PREDICTING IOWA HOUSE PRICES USING SUPERVISED MACHINE LEARNING ALGORITHMS

Team “Flat Outliers”:
Daniel Park, Dimitri Liakhovitski, Gwen Fernandez, & Henry Crosby

NY Data Science Bootcamp, November 2017

PRESENTATION FLOW

- Project background & objectives
- Out team's journey
- First approach to feature engineering – before Zeyu's "Kaggle workflow" lecture
- Identifying & eliminating outliers
- Second approach to feature engineering – after Zeyu's 'Kaggle workflow' lecture
- Prediction
- Lessons learned

PROJECT BACKGROUND & OBJECTIVES

KAGGLE COMPETITION “HOUSE PRICES: ADVANCED REGRESSION TECHNIQUES”

- Data from Kaggle Competition: https://www.kaggle.com/c/house-prices-advanced-regression-techniques
- Kaggle data has ~80 columns that capture different features of 1461 Ames, Iowa homes sold between 2006 and 2010, as well as their final sale prices.
- Objective: Develop a model that accurately predicts house sale prices based on house features.

OUR TEAM’S JOURNEY

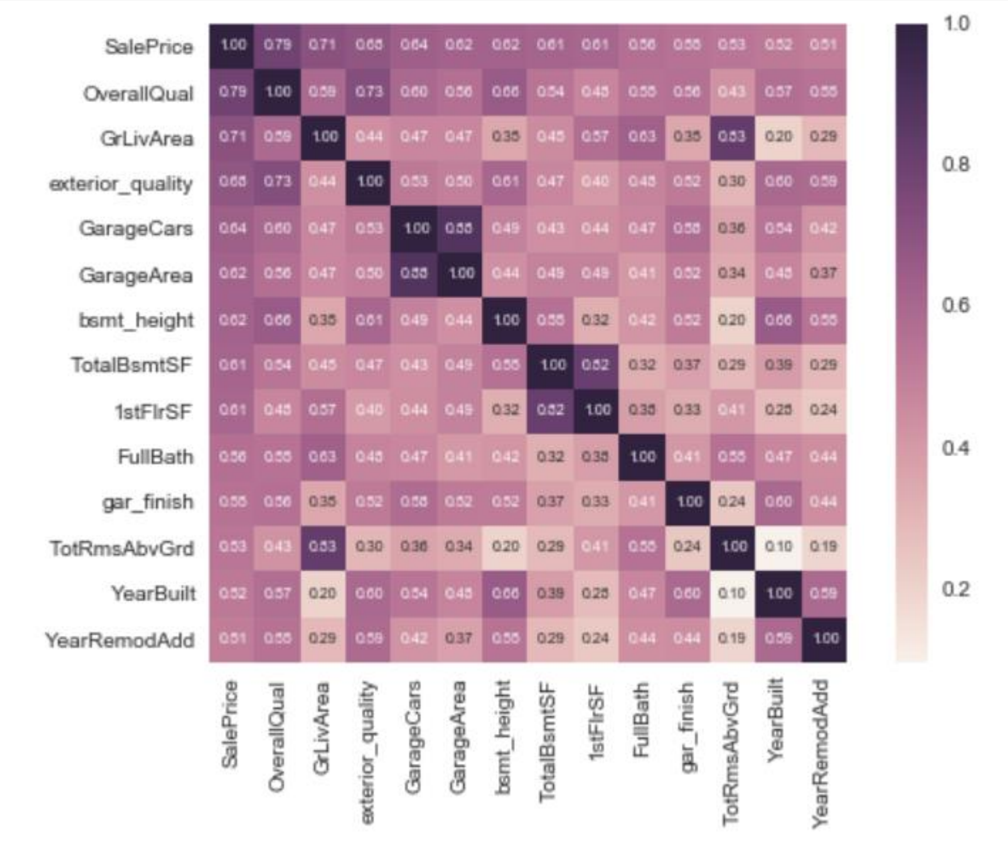
"FLAT OUTLIERS" WORKFLOW

1. Understanding and exploring the data;
2. Sharing files on git; Google Hangouts for communication/meetings;
3. Brainstorming and first round of feature engineering;
4. Modeling;
5. Building model ensembles;
6. Second round of feature engineering – a different approach; shorter turnaround;
7. Building ensemble again.

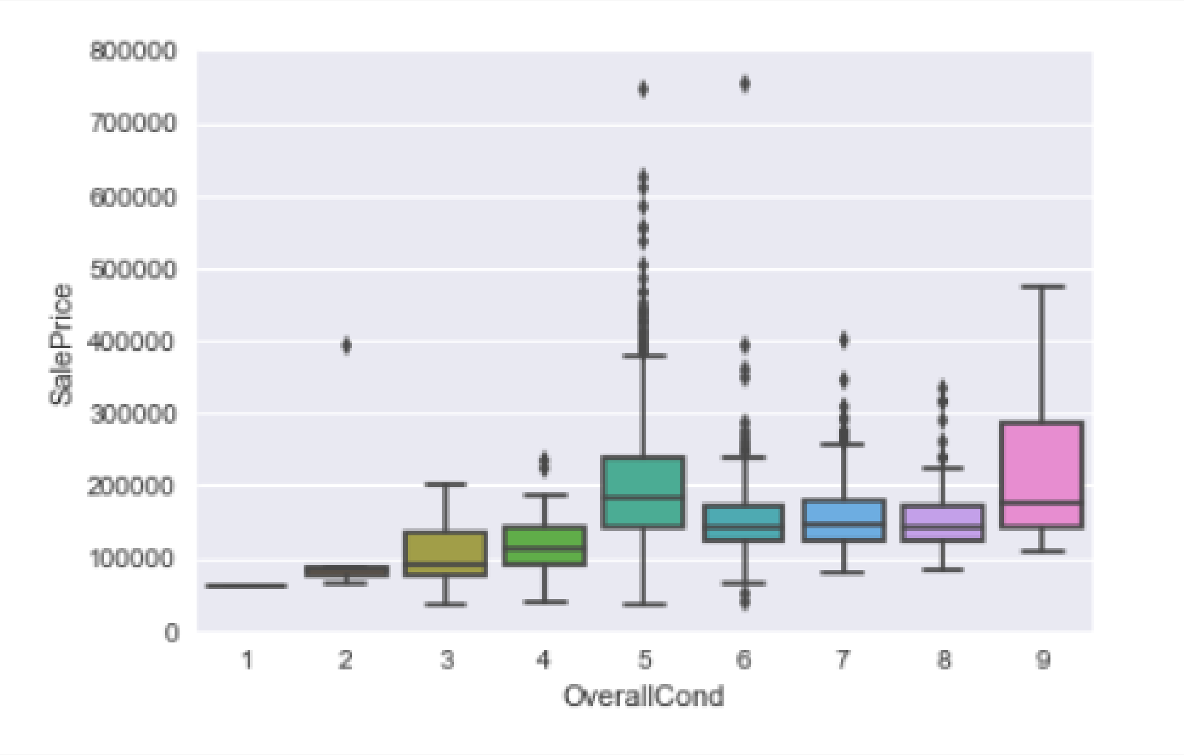
GITHUB – PUSH IT!

- To challenge ourselves, we agreed to use a less familiar language: Python
- After initial trepidation and emailing of files, we moved to Github:
  - The repository was on Gwen's git, but we all contributed
- We had a few hiccups and workarounds, but we stayed the course and found it useful
- By the end, we were all singing: https://www.youtube.com/watch?v=vCadcBR95oU

