Boston House Price Prediction

Challenge Inside! : Find out where you stand! Try quiz, solve problems & win rewards!
Go to Challenge

Overview

As the name suggests, Boston House Price Prediction Project predicts house prices based on various parameters like area, number of rooms, age, and more. In addition, the Boston House Price Prediction dataset has numeric values making it uncomplicated to compute and predict.

Moreover, this project can be developed for real-time applications in any city, helping developers, realtors, and buyers predict property value.

What are we Building?

In the Boston House Price Prediction Project, we are building a predictive model to evaluate the price of a house with provided parameters.

The parameters into consideration are - Per Capita Crime Rate by Town, Nitric Oxide Concentration, Weighted Distances to five Boston Employment Centres, Tax, etc.

In this project, we will use - Linear Regression, Random Forest, and decision tree classifier and compare the model accuracy.

Pre-requisites

To build the Boston House Price Prediction Project, the following pre-requisites will be a benefit -

- Understanding of Python
- Implementation of libraries like Pandas, Numpy, Seaborn, Matplotlib, and SciKit Learn
- Familiarity with Machine Learning algorithms like Regression, Decision Trees, and Random Forests
- Basic acquaintance with concepts like Data Cleaning, Processing, Model Evaluation, Model Testing, Model Training, and Feature detection

How are we going to build this?

Let us have a walkthrough of building the project -

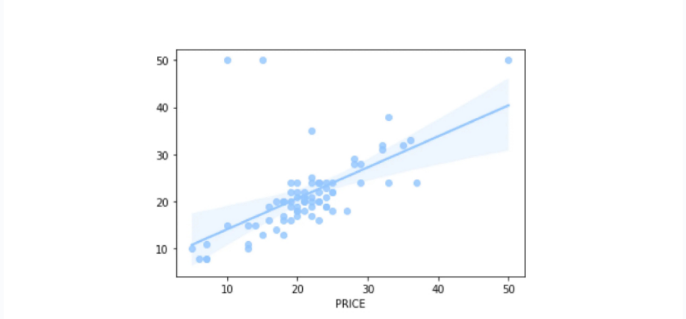
1. **Libraries and Data** - Importing the necessary libraries and dataset
2. **Analysis** - Reviewing the values, data types, and columns of the dataset
3. **Visualisation** - Data Visualisation to check for underlying trends and patterns in the dataset
4. **Feature Analysis** - Analysing the features to check if refactoring, dropping of columns, or missing data is to be filled
5. **Data Cleaning** - Filling in missing values and dropping columns that are not needed
6. **Feature Engineering** - Scoring the features to assess their significance in predicting
7. **Training, Testing, and Splitting data** - Getting the data prepared for prediction by dividing it into training and testing data
8. **Models** - Fitting the models and then predicting outputs
9. **Evaluation** - Obtaining the model score and checking their accuracy

Final Output

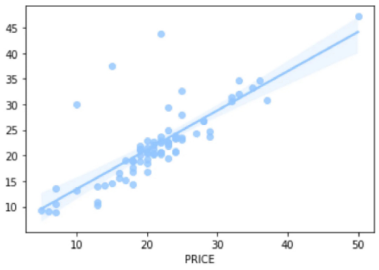
Given below is the score of each of the models applied to the dataset.

Linear Regression Graph -

Decision Tree Graph -



Random Forest Graph -



Requirements

- Environment - Either an online or offline compiler
- Libraries - Pandas, Numpy, Seaborn, Matplotlib, and SciKit Learn

About the data

The Boston Housing Data has 506 rows and 14 columns. All the values are either integers or float.

Below are the columns and their description -

- **CRIM** - Per Capita Crime Rate
- **ZN** - Proportion of Residential Land Zoned for Lots over 25,000 sq. ft
- **INDUS** - Proportion of Non-Retail Business Acres
- **CHAS** - Charles River Variable
- **NOX** - Nitric Oxide Concentration
- **RM** - Average number of Rooms
- **AGE** - Proportion of owner-occupied units built before 1940
- **DIS** - Weighted Distances to Boston Employment Centres
- **RAD** - Index of Accessibility to Radial Highways
- **TAX** - Property Tax Rate per $10,000
- **PTRATIO** - Student — Teacher Ration
- **B** - Proportion of people of African American descent
- **LSTAT** - Percentage of Lower Status of the Population
- **MEDV** - Median value of owner-occupied homes in $1000, i.e., Price of the property

Understanding the Problem Statement

Here, the goal is to build a model that can predict the price of a property based on parameters like crime rate, age of the property, air quality, etc.

