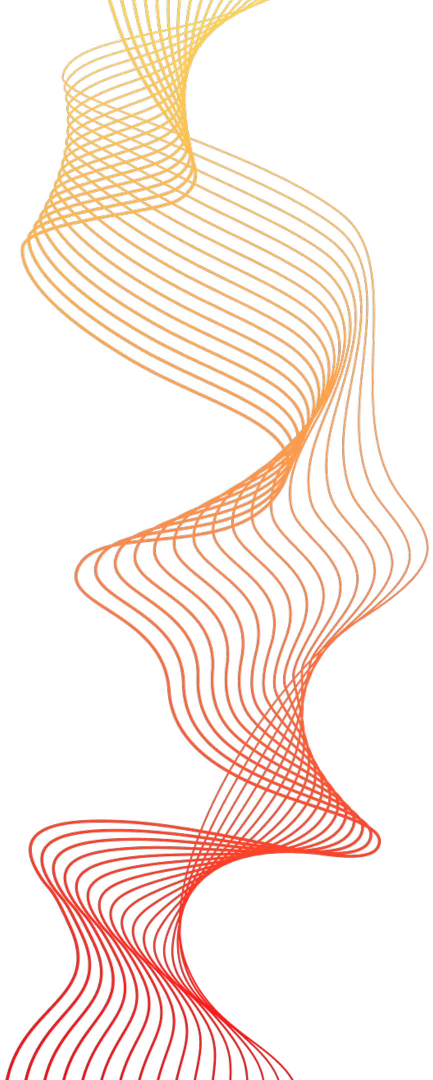


Enhancing Predictive Accuracy: Exploring the Power of Supervised Learning in Machine Learning

Understanding Regression and Classification

Regression - Predicting Continuous Values

Regression is a type of supervised learning where the goal is to predict continuous values. Imagine predicting the price of a house based on its features. It's like fitting a line to the data to make future predictions.

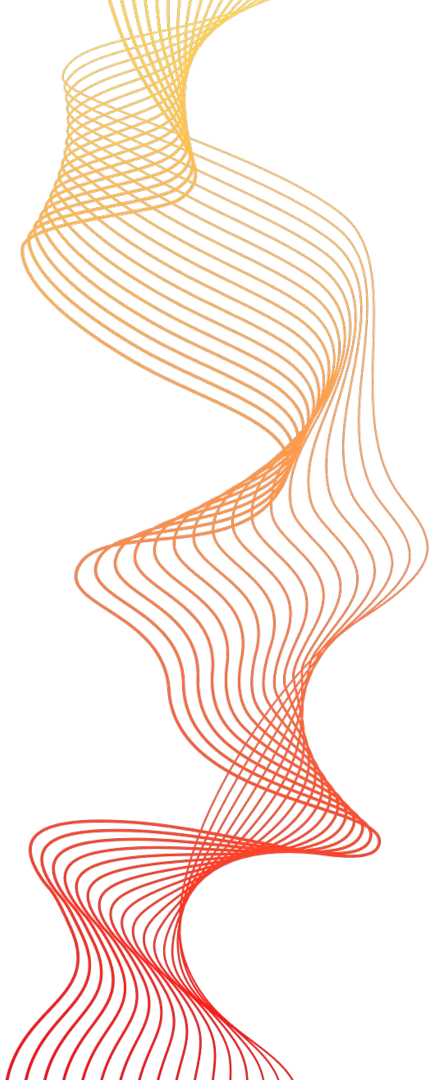


Types of Regression Algorithms

Tools for Continuous Predictions

Overview of common regression algorithms:

- Linear Regression
- Polynomial Regression
- Ridge Regression
- Lasso Regression



Types of Classification Algorithms

Choosing the Right Category

Overview of common classification algorithms:

- Logistic Regression
- Decision Trees
- Support Vector Machines
- K-Nearest Neighbors

Linear Regression

:

Definition: Linear Regression is a supervised machine learning algorithm used for predicting a continuous outcome variable (also called the dependent variable) based on one or more predictor variables (independent variables). It assumes a linear relationship between the predictors and the target variable.

Use Case: Predicting house prices based on features like square footage, number of bedrooms, etc.

Logistic Regression

Definition: Logistic Regression is a supervised machine learning algorithm used for binary classification tasks, where the outcome variable is categorical and has two classes. Despite its name, it's used for classification, not regression. It models the probability that an instance belongs to a particular class using a logistic function.

