```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Reading the CSV file
student_per = pd.read_csv("/content/student-mat.csv",delimiter=";")
student_per
```

{"type":"dataframe","variable_name":"student_per"}

```python
student_per.shape #This dataset has 395 samples and 33 features.
```

```
(395, 33)
```

```python
student_per.info() #This gives the information about the columns of dataset, Using this we find all the non-numerical columns in dataset
#The given dataset doesn't have any null values.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      395 non-null    object
 1   sex         395 non-null    object
 2   age         395 non-null    int64
 3   address     395 non-null    object
 4   famsize     395 non-null    object
 5   Pstatus     395 non-null    object
 6   Medu        395 non-null    int64
 7   Fedu        395 non-null    int64
 8   Mjob        395 non-null    object
 9   Fjob        395 non-null    object
 10  reason      395 non-null    object
 11  guardian    395 non-null    object
 12  traveltime  395 non-null    int64
 13  studytime   395 non-null    int64
 14  failures    395 non-null    int64
 15  schoolsup   395 non-null    object
 16  famsup      395 non-null    object
 17  paid        395 non-null    object
 18  activities  395 non-null    object
 19  nursery     395 non-null    object
 20  higher      395 non-null    object
 21  internet    395 non-null    object
 22  romantic    395 non-null    object
 23  famrel      395 non-null    int64
 24  freetime    395 non-null    int64
 25  goout       395 non-null    int64
 26  Dalc        395 non-null    int64
 27  Walc        395 non-null    int64
```

```
 28  health      395 non-null    int64
 29  absences    395 non-null    int64
 30  G1          395 non-null    int64
 31  G2          395 non-null    int64
 32  G3          395 non-null    int64
dtypes: int64(16), object(17)
memory usage: 102.0+ KB
```

```
student_per.describe()
```


```
{"summary":"{\n  \"name\": \"student_per\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
134.436252896189,\n        \"min\": 1.2760427246056245,\n
\"max\": 395.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          16.696202531645568,\n          17.0,\n
395.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Medu\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 138.80963938157987,\n        \"min\": 0.0,\n        \"max\":
395.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\n
395.0,\n          2.749367088607595,\n          3.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Fedu\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 138.92085462409693,\n
\"min\": 0.0,\n        \"max\": 395.0,\n        \"num_unique_values\":
7,\n        \"samples\": [\n          395.0,\n
2.5215189873417723,\n          3.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"traveltime\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
139.0946757987501,\n        \"min\": 0.6975047549086848,\n
\"max\": 395.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          395.0,\n          1.4481012658227848,\n
4.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"studytime\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 139.00700274471274,\n        \"min\":
0.839240346418556,\n        \"max\": 395.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          395.0,\n
2.0354430379746837,\n          4.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"failures\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
139.4513615014189,\n        \"min\": 0.0,\n        \"max\": 395.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.3341772151898734,\n          3.0,\n          0.743650973606249\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"famrel\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
```

138.45880901426744,\n        \"min\": 0.8966586076885056,\n
\"max\": 395.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          395.0,\n          3.9443037974683546,\n
5.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"freetime\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 138.63828826062982,\n        \"min\":
0.9988620396657205,\n        \"max\": 395.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          395.0,\n
3.2354430379746835,\n          4.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"goout\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 138.68948196584594,\
n        \"min\": 1.0,\n        \"max\": 395.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
3.108860759493671,\n          3.0,\n          395.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Dalc\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 139.0354623650101,\n
\"min\": 0.8907414280909659,\n        \"max\": 395.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          395.0,\n
1.481012658227848,\n          5.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Walc\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 138.87302263653973,\n
\"min\": 1.0,\n        \"max\": 395.0,\n        \"num_unique_values\":
7,\n        \"samples\": [\n          395.0,\n
2.2911392405063293,\n          3.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"health\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
138.50262599778412,\n        \"min\": 1.0,\n        \"max\": 395.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          395.0,\n
3.5544303797468353,\n          4.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"absences\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
136.85777166785417,\n        \"min\": 0.0,\n        \"max\": 395.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          395.0,\n
5.708860759493671,\n          8.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"G1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 136.30663508587594,\n
\"min\": 3.0,\n        \"max\": 395.0,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          10.90886075949367,\n
11.0,\n          395.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"G2\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 136.4163745266465,\n        \"min\": 0.0,\n        \"max\":