Algorithms

We have used three algorithms -

1. **Logistic Regression** An algorithm for classifying the data is logistic regression. It is applied to predict a binary outcome based on several independent factors.

2. **Decision Tree Classifier** A supervised learning technique named a decision tree can be used to tackle classification and regression problems and is usually preferred. It is a tree-structured classifier, where internal nodes stand in for a dataset's features, branches for decision-making, and each leaf node for the classification result.

3. **Random Forest** A "forest" is created by growing and combining various decision trees using the supervised machine learning method Random Forest. The Random Forest model is based on the idea that several uncorrelated models (the various decision trees) work significantly better together than they do separately.

Building the Project

1. **Load the data**

```
# Importing Libraries

# Utility Libraries
import numpy as np
import pandas as pd

# Visualisation Libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Data Processing Libraries
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn import model_selection
from sklearn.metrics import mean_squared_error, r2_score

# Algorithm Libraries
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

# Math Library
import math

# Importing Dataset
df = pd.read_csv("/kaggle/input/boston-housing-dataset/Housi

# Printing Dataset
df.head()
```

Output-

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

2. **Data Analysis**

```
# Data Analysis

# Checking for columns and their respective datatypes
df.info()
```

Output-

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
```

```
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

```
# Getting the number of rows and columns
df.shape
```

Output-

```
(506, 14)
```

```
# Calculating the mean, minimum, deviation, maximum and c e
df.describe()
```

Output-

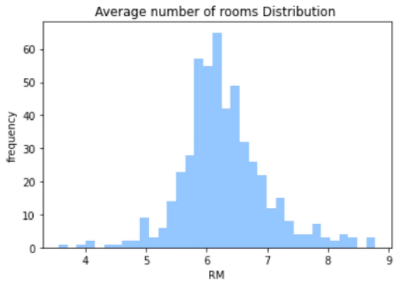
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.000000	506.000000	506.000000	506.000000	506.000000	486.000000	506.000000
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.549407	408.237154	18.455534	356.674032	12.715432	22.532806
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.105710	8.707259	168.537116	2.164946	91.294864	7.155871	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.100175	4.000000	279.000000	17.400000	375.377500	7.125000	17.025000
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.207450	5.000000	330.000000	19.050000	391.440000	11.430000	21.200000
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

SCALER Topics

3. Data Visualisation

```
fig=plt.figure()
ax=fig.add_subplot(1, 1, 1)
ax.hist(df['RM'], bins = 35)
plt.title("Average number of rooms Distribution ")
plt.xlabel("RM")
plt.ylabel("frequency")
plt.show()
```

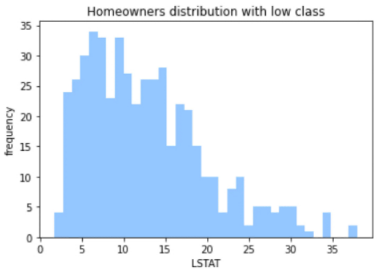
Output-



SCALER Topics

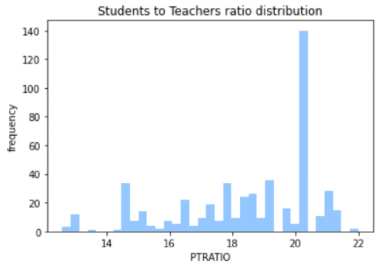
```
fig=plt.figure()
ax=fig.add_subplot(1, 1, 1)
ax.hist(df['LSTAT'], bins = 35)
plt.title("Homeowners distribution with low class")
plt.xlabel("LSTAT")
plt.ylabel("frequency")
plt.show()
```

Output-



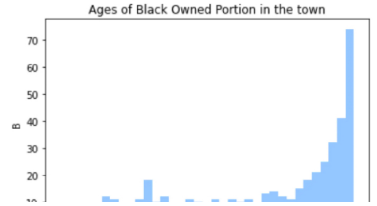

```
fig=plt.figure()
ax=fig.add_subplot(1, 1, 1)
ax.hist(df['PTRATIO'], bins = 35)
plt.title("Students to Teachers ratio distribution")
plt.xlabel("PTRATIO")
plt.ylabel("frequency")
plt.show()
```

