

65020 Open-Source Code for AI- Course Assignment

House Prices - Advanced Regression Techniques



Yuval Luria

318390275

Link to Kaggle profile submission
and place can be found in appendix
in the end of the report

Abstract-I recently completed a Data Science Programming course at Python, a 12-week training providing a deep dive into the world of data science, focusing on improving skills to analyze, predict, and convey data-driven facts from large datasets. In this project I'll be sharing the steps I worked on in the course as a chance to predict the cost of houses, we will go through all the steps from beginning till the end.

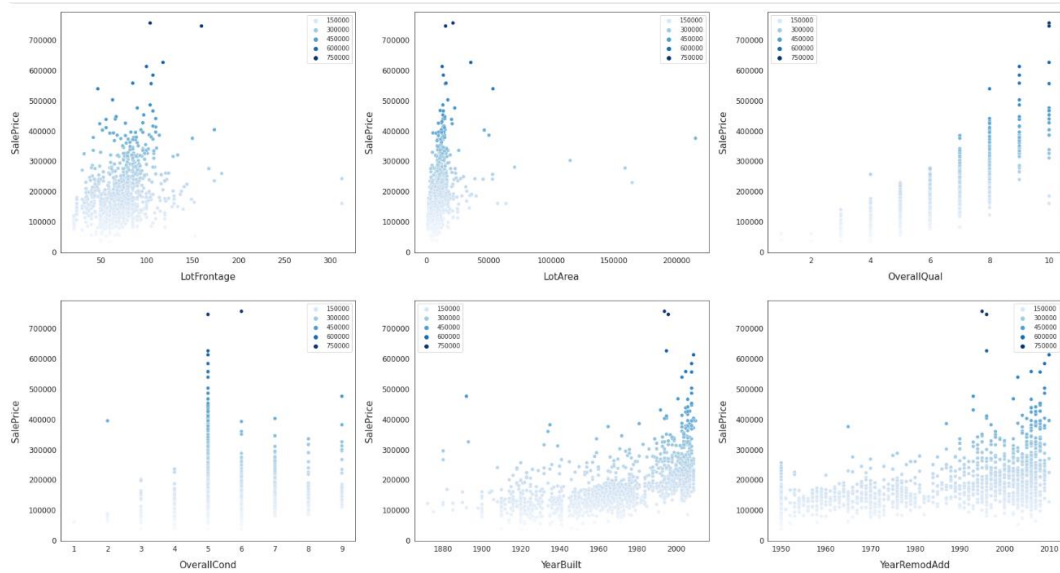
In a nutshell we can see how the model accuracy increased when we applied several transformations to our target variable and to other features in the dataset and we can see also the change in accuracy when we performed feature transformation and created new features and also when we trained multiple models at parallel and combined their predictions.

Introduction- Our goal is to predict the house prices using these features. Purchasing a home remains one of the biggest purchasing decisions that individuals make in their lifetime. Our project aims to predict sale prices of houses in Ames, Iowa by using various aspects of residential homes. The insights gathered could be used by individuals to complement their decision-making process when purchasing a house. This helps to maximize the value users can gain while keeping to a budget. Findings could also be used by a property agent to improve the probability of a sale through proper marketing of key variables. The dataset that we are working on consists of mainly categorical variables stored as integers and factors. We would be focusing on descriptive and predictive analytics, with suggestions on how additional data could be obtained to conduct more experiments to optimize the purchase. The dataset House Prices: Advanced Regression Techniques, contains 2 datasets: train and test sets. Train dataset contains 81 features and contains 1460 observations. This data set is taken from Kaggle.