395.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
10.713924050632912,\n          11.0,\n          395.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"G3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 136.35024783099098,\n
\"min\": 0.0,\n        \"max\": 395.0,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          10.415189873417722,\n
11.0,\n          395.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

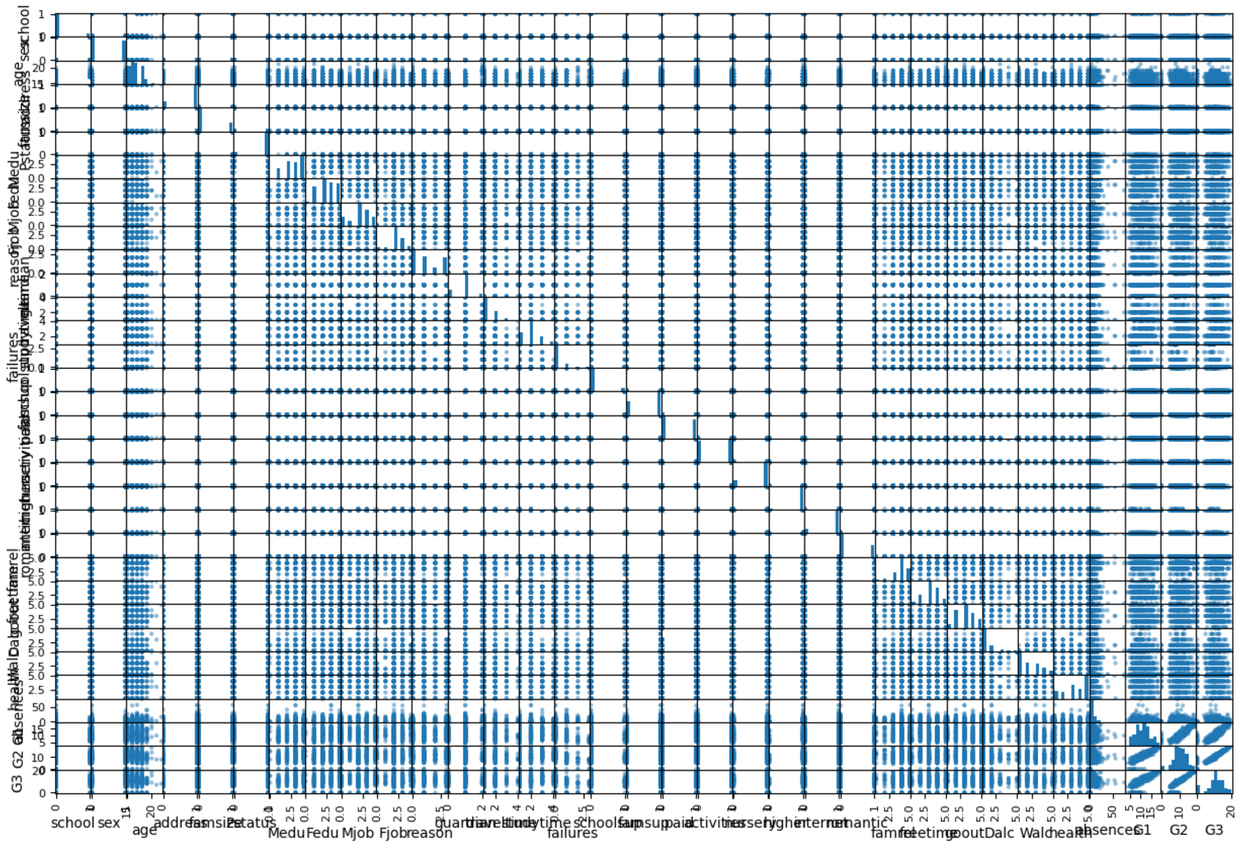
```
# Converting all the non-numerical columns in dataset to numerical
columns using Label Encoder
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
encode_colm = student_per.iloc[:,
[0,1,3,4,5,8,9,10,11,15,16,17,18,19,20,21,22]]
for i in encode_colm:
    student_per[i] = label.fit_transform(student_per[i])
student_per
```