INITIAL DATA EXPLORATION EXAMPLES

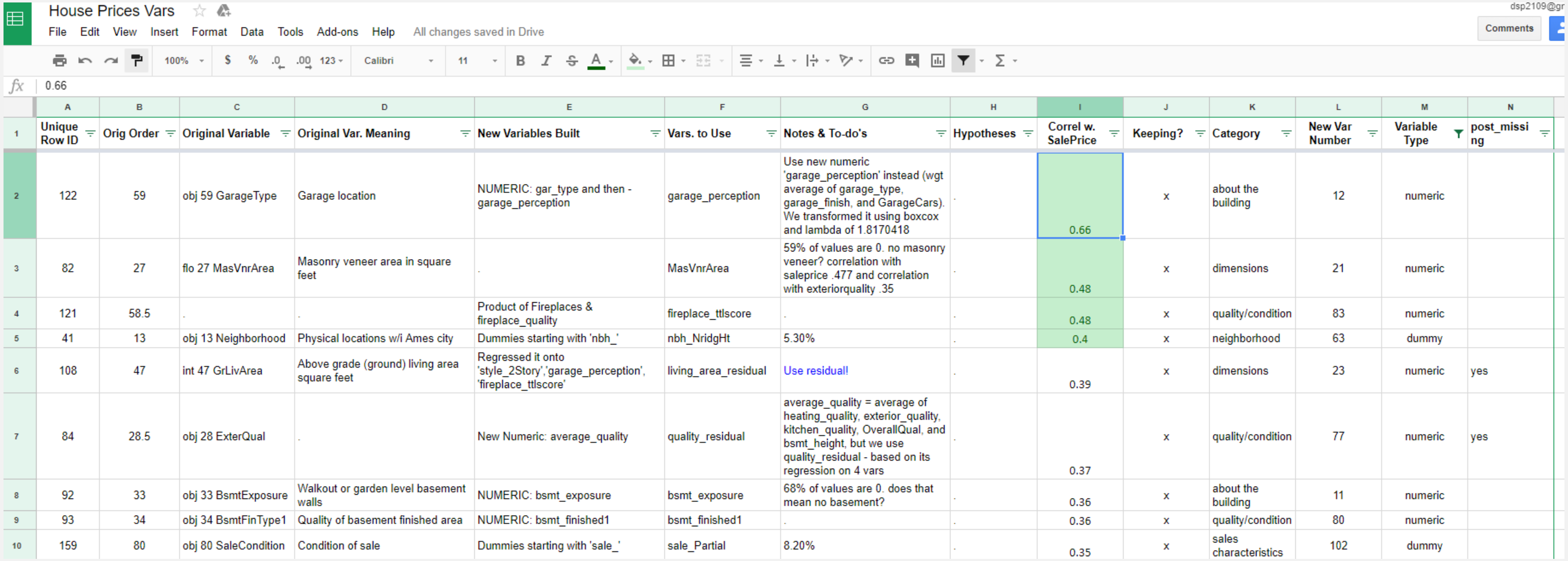


	SalePrice	OverallQual	GrLivArea	exterior_quality	GarageCars	GarageArea	bsmt_height	TotalBsmtSF	1stFlrSF	FullBath	gar_finish	TotRmsAbvGrd	YearBuilt	YearRemodAdd
SalePrice	1.00	0.79	0.71	0.68	0.64	0.62	0.62	0.61	0.61	0.56	0.55	0.53	0.52	0.51
OverallQual	0.79	1.00	0.59	0.73	0.60	0.56	0.66	0.54	0.48	0.55	0.56	0.43	0.57	0.55
GrLivArea	0.71	0.59	1.00	0.44	0.47	0.47	0.35	0.45	0.57	0.63	0.35	0.83	0.20	0.29
exterior_quality	0.68	0.73	0.44	1.00	0.53	0.50	0.61	0.47	0.40	0.48	0.52	0.30	0.60	0.59
GarageCars	0.64	0.60	0.47	0.53	1.00	0.88	0.49	0.43	0.44	0.47	0.58	0.36	0.54	0.42
GarageArea	0.62	0.56	0.47	0.50	0.88	1.00	0.44	0.49	0.49	0.41	0.52	0.34	0.48	0.37
bsmt_height	0.62	0.66	0.35	0.61	0.49	0.44	1.00	0.55	0.32	0.42	0.52	0.20	0.66	0.55
TotalBsmtSF	0.61	0.54	0.45	0.47	0.43	0.49	0.55	1.00	0.82	0.32	0.37	0.29	0.39	0.29
1stFlrSF	0.61	0.48	0.57	0.40	0.44	0.49	0.32	0.82	1.00	0.38	0.33	0.41	0.28	0.24
FullBath	0.56	0.55	0.63	0.48	0.47	0.41	0.42	0.32	0.38	1.00	0.41	0.55	0.47	0.44
gar_finish	0.55	0.56	0.35	0.52	0.58	0.52	0.52	0.37	0.33	0.41	1.00	0.24	0.60	0.44
TotRmsAbvGrd	0.53	0.43	0.83	0.30	0.36	0.34	0.20	0.29	0.41	0.55	0.24	1.00	0.10	0.19
YearBuilt	0.52	0.57	0.20	0.60	0.54	0.48	0.66	0.39	0.28	0.47	0.60	0.10	1.00	0.59
YearRemodAdd	0.51	0.55	0.29	0.59	0.42	0.37	0.55	0.29	0.24	0.44	0.44	0.19	0.59	1.00



BRAINSTORMING AND FEATURE ENGINEERING

- Came together to discuss findings
- Logged all observations and taxonomy of variables in a shared google sheet


House Prices Vars
File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive
dsp2109@gr
Comments
100%
$ % .0 .00 123
Calibri
11
B I S A
fx 0.66

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Unique Row ID	Orig Order	Original Variable	Original Var. Meaning	New Variables Built	Vars. to Use	Notes & To-do's	Hypotheses	Correl w. SalePrice	Keeping?	Category	New Var Number	Variable Type	post_missing
2	122	59	obj 59 GarageType	Garage location	NUMERIC: gar_type and then - garage_perception	garage_perception	Use new numeric 'garage_perception' instead (wgt average of garage_type, garage_finish, and GarageCars). We transformed it using boxcox and lambda of 1.8170418	.	0.66	x	about the building	12	numeric	
3	82	27	flo 27 MasVnrArea	Masonry veneer area in square feet	.	MasVnrArea	59% of values are 0. no masonry veneer? correlation with saleprice .477 and correlation with exteriorquality .35	.	0.48	x	dimensions	21	numeric	
4	121	58.5	.	.	Product of Fireplaces & fireplace_quality	fireplace_ttlscore	.	.	0.48	x	quality/condition	83	numeric	
5	41	13	obj 13 Neighborhood	Physical locations w/i Ames city	Dummies starting with 'nbh_'	nbh_NridgHt	5.30%	.	0.4	x	neighborhood	63	dummy	
6	108	47	int 47 GrLivArea	Above grade (ground) living area square feet	Regressed it onto 'style_2Story','garage_perception', 'fireplace_ttlscore'	living_area_residual	Use residual!	.	0.39	x	dimensions	23	numeric	yes
7	84	28.5	obj 28 ExterQual	.	New Numeric: average_quality	quality_residual	average_quality = average of heating_quality, exterior_quality, kitchen_quality, OverallQual, and bsmt_height, but we use quality_residual - based on its regression on 4 vars	.	0.37	x	quality/condition	77	numeric	yes
8	92	33	obj 33 BsmtExposure	Walkout or garden level basement walls	NUMERIC: bsmt_exposure	bsmt_exposure	68% of values are 0. does that mean no basement?	.	0.36	x	about the building	11	numeric	
9	93	34	obj 34 BsmtFinType1	Quality of basement finished area	NUMERIC: bsmt_finished1	bsmt_finished1	.	.	0.36	x	quality/condition	80	numeric	
10	159	80	obj 80 SaleCondition	Condition of sale	Dummies starting with 'sale_'	sale_Partial	8.20%	.	0.35	x	sales characteristics	102	dummy	

ONE WEEK LATER… DATA

- After many feature engineering struggles, we finalized our first data set and split up the models:
  - Linear – Daniel
  - Support Vector Machines and other – Henry
  - Random Forest – Gwen
  - Gradient Boosting Machines - Dimitri

ENSEMBLE LEARNING

- We came together to look at and learn from each others' models.
  - Great learning experience: learned from each other
- Averaged model predictions, stacked individual models with meta models.
- Pretty happy with our score on Kaggle Leader Board!

FEATURE ENGINEERING 2.0 & ENSEMBLES AGAIN

- 2nd round of feature engineering using LabelEncoding per Zeyu's lecture
- Much less focus on feature engineering, much faster, more predictors
- Ran it all again, with the new data – this time ignoring multi-collinearity
- Improved RMSE in most models

FIRST APPROACH TO FEATURE ENGINEERING – BEFORE ZEYU'S "KAGGLE FLOW" LECTURE

We focused on:

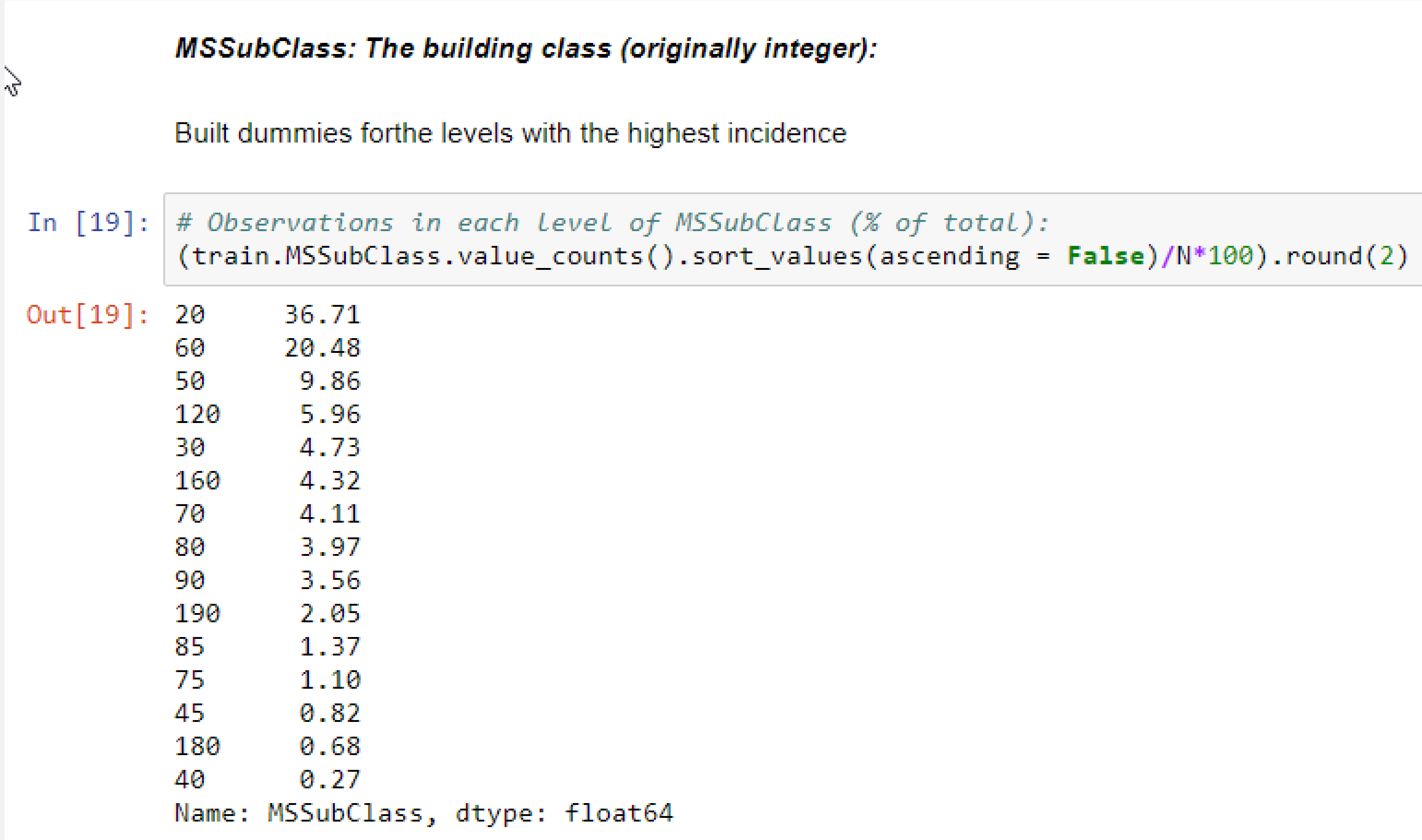
Predictor meaning, variance, redundancy & reducing mutlicollinearity

