Data Preparation/Feature Engineering

1. Overview

Data Preprocessing and feature engineering are two crucial steps in the machine learning pipeline and while they are related and sometimes overlap, they serve different purposes and address different aspects of preparing data for machine learning models.

Data Preprocessing:

It involves cleaning and transforming raw data into a format that is suitable for building and training machine learning models. The main goal of data preprocessing is to improve the quality of the data by addressing issues such as:

1. **Missing Values**: Filling in or removing data points that have missing values.
2. **Noise reduction**: Smoothing out noisy data to make patterns more discernible.
3. **Normalization/Standardization**: Scaling numerical data to a standard range or distribution, which helps certain algorithms converge more quickly.
4. **Encoding Categorical variables**: Converting categorical variables into a format that can be provided to ML models, such as one-hot encoding.
5. **Data cleaning**: Removing duplicates, correcting errors, or dealing with outliers in the data.
6. **Data splitting**: Dividing data into training, validation and test sets.

The aim is to create a clean, standardized dataset that can be used directly by machine learning algorithms.

Feature Engineering:

It is the process of using domain knowledge to extract and select the most relevant features from raw data that will contribute to the predictive power of the machine learning models or improve the performance models. This step goes beyond simply cleaning the data and involves:

1. **Feature creation**: Deriving new features from existing ones that may provide additional insight or predictive power. For example: extracting a day of the week from a date field.
2. **Feature Selection**: Identifying and selecting the most important features that contribute to the target variable. This helps in reducing the dimensionality of the data and can improve model performance.
3. **Feature Transformation**: Applying transformation to features to make them more suitable for modeling. For example, taking the log of variables to handle skewed data or creating polynomial features to capture non-linear relationships.
4. **Feature Encoding**: Beyond simple encoding done in preprocessing, this might involve more sophisticated techniques tailored to the specific needs of the model or the nature of the data.
5. **Dimensionality Reduction**: Techniques like PCA (Principal Component Analysis) are used to reduce the number of features while retaining most of the information in the data.

Feature engineering is more art than science, requiring domain knowledge and experimentation to find the best representation of the data for a machine learning model.

Both data preprocessing and feature engineering are essential steps in the data preparation phase of a machine learning project and the boundary between them can sometimes blur, as both involve transforming data into a more useful form.

2. Data Collection

We download the dataset from Kaggle.

Context

Marks secured by the students

Content

This data set consists of the marks secured by the students in various subjects.

Acknowledgements

http://roycekimmons.com/tools/generated_data/exams

Inspiration

To understand the influence of the parents background, test preparation etc on students performance

3. Data Cleaning

Importing Data:

- Load the dataset using appropriate libraries (e.g., pandas).
- Inspect the initial structure, types, and summary statistics of the data to understand its composition.

Handling Missing Values:

- **Identify Missing Values**: Check for missing values using functions like `isnull()` or `isna()` in pandas.
- **Imputation**: Depending on the nature and proportion of missing values, choose an appropriate imputation method:
    - **Mean/Median/Mode Imputation**: Replace missing numerical values with the mean, median, or mode of the column.
    - **Forward/Backward Fill**: Use `ffill()` or `bfill()` to propagate next/previous values.
    - **Drop Rows/Columns**: If a significant portion of data is missing, drop rows or columns with missing values.
- **Indicator Variables**: Create new binary variables to indicate where values were imputed.

Handling Outliers:

- **Identify Outliers**: Use techniques like Z-score, IQR (Interquartile Range), or visual methods (box plots, scatter plots) to detect outliers.
- **Treat Outliers**: Depending on the context and domain knowledge:

    - **Capping/Flooring**: Replace extreme values with the nearest acceptable threshold.
    - **Transformation**: Apply transformations (e.g., log, square root) to reduce the impact of outliers.
    - **Remove**: Exclude outliers if they are deemed errors or irrelevant.

Data Consistency:

- **Correct Inconsistent Data**: Ensure that categorical data is consistent (e.g., 'USA' and 'United States' should be unified).
- **Standardize Units**: Convert all measurements to a common unit where necessary.

Duplicate Data:

- **Identify and Remove Duplicates**: Use functions like `duplicated()` and `drop_duplicates()` in pandas to identify and remove duplicate rows.