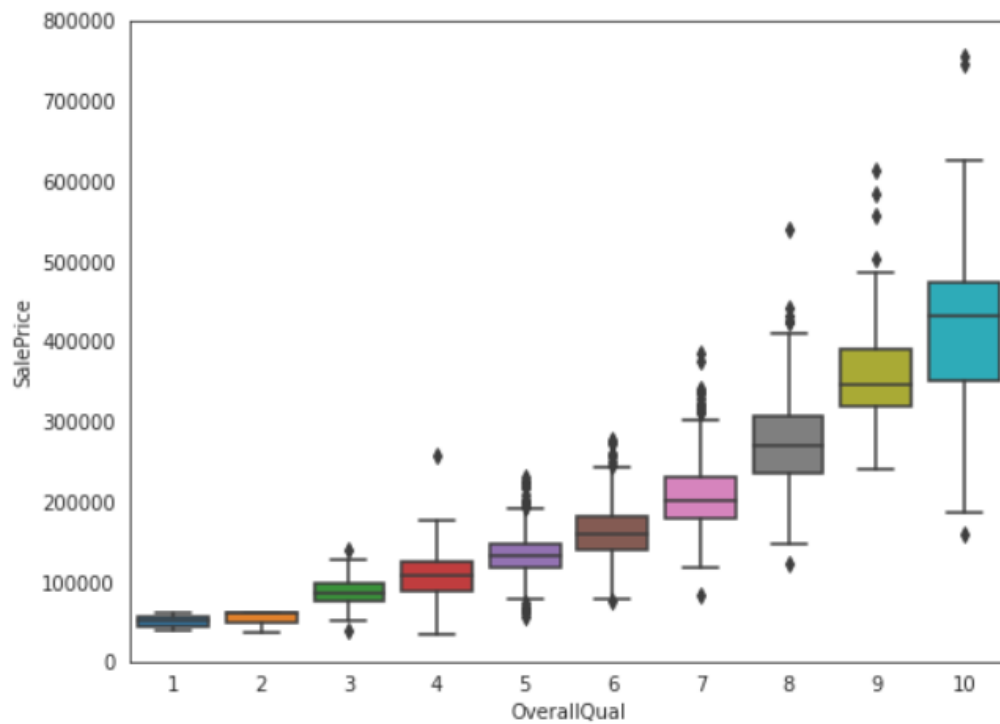
Methods- Exploratory Data Analysis (EDA)

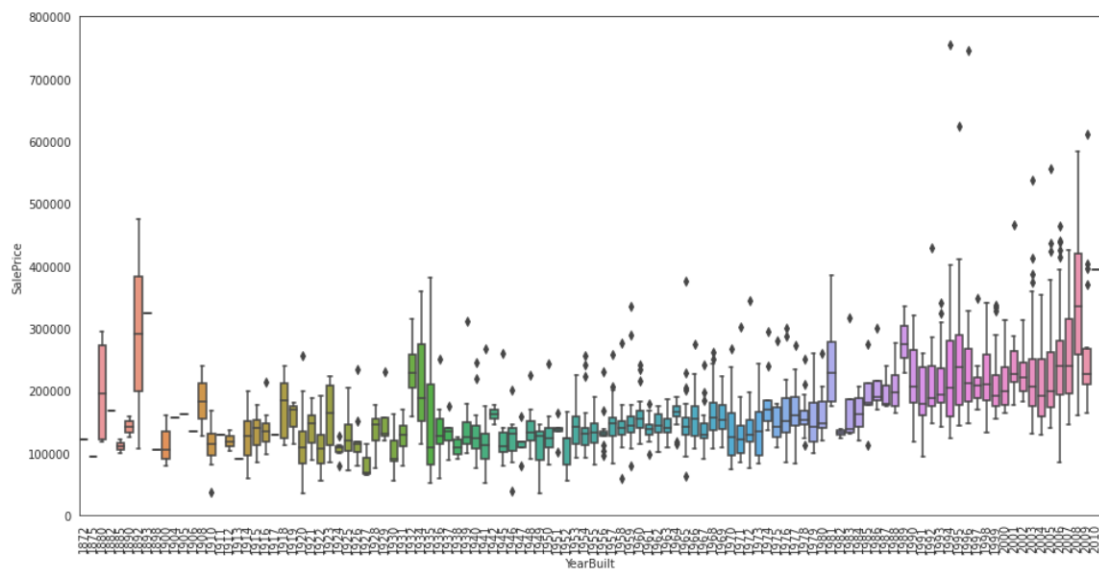
As with any data exercise, we began with some Exploratory Data Analysis.

