QBUS2810 Assignment 2

zeemaBlue 2020-11-17

Task A House Price Prediction

We have dataset that aach row in the data set describes the characteristics of the house. Our goal is to predict the sales price based on characteristics of houses.

The evaluation model is based on the root mean square error (RMSE) between the sales price predicted by the model and the actual sales price. Convert RMSE to a logarithmic scale to ensure that the error in predicting expensive houses and cheap houses has the same score impact. Models: each cross-validation fits many models (including lasso, ridge, svr, ker, ela, bay, etc.) All trained models overfit the training data to varying degrees. Therefore, in order to make the final prediction, I mixed their predictions together to get a more reliable prediction, by using Stacking.

In order to improve RMSE, I use following methods:

Feature Engineering: Mainly assign values to discrete variables, feature combination and PCA

Model fusion: mainly weighted average and stacking

Import necessary libraries

We are using numpy,pandas, matplotlib.pyplot, seaborn… this libs to explore What attributes and data are needed for our predict.

```
In [5]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt

In [2]: %matplotlib inline

In [3]: df_train = pd.read_csv('data/train.csv')
```

Import Data

We explore all column names , read Kaggle_House_Data_Description's details and explainations.

```
In [11]: # iterating the columns
         for col in df_train.columns:
             print(col)

Train_ID
MS SubClass
MS Zoning
Lot Frontage
Lot Area
Street
Alley
Lot Shape
Land Contour
Utilities
Lot Config
Land Slope
Neighborhood
Condition 1
Condition 2
Bldg Type
House Style
Overall Qual
Overall Cond
Year Built
Year Remod/Add
Roof Style
Roof Matl
Exterior 1st
Exterior 2nd
Mas Vnr Type
Mas Vnr Area
Heating QC
Central Air
Electrical
1st Flr SF
2nd Flr SF
Low Qual Fin SF
Gr Liv Area
Bsmt Full Bath
Bsmt Half Bath
Full Bath
Half Bath
Bedroom AbvGr
Kitchen AbvGr
Kitchen Qual
TotRms AbvGrd
Functional
Fireplaces
Fireplace Qu
Garage Type
Garage Yr Blt
Garage Finish
Garage Cars
Garage Area
Garage Qual
Garage Cond
Paved Drive
Wood Deck SF
Open Porch SF
Enclosed Porch
3Ssn Porch
Screen Porch
```

```
Screen Porch
Pool Area
Pool QC
Fence
Misc Feature
Misc Val
Mo Sold
Yr Sold
Sale Type
Sale Condition
SalePrice
```

Get all column names

```
In [4]: df_train
```

Out[4]:

	Train_ID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sal Conditio
0	1	60	RL	88.0	12128	Pave	NaN	IR1	Bnk	AllPub	...	0	NaN	MnPrv	NaN	0	11	2006	WD	Abnorm
1	2	120	RL	48.0	6240	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2009	WD	Norma
2	3	50	RL	120.0	17360	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	1	2010	WD	Norma
3	4	20	RL	73.0	8688	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	4	2006	WD	Norma
4	5	80	RL	88.0	15312	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	3	2009	COD	Norma
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
1565	1566	60	RL	107.0	13641	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2007	New	Parti
1566	1567	20	RL	90.0	11664	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2009	WD	Norma
1567	1568	80	RL	88.0	15400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	7	2009	WD	Norma
1568	1569	20	RL	NaN	9790	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2010	WD	Norma
1569	1570	20	RL	NaN	9373	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2009	WD	Norma

1570 rows × 81 columns

Get data shape and some records

A quick and simple look at the values and column names of the 81 columns, we can know that there are 80 features, including 1 predicted label, and 79 feature labels. The specific classification is as follows

```
In [4]: df_train
```

Out[4]:

ID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Condition	SalePrice
1	60	RL	88.0	12128	Pave	NaN	IR1	Bnk	AllPub	...	0	NaN	MnPrv	NaN	0	11	2006	WD	Abnorml	209000
2	120	RL	48.0	6240	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2009	WD	Normal	254000
3	50	RL	120.0	17360	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	1	2010	WD	Normal	172500
4	20	RL	73.0	8688	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	4	2006	WD	Normal	232000
5	80	RL	88.0	15312	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	3	2009	COD	Normal	148000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
566	60	RL	107.0	13641	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2007	New	Partial	492000
567	20	RL	90.0	11664	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2009	WD	Normal	250000
568	80	RL	88.0	15400	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	7	2009	WD	Normal	165000
569	20	RL	NaN	9790	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2010	WD	Normal	143000
570	20	RL	NaN	9373	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2009	WD	Normal	213000

Carefully read all the column names as features and the interpretation of the column data.

