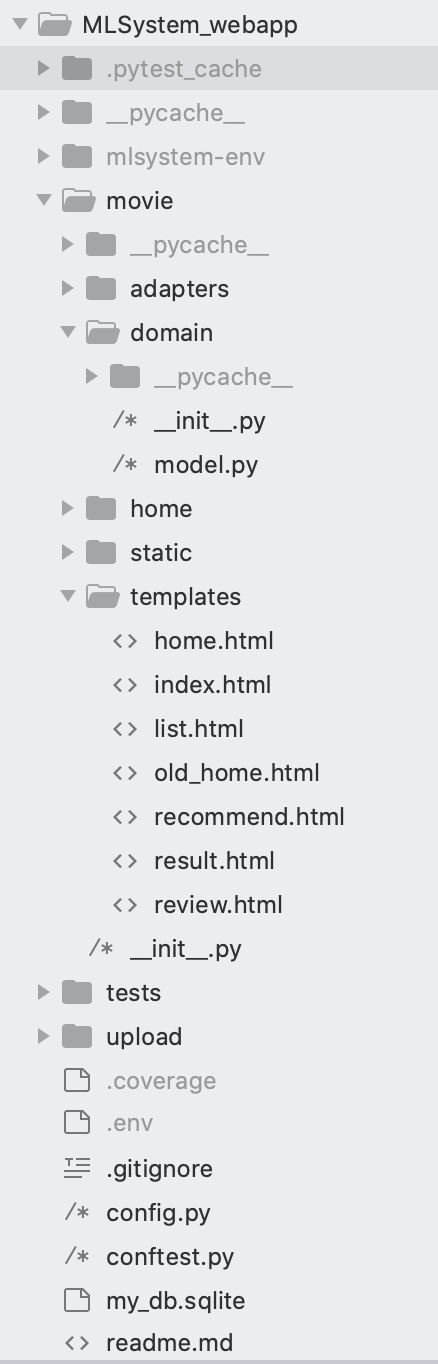
a11398\_a9364\_MLSystem

Task A House Price Prediction

工程整体架构

前后端架构:

Bootstrap，来自 Twitter，是目前最受欢迎的前端框架。Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的，它简洁灵活，使得 Web 开发更加快捷

Flask 是一个微型的 Python 开发的 Web 框架，基于Werkzeug WSGI工具箱和Jinja2 模板引擎。 Flask使用BSD授权。 Flask也被称为“microframework”，因为它使用简单的核心，用extension增加其他功能。Flask没有默认使用的数据库、窗体验证工具。然而，Flask保留了扩增的弹性，可以用Flask-extension加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。

jQuery是一个快速、简洁的JavaScript框架，是继Prototype之后又一个优秀的JavaScript代码库（或JavaScript框架）。 它封装JavaScript常用的功能代码，提供一种简便的JavaScript设计模式，优化HTML文档操作、事件处理、动画设计和Ajax交互

具体关键代码介绍：

forms（表单）（主要在template各种html）：存放表单对象

models（模型 ) (主要在domain/model.py）：存放数据模型，即库表在程序中的映射对象，以及对象之间的关系

routes（路由）：存放请求路由以及处理逻辑

static（静态文件）：Flask 约定存放静态文件的目录

templates（模板）：Flask 约定存放页面模板的目录

services（服务）：存放业务逻辑或者其他服务类功能

\_\_init\_\_.py：Flask app 初始化方法

config.py：项目配置文件

README.md - 仓库的说明，比如该项目的介绍等

wsgi.py - 项目的主要功能实现

config.py - 项目的配置

manage.py（非具体文件，你可以在命令行下使用） - 基于Flask-Script扩展的命令行脚本

requirements - 该项目所依赖的第三方包

unit\_tests - 单元测试相关代码

常用扩展（**Extensions**）：

**Flask**作为微框架（**microframework**），在开发过程中会经常使用各种扩展包。以下是一些常用扩展包的简介。

**Flask-SQLAlchemy -** 封装了**SQLAlchemy**，提供**ORM**

**Flask-Migrate -** 处理**SQLAlchemy**数据库的迁移（**migrations**）

**Flask-Script -** 支持在**Flask**里编写额外的脚本

**Flask-Bootstrap -** 封装了**Bootstrap**框架

**Flask-Login -** 提供账号**session**管理

**Flask-WTF -** 封装了**WTForms**，提供表单功能

**Flask-RESTful -** 提供快速构建**RESTAPIs**的能力

具体机器学习案例

**Business origin introduction**:

当我们拥有房屋属性数据和销售价格数据时，如果我们可以相互关联，这对于房地产网站和房屋购买者来说非常重要。

此外，该模型还可以更有效地帮助推荐系统发挥作用：知道预测价格对于价格敏感群体的房屋推荐非常重要。

我们有一个数据集，数据集中的每一行都描述了房屋的特征。我们的目标是根据房屋的特征预测销售价格。

评估模型基于模型预测的销售价格与实际销售价格之间的均方根误差（RMSE）。将RMSE转换为对数刻度，以确保预测昂贵房屋和廉价房屋的误差具有相同的得分影响。模型：每个交叉验证都适合许多模型（包括套索，山脊，svr，ker，ela，bay等）。所有训练后的模型在不同程度上过拟合训练数据。因此，为了做出最终的预测，我通过使用Stacking将它们的预测混合在一起以获得更可靠的预测。

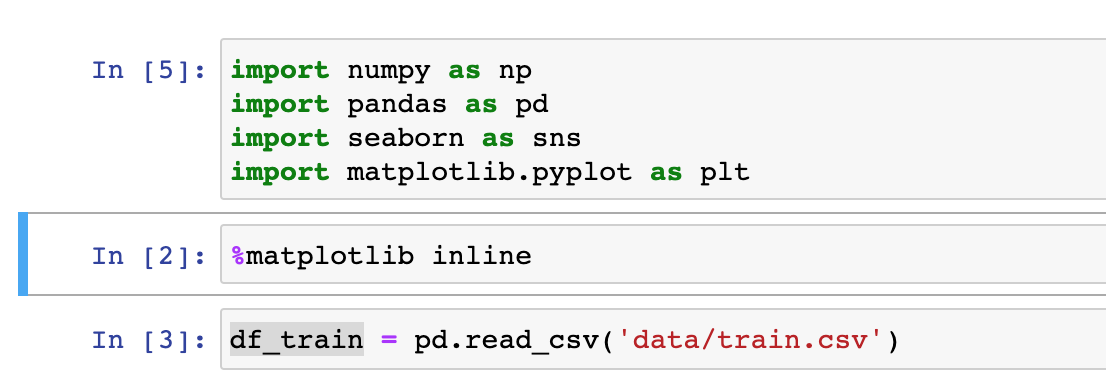
为了提高RMSE，我使用以下方法：

特征工程：主要为离散变量，特征组合和PCA分配值

模型融合：主要是加权平均和叠加

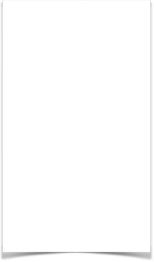
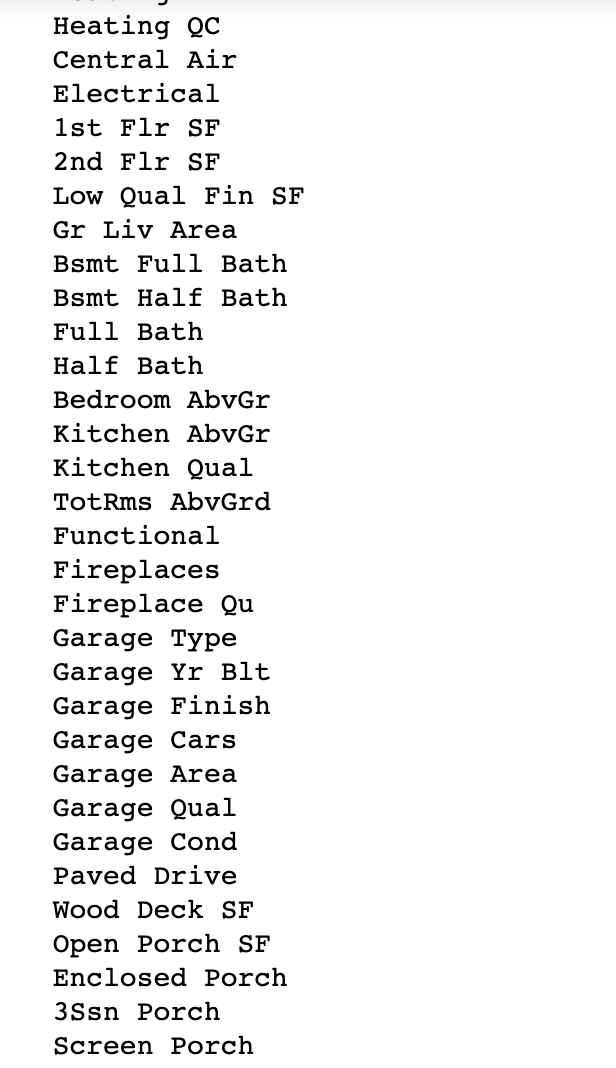
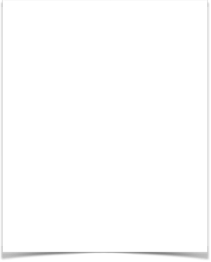
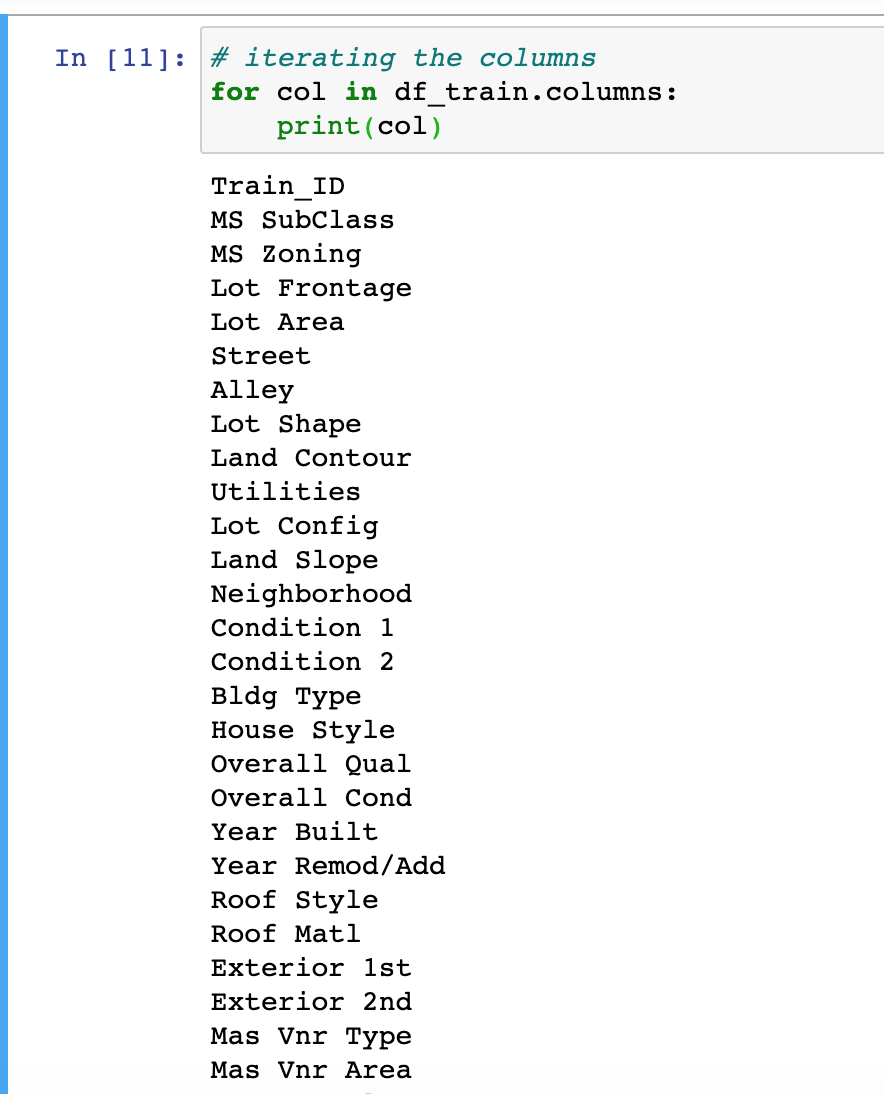
**Import necessary libraries**

