Housing Price Prediction

Lu Pang

Dataset

79 Features/Attributes

Target variable: Sales Price (y)

- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- ……
- MoSold: Month Sold
- YrSold: Year Sold
- SaleType: Type of sale

Dataset

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	2008	WD	Normal	250000
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	...	2009	WD	Normal	143000
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	...	2007	WD	Normal	307000
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	...	2009	WD	Normal	200000
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	...	2008	WD	Abnorml	129900
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	...	2008	WD	Normal	118000

df.head(10)

Target variable(y)

Data Cleaning

```
Id                259
MSSubClass          0
MSZoning            0
LotFrontage       259
LotArea             0
                 ...
MoSold              0
YrSold              0
SaleType            0
SaleCondition       0
SalePrice           0
Length: 81, dtype: int64
```

	const	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	SaleType_ConLw	SaleType_Ne
0	1.0	60	65.000000	8450	7	5	2003	2003	196.0	706	...	0	
1	1.0	20	80.000000	9600	6	8	1976	1976	0.0	978	...	0	
2	1.0	60	68.000000	11250	7	5	2001	2002	162.0	486	...	0	
3	1.0	70	60.000000	9550	7	5	1915	1970	0.0	216	...	0	
4	1.0	60	84.000000	14260	8	5	2000	2000	350.0	655	...	0	
5	1.0	50	85.000000	14115	5	5	1993	1995	0.0	732	...	0	
6	1.0	20	75.000000	10084	8	5	2004	2005	186.0	1369	...	0	
7	1.0	60	70.049958	10382	7	6	1973	1973	240.0	859	...	0	
8	1.0	50	51.000000	6120	7	5	1931	1950	0.0	0	...	0	
9	1.0	190	50.000000	7420	5	6	1939	1950	0.0	851	...	0	

```
# true/false for valid/missing data

pd.isna(df)

#create dummy variables for the categorical feature

x=pd.get_dummies(x)


#filling NaN's with the mean of the column

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

MODEL 01
Numpy Linear Regression

Model 01 - Numpy Linear Regression

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.933
Model:                            OLS   Adj. R-squared:                  0.919
Method:                 Least Squares   F-statistic:                     66.67
Date:                Wed, 09 Dec 2020   Prob (F-statistic):               0.00
Time:                        21:00:28   Log-Likelihood:                -16568.
No. Observations:                1460   AIC:                         3.364e+04
Df Residuals:                    1206   BIC:                         3.499e+04
Df Model:                         253
Covariance Type:            nonrobust
==============================================================================

==============================================================================
Omnibus:                      400.601   Durbin-Watson:                   1.917
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            14582.979
Skew:                           0.563   Prob(JB):                         0.00
Kurtosis:                      18.442   Cond. No.                     4.80e+17
==============================================================================
```

```
results = sm.OLS(y,x).fit()
print(results.summary())
```

Model 01 – Result Analysis

Adj. R-squared: 0.919

The modified version of R-squared which is adjusted for the number of variables in the regression. Here, 91.9% variation in y is explained by dependent variables (x1, x2, x3,...).

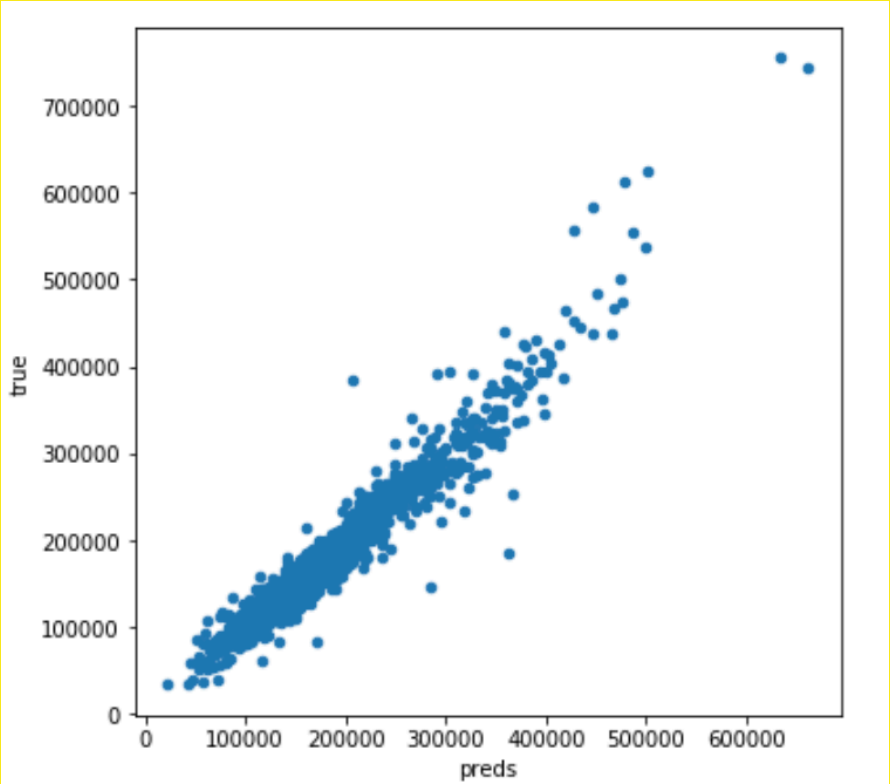
Prob(F-Statistic): 0.00

As per the above results, probability is close to zero. This implies that overall the regressions is meaningful.

