learn kaggle

part 1 : Machine Learning

##split the numeric and categorical variables

```
num_columns = []
cate_columns = []
for column in test.columns:
    if train.dtypes[column] != 'object':
        num_columns.append(column)
    else:
        cate_columns.append(column)
```

```
#a easier method
cat_train=train.select_dtypes(['object'])
num_train=train.select_dtypes(exclude=['object'])
```

##evaluate the accuracy of ragression model

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_log_error
```

##split train set and test set

```
from sklearn.model_selection import train_test_split
train_X,test_X,train_y,test_y=train_test_split(X,y)  #train_size (default:0.75f),random_state
```

##estimate the mean absolute error of DecisionTree with different max nodes

```
def get_mae(max_nodes,train_X,train_y,test_X,test_y):
    model=DecisionTreeRegressor(max_nodes=max_nodes,random_state=0)
    model.fit(train_X,train_y)
    pre=model.predict(test_X)
    mae=mean_absolute_error(pre,test_y)
    return mae
```

##random forset model

```
from sklearn.ensemble import RandomForestRegressor
```

##deal with missing data

1.delete col with nan in this col (which is not a good solution to deal with missing data)

```
nan_col=[col for col in train.columns if train[col].isnull().any()]
reduced_date=train.drop(nan_col,axis=1)
```

2.a better way

```
from sklearn.preprocessing import Imputer
my_imputer=Imputer()
data_with_imputed_values=my_imputer.fit_transform(train)
```

we can change the fill method by change the para strategy,its default value is 'mean'.the optinal value contain 'median','most_frequent'

3.get score from impution with extra columns showing what wad imputed

```
miss_col=[col for col in train.columns if train[col].isnull().any()]
for col in miss_col:
        train[col+"_miss"]=train[col].isnull()
my_imputer=Imputer()
train=my_imputer.fit_transform(train)
```

##one hot encoding

one hot encoding generally won't be used for variables taking more than 15 values.

from this point, we need to get a col with less than 15 distinct values

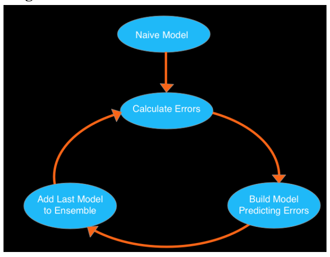
```
cat_cols=[col for col in train.columns if train[col].dtypes=='object'  and train[col].nunique()<15]
```

note:train[col].unique() returns an array, while nunique() returns a int

##delete rows which has nan values of several columns

```
train.dropna(axis=0,subset=['SalePrice'],inplace=True)
```

##xgboost



how to tune?

parameters:

n_estimators:specifies how many times to go through the model cycle described above

if its num is too big, it may bring the problem of overfitting,typical values range from 100 to 1000

early_stopping_rounds:cause the model to stop iterating when the validation score stops moving, even if we are not at the hard stop for n_estimators,

5 is a reasonable value
learning_rate:instead of getting predictions by simply adding up the predictions
from each component model, we will multiply the predictions from each model
by a small number before adding them in.this means each tree we add to this
ensemble helps less,this reduces propensity to overfit

##partial dependency plot
partial dependency plot how each variable or predictor affects the model's predictions
```
from sklearn.ensemble.partial_dependence import partial_dependence,plot_partial_dependence
model=GradientBoostingRegressor()
cols_to_use = ['Distance', 'Landsize', 'BuildingArea']
model.fit(X,y)  #here, we must fit before plot
plots=plot_partial_dependence(model,
                                 features=[0,2],
                                 X=X,
                                feature_names=cols_to_use,
                                grid_resolution=10)
```
there are some negative values, it doesn't mean it's really negative value.

##understanding pipeline
Pipelines are a simple way to keep your data processing and modeling code organized. Specifically, a pipeline bundles preprocessing and modeling steps so you can use the whole bundle as if it were a single step.

Most scikit-learn objects are either transformers or models.

Transformers are for pre-processing before modeling. The Imputer class (for filling in missing values) is an example of a transformer.

Models are used to make predictions. You will usually preprocess your data (with transformers) before putting it in a model.

example:
```
from sklearn.pipeline import make_pipeline
my_pipeline=make_pipeline(Imputer(),RandomForestRegressor())
my_pipeline.fit(train_X,train_y) #first it execute imputer, then train data with model
pre=my_pipeline.predict(test_X)
```

##cross-validation
train-test-split: suitable for small data set
ccross validation: suitable for bigger data set

example:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import Imputer
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
my_pip=make_pipeline(Imputer(),RandomForestRegressor())
scores=cross_val_score(my_pip,X,y,scoring='neg_mean_absolute_error') #para 'cv' default value:3
print scores
#result:[-322195.39079754 -303433.56326314 -283034.51985773]
```
different scoring function:
http://scikit-learn.org/stable/modules/model_evaluation.html

##leakage
two main types of leakage: Leaky Predictors and a Leaky Validation Strategies.

1.leaky predictors
This occurs when your predictors include data that will not be available at the time you make predictions.
solution:
To prevent this type of data leakage, any variable updated (or created) after the target value is realized should be excluded. Because when we use this model to make new predictions, that data won't be available to the model.
remember:

- To screen for possible leaky predictors, look for columns that are statistically correlated to your target.
- If you build a model and find it extremely accurate, you likely have a leakage problem.

2.leaky validation strategies
This occurs when you aren't careful distinguishing training data from validation data.
For example, this happens if you run preprocessing (like fitting the Imputer for missing values) before calling train_test_split.
solution:
If your validation is based on a simple train-test split, exclude the validation data from any type of fitting, including the fitting of preprocessing steps.