Student Performance Analysis and Prediction

BEGINNER CLASSIFICATION DATA ANALYSIS PYTHON REGRESSION STATISTICS

Introduction

This article uses to predict student performance. For many applications, including online customer service, marketing, and finance, the stock price is a crucial challenge. Given a student's performance using big organizations and institutions, it can be difficult to come up with a Student performance analysis and prediction system that is accurate across all models. This article will discuss how student performance prediction can solve this problem. Here we will deal with student performance analysis and prediction with help of a dataset.

Read more on what is predictive analytics for beginners here.

This article was published as a part of the Data Science Blogathon.

Table of Contents

Understanding the Problem Statement

This project understands how the student's performance (test scores) is affected by other variables such as Gender, Ethnicity, Parental level of education, and Lunch and Test preparation course.

The primary objective of higher education institutions is to impart quality education to their students. To achieve the highest level of quality in the education system, knowledge must be discovered to predict student enrollment in specific courses, identify issues with traditional classroom teaching models, detect unfair means used in online examinations, detect abnormal values in student result sheets, and predict student performance. This knowledge is hidden within educational datasets and can be extracted through data mining techniques.

This project focuses on evaluating students' capabilities in various subjects using a classification task. Data classification has many approaches, and the decision tree method and probabilistic classification method are utilized here. By performing this task, knowledge is extracted that describes students' performance in the end-semester examination. This helps in identifying dropouts and students who require special attention, enabling teachers to provide appropriate advising and counseling.

Data Collection

Dataset Source – Students performance dataset.csv. The data consists of 8 column and 1000 rows.

Import Data and Required Packages

Importing Pandas, Numpy, Matplotlib, Seaborn and Warings Library.

```
import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns import warnings warnings.filterwarnings("ignore")
```

Import the CSV Data as Pandas DataFrame

```
df = pd.read_csv("data/StudentsPerformance.csv")
```

Show the top 5 Recoreds

```
df.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

show the top 5 records on the dataset and look at the features.

To see the shape of the dataset

```
df.shape
```

```
(1000, 8)
```

And it will help to find the shape of the dataset.

Dataset Information

- gender: sex of students -> (Male/female)
- race/ethnicity: ethnicity of students -> (Group A, B, C, D, E)
- parental level of education: parents' final education ->(bachelor's degree, some college, master's degree, associate's degree)
- lunch: having lunch before test (standard or free/reduced)
- test preparation course: complete or not complete before test
- math score
- reading score
- writing score

After that, we check the data as the next step. There are a number of categorical features contained in the dataset, including multiple missing value kinds, duplicate values, check data types, and a number of unique value types.

Data Checks to Perform

- Check Missing values
- Check Duplicates
- Check data type
- Check the number of unique values in each column
- Check the statistics of the data set
- Check various categories present in the different categorical column

Check Missing Values

To check every column of the missing values or null values in the dataset.

```
df.isnull().sum()
```

If there are no missing values in the dataset.

Check Duplicates

If checking the our dataset has any duplicated values present or not

```
df.duplicated().sum()
```

There are no duplicates values in the dataset.

Check the Data Types

To check the information of the dataset like datatypes, any null values present in the dataset.

```
#check the null and Dtypes df.info()
```

Check the Number of Unique Values in Each Column

```
df.nunique()
```

Check Statistics of the Data Set

To examine the dataset’s statistics and determine the data’s statistics.

Insight

- The numerical data shown above shows that all means are fairly similar to one another, falling between 66 and 68.05.
- The range of all standard deviations, between 14.6 and 15.19, is also narrow.
- While there is a minimum score of 0 for math, the minimums for writing and reading are substantially higher at 10 and 17, respectively.
- We don’t have any duplicate or missing values, and the following codes will provide a good data checking.

Exploring Data