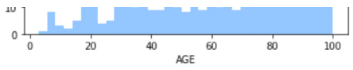
Output-




```
fig=plt.figure()
ax=fig.add_subplot(1, 1, 1)
ax.hist(df['AGE'], bins = 35)
plt.title("Ages of Black Owned Portion in the town")
plt.xlabel("AGE")
plt.ylabel("B")
plt.show()
```

Output-



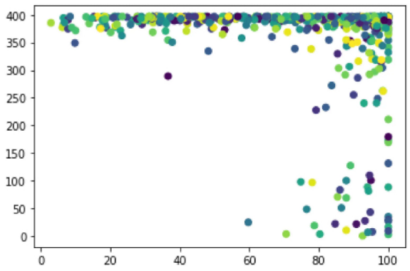


SCALER Topics

```
N = 506
x = df.AGE
y = df.B
colors = np.random.rand(N)

plt.scatter(x, y, c=colors)
plt.show()
```

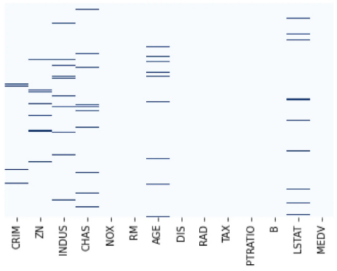
Output-



SCALER Topics

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='B
```

Output-



SCALER Topics

4. Data Cleaning

```
df = df.fillna(df.mean())
df.isnull().sum()
```

Output-

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
```

```
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

```
df.rename(columns={'MEDV':'PRICE'}, inplace = True)
```

5. Feature Engineering

```
corr = df.corr()
corr.shape
```

Output-

```
(14, 14)
```

```
df.shape
```

Output-

```
(506, 14)
```

```
X = df.iloc[:,0:13] #independent columns
y = df.iloc[:,-1] #target column i.e price range

y = np.round(df['PRICE'])
#Apply SelectKBest class to extract top 5 best features
bestfeatures = SelectKBest(score_func=chi2)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
# Concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['SPECS','SCORE'] #naming the datafr
featureScores
```

Output-

	SPECS	SCORE
0	CRIM	5332.588364
1	ZN	5817.411645
2	INDUS	826.590604
3	CHAS	59.037323
4	NOX	5.073299
5	RM	21.981504
6	AGE	2201.715129
7	DIS	163.919426
8	RAD	1445.257647
9	TAX	14817.836927
10	PTRATIO	45.692587
11	B	3340.486412
12	LSTAT	1329.940743

SCALER Topics

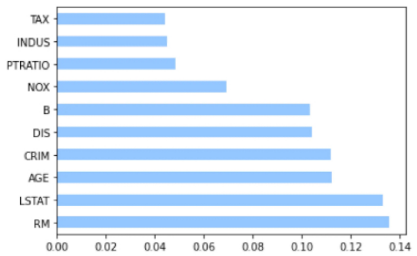
```
print(featureScores.nlargest(8,'SCORE')) #print 5 best fe u
```

Output-

```
    SPECS         SCORE
9     TAX  14817.836927
1      ZN   5817.411645
0    CRIM   5332.588364
11      B   3340.486412
6     AGE   2201.715129
8     RAD   1445.257647
12  LSTAT   1329.940743
2   INDUS    826.590604
```

```
# Plot graph of feature importances for better visualizat n
feat_importances = pd.Series(model.feature_importances_, ind
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

Output-



6. Train the model

```
model = ExtraTreesClassifier()
model.fit(X,y)
```

Output-

```
ExtraTreesClassifier()
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,t t

# a benchmark regressor that takes mean of training sample a
class BenchmarkRegressor:
    def __init__(self):
        pass

    def fit(self, X, y, **kwargs):
        self.mean = y.mean()

    def predict(self, X):
        return [self.mean] * len(X)

    def get_params(self, deep=False):
        return {}

bm_regr = BenchmarkRegressor()
lr_regr = LinearRegression()
dt_regr = DecisionTreeRegressor()
rf_regr = RandomForestRegressor()
```

```
# create a list of models and evaluate each model
models = [
    ('Benchmark', bm_regr),
    ('LR', lr_regr),
    ('Decision Tree', dt_regr),
    ('Random Forest', rf_regr)
]
```