Let's look at the data, first let's see some of the numeric features

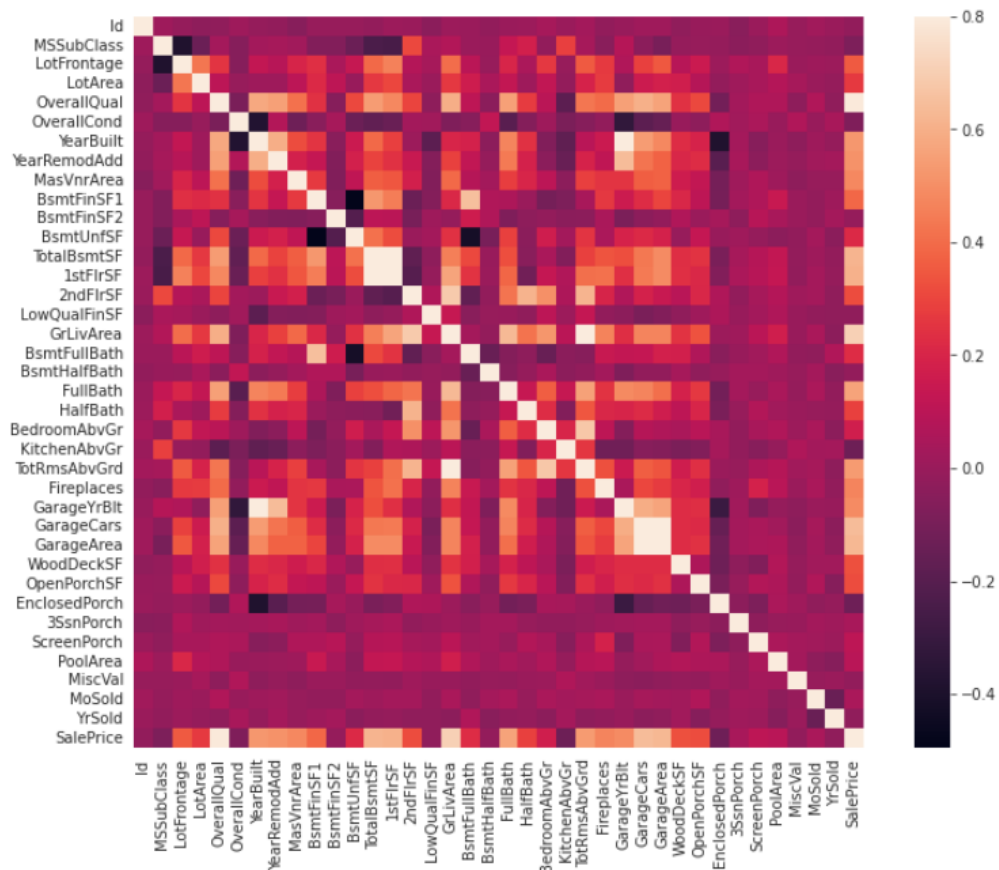


Now lets view the relationship between categorical:





Now let's look at the correlation matrix to see the most correlated features.

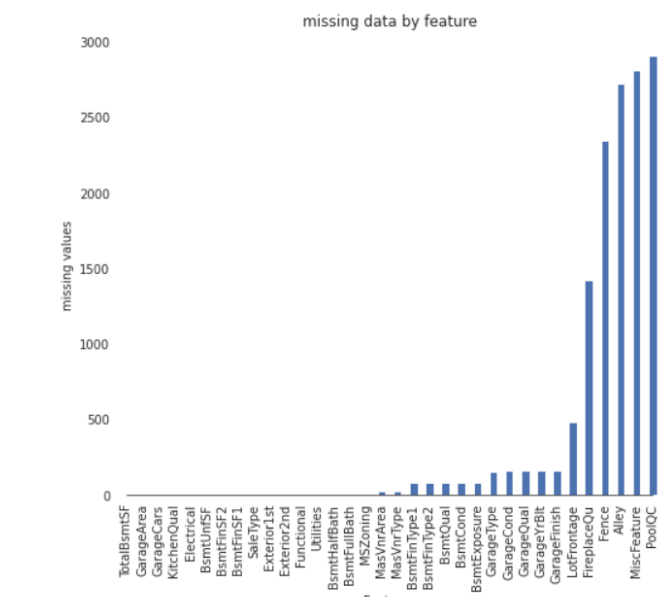


Now that we have seen the data we can begin with PRE-PROCESSING -

First is Data Cleaning-

First, we combine the test and train to look for null/empty features. Then we drop the Id cause its unique for every observation and isn't helping the model. We also drop target since it's the feature we want to predict.

