


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

```
dataset = pd.read_excel('ENB2012_data.xlsx')
dataset
```



	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
<b>0</b>	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
<b>1</b>	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
<b>2</b>	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
<b>3</b>	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
<b>4</b>	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28
...	...	...	...	...	...	...	...	...	...	...
<b>763</b>	0.64	784.0	343.0	220.50	3.5	5	0.4	5	17.88	21.40
<b>764</b>	0.62	808.5	367.5	220.50	3.5	2	0.4	5	16.54	16.88
<b>765</b>	0.62	808.5	367.5	220.50	3.5	3	0.4	5	16.44	17.11
<b>766</b>	0.62	808.5	367.5	220.50	3.5	4	0.4	5	16.48	16.61
<b>767</b>	0.62	808.5	367.5	220.50	3.5	5	0.4	5	16.64	16.03

768 rows × 10 columns

```
import pandas as pd
import numpy as np
```

```
dataset.dropna()
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33



```
X = dataset.iloc[:, :-1].values
```

```
y = dataset['Y2'].values
```

```
print(X.shape)
```

```
print(y.shape)
```

```
(768, 9)
```

```
(768,)
```

```
768 0.98 514.5 294.0 110.25 7.0 2 0.0 0 15.55 21.33
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

## Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

```
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

```
y_pred_lr = lr.predict(X_test)
```

```
print(y_pred_lr)
```

```
[17.1961682 14.40403505 39.26971605 22.08349881 33.17730559 30.7609296
 30.00502366 30.96727122 31.83504694 26.89952375 11.62181962 43.46382784
 14.37609801 43.64874522 42.18373032 29.45764819 14.71789418 31.08192167
 16.90955079 15.48468305 15.25471954 33.42829831 14.19409027 39.83484556
 11.37454107 17.18600514 15.84731706 16.38597446 15.06305657 40.55338255
 15.0735664 15.62782665 41.67176942 15.80289284 29.99075779 36.8221227
 16.30010797 36.25114839 19.17051261 14.45357665 14.82276964 41.46814237
 17.28352257 28.45170274 23.87760691 26.70059103 17.06719861 15.34355282
 31.11695366 34.54681104 33.20642814 15.92615713 34.38770235 16.5008912
 37.51799051 14.81382489 19.56583311 29.65785916 41.49581702 16.73272356
 16.98317425 15.32487546 33.04130191 13.9686625 15.95657098 14.77631042]
```

```

40.78470838 14.04753489 28.44431041 41.39075019 33.17437431 14.21934207
27.02237855 18.92937108 17.99212119 28.39598701 33.0525655 40.03866788
32.89525177 17.00989299 36.8712198 31.94389068 15.20343694 16.12695941
31.96107776 28.59122851 26.27060813 33.26473285 15.12823615 19.02704508
17.02557124 29.60713167 15.89374007 27.40474426 40.96836677 16.76494854
31.67102254 16.83815681 11.94031434 36.8981735 39.05918047 26.27963574
24.12621227 34.83184462 14.20835614 32.40167759 42.12156123 15.83056749
15.69024866 19.76114561 18.69483791 34.88473557 34.1019537 18.32198948
15.63810828 18.01730198 14.67792524 38.57696842 29.21650667 38.62706374
35.42388635 15.6349171 16.08087366 37.0043277 15.32736933 40.74609468
16.84905717 29.83989743 15.08553048 33.41797579 34.75206508 27.20943176
37.55765019 15.17577615 37.72967858 11.54968934 31.75837242 26.96973812
34.64359405 16.47836533 20.00454864 20.92465624 30.9038532 14.44161416
37.71778868 18.51468227 31.33034952 38.3262356 14.95289231 16.03957402
29.52526997 14.62608026 16.05598509 16.3492315 17.50492301 35.2121455
31.30194205 27.28268491 15.27125868 26.40125767 41.52284762 16.32944353
26.77695499 33.57677132 16.5144744 21.0032009 15.35864406 32.80841189
15.31027551 33.650643 40.45213321 15.27269958 33.30907848 41.53583675
35.08747641 26.34158244 31.63720507 30.15489238 16.35097479 26.1640224
14.28222623 43.61432179 30.66045367 12.32656048 15.24170948 27.51249185
14.82336293 15.60010951 30.22586669 16.97063841 15.45564608 15.56565058]

```