WE DECIDED TO OMIT VARIABLES THAT HAD VIRTUALLY NO VARIANCE

- Such as:
  - *Street*: Type of road access to property – as 99.6% of houses had paved road acces
  - *Utilities*: Type of utilities available – as 99.9% of houses had all public utilities
  - *RoofMatl*: Roof material – as 98.2% had ‘Standard (Composite) Shingles’
  - *Heating*: Type of heating – as 97.8% had gas heating

FOR EACH 'OBJECT' COLUMN WE LOOKED AT ITS MEANING AND LEVEL FREQUENCY

- Example with 'MSSubClass':



MSSubClass: The building class (originally integer):

Built dummies forthe levels with the highest incidence

In [19]:
```
# Observations in each level of MSSubClass (% of total):
(train.MSSubClass.value_counts().sort_values(ascending = False)/N*100).round(2)
```

Out[19]:
```
20     36.71
60     20.48
50      9.86
120     5.96
30      4.73
160     4.32
70      4.11
80      3.97
90      3.56
190     2.05
85      1.37
75      1.10
45      0.82
180     0.68
40      0.27
Name: MSSubClass, dtype: float64
```

WE BUILT DUMMIES FOR HIGHER INCIDENCE LEVELS: EITHER “MANUALLY” OR USING pd.get_dummies()

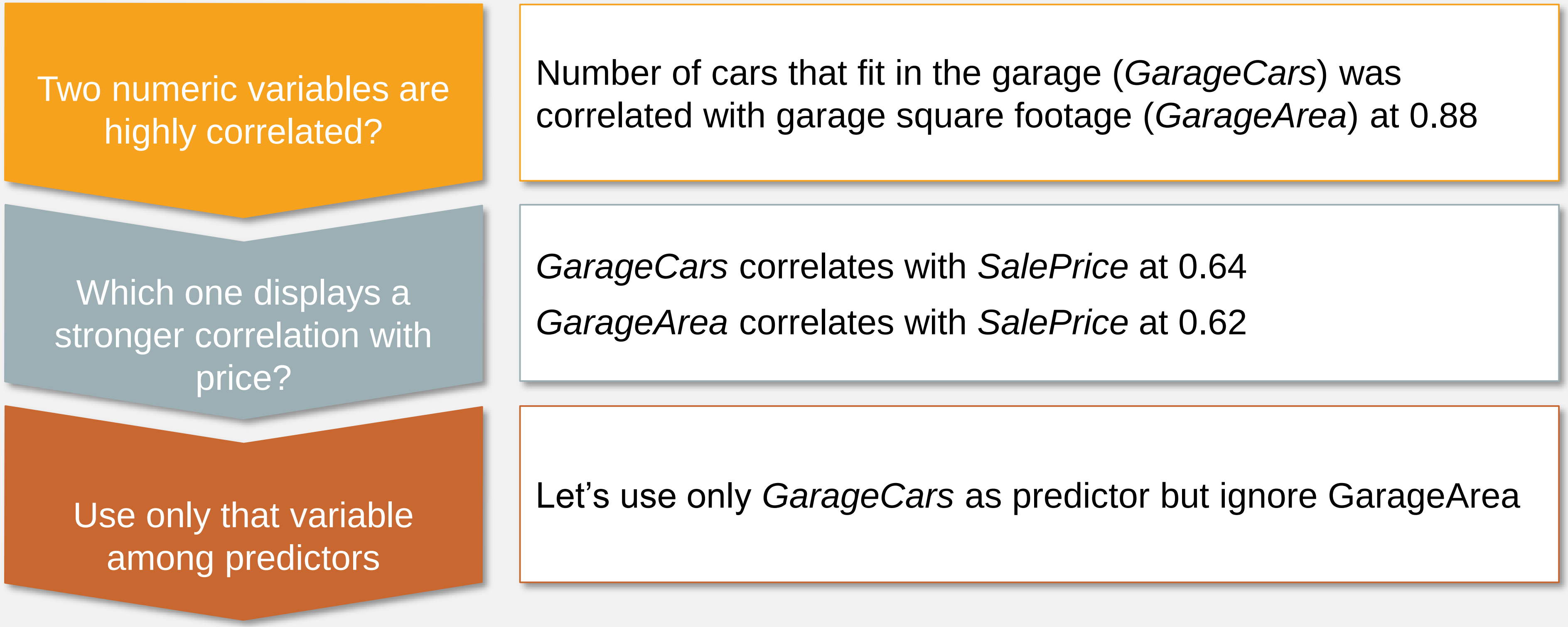
Two methods:

- “Manual” dummy calculation
- Using *pd.get_dummies()* and then dropping dummies with very low incidence

WE TRANSFORMED ‘OBJECT’ COLUMNS THAT HAD QUANTITATIVE MEANING – USING OUR JUDGEMENT

- Example of ‘theory driven’ transformation of Exterior Quality column:
  - Ex = 3
  - Gd = 2
  - TA = 1
  - Others = 0
- Example of ‘theory driven’ transformation of Garage Type column:
  - 2Types = 3
  - BuiltIn = 2.5
  - Basment = 2
  - Attchd = 2
  - Detchd = 1.5
  - CarPort = 1
  - No garage = 0

WE EXCLUDED SOME NUMERIC VARIABLES IF THEY WERE REDUNDANT STATISTICALLY & CONCEPTUALLY



Two numeric variables are highly correlated?

Number of cars that fit in the garage (*GarageCars*) was correlated with garage square footage (*GarageArea*) at 0.88

Which one displays a stronger correlation with price?

GarageCars correlates with *SalePrice* at 0.64

GarageArea correlates with *SalePrice* at 0.62

Use only that variable among predictors

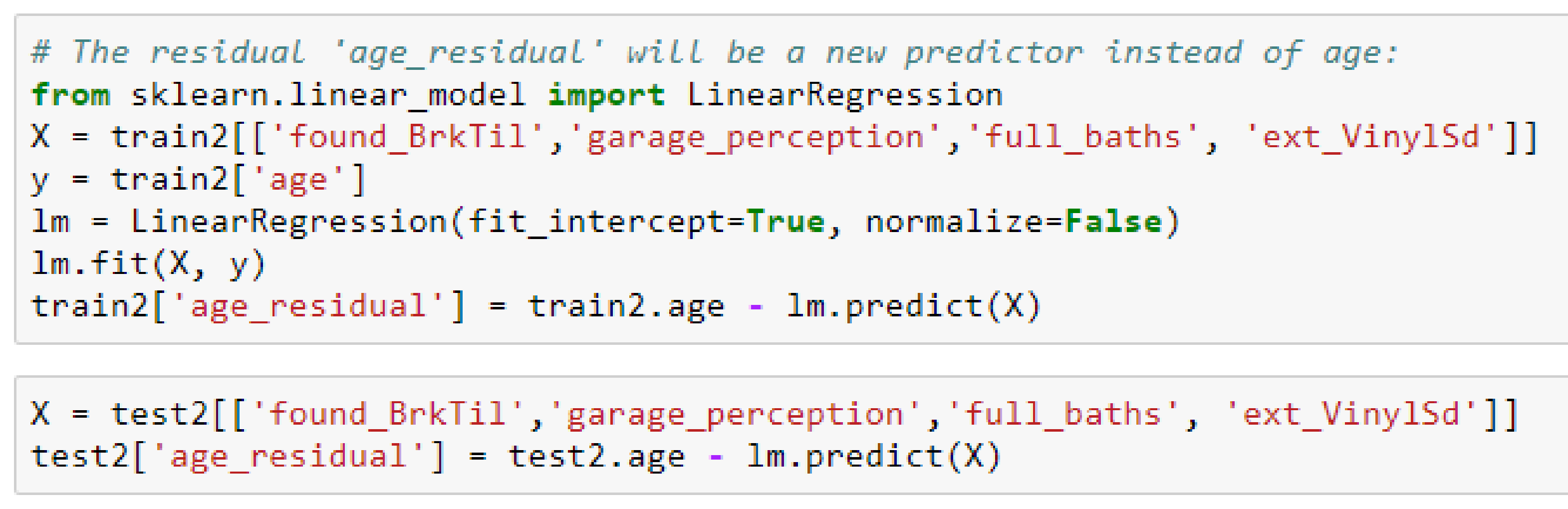
Let's use only *GarageCars* as predictor but ignore GarageArea

WE COMBINED SOME HIGHLY CORRELATED NUMERIC VARIABLES INTO ONE AND USED ONLY THE NEW ONE IN OUR MODELS

- For example, we combined into one new variable ‘*average_quality*’ the following quality-related variables that were highly correlated – and then omitted those 4 from our models.
  - 'exterior_quality',
  - 'heating_quality',
  - 'kitchen_quality',
  - 'OverallQual'

TO REDUCE MULTICOLLINEARITY, SEVERAL TIMES WE REGRESSED ONE VARIABLE ONTO SEVERAL OTHERS AND KEPT ONLY ITS RESIDUAL

- Example of regressing age of the house onto brick & tile foundation dummy, garage perception, number of full baths, and exterior vinyl siding dummy – because age was highly correlated with all those variables:



```
# The residual 'age_residual' will be a new predictor instead of age:
from sklearn.linear_model import LinearRegression
X = train2[['found_BrkTil','garage_perception','full_baths', 'ext_VinylSd']]
y = train2['age']
lm = LinearRegression(fit_intercept=True, normalize=False)
lm.fit(X, y)
train2['age_residual'] = train2.age - lm.predict(X)
```

```
X = test2[['found_BrkTil','garage_perception','full_baths', 'ext_VinylSd']]
test2['age_residual'] = test2.age - lm.predict(X)
```

