AIM: Predicting math's score using linear reg model.

Dataset → Lunch Type
Gender
Race_ethnicity
parental_level_of_education
test_preparation_course

Numerical col [writing_score
reading_score
Math_score

↓
Standard scaler
↓
At first used simple imputer to find the missing values then put median into it.
↓
Then used standard scaler for standardization [-3 to 3]

categorical col
↓
One hot encoder
↓
gives diff bin col as per the given input

there (strategy = "median")
↳ if here strategy = "mean" then we put mean values in place of missing val

Normalization [-1 to 1]
↓
Normal distribution

```
preprocessor = ColumnTransformer(
    [
        ("OneHotEncoder", oh_transformer, cat_features),
        ("StandardScaler", numeric_transformer, num_features)
    ]
)
```

Column Transformer is used for applying diff pre-processing steps to diff col of pandas. It allows to perform column specific transformations as a part of pipeline, which is particularly useful when you have mixed datatype and need to apply diff preprocessing techniques to each type.

mixed datatype
- Numerical data Columns
- Categorical data columns

Pipeline = Is a tool that allows multiple steps (such as preprocessing & model training) into a single cohesive workflow.

X = preprocessor.fit_transform(X)

fit_transform() → Is used to combine 2 steps fitting & transforming data.

learn something from the training data	then apply that learned transformation to the same data
Here it will compute the mean and standard deviation from the training data	Standard scaler will use the computed mean & standard deviation to scale the data

fit() → In this step

The OneHotEncoder will examine the data to identify all the unique categories in each feature (column)
It learns the unique categories to know how many binary columns will be needed.

In transform() step

The encoder transforms the original ~~step~~ data by creating a binary column for each unique category

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

X_train = 80% of the ~~training~~ dataset used for training
X_test = 20% of the dataset used for testing

Y_train = 80% of Math score
Y_test = 20% of Math score
test_size = 0.2 → splitting to 80 – 20

random_state = 42 "Here the random selection of value ~~from~~ for training & testing will be same

If we will not mention random state then it will

take values randomly from internally which will show unpredictible machine behaviour.

X_train.shape

Suppose X_train is a matrix with 100 rows and 5 cols. The shape would look like

X_train.shape
Output: (100, 5)

```
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true,
                predicted))
    r2_square = r2_score(true, predicted)
    return mae, rmse, r2_square
```

```
models = {
    "Linear Regression" : LinearRegression(),
    "Lasso" : Lasso(),
    "Ridge" : Ridge(),
    "K-Neighbors Regressor" : KNeighborsRegressor(),
    "DecisionTree" : DecisionTreeRegressor(),
    "Random forest Regressor" : RandomforestRegressor(),
    "XGBRegressor" : XGBRegressor(),
    "CatBoosting Regressor" : CatBoostRegressor
            (verbose = False),
    "AdaBoost Regressor" : AdaBoostRegressor()
}

model_list = []
r2_list = []

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) → training

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test) → testing

    model_train_mae, model_train_rmse,
    model_train_r2 = evaluate_model(y_train,
        y_train_pred)

    model_test_mae, model_test_rmse, model_test_
    r2 = evaluate_model(y_test, y_test_pred)
```

```
print(list(models.keys())[i])
model_list.append(list(models.keys())[i])

print("Model Performance for Training set")
print("- RMSE : {:.4f}".format(model_train_rmse))
print("- Mean Absolute Error : {:.4f}".format(model_train_mae))
print("- R2 Score : {:.4f}".format(model_train_r2))

print("----------")

print("Model Performance for Test set")
print("-Root Mean Squared Error : {:.4f}".format(model_test_rmse))
print("- MAE : {:.4f}".format(model_test_mae))
~~r2_list.appen~~
print("-R2 score : {:.4f}".format(model_test_r2))
r2_list.append(model_test_r2)

print('='*35)
print('\n')
```

```
pd.DataFrame(list(zip(model_list, r2_list)),
    columns=['Model Name', 'R2_Score'])
    sort_values(by=["R2_score"], ascending=False)
```

Model Name	R2-score
Ridge	0.880593
Linear Regression	0.880345

Linear Regression

```
lin_model = LinearRegression(fit_intercept=True)
lin_model = lin_model.fit(X_train, y_train)
y_pred = lin_model.predict(X_test)
score = r2_score(y_test, y_pred) * 100
print("Accuracy : %.2f" %score)
```

Accuracy : 88.03