```

coefficients = lr.coef_
intercept = lr.intercept_
print('The coefficient is {} and the interception is {}'.format(coefficients[0], intercept))

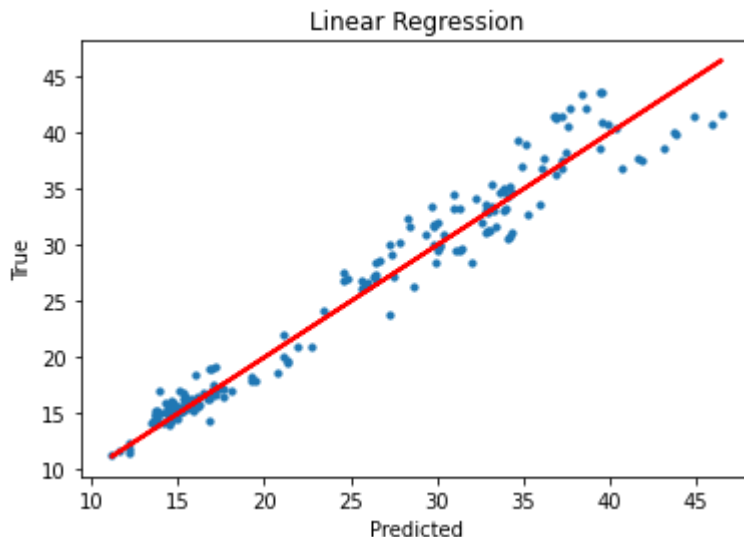
```

The coefficient is -1.6074934723012793 and the interception is 24.496491007349682

```

plt.scatter(y_test, y_pred_lr, s=10)
plt.plot(y_test, y_test, c='red', lw=2)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Linear Regression")
plt.show()

```



```

from sklearn.metrics import mean_squared_error, r2_score
rmse = mean_squared_error(y_test, y_pred_lr, squared=False)
r2 = r2_score(y_test, y_pred_lr)
print("Root mean squared error:", rmse)
print("R2 score:", r2)

```

```

Root mean squared error: 1.952846468214747
R2 score: 0.9587841435290855

```

```

results = pd.DataFrame(['LinearRegression', rmse, r2]),
                      columns = ['Model', 'RMSE', 'R2-Score'])

```

## Ridge Regression

```

from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
import decimal

```

```

def float_range(start, stop, step):
    while start < stop:
        yield float(start)
        start += decimal.Decimal(step)

```

```

params = {'alpha': list(float_range(0, 1, '0.1'))}
grid_search_cv = GridSearchCV(Ridge(), params, refit = False)
grid_search_cv.fit(X_train, y_train)
grid_search_cv.best_params_

```

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:157: LinAlgWarning
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
{'alpha': 0.9}

```



```

from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.7, random_state=0)
ridge.fit(X_train, y_train)

```

```

Ridge(alpha=0.7, random_state=0)

```

```

y_pred_ridge = ridge.predict(X_test)
print(y_pred_ridge)

```

```

[17.23037703 14.31411674 39.15198542 22.13898652 33.15669201 30.698484
 29.98356781 31.03625817 31.80361514 26.96973982 11.53411621 43.33202054
 14.38140839 43.53842624 42.11575816 29.44941113 14.62934106 31.00650676
 16.95803936 15.44775817 15.26406993 33.40850057 14.16073263 39.76026682
 11.2711771 17.15987353 15.87573425 16.39107358 14.95672357 40.4894504
 15.15325327 15.63364345 41.610475 15.80170839 29.99753149 36.81177413]