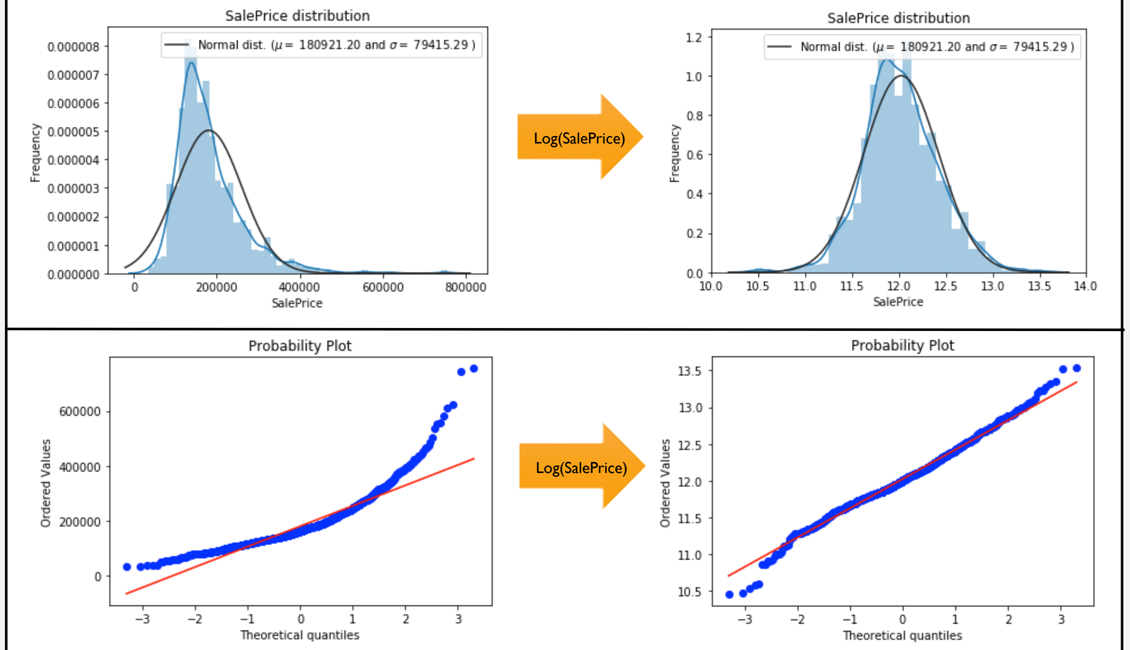
WE IMPUTED MISSING VALUES USING MICE FROM ‘fancyimpute’ PACKAGE

- ‘fanicyimpute’ package is a Python translation of R package ‘mice’ – the best library for missing data imputations.
- It is a bit tricky to install ‘fancyimpute’ on Windows.
- MICE stands for Multiple Imputation by Chained Equations.
- As imputation method, we used MICE.
- Details on MICE: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/

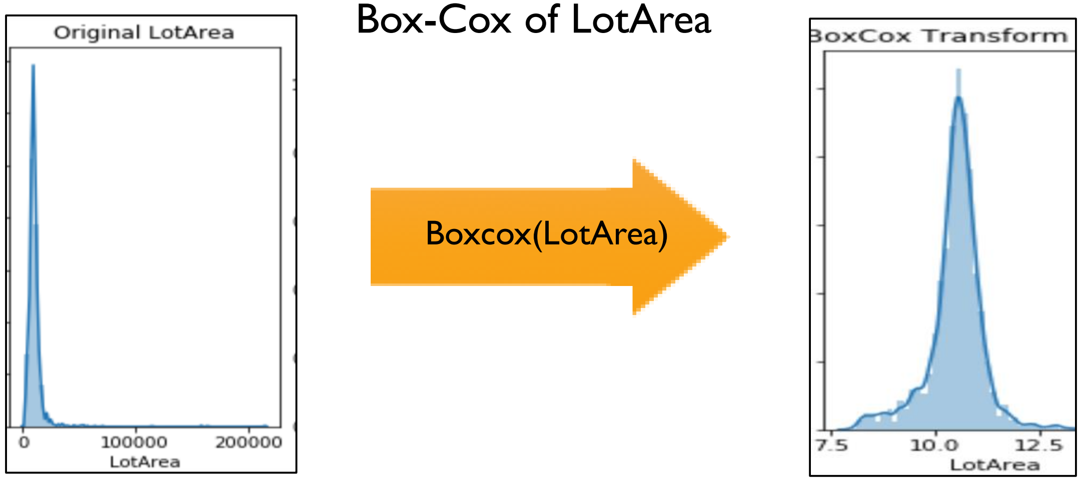
FINALLY, WE TRANSFORMED SOME PREDICTORS & DV AND SCALED (STANDARDIZED) ALL PREDICTORS

- We used the log of the DV SalePrice for modeling.
- We transformed several numeric predictors using Box-Cox transformation – to make them less skewed.
- We scaled (standardized) all predictors using StandardScaler.