```
print("Categories in 'gender' variable: ",end=" ") print(df["gender"].unique()) print("Categories in 'race/ethnicity' variable: ",end=" ") print(df["race/ethnicity"].unique()) print("Categories in 'parental level of education' variable: ",end=" ") print(df["parental level of education"].unique()) print("Categories in 'lunch' variable: ",end=" ") print(df["lunch"].unique()) print("Categories in 'test preparation course' variable: ",end=" ") print(df["test preparation course"].unique())
```

The unique values in the dataset will be provided and presented in a pleasant way in the code above.

The output will following:

We define the numerical and categorical columns:

```
#define numerical and categorical columns numeric_features = [feature for feature in df.columns if df[feature].dtype != "object"] categorical_features = [feature for feature in df.columns if df[feature].dtype == "object"] print("We have {} numerical features: {}".format(len(numeric_features),numeric_features)) print("We have {} categorical features: {}".format(len(categorical_features),categorical_features))
```

The above code will use separate the numerical and categorical features and count the feature values.

Exploring Data (Visualization)

Visualize Average Score Distribution to Make Some Conclusion

- Histogram
- Kernel Distribution Function (KDE)

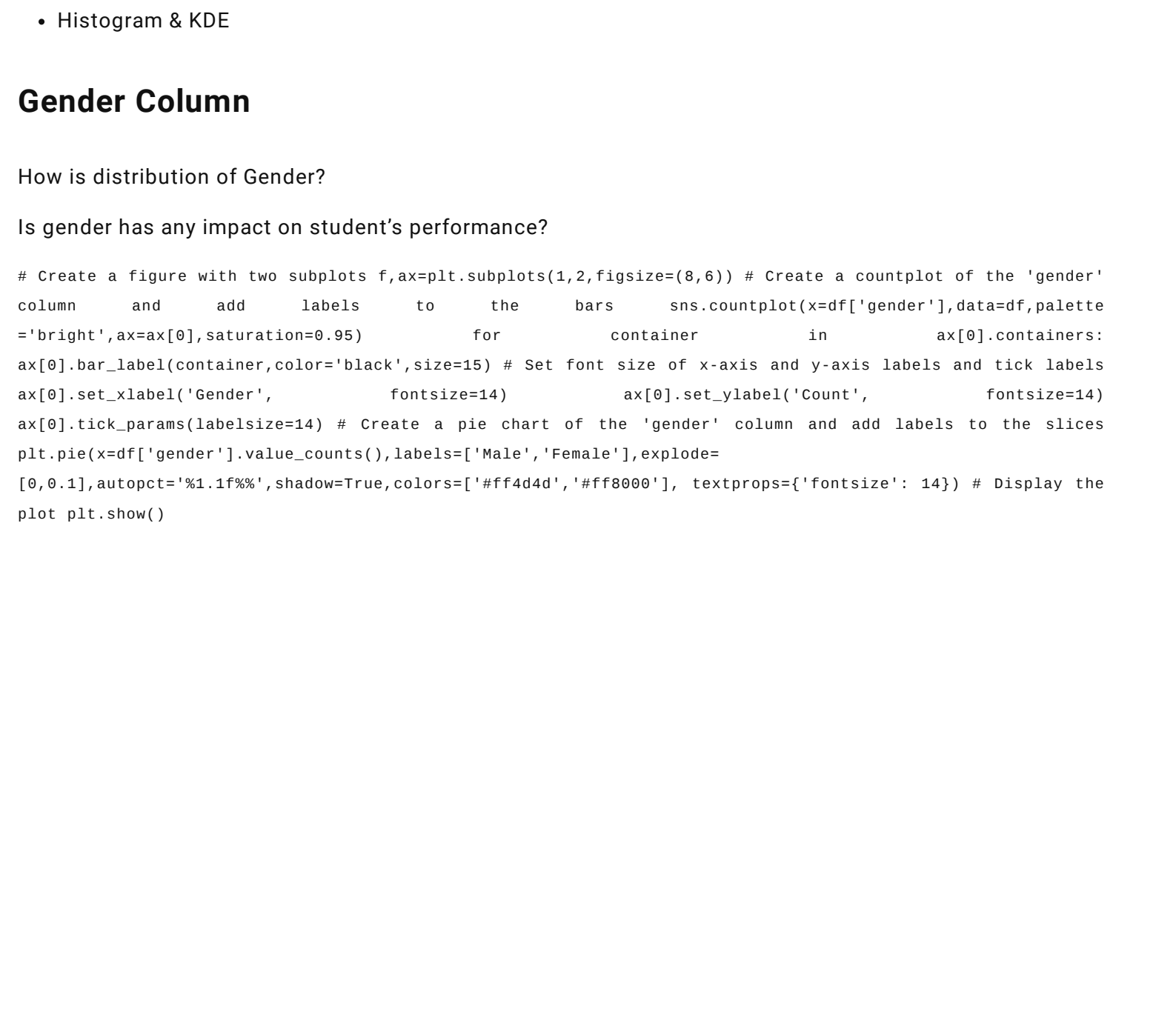
- Histogram & KDE

Gender Column

How is distribution of Gender?

Is gender has any impact on student's performance?