Use Case: Predicting whether an email is spam or not spam, or predicting whether a patient has a particular medical condition.

Decision Trees

:

Definition: Decision Trees are a versatile supervised machine learning algorithm used for both classification and regression tasks. They make decisions based on a tree-like graph structure, where each internal node represents a decision based on a feature, each branch represents an outcome of that decision, and each leaf node represents the final prediction or outcome.

Use Case: In a classification task, a Decision Tree might be used to determine the species of a flower based on features like petal length and width.

Evaluating Models

Measuring Success

Discussion on metrics:

- Accuracy
- Precision
- Recall
- F1 Score

LET’S CODE

Setting the Stage

Alright, let's dive into the code together!

Quick heads up about what we're getting into.

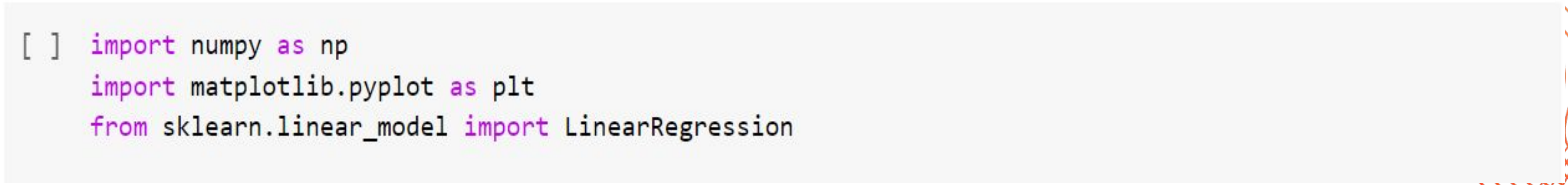
Introduction to the dataset: Hours Studied vs. Exam Scores.

We're going to work on predicting exam scores based on hours studied. Think of it as unraveling the magic behind predicting outcomes using machine learning.

Importing Libraries

Let's bring in the cool coding tools!

Simple explanation and code snippet for importing libraries:



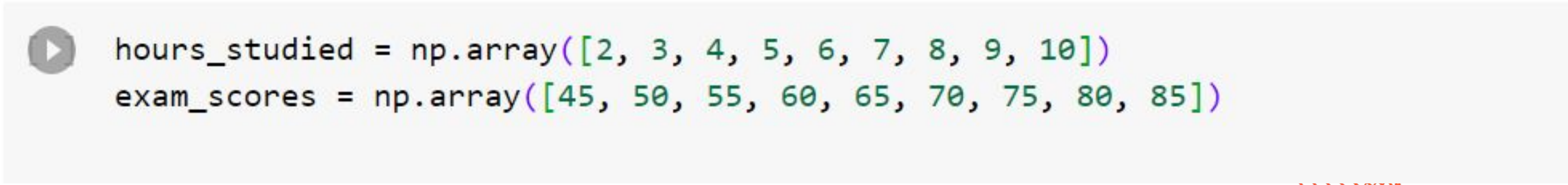
```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.linear_model import LinearRegression
```

Explanation: We're grabbing some tools to make our coding journey smooth. numpy for numerical operations, matplotlib for plotting, and LinearRegression from scikit-learn for our star prediction model.

Preparing the Data

Time to set the stage for our data drama!

Code snippet for defining hours_studied and exam_scores arrays:



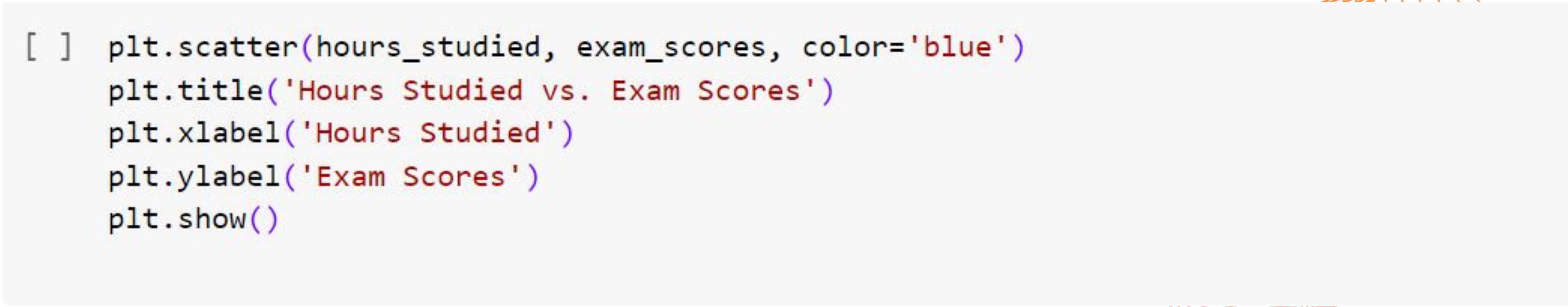
```python
hours_studied = np.array([2, 3, 4, 5, 6, 7, 8, 9, 10])
exam_scores = np.array([45, 50, 55, 60, 65, 70, 75, 80, 85])
```