Data Transformation:

- **Normalization/Scaling**: Normalize or scale numerical features using techniques like Min-Max Scaling or Standard Scaling to bring features to a similar scale.
- **Encoding Categorical Variables**: Convert categorical variables into numerical format using techniques like One-Hot Encoding or Label Encoding.

Feature Engineering:

- **Create New Features**: Derive new features that may provide additional insights (e.g., interaction terms, polynomial features).
- **Feature Selection**: Choose the most relevant features based on statistical tests, domain knowledge, or feature importance metrics.

Data Quality Checks:

- **Validate Data**: Conduct thorough checks to ensure the cleaned data aligns with expectations and there are no remaining anomalies.

4. Exploratory Data Analysis (EDA)

Data Overview:

- The dataset consists of 1,000 rows and 8 columns, including features such as student demographics, academic scores, and lunch.
- Summary statistics reveal that the average score in mathematics is 66, with a standard deviation of 15.

Visualization:

- **Univariate Analysis**:

    - Histograms were created for numerical variables to understand their distribution. For example, the histogram of mathematics scores showed a normal distribution with a slight right skew.
    - Bar charts for categorical variables like "gender" and "lunch" provided insights into their frequency distribution.
- **Bivariate Analysis**:
    - Scatter plots were used to examine the relationship between Parental level of Education and academic scores, revealing a positive correlation.
    - A correlation heatmap indicated strong correlations between mathematics scores, reading scores, and writing scores.
- **Multivariate Analysis**:
    - Pair plots were employed to visualize the relationships among multiple numerical features, helping to identify clusters and patterns.

Key Insights:

- **Trends and Patterns**:
    - Students who participate in test preparation courses tend to have higher academic scores, especially in mathematics.
    - There is a noticeable gender gap in academic performance, with female students slightly outperforming male students.
- **Outliers and Anomalies**:
    - A few students had exceptionally low attendance rates, which correlated with lower academic performance. These outliers were considered for further investigation.
- **Correlations**:
    - Strong correlations between study habits and academic performance suggest that enhancing study techniques could lead to better outcomes.

5. Feature Engineering

Feature engineering is the process of creating new features or transforming existing ones to enhance the performance of machine learning models. Here's a detailed account of the steps taken and the rationale behind each decision:

1. **Transforming Existing Features**:
    - **Encoding Categorical Variables**: Use One-Hot Encoding for categorical variables such as `gender`, `race/ethnicity`, `parental level of education`, `lunch`, and `test preparation course`. This technique converts categorical variables into binary vectors, allowing the model to interpret and utilize these features effectively.
    - **Standardizing Numerical Scores**: Normalize `reading score` and `writing score` to bring them to a similar scale, ensuring that no single feature dominates the learning process. This can be done using techniques like Min-Max Scaling or Standard Scaling.
2. **Handling Missing Values**:

    - **Imputation**: If there are any missing values in `parental level of education` or other features, use appropriate imputation methods such as mean/mode imputation or forward/backward fill to handle them.
3. **Interaction Terms**:
    - **Interaction between Preparation Course and Scores**: Create an interaction term between `test preparation course` and `reading score` or `writing score` to capture the combined effect of test preparation on academic performance.

The rationale behind each feature engineering decision is based on the following considerations:

- **Improving Model Performance**: New features and transformations aim to provide more relevant and interpretable information to the model, enhancing its predictive accuracy.
- **Handling Data Distribution**: Transformations like standardizing scores help in normalizing distributions and capturing relationships between different features.
- **Simplifying Complex Relationships**: Interaction terms help in uncovering complex relationships between features that might not be apparent individually.
- **Standardization and Comparability**: Features like `Total Score` and normalized scores ensure that the data is standardized, making it easier to compare across observations.

6. Data Transformation

In this project, data scaling, normalization, and encoding were crucial steps to prepare the dataset for machine learning models. These steps ensure that the features are on a similar scale and the categorical variables are converted into a format that can be interpreted by the models.

Data Encoding

To handle categorical variables, One-Hot Encoding was used. This method converts categorical variables into binary (0 or 1) indicators, which allows machine learning algorithms to process these variables.

Data Scaling and Normalization

For numerical features, Standard Scaling was applied. This technique standardizes the features by removing the mean and scaling to unit variance. Standard scaling helps in improving the convergence of some machine learning algorithms.