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

View the default data of each data column in the data set, and see which columns have too few data and need to be removed

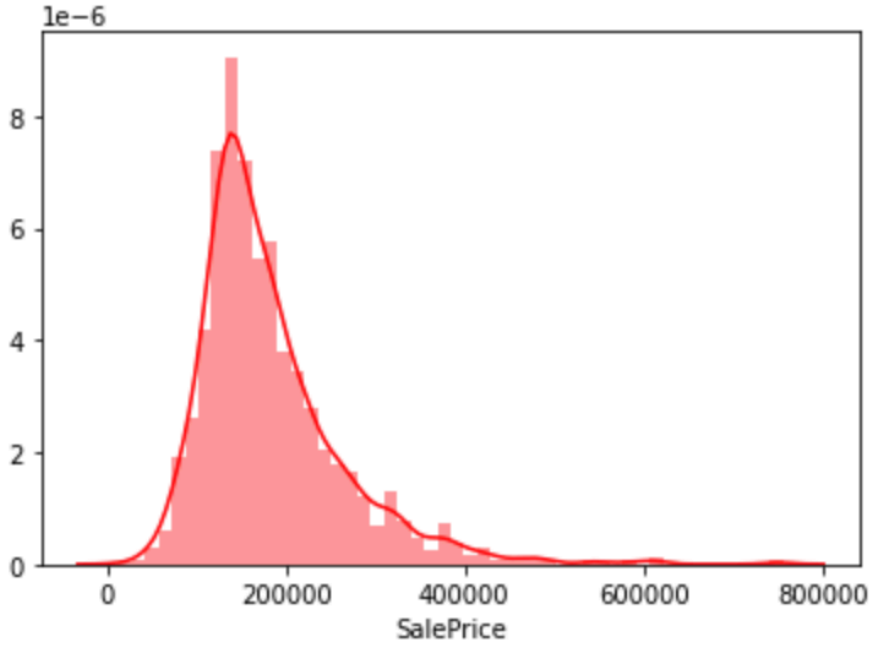
```
In [21]: df_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1570 entries, 0 to 1569
Data columns (total 81 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Train_ID         1570 non-null   int64
 1   MS SubClass      1570 non-null   int64
 2   MS Zoning        1570 non-null   object
 3   Lot Frontage     1306 non-null   float64
 4   Lot Area         1570 non-null   int64
 5   Street           1570 non-null   object
 6   Alley            98 non-null     object
 7   Lot Shape        1570 non-null   object
 8   Land Contour     1570 non-null   object
 9   Utilities        1570 non-null   object
 10  Lot Config       1570 non-null   object
 11  Land Slope       1570 non-null   object
 12  Neighborhood     1570 non-null   object
 13  Condition 1      1570 non-null   object
 14  Condition 2      1570 non-null   object
 15  Bldg Type        1570 non-null   object
 16  House Style      1570 non-null   object
 17  Overall Qual     1570 non-null   int64
 18  Overall Cond     1570 non-null   int64
 19  Year Built       1570 non-null   int64
 20  Year Remod/Add   1570 non-null   int64
 21  Roof Style       1570 non-null   object
 22  Roof Matl        1570 non-null   object
 23  Exterior 1st     1570 non-null   object
 24  Exterior 2nd     1570 non-null   object
 25  Mas Vnr Type     1555 non-null   object
 26  Mas Vnr Area     1555 non-null   float64
 27  Exter Qual       1570 non-null   object
 28  Exter Cond       1570 non-null   object
 29  Foundation       1570 non-null   object
 30  Bsmt Qual        1528 non-null   object
 31  Bsmt Cond        1528 non-null   object
 32  Bsmt Exposure    1526 non-null   object
 33  BsmtFin Type 1   1528 non-null   object
 34  BsmtFin SF 1     1569 non-null   float64
 35  BsmtFin Type 2   1528 non-null   object
 36  BsmtFin SF 2     1569 non-null   float64
 37  Bsmt Unf SF      1569 non-null   float64
 38  Total Bsmt SF    1569 non-null   float64
 39  Heating          1570 non-null   object
 40  Heating QC       1570 non-null   object
 41  Central Air      1570 non-null   object
 42  Electrical       1570 non-null   object
 43  1st Flr SF       1570 non-null   int64
 44  2nd Flr SF       1570 non-null   int64
 45  Low Qual Fin SF  1570 non-null   int64
 46  Gr Liv Area      1570 non-null   int64
 47  Bsmt Full Bath   1569 non-null   float64
 48  Bsmt Half Bath   1569 non-null   float64
 49  Full Bath        1570 non-null   int64
 50  Half Bath        1570 non-null   int64
 51  Bedroom AbvGr    1570 non-null   int64
 52  Kitchen AbvGr    1570 non-null   int64
 53  Kitchen Qual     1570 non-null   object
 54  TotRms AbvGrd    1570 non-null   int64
 55  Functional       1570 non-null   object
 56  Fireplaces       1570 non-null   int64
 56  Fireplaces       1570 non-null   int64
 57  Fireplace Qu     821 non-null    object
 58  Garage Type      1488 non-null   object
 59  Garage Yr Blt    1487 non-null   float64
 60  Garage Finish    1487 non-null   object
 61  Garage Cars      1570 non-null   int64
 62  Garage Area      1570 non-null   int64
 63  Garage Qual      1487 non-null   object
 64  Garage Cond      1487 non-null   object
 65  Paved Drive      1570 non-null   object
 66  Wood Deck SF     1570 non-null   int64
 67  Open Porch SF    1570 non-null   int64
 68  Enclosed Porch   1570 non-null   int64
 69  3Ssn Porch       1570 non-null   int64
 70  Screen Porch     1570 non-null   int64
 71  Pool Area        1570 non-null   int64
 72  Pool QC          8 non-null      object
 73  Fence            303 non-null    object
 74  Misc Feature     56 non-null     object
 75  Misc Val         1570 non-null   int64
 76  Mo Sold          1570 non-null   int64
 77  Yr Sold          1570 non-null   int64
 78  Sale Type        1570 non-null   object
 79  Sale Condition   1570 non-null   object
 80  SalePrice        1570 non-null   int64
dtypes: float64(9), int64(29), object(43)
memory usage: 993.6+ KB
```