Explanation: This is where we introduce our actors - the hours students studied and their corresponding exam scores. This data will be the heart of our prediction model.

Visualizing the Data

Let’s see what our data looks like - graph time!

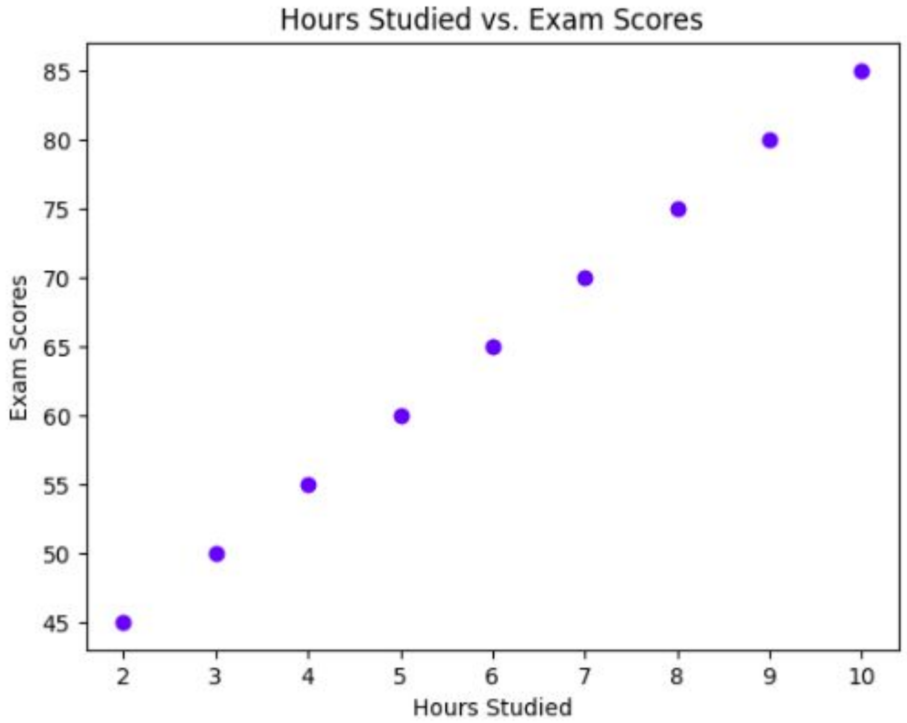
Code snippet for creating a scatter plot:



```
[ ]  plt.scatter(hours_studied, exam_scores, color='blue')
     plt.title('Hours Studied vs. Exam Scores')
     plt.xlabel('Hours Studied')
     plt.ylabel('Exam Scores')
     plt.show()
```

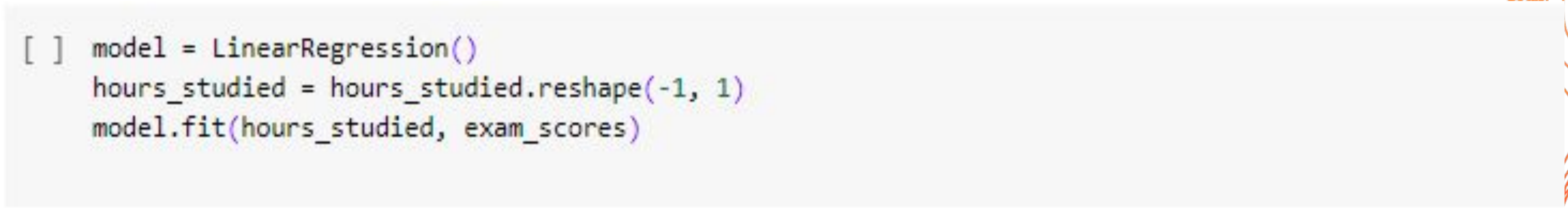
Explanation: "We're painting a picture of our data. The x-axis shows hours studied, the y-axis shows exam scores. A scatter plot gives us a visual story of how they relate.