Code Snippet for Data Scaling, Normalization, and Encoding

```
# Basic Import

import numpy as np
```

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# 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, Ridge, Lasso

from sklearn.model_selection import RandomizedSearchCV

from catboost import CatBoostRegressor

from xgboost import XGBRegressor


import warnings

warnings.filterwarnings('ignore')


# Load the dataset

df = pd.read_csv('/kaggle/input/students-performance-in-exams/StudentsPerformance.csv')


# Drop the target variable

X = df.drop(columns=['math_score'], axis=1)

y = df['math_score']
```

```
# Display unique categories for each categorical feature

print("Categories in 'gender' variable:", df['gender'].unique())

print("Categories in 'race_ethnicity' variable:", df['race_ethnicity'].unique())

print("Categories in 'parental level of education' variable:",
df['parental_level_of_education'].unique())

print("Categories in 'lunch' variable:", df['lunch'].unique())

print("Categories in 'test preparation course' variable:", df['test_preparation_course'].unique())


# Identify numerical and categorical features

num_features = X.select_dtypes(exclude="object").columns

cat_features = X.select_dtypes(include="object").columns


# Preprocessing

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer


numeric_transformer = StandardScaler()

oh_transformer = OneHotEncoder()


preprocessor = ColumnTransformer(

    transformers=[

        ("OneHotEncoder", oh_transformer, cat_features),

        ("StandardScaler", numeric_transformer, num_features),

    ]