```
# Create a figure with two subplots f,ax=plt.subplots(1,2,figsize=(8,6)) # Create a countplot of the 'gender' column and add labels to the bars sns.countplot(x=df['gender'],data=df,palette ='bright',ax=ax[0],saturation=0.95) for container in ax[0].containers: ax[0].bar_label(container,color='black',size=15) # Set font size of x-axis and y-axis labels and tick labels ax[0].set_xlabel('Gender', fontsize=14) ax[0].set_ylabel('Count', fontsize=14) ax[0].tick_params(labelsize=14) # Create a pie chart of the 'gender' column and add labels to the slices plt.pie(x=df['gender'].value_counts(),labels=['Male','Female'],explode= [0,0.1],autopct='%1.1f%%',shadow=True,colors=['#ff4d4d','#ff8000'], textprops={'fontsize': 14}) # Display the plot plt.show()
```



Gender has balanced data with female students are 518 (48%) and male students are 482 (52%)

Race/Ethnicity Column

```
# Define a color palette for the countplot colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd'] # blue, orange, green, red, purple are respectiively the color names for the color codes used above # Create a figure with two subplots f, ax = plt.subplots(1, 2, figsize=(12, 6)) # Create a countplot of the 'race/ethnicity' column and add labels to the bars sns.countplot(x=df['race/ethnicity'], data=df, palette=colors, ax=ax[0], saturation=0.95) for container in ax[0].containers: ax[0].bar_label(container, color='black', size=14) # Set font size of x-axis and y-axis labels and tick labels ax[0].set_xlabel('Race/Ethnicity', fontsize=14) ax[0].set_ylabel('Count', fontsize=14) ax[0].tick_params(labelsize=14) # Create a dictionary that maps category names to colors in the color palette color_dict = dict(zip(df['race/ethnicity'].unique(), colors)) # Map the colors to the pie chart slices pie_colors = [color_dict[race] for race in df['race/ethnicity'].value_counts().index] # Create a pie chart of the 'race/ethnicity' column and add labels to the slices plt.pie(x=df['race/ethnicity'].value_counts(), labels=df['race/ethnicity'].value_counts().index, explode=[0.1, 0, 0, 0, 0], autopct='%1.1f%%', shadow=True, colors=pie_colors, textprops={'fontsize': 14}) # Set the aspect ratio of the pie chart to 'equal' to make it a circle plt.axis('equal') # Display the plot plt.show()
```

id = Insights>Insights

- Most of the student belonging from group C /group D.
- Lowest number of students belong to group A.

Parental Level of Education Column