Visualization



Creating the Model

Introducing the star of our show - Linear Regression!

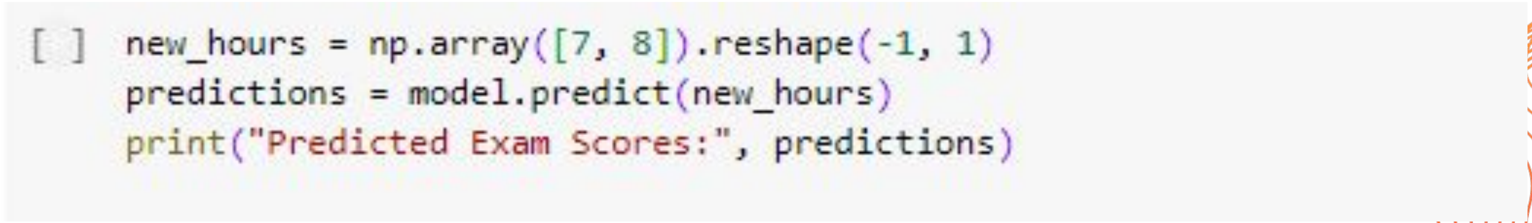


```
[ ]  model = LinearRegression()
     hours_studied = hours_studied.reshape(-1, 1)
     model.fit(hours_studied, exam_scores)
```

Explanation: Our main character - Linear Regression! We're reshaping the data and teaching the model using the fit method. It's like training our ML superhero to understand the relationship between hours studied and exam scores.

Making Predictions

We're predicting the future!



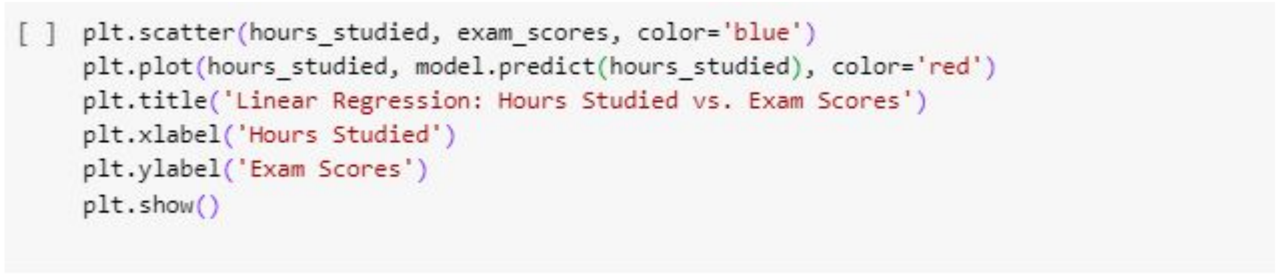
```
[ ]  new_hours = np.array([7, 8]).reshape(-1, 1)
     predictions = model.predict(new_hours)
     print("Predicted Exam Scores:", predictions)
```

Explanation: Time to put our model to the test! We're giving it new hours to study, and it's predicting the corresponding exam scores. It's like asking our ML fortune teller for predictions.

Visualizing the Regression Line

Caption: Time to draw the line between reality and our predictions!"

