Answers to Lab 2

95-791 Data Mining (Fall 2021)

Prof. Gongora-Svartzman

Date: Friday, October 29th, 2021

Topics covered in this Lab:

- Linear and Polynomial Regression with scikit-learn
- Ridge and Lasso Regression
- Cross Validation

Changing the author field and file name.

(a) Change the `name:` field on the Rmd document from Your Name Here to your own name.

(b) Rename this file to "Lab2_F21_YourHameHere.ipynb", where YourNameHere is changed to your own name.

Installing and loading packages

Before you begin this Lab make sure you have installed all the required libraries. Load the libraries as indicated below.

You only need to install libraries once. Once they're installed, you may use them by **importing** the libraries using the `import` command. For today's lab, you'll want to run the following code

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import plotly.express as px
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold
, cross_val_score
from sklearn.datasets import load_diabetes
%matplotlib inline
plt.style.use('seaborn-white')
```

1. Data Processing

> For Lab you might want to have your lecture slides and the ISRL textbook (An Introduction to Statistical Learning) open (Chapters 3, 4, and 6) as you go through the exercises.

In today's Lab we are going to switch from using statsmodels to [scikitlearn (https://scikit-learn.org/stable/)](https://scikit-learn.org/stable/). Scikit-learn works with vectors rather than formulas to compute our models. Follow along the next exercises to learn more.

1)a) Begin by loading the dataset provided into a dataframe called `kc_housing` . Print the DESCR of the dataset.

In [2]:
```
kc_housing = pd.read_csv("kc_house_data.csv")
kc_housing.head()
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

1)b) Verify if there are any missing values. If they are deal with them appropriately. Additionally, drop any columns that might not be relevant to a regression model.

In [3]:
```
kc_housing.isnull().sum(axis=0)
```

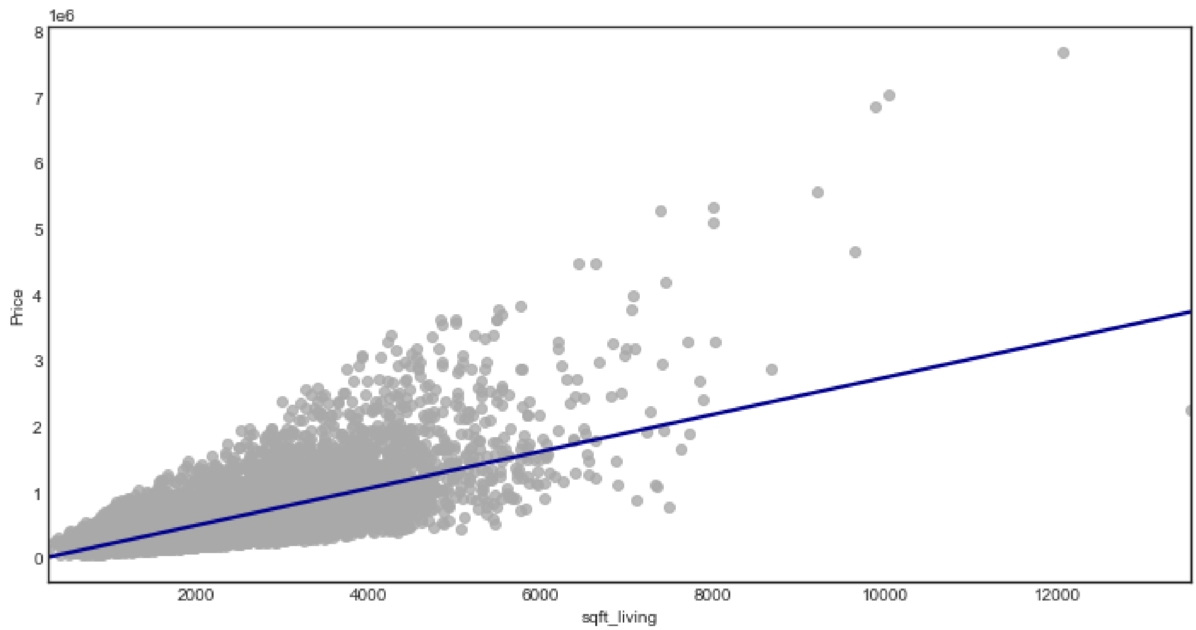
Out[3]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [4]:
```
kc_housing = kc_housing.drop(['lat','long','id','date'],axis=1)
kc_housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21613 non-null  float64
 1   bedrooms       21613 non-null  int64  
 2   bathrooms      21613 non-null  float64
 3   sqft_living    21613 non-null  int64  
 4   sqft_lot       21613 non-null  int64  
 5   floors         21613 non-null  float64
 6   waterfront     21613 non-null  int64  
 7   view           21613 non-null  int64  
 8   condition      21613 non-null  int64  
 9   grade          21613 non-null  int64  
 10  sqft_above     21613 non-null  int64  
 11  sqft_basement  21613 non-null  int64  
 12  yr_built       21613 non-null  int64  
 13  yr_renovated   21613 non-null  int64  
 14  zipcode        21613 non-null  int64  
 15  sqft_living15  21613 non-null  int64  
 16  sqft_lot15     21613 non-null  int64  
dtypes: float64(3), int64(14)
memory usage: 2.8 MB
```

