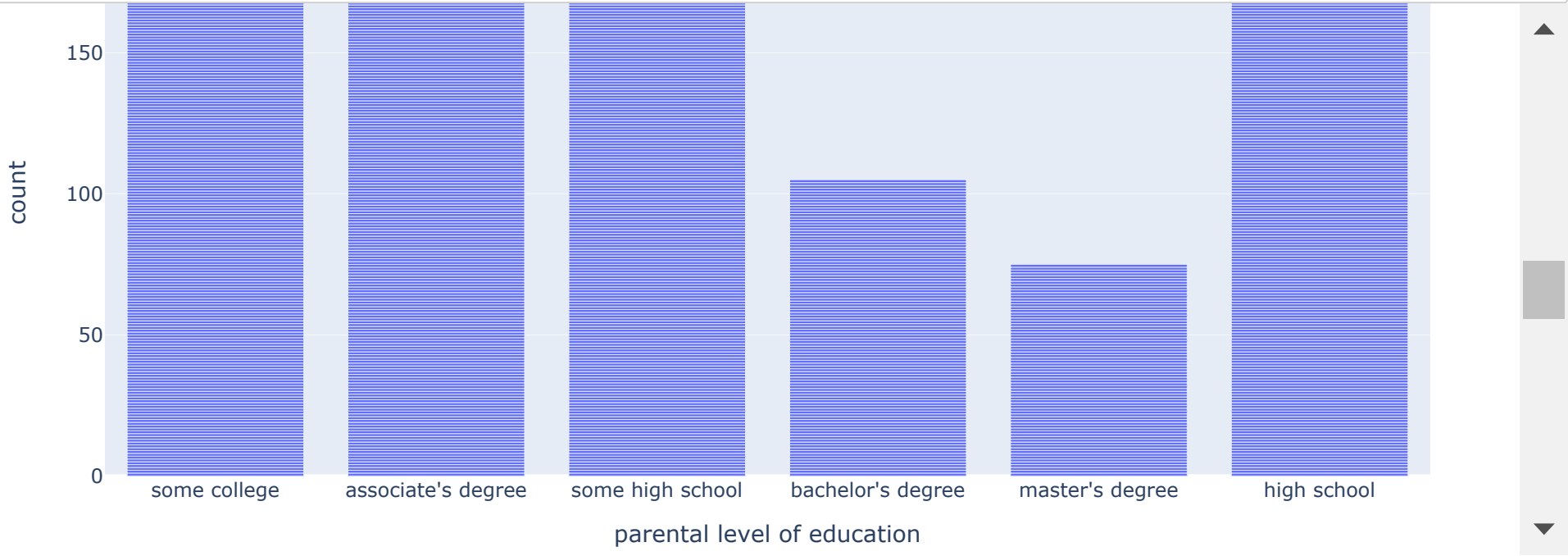
In [1]:
```
import pandas as pd
import numpy as np
import plotly.graph_objs as go
import plotly.express as px
```