{"type":"dataframe","variable_name":"student_per"}

```
# Correlation Matrix
correlation_Matrix = student_per.corr()
correlation_Matrix
```

{"type":"dataframe","variable_name":"correlation_Matrix"}

```
# Scatter Matrix
pd.plotting.scatter_matrix(student_per, figsize=(15, 10))
plt.show()
```



```
# Storing the input attributes and the target variable
X = student_per.iloc[:,:32] # Input attributes
y = student_per.iloc[:,32] # Target variable(G3)
X
```

{"type":"dataframe","variable_name":"X"}

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.20,random_state=42)

from sklearn.preprocessing import StandardScaler
stand_scale = StandardScaler()
stand_scale.fit(X_train)
X_train_std = stand_scale.transform(X_train)
X_test_std = stand_scale.transform(X_test)

from sklearn.linear_model import LinearRegression
LR_model = LinearRegression()
LR_model.fit(X_train_std,y_train)
LR_score = LR_model.score(X_test_std,y_test) # Finding the R2 score of
the model
print("R2 score:",LR_score)
```

```
R2 score: 0.7545777855043497
```

#Hyper parameter tuning

#1. SVM with RBF kernel.

```
from sklearn.svm import SVR
rbf_model = SVR(kernel='rbf')

rbf_model.get_params()

{'C': 1.0,
 'cache_size': 200,
 'coef0': 0.0,
 'degree': 3,
 'epsilon': 0.1,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

Tuning the hyperparameters C and gamma

```
hyp_grid_rbf = [{
    'C': [0.01, 0.1, 1, 10, 100,100],
    'gamma': [0.001, 0.009,1]
}]
hyp_grid_rbf

[{'C': [0.01, 0.1, 1, 10, 100, 100], 'gamma': [0.001, 0.009, 1]}]
```

Performing the Grid Search

```
from sklearn.model_selection import GridSearchCV
grid_search_rbf =
GridSearchCV(estimator=rbf_model,param_grid=hyp_grid_rbf,cv=5)
grid_search_rbf.fit(X_train_std,y_train)
print("Best Parameters:",grid_search_rbf.best_params_)
print("Best Score:",grid_search_rbf.best_score_)
print("Best Estimator:",grid_search_rbf.best_estimator_)

Best Parameters: {'C': 100, 'gamma': 0.001}
Best Score: 0.8230913992113609
Best Estimator: SVR(C=100, gamma=0.001)
```

R2 score of the model after the hyperparamter tuning

```
from sklearn.svm import SVR
rbf_model_hyp = SVR(kernel='rbf',C=100,gamma=0.001)
rbf_model_hyp.fit(X_train_std,y_train)
```

```
rbf_score = rbf_model_hyp.score(X_test_std,y_test)
print("R2 Score:",rbf_score)
```

```
R2 Score: 0.7851580302181262
```

#2. Random Forest

```
from sklearn.ensemble import RandomForestRegressor
RF_model = RandomForestRegressor(random_state=42)

RF_model.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

We are tuning the hyperparameters( n_estimators,maximum_depth,maximum_leaf_nodes,maximum_features,minimum_samples_split and minimum_samples_leaf)

```
hyp_grid_rf = {'n_estimators':[i for i in range(10,101,10)],
               'max_depth':[j for j in range(2,21,2)],
               'max_leaf_nodes':[k for k in range(20,41,2)],
               'max_features':[l for l in range(2,33,2)],
               'min_samples_split':[m for m in range(2,21,2)],
               'min_samples_leaf':[n for n in range(1,21,2)]}
hyp_grid_rf
```

```
{'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
 'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
 'max_leaf_nodes': [20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40],
 'max_features': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,
30, 32],
```

```
 'min_samples_split': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
 'min_samples_leaf': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]}
```

Performing Random Search