```
plt.rcParams['figure.figsize'] = (15, 9) plt.style.use('fivethirtyeight') sns.histplot(df["parental level of education"], palette = 'Blues') plt.title('Comparison of Parental Education', fontweight = 30, fontsize = 20) plt.xlabel('Degree') plt.ylabel('count') plt.show()
```

id = Insights>Insights

- Largest number of parents are from college.

Bivariate Analysis

```
df.groupby('parental level of education').agg('mean').plot(kind='barh',figsize=(10,10))
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.) plt.show()
```

id = Insights>Insights

- The score of student whose parents possess master and bachelor level education are higher than others.

Maximum Score of Students in All Three Subjects

```
plt.fig


figsize=(18,8))     plt.subplot(1,     4,     1)     plt.title('MATH     SCORES')     sns.violinplot(y='math score',data=df,color='red',linewidth=3)   plt.subplot(1,   4,   2)   plt.title('READING   SCORES')   sns.violinplot(y='reading  score',data=df,color='green',linewidth=3)  plt.subplot(1,  4,  3)  plt.title('WRITING  SCORES')  sns.violinplot(y='writing score',data=df,color='blue',linewidth=3) plt.show()
```

id = Insights>Insights

From the above three plots its clearly visible that most of the students score in between 60-80 in Maths whereas in reading and writing most of them score from 50-80.

Multivariate Analysis Using Pie Plot

```
# Set figure size plt.rcParams['figure.figsize'] = (12, 9) # First row of pie charts plt.subplot(2, 3, 1) size = df['gender'].value_counts() labels = 'Female', 'Male' color = ['red','green'] plt.pie(size, colors=color, labels=labels, autopct='%.2f%%') plt.title('Gender', fontsize=20) plt.axis('off') plt.subplot(2, 3, 2) size = df['race/ethnicity'].value_counts() labels = 'Group C', 'Group D', 'Group B', 'Group E', 'Group A' color = ['red', 'green', 'blue', 'cyan', 'orange'] plt.pie(size, colors=color, labels=labels, autopct='%.2f%%') plt.title('Race/Ethnicity', fontsize=20) plt.axis('off') plt.subplot(2, 3, 3) size = df['lunch'].value_counts() labels = 'Standard', 'Free' color = ['red', 'green'] plt.pie(size, colors=color, labels=labels, autopct='%.2f%%') plt.title('Lunch', fontsize=20) plt.axis('off') # Second row of pie charts plt.subplot(2, 3, 4) size = df['test preparation course'].value_counts() labels = 'None', 'Completed' color = ['red', 'green'] plt.pie(size, colors=color, labels=labels, autopct='%.2f%%') plt.title('Test Course', fontsize=20) plt.axis('off') plt.subplot(2, 3, 5) size = df['parental level of education'].value_counts() labels = 'Some College', "Associate's Degree", 'High School', 'Some High School', "Bachelor's Degree", "Master's Degree" color = ['red', 'green', 'blue', 'cyan', 'orange', 'grey'] plt.pie(size, colors=color, labels=labels, autopct='%.2f%%') plt.title('Parental Education', fontsize=20) plt.axi
```

```
ff') # Remove extra subplot plt.subplot(2, 3, 6).remove() # Add super title plt.suptitle('Comparison of Student Attributes', fontsize=20, fontweight='bold') # Adjust layout and show plot # This is removed as there are only 5 subplots in this figure and we want to arrange them in a 2x3 grid. # Since there is no 6th subplot, it is removed to avoid an empty subplot being shown in the figure. plt.tight_layout() plt.subplots_adjust(top=0.85) plt.show()
```

id = Insights>Insights

- The number of Male and Female students is almost equal.
- The number of students is higher in Group C.
- The number of students who have standard lunch is greater.
- The number of students who have not enrolled in any test preparation course is greater.
- The number of students whose parental education is “Some College” is greater followed closely by “Associate’s Degree”.

From the above plot, it is clear that all the scores increase linearly with each other.

Student’s Performance is related to lunch, race, and parental level education.

- Females lead in pass percentage and also are top-scorers.
- Student Performance is not much related to test preparation course.
- The finishing preparation course is beneficial.

Model Training

Import Data and Required Packages

Importing scikit library algorithms to import regression algorithms.

