

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
import pandas as pd

df = pd.read_csv("/content/drive/MyDrive/Midocean - MD - Assignment/Semester 1/Practical Machine Learning and Data Exploration/Week 10/student_performance_dataset.csv")
```

```
# Check for missing values
print("Missing values per column before cleaning:")
display(df.isnull().sum())

# Handle missing values
# Fill numerical columns with mean
for col in df.select_dtypes(include=['number']).columns:
    df[col] = df[col].fillna(df[col].mean())

# Fill categorical columns with mode
for col in df.select_dtypes(include=['object']).columns:
    df[col] = df[col].fillna(df[col].mode()[0])

print("\nMissing values per column after cleaning:")
display(df.isnull().sum())
```

Missing values per column before cleaning:

	0
student_id	0
gender	0
study_time_hours	0
attendance_percent	0
sleep_hours	0
parental_education	102
internet_access	0
extracurricular_activities	0
part_time_job	0
previous_grade	0
final_exam_score	0
final_grade	0

dtype: int64

Missing values per column after cleaning:

	0
student_id	0
gender	0
study_time_hours	0
attendance_percent	0
sleep_hours	0
parental_education	0
internet_access	0
extracurricular_activities	0
part_time_job	0
previous_grade	0
final_exam_score	0
final_grade	0

dtype: int64

```
# Display the cleaned dataframe info to verify types and counts
display(df.info())
display(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   student_id                            1000 non-null   int64
1   gender                                1000 non-null   object
2   study_time_hours                       1000 non-null   float64
3   attendance_percent                     1000 non-null   float64
4   sleep_hours                           1000 non-null   float64
5   parental_education                     1000 non-null   object
6   internet_access                        1000 non-null   object
7   extracurricular_activities             1000 non-null   object
8   part_time_job                          1000 non-null   object
9   previous_grade                         1000 non-null   float64
10  final_exam_score                      1000 non-null   float64
11  final_grade                            1000 non-null   object
dtypes: float64(5), int64(1), object(6)
memory usage: 93.9+ KB
None
```

	student_id	gender	study_time_hours	attendance_percent	sleep_hours	parental_education	internet_access	extracurricular_activities	part_time_job	previous_grade	final_exam_score	final_g
0	1	Male	4.0	98.0	6.5	Bachelors	Yes	Yes	No	76.9	100.0	
1	2	Female	6.3	100.0	5.7	High School	Yes	Yes	Yes	75.5	100.0	
2	3	Male	4.9	85.3	7.9	Bachelors	Yes	No	Yes	88.5	97.3	
3	4	Male	2.6	77.5	8.0	High School	Yes	Yes	No	85.1	83.8	
4	5	Male	2.2	89.6	4.6	Bachelors	Yes	No	Yes	61.8	68.3	

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

```
# Prepare features and target
# Dropping student_id as it's an identifier, and final_grade as it's likely a direct mapping of the score
ml_df = df.copy()
le = LabelEncoder()

for col in ml_df.select_dtypes(include=['object']).columns:
    ml_df[col] = le.fit_transform(ml_df[col])

X = ml_df.drop(columns=['student_id', 'final_exam_score', 'final_grade'])
y = ml_df['final_exam_score']

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

print(f"Feature shape: {X.shape}")

Feature shape: (1000, 9)
```

```
# Define models and parameters for Grid Search
models = {
    'RandomForest': {
        'model': RandomForestRegressor(random_state=42),
        'params': {
            'n_estimators': [50, 100],
            'max_depth': [None, 10, 20]
        }
    },
    'GradientBoosting': {
        'model': GradientBoostingRegressor(random_state=42),
        'params': {
            'n_estimators': [50, 100],
            'learning_rate': [0.01, 0.1]
        }
    }
}

results = {}

for name, config in models.items():
    print(f"Running Grid Search for {name}...")
    gs = GridSearchCV(config['model'], config['params'], cv=3, scoring='neg_mean_squared_error')
    gs.fit(X_train, y_train)
    best_model = gs.best_estimator_

    y_pred = best_model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    results[name] = {
        'best_params': gs.best_params_,
        'mse': mse,
        'rmse': np.sqrt(mse),
        'r2': r2
    }

for name, metrics in results.items():
    print(f"\n--- {name} Results ---")
    print(f"Best Params: {metrics['best_params']}")
    print(f"RMSE: {metrics['rmse']:.4f}")
    print(f"R2 Score: {metrics['r2']:.4f}")
```

```
Running Grid Search for RandomForest...
Running Grid Search for GradientBoosting...