主要使用： numpy,pandas, matplotlib.pyplot, seaborn… this libs to explore What attributes and data are needed for our predict.

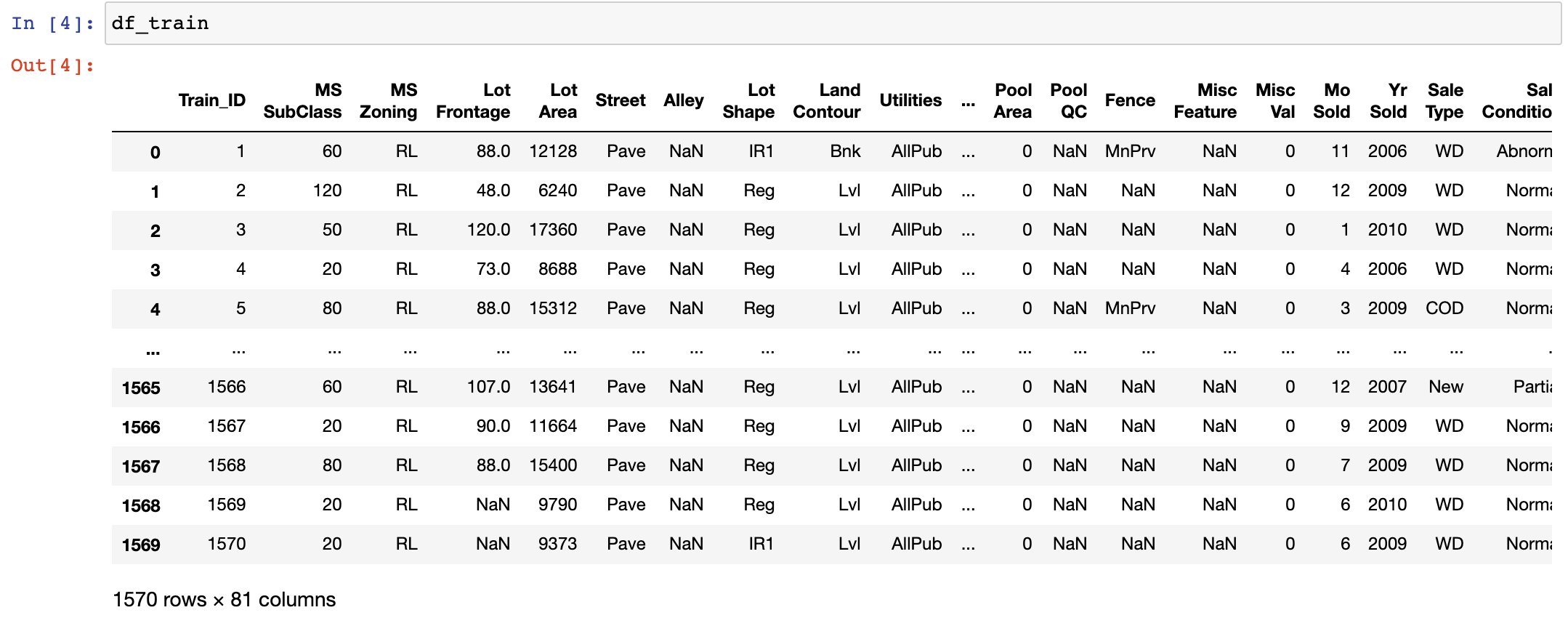


**Import Data**

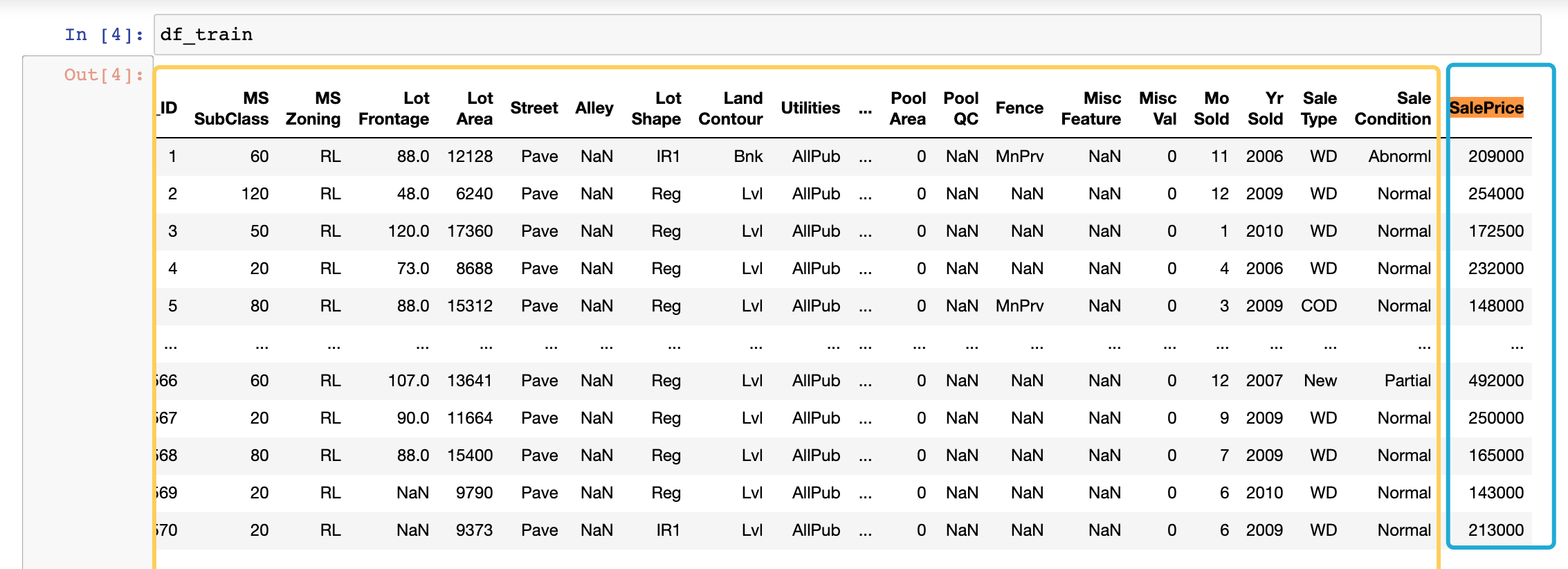
We explore all column names , read Kaggle\_House\_Data\_Description’s details and explainations.



Get all column names



Get data shape and some records

快速查看一下81列的值和列名，我们可以知道有80个要素，其中包括1个预测标签和79个要素标签。 具体分类如下 

仔细读取所有列名称作为特征以及列数据的解释。

For example:

MSSubClass: Identifies the type of dwelling involved in the sale.

MSZoning: Identifies the general zoning classification of the sale.