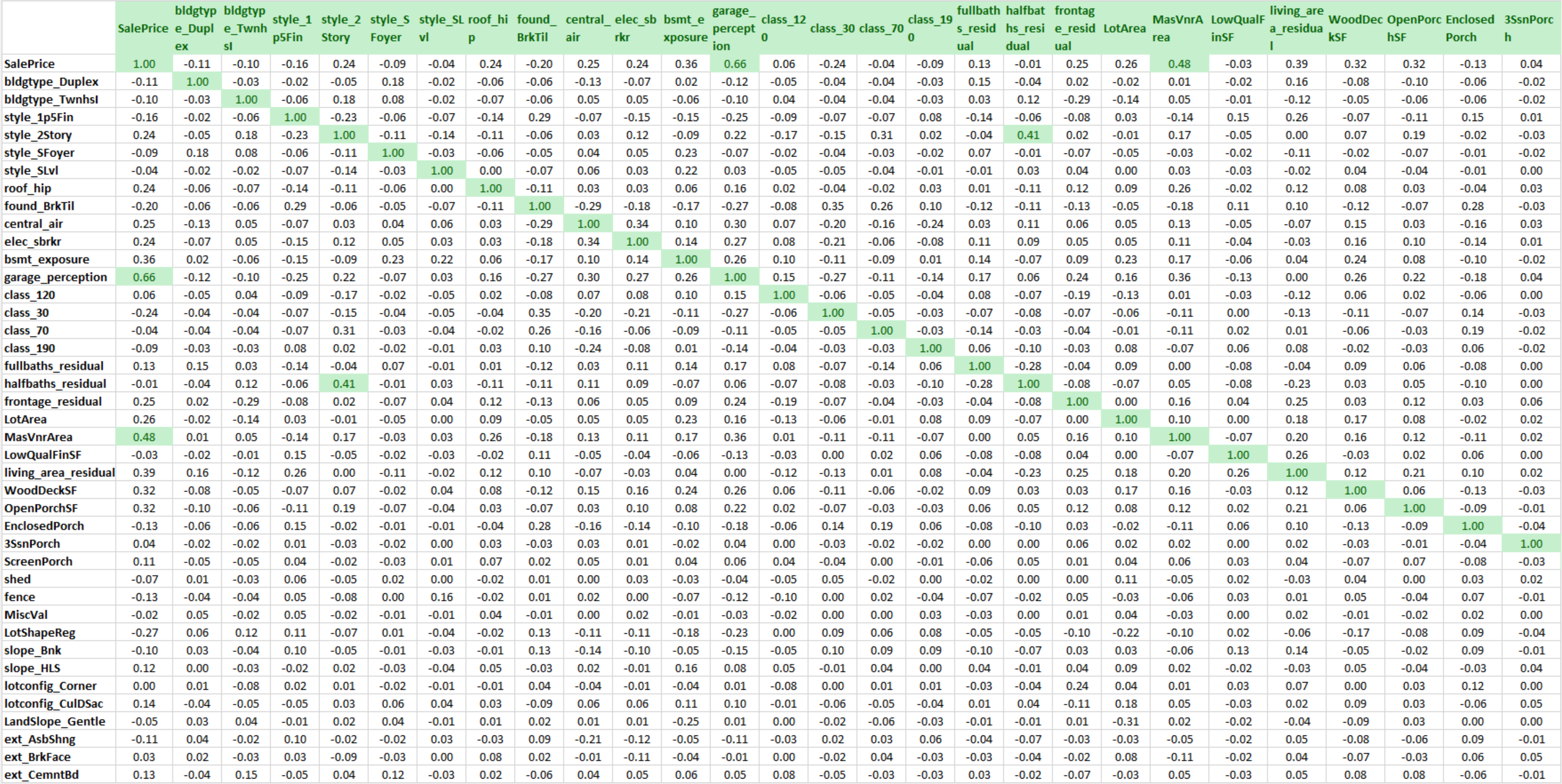
Log of SalePrice



Box-Cox of LotArea



WE KEPT UPDATING AND STUDYING THE CORRELATION MATRIX, STRIVING TO REDUCE PREDICTOR MULTICOLINEARITY

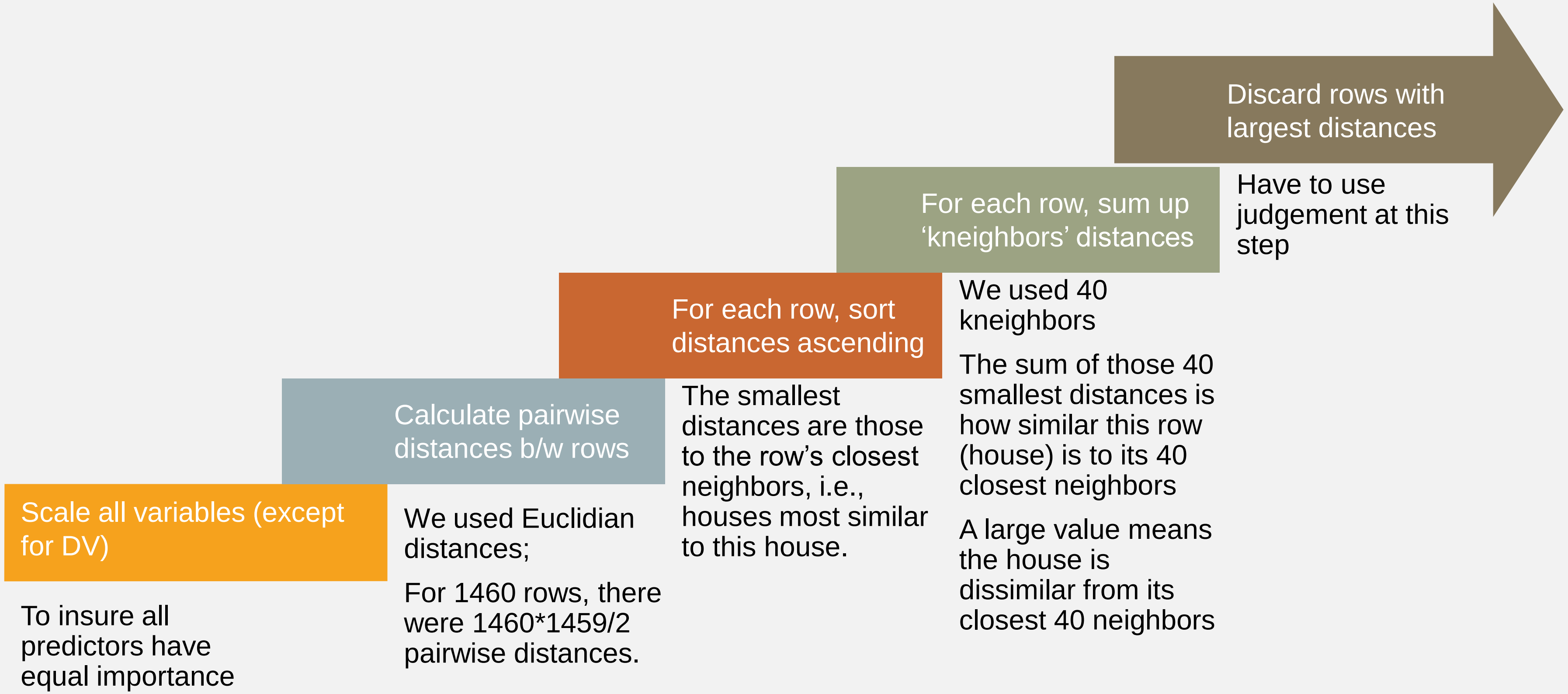


	SalePrice	bldgtype_Duplex	bldgtype_Twnhsl	style_1p5Fin	style_2Story	style_SFoyer	style_SLvl	roof_hip	found_BrkTil	central_air	elec_sbrkr	bsmt_exposure	garage_perception	class_120	class_30	class_70	class_190	fullbaths_residual	halfbaths_residual	frontage_residual	LotArea	MasVnrArea	LowQualFinSF	living_area_residual	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch
SalePrice	1.00	-0.11	-0.10	-0.16	0.24	-0.09	-0.04	0.24	-0.20	0.25	0.24	0.36	0.66	0.06	-0.24	-0.04	-0.09	0.13	-0.01	0.25	0.26	0.48	-0.03	0.39	0.32	0.32	-0.13	0.04
bldgtype_Duplex	-0.11	1.00	-0.03	-0.02	-0.05	0.18	-0.02	-0.06	-0.06	-0.13	-0.07	0.02	-0.12	-0.05	-0.04	-0.04	-0.03	0.15	-0.04	0.02	-0.02	0.01	-0.02	0.16	-0.08	-0.10	-0.06	-0.02
bldgtype_TwnhsI	-0.10	-0.03	1.00	-0.06	0.18	0.08	-0.02	-0.07	-0.06	0.05	0.05	-0.06	-0.10	0.04	-0.04	-0.04	-0.03	0.03	0.12	-0.29	-0.14	0.05	-0.01	-0.12	-0.05	-0.06	-0.06	-0.02
style_1p5Fin	-0.16	-0.02	-0.06	1.00	-0.23	-0.06	-0.07	-0.14	0.29	-0.07	-0.15	-0.15	-0.25	-0.09	-0.07	-0.07	0.08	-0.14	-0.06	-0.08	0.03	-0.14	0.15	0.26	-0.07	-0.11	0.15	0.01
style_2Story	0.24	-0.05	0.18	-0.23	1.00	-0.11	-0.14	-0.11	-0.06	0.03	0.12	-0.09	0.22	-0.17	-0.15	0.31	0.02	-0.04	0.41	0.02	-0.01	0.17	-0.05	0.00	0.07	0.19	-0.02	-0.03
style_SFoyer	-0.09	0.18	0.08	-0.06	-0.11	1.00	-0.03	-0.06	-0.05	0.04	0.05	0.23	-0.07	-0.02	-0.04	-0.03	-0.02	0.07	-0.01	-0.07	-0.05	-0.03	-0.02	-0.11	-0.02	-0.07	-0.01	-0.02
style_SLvl	-0.04	-0.02	-0.02	-0.07	-0.14	-0.03	1.00	0.00	-0.07	0.06	0.03	0.22	0.03	-0.05	-0.05	-0.04	-0.01	-0.01	0.03	0.04	0.00	0.03	-0.03	-0.02	0.04	-0.04	-0.01	0.00
roof_hip	0.24	-0.06	-0.07	-0.14	-0.11	-0.06	0.00	1.00	-0.11	0.03	0.03	0.06	0.16	0.02	-0.04	-0.02	0.03	0.01	-0.11	0.12	0.09	0.26	-0.02	0.12	0.08	0.03	-0.04	0.03
found_BrkTil	-0.20	-0.06	-0.06	0.29	-0.06	-0.05	-0.07	-0.11	1.00	-0.29	-0.18	-0.17	-0.27	-0.08	0.35	0.26	0.10	-0.12	-0.11	-0.13	-0.05	-0.18	0.11	0.10	-0.12	-0.07	0.28	-0.03
central_air	0.25	-0.13	0.05	-0.07	0.03	0.04	0.06	0.03	-0.29	1.00	0.34	0.10	0.30	0.07	-0.20	-0.16	-0.24	0.03	0.11	0.06	0.05	0.13	-0.05	-0.07	0.15	0.03	-0.16	0.03
elec_sbrkr	0.24	-0.07	0.05	-0.15	0.12	0.05	0.03	0.03	-0.18	0.34	1.00	0.14	0.27	0.08	-0.21	-0.06	-0.08	0.11	0.09	0.05	0.05	0.11	-0.04	-0.03	0.16	0.10	-0.14	0.01
bsmt_exposure	0.36	0.02	-0.06	-0.15	-0.09	0.23	0.22	0.06	-0.17	0.10	0.14	1.00	0.26	0.10	-0.11	-0.09	0.01	0.14	-0.07	0.09	0.23	0.17	-0.06	0.04	0.24	0.08	-0.10	-0.02
garage_perception	0.66	-0.12	-0.10	-0.25	0.22	-0.07	0.03	0.16	-0.27	0.30	0.27	0.26	1.00	0.15	-0.27	-0.11	-0.14	0.17	0.06	0.24	0.16	0.36	-0.13	0.00	0.26	0.22	-0.18	0.04
class_120	0.06	-0.05	0.04	-0.09	-0.17	-0.02	-0.05	0.02	-0.08	0.07	0.08	0.10	0.15	1.00	-0.06	-0.05	-0.04	0.08	-0.07	-0.19	-0.13	0.01	-0.03	-0.12	0.06	0.02	-0.06	0.00
class_30	-0.24	-0.04	-0.04	-0.07	-0.15	-0.04	-0.05	-0.04	0.35	-0.20	-0.21	-0.11	-0.27	-0.06	1.00	-0.05	-0.03	-0.07	-0.08	-0.07	-0.06	-0.11	0.00	-0.13	-0.11	-0.07	0.14	-0.03
class_70	-0.04	-0.04	-0.04	-0.07	0.31	-0.03	-0.04	-0.02	0.26	-0.16	-0.06	-0.09	-0.11	-0.05	-0.05	1.00	-0.03	-0.14	-0.03	-0.04	-0.01	-0.11	0.02	0.01	-0.06	-0.03	0.19	-0.02
class_190	-0.09	-0.03	-0.03	0.08	0.02	-0.02	-0.01	0.03	0.10	-0.24	-0.08	0.01	-0.14	-0.04	-0.03	-0.03	1.00	0.06	-0.10	-0.03	0.08	-0.07	0.06	0.08	-0.02	-0.03	0.06	-0.02
fullbaths_residual	0.13	0.15	0.03	-0.14	-0.04	0.07	-0.01	0.01	-0.12	0.03	0.11	0.14	0.17	0.08	-0.07	-0.14	0.06	1.00	-0.28	-0.04	0.09	0.00	-0.08	-0.04	0.09	0.06	-0.08	0.00
halfbaths_residual	-0.01	-0.04	0.12	-0.06	0.41	-0.01	0.03	-0.11	-0.11	0.11	0.09	-0.07	0.06	-0.07	-0.08	-0.03	-0.10	-0.28	1.00	-0.08	-0.07	0.05	-0.08	-0.23	0.03	0.05	-0.10	0.00
frontage_residual	0.25	0.02	-0.29	-0.08	0.02	-0.07	0.04	0.12	-0.13	0.06	0.05	0.09	0.24	-0.19	-0.07	-0.04	-0.03	-0.04	-0.08	1.00	0.00	0.16	0.04	0.25	0.03	0.12	0.03	0.06
LotArea	0.26	-0.02	-0.14	0.03	-0.01	-0.05	0.00	0.09	-0.05	0.05	0.05	0.23	0.16	-0.13	-0.06	-0.01	0.08	0.09	-0.07	0.00	1.00	0.10	0.00	0.18	0.17	0.08	-0.02	0.02
MasVnrArea	0.48	0.01	0.05	-0.14	0.17	-0.03	0.03	0.26	-0.18	0.13	0.11	0.17	0.36	0.01	-0.11	-0.11	-0.07	0.00	0.05	0.16	0.10	1.00	-0.07	0.20	0.16	0.12	-0.11	0.02
LowQualFinSF	-0.03	-0.02	-0.01	0.15	-0.05	-0.02	-0.03	-0.02	0.11	-0.05	-0.04	-0.06	-0.13	-0.03	0.00	0.02	0.06	-0.08	-0.08	0.04	0.00	-0.07	1.00	0.26	-0.03	0.02	0.06	0.00
living_area_residual	0.39	0.16	-0.12	0.26	0.00	-0.11	-0.02	0.12	0.10	-0.07	-0.03	0.04	0.00	-0.12	-0.13	0.01	0.08	-0.04	-0.23	0.25	0.18	0.20	0.26	1.00	0.12	0.21	0.10	0.02
WoodDeckSF	0.32	-0.08	-0.05	-0.07	0.07	-0.02	0.04	0.08	-0.12	0.15	0.16	0.24	0.26	0.06	-0.11	-0.06	-0.02	0.09	0.03	0.03	0.17	0.16	-0.03	0.12	1.00	0.06	-0.13	-0.03
OpenPorchSF	0.32	-0.10	-0.06	-0.11	0.19	-0.07	-0.04	0.03	-0.07	0.03	0.10	0.08	0.22	0.02	-0.07	-0.03	-0.03	0.06	0.05	0.12	0.08	0.12	0.02	0.21	0.06	1.00	-0.09	-0.01
EnclosedPorch	-0.13	-0.06	-0.06	0.15	-0.02	-0.01	-0.01	-0.04	0.28	-0.16	-0.14	-0.10	-0.18	-0.06	0.14	0.19	0.06	-0.08	-0.10	0.03	-0.02	-0.11	0.06	0.10	-0.13	-0.09	1.00	-0.04
3SsnPorch	0.04	-0.02	-0.02	0.01	-0.03	-0.02	0.00	0.03	-0.03	0.03	0.01	-0.02	0.04	0.00	-0.03	-0.02	-0.02	0.00	0.00	0.06	0.02	0.02	0.00	0.02	-0.03	-0.01	-0.04	1.00
ScreenPorch	0.11	-0.05	-0.05	0.04	-0.02	-0.03	0.01	0.07	0.02	0.05	0.01	0.04	0.06	0.04	-0.04	0.00	-0.01	-0.06	0.05	0.01	0.04	0.06	0.03	0.04	-0.07	0.07	-0.08	-0.03
shed	-0.07	0.01	-0.03	0.06	-0.05	0.02	0.00	-0.02	0.01	0.00	0.03	-0.03	-0.04	-0.05	0.05	-0.02	0.00	-0.02	0.00	0.00	0.11	-0.05	0.02	-0.03	0.04	0.00	0.03	0.02
fence	-0.13	-0.04	-0.04	0.05	-0.08	0.00	0.16	-0.02	0.01	0.02	0.00	-0.07	-0.12	-0.10	0.00	0.02	-0.04	-0.07	-0.02	0.05	-0.03	-0.06	0.03	0.01	0.05	-0.04	0.07	-0.01
MiscVal	-0.02	0.05	-0.02	0.05	-0.02	-0.01	-0.01	0.04	-0.01	0.00	0.02	-0.01	-0.03	-0.02	0.00	0.00	0.03	-0.03	0.00	0.01	0.04	-0.03	0.00	0.02	-0.01	-0.02	0.02	0.00
LotShapeReg	-0.27	0.06	0.12	0.11	-0.07	0.01	-0.04	-0.02	0.13	-0.11	-0.11	-0.18	-0.23	0.00	0.09	0.06	0.08	-0.05	-0.05	-0.10	-0.22	-0.10	0.02	-0.06	-0.17	-0.08	0.09	-0.04
slope_Bnk	-0.10	0.03	-0.04	0.10	-0.05	-0.01	-0.03	-0.01	0.13	-0.14	-0.10	-0.05	-0.15	-0.05	0.10	0.09	0.09	-0.10	-0.07	0.03	0.03	-0.06	0.13	0.14	-0.05	-0.02	0.09	-0.01
slope_HLS	0.12	0.00	-0.03	-0.02	0.02	-0.03	-0.04	0.05	-0.03	0.02	-0.01	0.16	0.08	0.05	-0.01	0.04	0.00	0.04	-0.01	0.04	0.09	0.02	-0.02	-0.03	0.05	-0.04	-0.03	0.04
lotconfig_Corner	0.00	0.01	-0.08	0.02	0.01	-0.02	-0.01	-0.01	0.04	-0.04	-0.01	-0.04	0.01	-0.08	0.00	0.01	0.01	-0.03	-0.04	0.24	0.04	0.01	0.03	0.07	0.00	0.03	0.12	0.00
lotconfig_CulDSac	0.14	-0.04	-0.05	-0.05	0.03	0.06	0.04	0.03	-0.09	0.06	0.06	0.11	0.10	-0.01	-0.06	-0.05	-0.04	0.01	0.04	-0.11	0.18	0.05	-0.03	0.02	0.09	0.03	-0.06	0.05
LandSlope_Gentle	-0.05	0.03	0.04	-0.01	0.02	0.04	-0.01	0.01	0.02	0.01	0.01	-0.25	0.01	0.00	-0.02	-0.06	-0.03	-0.01	-0.01	0.01	-0.31	0.02	-0.02	-0.04	-0.09	0.03	0.00	0.00
ext_AsbShng	-0.11	0.04	-0.02	0.10	-0.02	-0.02	0.03	-0.03	0.09	-0.21	-0.12	-0.05	-0.11	-0.03	0.02	0.03	0.06	-0.04	-0.07	-0.03	-0.03	-0.05	-0.02	0.05	-0.08	-0.06	0.09	-0.01
ext_BrkFace	0.03	0.02	-0.03	0.03	-0.09	-0.03	0.00	0.08	0.02	-0.01	-0.11	-0.04	-0.01	0.00	-0.02	0.04	-0.03	-0.03	-0.04	-0.02	0.08	-0.11	-0.02	0.04	-0.07	-0.03	0.06	0.05
ext_CemntBd	0.13	-0.04	0.15	-0.05	0.04	0.12	-0.03	0.02	-0.06	0.04	0.05	0.06	0.05	0.08	-0.05	-0.03	-0.03	0.03	-0.02	-0.07	-0.03	0.05	-0.03	0.05	0.08	0.08	-0.06	-0.01

