**MODEL \_\_EXPLAINATION**

**Line 1:**  
import pandas as pd  
➡ Pandas ek powerful library hai jo data analysis aur CSV file read/write ke liye use hoti hai. Hum isse data ko load karke us par kaafi operations kar sakte hain.

**Line 2:**  
import seaborn as sns  
➡ Seaborn ek visualization library hai jo beautiful and informative graphs banane ke liye use hoti hai. Hum isse heatmap jaise plots banate hain.

**Line 3:**  
import matplotlib.pyplot as plt  
➡ Matplotlib ka pyplot module graphs jaise bar charts, histograms banane ke liye use hota hai. Yeh visualization ke liye zaroori hai.

**Line 4:**  
from sklearn.model\_selection import train\_test\_split  
➡ Is function ka use data ko training aur testing parts mein divide karne ke liye hota hai. Model training ke liye training data aur accuracy check karne ke liye testing data use hota hai.

**Line 5:**  
from sklearn.svm import SVR  
➡ SVR yaani Support Vector Regression ek regression model hai jo numerical data predict karta hai. Jaise: hours padhai = score.

**Line 6:**  
from sklearn.preprocessing import StandardScaler  
➡ StandardScaler ka use features ko scale karne ke liye hota hai taake unka range same ho jaye. Scaling models ke liye important hoti hai.

**Line 7:**  
from sklearn.pipeline import make\_pipeline  
➡ Pipeline banane ke liye use hota hai jisme multiple steps (jaise scaling + model) ko ek flow mein combine kiya jaata hai.

**Line 8:**  
import pickle  
➡ Pickle Python ka built-in module hai jiska use trained model ko file mein save karne aur baad mein load karne ke liye hota hai.

**Line 9:**  
class StudentScorePredictor:  
➡ Yeh ek custom class hai jisme sari functionalities defined hongi – jaise data load, analysis, model training aur saving.

**Line 10:**  
def \_\_init\_\_(self, filepath):  
➡ Constructor function hai, jab bhi object create hoga toh file path ko input lega aur class ke andar save karega.

**Line 11:**  
self.filepath = filepath  
➡ Yeh file path ko class ke variable mein store karta hai taake baad mein use ho sake.

**Line 12:**  
self.data = None  
➡ Data ko initially empty set kiya gaya hai. Isme baad mein CSV file se load kiya gaya data store hoga.

**Line 13:**  
self.model = None  
➡ Model ko bhi initially None set kiya gaya hai, baad mein trained model is variable mein store hoga.

**Line 14:**  
def load\_and\_understand\_data(self):  
➡ Yeh method data load karta hai aur data ka basic understanding deta hai – jaise structure, null values, etc.

**Line 15:**  
print("-----------Loading dataset-----------")  
➡ Console pe yeh message show karta hai taake user ko pata chale ke ab data load ho raha hai.

**Line 16:**  
self.data = pd.read\_csv(self.filepath)  
➡ CSV file ko read karta hai aur pandas DataFrame mein convert karta hai taake data ko use kiya ja sake.

**Line 17:**  
print("First 5 rows of the dataset:")  
➡ Console mein message print karta hai jisse pata chale ki ab data ka preview dikhaya jaega.

**Line 18:**  
print(self.data.head())  
➡ DataFrame ki pehli 5 rows show karta hai taake data kaisa dikhta hai woh samajh aaye.

**Line 19:**  
print("\\n-----------Dataset Info-----------:")  
➡ Message print karta hai jisse next output ka context mile – yeh dataset ki summary show karega.

**Line 20:**  
print(self.data.info())  
➡ Data ka structure, columns, data types, null values etc. dikhata hai. Yeh step data understanding ke liye important hai.

**Line 21:**  
print("\\n-----------Missing Values-----------:")  
➡ Message print karta hai taake user samjhe ki ab missing values check hone wali hain.

**Line 22:**  
print(self.data.isnull().sum())  
➡ Har column mein kitni missing values hain yeh count karke print karta hai.

**Line 23:**  
def perform\_univariate\_analysis(self):  
➡ Yeh method individual columns ka analysis karta hai – jisme statistics aur histograms banaye jaate hain.

**Line 24:**  
print("\\n-----------Univariate Analysis-----------:")  
➡ Console mein heading print karta hai taake output samajhne mein asani ho.

**Line 25:**  
print(self.data.describe())  
➡ Describe function data ka mean, min, max, standard deviation waghera dikhata hai.

**Line 26:**  
self.data.hist(figsize=(10, 6))  
➡ Sare numeric columns ke histograms plot karta hai jisse data distribution samajh aata hai.

**Line 27:**  
plt.tight\_layout()  
➡ Plot ka layout thoda adjust karta hai taake overlapping na ho aur clear dikhe.

**Line 28:**  
plt.show()  
➡ Sare plots display karta hai.

**Line 29:**  
def perform\_bivariate\_analysis(self):  
➡ Yeh method do features ke beech ka relation check karta hai, jaise Hours aur Scores.

**Line 30:**  
print("\\n -----------Bivariate Analysis (Correlation Heatmap bw Hours and study)-----------:")  
➡ Message print karta hai jo batata hai ke ab correlation heatmap show hoga.

**Line 31:**  
plt.figure(figsize=(10, 6))  
➡ Figure ka size set karta hai jisse heatmap properly dikhe.

**Line 32:**  
sns.heatmap(self.data.corr(), annot=True, cmap='coolwarm')  
➡ Correlation matrix plot karta hai aur heatmap ke form mein show karta hai.