```

```

16.28983523 36.26570751 19.20059162 14.46374621 14.8208406 41.3381245
17.25441569 28.47878156 23.92969924 26.71963789 16.96115736 15.3198862
31.07797197 34.51981241 33.17398777 15.95652451 34.34177116 16.577676
37.50969827 14.69378446 19.64436219 29.65465842 41.36230262 16.74855484
17.0046116 15.3061343 33.02341756 13.90614294 15.90448614 14.75266716
40.64874875 14.07140038 28.45909393 41.25045189 33.18628541 14.16219969
27.08325659 18.98712383 17.95573557 28.41941502 33.07070466 39.93461719
32.92013445 16.99387258 36.79066088 31.88459452 15.18321256 16.21098233
31.98912398 28.62768573 26.28895305 33.17867868 15.21468379 19.06751615
16.96857938 29.58916999 15.82524169 27.38108628 40.92439738 16.87754192
31.70252494 16.87141589 11.90389511 36.85884303 38.93310234 26.35054195
24.14296807 34.80597839 14.12649309 32.37032882 42.0703553 15.8258865
15.65789389 19.82097552 18.77513591 34.84745226 34.08450817 18.27250571
15.57168835 17.98886002 14.66299009 38.47450936 29.23594333 38.64262257
35.41714491 15.56262178 16.01112055 36.93770004 15.29718797 40.69470295
16.8073968 29.8329976 15.06570727 33.46326554 34.67976604 27.20447296
37.46168009 15.13046987 37.67903131 11.55066845 31.78105225 26.9841228
34.63398447 16.46696673 19.95336163 20.94319498 30.89969261 14.39189412
37.63014297 18.56705136 31.29891904 38.253404 14.98248823 16.14828371
29.53185719 14.61945224 16.12517473 16.44067676 17.42510837 35.15734004
31.25272029 27.32316545 15.32228861 26.42910985 41.38372634 16.24021807
26.78273445 33.53449472 16.4787865 20.98131755 15.38498723 32.84593268
15.43788218 33.6631676 40.4081039 15.23575744 33.23824419 41.44554317
35.03969531 26.37476065 31.65177684 30.16033952 16.40407555 26.21564205
14.19853599 43.48554411 30.67855643 12.38541092 15.24901908 27.52100353
14.73637342 15.54564522 30.24614713 17.0213349 15.52457157 15.50859178]

```

```

coefficients = ridge.coef_
intercept = ridge.intercept_
print('The coefficient is {} and the interception is {}'.format(coefficients[0], intercept))

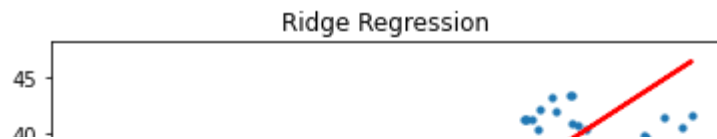
```

The coefficient is -1.4526154955786839 and the interception is 24.502239583333346

```

plt.scatter(y_test, y_pred_ridge, s=10)
plt.plot(y_test, y_test, c='red', lw=2)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Ridge Regression")
plt.show()

```



```
from sklearn.metrics import mean_squared_error, r2_score
rmse = mean_squared_error(y_test, y_pred_ridge, squared=False)
r2 = r2_score(y_test, y_pred_ridge)
print("Root mean squared error:", rmse)
print("R2 score:", r2)
```

```
Root mean squared error: 1.944725437570994
R2 score: 0.9591262280421611
```



```
model_results = pd.DataFrame([['RidgeRegression', rmse, r2]],
                              columns = ['Model', 'RMSE', 'R2-Score'])
results = results.append(model_results, ignore_index = True)
```