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Alley: Type of alley access to property

…

Fence: Fence quality

MiscFeature: Miscellaneous feature not covered in other categories

MiscVal: $Value of miscellaneous feature

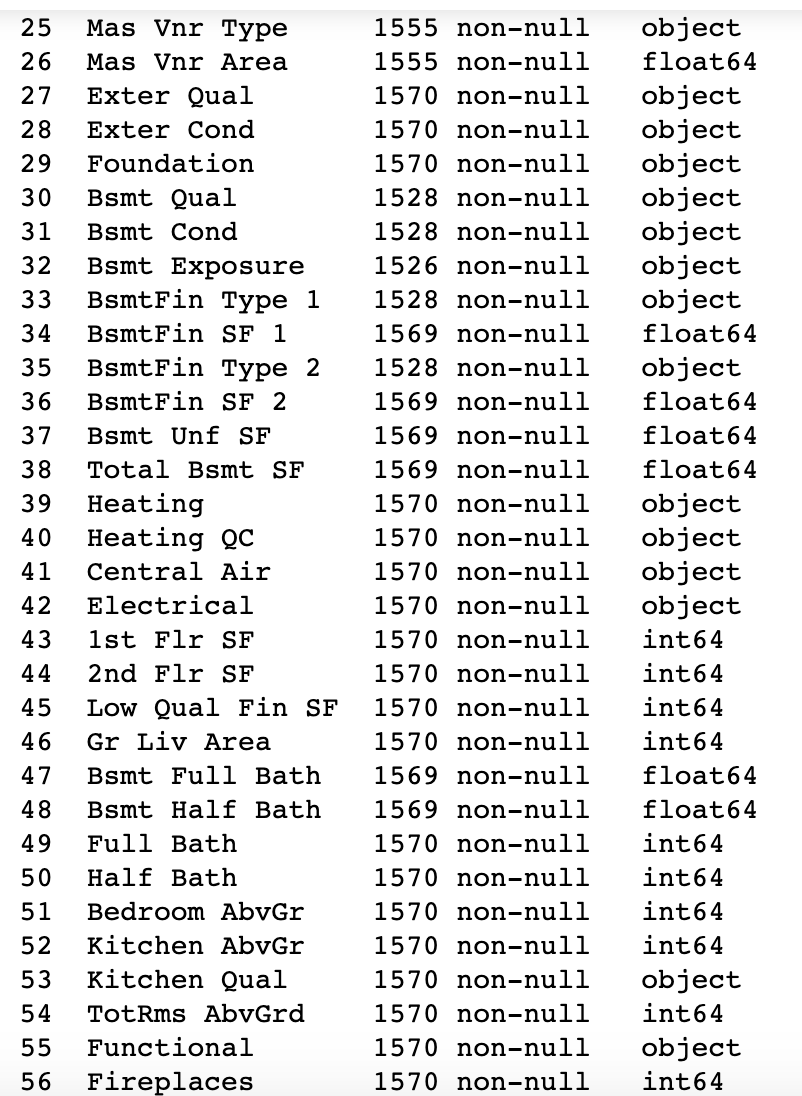
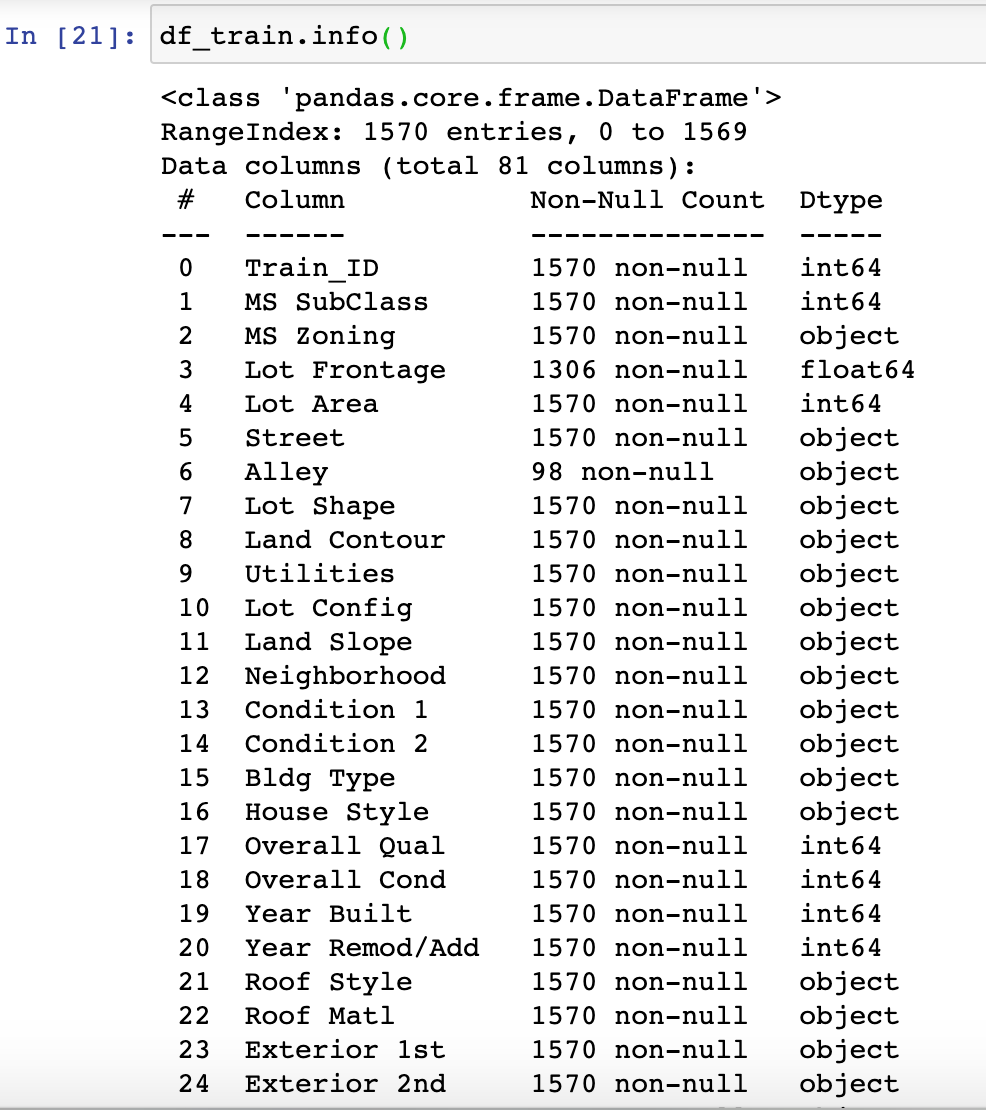
MoSold: Month Sold (MM)

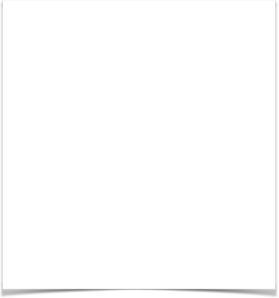
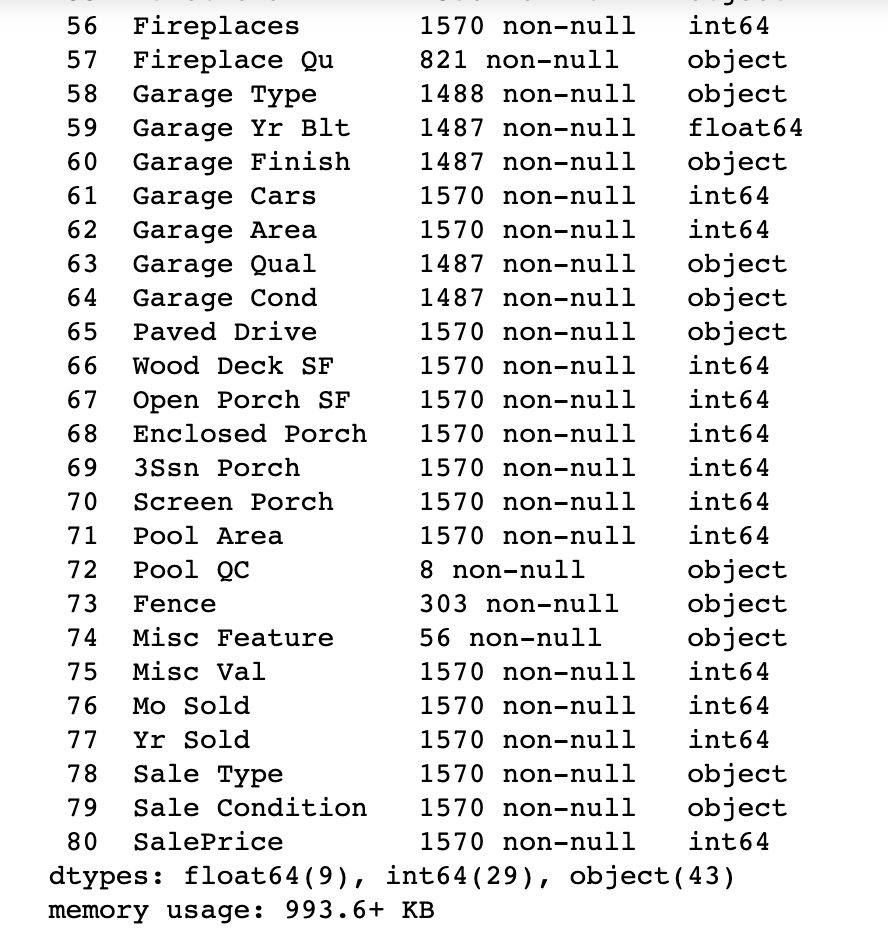
YrSold: Year Sold (YYYY)