IDENTIFYING AND ELIMINATING OUTLIERS

OUTLIERS IN THE TRAINING SET COULD BIAS OUR MODEL!

- We decided to identify them and get rid of them – before fitting our models.
- We built a function that helps identify outliers in TRAIN – using scipy.spatial.distance:



Scale all variables (except for DV)

To insure all predictors have equal importance

Calculate pairwise distances b/w rows

We used Euclidian distances;

For 1460 rows, there were 1460*1459/2 pairwise distances.

For each row, sort distances ascending

The smallest distances are those to the row's closest neighbors, i.e., houses most similar to this house.

For each row, sum up 'kneighbors' distances

We used 40 kneighbors

The sum of those 40 smallest distances is how similar this row (house) is to its 40 closest neighbors

A large value means the house is dissimilar from its closest 40 neighbors

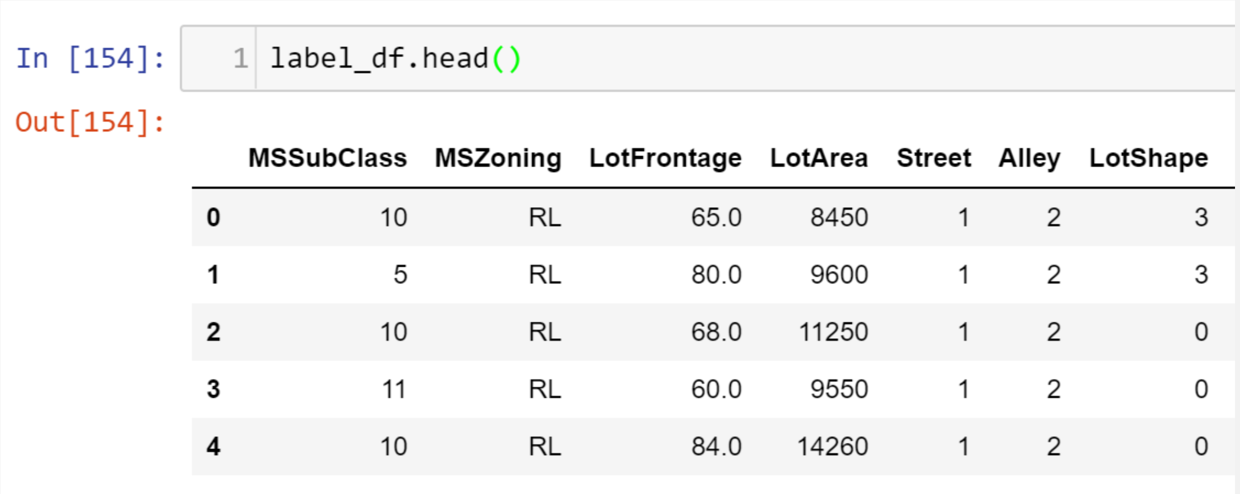
Discard rows with largest distances

Have to use judgement at this step

SECOND APPROACH TO FEATURE ENGINEERING – AFTER ZEYU’S ‘KAGGLE WORKFLOW’ LECTURE

USE OF LABEL ENCODING

- We decided to leverage Zeyu's 'label encoding' to recode all truly categorical variables
  - Categorical variables that were truly numeric were recoded same as before – using 'theory-driven' transformation
  - All truly categorical variables were recoded using label encoding
  - As a result, our second dataset contained no dummy variables
- We tried something new now – we ignored multicollinearity!



```
In [154]: label_df.head()
```

Out[154]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	10	RL	65.0	8450	1	2	3
1	5	RL	80.0	9600	1	2	3
2	10	RL	68.0	11250	1	2	0
3	11	RL	60.0	9550	1	2	0
4	10	RL	84.0	14260	1	2	0

PREDICTION!

PREDICTION FLOW & METHODS

Flow:

1. Run several regression models – separately
2. Use Cross-Validated Grid Search to tune hyperparameters for each model and compare scores
3. Submit predictions to Kaggle to check models' performance on Public Leaderboard
4. Combine individual models using several different ensembling methods

Supervised Learning Methods Used:

- Support Vector Regression (*sklearn.svm.SVR*)
- Random Forests Regression (*sklearn.ensemble.RandomForestRegressor*)
- Gradient Boosted Regression (*sklearn.ensemble. GradientBoostingRegressor*)
- Linear Models
  - Ridge, Lasso, Elastic Net, and Kernel Ridge (*sklearn.linear_model.Lasso, sklearn.linear_model.ElasticNet, sklearn.linear_model.KernelRidge*)

INDIVIDUAL MODELS: HYPERPARAMETERS AND SCORE ON GRID SEARCH + CV

Model	Parameters/Settings	CV RMSLE: Mean (SD)
Support Vector Regression	C=9, linear kernel, epsilon=0.009 C=9, linear kernel, epsilon=0.009	0.1245
Gradient Boosting Regression	max_depth = 2, max_features = 12, min_samples_split = 10, subsample = 0.7, learning_rate = 0.01, n_estimators = 400	0.1147
Lasso	alpha = 0.0003	0.1203
Elastic Net Regression	alpha = 0.0049, L1_ratio = 0.61	0.1174
Random Forest	n_estimators=800, max_features=13, oob_score=True |max_depth=7	0.1496
Kernel Ridge Regression	alpha=10, kernel='polynomial', degree=3, coef0=4	0.1165

Note: RMSLE is root mean squared log error, the scoring metric of the competition.