```
from sklearn.model_selection import RandomizedSearchCV
random_search_rf =
RandomizedSearchCV(estimator=RF_model,param_distributions=hyp_grid_rf,
cv=5)
random_search_rf.fit(X_train_std,y_train)
print("Best Parameters:",random_search_rf.best_params_)
print("Best Score:",random_search_rf.best_score_)
print("Best Estimator:",random_search_rf.best_estimator_)

Best Parameters: {'n_estimators': 10, 'min_samples_split': 6,
'min_samples_leaf': 11, 'max_leaf_nodes': 36, 'max_features': 32,
'max_depth': 12}
Best Score: 0.8589145158962216
Best Estimator: RandomForestRegressor(max_depth=12, max_features=32,
max_leaf_nodes=36,
                      min_samples_leaf=11, min_samples_split=6,
n_estimators=10,
                      random_state=42)
```

R2 Score of the model after hyperparameter tuning

```
from sklearn.ensemble import RandomForestRegressor
RF_model_hyp =
RandomForestRegressor(n_estimators=60,max_depth=18,max_leaf_nodes=22,m
ax_features=24,min_samples_split=2,min_samples_leaf=5,n_jobs=-
1,random_state=42)
RF_model_hyp.fit(X_train_std,y_train)
RF_score = RF_model_hyp.score(X_test_std,y_test)
print("R2 Score:",RF_score)

R2 Score: 0.8315030232671886
```

#3. Adaboost using decision tree stumps

```
from sklearn.ensemble import AdaBoostRegressor
ADA_model = AdaBoostRegressor(random_state=42)

ADA_model.get_params()

{'estimator': None,
 'learning_rate': 1.0,
 'loss': 'linear',
 'n_estimators': 50,
 'random_state': 42}
```

We are tuning the hyperparameters n_estimators and learning rate

```
hyp_grid_ada = [{'n_estimators':[i for i in range(10,201,20)],
                 'learning_rate':np.linspace(0.15,0.5,8)}]
hyp_grid_ada
```

```
[{'n_estimators': [10, 30, 50, 70, 90, 110, 130, 150, 170, 190],
  'learning_rate': array([0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45,
0.5 ])}]
```

Performing the Grid Search

```
from sklearn.model_selection import GridSearchCV
grid_search_ada =
GridSearchCV(estimator=ADA_model,param_grid=hyp_grid_ada,cv=5)
grid_search_ada.fit(X_train_std,y_train)
print("The Best Parameters:",grid_search_ada.best_params_)
print("The Best Score:",grid_search_ada.best_score_)
print("The Best Estimator:",grid_search_ada.best_estimator_)
```

```
The Best Parameters: {'learning_rate':
np.float64(0.44999999999999996), 'n_estimators': 70}
The Best Score: 0.8968550900297902
The Best Estimator:
AdaBoostRegressor(learning_rate=np.float64(0.44999999999999996),
                  n_estimators=70, random_state=42)
```

R2 Score of the model after hyperparamter tuning

```
from sklearn.ensemble import AdaBoostRegressor
ADA_model_hyp =
AdaBoostRegressor(n_estimators=70,learning_rate=0.25,random_state=42)
ADA_model_hyp.fit(X_train_std,y_train)
ADA_score = ADA_model_hyp.score(X_test_std,y_test)
print("R2 Score:",ADA_score)
```

```
R2 Score: 0.8325293553061772
```

#4. Gradient Boosting

```
from sklearn.ensemble import GradientBoostingRegressor
GBR_model = GradientBoostingRegressor(random_state=42)

GBR_model.get_params()
```

```
{'alpha': 0.9,
 'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
```

```
 'loss': 'squared_error',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': 42,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

We are tuning the hyperparameters n_estimators,max_depth and learning_rate

```
hyp_grid_gbr = {'n_estimators':[i for i in range(10,101,10)],
                'learning_rate':np.linspace(0.1,0.5,5),
                'max_depth':[j for j in range(2,21,2)]}
hyp_grid_gbr
```

```
{'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
 'learning_rate': array([0.1, 0.2, 0.3, 0.4, 0.5]),
 'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]}
```