Check for missing values- then our goal is to get a look at the data and see if there is any missing.



categorical values

Now we want to fill categorical values. Note: there is a difference between the nans, since some are meaningful exp. alley N/A means no alley.so we will put none to take it as a category and the second option is when the nun is meaningless, we will use the mode function.

```

# when the nun is meaningfull so we will put none instead- exp. alley N/A means no alley
for column in [
    'Alley',
    'BsmtQual',
    'BsmtCond',
    'BsmtExposure',
    'BsmtFinType1',
    'BsmtFinType2',
    'FireplaceQu',
    'GarageType',
    'GarageFinish',
    'GarageQual',
    'GarageCond',
    'PoolQC',
    'Fence',
    'MiscFeature'
]:
    data2[column] = data2[column].fillna("None")

# when the nun is meaningless
for column in [
    'MSZoning',
    'Utilities',
    'Exterior1st',
    'Exterior2nd',
    'MasVnrType',
    'Electrical',
    'KitchenQual',
    'Functional',
    'SaleType'
]:
    data2[column] = data2[column].fillna(data2[column].mode()[0])

```

numeric values

filling missing numeric values using knn function.

The k-nearest neighbors' algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. For the following features were filled using KNN.

```

def knn_impute(df, na_target):
    df = df.copy()

    numeric_df = df.select_dtypes(np.number)
    non_na_columns = numeric_df.loc[:, numeric_df.isna().sum() == 0].columns

    y_train = numeric_df.loc[numeric_df[na_target].isna() == False, na_target]
    X_train = numeric_df.loc[numeric_df[na_target].isna() == False, non_na_columns]
    X_test = numeric_df.loc[numeric_df[na_target].isna() == True, non_na_columns]

    knn = KNeighborsRegressor()
    knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)

    df.loc[df[na_target].isna() == True, na_target] = y_pred

    return df

```

```

for column in [
    'LotFrontage',
    'MasVnrArea',
    'BsmtFinSF1',
    'BsmtFinSF2',
    'BsmtUnfSF',
    'TotalBsmtSF',
    'BsmtFullBath',
    'BsmtHalfBath',
    'GarageYrBlt',
    'GarageCars',
    'GarageArea'
]:
    data3 = knn_impute(data3, column)

```

After separating the data into numerical and categorical variables, we will do feature engineering (was executed not in the first try, used to improve the predictions). Feature engineering is the process of improving a model's accuracy by using domain knowledge to select and transform data's most relevant variables into features of predictive models that better represent the underlying problem.

The features we created are presented in the image below:

```

]:
data5["SqFtPerRoom"] = data5["GrLivArea"] / (data5["TotRmsAbvGrd"] +
                                              data5["FullBath"] +
                                              data5["HalfBath"] +
                                              data5["KitchenAbvGr"])

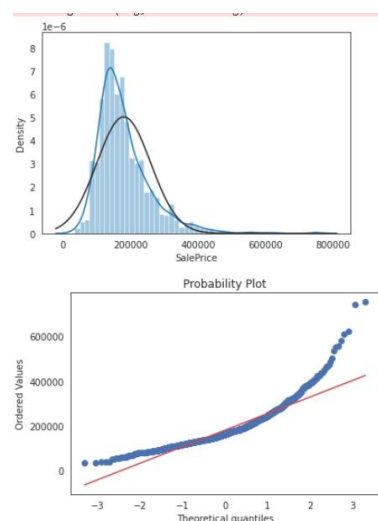
data5['Total_Home_Quality'] = data4['OverallQual'] + data5['OverallCond']

data5['Total_Bathrooms'] = (data5['FullBath'] + (0.5 * data5['HalfBath']) +
                           data5['BsmtFullBath'] + (0.5 * data5['BsmtHalfBath']))

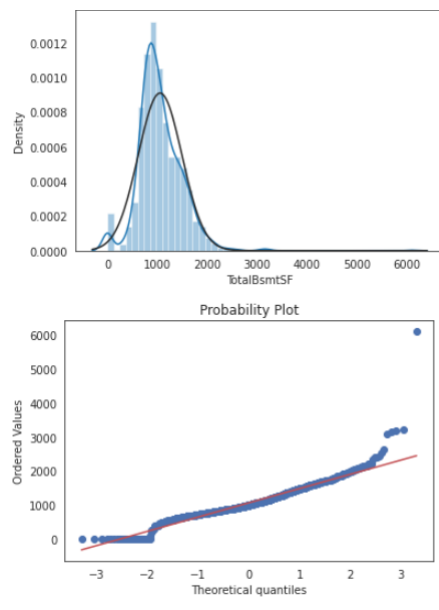
data5["HighQualSF"] = data5["1stFlrSF"] + data5["2ndFlrSF"]

```

Now, before we conduct our predictive analytics, we plotted the Sale Price and the histogram plot below reflects a right-skewed distribution, indicating we would have to take a logarithmic function of Sale Price.