ENSEMBLING METHODS

- Ensembling methods used:

  1. Simple averaging: used the best tuned hyperparameters to fit individual models and averaged their predictions
  2. Stacking: used the same hyperparameters but trained a meta model (either a lasso or a gradient boosting machine) to produce predictions
  3. Due to the poor performance of our tree models, we averaged the predictions of just SVR and Elastic Net

ENSEMBLE PREDICTION RESULTS

Ensembling Approach	CV	Kaggle LB	Average
Simple Average (no random forests), new data	0.113	0.1220	0.119
Stacked Ensemble (meta = GBR), new data	0.135	0.1358	0.123
Best Individual (Elastic Net), new data	0.114	0.1169	n/a
Weighted averages of SVR, Elastic Net, KRR, and GBR predictions, new data	n/a	0.1154	n/a

LESSONS LEARNED

LESSONS LEARNED

- HAVE FUN! We had a great time working together, and it was a thrill as it all came together at the end
- Wrap it up at the right moment: learn to recognize diminishing returns
- Feature engineering is tough: so many ways to cut the data – hard to know a-priori what one is best
- Prediction ≠ Interpretation: Overemphasis of prediction lessens the need to understand the data and models
- Is collinearity bad? Eliminating it at the cost of losing features might not help even for linear models, or..?
- CV or Leaderboard??
  - Our individual linear models actually performed best for us on leaderboard
  - Our averaged model predictions performed the best on CV, close to best on LB
  - Our fancy ensemble with various meta-models didn't do quite as well

HYPERPARAMETERS ‘CHEAT SHEET’

RANDOM FOREST

- *n_estimators*: number of trees you want to build before the maximum voting/taking averages of predictions. A larger number of trees will allow the model to perform better, but will take longer to run
- *max_features*: dictates the max number of features RF can try within each split

  Increasing max_features generally improves the RF model because it allows for a larger number of options to be considered for each tree, but this also decreases the chance of overfitting
- *max_depth*: depth of tree = length of longest path from root node to leaf along a single decision tree. Minimizing the depth of individual trees within RF will fight overfitting.

SUPPORT VECTOR MACHINES (SVM)

- Kernel type - the type of kernel used. We tried:
  - *‘poly*' - polynomial kernel
  - '*rbf*' - radial basis function kernel = radial kernel
- *‘degree’* (for polynomial kernel only) – the nth degree of the polynomial (linear = 1, quadratic =2, etc.)
- *‘gamma’* (for radial kernel only): the hyperparameter involved in the radial kernel equation (and other nonlinear type kernels) - similar to an exponent:

$$K(x_i, x_{i'}) = e^{(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)}$$

- *‘C’* - helps determine the threshold of tolerable violations to the margin and hyperplane - it's like an "error" budget. As C increases - the margin gets larger variance decreases, and bias increases
  <u>Important:</u> For regression we can't 'missclasify' so the goal is instead to maximize the size of the margin, but include as many points as possible within the margin
- *‘epsilon’* - the error term measured from any point to the hyperplane; it’s a measure for whether or not a given point is inside the margin. By setting epsilon we are controlling how much error we allow the model per point. Super small epsilon leads to extreme overfitting.

GRADIENT BOOSTING

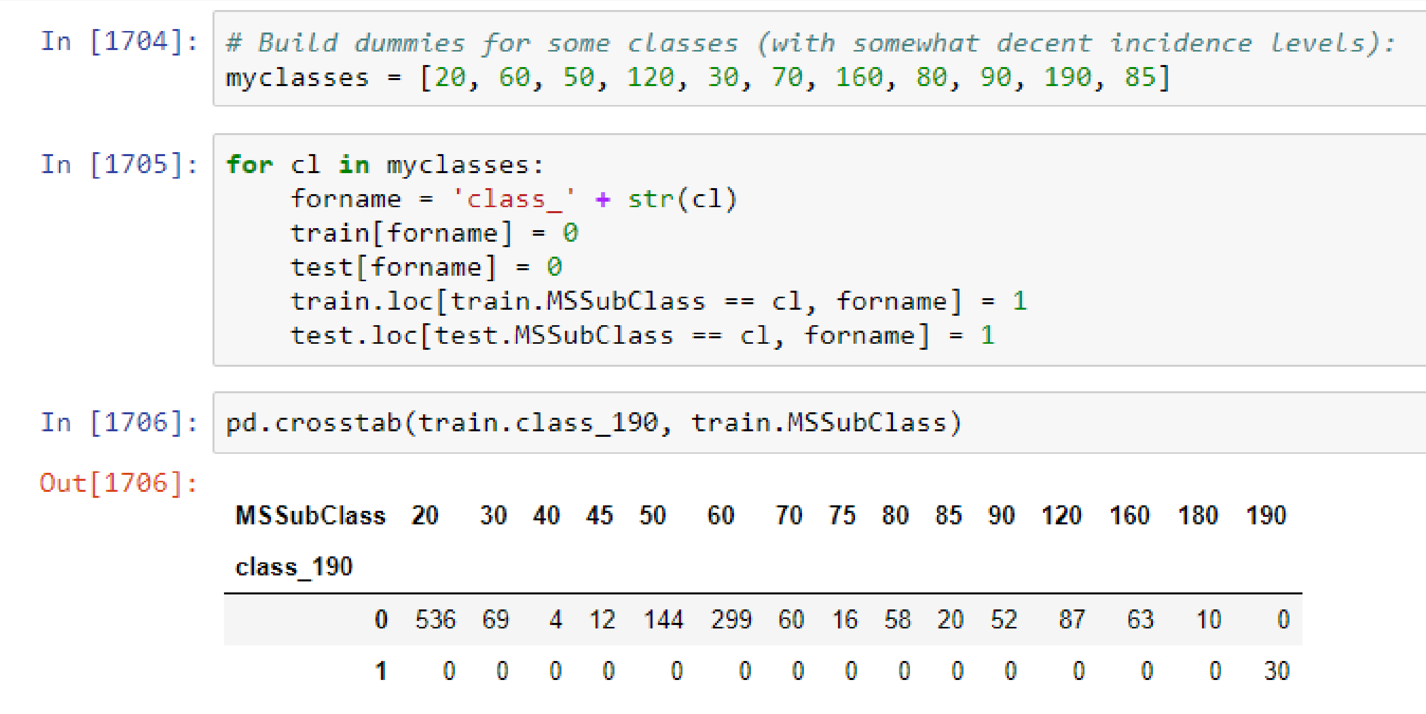
- We used trees so, all parameters that apply to depth of tree, number of features used at each split also apply here
- ‘*n_estimators*’ – total number of trees we want to build
- ‘*learning_rate*’ – the shrinkage factor = the degree to which we shrink each successive tree’s predictions. Small values mean that the learning is happening very slowly (large shrinkage). Large values imply that we shrink each tree’s predictions less so that the trees are ‘learning’ faster. However, that could lead to our missing the optimum point we are looking for.

REGRESSIONS – RIDGE, LASSO, ELASTIC NET, KERNEL

APPENDIX

WE BUILT DUMMIES FOR HIGHER INCIDENCE LEVELS: EITHER “MANUALLY” OR USING pd.get_dummies()

- “Manual” dummy calculation:



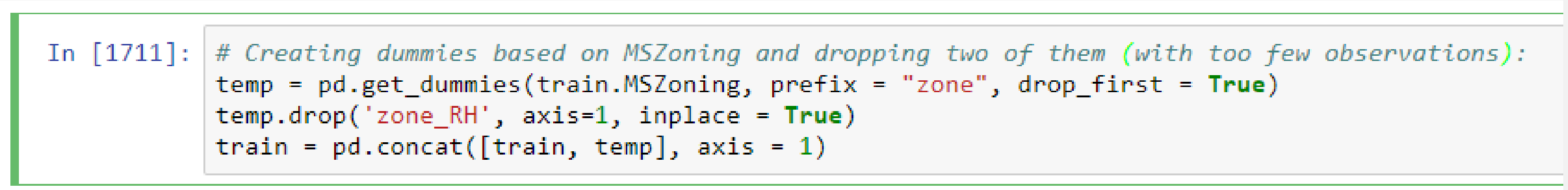
```
In [1704]: # Build dummies for some classes (with somewhat decent incidence levels):
           myclasses = [20, 60, 50, 120, 30, 70, 160, 80, 90, 190, 85]

In [1705]: for cl in myclasses:
               forname = 'class_' + str(cl)
               train[forname] = 0
               test[forname] = 0
               train.loc[train.MSSubClass == cl, forname] = 1
               test.loc[test.MSSubClass == cl, forname] = 1

In [1706]: pd.crosstab(train.class_190, train.MSSubClass)

Out[1706]:
            MSSubClass  20   30  40  45  50   60   70  75  80  85  90  120  160  180  190
            class_190
                     0  536  69   4  12  144  299  60  16  58  20  52   87   63   10    0
                     1    0   0   0   0    0    0   0   0   0   0   0    0    0    0   30
```