Model 01 – Numpy Linear Regression

Adjusted R^2: 0.919
RMSE of the prediction: 20515

```
def rmse_accuracy_percentage(actual,predicted):

    print("RMSE is:",np.round(np.sqrt(sum(((np.array(actual)-np.array(predicted))**2))/len(actual)),0))
```



Model 01 – Result Analysis

	coef	std err	t	P>|t|	[0.025	0.975]
LotArea	0.7065	0.109	6.462	0.000	0.492	0.921
OverallQual	6813.0564	1012.800	6.727	0.000	4826.011	8800.102
OverallCond	5796.7223	870.860	6.656	0.000	4088.154	7505.291
YearBuilt	319.6074	76.938	4.154	0.000	168.661	470.554
MasVnrArea	20.7223	5.782	3.584	0.000	9.378	32.067
BsmtFinSF1	15.8251	2.890	5.475	0.000	10.154	21.496
TotalBsmtSF	22.7876	4.059	5.614	0.000	14.823	30.752
1stFlrSF	18.6926	6.378	2.931	0.003	6.179	31.206
2ndFlrSF	36.5205	5.461	6.688	0.000	25.807	47.234
LowQualFinSF	-29.5607	13.905	-2.126	0.034	-56.841	-2.280
GrLivArea	25.6525	5.696	4.504	0.000	14.477	36.828
BedroomAbvGr	-3660.8139	1363.663	-2.685	0.007	-6336.228	-985.399
ScreenPorch	35.8189	12.487	2.869	0.004	11.321	60.317
PoolArea	685.3756	226.508	3.026	0.003	240.981	1129.770

Area Square Feet

Model 01 – Result Analysis

Location

	coef	std err	t	P>|t|	[0.025	0.975]
Neighborhood_NoRidge	2.945e+04	8150.614	3.613	0.000	1.35e+04	4.54e+04
Neighborhood_NridgHt	2.201e+04	7627.262	2.886	0.004	7047.580	3.7e+04
Neighborhood_StoneBr	4.322e+04	8622.654	5.013	0.000	2.63e+04	6.01e+04
Condition2_PosN	-1.902e+05	2.72e+04	-6.990	0.000	-2.44e+05	-1.37e+05
RoofMatl_ClyTile	-5.24e+05	5.16e+04	-10.150	0.000	-6.25e+05	-4.23e+05
RoofMatl_CompShg	5.146e+04	2.24e+04	2.294	0.022	7445.548	9.55e+04
RoofMatl_Membran	1.463e+05	3.33e+04	4.387	0.000	8.09e+04	2.12e+05
RoofMatl_Metal	1.144e+05	3.27e+04	3.500	0.000	5.03e+04	1.79e+05
RoofMatl_WdShngl	1.061e+05	2.42e+04	4.383	0.000	5.86e+04	1.54e+05
GarageQual_Ex	9.772e+04	2.44e+04	4.008	0.000	4.99e+04	1.46e+05
GarageQual_Fa	-2.731e+04	7886.261	-3.463	0.001	-4.28e+04	-1.18e+04
GarageQual_TA	-2.13e+04	7614.008	-2.797	0.005	-3.62e+04	-6359.745
GarageCond_Ex	-9.473e+04	2.83e+04	-3.349	0.001	-1.5e+05	-3.92e+04
PoolQC_Fa	-4.132e+05	1.49e+05	-2.769	0.006	-7.06e+05	-1.2e+05
PoolQC_Gd	-3.832e+05	1.47e+05	-2.611	0.009	-6.71e+05	-9.52e+04

?

Quality

Model 01 – Result Analysis

#**1 OLS Regression Assumption - the errors are normally distributed.**

Prob(Omnibus) is supposed to be close to the 1 in order for it to satisfy the OLS assumption. In this case Prob(Omnibus) is 0.000, Prob(JB) is 5.56e-116 (0.000), which implies that the OLS assumption is not satisfied. Jarque-Bera (JB) is 530.769, which indicates that the errors are not normally distributed.

#**2 OLS Regression Assumption - homoscedasticity**

Durbin-watson: this implies that the variance of errors is constant. A value between 1 to 2 is preferred. Here, it is 1.256 implying that the regression results are reliable from the interpretation side of this metric.