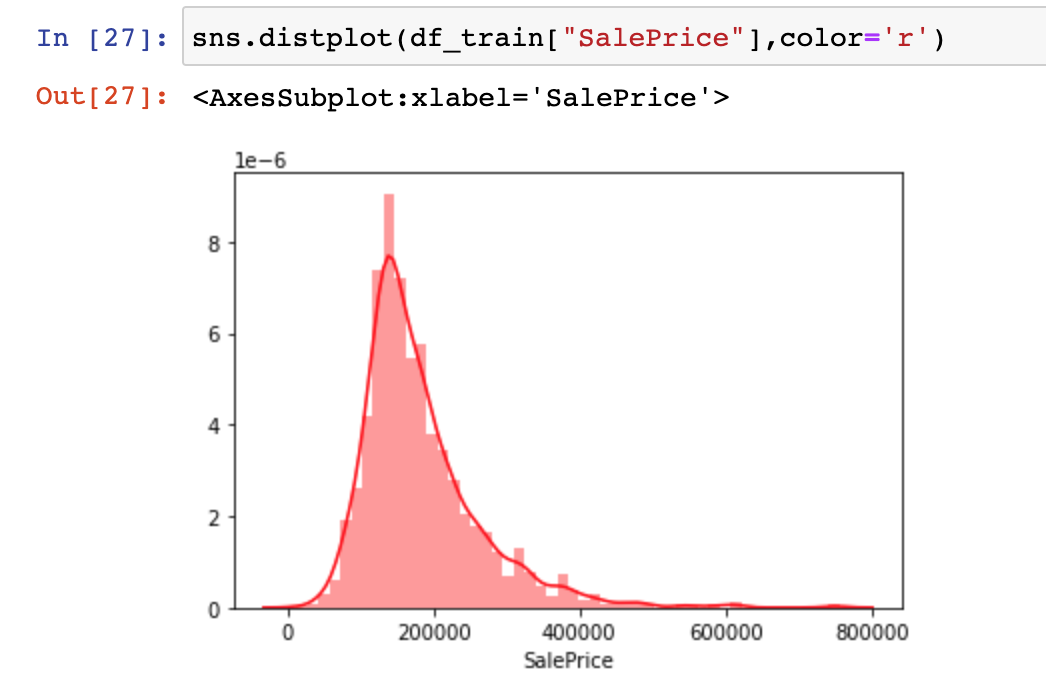
SaleType: Type of sale

SaleCondition: Condition of sale

**Look at the statistics**

查看数据集中每个数据列的默认数据，并查看哪些列的数据太少而需要删除 



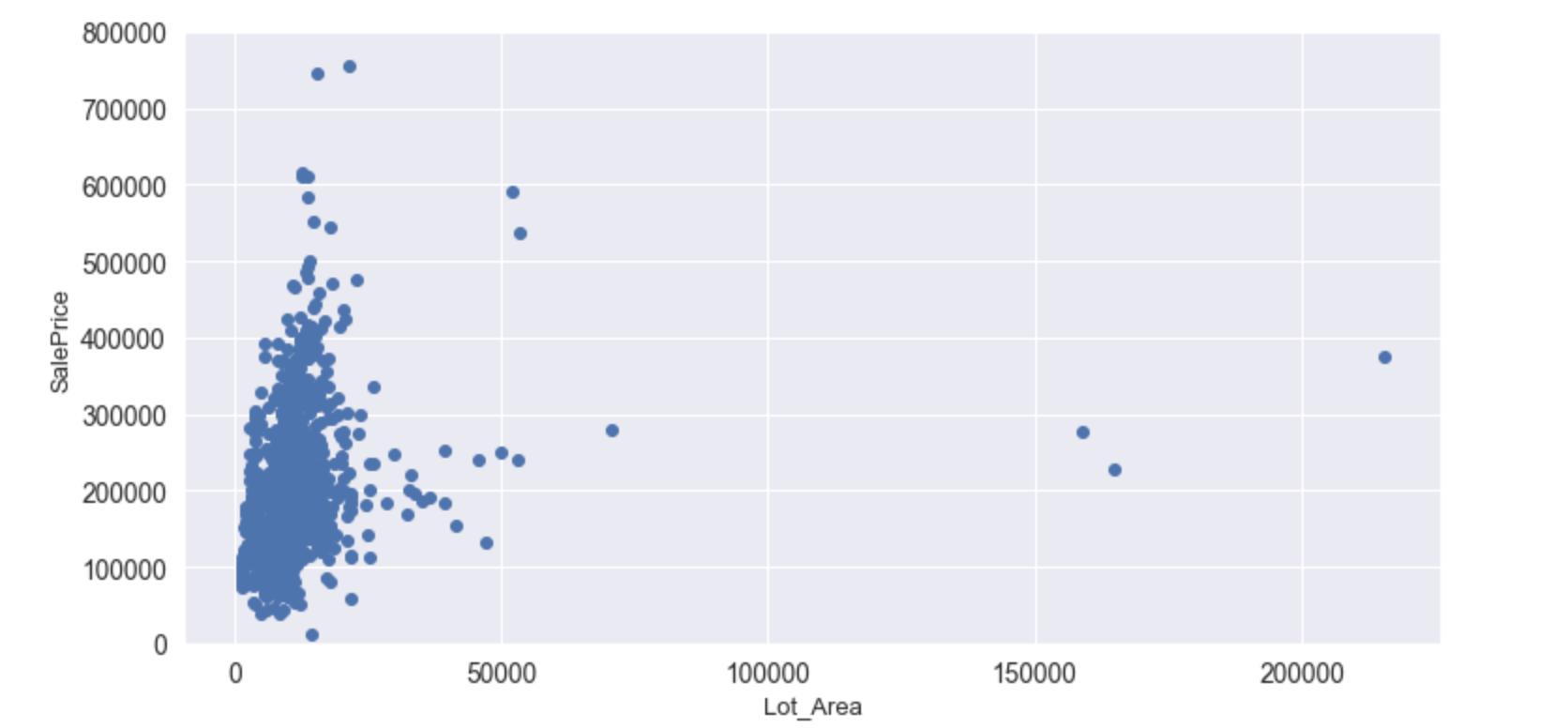
Through visualization, analyze the statistical law of the SalePrice itself

通过查看SalesPrice列的分布，我们发现大多数房屋价格都在$ 400,000之内，并且数据分布略向左

我们需要找到与salesprice最相关的列，我们进行测试，首先找到前20名，然后是前10名

我们可以看到20列的相关热图过多，因此我们将其分为10列，如下所示:

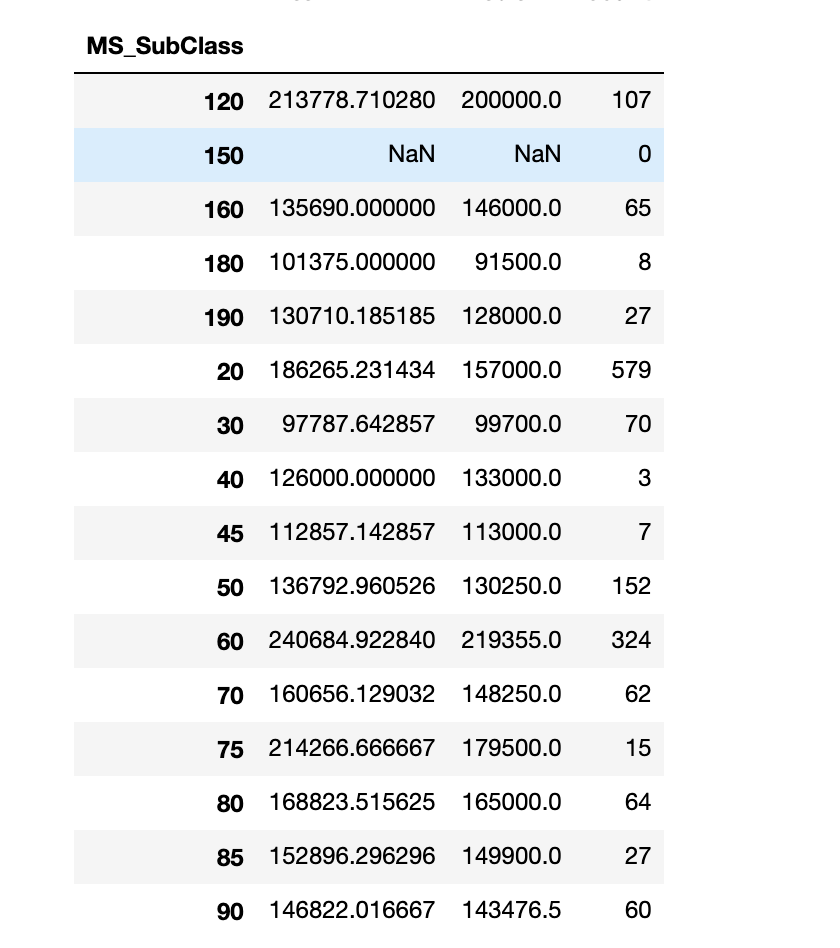


销售价格越相关，颜色越浅，通过观察：从图中可以看出 Overall Qual, Gr Live Area, Total Bsmt SF, 1st Flr SF, Garage Cars. 相关系数都比较高.

When we do visualization of price and columns we want to observe, we can find outlier points : right-bottom-corner points are not wanted. so we use :

train.drop(train[(train[“Lot\_Area"]>50000)&(train["SalePrice"]<400000)].index,inplace=True) to filter outlier data out.

**Feature Engineering**

或离散要素，通常使用pandas中的get\_dummies进行数字化，但这在Kaggle竞赛中可能还不够，因此我在下面使用的方法是按要素分组并计算要素的每个值的SalePrice平均值和中位数 ，然后以此为基础对值进行排序和分配，例如：:

Take MSSubClass as example : the feature of MSSubClass represents the type of house, and the data is grouped by it.

So We do the mapping as following:

MS\_SubClass.map({'180':1,

'30':2, '45':2,

'190':3, '50':3, '90':3,

'85':4, '40':4, '160':4,

'70':5, '20':5, '75':5, '80':5, '150':5,

'120': 6, '60':6})

