

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

Dataset → Gender

Lunch Type

Race, ethnicity

parental level of education

test preparation course

writing score

reading score

Math score

Categorical

col

↓

one

not

encoder

↓

gives diff  
bin col  
as per the  
given input

Numerical  
col

Standard  
Scaler

At first used  
simple imputer  
to find the  
missing values  
then put  
median into it.

Here (Strategy = median)

if there  
strategy = mean  
then we put  
mean values  
in place of  
missing val

Then used  
standard scaler  
for standardization  
[-3 to 3]

Normalization  
[0 to 1]

Normal  
distribution

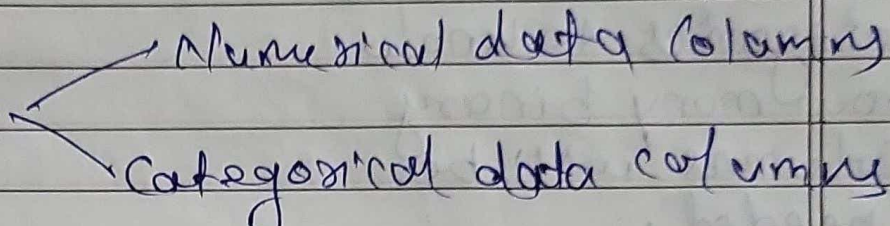
preprocessor = ColumnTransformer(  
[

("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 column  
Categorical data column

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

$X = \text{preprocessor.fit\_transform}(X)$

$\text{fit\_transform}()$   $\rightarrow$  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 ~~the~~ scale the data



fit()  $\rightarrow$  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 data by creating a binary column for each unique category

$X_{train}, X_{test}, y_{train}, y_{test} = \text{train\_test\_split}(X, y, \text{test\_size}=0.2, \text{random\_state}=42)$

$X_{train}$  = 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

$\text{test\_size}=0.2 \rightarrow$  splitting to 80-20

$\text{random\_state}=42$  " Here the random selection of value from training & testing will be same for

If we will not mention random state then it will



take values randomly from internally  
which will show unpredictable 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(),
    "RandomForest Regressor": RandomForestRegressor(),
    "XGBRegressor": XGBRegressor(),
    "Cat Boosting 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 = test.append
```

```
print("- R2 score : {:.4f}".format(model.test_r2))
```

```
r2_list.append(model.test_r2)
```

```
print("\n * 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