## Support Vector Machine

```
from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train, y_train)
```

```
SVR()
```

```
y_pred_svr = svr.predict(X_test)
print(y_pred_svr)
```

```
[17.74087883 13.50557653 35.54057997 24.24547693 31.92906286 29.38389125
 29.83193903 31.96216521 30.67411473 27.12437837 14.60053168 38.013464
 15.36451664 39.37769948 37.53347116 27.28934316 14.11131006 29.31612674
 17.44278051 14.63894657 15.03576709 32.85895244 13.65561605 37.31425667
 15.12370336 16.82013949 16.61396967 16.93285952 14.40065933 35.34461871
 16.81164469 15.82038091 37.89770004 15.88623358 30.12483003 36.79502009
 16.59147405 34.63037957 19.55731321 16.23123793 14.69711018 38.70312654
 17.86607581 30.33546863 23.55863164 26.0646488 15.52412276 15.41450724
 29.92200657 33.20297132 32.95625751 15.67856269 33.12973095 17.75490606
 36.28527666 14.65643021 22.96776935 30.58446975 37.30451984 17.16165518
 17.7749036 14.82192973 33.48091965 14.31836158 15.02022184 14.58852803
 37.27666983 14.91627012 26.75618815 37.69173798 32.29489965 13.68498389
 27.17997522 18.93949977 16.98010679 30.22371666 31.96035719 35.94369533
 31.17687885 17.23378329 35.62282593 30.24595903 14.6667905 17.24937363
 32.75496017 30.33382017 26.25285085 29.80824281 16.87135095 19.33593713
 16.81481157 27.17858787 15.01631531 25.87400811 36.65361795 18.65241949
 33.72250468 17.27934803 15.08799479 38.26945043 35.5117735 26.99287874
 23.8426005 33.40375499 15.07632135 30.91867025 35.64523705 16.2796919
 15.23754543 23.22053415 18.99862741 34.26558255 32.92718886 16.7857683
 15.05897776 17.51779312 15.88516773 34.11198876 26.93979224 35.33061567
 35.20308455 14.78401216 15.15734315 36.48577115 14.49656931 39.1587149
 16.6920561 30.17064702 14.7297215 30.61882545 33.21139188 25.80591575]
```

```

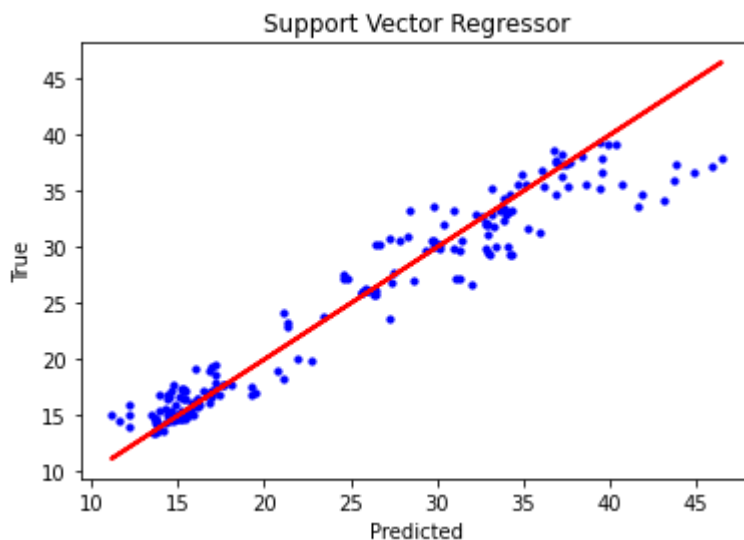
34.6583281 14.41929122 35.48420237 14.00429077 33.2078029 26.06671265
33.27043219 16.46591704 18.37851123 19.94127077 29.63001037 13.7903135
33.65735208 19.13759605 29.49979688 37.40798403 15.10089245 17.20510014
30.12868236 14.90176778 17.68506309 17.13622919 17.24053506 34.74600327
29.35079321 27.6653016 15.9571421 26.1433768 37.54255933 15.67307194
25.87855704 32.11425067 16.57703972 19.99353701 16.11829875 31.59641802
17.44477493 31.29828177 39.243437 14.65256809 29.70878263 36.68258304
32.45571016 26.2250741 30.04316615 30.8120547 16.95612338 26.02518777
13.53488232 37.98066642 29.99001571 15.88709525 16.47636179 27.58924029
13.57550067 14.77681597 30.689275 16.77214178 16.49035135 15.13412643]