Similar to MS\_SubClass.map, a total of more than 20 features are roughly ranked, as following:

def map\_into\_new\_values():

full["oMSSubClass"] = full.MS\_SubClass.map({'180':1,

'30':2, '45':2,

'190':3, '50':3, '90':3,

'85':4, '40':4, '160':4,

'70':5, '20':5, '75':5, '80':5, '150':5,

'120': 6, '60':6})

full["oMSZoning"] = full.MS\_Zoning.map({'C (all)':1, 'RH':2, 'RM':2, 'RL':3, 'FV':4})

full["oNeighborhood"] = full.Neighborhood.map({'MeadowV':1,

'IDOTRR':2, 'BrDale':2,

'OldTown':3, 'Edwards':3, 'BrkSide':3,

'Sawyer':4, 'Blueste':4, 'SWISU':4, 'NAmes':4,

'NPkVill':5, 'Mitchel':5,

'SawyerW':6, 'Gilbert':6, 'NWAmes':6,

'Blmngtn':7, 'CollgCr':7, 'ClearCr':7, 'Crawfor':7,

'Veenker':8, 'Somerst':8, 'Timber':8,

'StoneBr':9,

'NoRidge':10, 'NridgHt':10})

full["oCondition1"] = full.Condition\_1.map({'Artery':1,

'Feedr':2, 'RRAe':2,

'Norm':3, 'RRAn':3,

'PosN':4, 'RRNe':4,

'PosA':5 ,'RRNn':5})

full["oBldgType"] = full.Bldg\_Type.map({'2fmCon':1, 'Duplex':1, 'Twnhs':1, '1Fam':2, 'TwnhsE':2})

full["oHouseStyle"] = full.House\_Style.map({'1.5Unf':1,

'1.5Fin':2, '2.5Unf':2, 'SFoyer':2,

'1Story':3, 'SLvl':3,

'2Story':4, '2.5Fin':4})

full["oExterior1st"] = full.Exterior\_1st.map({'BrkComm':1,

'AsphShn':2, 'CBlock':2, 'AsbShng':2,

'WdShing':3, 'Wd Sdng':3, 'MetalSd':3, 'Stucco':3, 'HdBoard':3,

'BrkFace':4, 'Plywood':4,

'VinylSd':5,

'CemntBd':6,

'Stone':7, 'ImStucc':7})

full["oMasVnrType"] = full.Mas\_Vnr\_Type.map({'BrkCmn':1, 'None':1, 'BrkFace':2, 'Stone':3})

full["oExterQual"] = full.Exter\_Qual.map({'Fa':1, 'TA':2, 'Gd':3, 'Ex':4})

full["oFoundation"] = full.Foundation.map({'Slab':1,

'BrkTil':2, 'CBlock':2, 'Stone':2,

'Wood':3, 'PConc':4})

full["oBsmtQual"] = full.Bsmt\_Qual.map({'Fa':2, 'None':1, 'TA':3, 'Gd':4, 'Ex':5})

full["oBsmtExposure"] = full.Bsmt\_Exposure.map({'None':1, 'No':2, 'Av':3, 'Mn':3, 'Gd':4})

full["oHeating"] = full.Heating.map({'Floor':1, 'Grav':1, 'Wall':2, 'OthW':3, 'GasW':4, 'GasA':5})

full["oHeatingQC"] = full.Heating\_QC.map({'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})

full["oKitchenQual"] = full.Kitchen\_Qual.map({'Fa':1, 'TA':2, 'Gd':3, 'Ex':4})

full["oFunctional"] = full.Functional.map({'Maj2':1, 'Maj1':2, 'Min1':2, 'Min2':2, 'Mod':2, 'Sev':2, 'Typ':3})

full["oFireplaceQu"] = full.Fireplace\_Qu.map({'None':1, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})

full["oGarageType"] = full.Garage\_Type.map({'CarPort':1, 'None':1,

'Detchd':2,

'2Types':3, 'Basment':3,

'Attchd':4, 'BuiltIn':5})

full["oGarageFinish"] = full.Garage\_Finish.map({'None':1, 'Unf':2, 'RFn':3, 'Fin':4})

full["oPavedDrive"] = full.Paved\_Drive.map({'N':1, 'P':2, 'Y':3})

full["oSaleType"] = full.Sale\_Type.map({'COD':1, 'ConLD':1, 'ConLI':1, 'ConLw':1, 'Oth':1, 'WD':1,

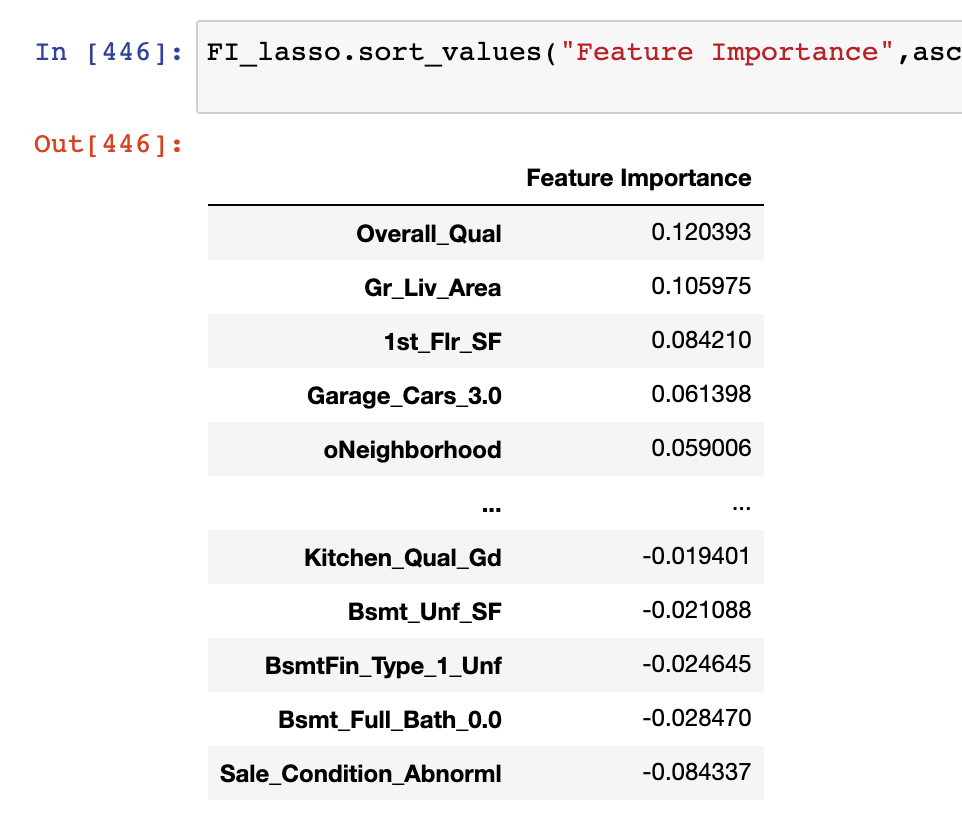
'CWD':2, 'Con':3, 'New':3})

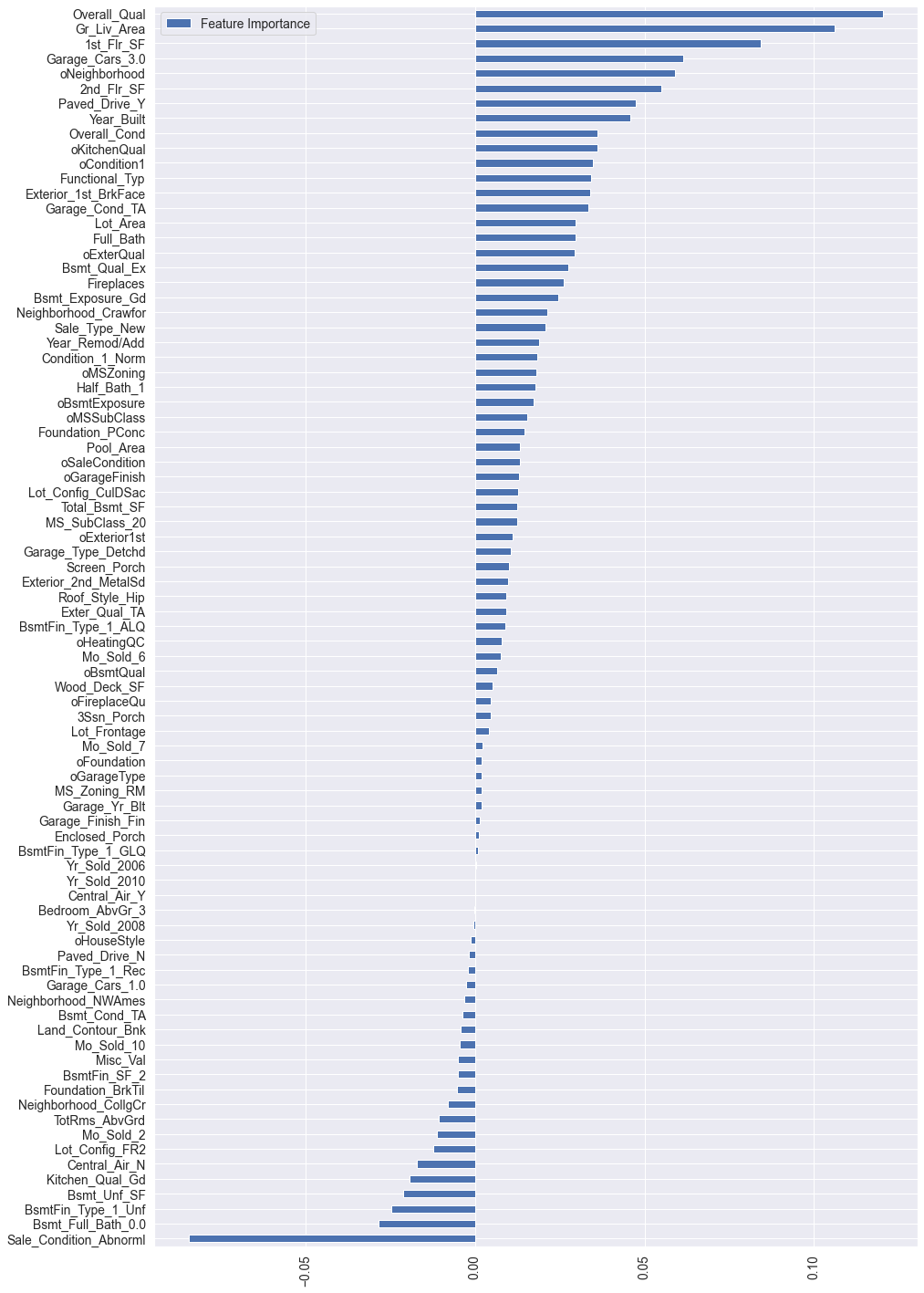
full["oSaleCondition"] = full.Sale\_Condition.map({'AdjLand':1, 'Abnorml':2, 'Alloca':2, 'Family':2, 'Normal':3, 'Partial':4})

I have converted the values of dozens of columns in total, in order to facilitate classification, you can refer to the code for details

**Feature combination**

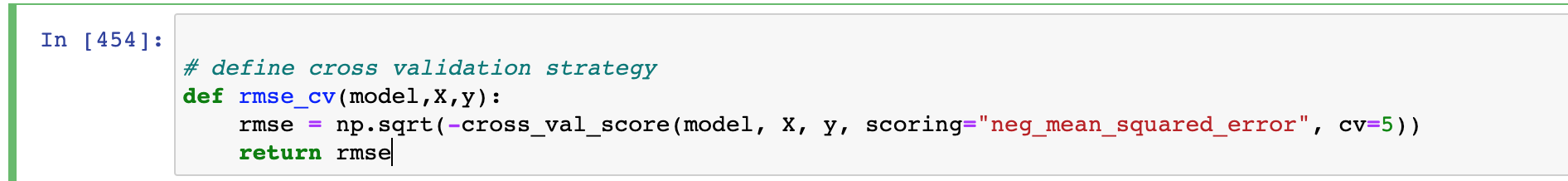
合并原始功能通常会产生意想不到的结果。 但是，此数据集中有许多原始特征，不可能将所有特征一一合并，因此这里我们首先使用Lasso进行特征筛选，然后选择一些更重要的特征进行组合





Finally, these features were added, which also included many other attempts. For details, please refer to the functions and classes related to add\_feature

PCA是非常重要的部分，它可以大大提高最终分数。 因为我添加的这些新功能与原始功能高度相关，所以这可能导致强大的多重共线性（Multicollinearity），而PCA可以修饰这种关联。 因为此处使用PCA的目的不是降低尺寸，所以n\_components使用的尺寸与原始尺寸相似。 这是我多方实验的结果，也就是说，在前面添加了XX个功能，然后缩小为XX个尺寸.