)
```

```
# Apply the transformations

X = preprocessor.fit_transform(X)
```

Explanation:

1. **One-Hot Encoding**: This technique was used to convert categorical variables (`gender`, `race/ethnicity`, `parental level of education`, `lunch`, `test preparation course`) into binary vectors.
2. **Standard Scaling**: Numerical features were standardized using `StandardScaler` to ensure that they have a mean of 0 and a standard deviation of 1.

These steps are crucial as they transform the raw data into a format that can be effectively utilized by various machine learning algorithms, improving the model's performance and accuracy.

Model Exploration

1. Model Selection

Selecting the appropriate machine learning models is crucial for achieving high performance and reliable predictions. In this project, several models were considered to predict the `math_score` based on various student attributes. Here's the rationale behind choosing each model, along with their strengths and weaknesses:

1. Linear Regression

- **Rationale**: Linear Regression is a simple and interpretable model that establishes a linear relationship between the target variable and the features. It serves as a good baseline model for regression tasks.
- **Strengths**: Easy to understand and implement, computationally efficient, works well with linearly separable data.
- **Weaknesses**: Assumes linear relationships, sensitive to outliers, may not capture complex patterns in the data.

2. Lasso and Ridge Regression

- **Rationale**: Both Lasso and Ridge Regression add regularization to the linear regression model to prevent overfitting. Lasso performs feature selection by shrinking some coefficients to zero, while Ridge distributes the regularization across all coefficients.
- **Strengths**: Reduce overfitting, improve generalization, Lasso performs feature selection.
- **Weaknesses**: May still struggle with non-linear relationships, choice of regularization parameter can be challenging.

3. K-Neighbors Regressor

- **Rationale**: K-Neighbors Regressor is a non-parametric model that predicts the target by averaging the values of the K-nearest neighbors. It captures local patterns in the data.
- **Strengths**: Simple to understand, effective for small datasets, captures non-linear relationships.
- **Weaknesses**: Computationally expensive for large datasets, sensitive to the choice of K and distance metric.

4. Decision Tree Regressor

- **Rationale**: Decision Tree Regressor splits the data into homogeneous subsets based on feature values. It can capture complex interactions between features.
- **Strengths**: Easy to interpret, handles both numerical and categorical data, captures non-linear relationships.
- **Weaknesses**: Prone to overfitting, sensitive to small variations in the data, requires pruning for optimal performance.

5. Random Forest Regressor

- **Rationale**: Random Forest Regressor is an ensemble model that combines multiple decision trees to improve prediction accuracy and control overfitting.
- **Strengths**: Reduces overfitting, handles large datasets, captures complex interactions, robust to noise.
- **Weaknesses**: Less interpretable than single decision trees, computationally intensive, requires tuning of hyperparameters.

6. XGBoost Regressor

- **Rationale**: XGBoost is a powerful gradient boosting algorithm that builds trees sequentially to minimize prediction errors. It is highly effective for structured/tabular data.
- **Strengths**: High prediction accuracy, handles missing values, effective for large datasets, robust to overfitting.
- **Weaknesses**: Complex implementation, requires careful hyperparameter tuning, computationally intensive.

7. CatBoost Regressor

- **Rationale**: CatBoost is a gradient boosting algorithm optimized for categorical features. It automatically handles categorical variables without the need for extensive preprocessing.
- **Strengths**: Excellent handling of categorical data, high accuracy, robust to overfitting, less prone to overfitting on small datasets.
- **Weaknesses**: Computationally intensive, may require substantial memory, complex to tune.

8. AdaBoost Regressor

- **Rationale**: AdaBoost is an ensemble model that combines weak learners (e.g., decision trees) in a sequential manner to improve accuracy. It focuses on correcting errors made by previous models.

- **Strengths**: Effective at reducing bias, improves model accuracy, less prone to overfitting.
- **Weaknesses**: Sensitive to noisy data and outliers, may require careful tuning of hyperparameters, computationally expensive for large datasets.

2. Model Training

Hyperparameters Used

Each model has specific hyperparameters that influence its performance. Here are the hyperparameters used for each model based on the provided code:

- **Linear Regression**: No hyperparameters specified.
- **Lasso**: No specific hyperparameters mentioned, but typically it includes the regularization parameter `alpha`.
- **Ridge**: Similar to Lasso, typically includes the regularization parameter `alpha`.
- **K-Neighbors Regressor**: Default hyperparameters, but it typically includes `n_neighbors` (number of neighbors).
- **Decision Tree Regressor**: Default hyperparameters.
- **Random Forest Regressor**: Default hyperparameters.
- **XGBoost Regressor**: Default hyperparameters.
- **CatBoost Regressor**: `verbose=False` to suppress output during training.
- **AdaBoost Regressor**: Default hyperparameters.

Cross-Validation Techniques

Although the provided code does not explicitly show cross-validation, it is a crucial technique to evaluate model performance. Cross-validation helps in assessing how the model generalizes to unseen data and prevents overfitting.

A common approach is to use K-Fold Cross-Validation, where the dataset is split into K equal-sized folds. The model is trained K times, each time using a different fold as the validation set and the remaining K-1 folds as the training set. The performance metrics are averaged across the K runs.

3. Model Evaluation

To assess the performance of the regression models, the following evaluation metrics were used:

- **Mean Absolute Error (MAE)**: Measures the average magnitude of the errors in a set of predictions, without considering their direction. It's calculated as the average absolute difference between the actual and predicted values.
- **Mean Squared Error (MSE)**: Measures the average squared difference between the actual and predicted values. It penalizes larger errors more than smaller ones.

- **Root Mean Squared Error (RMSE)**: The square root of the MSE, providing an error metric that is on the same scale as the target variable.
- **R2 Score (Coefficient of Determination)**

4. Code Implementation

Data Preparation/Feature Engineering

```
# Basic Import

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns



# Load the dataset

df = pd.read_csv('/kaggle/input/students-performance-in-exams/StudentsPerformance.csv')



# Display unique categories for each categorical feature

print("Categories in 'gender' variable:", df['gender'].unique())

print("Categories in 'race/ethnicity' variable:", df['race_ethnicity'].unique())

print("Categories in 'parental level of education' variable:",
df['parental_level_of_education'].unique())

print("Categories in 'lunch' variable:", df['lunch'].unique())

print("Categories in 'test preparation course' variable:", df['test_preparation_course'].unique())



# Define the target variable and drop it from the feature set

y = df['math_score']

X = df.drop(columns=['math_score'], axis=1)



# Identify numerical and categorical features
```

```
num_features = X.select_dtypes(exclude="object").columns

cat_features = X.select_dtypes(include="object").columns


# Preprocessing

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer


# Create transformers for numerical and categorical features

numeric_transformer = StandardScaler()

oh_transformer = OneHotEncoder()