--- RandomForest Results ---
Best Params: {'max_depth': 10, 'n_estimators': 100}
RMSE: 6.8267
R2 Score: 0.5670

--- GradientBoosting Results ---
Best Params: {'learning_rate': 0.1, 'n_estimators': 100}
RMSE: 6.7962
R2 Score: 0.5709
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

# Method 1: Random Forest Feature Importance
# Using the best parameters found in the previous grid search
best_rf = RandomForestRegressor(max_depth=10, n_estimators=100, random_state=42)
best_rf.fit(X_train, y_train)

rf_importances = pd.DataFrame({
    'Feature': X.columns,
    'Importance': best_rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Method 2: Recursive Feature Elimination (RFE) with Linear Regression
lr = LinearRegression()
rfe = RFE(estimator=lr, n_features_to_select=1)
rfe.fit(X_train, y_train)

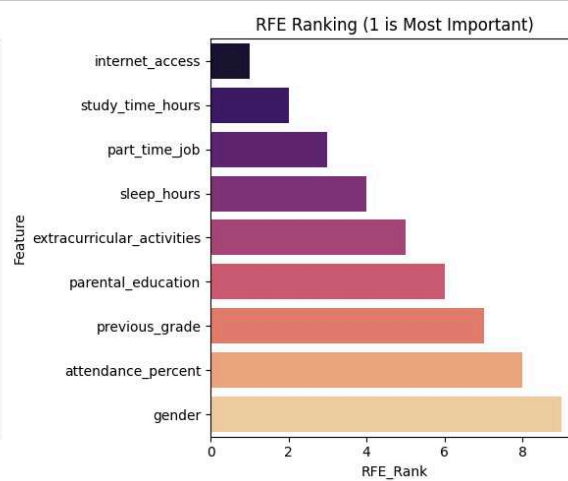
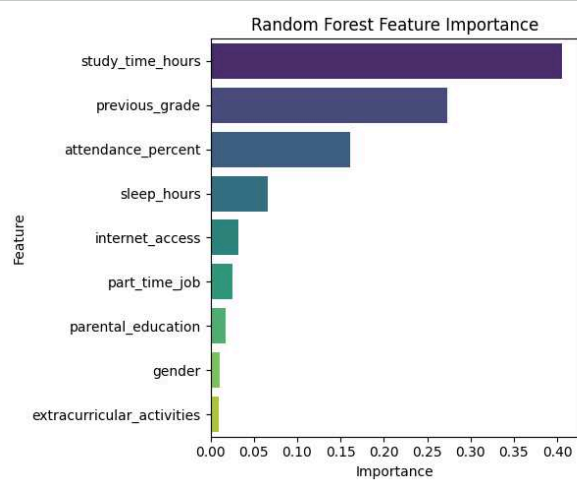
rfe_ranking = pd.DataFrame({
    'Feature': X.columns,
    'RFE_Rank': rfe.ranking_
}).sort_values(by='RFE_Rank')

# Visualize the results
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.barplot(x='Importance', y='Feature', data=rf_importances, hue='Feature', palette='viridis',
            plt.title('Random Forest Feature Importance')

plt.subplot(1, 2, 2)
sns.barplot(x='RFE_Rank', y='Feature', data=rfe_ranking, hue='Feature', palette='magma', legend
            plt.title('RFE Ranking (1 is Most Important)')

plt.tight_layout()
plt.show()
```



```
# Select top 5 features based on RF importance
top_features = rf_importances['Feature'].head(5).tolist()
print(f"Selected Features: {top_features}")

# Subset the data
X_train_selected = X_train[top_features]
X_test_selected = X_test[top_features]

# Initialize and train the Gradient Boosting Regressor (best performing model from previous gri
final_model = GradientBoostingRegressor(learning_rate=0.1, n_estimators=100, random_state=42)
final_model.fit(X_train_selected, y_train)

# Predict and Evaluate
y_pred_selected = final_model.predict(X_test_selected)
mse_selected = mean_squared_error(y_test, y_pred_selected)
rmse_selected = np.sqrt(mse_selected)
r2_selected = r2_score(y_test, y_pred_selected)

print(f"\n--- Model Performance with Selected Features ---")
print(f"RMSE: {rmse_selected:.4f}")
print(f"R2 Score: {r2_selected:.4f}")
```

Selected Features: ['study_time_hours', 'previous_grade', 'attendance_percent', 'sleep_hours', 'internet_access']

--- Model Performance with Selected Features ---
 RMSE: 6.7355
 R2 Score: 0.5785

```
# Visualize Actual vs Predicted scores
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_selected, alpha=0.5, color='teal')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Actual Final Exam Score')
plt.ylabel('Predicted Final Exam Score')
plt.title('Actual vs Predicted Student Performance (Selected Features)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