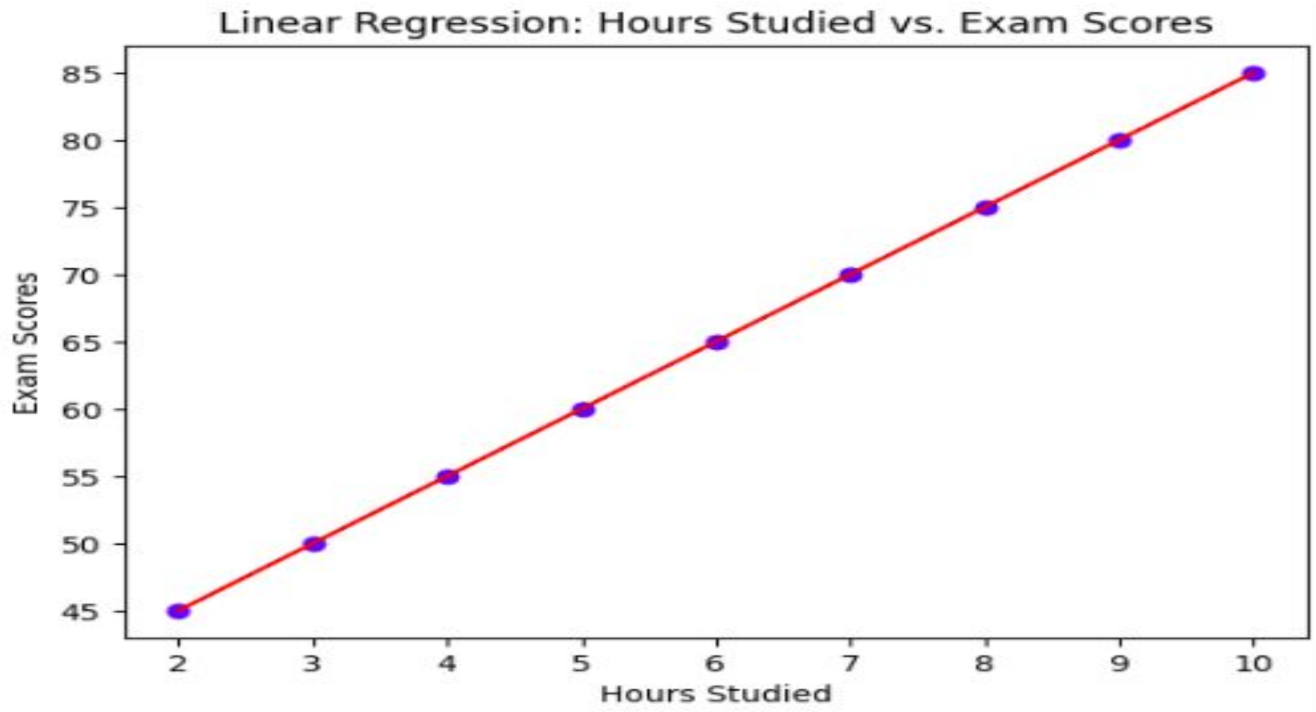
Code snippet for plotting the regression line:



```
[ ]  plt.scatter(hours_studied, exam_scores, color='blue')
     plt.plot(hours_studied, model.predict(hours_studied), color='red')
     plt.title('Linear Regression: Hours Studied vs. Exam Scores')
     plt.xlabel('Hours Studied')
     plt.ylabel('Exam Scores')
     plt.show()
```