# Create a column transformer to apply the transformations

preprocessor = ColumnTransformer(

    transformers=[

        ("OneHotEncoder", oh_transformer, cat_features),

        ("StandardScaler", numeric_transformer, num_features),

    ]

)


# Apply the transformations to the feature set

X = preprocessor.fit_transform(X)
```

Explanation:

- **Data Import**: Import necessary libraries and load the dataset.
- **Display Categories**: Print unique categories for each categorical feature to understand the data.
- **Feature and Target Separation**: Define the target variable `math_score` and separate it from the feature set.
- **Identify Features**: Identify numerical and categorical features for processing.

- **Preprocessing**: Use `OneHotEncoder` for categorical features and `StandardScaler` for numerical features. Apply these transformations using a `ColumnTransformer`.

Model Exploration:

```
# Import necessary libraries for modeling

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.linear_model import LinearRegression, Ridge, Lasso

from catboost import CatBoostRegressor

from xgboost import XGBRegressor


# Define a function to evaluate the 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, rmse, r2_square


# Define the models to be tested

models = {

    "Linear Regression": LinearRegression(),

    "Lasso": Lasso(),

    "Ridge": Ridge(),
```

```
    "K-Neighbors Regressor": KNeighborsRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),
    "XGBRegressor": XGBRegressor(),
    "CatBoosting Regressor": CatBoostRegressor(verbose=False),
    "AdaBoost Regressor": AdaBoostRegressor()
}

# Split the data into training and testing sets
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)

# Initialize lists to store model names and R2 scores
model_list = []
r2_list = []

# Loop through each model, train it, make predictions, and evaluate
for model_name, model in models.items():
    model.fit(X_train, y_train)  # Train the model

    # Make predictions on training and testing sets
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Evaluate the model on the training and testing datasets
```

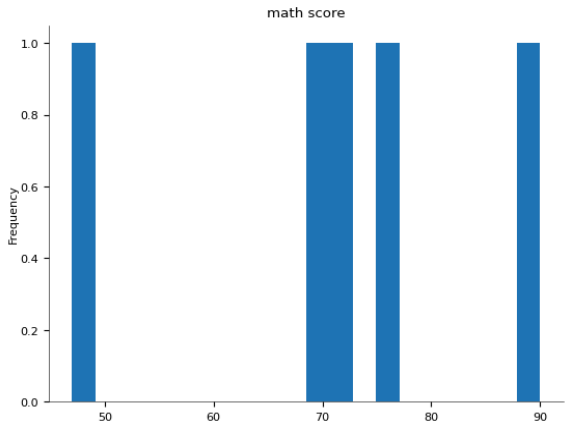
```
    model_train_mae, model_train_rmse, model_train_r2 = evaluate_model(y_train, y_train_pred)

    model_test_mae, model_test_rmse, model_test_r2 = evaluate_model(y_test, y_test_pred)


    # Print model performance

    print(model_name)

    model_list.append(model_name)


    print('Model performance for Training set')

    print("- Root Mean Squared Error: {:.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("- Mean Absolute Error: {:.4f}".format(model_test_mae))

    print("- R2 Score: {:.4f}".format(model_test_r2))

    r2_list.append(model_test_r2)


    print('='*35)