```

```

plt.scatter(y_test, y_pred_svr, c='blue', s=10)
plt.plot(y_test, y_test, c='red', lw=2)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Support Vector Regressor")
plt.show()

```



```

from sklearn.metrics import mean_squared_error, r2_score
rmse = mean_squared_error(y_test, y_pred_svr, squared=False)
r2 = r2_score(y_test, y_pred_svr)
print("Root mean squared error:", rmse)
print("R2 score:", r2)

```

```

Root mean squared error: 2.4788160021738745
R2 score: 0.9335925643754233

```

```

model_results = pd.DataFrame([['SupportVectorRegression', rmse, r2]],
                              columns = ['Model', 'RMSE', 'R2-Score'])
results = results.append(model_results, ignore_index = True)

```

## Support Vector Machine

```

from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
import decimal

def float_range(start, stop, step):
    while start < stop:
        yield float(start)
        start += decimal.Decimal(step)

params = {'kernel':['linear', 'poly', 'rbf'], 'C': list(range(1, 10)), 'gamma':['auto'], 'eps
grid_search_cv = GridSearchCV(SVR(), params, refit = False)
grid_search_cv.fit(X_train, y_train)
grid_search_cv.best_params_

{'C': 9, 'epsilon': 0.2, 'gamma': 'auto', 'kernel': 'rbf'}

from sklearn.svm import SVR
svr = SVR(kernel='rbf', C=9, gamma='auto', epsilon=0.1)
svr.fit(X_train, y_train)

SVR(C=9, gamma='auto')

y_pred_svr = svr.predict(X_test)
print(y_pred_svr)

[17.80374494 13.52510672 36.52057273 22.85717736 32.19617201 29.87956598
29.14622028 31.42969129 31.80951763 27.08111624 11.86421344 39.79094306
14.80261211 41.37476273 41.48743878 27.79273801 13.91217543 30.96987764
15.41408655 14.90870197 15.21672237 32.84175715 13.70274665 40.49853401
12.14584202 16.82022323 14.36318787 16.74195466 13.84357002 37.47739492
14.70458912 15.92674852 41.36546579 15.88034614 29.72220776 36.42683086
16.21834841 34.62888873 18.06481534 14.97001118 15.18540946 38.97422316
17.43655876 29.18156526 23.78758269 25.8711217 16.17716863 15.22348191
30.50236283 35.58948772 32.78964516 14.36131141 33.39914619 17.57940132
36.16357243 13.76846954 20.50729978 31.69312382 40.0354172 17.29161506
17.66528042 14.72349671 33.32351343 13.67306412 15.2751366 14.42990258
39.49232029 14.59566409 27.72876516 39.75870129 32.91211525 13.77668174
27.18048607 17.82471002 18.27635414 29.54293225 31.88179676 39.58879299
30.48344068 16.78688626 36.96915353 30.22828527 15.36317343 14.89348678
34.8111789 26.96638038 26.77228506 31.44603405 15.00456355 18.00268202
16.85093819 28.06886184 15.22174822 25.07499754 39.15502785 16.28974754
31.69558151 16.97662946 12.40527761 36.97270287 36.11227743 27.16267268
23.33944904 34.00756954 13.91353767 32.39375522 38.3735507 16.1470217
15.09442744 21.03433011 19.61698388 34.23477674 32.35112898 18.65370187
15.14610502 18.22307813 14.87997925 36.6329157 27.39552358 36.01430078
36.40678658 15.14535886 15.34665804 36.12802148 14.89374443 40.15840352
16.94201332 29.66879281 14.72531361 30.03646575 33.8969037 24.84135053
36.02527487 14.79390023 36.44482004 12.29894794 33.89068316 26.04929121
33.8761466 16.27735171 20.23530465 21.92806749 31.03466031 13.5928642
37.05554249 17.48827085 31.97673099 37.22159064 14.80488068 15.42550895
30.71702306 14.90596662 14.76446963 15.32964564 17.25020732 34.19711582
29.9992255 26.60381448 15.71676726 26.23710852 38.06349653 15.50467078

