```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
file_path = '/content/drive/MyDrive/student_performance_dataset.csv'
# Update this path accordingly
df = pd.read_csv(file_path)

# Define independent variables (features) and dependent variable
(target)
X = df[['Study Hours (per week)', 'Attendance (%)']]
y = df['Grade (out of 100)']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```python
# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 0.8013501096511958
R-squared: 0.9810331334993799
```

```python
# Coefficients
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [1.73852858 0.60041015]
Intercept: 12.592924891053563
```

```python
# Predicting the grade of a new student with specific features
new_student = pd.DataFrame({'Study Hours (per week)': [10],
'Attendance (%)': [85]})
predicted_grade = model.predict(new_student)
print("Predicted Grade for new student:", predicted_grade[0])
```

```
Predicted Grade for new student: 81.01307357087927
```

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

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
df =
pd.read_csv('/content/drive/MyDrive/student_performance_dataset.csv')

# Define independent variables (features) and dependent variable
(target)
X = df[['Study Hours (per week)', 'Attendance (%)']]
y = df['Grade (out of 100)']

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize k-NN regressor (set k=3 for example)
k = 3
knn_regressor = KNeighborsRegressor(n_neighbors=k)

# Train the model
knn_regressor.fit(X_train_scaled, y_train)
```

```
KNeighborsRegressor(n_neighbors=3)
```

```
# Predict on the test set
y_pred = knn_regressor.predict(X_test_scaled)

# Evaluate performance (e.g., using RMSE and R^2)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R^2 Score: {r2}')
```

```
Root Mean Squared Error (RMSE): 1.5811388300841898
R^2 Score: 0.9408284023668639
```

```
# Example: Predict grade for a new student
new_student = [[9, 85]]  # Study Hours = 9, Attendance % = 85
new_student_scaled = scaler.transform(new_student)
```

```
predicted_grade = knn_regressor.predict(new_student_scaled)

print(f'Predicted Grade: {predicted_grade[0]}')
```

```
Predicted Grade: 78.66666666666667
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but StandardScaler
was fitted with feature names
  warnings.warn(
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score

# Load dataset
df =
pd.read_csv('/content/drive/MyDrive/student_performance_dataset.csv')

# Display the first few rows to understand the structure
print(df.head())
```

```
   Study Hours (per week)  Attendance (%)  Grade (out of 100)
0                       5              80                  70
1                      10              90                  85
2                       3              60                  50
3                       8              75                  78
4                      15              95                  92
```

```
# Define independent variables (features) and dependent variable
(target)
X = df[['Study Hours (per week)', 'Attendance (%)']]
y = df['Grade (out of 100)']

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize logistic regression model
log_reg = LogisticRegression(random_state=42)

# Train the model
log_reg.fit(X_train_scaled, y_train)
```

```
LogisticRegression(random_state=42)
```

```python
# Predict on the test set
y_pred = log_reg.predict(X_test_scaled)

# Evaluate performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{class_report}')
```

```
Accuracy: 0.0
Confusion Matrix:
[[0 0 0 0]
 [1 0 0 0]
 [0 0 0 1]
 [0 0 0 0]]
Classification Report:
              precision    recall  f1-score   support

          70       0.00      0.00      0.00       0.0
          72       0.00      0.00      0.00       1.0
          85       0.00      0.00      0.00       1.0
          95       0.00      0.00      0.00       0.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Example: Predict for a new student with Study Hours = 9 and
Attendance % = 85
new_student = [[9, 85]]
new_student_scaled = scaler.transform(new_student)
predicted_grade = log_reg.predict(new_student_scaled)

print(f'Predicted Grade: {predicted_grade[0]}')
```

```
Predicted Grade: 88
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but StandardScaler
was fitted with feature names
  warnings.warn(
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load dataset
df =
pd.read_csv('/content/drive/MyDrive/student_performance_dataset.csv')

# Display the first few rows to understand the structure
print(df.head())
```

```
   Study Hours (per week)  Attendance (%)  Grade (out of 100)
0                       5              80                  70
1                      10              90                  85
2                       3              60                  50
3                       8              75                  78
4                      15              95                  92
```