    print('\n')
```

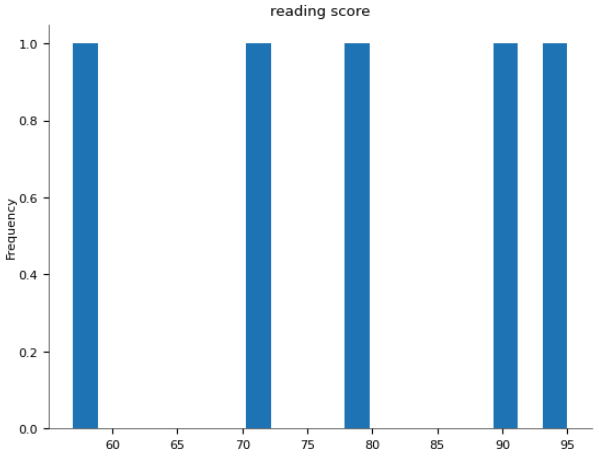
Explanation:

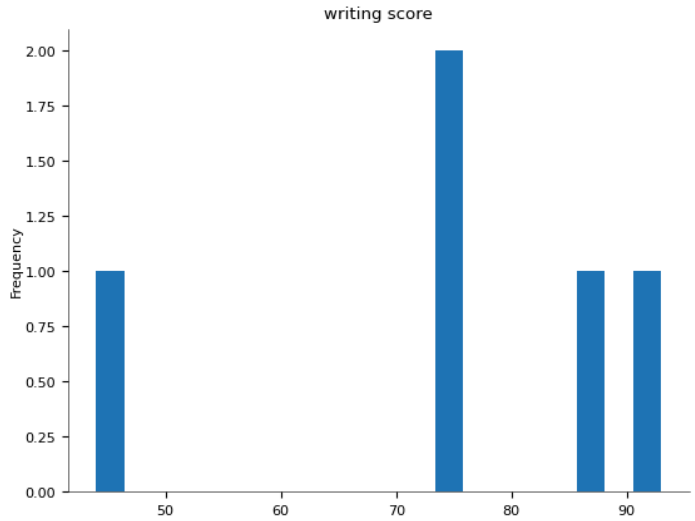
- **Import Libraries**: Import necessary libraries for modeling and evaluation.
- **Evaluation Function**: Define a function to calculate evaluation metrics (MAE, MSE, RMSE, R2 score).

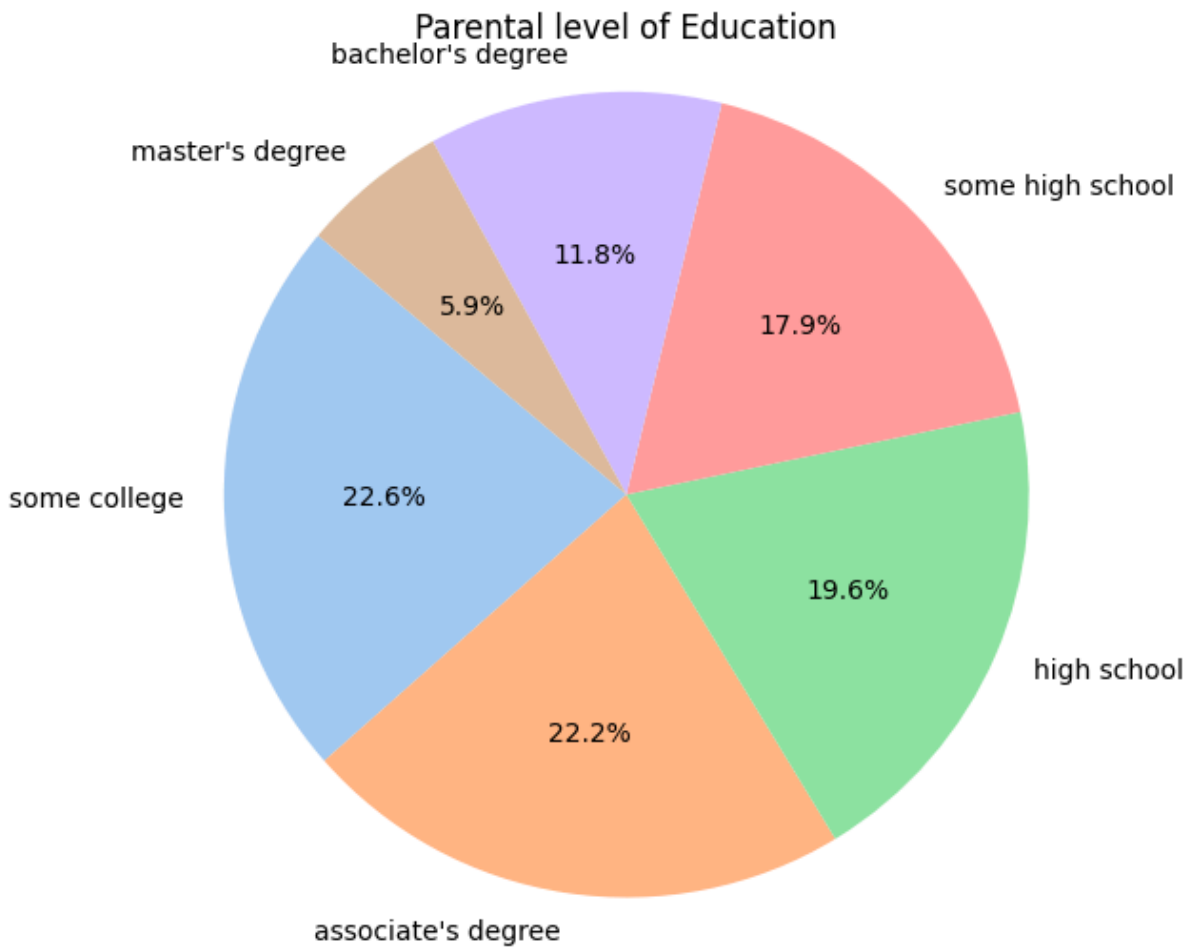
- **Model Definition**: Define a dictionary of models to be tested.
- **Data Splitting**: Split the data into training and testing sets using `train_test_split`.