Most /of the variables that deal with the actual physical space of the apartment are skewed, which makes sense, as people tend to live in smaller homes/apartments apart from the extremely wealthy. For example, the numeric feature "total BsmtSF"



so in order to make a model more comfortable to predict we will use the log transform for the skewed features to make them normally distributed, we will use a threshold of 0.8 and to check all the features we will make a table with a skewed column as in the image bellow.

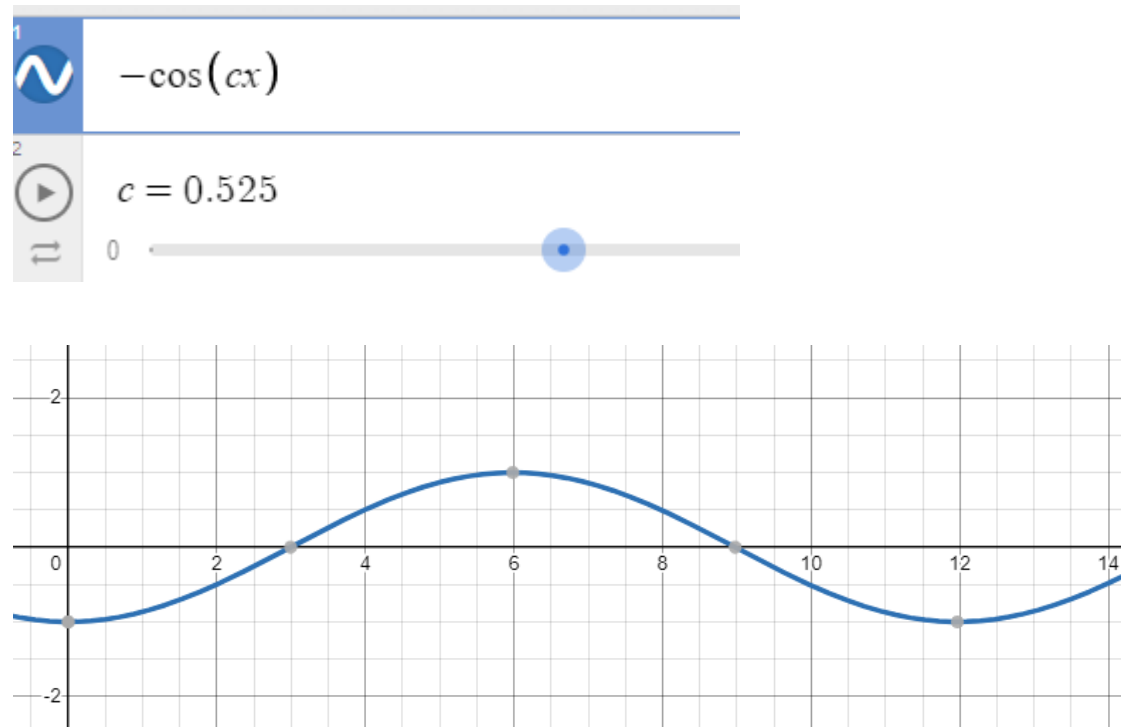
	Feature	Skew	Skewed
0	LotFrontage	1.340751	True
1	LotArea	12.822431	True
2	OverallQual	0.197110	False
3	OverallCond	0.570312	False
4	YearBuilt	-0.599806	False
5	YearRemodAdd	-0.451020	False
6	MasVnrArea	2.603682	True
7	BsmtFinSF1	1.425516	True
8	BsmtFinSF2	4.146111	True
9	BsmtUnfSF	0.919322	True
10	TotalBsmtSF	1.162806	True
11	1stFlrSF	1.469604	True
12	2ndFlrSF	0.861675	True
13	LowQualFinSF	12.088761	True
14	GrLivArea	1.269358	True
15	BsmtFullBath	0.624373	False

Because the $\log(x)$ isn't defined at zero and some of these features are close or zero we will use $\log(1+x)$ transform.

```
for column in skew_df.query("Skewed == True")['Feature'].values:
    data5[column] = np.log1p(data5[column])
```

Next is the cosine transform – we use this for cycle feature. months are a cycle because 1 and 12 are the most far but 12 is close to 1 .