Through visualization, analyze the statistical law of the SalePrice itself

```
In [27]: sns.distplot(df_train["SalePrice"],color='r')
```

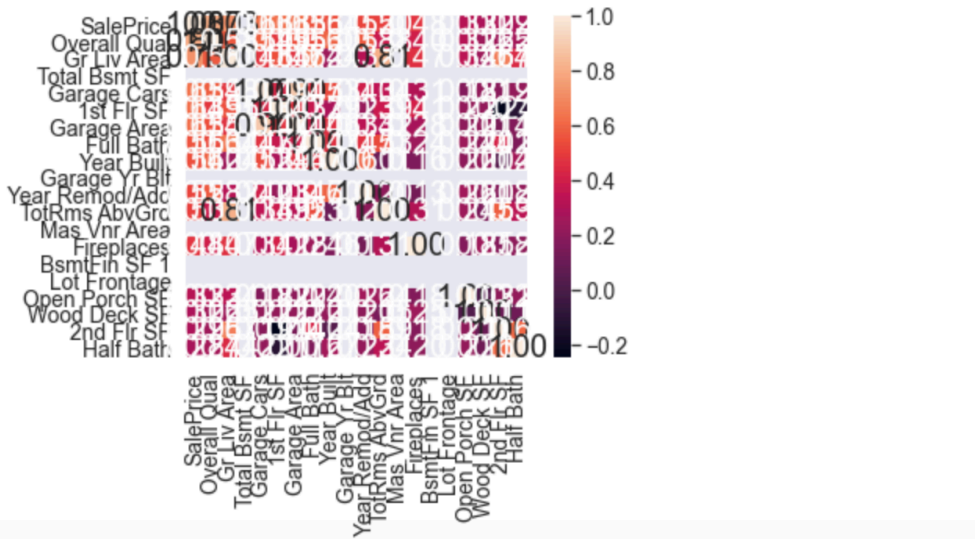
```
Out[27]: <AxesSubplot:xlabel='SalePrice'>
```



We need to find the most relevant columns to salesprice, we do test, first find top 20, then top 10

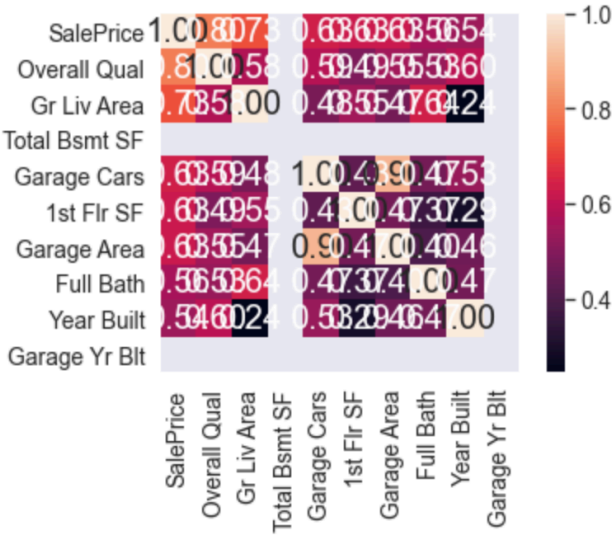
```
In [41]: i = 20
cols = df_train.corr().nlargest(i, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(df_train[cols].values.T)
sns.set(font_scale=1.25)
sns.heatmap(cm, cbar=True, annot=True, square=True,
            fmt='.2f', annot_kws={'size': 20},
          xticklabels=cols.values, yticklabels=cols.values )
```

```
Out[41]: <AxesSubplot:>
```



```
In [42]: i = 10
cols = df_train.corr().nlargest(i, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(df_train[cols].values.T)
sns.set(font_scale=1.25)
sns.heatmap(cm, cbar=True, annot=True, square=True,
            fmt='.2f', annot_kws={'size': 20},
          xticklabels=cols.values, yticklabels=cols.values )
```

```
Out[42]: <AxesSubplot:>
```



Feature Engineering

For discrete features, get_dummies in pandas is generally used for digitization, but this may not be enough in this competition, so the method I use below is to group by feature and calculate the average of SalePrice for each value of the feature And the median, and then sort and assign values based on this, as an example:

The feature of MSSubClass represents the type of house, and the data is grouped by it.