7. Model evaluation

```
print("Root Mean Square Error (RMSE) score\n")
scoring = 'neg_mean_squared_error'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_tr
```

```
    sqrt_cv_results = [math.sqrt(abs(i)) for i in cv_results
    print("{}: {} ({})".format(name, np.mean(sqrt_cv_results
    print('Result from each iteration of cross validation:',
```

Output-

```
Root Mean Square Error (RMSE) score

Benchmark: 9.410735133688892 (1.0020278822611464)
Result from each iteration of cross validation: [ -91.483150
  -76.17510967  -81.32270363  -76.26022742  -88.35262304  -6

LR: 4.918337861592024 (0.9662284549855387)
Result from each iteration of cross validation: [-31.6876348
 -13.23158325 -23.73434699 -34.52602986 -22.30431381 -14.394

Decision Tree: 4.9694082912079285 (0.7488625967173612)
Result from each iteration of cross validation: [-31.8372093
 -18.90697674 -25.69767442 -27.69767442 -29.27906977 -29.372

Random Forest: 3.4589709161843167 (0.9235225616741916)
Result from each iteration of cross validation: [-11.7340744
  -5.23625814 -15.9700186   -9.98836047 -12.20895116  -7.879
```

```
print("R-squared Value\n")
scoring = 'r2'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_tr
    print("{}: {} ({})".format(name, cv_results.mean(), cv_r
    print('Result from each iteration of cross validation:',
```

Output-

```
R-squared Value

Benchmark: -0.00787063015356233 (0.008119222980965124)
Result from each iteration of cross validation: [-9.02138916
 -2.42765734e-02 -1.35121089e-02 -2.97277890e-03 -1.97223021
 -2.45452983e-03 -9.92116641e-05]

LR: 0.7188916120955359 (0.08833269965437104)
Result from each iteration of cross validation: [0.65049847
 0.7072785  0.53833127 0.74693382 0.76234357]

Decision Tree: 0.6487255353183778 (0.15226230545185646)
Result from each iteration of cross validation: [0.59677881
 0.46307364 0.63181417 0.7218895  0.54540422]

Random Forest: 0.864490356518882 (0.059004521023005234)
Result from each iteration of cross validation: [0.88562118
 0.80097256 0.86926611 0.87325317 0.86537445]
```

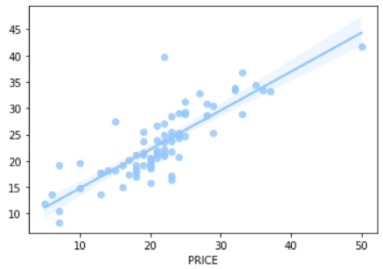
7. Model Testing

```
model1 = lr_regr
model1.fit(X_train, y_train)
y_pred = model1.predict(X_test)

rmse_score = np.sqrt(mean_squared_error(y_test, y_pred))
rsquared_score = r2_score(y_test, y_pred)
print('RMSE score:', rmse_score)
print('R2 score:', rsquared_score)
sns.regplot(y_test, y_pred);
```

Output-

```
RMSE score: 4.531197238787371
R2 score: 0.6308232714816586
```



SCALER Topics

```
print("Training Accuracy:",model1.score(X_train,y_train)*0
print("Testing Accuracy:",model1.score(X_test,y_test)*100)
print("Model Accuracy:",r2_score(y,model1.predict(X))*100)
```

Output-

```
Training Accuracy: 73.44737531372365
Testing Accuracy: 63.08232714816586
Model Accuracy: 72.51231699897534
```

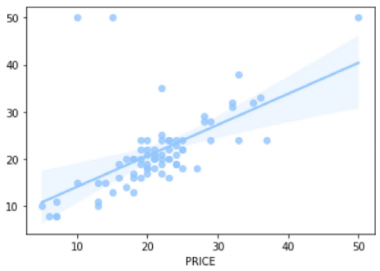
```
model2 = dt_regr
model2.fit(X_train, y_train)
y_pred = model2.predict(X_test)

rmse_score = np.sqrt(mean_squared_error(y_test, y_pred))
rsquared_score = r2_score(y_test, y_pred)
print('RMSE score:', rmse_score)
print('R2 score:', rsquared_score)

sns.regplot(y_test, y_pred);
```

Output-

```
RMSE score: 7.164422333197636
R2 score: 0.07706579668277136
```



SCALER Topics

```
print("Training Accuracy:",model2.score(X_train,y_train)*0
print("Testing Accuracy:",model2.score(X_test,y_test)*100)
print("Model Accuracy:",r2_score(y,model2.predict(X))*100)
```