- **Model Training and Evaluation**: Loop through each model, train it, make predictions on both training and testing sets, and evaluate the model's performance using the defined evaluation metrics. Print the results for each model.

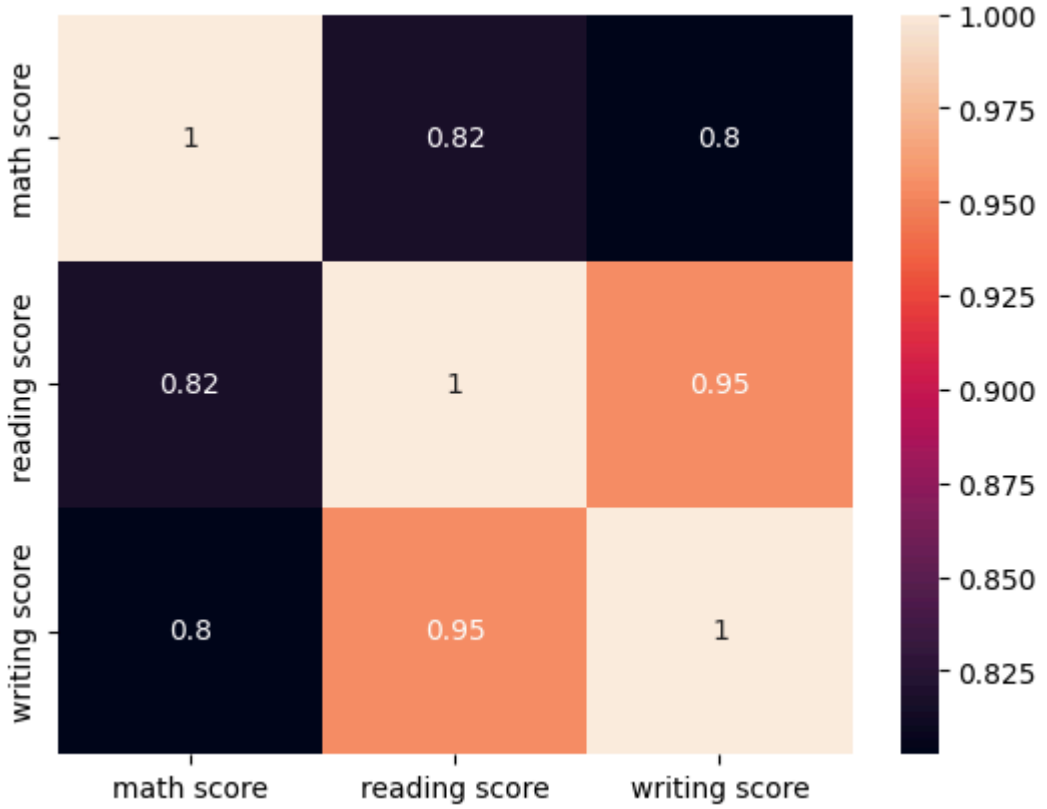
Please provide images from your data and models (I want to see different visualizations in EDA part as we did pair coding session)!!!!

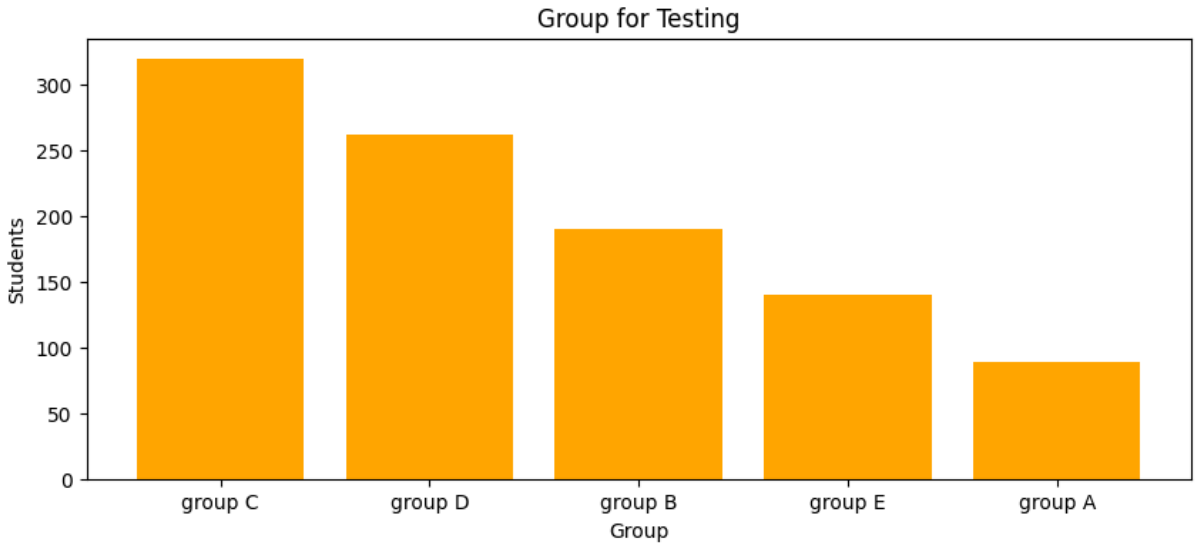






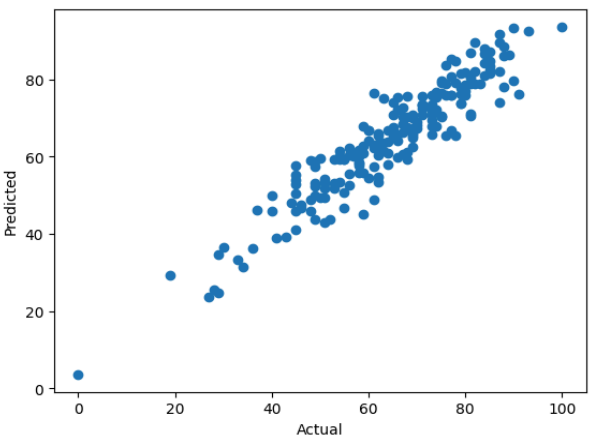

Parental level of Education
bachelor's degree
master's degree
some high school
some college
high school
associate's degree
11.8%
5.9%