```



```

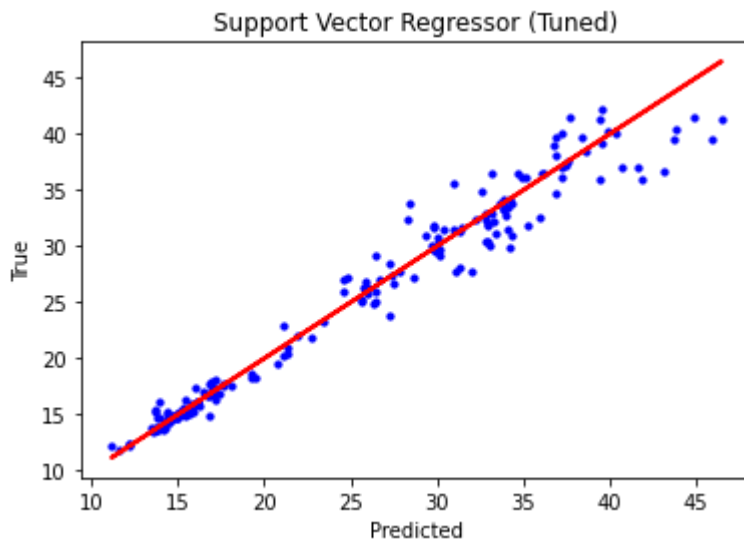
25.22597718 32.42651685 16.16980971 22.04394788 15.91825907 31.8598131
15.24005285 32.61457005 39.98675743 15.06563047 31.2462258 41.54564101
33.30270947 26.34599028 31.12142317 28.40726334 16.75480195 25.04432325
13.60569599 42.16622078 31.56208831 12.35049324 15.62519454 26.01560297
13.73932177 15.06460163 27.81607224 15.17414629 14.66665443 15.16215978]

```

```

plt.scatter(y_test, y_pred_svr, c='blue', s=10)
plt.plot(y_test, y_test, c='red', lw=2)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Support Vector Regressor (Tuned)")
plt.show()

```



```

from sklearn.metrics import mean_squared_error, r2_score
rmse = mean_squared_error(y_test, y_pred_svr, squared=False)
r2 = r2_score(y_test, y_pred_svr)
print("Root mean squared error:", rmse)
print("R2 score:", r2)

```

```

Root mean squared error: 1.7705835893045028
R2 score: 0.966118627332151

```

```

model_results = pd.DataFrame([[ 'TunedSupportVectorRegression', rmse, r2]],
                              columns = ['Model', 'RMSE', 'R2-Score'])
results = results.append(model_results, ignore_index = True)

```

## Neural Network

```

import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.compose import ColumnTransformer
import keras

```

```
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from keras.models import Sequential
ann = Sequential()
ann.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
```

```
second = Sequential()
```

```
second.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
```

```
output = Sequential()
```

```
output.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
```

```
min_max_scaler = preprocessing.MinMaxScaler()
```

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
```

```
min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X_scale, y, test_size=0.3)
X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test, test_size=0.5)
```

```
output.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics = ['mse'])
```

```
X_scaled = min_max_scaler.fit_transform(X)
X_train, X_test, Y_train, y_test = train_test_split(X_scaled,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=42)
```

```
hist = output.fit(X_train, Y_train,
                  batch_size=10, epochs=100,
                  validation_data=(X_val, Y_val))
```

```
54/54 [=====] - 0s 2ms/step - loss: 656.0491 - mse: 656.0491 ^
Epoch 25/100
54/54 [=====] - 0s 2ms/step - loss: 655.9662 - mse: 655.9662
Epoch 26/100
54/54 [=====] - 0s 2ms/step - loss: 655.8915 - mse: 655.8915
Epoch 27/100
54/54 [=====] - 0s 2ms/step - loss: 655.8232 - mse: 655.8232
Epoch 28/100
54/54 [=====] - 0s 2ms/step - loss: 655.7614 - mse: 655.7614
Epoch 29/100
54/54 [=====] - 0s 2ms/step - loss: 655.7047 - mse: 655.7047
Epoch 30/100
54/54 [=====] - 0s 2ms/step - loss: 655.6526 - mse: 655.6526
Epoch 31/100
54/54 [=====] - 0s 2ms/step - loss: 655.6047 - mse: 655.6047
Epoch 32/100
54/54 [=====] - 0s 2ms/step - loss: 655.5607 - mse: 655.5607
Epoch 33/100
54/54 [=====] - 0s 2ms/step - loss: 655.5200 - mse: 655.5200
Epoch 34/100
54/54 [=====] - 0s 2ms/step - loss: 655.4822 - mse: 655.4822
Epoch 35/100
54/54 [=====] - 0s 2ms/step - loss: 655.4471 - mse: 655.4471
Epoch 36/100
54/54 [=====] - 0s 2ms/step - loss: 655.4147 - mse: 655.4147
Epoch 37/100
54/54 [=====] - 0s 2ms/step - loss: 655.3844 - mse: 655.3844
Epoch 38/100
54/54 [=====] - 0s 2ms/step - loss: 655.3560 - mse: 655.3560
Epoch 39/100
54/54 [=====] - 0s 2ms/step - loss: 655.3296 - mse: 655.3297
Epoch 40/100
54/54 [=====] - 0s 2ms/step - loss: 655.3049 - mse: 655.3049
Epoch 41/100
54/54 [=====] - 0s 2ms/step - loss: 655.2819 - mse: 655.2819
Epoch 42/100
54/54 [=====] - 0s 3ms/step - loss: 655.2599 - mse: 655.2599
Epoch 43/100
54/54 [=====] - 0s 2ms/step - loss: 655.2394 - mse: 655.2394
```

```

Epoch 44/100
54/54 [=====] - 0s 2ms/step - loss: 655.2202 - mse: 655.2202
Epoch 45/100
54/54 [=====] - 0s 2ms/step - loss: 655.2020 - mse: 655.2020
Epoch 46/100
54/54 [=====] - 0s 2ms/step - loss: 655.1848 - mse: 655.1848
Epoch 47/100
54/54 [=====] - 0s 2ms/step - loss: 655.1686 - mse: 655.1686
Epoch 48/100
54/54 [=====] - 0s 2ms/step - loss: 655.1533 - mse: 655.1533
Epoch 49/100
54/54 [=====] - 0s 2ms/step - loss: 655.1389 - mse: 655.1389
Epoch 50/100
54/54 [=====] - 0s 2ms/step - loss: 655.1252 - mse: 655.1252
Epoch 51/100
54/54 [=====] - 0s 2ms/step - loss: 655.1122 - mse: 655.1122
Epoch 52/100
54/54 [=====] - 0s 2ms/step - loss: 655.0999 - mse: 655.0999

```

```
output.evaluate(X_test, y_test)
```

```

6/6 [=====] - 0s 4ms/step - loss: 683.6430 - mse: 683.6430
[683.6430053710938, 683.6430053710938]

```

```
y_pred_ann = output.predict(X_test)
```

```

plt.scatter(y_test, y_pred_ann, c='red', s=10)
plt.plot(y_test, y_test, c='blue', lw=2)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Neural Network")
plt.show()

```

```
from sklearn.metrics import mean_squared_error, r2_score
rmse = mean_squared_error(y_test, y_pred_ann, squared=False)
r2 = r2_score(y_test, y_pred_ann)
print("Root mean squared error:", rmse)
print("R2 score:", r2)
```

Root mean squared error: 26.146567698026367  
R2 score: -6.322251945186935

```
model_results = pd.DataFrame([['NeuralNetwork', rmse, r2]],
                              columns = ['Model', 'RMSE', 'R2-Score'])
results = results.append(model_results, ignore_index = True)
results
```

	Model	RMSE	R2-Score	
0	LinearRegression	1.952846	0.958784	
1	RidgeRegression	1.944725	0.959126	
2	SupportVectorRegression	2.478816	0.933593	
3	TunedSupportVectorRegression	1.770584	0.966119	
4	NeuralNetwork	26.146568	-6.322252	