```
# Modelling from sklearn.metrics import mean_squared_error, r2_score from sklearn.neighbors import KNeighborsRegressor from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor from sklearn.svm import SVR from sklearn.linear_model import LinearRegression,Lasso from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error from sklearn.model_selection imp


RandomizedSearchCV from catboost import CatBoostRegressor from xgboost import XGBRegressor import warnings
```

Splitting the X and Y Variables

This separation of the dependent variable(y) and independent variables(X) is one the most important in our project we use the math score as a dependent variable. Because so many students lack in math subjects it will almost 60% to 70% of students in classes 7-10 students are fear of math subjects that's why I am choosing the math score as a dependent score.

It will use to improve the percentage of math scores and increase the grad f students and also remove fear in math.

```
X = df.drop(columns="math score",axis=1) y = df["math score"]
```

Create Column Transformer with 3 Types of Transformers

```
num_features = X.select_dtypes(exclude="object").columns cat_features = X.select_dtypes(include="object").columns from sklearn.preprocessing import OneHotEncoder,StandardScaler from sklearn.compose import ColumnTransformer numeric_transformer = StandardScaler() oh_transformer = OneHotEncoder() preprocessor = Column transformer( [ ("OneHotEncoder", oh_transformer, cat_features), ("StandardScaler", numeric_transformer, num_features), ] ) X = preprocessor.fit_transform(X)
```

Separate Dataset into Train and Test

To separate the dataset into train and test to identify the training size and testing size of the dataset.

```
from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42) X_train.shape, X_test.shape
```

Create an Evaluate Function for Model Training

This function is use to evaluate the model and build a good model.

```
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, mse, rmse, r2_square
```

To create a models variable and form a dictionary formate.

```
models = { "Linear Regression": LinearRegression(), "Lasso": Lasso(), "K-Neighbors Regressor": KNeighborsRegressor(), "Decision Tree": DecisionTreeRegressor(), "Random Forest Regressor": RandomForestRegressor(), "Gradient Boosting": GradientBoostingRegressor(), "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) # Train model # Make predictions y_train_pred = model.predict(X_train) y_test_pred = model.predict(X_test) # Evaluate Train and Test dataset model_train_mae, model_train_mse, model_train_rmse, model_train_r2 = evaluate_model(y_train, y_train_pred) model_test_mae, model_test_mse, 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("- Root Mean Squared Error: {:.4f}".format(model_train_rmse)) print("- Mean Squared Error: {:.4f}".format(model_train_mse)) 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("- Mean Squared Error: {:.4f}".format(model_test_rmse)) print("- Mean Absolute Error: {:.4f}".format(model_test_mae)) print("- R2 Score: {:.4f}". for model_test_r2)) r2_list.append(model_test_r2) print('='*35) print('\n')
```

The output of before tuning all algorithms’ hyperparameters. And it provides the RMSE, MSE, MAE, and R2 score values for training and test data.

Hyperparameter Tuning

It will give the model with most accurate predictions and improve prediction accuracy.

This will give the optimized value of hyperparameters, which maximize your model predictive accuracy.

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV from sklearn.metrics import make_scorer
# Define hyperparameter ranges for each model param_grid = { "Linear Regression": {}, "Lasso": {"alpha":
```

```
[1]}, "K-Neighbors Regressor": {"n_neighbors": [3, 5, 7],}, "Decision Tree": {"max_depth": [3, 5, 7],'criterion':['squared_error', 'friedman_mse', 'absolute_error', 'poisson']}, "Random Forest Regressor": {'n_estimators': [8,16,32,64,128,256], "max_depth": [3, 5, 7]}, "Gradient Boosting": {'learning_rate': [.1,.01,.05,.001],'subsample':[0.6,0.7,0.75,0.8,0.85,0.9], 'n_estimators': [8,16,32,64,128,256]}, "XGBRegressor": {'depth': [6,8,10],'learning_rate': [0.01, 0.05, 0.1],'iterations': [30, 50, 100]}, "CatBoosting Regressor": {"iterations": [100, 500], "depth": [3, 5, 7]}, "AdaBoost Regressor": {'learning_rate':[.1,.01,0.5,.001],'n_estimators': [8,16,32,64,128,256]} } model_list = [] r2_list =[] for model_name, model in models.items(): # Create a scorer object to use in grid search scorer = make_scorer(r2_score) # Perform grid search to find the best hyperparameters grid_search = GridSearchCV( model, param_grid[model_name], scoring=scorer, cv=5, n_jobs=-1 ) # Train the model with the best hyperparameters


rid_search.fit(X_train, y_train) # Make predictions y_train_pred = grid_search.predict(X_train) y_test_pred = grid_search.predict(X_test) # Evaluate Train and Test dataset model_train_mae, model_train_mse, model_train_rmse, model_train_r2 = evaluate_model(y_train, y_train_pred) model_test_mae, model_test_mse, model_test_rmse, model_test_r2 = evaluate_model(y_test, y_test_pred) print(model_name) model_list.append(model_name) print('Best hyperparameters:', grid_search.best_params_) print('Model performance for Training set') print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse)) print("- Mean Squared Error: {:.4f}".format(model_train_mse)) 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("- Mean Squared Error: {:.4f}".format(model_test_rmse)) print("- Mean Absolute Error: {:.4f}".format(model_test_mae)) print("- R2 Score: {:.4f}".for


model_test_r2)) r2_list.append(model_test_r2) print('='*35) print('\n')
```