MODEL 02
Support Vector Machine

Model 02 -SVM

R^2 of the prediction: 0.227
RMSE of the prediction: 72896

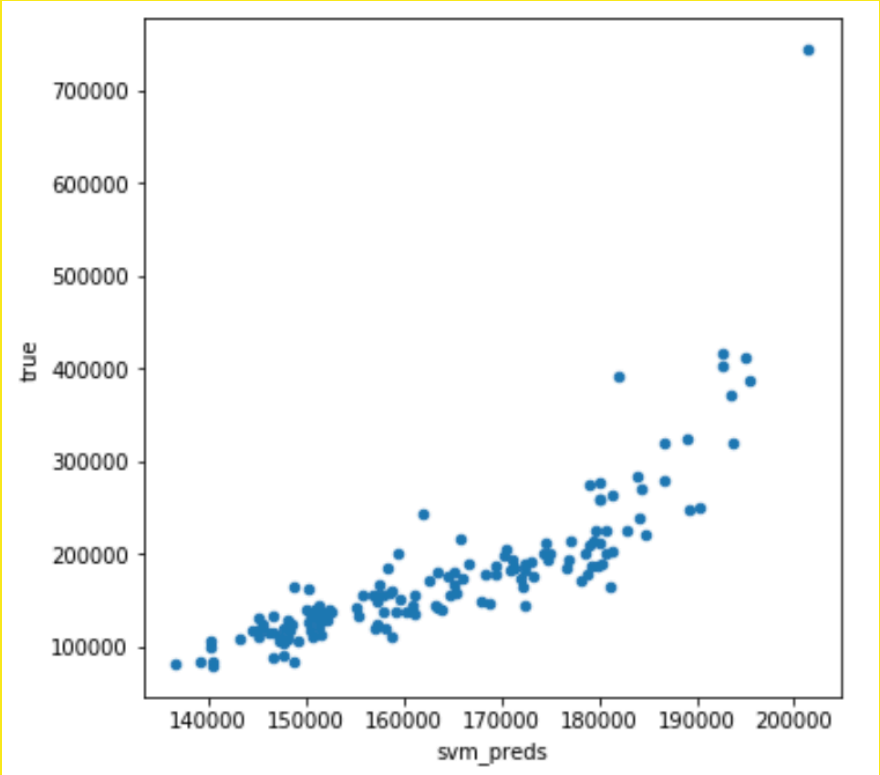
```
from sklearn.svm import SVR
regr = SVR(kernel = 'linear', C=1.0, epsilon=0.1)
regr.fit(x_train, y_train)
y_pred = regr.predict(x_test)
```

```
# calculate the R^2 produced by the SVR model

r2_score(y_test, y_pred)

# calculate the rmse

mse=mean_squared_error(y_test, y_pred)

rmse=np.sqrt(mse)
```

Model 02 -SVM

R^2 of the prediction: 0.227
RMSE of the prediction: 72896

```
svm_preds =pd.DataFrame({"svm_preds":svm_y_pred, "true":y_test})

svm_preds.plot(x = "svm_preds", y = "true",kind = "scatter")
```