Combining the original features can usually produce unexpected results. However, there are many original features in this data set, and it is impossible to combine all of them one by one, so here we use Lasso for feature screening first, and select some of the more important features for combination

MS_SubClass			
120	213778.710280	200000.0	107
150	NaN	NaN	0
160	135690.000000	146000.0	65
180	101375.000000	91500.0	8
190	130710.185185	128000.0	27
20	186265.231434	157000.0	579
30	97787.642857	99700.0	70
40	126000.000000	133000.0	3
45	112857.142857	113000.0	7
50	136792.960526	130250.0	152
60	240684.922840	219355.0	324
70	160656.129032	148250.0	62
75	214266.666667	179500.0	15
80	168823.515625	165000.0	64
85	152896.296296	149900.0	27
90	146822.016667	143476.5	60

```
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["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,
```

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

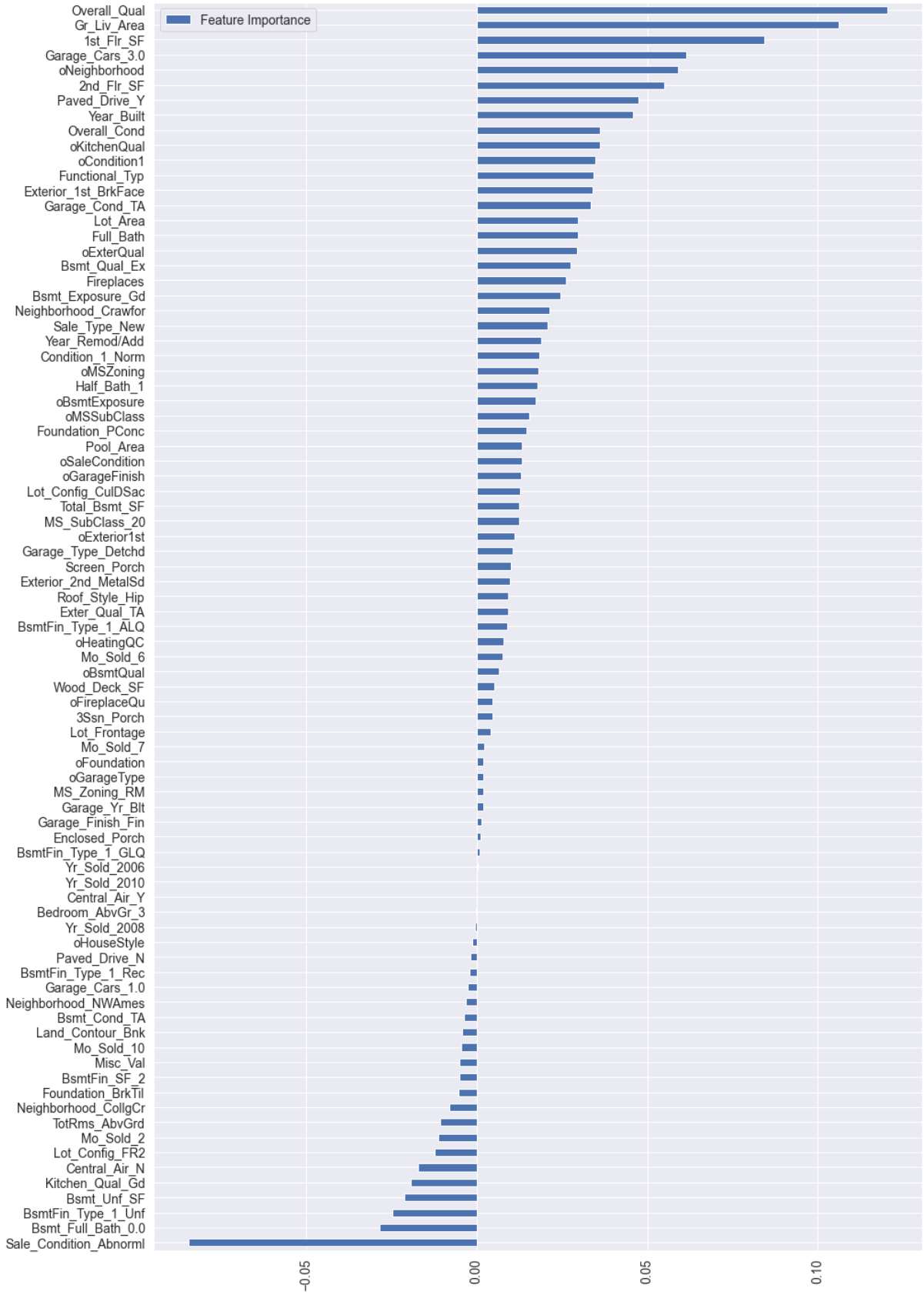
Methodology

Combining the original features can usually produce unexpected results. However, there are many original features in this data set, and it is impossible to combine all of them one by one, so here we use Lasso for feature screening first, and select some of the more important features for combination

```
In [446]: FI_lasso.sort_values("Feature Importance",asc
```

Out[446]:

	Feature Importance
Overall_Qual	0.120393
Gr_Liv_Area	0.105975
1st_Flr_SF	0.084210
Garage_Cars_3.0	0.061398
oNeighborhood	0.059006
...	...
Kitchen_Qual_Gd	-0.019401
Bsmt_Unf_SF	-0.021088
BsmtFin_Type_1_Unf	-0.024645
Bsmt_Full_Bath_0.0	-0.028470
Sale_Condition_Abnorml	-0.084337