So, we will define a -cos function that translates the ordinal feature into a cyclic one. I.e., 12 and 1 are similar because they are close.



```
data5['MoSold'] = (-np.cos(0.525 * data5['MoSold']))
```

Then we want to ensure that all the features are in the right type - MSSubClass, which "identifies the type of dwelling involved in the sale", is encoded as numeric but is a categorical variable.

```
data5['MSSubClass'] = data5['MSSubClass'].astype(str)
```

Our final part is to encode all the categorical feature, we will use pandas get dummies for one hot encoding and apply it on our dataset.

```
data5 = pd.get_dummies(data5)
data5
```

Our next step is scaling. Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data, we will use scaling and particularly python standard scaler and we will fit the data.

```
scaler = StandardScaler()
scaler.fit(data6)

data6 = pd.DataFrame(scaler.transform(data6), index=data6.index, columns=data6.columns)
```

Now we finished cleaning the data so we will split the train and test data to the original but transformed ones (because we can't use the test in our predictions!).

```
train_f = data6.loc[:house_train.index.max(), :].copy()
test_f = data6.loc[house_train.index.max() + 1:, :].reset_index(drop=True).copy()
```

Results-

Time to build some models! And to evaluate the best we used 10 folds cross validation. We began by creating some benchmarks using a Linear Regression model on the scaled data set. We then prepared a series of fits using three regularized linear regression models. The models we fit were:

- A naive Ridge Regression with $\alpha=1$.

```
In [29]: kf = KFold(n_splits=10)
result = np.exp(np.sqrt(-cross_val_score(Ridge(alpha=1.0), train_f, lg_target, scoring='neg_mean_squared_error', cv=kf)))
result
```

- An ensemble method using random forest, ridge and SVM calculated for each one the rmse on the 10 folds cross validations and choose the weights according to rmse rate.

```
-----
svm
1.1358741490529998
-----
random forest
1.1549027314378488
-----
ridge
1.1458220592157082
```

```
In [39]: final_prediction = (
    0.5 * np.exp(models['svm'].predict(test_f)) +
    0.2 * np.exp(models['random forest'].predict(test_f)) +
    0.3 * np.exp(models['ridge'].predict(test_f))
)
```

- An ensemble method with feature engineering using the 3 models and custom weights.

```

"""
for name, result in results.items():
    print("-----\n" + name)
    print(np.mean(result))

-----
svm
1.135242926780856
-----
random forest
1.1526396854752794
-----
ridge
1.1463386890882548
"""

>
final_prediction = (
    0.5 * np.exp(models['svm'].predict(test_f)) +
    0.2 * np.exp(models['random forest'].predict(test_f)) +
    0.3 * np.exp(models['ridge'].predict(test_f))
)

```

- An ensemble Bayesian Ridge Regression, random forest, ridge and SVM with hyperparameter optimization, and compared their rmse for choosing weights

```

for name, model in models.items():
    model.fit(train_f, lg_target)
    print(name + " trained.")

```

```

svm trained.
random forest trained.
ridge trained.
bayesian_ridge trained.

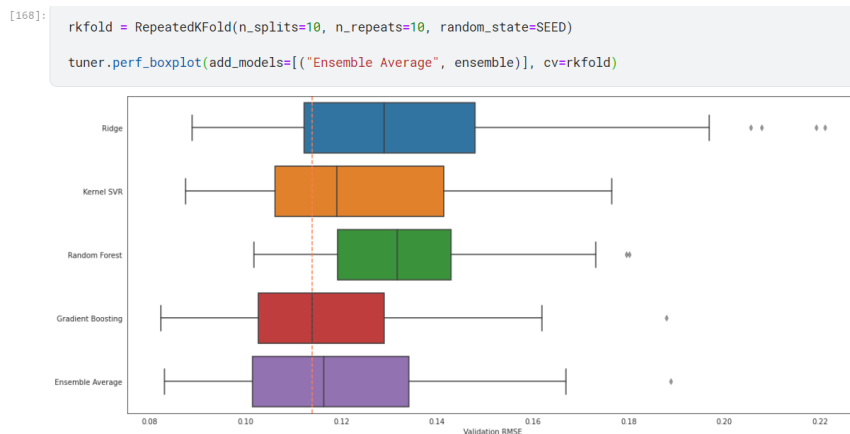
```

```

final_prediction = (
    0.5 * np.exp(models['svm'].predict(test_f)) +
    0.2 * np.exp(models['random forest'].predict(test_f)) +
    0.3 * np.exp(models['ridge'].predict(test_f)) +
    0.2 * np.exp(models['bayesian_ridge'].predict(test_f))
)

```

- An hyperparameter optimization using Ridge, Random Forest, Kernel SVM, Gradient Boosting, and choosing the prediction with the ensemble average.