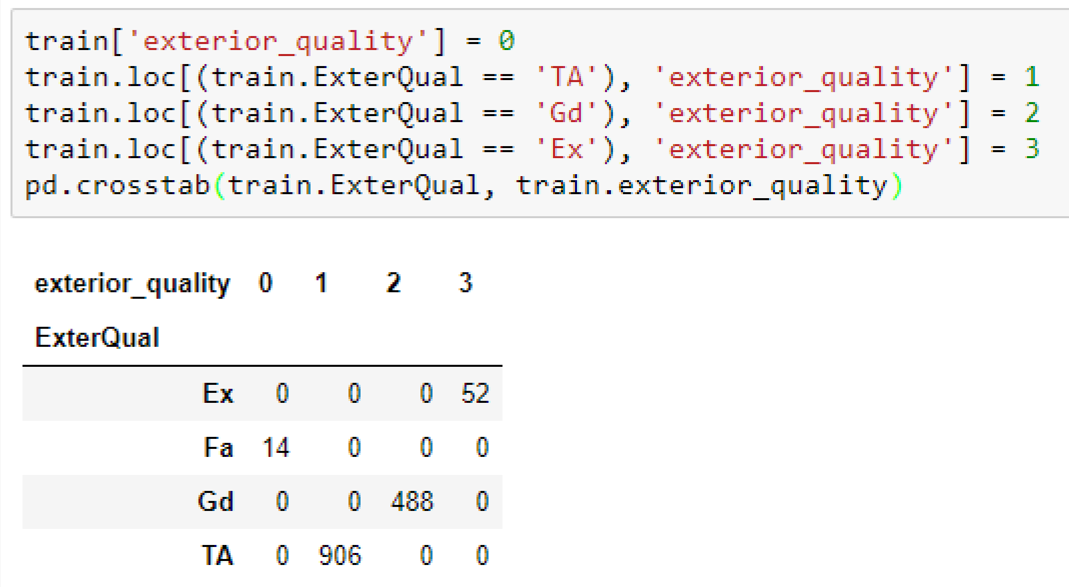
- Using pd.get_dummies()



```
In [1711]: # Creating dummies based on MSZoning and dropping two of them (with too few observations):
           temp = pd.get_dummies(train.MSZoning, prefix = "zone", drop_first = True)
           temp.drop('zone_RH', axis=1, inplace = True)
           train = pd.concat([train, temp], axis = 1)
```

WE TRANSFORMED 'OBJECT' COLUMNS THAT HAD QUANTITATIVE MEANING – USING OUR JUDGEMENT

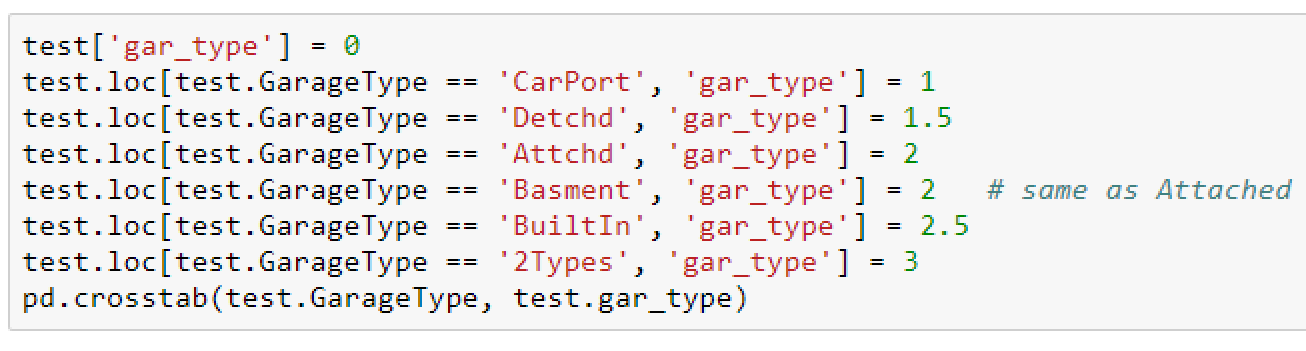
- Example of 'theory driven' transformation of Exterior Quality column:



```
train['exterior_quality'] = 0
train.loc[(train.ExterQual == 'TA'), 'exterior_quality'] = 1
train.loc[(train.ExterQual == 'Gd'), 'exterior_quality'] = 2
train.loc[(train.ExterQual == 'Ex'), 'exterior_quality'] = 3
pd.crosstab(train.ExterQual, train.exterior_quality)
```

exterior_quality	0	1	2	3
ExterQual				
Ex	0	0	0	52
Fa	14	0	0	0
Gd	0	0	488	0
TA	0	906	0	0

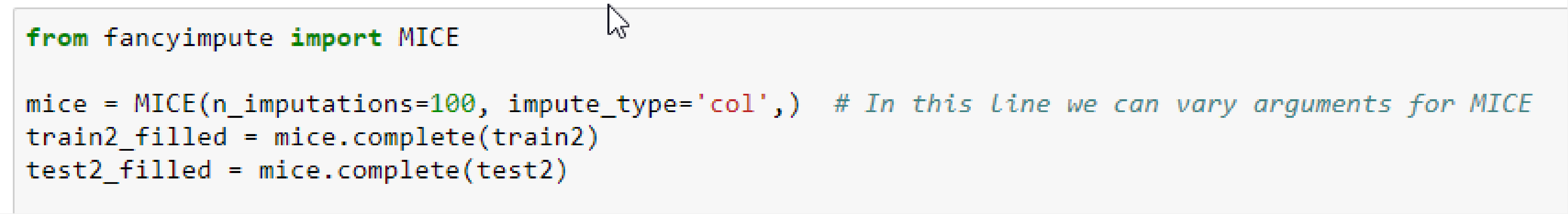
- Example of 'theory driven' transformation of Garage Type column:



```
test['gar_type'] = 0
test.loc[test.GarageType == 'CarPort', 'gar_type'] = 1
test.loc[test.GarageType == 'Detchd', 'gar_type'] = 1.5
test.loc[test.GarageType == 'Attchd', 'gar_type'] = 2
test.loc[test.GarageType == 'Basment', 'gar_type'] = 2   # same as Attached
test.loc[test.GarageType == 'BuiltIn', 'gar_type'] = 2.5
test.loc[test.GarageType == '2Types', 'gar_type'] = 3
pd.crosstab(test.GarageType, test.gar_type)
```

WE IMPUTED MISSING VALUES USING MICE FROM ‘fancyimpute’ PACKAGE

- ‘fanicyimpute’ package is a Python translation of R package ‘mice’ – the best library for missing data imputations.
- ‘fancyimpute’ is a bit tricky to install on Windows
- MICE stands for Multiple Imputation by Chained Equations
- Our code for imputation – we used MICE:

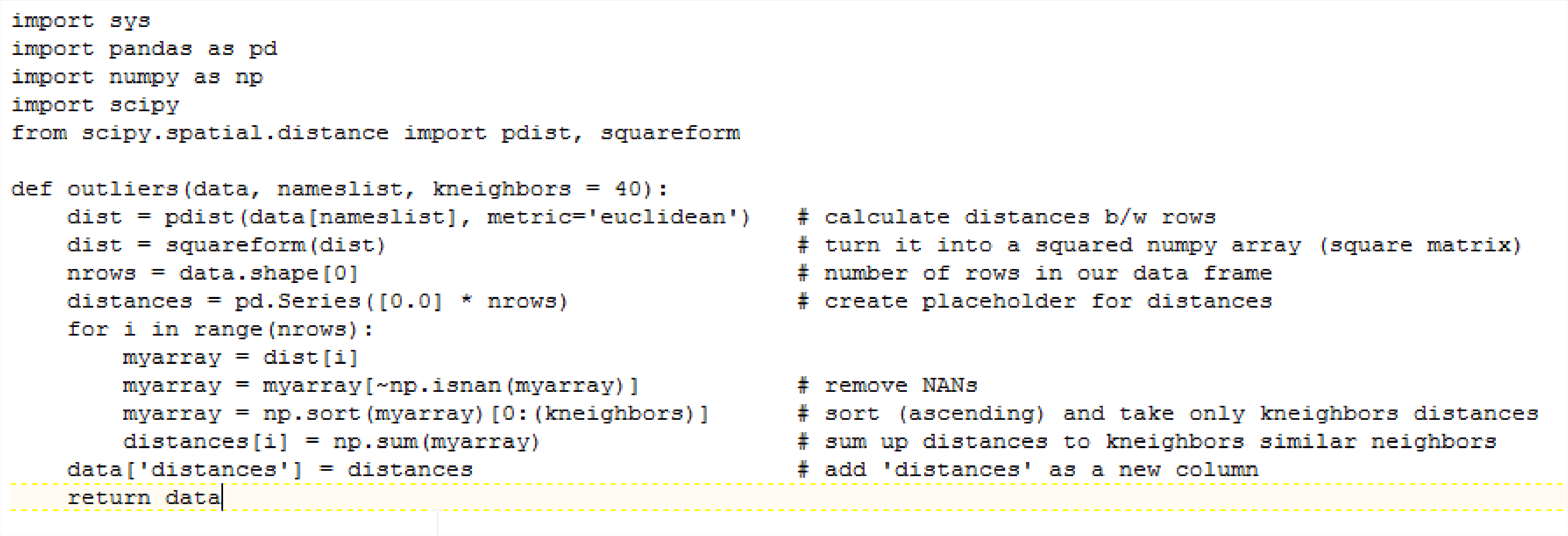


```
from fancyimpute import MICE

mice = MICE(n_imputations=100, impute_type='col',)  # In this line we can vary arguments for MICE
train2_filled = mice.complete(train2)
test2_filled = mice.complete(test2)
```

- Some detail on MICE: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/

OUR OUTLIERS FUNCTION



```
import sys
import pandas as pd
import numpy as np
import scipy
from scipy.spatial.distance import pdist, squareform

def outliers(data, nameslist, kneighbors = 40):
    dist = pdist(data[nameslist], metric='euclidean')   # calculate distances b/w rows
    dist = squareform(dist)                             # turn it into a squared numpy array (square matrix)
    nrows = data.shape[0]                               # number of rows in our data frame
    distances = pd.Series([0.0] * nrows)                # create placeholder for distances
    for i in range(nrows):
        myarray = dist[i]
        myarray = myarray[~np.isnan(myarray)]           # remove NANs
        myarray = np.sort(myarray)[0:(kneighbors)]      # sort (ascending) and take only kneighbors distances
        distances[i] = np.sum(myarray)                  # sum up distances to kneighbors similar neighbors
    data['distances'] = distances                       # add 'distances' as a new column
    return data
```