Feature Importance
Overall_Qual
Gr_Liv_Area
1st_Flr_SF
Garage_Cars_3.0
oNeighborhood
2nd_Flr_SF
Paved_Drive_Y
Year_Built
Overall_Cond
oKitchenQual
oCondition1
Functional_Typ
Exterior_1st_BrkFace
Garage_Cond_TA
Lot_Area
Full_Bath
oExterQual
Bsmt_Qual_Ex
Fireplaces
Bsmt_Exposure_Gd
Neighborhood_Crawfor
Sale_Type_New
Year_Remod/Add
Condition_1_Norm
oMSZoning
Half_Bath_1
oBsmtExposure
oMSSubClass
Foundation_PConc
Pool_Area
oSaleCondition
oGarageFinish
Lot_Config_CulDSac
Total_Bsmt_SF
MS_SubClass_20
oExterior1st
Garage_Type_Detchd
Screen_Porch
Exterior_2nd_MetalSd
Roof_Style_Hip
Exter_Qual_TA
BsmtFin_Type_1_ALQ
oHeatingQC
Mo_Sold_6
oBsmtQual
Wood_Deck_SF
oFireplaceQu
3Ssn_Porch
Lot_Frontage
Mo_Sold_7
oFoundation
oGarageType
MS_Zoning_RM
Garage_Yr_Blt
Garage_Finish_Fin
Enclosed_Porch
BsmtFin_Type_1_GLQ
Yr_Sold_2006
Yr_Sold_2010
Central_Air_Y
Bedroom_AbvGr_3
Yr_Sold_2008
oHouseStyle
Paved_Drive_N
BsmtFin_Type_1_Rec
Garage_Cars_1.0
Neighborhood_NWAmes
Bsmt_Cond_TA
Land_Contour_Bnk
Mo_Sold_10
Misc_Val
BsmtFin_SF_2
Foundation_BrkTil
Neighborhood_CollgCr
TotRms_AbvGrd
Mo_Sold_2
Lot_Config_FR2
Central_Air_N
Kitchen_Qual_Gd
Bsmt_Unf_SF
BsmtFin_Type_1_Unf
Bsmt_Full_Bath_0.0
Sale_Condition_Abnorml
−0.05
0.00
0.05
0.10

PCA is a very important part, which greatly improves the final score. Because these new features I added are highly correlated with the original features, this may lead to strong multicollinearity (Multicollinearity), and PCA can just decorate the correlation. Because the purpose of using PCA here is not to reduce dimensionality, n_components uses dimensions that are similar to the original. This is the result of my multi-party experiment, that is, add XX features in the front, and then reduce to XX dimensions.

First define the cross-validation evaluation index of RMSE

In [454]:
```
# define cross validation strategy
def rmse_cv(model,X,y):
    rmse = np.sqrt(-cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=5))
    return rmse
```

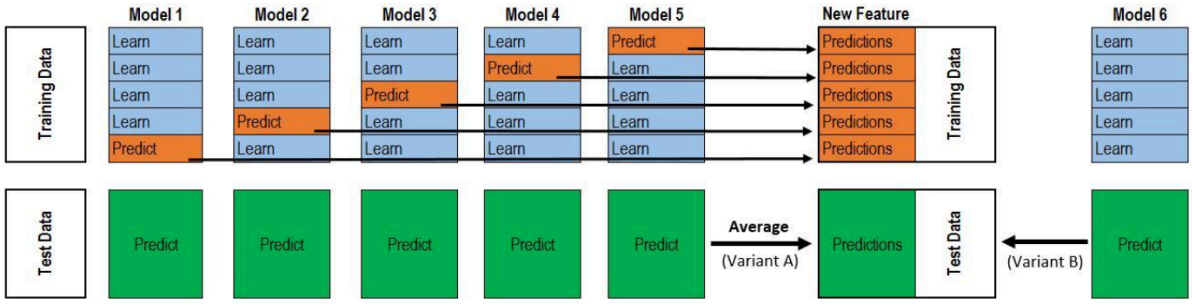
12 algorithms and 5-fold cross-validation are used to evaluate the baseline effect:
LinearRegression
Ridge
Lasso
Random Forrest
Gradient Boosting Tree
Support Vector Regression
Linear Support Vector Regression
ElasticNet
Stochastic Gradient Descent
BayesianRidge
KernelRidge
ExtraTreesRegressor

In [456]:
```
names = ["LR", "Ridge", "Lasso", "RF", "GBR", "SVR", "LinSVR", "Ela","SGD","Bay","Ker","Extra"]
for name, model in zip(names, models):
    score = rmse_cv(model, X_scaled, y_log)
    print("{}: {:.6f}, {:.4f}".format(name,score.mean(),score.std()))
```

```
LR: 0.138429, 0.0128
Ridge: 0.129480, 0.0130
Lasso: 0.137974, 0.0096
RF: 0.161097, 0.0104
GBR: 0.145741, 0.0102
SVR: 0.143059, 0.0198
LinSVR: 0.132926, 0.0093
Ela: 0.118150, 0.0126
SGD: 0.157843, 0.0054
Bay: 0.117425, 0.0129
Ker: 0.116635, 0.0126
Extra: 0.155942, 0.0119
```

After many long rounds of testing, **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.