17.9%
22.6%
19.6%
22.2%

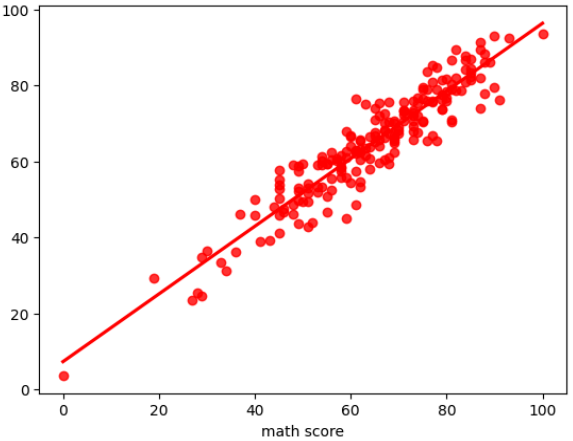

math score
reading score
writing score
1
0.82
0.8
0.82
1
0.95
0.8
0.95
1
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825


Group for Testing
Students
300
250
200
150
100
50
0
group C
group D
group B
group E
group A
Group


Relationship Between Reading Score and Math Score
Math (Scores)
Reading (Scores)
0
20
40
60
80
100


Relationship Between Reading Score and Writing Score
Writing (Scores)
Reading (Scores)
20
40
60
80
100


Predicted
Actual
0
20
40
60
80
100


math score
0
20
40
60
80
100