Outputs

The output of after tuning all algorithms' hyperparameters. And it provides the RMSE, MSE, MAE, and R2 score values for training and test data.

If we choose Linear regression as the final model because that model will get a training set r2 score is 87.42 and a testing set r2 score is 88.03.

Model Selection

This is used to select the best model of all of the regression algorithms.

In linear regression, we got 88.03 curacy in all of the regression models that’s why we choose model.

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

Accuracy of the model is 88.03%

```
plt.scatter(y_test,y_pred) plt.xlabel('Actual') plt.ylabel('Predicted') plt.show()
```

```
sns.regplot(x=y_test,y=y_pred,ci=None,color ='red')
```

Difference Between Actual and Predicted Values

```
pred_df=pd.DataFrame({'Actual    Value':y_test,'Predicted    Value':y_pred,'Difference':y_test-y_pred})
pred_df
```

Convert the Model to Pickle File

```
# loading library import pickle # create an iterator object with write permission - model.pkl with open('model_pkl', 'wb') as files: pickle.dump(model, files) # load saved model with open('model_pkl' , 'rb') as f: lr = pickle.load(f)
```

Conclusion

This brings us to an end to the student's performance prediction. Let us review our work. First, we started by defining our problem statement, looking into the algorithms we were going to use and the regression implementation pipeline. Then we moved on to practically implementing the identification and regression algorithms like Linear Regression, Lasso, K-Neighbors Regressor, Decision Tree, Random Forest Regressor, XGBRegressor, CatBoosting Regressor, and AdaBoost Regressor. Moving forward, we compared the performances of these models. Lastly, we built a Linear regression model that proved that it works best for student performance prediction problems.

The key takeaways from this student performance prediction are:

- Identification of student performance prediction is important for many institutions.
- Linear regression gives better accuracy compared to other regression problems.
- Linear regression is the best fit for the problem
- Linear regression provides an accuracy of 88%, giving out the most accurate results.

I hope you like my article on “Student performance analysis and prediction.” The entire code can be found in my GitHub repository. You can connect with me here on LinkedIn.

Article Url - https://www.analyticsvidhya.com/blog/2023/04/student-performance-analysis-and-prediction/

Sai Battula