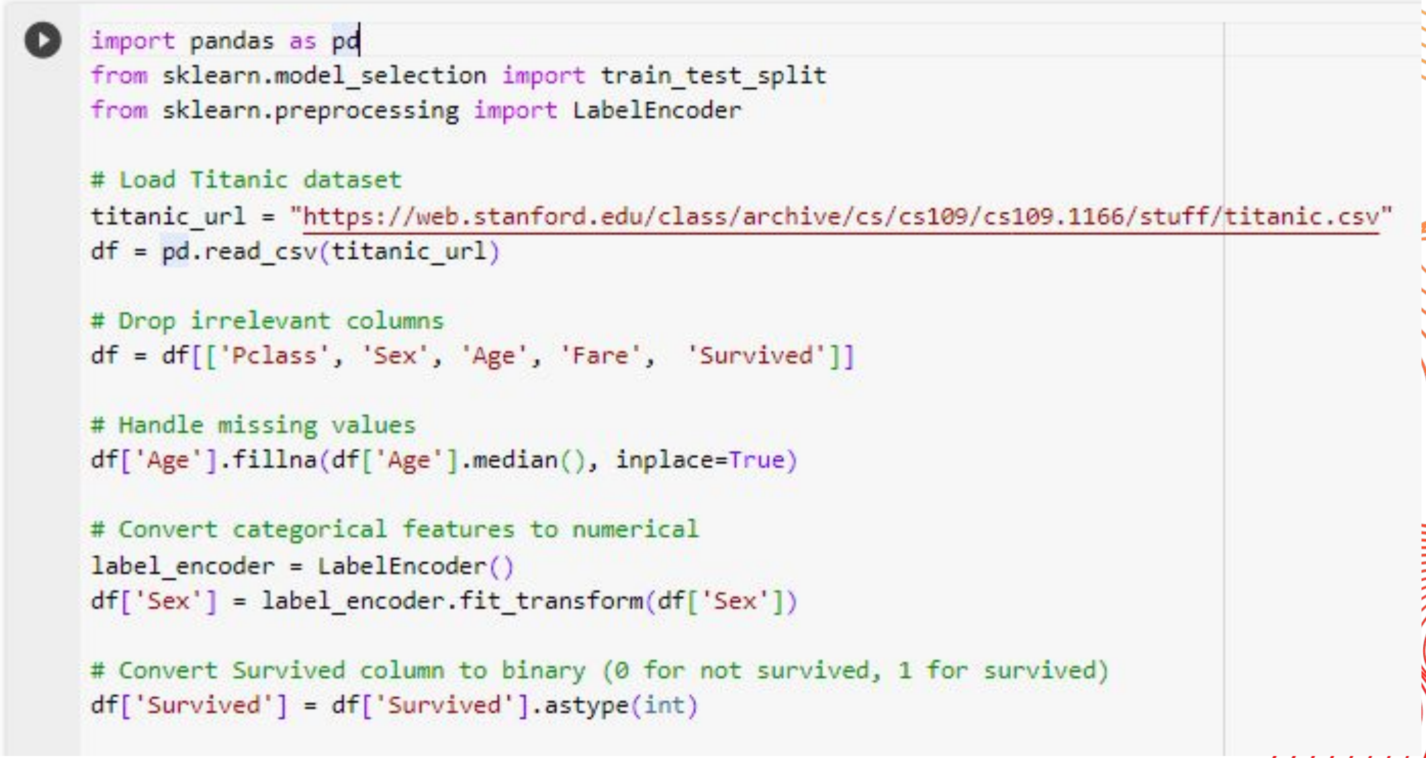
The scatter plot shows our actual data, and the red line represents our model's predictions - it's like drawing the best-fit line through our data points.

Visualizing the regression line



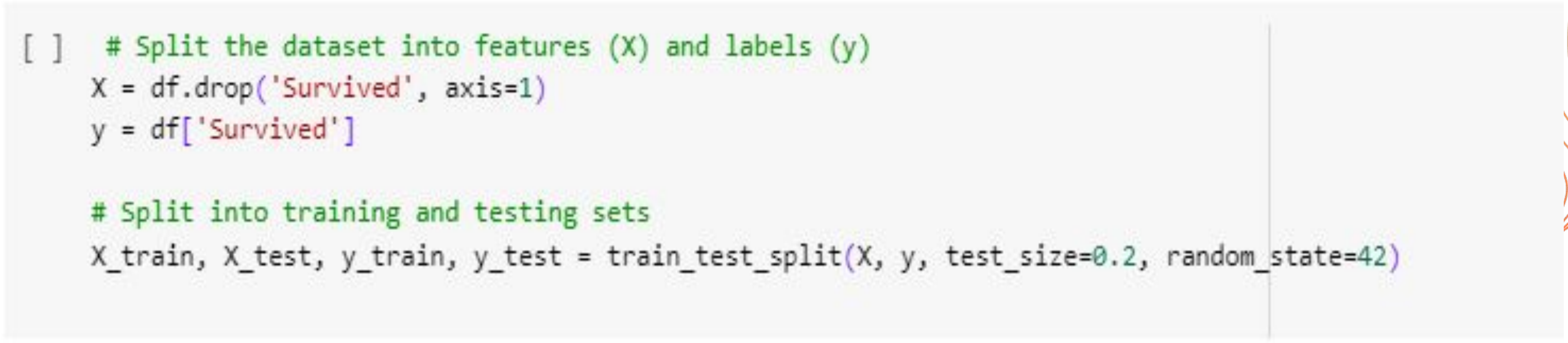
LOGISTIC REGRESSION CODING

Import and clean



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load Titanic dataset
titanic_url = "https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv"
df = pd.read_csv(titanic_url)