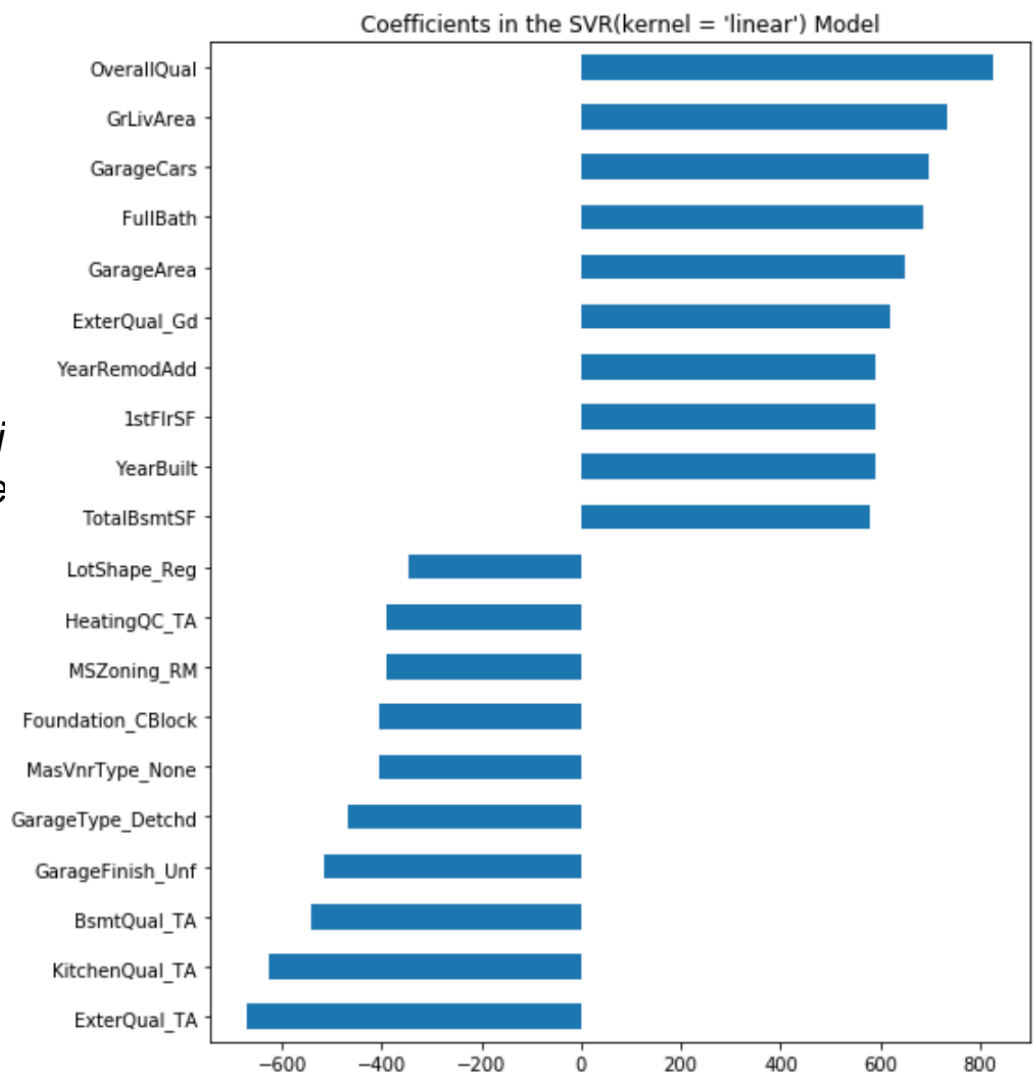
Model 02 -SVM

Coef_:*ndarray of shape (1, n_features)*

Weights assigned to the features (coeffici the primal problem). This is only available case of a linear kernel.

sklearn.svm.SVR

```
coef=regr.coef_
imp=pd.concat([coef.sort_values().head(10),
coef.sort_values().tail(10)])
```



MODEL 03
Random Forest

Model 03- Random Forest Regressor

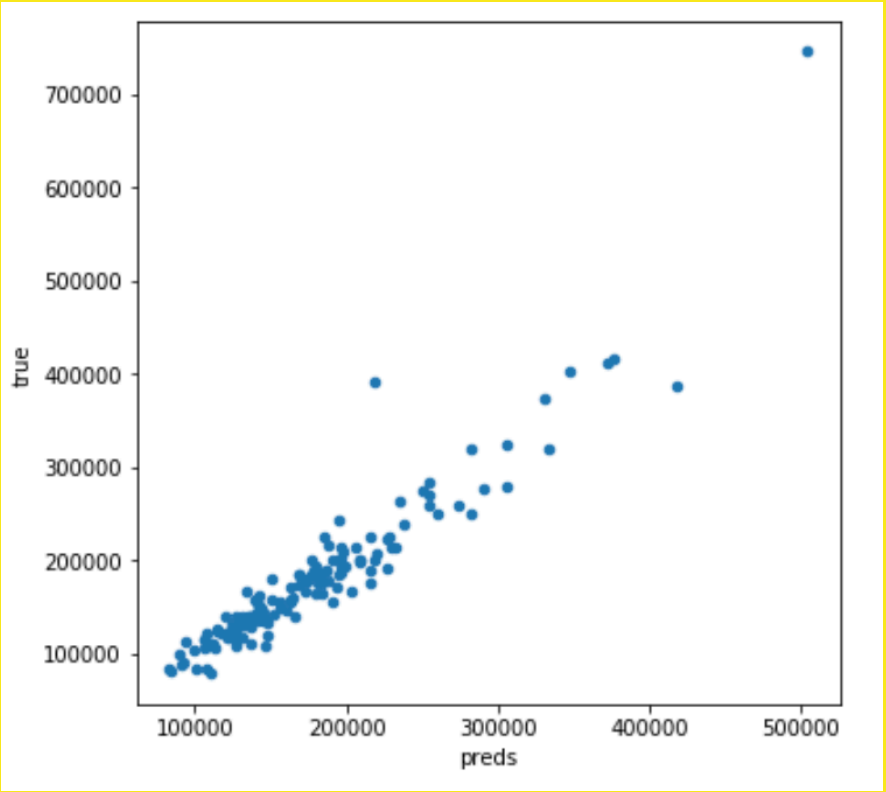
R^2 of the prediction: 0.868
RMSE of the prediction: 30073

```
# Fitting Regressor to the Training set
from sklearn.ensemble import RandomForestRegressor

Model=RandomForestRegressor(random_state = 1)

# fit model
model.fit(x_train, y_train)

# Predicting the Test set results
y_pred = model.predict(x_test)
```

Model 03-
Random Forest Regressor

R^2 of the prediction: 0.868
RMSE of the prediction: 30073

```
rf_preds = pd.DataFrame({"preds":rf_y_pred, "true":y_test})

rf_preds.plot(x = "preds", y = "true",kind = "scatter")
```