In [2]:
```
data=pd.read_csv("performance.csv")
data.describe()
print(data)
```

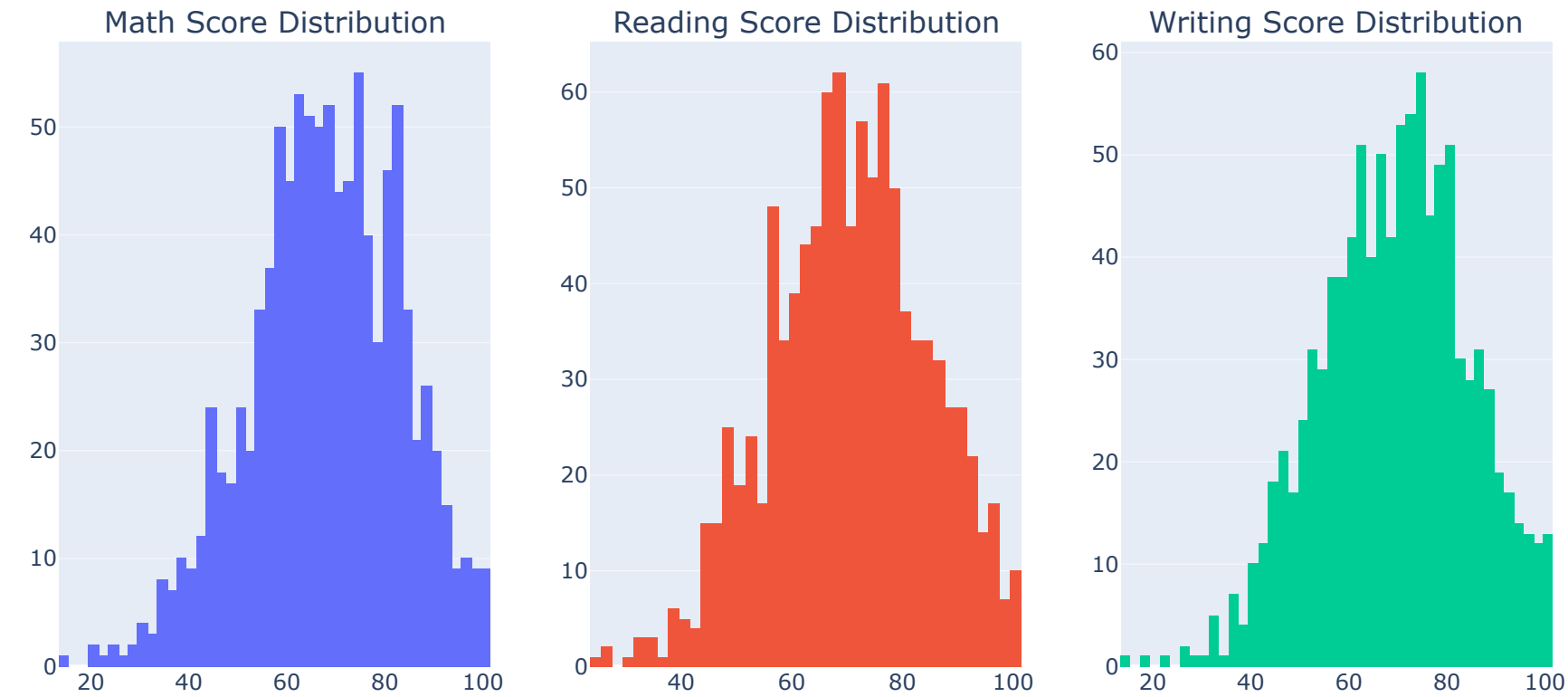
```
     gender race/ethnicity parental level of education         lunch  \
0    female        group D                some college      standard
1      male        group D          associate's degree      standard
2    female        group D                some college  free/reduced
3      male        group B                some college  free/reduced
4    female        group D          associate's degree      standard
..      ...            ...                         ...           ...
995    male        group C                some college      standard
996    male        group C                some college      standard
997  female        group A                 high school      standard
998    male        group E                 high school      standard
999    male        group D                 high school      standard

    test preparation course  math score  reading score  writing score
0                 completed          59             70             78
1                      none          96             93             87
2                      none          57             76             77
3                      none          70             70             63
4                      none          83             85             86
..                      ...         ...            ...            ...
995                    none          77             77             71
996                    none          80             66             66
997               completed          67             86             86
998                    none          80             72             62
999                    none          58             47             45

[1000 rows x 8 columns]
```

In [3]:
```
# Gender Distribution
fig = px.bar(data, x='gender', title='Gender Distribution')
fig.show()
# Race/Ethnicity Distribution
fig = px.pie(data, names='race/ethnicity', title='Race/Ethnicity Distribution')
fig.show()
# Parental Level of Education Distribution
fig = px.bar(data, x='parental level of education', title='Parental Level of Education Distribution')
fig.show()
# Lunch Type Distribution
fig = px.pie(data, names='lunch', title='Lunch Type Distribution')
fig.show()
# Test Preparation Course Completion
fig = px.bar(data, x='test preparation course', title='Test Preparation Course Completion')
fig.show()
```