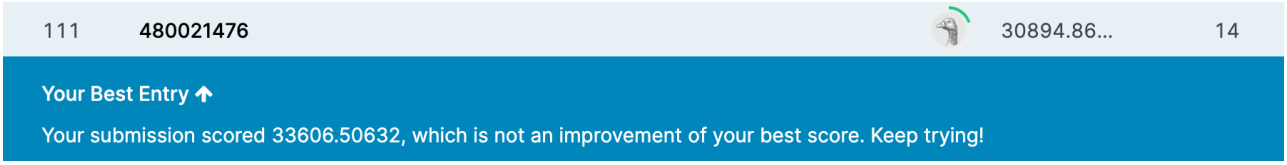
After many long rounds of testing, these six models were finally selected

```
In [463]: # stack_model = stacking(mod=[ker],meta_model=ker)
          stack_model = stacking(mod=[lasso,ridge,svr,ker,ela,bay],meta_model=ker)
```

```
{'alpha': 0.0004, 'max_iter': 10000} 0.11887620008544973
                                params  mean_test_score  std_test_score
0  {'alpha': 0.0004, 'max_iter': 10000}         0.118876        0.003142
1  {'alpha': 0.0005, 'max_iter': 10000}         0.118887        0.003074
2  {'alpha': 0.0007, 'max_iter': 10000}         0.119100        0.002933
3  {'alpha': 0.0006, 'max_iter': 10000}         0.118946        0.003019
4  {'alpha': 0.0009, 'max_iter': 10000}         0.119760        0.002834
5  {'alpha': 0.0008, 'max_iter': 10000}         0.119446        0.002878
{'alpha': 35} 0.11709027873708948
          params  mean_test_score  std_test_score
0  {'alpha': 35}         0.117090        0.002970
1  {'alpha': 40}         0.117116        0.002951
2  {'alpha': 45}         0.117175        0.002934
3  {'alpha': 50}         0.117259        0.002920
4  {'alpha': 55}         0.117361        0.002908
5  {'alpha': 60}         0.117477        0.002897
6  {'alpha': 65}         0.117604        0.002888
7  {'alpha': 70}         0.117739        0.002880
8  {'alpha': 80}         0.118030        0.002865
```

Validation set Kaggle results

submission.csv 4 hours ago by 480021476 t	30894.86877
submission.csv 4 hours ago by 480021476 t	33566.56261
submission.csv 4 hours ago by 480021476 t6	32783.19914



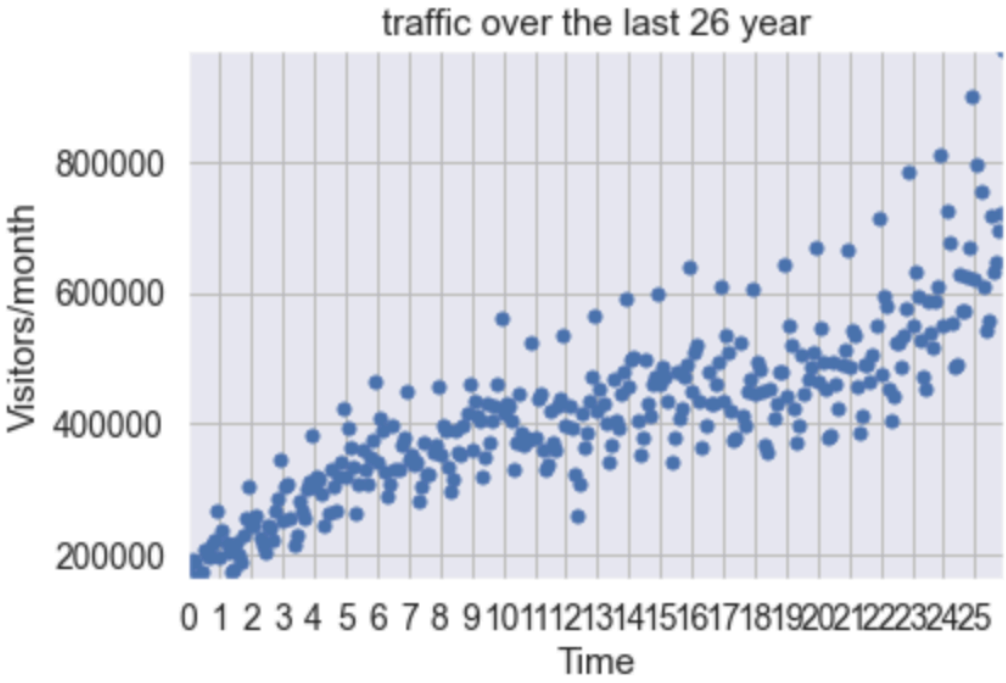
Final analysis, conclusion, limitations

It is important to understand the data, and it is important to clean and transform the data. There are both discrete and continuous features, and there are a lot of missing values. Fortunately, the contestant provided the file data_description.txt, which describes the meaning of each feature. After understanding the content, most of the missing values can be smoothly interpolated.

If I have more time and more data, I want to try the neural network method. In the training process, as long as there are enough input x and output y, a better neural network model can be trained. This model is In similar housing price prediction problems, more accurate results can be obtained.