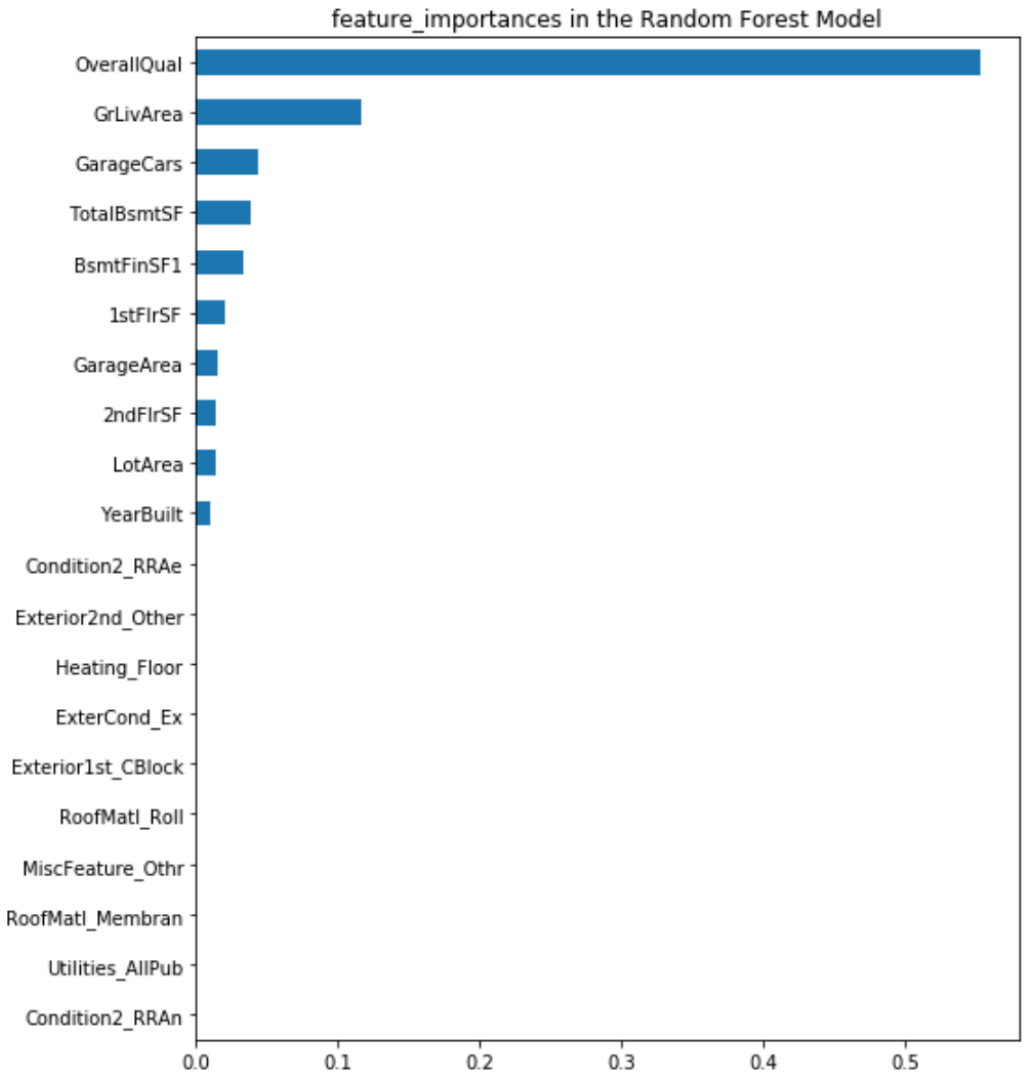
Model 03-
Random Forest Regressor

feature_importances_

The impurity-based feature importances.

The higher, the more important the feature. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance.

sklearn.ensemble.RandomForestRegressor



MODEL 04
Decision Tree

Model 04- Decision Tree Regressor

R^2 of the prediction: 0.824
RMSE of the prediction: 34746

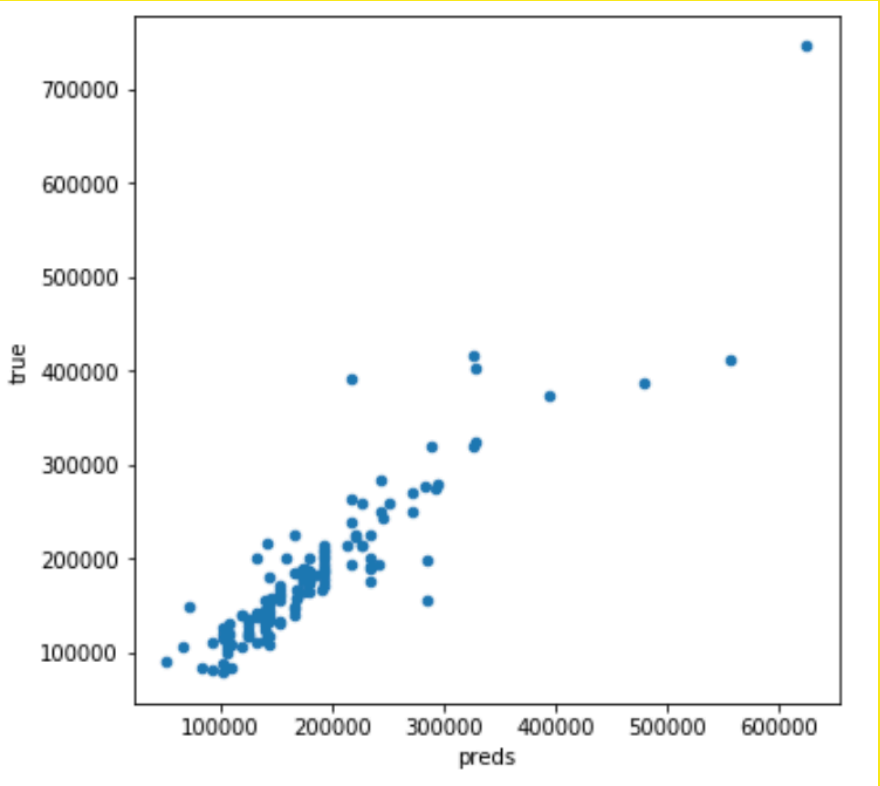
```
# Fitting Regressor to the Training set
from sklearn.tree import
DecisionTreeRegressor

Model=DecisionTreeRegressor(random
_state =1,max_leaf_nodes=100)

# fit model
model.fit(x_train, y_train)

# Predicting the Test set results
y_pred = model.predict(x_test)
```

