

- Runtime environment (Cloud - colab)

```
print("sklearn version", sklearn.__version__)
print("python version", sys.version)

sklearn version 1.0.2
python version 3.7.13 (default, Mar 16 2022, 17:37:17)
[GCC 7.5.0]
```

1. SVR

```
[148] from sklearn.svm import SVR

svm_reg = SVR(kernel="linear", C=30000)
svm_reg.fit(housing_prepared, housing_labels)
housing_predictions = svm_reg.predict(housing_prepared)
svm_mse = mean_squared_error(housing_labels, housing_predictions)
svm_rmse = np.sqrt(svm_mse)
svm_rmse

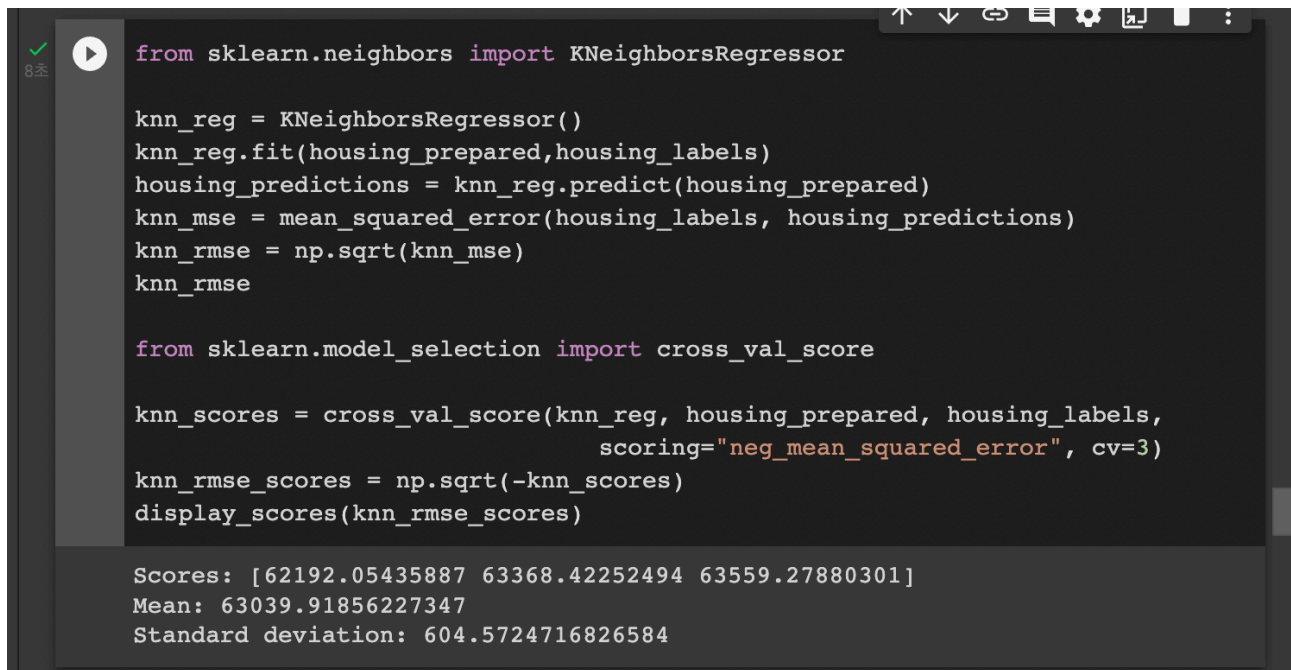
70158.80248353248

from sklearn.model_selection import cross_val_score

svm_scores = cross_val_score(svm_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=3)
svm_rmse_scores = np.sqrt(-svm_scores)
display_scores(svm_rmse_scores)

Scores: [68813.3993301  83178.17296742 70915.38578773]
Mean: 74302.31936175143
Standard deviation: 6334.570213729821
```

## 2. KNeighborsRegressor



A screenshot of a Jupyter Notebook interface. The top bar shows a green checkmark, a play button, and a timer at 8초. The code cell contains the following Python code:

```
from sklearn.neighbors import KNeighborsRegressor

knn_reg = KNeighborsRegressor()
knn_reg.fit(housing_prepared, housing_labels)
housing_predictions = knn_reg.predict(housing_prepared)
knn_mse = mean_squared_error(housing_labels, housing_predictions)
knn_rmse = np.sqrt(knn_mse)
knn_rmse

from sklearn.model_selection import cross_val_score

knn_scores = cross_val_score(knn_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=3)
knn_rmse_scores = np.sqrt(-knn_scores)
display_scores(knn_rmse_scores)
```

The output cell shows the following results:

```
Scores: [62192.05435887 63368.42252494 63559.27880301]
Mean: 63039.91856227347
Standard deviation: 604.5724716826584
```

### Summary description:

- which model you used : KNeighborsRegressor
- difference value of “Mean” value between two models’ outputs : **11,262.4007995**  
(74302.31936175143-63039.91856227347)
- any reason why you believe that your new one is better than SVR :  
K-Nearest Neighbor (KNN)은 데이터가 주어지면 샘플에 가장 가까운 샘플 k개를 선택한다. 회귀이기에 이웃한 샘플의 타깃은 어떤 클래스가 아니라 임의의 수치이며 이 수치들의 평균을 구하여 새로운 샘플의 타깃값을 예측하는 방식이다. 기본적으로 SVR은 광범위한 변수 세트에 대한 최적화 전략이 향상되어 선형 회귀, KNN 및 Elastic Net과 같은 다른 알고리즘보다 더 나은 성능 예측을 수행한다고 한다. 하지만, KNN의 mean값이 더 작게 나온 것에 대한 의문이 들어 고민을 해보았는데, 모델의 parameter값으로 인해 이러한 결과를 보이는 것 같다.

### 3. GradientBoostingRegressor

```
24 ✓ ▶ from sklearn.ensemble import GradientBoostingRegressor

gdb_reg = GradientBoostingRegressor(random_state=1)
gdb_reg.fit(housing_prepared, housing_labels)
housing_predictions = gdb_reg.predict(housing_prepared)
gdb_mse = mean_squared_error(housing_labels, housing_predictions)
gdb_rmse = np.sqrt(gdb_mse)
gdb_rmse

from sklearn.model_selection import cross_val_score

gdb_scores = cross_val_score(gdb_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=3)
gdb_rmse_scores = np.sqrt(-gdb_scores)
display_scores(gdb_rmse_scores)

[→ Scores: [51845.13303491 53479.07029821 54084.76357782]
Mean: 53136.32230364679
Standard deviation: 945.9011163987624
```

#### Summary description:

- which model you used : GradientBoostingRegressor
- difference value of “Mean” value between two models’ outputs : **21,165.9970581**  
(74302.31936175143-53136.32230364679)
- any reason why you believe that your new one is better than SVR :  
GradientBoostingRegressor는 여러개의 결정 트리를 묶어 모델을 만든다. 이는 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습시킨다는 특징이 있다. 즉, 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리를 만든다. 기본적으로 수업시간에 학습한 RandomForestRegressor처럼 앙상블 학습을 진행하므로 SVR과 같은 단일모델에 비해 비교적 좋은 성능을 나타낸다. 해당 모델은 좋은 성능을 보이는 반면 매개변수에 대해 민감하다는 점이 있지만 이번 과제에서는 매개변수와 상관없이 학습을 진행하였다.

#### 4. VotingRegressor

```
✓ [192] from sklearn.ensemble import GradientBoostingRegressor
1분 from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingRegressor

reg1 = GradientBoostingRegressor(random_state=1)
reg2 = RandomForestRegressor(random_state=1)
ereg = VotingRegressor(estimators=[('gb', reg1), ('rf', reg2)])
ereg = ereg.fit(housing_prepared, housing_labels)
housing_predictions = ereg.predict(housing_prepared)
ereg_mse = mean_squared_error(housing_labels, housing_predictions)
ereg_rmse = np.sqrt(ereg_mse)
ereg_rmse

from sklearn.model_selection import cross_val_score

ereg_scores = cross_val_score(ereg, housing_prepared, housing_labels,
                              scoring="neg_mean_squared_error", cv=3)
ereg_rmse_scores = np.sqrt(-ereg_scores)
display_scores(ereg_rmse_scores)

Scores: [49792.97091959 51040.47221777 51826.04000959]
Mean: 50886.494382314304
Standard deviation: 837.1078630357094
```

#### Summary description:

- which model you used : VotingRegressor
- difference value of “Mean” value between two models’ outputs : **23,415.8249794**  
(74302.31936175143-50886.494382314304)
- any reason why you believe that your new one is better than SVR :  
앞선 모델들을 바탕으로 앙상블 방식의 이점을 활용해 VotingRegressor 모델을 사용함으로써 성능이 비교적 좋은 GradientBoostingRegressor와 RandomForestRegressor를 combine 해보았다. VotingRegressor는 서로 다른 ML regressor를 combine하고 average predicted value를 반환한다. 따라서 각 모델의 weaknesses balance를 맞추기 위해 동등하게 잘 수행되는 model set에 유용하다고 한다. 이러한 이유들로 인해 SVR보다 좋은 성능을 나타낸다.