- Linear regression

Linear Regression

```
In [34]: # Create linear regression object
         LR = LinearRegression()

         # Train the model using the training sets
         LR.fit(train_f,target)

Out[34]: LinearRegression()

In [35]: submit = pd.DataFrame({'Id': test_ids, 'SalePrice': LR.predict(test_f)})
         submit.to_csv('submission.csv', index=False)
```

For hyperparameter optimization we used optuna library, an open source hyperparameter optimization framework to automate hyperparameter search. Also, we used the grid search for the last model to check if it improves accuracy.

All the results presented in the table below.

model	Rmse
Ridge regression	0.14380
using ensembling method with regression + random forest + SVM	0.13358
using ensembling method with regression + random forest + SVM+ feature engineering	0.133150
using ensembling method with regression + random forest + SVM+ feature engineering +hyperparameter optimization	0.13315
An hyperparameter optimization using Ridge, Random Forest, Kernel SVM, Gradient Boosting	0.12626
Linear regression	1.98824

Discussion

Before we dive into the results and their meaning, we can see some obvious correlations between some features to the house price. we can see that *'SalePrice' and 'GrLivArea' have a linear relationship*, also we can see that *'TotalBsmntSF' and 'SalePrice' have an exponential reaction*. Now let's move the categorial features: we can see that *'SalePrice' is highly related to 'OverallQual' which makes a perfect sense together with the 'SalePrice' and 'YearBuilt' strong relation the people tend to pay more for new and more quality houses then old ones*.

Now that we take into considerations all the relationship, we will analyze the results.

The most accurate model was a model with an hyperparameter optimization using Ridge, Random Forest, Kernel SVM, Gradient Boosting, and it received a rmse of 0.12626.

The second place was still an hyperparameter optimization and using ensembling method with regression + random forest + SVM+ feature engineering. It we received a rmse of 0.1335, that is a drastic decrease of 0.659% accuracy.

The next place was without hyperparameter optimization and without feature engineering it was a model using ensembling method with regression + random forest + SVM. And because we didn't do those actions that can help the model, we got a rmse of 0.13358 which is a decrease of 0.008% accuracy from the previous one.

The last ones were the Ridge regression and the linear regression the ridge obtained a rmse 0.1438 which is far as 1.689% accuracy of our best try. which you can see the importance of working with parallel models together rather than only one.

The worst was linear regression, it's the simplest model and because we worked only with him and no feature engineering we got rmse of 1.98824 which is far 186% from our best one.

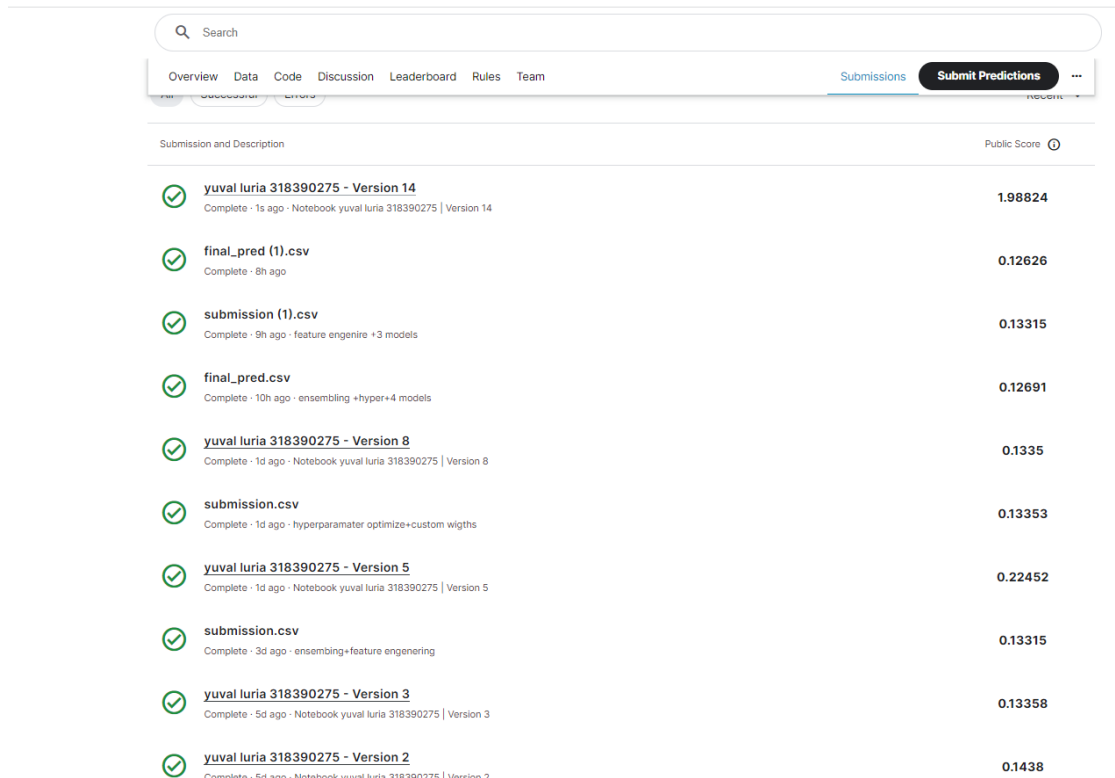
Regularized models perform well on this dataset. A note on the bias/variance tradeoff: According to the Gauss-Markov theorem, the model fit by the Ordinary Least Squares (OLS) is the least biased estimator of all possible estimators. In other words, it fits the data it has seen better than all possible models.