Model 04-
Decision Tree Regressor

R^2 of the prediction: 0.824
RMSE of the prediction: 34746

```
dt_preds = pd.DataFrame({"preds":dt_y_pred, "true":y_test})
```

```
dt_preds.plot(x = "preds", y = "true",kind = "scatter")
```



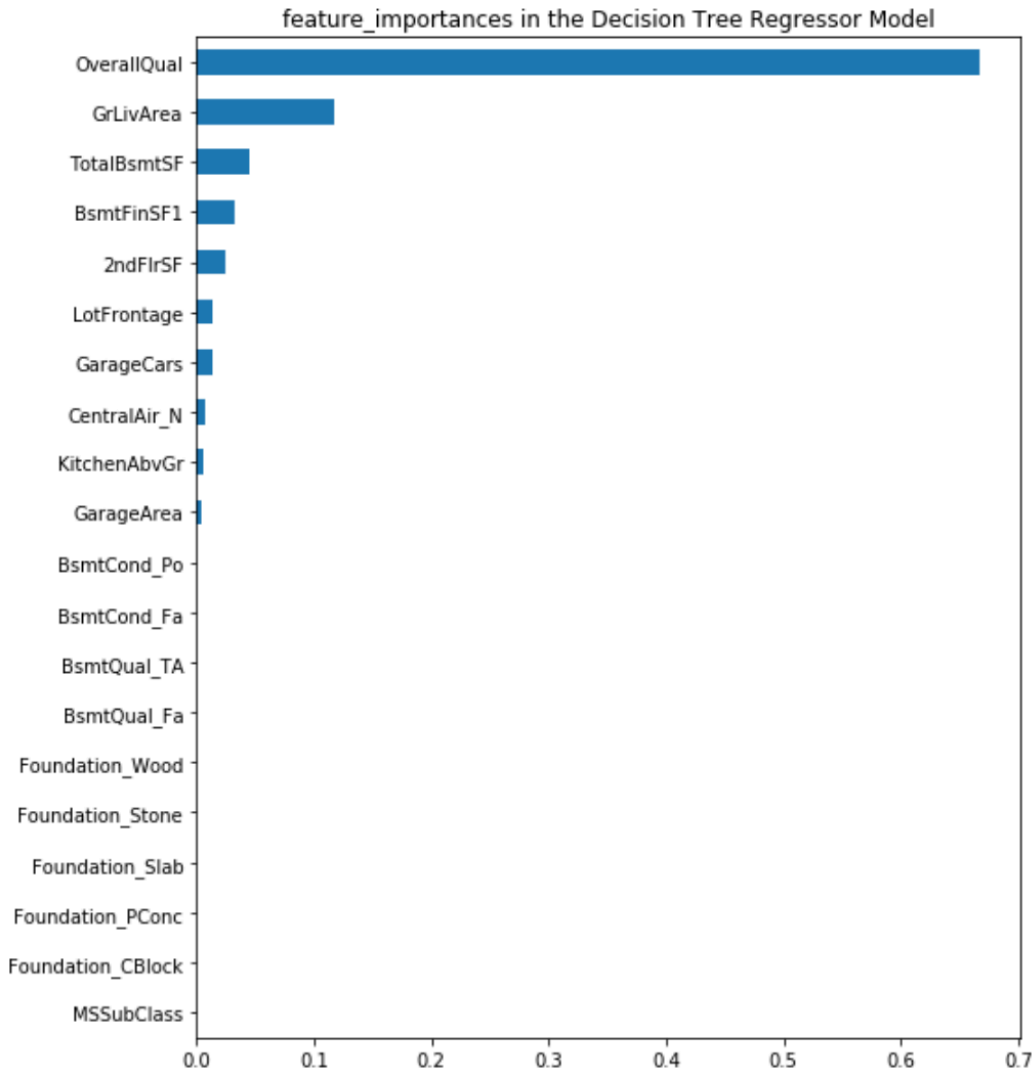
Model 04-
Decision Tree Regressor

feature_importances_:

ndarray of shape (n_features,)

Normalized total reduction of criteria by feature (Gini importance).