Output-

```
Training Accuracy: 100.0
Testing Accuracy: 7.706579668277136
Model Accuracy: 90.89195367761111
```

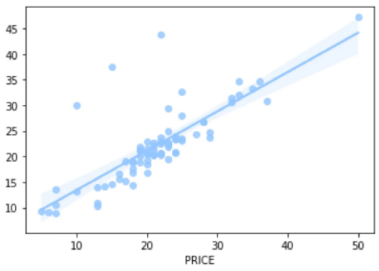
```
model3 = rf_regr
model3.fit(X_train, y_train)
y_pred = model3.predict(X_test)

rmse_score = np.sqrt(mean_squared_error(y_test, y_pred))
rsquared_score = r2_score(y_test, y_pred)
print('RMSE score:', rmse_score)
print('R2 score:', rsquared_score)

sns.regplot(y_test, y_pred);
```

Output-

```
RMSE score: 4.912976774774504
R2 score: 0.5659919298201923
```



SCALER Topics

```
print("Training Accuracy:",model3.score(X_train,y_train)*0
print("Testing Accuracy:",model3.score(X_test,y_test)*100)
print("Model Accuracy:",r2_score(y,model3.predict(X))*100)
```

Output-

```
Training Accuracy: 98.07347176907356
Testing Accuracy: 56.59919298201923
Model Accuracy: 93.98692315266736
```

What's next

To enhance this project further -

- **Web Implementation** - Flask can be used to have an interface
- **Models** - Here, we have used three models, other models can be tried out to improve accuracy further

Conclusion

- Boston House Price prediction is the best regarding working on numeric data.
- In this project, the best accuracy was that of Random Forest at 93%.

Challenge Time!

Time to test your skills and win rewards!

Start Challenge

Note: Rewards will be credited after the next product update.

This article is written by

B BINAY KUMAR GUPTA

Updated - 4 May 2023 • 9 mins read • Published : 18 Jan 2023

Scroll to top!

Machine Learning

How would you rate this article?

Prev

The Titanic Dataset Project

Next

Customer Churn Prediction Project in Machine Learning

Article Outline

Log in to check your progress

Free Courses by top Scaler instructors

Java Course - Mastering the...

A Course by **Tarun Luthra**

Embark on your programming journey with our comprehensive Free Java...

5 91989

Python Course for Beginner...

A Course by **Rahul Janghu**

Welcome to the free Python course online for beginners, designed to hel...

4.90 85252

JavaScript Course With...

A Course by **Mrinal Bhattacharya**

Kickstart your journey into web development with this free JavaScri...

4.8 46208

C++ Course: Learn the...

A Course by **Prateek Narang**

Gain programming expertise with our Free C++ Course! Covering basics to...

5 45834

View All

Got suggestions? We would love to hear your feedback

Got suggestions? We would love to hear your feedback.

SCALER Topics | A Free learning platform made with ♥ by SCALER

Explore Scaler

- Academy
- Data Science & ML
- Neovarsity

Explore Topics

- Courses
- Challenges
- Contest
- Topics
- Articles
- Events

Resources

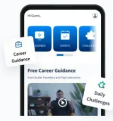
- About Us
- Blog
- Careers
- Review

Download the SCALER app!

Get all scaler resources under one roof!

4.4 ★ 1.71 K Reviews | 100K+ Downloads

GET IT ON Google Play



Popular Free Certification Courses

Java Course for Beginners | C++ Course with Certificate | Python Course for Beginners | Javascript Free Course for Beginners | Data Science Course for Beginners | DBMS Course | Python and SQL for Data Science Course | DSA Problem Solving for Interviews | Instagram System Design Course | Dynamic Programming Course | All Courses

Popular Tutorials

Python Tutorial | Java Tutorial | DBMS Tutorial | Javascript Tutorial | C++ Tutorial | SQL Tutorial | Software Engineering Tutorial | Data Science Tutorial | Pandas Tutorial | Deep Learning Tutorial | All Tutorials

Compilers

Python Compiler | Java Compiler | Javascript Compiler | C Compiler | C++ Compiler

Tools

Json Validator | SQL Formatter | XML Formatter | CSS Formatter | JavaScript Formatter

Privacy Policy • Terms of Use • Contact Us