It does not necessarily perform well, however, against data that it has not seen. A regularized model penalizes model complexity by limiting the size of the betas. The effect of this is that the model introduces more bias than the OLS model but becomes more statistically stable and invariant. In other words, it prevents us from overfitting and is better able to generalize to new data.











To conclude, it was an interesting experiment and I think I learned a lot and for sure will continue exploring Kaggle and handling datasets.
Vered, thank you so much it been a very interesting and useful course.

Appendix

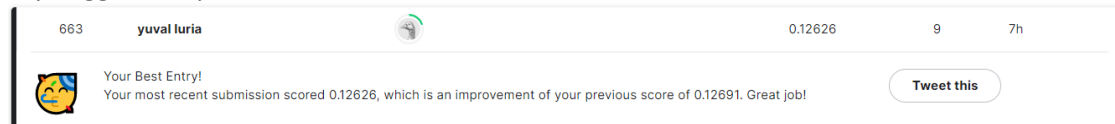
1. Pictures of all my submissions



The screenshot shows the Kaggle submission history for user 'yuval luria'. The table lists 10 submissions, each with a status icon (green checkmark), a title, a description, and a public score. The submissions are ordered by score, with the highest score at the top.

Submission and Description	Public Score
 yuval luria 318390275 - Version 14 Complete · 1s ago · Notebook yuval luria 318390275 Version 14	1.98824
 final_pred (1).csv Complete · 8h ago	0.12626
 submission (1).csv Complete · 9h ago · feature enginire +3 models	0.13315
 final_pred.csv Complete · 10h ago · ensembling +hyper+4 models	0.12691
 yuval luria 318390275 - Version 8 Complete · 1d ago · Notebook yuval luria 318390275 Version 8	0.1335
 submission.csv Complete · 1d ago · hyperparameter optimize+custom wlgths	0.13353
 yuval luria 318390275 - Version 5 Complete · 1d ago · Notebook yuval luria 318390275 Version 5	0.22452
 submission.csv Complete · 3d ago · ensembling+feature engineering	0.13315
 yuval luria 318390275 - Version 3 Complete · 5d ago · Notebook yuval luria 318390275 Version 3	0.13358
 yuval luria 318390275 - Version 2 Complete · 5d ago · Notebook yuval luria 318390275 Version 2	0.1438

2. My Kaggle best place



The screenshot shows a notification from Kaggle titled 'Your Best Entry!'. It states that the user's most recent submission scored 0.12626, which is an improvement over their previous score of 0.12691. The notification includes a 'Tweet this' button.

Rank	Username	Score	Rank	Time
663	yuval luria	0.12626	9	7h

3. Link to my Kaggle profile

<https://www.kaggle.com/yuval luria>

References

1. <https://www.kaggle.com/code/kylegraupe/testing-42-regressors-no-cheating>
2. <https://www.kaggle.com/code/racksonsit/house-price-prediction-regression>
3. <https://optuna.org/>
4. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
5. https://www.analyticsvidhya.com/blog/2021/05/bayesian-optimization-bayes_opt-or-hyperopt/