First define the cross-validation evaluation index of RMSE

**model selection**：12 algorithms and 5-fold cross-validation are used to evaluate the baseline effect:

**LinearRegression**

**Ridge**

**Lasso**

**Random Forrest**

**Gradient Boosting Tree**

**Support Vector Regression**

**Linear Support Vector Regression**

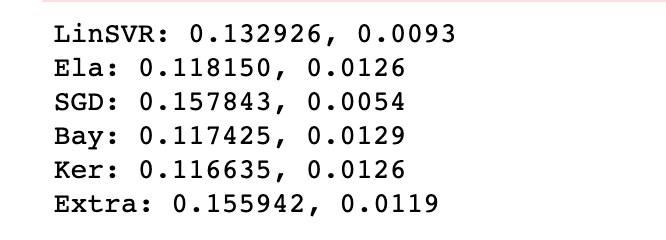
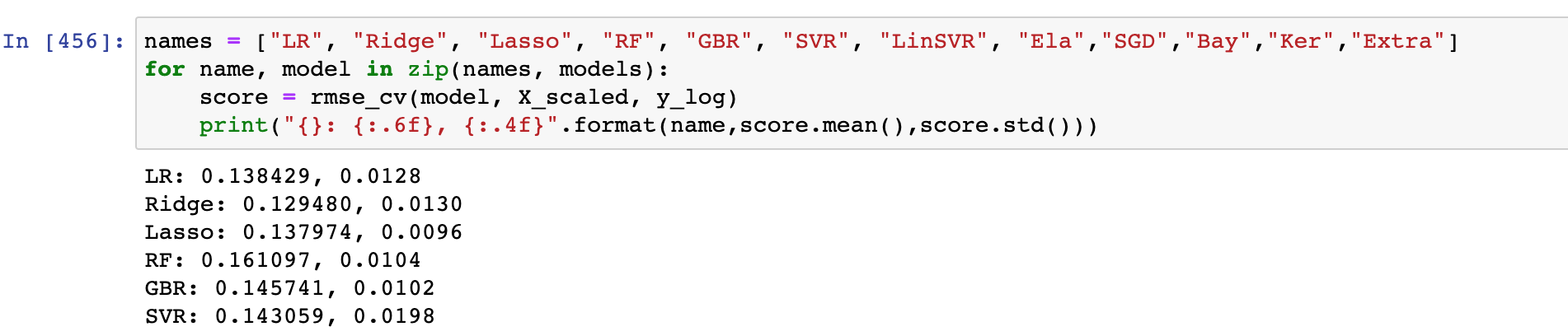
**ElasticNet**

**Stochastic Gradient Descent**

**BayesianRidge**

**KernelRidge**

**ExtraTreesRegressor**



除了测试结果外，我的想法是：树模型通常不如线性模型，这可能是由于get\_dummies /带来的数据稀疏性或数据集不足以使树模型具有最佳性能。

但是，这些模型参数尚未调整。 在做出选择之前，我们应该建立一种调整参数的方法。 我们应始终牢记，评估指标是RMSE.

method to adjust parameter , for all model to use:

grid\_search = GridSearchCV(self.model,param\_grid,cv=5, scoring="neg\_mean\_squared\_error")

grid\_search.fit(X,y)

print(grid\_search.best\_params\_, np.sqrt(-grid\_search.best\_score\_))

grid\_search.cv\_results\_['mean\_test\_score'] = np.sqrt(-grid\_search.cv\_results\_['mean\_test\_score'])

print(pd.DataFrame(grid\_search.cv\_results\_)[['params','mean\_test\_score','std\_test\_score']])

SVR

grid(SVR()).grid\_get(X\_scaled,y\_log,{'C':[11,13,15],'kernel':["rbf"],"gamma":[0.0003,0.0004],"epsilon":[0.008,0.009]})

results stablized at:

std\_test\_score

0 0.001553

1 0.001609

2 0.001555

….

11 0.001617

Kernel

param\_grid={'alpha':[0.2,0.3,0.4], 'kernel':["polynomial"], 'degree':[3],'coef0':[0.8,1]}

grid(KernelRidge()).grid\_get(X\_scaled,y\_log,param\_grid)

results stablized at:

std\_test\_score

0 0.001209

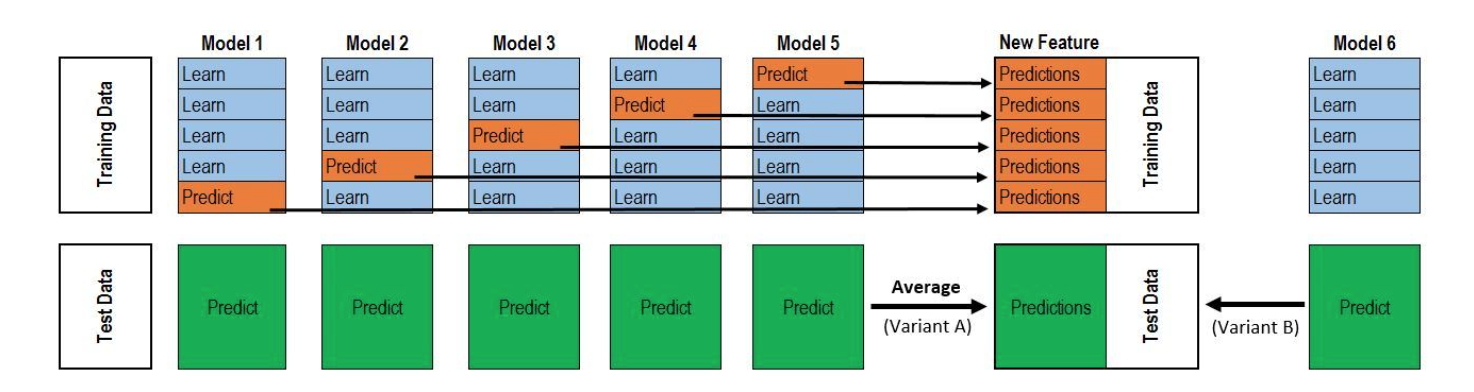
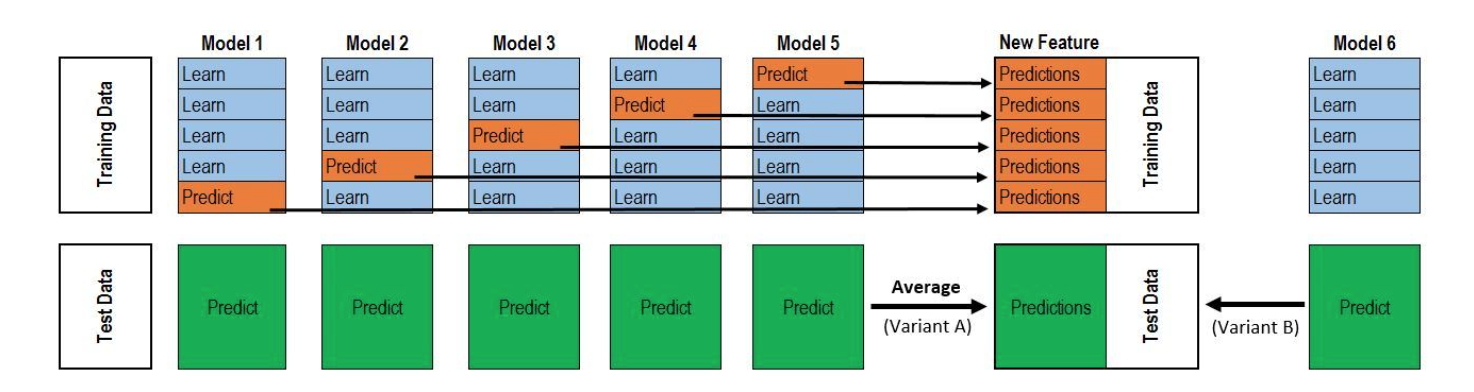
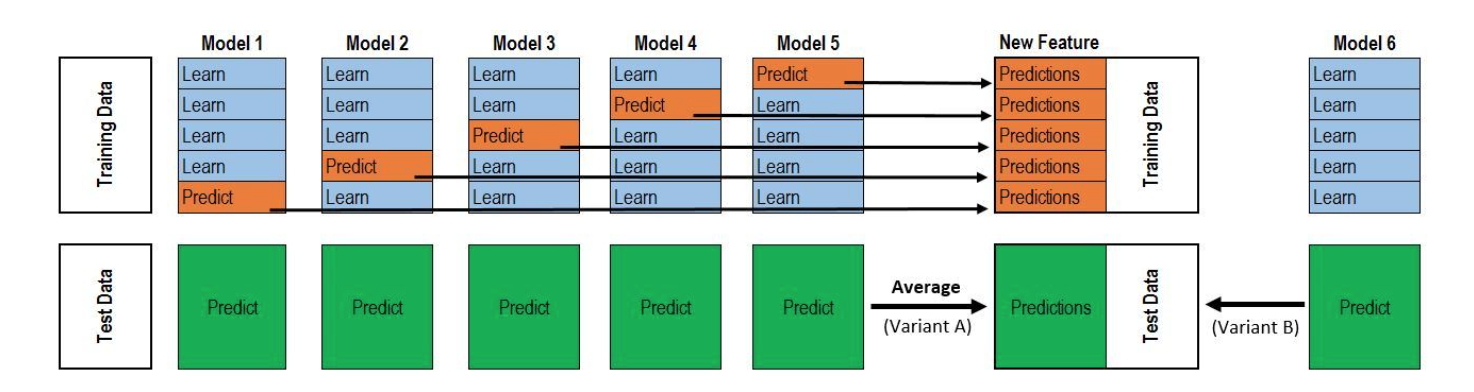
1 0.001243