Performing Random Search

```
from sklearn.model_selection import RandomizedSearchCV
random_search_gbr =
RandomizedSearchCV(estimator=GBR_model,param_distributions=hyp_grid_gb
r,cv=5)
random_search_gbr.fit(X_train_std,y_train)
print("Best Parameters:",random_search_gbr.best_params_)
print("Best Score:",random_search_gbr.best_score_)
print("Best Estimator:",random_search_gbr.best_estimator_)
```

```
Best Parameters: {'n_estimators': 10, 'max_depth': 4, 'learning_rate':
np.float64(0.30000000000000004)}
Best Score: 0.8867884616886164
Best Estimator:
GradientBoostingRegressor(learning_rate=np.float64(0.30000000000000004
),
                          max_depth=4, n_estimators=10,
random_state=42)
```

R2 Score of the model after hyperparamter tuning

```
from sklearn.ensemble import GradientBoostingRegressor
GBR_model_hyp =
GradientBoostingRegressor(n_estimators=20,learning_rate=0.3,max_depth=
4,random_state=42)
GBR_model_hyp.fit(X_train_std,y_train)
GBR_score = GBR_model_hyp.score(X_test_std,y_test)
print("R2 Score:",GBR_score)
```

```
R2 Score: 0.811762031831491
```

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.get_params()
```

```
{'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'random_state': 42,
 'splitter': 'best'}
```

We are tuning the hyperparameters max_depth,max_features,max_leaf_nodes,min_samples_split and min_samples_leaf

```
hyp_grid_tree = {'max_depth':[i for i in range(2,21,2)],
                 'max_features':[j for j in range(2,33,2)],
                 'max_leaf_nodes':[k for k in range(20,41,2)],
                 'min_samples_split':[l for l in range(2,21,2)],
                 'min_samples_leaf':[m for m in range(1,21,2)]}
hyp_grid_tree
```

```
{'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
 'max_features': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,
30, 32],
 'max_leaf_nodes': [20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40],
 'min_samples_split': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
 'min_samples_leaf': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]}
```

Performing Random Search

```
from sklearn.model_selection import RandomizedSearchCV
random_search_tree =
RandomizedSearchCV(estimator=tree_model,param_distributions=hyp_grid_t
ree,cv=5)
random_search_tree.fit(X_train_std,y_train)
print("Best Parameters:",random_search_tree.best_params_)
print("Best Score:",random_search_tree.best_score_)
print("Best Estimator:",random_search_tree.best_estimator_)
```

```
Best Parameters: {'min_samples_split': 4, 'min_samples_leaf': 7,
'max_leaf_nodes': 22, 'max_features': 24, 'max_depth': 4}
Best Score: 0.7806317573935685
Best Estimator: DecisionTreeRegressor(max_depth=4, max_features=24,
max_leaf_nodes=22,
                      min_samples_leaf=7, min_samples_split=4,
random_state=42)
```

R2 score of the model after hyperparameter tuning

```
from sklearn.tree import DecisionTreeRegressor
tree_model_hyp =
DecisionTreeRegressor(max_depth=8,max_features=28,max_leaf_nodes=20,mi
n_samples_split=8,min_samples_leaf=7,random_state=42)
tree_model_hyp.fit(X_train_std,y_train)
tree_score = tree_model_hyp.score(X_test_std,y_test)
print("R2 Score:",tree_score)
```

```
R2 Score: 0.8493239751834084
```

```
import matplotlib.pyplot as plt

# Example model names — adjust if you know the actual ones
model_names = [
    "Linear Regression", "Random Search SVM RBF", "Random Search
Random Forest", "Random Search Ada Boost",
    "Random Search Gradient Boost", "Random Search Decision Tree"
]
r2_scores = [ LR_score, rbf_score, RF_score, ADA_score, GBR_score,
tree_score ]


# Create the bar chart
plt.figure(figsize=(12, 6))
bars = plt.bar(model_names, r2_scores, color='cornflowerblue')

# Add R² values on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01,
f'{yval:.2f}', ha='center', va='bottom')
```

```python
# Labeling
plt.title("Model Performance (R² Score)")
plt.xlabel("Model Type")
plt.ylabel("R² Score")
plt.ylim(0, 1.1)
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```