sklearn.tree.DecisionTreeRegressor



compare the results of the different algorithms

Comparisons-Error Analysis

Model 01-
Numpy Linear Regression

- Adjusted R^2: 0.919
- RMSE: 20515

Model 02-
Support Vector Regression

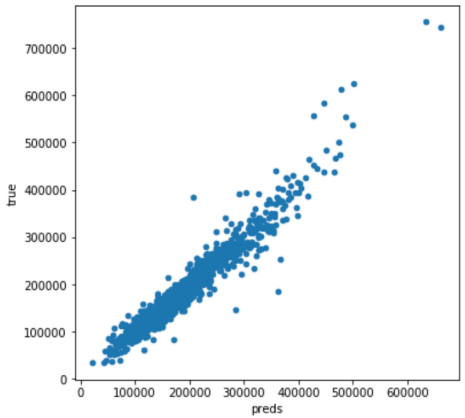
- R^2: 0.227
- RMSE: 72896

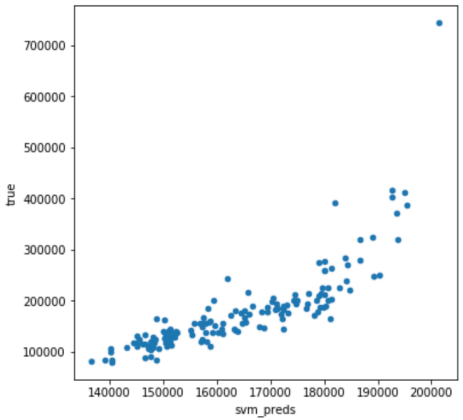
Model 03-
Random Forest Regressor

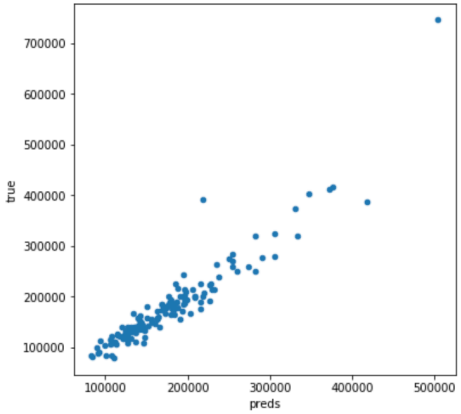
- R^2: 0.868
- RMSE: 30073

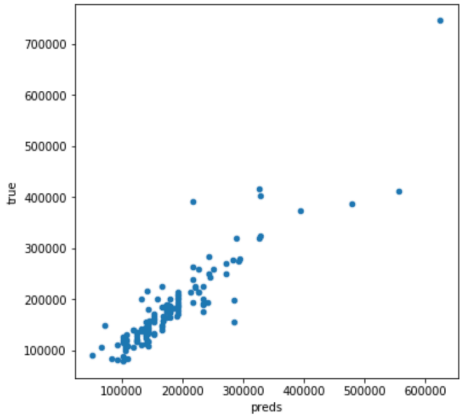
Model 04-
Decision Tree Regressor

- R^2: 0.824
- RMSE: 34746









X axis=predicted values, Y axis=actual values

Comparisons-Error Analysis

Model 01-
Numpy Linear Regression

- Adjusted R^2: 0.919
- RMSE: 20515

Model 02-
Support Vector Regression

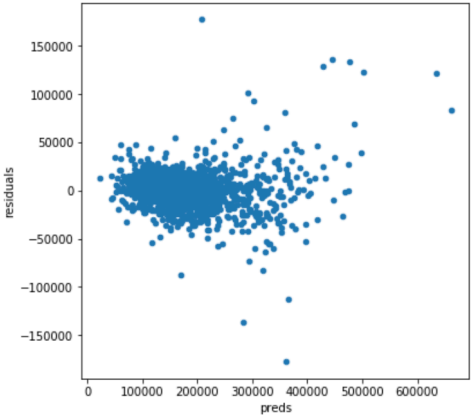
- R^2: 0.227
- RMSE: 72896

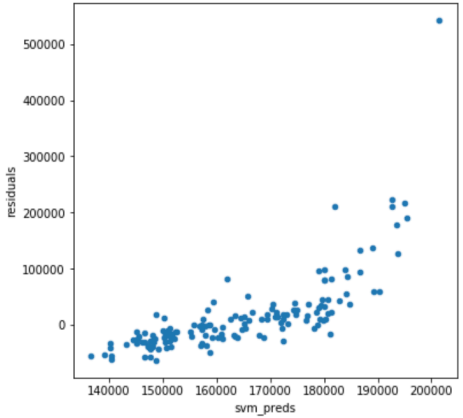
Model 03-
Random Forest Regressor

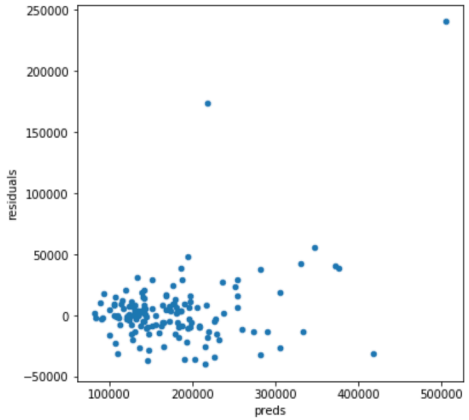
- R^2: 0.868
- RMSE: 30073

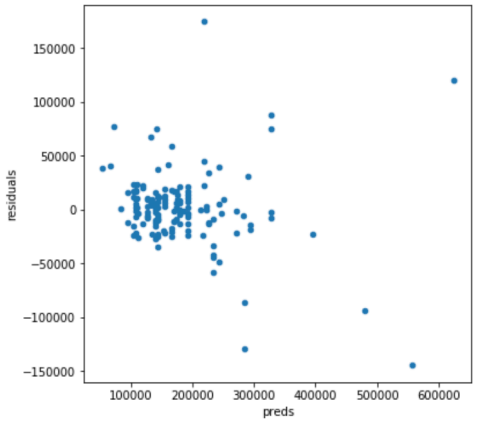
Model 04-
Decision Tree Regressor

- R^2: 0.824
- RMSE: 34746









X axis=predicted values, Y axis=residuals

Comparisons-Feature Importance

Model 01-NLR