# Drop irrelevant columns
df = df[['Pclass', 'Sex', 'Age', 'Fare',  'Survived']]

# Handle missing values
df['Age'].fillna(df['Age'].median(), inplace=True)

# Convert categorical features to numerical
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])

# Convert Survived column to binary (0 for not survived, 1 for survived)
df['Survived'] = df['Survived'].astype(int)
```

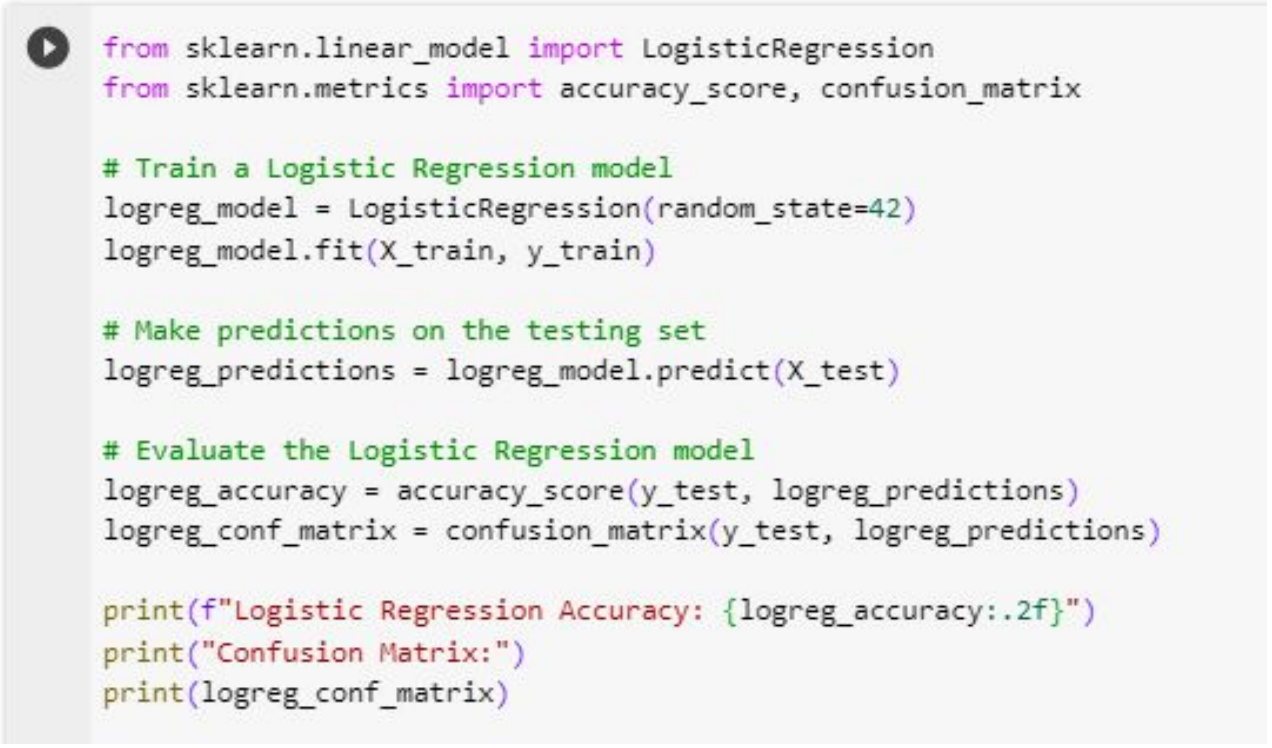
Split



```
[ ]  # Split the dataset into features (X) and labels (y)
    X = df.drop('Survived', axis=1)
    y = df['Survived']

    # Split into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Split dataset into training and testing.

Model, predict and Evaluation



```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Train a Logistic Regression model
logreg_model = LogisticRegression(random_state=42)
logreg_model.fit(X_train, y_train)

# Make predictions on the testing set
logreg_predictions = logreg_model.predict(X_test)

# Evaluate the Logistic Regression model
logreg_accuracy = accuracy_score(y_test, logreg_predictions)
logreg_conf_matrix = confusion_matrix(y_test, logreg_predictions)

print(f"Logistic Regression Accuracy: {logreg_accuracy:.2f}")
print("Confusion Matrix:")
print(logreg_conf_matrix)
```

Discussion

That's a wrap! Feel free to ask anything or experiment with the code. Understanding these steps is like having the backstage pass to machine learning tricks.

Presented by: Khadijah S. Mohammed
Khadijah@culminatech.com