**Line 33:**  
plt.title('Feature Correlation Heatmap')  
➡ Plot ka title set karta hai.

**Line 34:**  
plt.show()  
➡ Heatmap display karta hai.

**Line 35:**  
def split\_data(self, target\_column, test\_size=0.2):  
➡ Data ko training aur testing sets mein split karne ke liye function banaya gaya hai.

**Line 36:**  
print("\\nSplitting data into train and test sets...")  
➡ Message print karta hai taake user ko pata chale ke data split ho raha hai.

**Line 37:**  
X = self.data.drop(columns=[target\_column])  
➡ Target column ko hata ke baaki columns ko input features (X) ke roop mein store karta hai.

**Line 38:**  
y = self.data[target\_column]  
➡ Target column ko y (output variable) ki shkl mein store karta hai.

**Line 39:**  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=42)  
➡ Data ko training aur testing parts mein split karta hai. 80% training aur 20% testing (default) hota hai.

**Line 40:**  
print("Data split completed.")  
➡ Confirmation message deta hai ki data ka split ho chuka hai.

**Line 41:**  
return X\_train, X\_test, y\_train, y\_test  
➡ Split kiye gaye data ko return karta hai taake baad mein use kiya ja sake.

**Line 42:**  
def train\_model(self, X\_train, y\_train):  
➡ Model ko train karne ke liye function banaya gaya hai.

**Line 43:**  
print("\\nTraining SVM model...")  
➡ Message print karta hai ki ab model training start hone wali hai.

**Line 44:**  
self.model = make\_pipeline(StandardScaler(), SVR())  
➡ Ek pipeline banata hai jisme pehle data scale hota hai aur phir SVR model use hota hai.

**Pipeline kya hai? (What is Pipeline?)**

Pipeline ek tarah ka process flow hota hai jisme multiple steps ko ek sequence mein jod dete hain.

Jaise tumhara data pehle scale hoga (normalize karna), phir model train hoga, aur fir prediction hoga — sab steps ek sath manage karne ke liye Pipeline use karte hain.

Isse fayda ye hota hai ki code clean aur easy to manage ho jata hai. Aur agar tum future mein naye steps add karna chaho, toh wo asaani se ho jata hai.

**Simple example:**  
Data → Scaling → Model Training → Prediction — Sab ek flow mein.

**StandardScaler kya hai? (What is StandardScaler?)**

StandardScaler ek data preprocessing technique hai jo data ke features ko normalize karta hai.

Matlab, yeh har feature (column) ka mean 0 aur standard deviation 1 banata hai.

Iska matlab hota hai ki features same scale pe aate hain, jisse machine learning model ko train karne mein asaani hoti hai.

**Kyoon use karte hain?**  
Kyunki agar features ke scale bahut alag-alag honge (jaise ek feature 1-10 ke beech ho aur doosra 1000-10000 ke beech), toh model confuse ho sakta hai aur ache results nahi dega.

StandardScaler se data uniform scale pe aa jata hai, jisse model ki performance better hoti hai.

**Line 45:**  
self.model.fit(X\_train, y\_train)  
➡ Training data se model ko train karta hai.

**Line 46:**  
print("Model trained successfully.")  
➡ Confirmation deta hai ki model training complete ho chuki hai.

**Line 47:**  
def save\_model(self, filename='svm\_model.pkl'):  
➡ Trained model ko file mein save karne ke liye function banaya gaya hai.

**Line 48:**  
print(f"\\nSaving model to '{filename}'...")  
➡ Message print karta hai jo batata hai ki model save ho raha hai.

**Line 49:**  
with open(filename, 'wb') as file:  
➡ File open karta hai binary mode mein jisme model save kiya jaega.

**Line 50:**  
pickle.dump(self.model, file)  
➡ Model ko file mein dump/save karta hai.

**Line 51:**  
print("Model saved successfully.")  
➡ Confirmation message deta hai ki model file mein save ho chuka hai.

**Line 52:**  
def predict(self, X):  
➡ Naye data ke upar prediction karne ke liye function banaya gaya hai.

**Line 53:**  
if self.model:  
➡ Check karta hai ki model trained hai ya nahi.

**Line 54:**  
return self.model.predict(X)  
➡ Trained model se prediction return karta hai.

**Line 55:**  
else:  
➡ Agar model trained nahi hai toh error dega.

**Line 56:**  
raise Exception("Model not trained yet.")  
➡ Error message raise karta hai agar model train nahi hua ho.

**Line 57:**  
if \_\_name\_\_ == "\_\_main\_\_":  
➡ Yeh check karta hai ki file directly run ho rahi hai ya import ki gayi hai. Sirf direct run hone par code execute hoga.

**Line 58:**  
predictor = StudentScorePredictor("score.csv")  
➡ Class ka object banata hai aur "score.csv" file path ke sath initialize karta hai.

**Line 59:**  
predictor.load\_and\_understand\_data()  
➡ Data ko load karta hai aur uska structure samjhata hai.

**Line 60:**  
predictor.perform\_univariate\_analysis()  
➡ Univariate analysis karta hai jaise histogram aur describe() function.

**Line 61:**  
predictor.perform\_bivariate\_analysis()  
➡ Correlation heatmap ke through feature relationships ko dikhata hai.

**Line 62:**  
X\_train, X\_test, y\_train, y\_test = predictor.split\_data(target\_column="Scores")  
➡ Data ko input/output mein split karta hai aur training/testing ke liye divide karta hai.

**Line 63:**  
predictor.train\_model(X\_train, y\_train)  
➡ Model ko training data par train karta hai.

**Line 64:**  
predictor.save\_model()  
➡ Trained model ko file mein save karta hai future use ke liye.