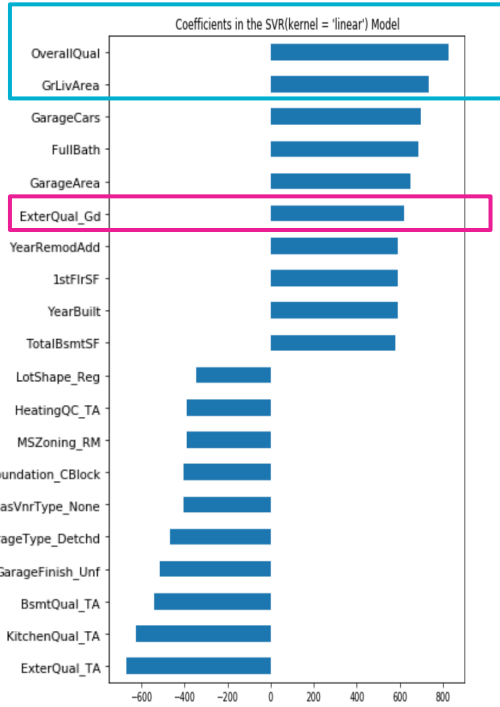
- Adjusted R^2: 0.919
- RMSE: 20515

LotArea
OverallQual
OverallCond
YearBuilt
MasVnrArea
BsmtFinSF1
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BedroomAbvGr
ScreenPorch
PoolArea

Neighborhood_NoRidge
Neighborhood_NridgHt
Neighborhood_StoneBr
Condition2_PosN
RoofMatl_ClyTile
RoofMatl_CompShg
RoofMatl_Membran
RoofMatl_Metal
RoofMatl_WdShngl
GarageQual_Ex
GarageQual_Fa
GarageQual_TA
GarageCond_Ex
PoolQC_Fa
PoolQC_Gd

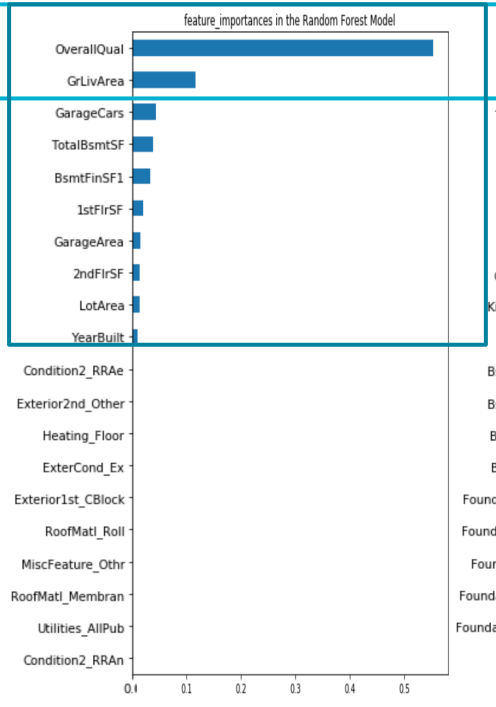
Model 02-SVR

- R^2: 0.227
- RMSE: 72896



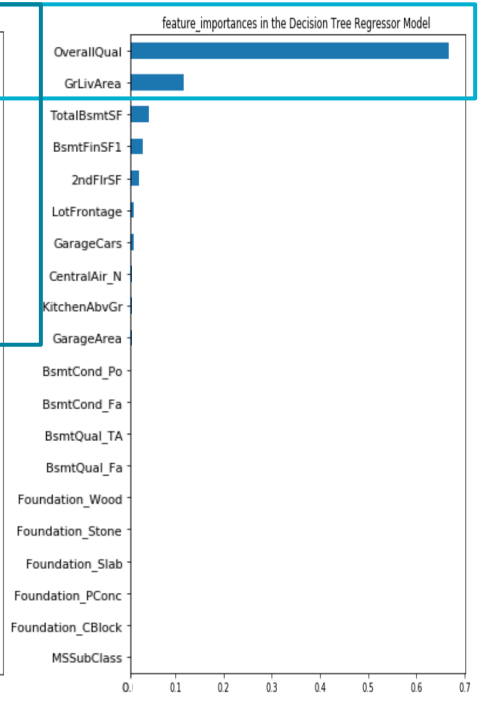
Model 03-RFR

- R^2: 0.868
- RMSE: 30073



Model 04-DTR

- R^2: 0.824
- RMSE: 34746



Conclusions

- Random Forest Regressor works better than Support Vector Regression (SVR) and Decision Tree Regressor.
- Numpy Linear Regression has the highest R^2 and the lowest RMSE (Root-Mean-Square Error)
- I would suggest both Model 1 (NLR) and Model 3 (RFR) to conduct the housing price prediction.

Formulars

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

$$\text{R-Square} = 1 - \frac{\sum(Y\_actual - Y\_predicted)^2}{\sum(Y\_actual - Y\_mean)^2}$$

Questions?

Lu Pang