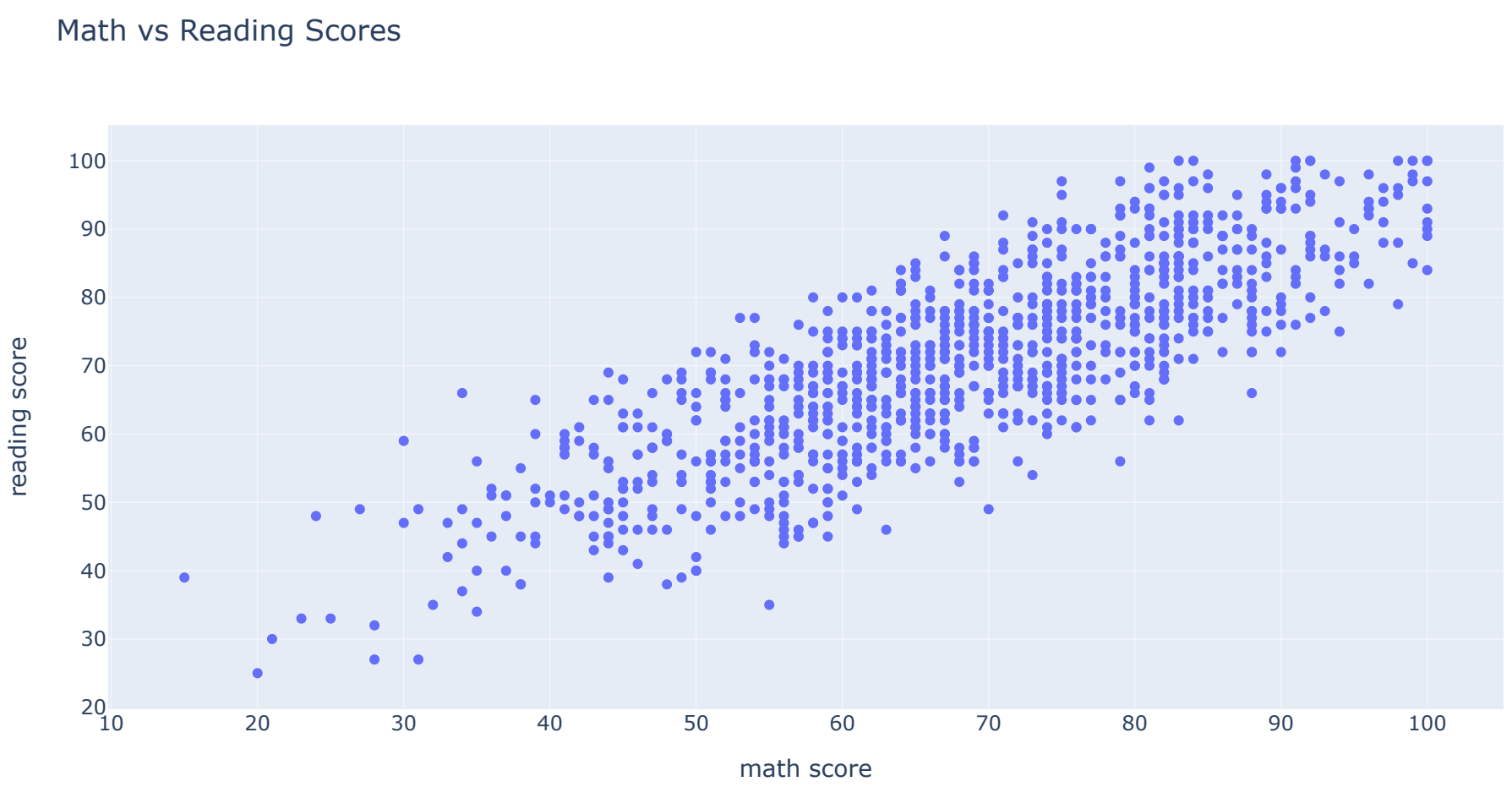


In [4]:

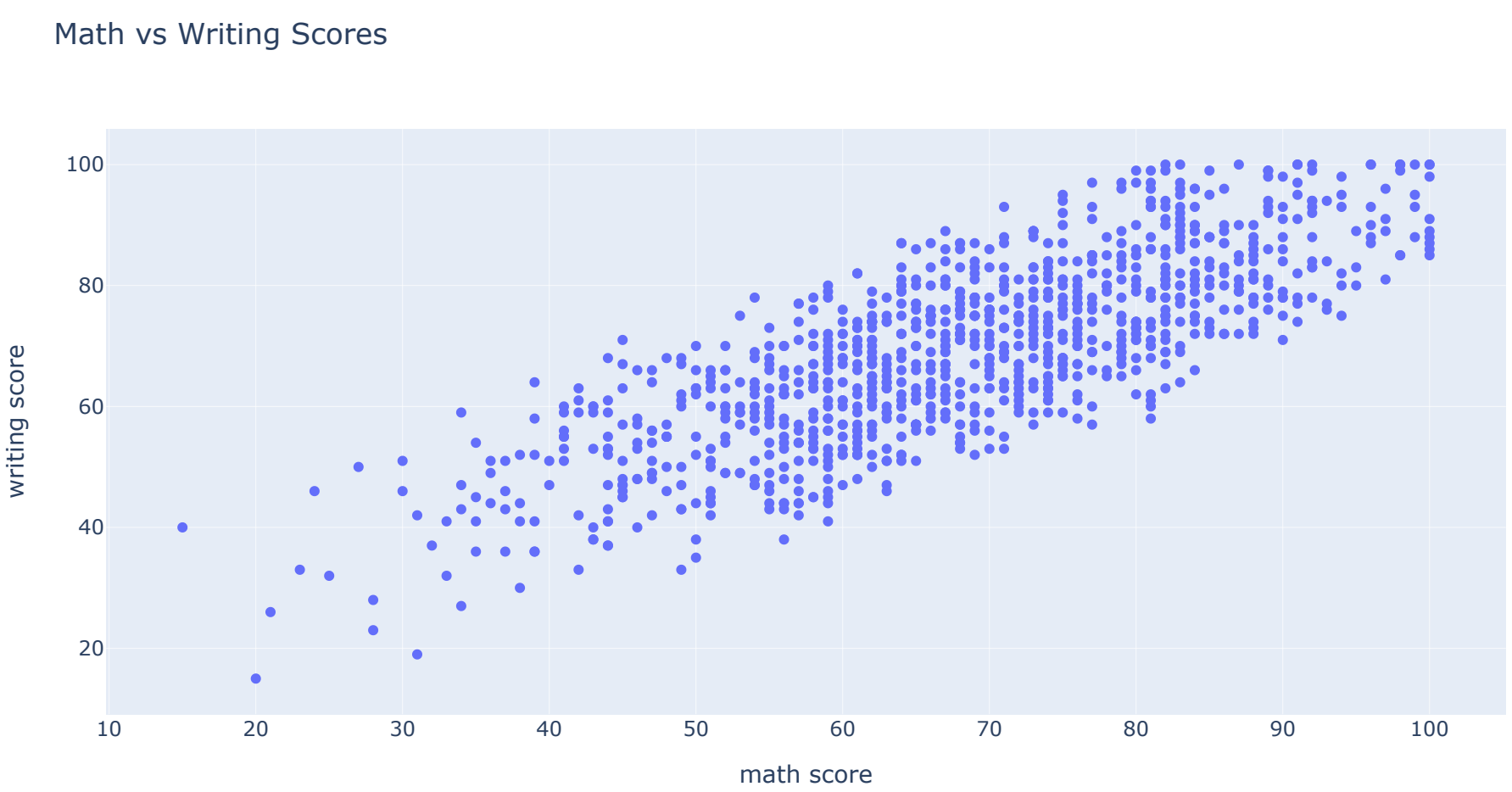
```
#for creating subplots importing libraries from plotly
from plotly.subplots import make_subplots
# Create subplots
fig = make_subplots(rows=1, cols=3, subplot_titles=('Math Score Distribution', 'Reading Score Distribution', 'Writing Score Dis
# Math Score Distribution among students
math_fig = go.Histogram(x=data['math score'], name='Math Score')
fig.add_trace(math_fig, row=1, col=1)

# Reading Score Distribution among students
reading_fig = go.Histogram(x=data['reading score'], name='Reading Score')
fig.add_trace(reading_fig, row=1, col=2)

# Writing Score Distribution among students
writing_fig = go.Histogram(x=data['writing score'], name='Writing Score')
fig.add_trace(writing_fig, row=1, col=3)

# Final layout update
fig.update_layout(title_text='Distributions of Math, Reading, and Writing Scores', showlegend=False)

fig.show()
```


Distributions of Math, Reading, and Writing Scores
Math Score Distribution
Reading Score Distribution
Writing Score Distribution
0
10
20
30
40
50
60
20
40
60
80
100

In [5]:
```
#For Finding correlation among
# Correlation between Scores
fig = px.scatter(data, x='math score', y='reading score', title='Math vs Reading Scores')
fig.show()

fig = px.scatter(data, x='reading score', y='writing score', title='Reading vs Writing Scores')
fig.show()

fig = px.scatter(data, x='math score', y='writing score', title='Math vs Writing Scores')
fig.show()
```