2 0.001189

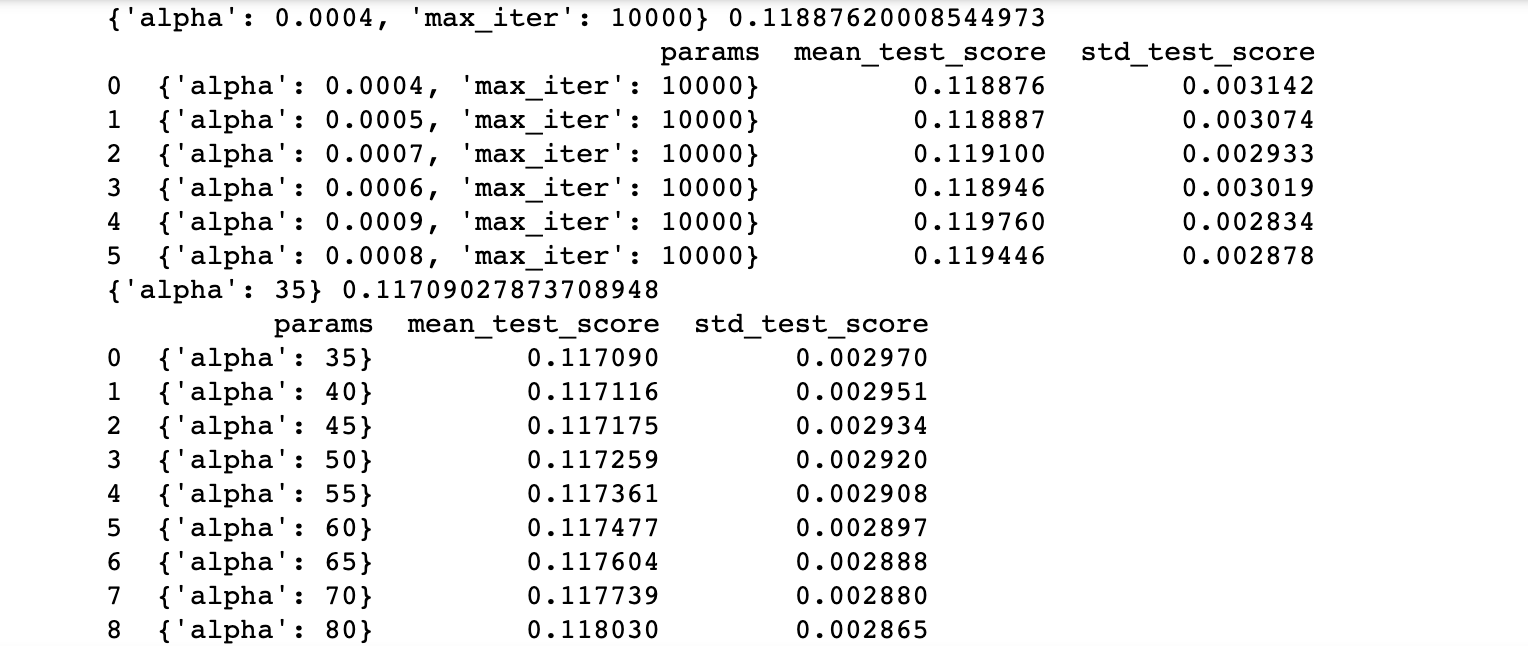
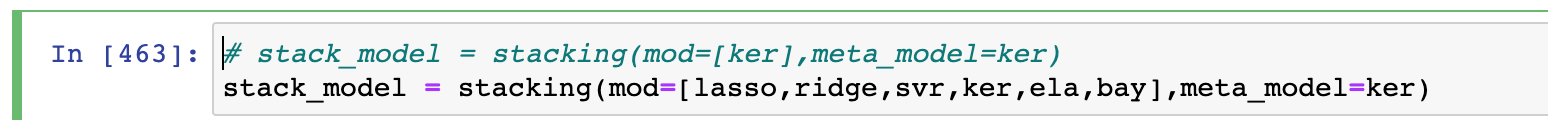
3 0.001210

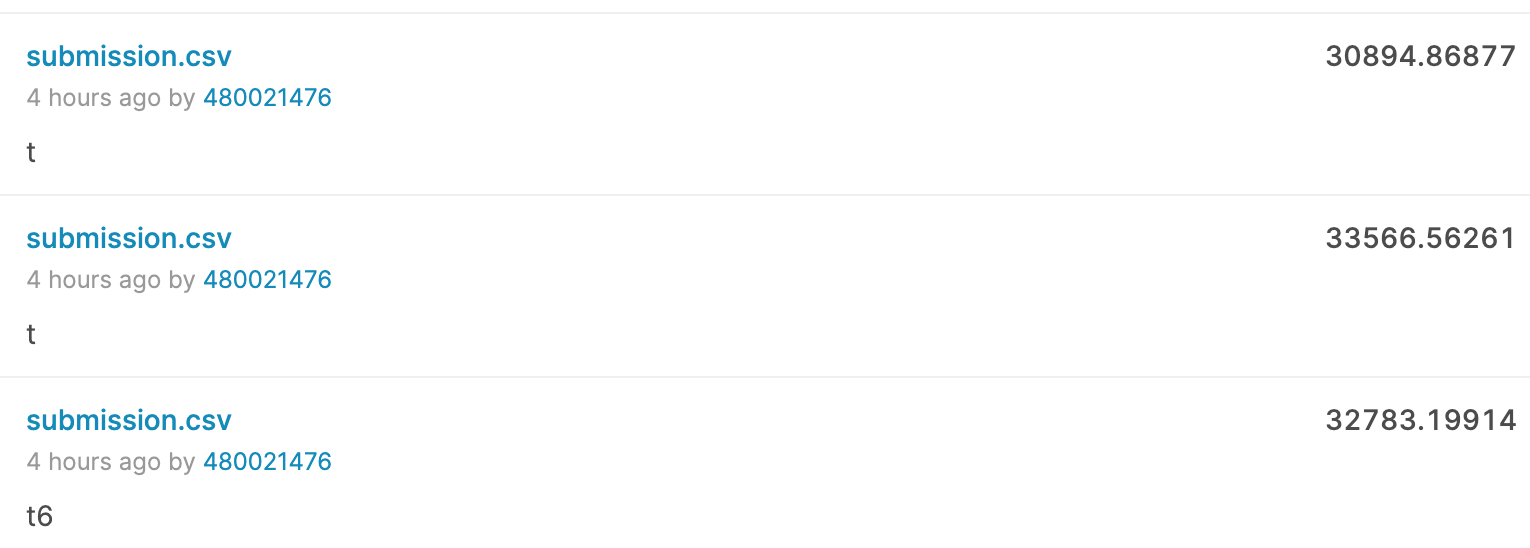
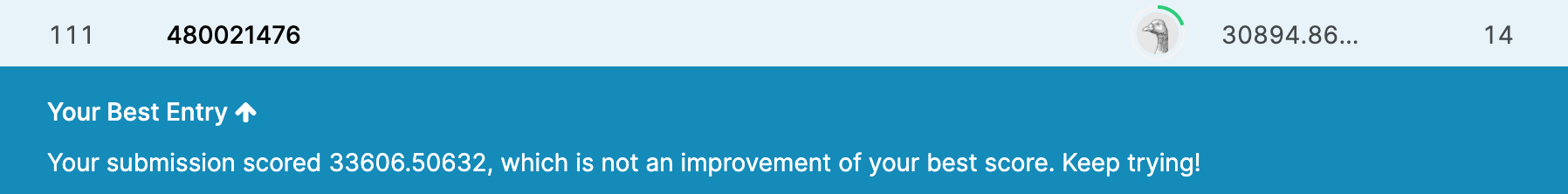
4 0.001181

5 0.001191   
 After many long rounds of testing among all the models, **SVR, KER** are my best 2 models. If I use single model, I will choose from SVR and Kernel Ridge two models.

But I want to combine Weighted average and stacking all 5 models.

In the two-layer stacking as shown in the figure, there are 5 models in the first layer and 1 meta-model in the second layer. The function of the first layer model is to train to obtain a [formula] feature matrix for input to the second layer model training, where n is the number of training data rows and m is the number of the first layer model.