```
# Define independent variables (features) and dependent variable
(target)
X = df[['Study Hours (per week)', 'Attendance (%)']]
y = df['Grade (out of 100)']
```

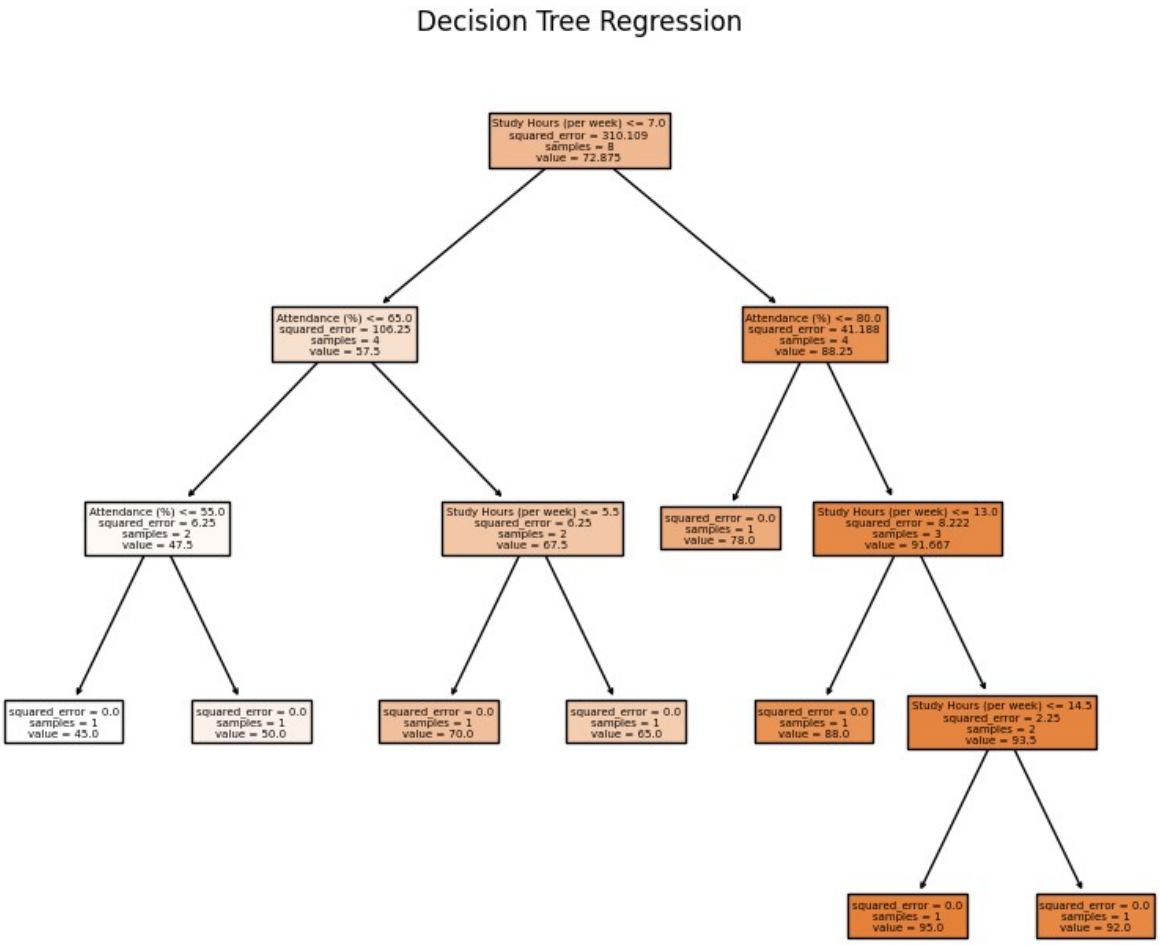
```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize decision tree regressor
dt_regressor = DecisionTreeRegressor(random_state=42)

# Train the model
dt_regressor.fit(X_train, y_train)
```

```
DecisionTreeRegressor(random_state=42)
```

```
# Predict on the test set
y_pred = dt_regressor.predict(X_test)

# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared (R2): {r2}')
```

```
Mean Squared Error (MSE): 29.0
R-squared (R2): 0.31360946745562135
```

```
from sklearn.tree import plot_tree

plt.figure(figsize=(10, 8))
plot_tree(dt_regressor, feature_names=X.columns, filled=True)
plt.title("Decision Tree Regression")
plt.show()
```




```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load dataset
df =
pd.read_csv('/content/drive/MyDrive/student_performance_dataset.csv')