Task B forecast visitors

First transform the data, turn the month into a number, and then visualize the effective data after the conversion to find the law. First transform the data, turn the month into a number, and then visualize the effective data after the conversion to find the law.



We have to choose an algorithm to predict future visits, which is obviously supervised learning.

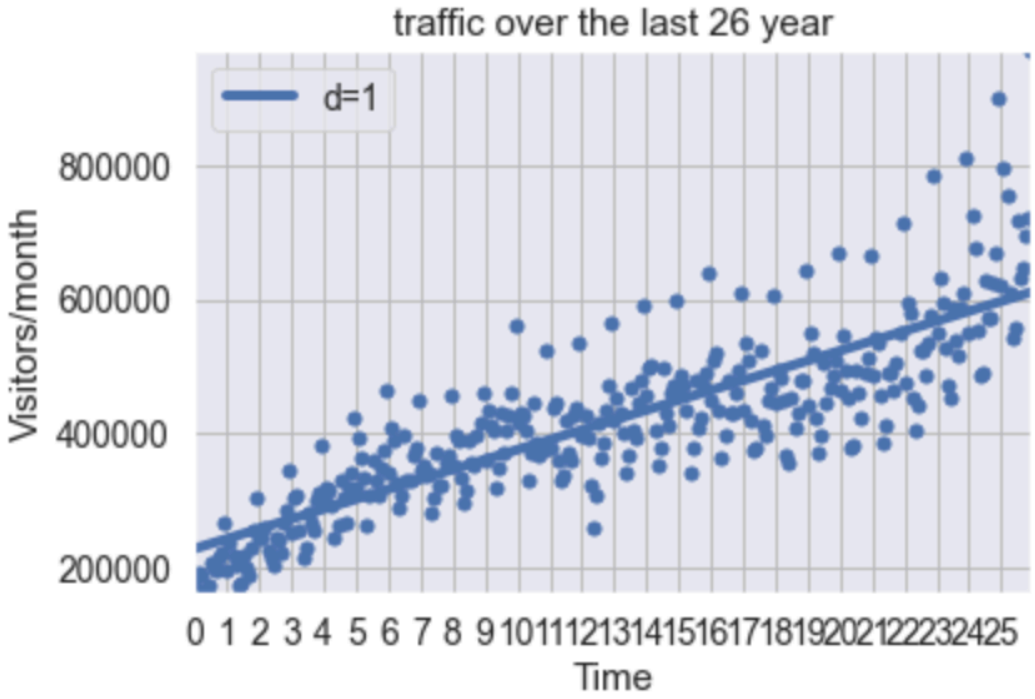
Before building our first model, we need to design an evaluation function to judge what kind of model is good. That is, the error function, which can be calculated like this,

```
def error(f, x,y):
    return sp.sum( (f(x) - y)**2 )
```

and evaluated by the square of the difference between the predicted value of the model and the true value (the training sample has been provided)

Linear regression

Linear regression: It is easy to know that this is actually a fitting problem. Fit these data to the best model (ie a function, and then use this function to predict new data). Starting from the simplest case, we first go to a straight line to fit the data. SciPy provides the function polyfit(), as long as the data x and y and the order of the polynomial are given (a straight line is a function of 1st order), it can find the function of the model so that the previously defined error function is minimized (only the smallest error is the surface The best model). Type in: fp1,residuals,rank,sv,rcond=sp.polyfit(x,y,1,full=True)




```
Model parameters of fp1: [  1229.75866859 228179.89882854]
Error of the model of fp1: [1.62777855e+12]
error= 1627778546842.062
```

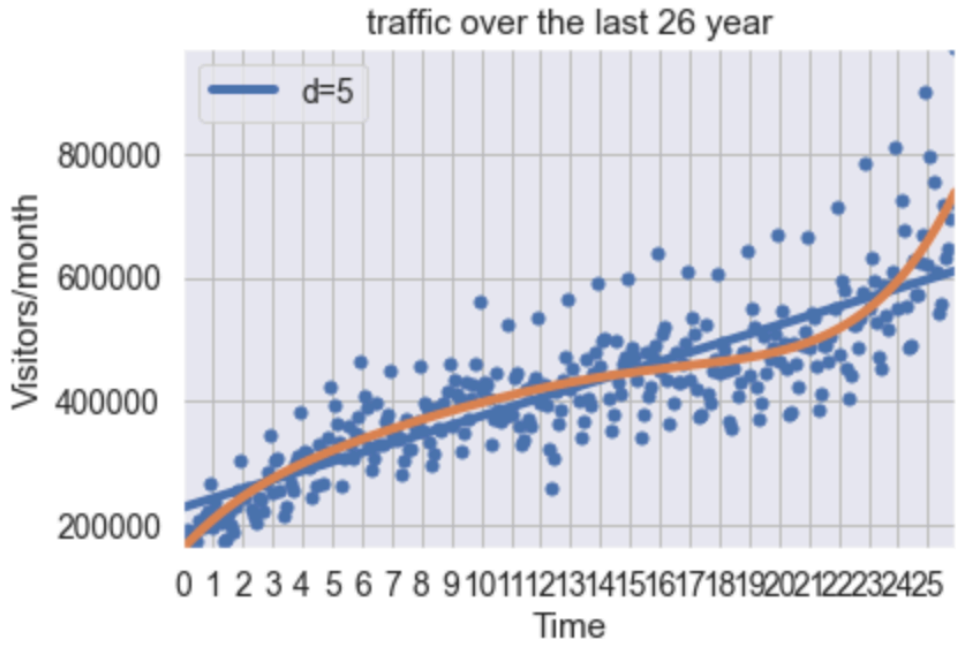
forecasting results for 24 months of monthly number of visitors following the last period in the dataset:

In [469]:
```
for i in range(312, 312+24):
    r = fp1[0]*i + fp1[1]
    print('future month ', i, ' predict visitors:',r)
```

```
future month  312  predict visitors: 611864.6034297964
future month  313  predict visitors: 613094.3620983902
future month  314  predict visitors: 614324.120766984
future month  315  predict visitors: 615553.8794355777
future month  316  predict visitors: 616783.6381041715
future month  317  predict visitors: 618013.3967727652
future month  318  predict visitors: 619243.155441359
future month  319  predict visitors: 620472.9141099527
future month  320  predict visitors: 621702.6727785466
future month  321  predict visitors: 622932.4314471403
future month  322  predict visitors: 624162.1901157341
future month  323  predict visitors: 625391.9487843278
future month  324  predict visitors: 626621.7074529217
future month  325  predict visitors: 627851.4661215154
future month  326  predict visitors: 629081.2247901092
future month  327  predict visitors: 630310.9834587029
future month  328  predict visitors: 631540.7421272966
future month  329  predict visitors: 632770.5007958905
future month  330  predict visitors: 634000.2594644842
future month  331  predict visitors: 635230.018133078
future month  332  predict visitors: 636459.7768016717
future month  333  predict visitors: 637689.5354702655
future month  334  predict visitors: 638919.2941388593
future month  335  predict visitors: 640149.0528074531
```

Polynomial regression: A straight line is a first-order function, obviously not optimized enough, and then we start to consider the second-order curve, the third-order curve... Up to the fifth-order curve, it can find the function of the model, so that the previously defined error function is minimized

```
# n次系数
f2p = sp.polyfit(x, y, 5)
print('f2p=',f2p)
f2 = sp.poly1d(f2p)
print(colored('error=','red'),error(f2,x,y))
fx2 = sp.linspace(0,x[-1], 1000)
# generate X-values for plotting
plt.plot(fx2, f2(fx2), linewidth=4)
plt.legend(["d=%i" % f2.order], loc="upper left")
```



the error function value is lower than line-function model.

forecasting results for 24 months of monthly number of visitors following the last period in the dataset:

```
In [470]: yvals=np.polyval(f2p, list(range(312, 312+24)))
          i = 311
          for y in yvals:

              print('future month ', i+1, 'n-degree-coefficient-curve predict visitors:',y)
              i += 1
```

```
future month  312 n-degree-coefficient-curve predict visitors: 746489.1380864521
future month  313 n-degree-coefficient-curve predict visitors: 755001.3900901556
future month  314 n-degree-coefficient-curve predict visitors: 763719.5045037749
future month  315 n-degree-coefficient-curve predict visitors: 772647.192118837
future month  316 n-degree-coefficient-curve predict visitors: 781788.2055798827
future month  317 n-degree-coefficient-curve predict visitors: 791146.3396148467
future month  318 n-degree-coefficient-curve predict visitors: 800725.431265444
future month  319 n-degree-coefficient-curve predict visitors: 810529.3601175563
future month  320 n-degree-coefficient-curve predict visitors: 820562.048531628
future month  321 n-degree-coefficient-curve predict visitors: 830827.4618730447
future month  322 n-degree-coefficient-curve predict visitors: 841329.6087425204
future month  323 n-degree-coefficient-curve predict visitors: 852072.541206494
future month  324 n-degree-coefficient-curve predict visitors: 863060.3550275047
future month  325 n-degree-coefficient-curve predict visitors: 874297.1898945894
future month  326 n-degree-coefficient-curve predict visitors: 885787.2296536611
future month  327 n-degree-coefficient-curve predict visitors: 897534.702537907
future month  328 n-degree-coefficient-curve predict visitors: 909543.8813981672
future month  329 n-degree-coefficient-curve predict visitors: 921819.0839333211
future month  330 n-degree-coefficient-curve predict visitors: 934364.6729206829
future month  331 n-degree-coefficient-curve predict visitors: 947185.056446383
future month  332 n-degree-coefficient-curve predict visitors: 960284.6881357585
future month  333 n-degree-coefficient-curve predict visitors: 973668.0673837338
future month  334 n-degree-coefficient-curve predict visitors: 987339.7395852176
future month  335 n-degree-coefficient-curve predict visitors: 1001304.2963654867
```

To visually compare the pros and cons of algorithms. This time Use simple linear regression and polynomial regression to solve the problem. In simple situation like this: dataset is small, we can use this way.

In the future, you can also try to deal with the number of visitors in a time series method, if the dataset is more related with time and If the trend after visualization is still not obvious enough。