After many long rounds of testing, these six models were finally selected

**Validation set Kaggle results   
**

**Final analysis, conclusion, limitations**

了解数据很重要，清理和转换数据也很重要。 既有离散功能又有连续功能，并且有很多缺失值。 幸运的是，参赛者提供了data\_description.txt文件，该文件描述了每个功能的含义。 了解内容后，可以对大多数缺失值进行平滑插值。

如果我有更多的时间和更多的数据，我想尝试一下神经网络方法。 在训练过程中，只要有足够的输入x和输出y，就可以训练更好的神经网络模型。 在类似的住房价格预测问题中，可以获得更准确的结果。

**ai**项目的工程化

flask

前后端分离是目前互联网项目开发的标准使用方式，其核心思想简单理解为是前端页面（或者app等多端）通过ajax（或者其他请求方式）调用后端的restuful api接口并使用json数据进行交互。其目的是为了将项目解耦合，达到"术业有专攻”的效果。因为在以往的web项目中，后端人员的工作量非常大，用过jsp的人都知道，那真的是又当爹又当妈，既要会写后台逻辑还要会弄样式。但是采用了前后端分离的架构之后，前后端人员就可以各司其职了。

前端人的追求

前端追求的是：页面表现，速度流畅，兼容性，用户体验等等。

把精力放在html5，css3，jquery，angularjs，bootstrap，reactjs，vuejs，webpack，less/sass，gulp，nodejs，Google V8引擎，模块化，设计模式，浏览器兼容性，性能优化等等。

后端人的追求

后端追求的是：三高（高并发，高可用，高性能），安全，存储，业务等等。

把精力放在语言基础，设计模式，底层原理，linux，mysql事务隔离与锁机制，mongodb，http/tcp，多线程，分布式架构，弹性计算架构，微服务架构，性能优化，以及相关的项目管理等等。

应用服务器、Web服务器、Restful

应用服务器：一般指像tomcat，jetty这类的服务器可以解析动态资源也可以解析静态资源，但解析静态资源的能力没有web服务器好。

Web服务器：一般指像nginx，apache这类的服务器，他们一般只能解析静态资源。

静态资源就是类似于html、js、图片这些多次访问也不会变化的资源  
一般都是只有web服务器才能被外网访问，应用服务器只能内网访问。

RESTful

REST的全称是representational state transfer，即表征状态转移。在理解这个名词之前我们先来看几个名词（感觉需要知道的前置知识好多啊）。

资源(resources)

所谓的资源就是网络上的一个实体，它可以使一个图片，一个文本，一个服务，你可以用一个URI指向它，每种资源对应一个特定的URI，要获取这个资源访问它的URI就行了，所谓的上网，其实就是与网络上的资源进行一系列的互动就是了。

表征(representation)

怎么把资源表现出来就是表征的意义，比如一段文本是txt、html还是json，图片是jpg还是png，以http协议为例，就是Accept和content-type中的内容，说明了资源的类型。

状态转移(state tranfer)

访问一个网站，就是客户端和服务端的一个交互过程，客户端想要操作服务端，就必须通过某种手段让服务端的状态发生变化，具体到http协议中就是http的几种方法：GET用来获取资源，POST用来新建资源，PUT用来更新资源，DELETE用来删除资源。

理解RESTful

1. 使用URI来表示每一个资源
2. 为每一个资源确定它的表现形式
3. 使用4个方法来操作这些资源

什么是Flask？

介绍完前后端分离后，我们就开始介绍下Flask是什么吧。

Flask是一个使用 Python 编写的轻量级 Web 应用框架。其 WSGI 工具箱采用 Werkzeug ，模板引擎则使用 Jinja2。Flask也被称为 “microframework” ，即"微框架"，因为它使用简单的核心，用 extension 增加其他功能。Flask没有默认使用的数据库、窗体验证工具。

理解下"微"

"微"框架中的“微”(micro) 并不表示你需要把整个 Web 应用塞进单个 Python 文件（虽然确实可以 ），也不意味着 Flask 在功能上有所欠缺。微框架中的“微”意味着 Flask 旨在保持核心简单而易于扩展。Flask 不会替你做出太多决策——比如使用何种数据库。而那些 Flask 所选择的——比如使用何种模板引擎——则很容易替换。除此之外的一切都由可由你掌握。默认情况下，Flask 不包含数据库抽象层、表单验证，或是其它任何已有多种库可以胜任的功能。然而，Flask 支持用扩展来给应用添加这些功能，如同是 Flask 本身实现的一样。众多的扩展提供了数据库集成、表单验证、上传处理、各种各样的开放认证技术等功能。Flask 也许是“微小”的，但它已准备好在需求繁杂的生产环境中投入使用。

什么是wsgi？

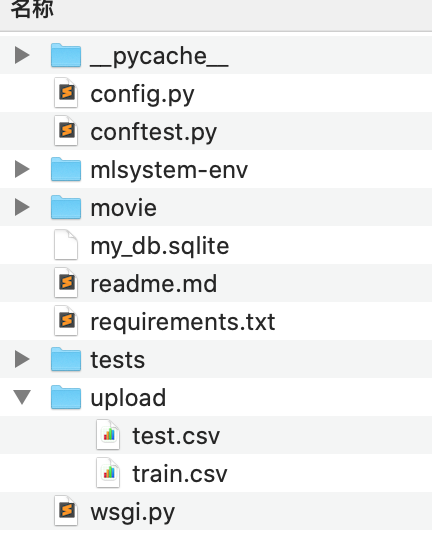
全名Web Server Gateway Interface，即服务器网关接口，是应用程序和Web服务器之间的一种接口。可以理解为是服务器程序和应用程序的一个约定，规定了各自使用的接口和功能，以便二和互相配合。

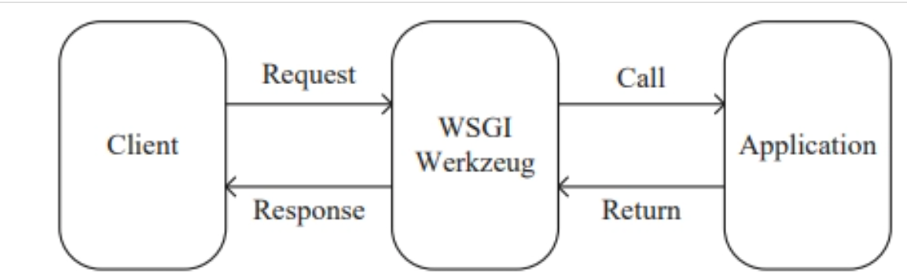
为什么选Flask？

我把Flask称为毕业设计大杀器自然是有他的道理的，接下来我们看看他的几大优点：

1. 插件多。查找资料方便
2. 没有太多繁琐的配置步骤
3. 各种中文资料、网友的受虐后的心得文章，查询方便
4. 部署也非常方便
5. 社区非常活跃

Flask是微型web框架，框架本身十分精简，微型并不代表其功能弱，核心代码基于Werkzeug, Jinja 2 这两个库,它以插件形式的进行功能扩展，且插件易于安装与使用，并且可以自行开发扩展插件。与其他web框架类似，flask中请求(request),路由(route),响应(response)构成其完整的一个基本http流程。



这是我们ai代码进行api工程化之后的组织结构

Flask的基本模式为在程序里将一个视图函数分配给一个URL，每当用户访问这个URL时，系统就会执行给该URL分配好的视图函数，获取函数的返回值并将其显示到浏览器上，其工作过程如上图。

Flask有两个主要依赖：路由、调试和Web服务器网关接口（WebServerGatewayInterface，WSGI）子系统由Werkzeug提供；模板系统由Jinja2提供。Werkzeug和Jinja2都是由Flask的核心开发者开发而成。对于数据库访问、验证Web表单和用户身份认证等一系列功能，Flask框架是不支持的。这些功能都是以扩展组件的方式进行实现，然后再与Flask框架集成。开发者可以根据项目的需求进行相应的扩展，或者自行开发。这与大型框架恰恰相反，大型框架本身做出了大部分决定，难以灵活改变方案。

当你点击一个网站，客户端（如浏览器）把请求发送给Web服务器，Web服务器把请求发送给Flask应用实例。应用实例需要知道每个url的请求需要执行哪些代码，所以保存了一个URL到Python函数的映射关系。处理url和函数之间关系的程序成为路由。 Flask中定义路由最方便的方式是用装饰器app.route。我们是在：

@app.route('/predict/')

@app.route("/result", methods=["GET"])

@flask\_login.login\_required

def result():

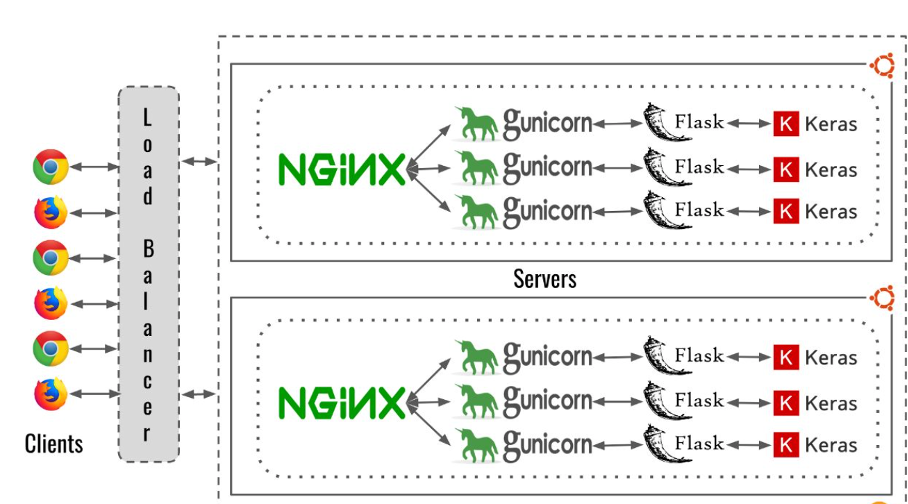
global data\_visual\_list, model\_name\_list

在result中进行不同算法的效果调用。

前台网页是使用bootstrap进行UI编写。

具体Bootstrap 是一个用于快速开发 Web 应用程序和网站的前端框架。Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的：

基本结构：Bootstrap 提供了一个带有网格系统、链接样式、背景的基本结构。这将在 Bootstrap 基本结构 部分详细讲解：

* CSS：Bootstrap 自带以下特性：全局的 CSS 设置、定义基本的 HTML 元素样式、可扩展的 class，以及一个先进的网格系统。这将在 Bootstrap CSS 部分详细讲解。
* 组件：Bootstrap 包含了十几个可重用的组件，用于创建图像、下拉菜单、导航、警告框、弹出框等等。这将在 布局组件 部分详细讲解。
* JavaScript 插件：Bootstrap 包含了十几个自定义的 jQuery 插件。您可以直接包含所有的插件，也可以逐个包含这些插件。这将在 Bootstrap 插件 部分详细讲解。
* 定制：您可以定制 Bootstrap 的组件、LESS 变量和 jQuery 插件来得到您自己的版本我们在前端bootstrap页面发送请求给后端，后端api调用算法和模型，计算出不同的时间和准确度，然后返回api到前端，通过template进行展示