# Display the first few rows to understand the structure
print(df.head())
```

```
   Study Hours (per week)  Attendance (%)  Grade (out of 100)
0                       5              80                  70
1                      10              90                  85
2                       3              60                  50
3                       8              75                  78
4                      15              95                  92
```

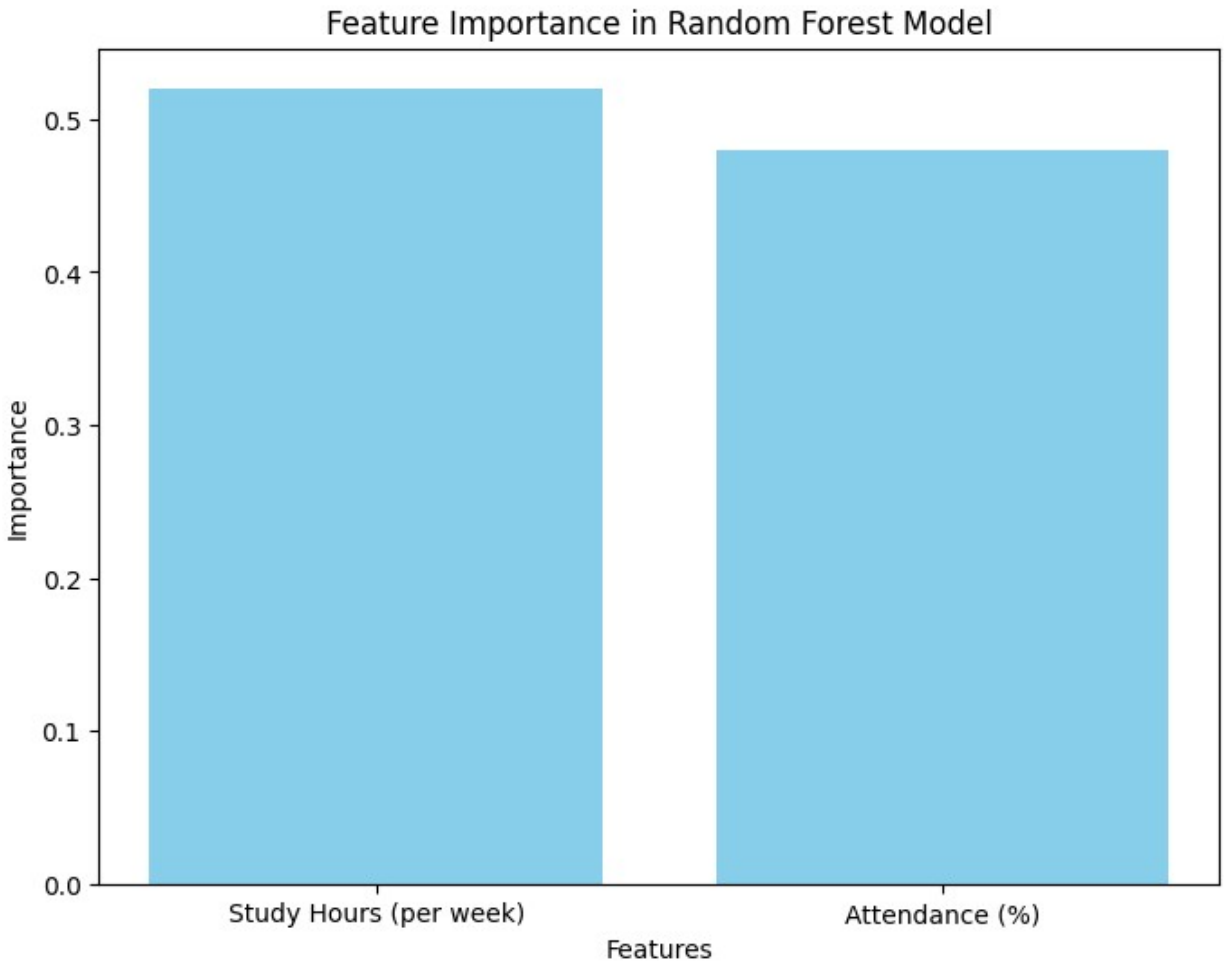
```
# Define independent variables (features) and dependent variable
(target)
X = df[['Study Hours (per week)', 'Attendance (%)']]
y = df['Grade (out of 100)']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize Random Forest regressor
rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)

# Train the model
rf_regressor.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=42)
```

```
# Predict on the test set
y_pred = rf_regressor.predict(X_test)

# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'R-squared (R2): {r2}')
```

```
Mean Squared Error (MSE): 1.1382499999999933
R-squared (R2): 0.9730591715976333
```

```
# Feature importance
feature_importances = rf_regressor.feature_importances_
features = X.columns

# Visualize feature importance
plt.figure(figsize=(8, 6))
plt.bar(features, feature_importances, color='skyblue')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance in Random Forest Model')
plt.show()
```


Feature Importance in Random Forest Model
Importance
0.0
0.1
0.2
0.3
0.4
0.5
Study Hours (per week)
Attendance (%)
Features