Reading vs Writing Scores
writing score
reading score
100
80
60
40
20
30
40
50
60
70
80
90
100


Math vs Writing Scores
writing score
math score
100
80
60
40
20
10
20
30
40
50
60
70
80
90
100

In [11]:
```
#Building Algorithm
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
# Define the parameter grid
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 1.0],
    'min_samples_split': [2, 5, 10]
}
# Initialize the Gradient Boosting Regressor
gbr_math = GradientBoostingRegressor(random_state=42)
gbr_reading = GradientBoostingRegressor(random_state=42)
gbr_writing = GradientBoostingRegressor(random_state=42)

# Initialize the Grid Search for each target
grid_search_math = GridSearchCV(estimator=gbr_math, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_reading = GridSearchCV(estimator=gbr_reading, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs
grid_search_writing = GridSearchCV(estimator=gbr_writing, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs

# Fit the Grid Search for each target
grid_search_math.fit(X_train, y_train_math)
grid_search_reading.fit(X_train, y_train_reading)
grid_search_writing.fit(X_train, y_train_writing)

# Get the best parameters for each target
best_params_math = grid_search_math.best_params_
best_params_reading = grid_search_reading.best_params_
best_params_writing = grid_search_writing.best_params_
print(f"Best parameters for Math Score: {best_params_math}")
print(f"Best parameters for Reading Score: {best_params_reading}")
print(f"Best parameters for Writing Score: {best_params_writing}")

# Use the best estimators
best_gbr_math = grid_search_math.best_estimator_
best_gbr_reading = grid_search_reading.best_estimator_
best_gbr_writing = grid_search_writing.best_estimator_
```

```
Best parameters for Math Score: {'learning_rate': 0.01, 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 300, 'subsam
ple': 0.8}
Best parameters for Reading Score: {'learning_rate': 0.01, 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 300, 'sub
sample': 0.8}
Best parameters for Writing Score: {'learning_rate': 0.01, 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 300, 'sub
sample': 0.8}
```

In [12]:

```python
#Model Evaluation
# Make predictions with the best models
y_pred_math_best = best_gbr_math.predict(X_test)
y_pred_reading_best = best_gbr_reading.predict(X_test)
y_pred_writing_best = best_gbr_writing.predict(X_test)

# Evaluate the best models
mse_math_best = mean_squared_error(y_test_math, y_pred_math_best)
r2_math_best = r2_score(y_test_math, y_pred_math_best)

mse_reading_best = mean_squared_error(y_test_reading, y_pred_reading_best)
r2_reading_best = r2_score(y_test_reading, y_pred_reading_best)

mse_writing_best = mean_squared_error(y_test_writing, y_pred_writing_best)
r2_writing_best = r2_score(y_test_writing, y_pred_writing_best)

print(f"Best model MSE for Math Score: {mse_math_best}, R²: {r2_math_best}")
print(f"Best model MSE for Reading Score: {mse_reading_best}, R²: {r2_reading_best}")
print(f"Best model MSE for Writing Score: {mse_writing_best}, R²: {r2_writing_best}")
```

```
Best model MSE for Math Score: 153.23532902122923, R²: 0.30588593052927027
Best model MSE for Reading Score: 142.0780844407197, R²: 0.16196859557231935
Best model MSE for Writing Score: 143.54775912655154, R²: 0.24623104848481658
```

In [13]:
```python
#Comparing different models
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

# Initialize models
models = {
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(n_estimators=100, random_state=42),
    'Ridge Regression': Ridge(),
    'SVR': SVR()
}

# Train and evaluate models
for name, model in models.items():
    model.fit(X_train, y_train_math)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test_math, y_pred)
    r2 = r2_score(y_test_math, y_pred)
    print(f"{name} - MSE: {mse}, R2: {r2}")
```

```
Random Forest - MSE: 168.52362461867443, R2: 0.23663413892092666
Gradient Boosting - MSE: 153.16736808747044, R2: 0.3061937749447693
Ridge Regression - MSE: 156.24797363543118, R2: 0.2922394755871265
SVR - MSE: 165.39681220266255, R2: 0.2507977427348287
```

In [ ]: