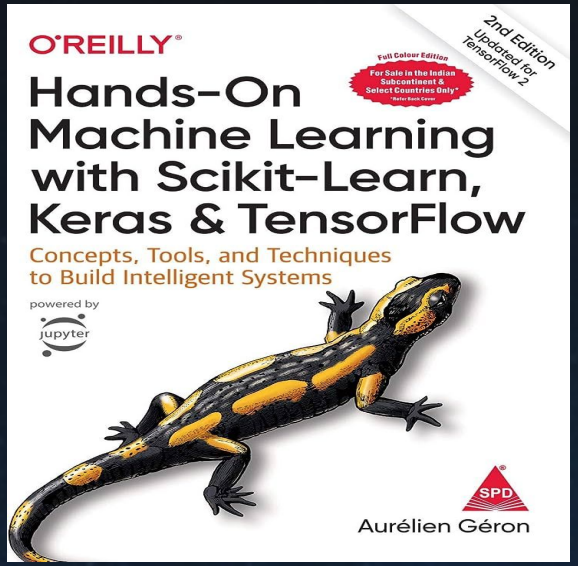
PROJECT COMPARISON

Harshit-Budakoti

https://github.com/Harshit-Budakoti/california_house_price_prediction



handson-ml2/02_end_to_end_machine_learning_project.ipynb at master · ageron/handson-ml2 · GitHub

Disclaimer : The intent of this report is not to reach a conclusion on which platform is better. I also used normalization to avoid comparing the raw execution times. Both the platforms are good at their core competencies and I only intend to compare the efficiency of my code and the model used for the ML project with respect to the book’s code for the project.

To compare efficiencies on level grounds I have used a Normalization factor after observing run-times of exact same code in both the platforms separately.

Jupyter Notebook runs locally.
My project was developed in this mode,

Normalized Exec.Time for Google COOP = Raw-time/4.9

The book's code was run by me to compare execution time in GOOGLE COLLAB..

EXECUTION TIME DATA in seconds

Stage	My_project	Normalized_Book_project	Book_project
Data Extraction	0.0611	0.1247520325	0.61378
Data Exploration and cleaning	0.14299	0.04053861789	0.19945
Missing value handling	0.09779	0.121898374	0.59974
Categorical handling	0.02856	0.02763443089	0.1359614
train test split	0.03661	0.01174796748	0.0578
Transformations and Preprocessing	0.0462	0.06199186992	0.305

EXCECUTION TIME EFFICIENCY ANALYSIS

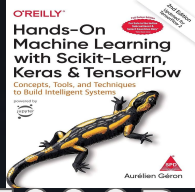
Stage	% Normalized Efficiency		% Raw efficiency	
Data Extraction	51.02284206	% BETTER	90.0452931	%BETTER
Data Exploration and cleaning	-252.7253948	% WORSE	28.30784658	%BETTER
Missing value handling	19.77743689	% BETTER	83.69460099	%BETTER
Categorical handling	-3.349332972	% WORSE	78.99403801	%BETTER
train test split	-211.6283737	% WORSE	36.66089965	%BETTER
Transformations and Preprocessing	25.47409836	% BETTER	84.85245902	%BETTER

CODE PERFORMANCE

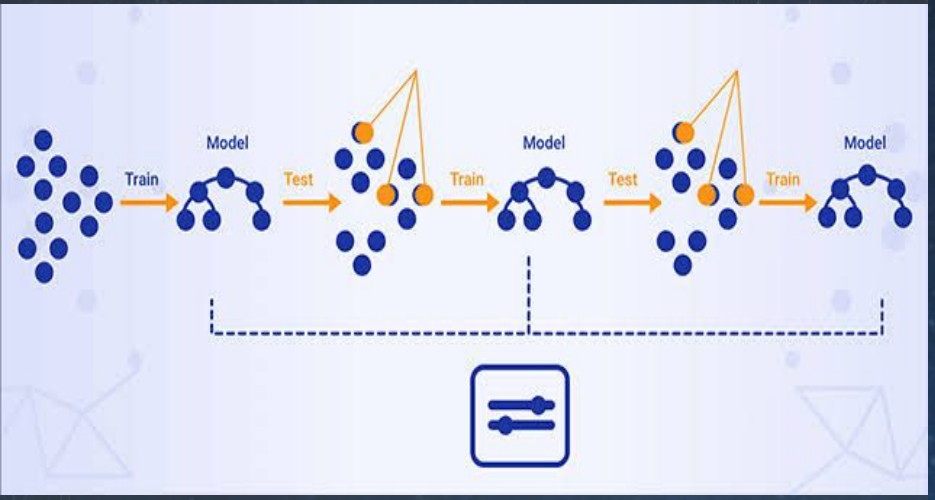
After taking normalization into consideration we can see that our code is better in the following areas :-

1) **Data Extraction :** 51.02% faster
2) **Missing value handling : 19.77% faster**
3) **Transformations and Preprocessing : 25.47% faster**
   This is a particularly important stage in the project where out code is much shorter and faster.

The author's code is relatively faster in Data Cleaning and Exploration and in train-test split stages.

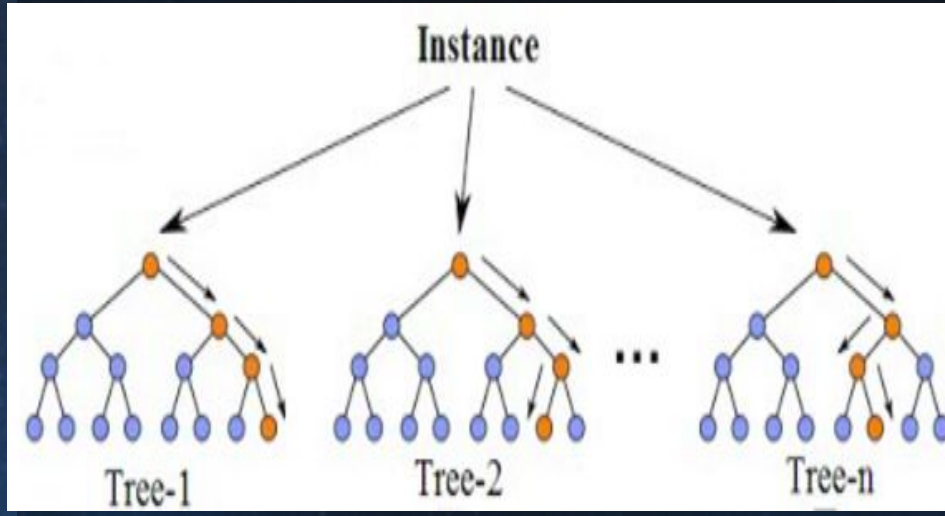


GRADIENT BOOSTING REGRESSOR



GradientBoostingRegressor(max_depth=7, n_estimators=500,learning_rate = 0.1)

RANDOM FOREST REGREGSSOR



RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)

ML MODEL EFFICIENCY METRICS

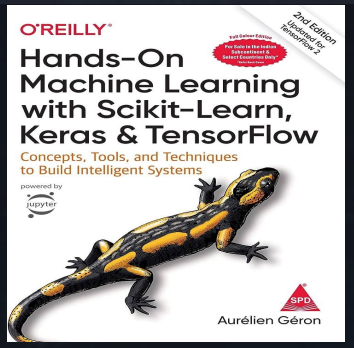
Metric	My_project	Book_project
Hyperparameter Tuning Algorithm time	1028 seconds	459 seconds
r2_score	0.8205061755	0.7749207517
FINAL RMSE	44684.43288	47730.2269

MODEL ACCURACY

The hyperparameter tuning stage in our project takes up nearly 2.23x the time taken by the book's project code. But this has resulted in a better BEST_MODEL for our case.

1) **R2_score : An important metric for accuracy of regressor models. This is coefficient of determination in the field of statistics.**
   R2_score of my model is 4.558 % more than the author's model which a significant upgrade.
2) **FINAL MODEL Root Mean Square Error : An import error measuring metric.**
   The RMSE value of my model is 3045.79 less than author's model. This is a 6.38% decrease in RMSE value.

FINAL VERDICT



4.558 % better

R2_value : 82.05%

R2_value : 77.49%

RMSE: 44684.43

6.38 % better

RMSE: 47730.22