1)c) We will use `price` as our target variable. Graph the predictor `sqft_living` against the target variable. Use regplot() for this and describe the relationship between those two variables.

In [5]:
```
sns.axes_style("whitegrid")
plt.figure(figsize=(12,6))
lmplot = sns.regplot(x="sqft_living",y="price",data=kc_housing, ci=None, order=1,
             scatter_kws={"color": "darkgrey"}, line_kws={"color": "darkblue"})
lmplot.set(xlabel='sqft_living',
           ylabel='Price')
plt.show(lmplot)
```



For Price vs sqft_living, there seems to be an upwards positive trend, meaning as the sqft_living increases so does the price of the home. A linear regression was drawn over it, and it might not provide the best approximation, as many data point are far from the linear regression model.

--> There can be many different answers for this question.

1)d) Split your dataframe into `X` and `y` dataframe. Print your taget variable.

In [6]:
```
X = kc_housing[kc_housing.columns.difference(['price'])]
y = kc_housing['price']
y
```

Out[6]:
```
0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           ...
21608    360000.0
21609    400000.0
21610    402101.0
21611    400000.0
21612    325000.0
Name: price, Length: 21613, dtype: float64
```

1)d) During the lectures we talked about spliting our dataset into training and testing so that we can validate our models. One easy way of doing this is using [train_test_split (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) from scikit-learn. Split your `X` and `y` from the kc_housing dataset into `X_train` , `X_test` , `y_train` and `y_test` . Use a 75-25 ratio for the split and a random_state=1. Print out your y_test.

**Hint: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=your_number, random_state=your_number)

In [7]:
```
housing_X_train, housing_X_test, housing_y_train, housing_y_test = train
_test_split(X, y,

test_size=0.25, random_state=1)
housing_y_test
```

Out[7]:
```
15544     459000.0
17454     445000.0
21548    1057000.0
3427      732350.0
8809      235000.0
           ...
12416     680000.0
8253      267500.0
4251      725000.0
11404     253500.0
13206     324950.0
Name: price, Length: 5404, dtype: float64
```

2. Linear and Plynomial Regression with Scikit-Learn

Linear Regression with scikit-learn requires you to have your data in vector (array) form rather than formulas (like statsmodels. Look at the steps below to get an idea.

We import LinearRegression from scikit-learn:

```
from sklearn.linear_model import LinearRegression
```

Most models on scikit-learn are python classes, which means we'll have to create an object of this class, and we'll have access to its attributes and methods.

```
lm = LinearRegression()
```

The next step is fitting our dataset to our `lm` model. So far its just an empty object of class LinearRegression. Pretend we already have a dataset `df` with all our data. We need to separate our dataset into `X` and `y` before fitting it to our model.

```
X = df.drop(['y'],axis=1)
y = df['y']
```

Once separated we can use our `X` and `y` in our `lm` model:

```
lm.fit(X,y)
```

From the documentation of [LinearRegression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression#sklearn.linear_model.LinearRegression)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression#sklearn.linear_model.LinearRegression) you'll see that there are a few methods associated with this class:

- fit(X, y[, sample_weight])
- get_params([deep])
- predict(X)
- score(X, y[, sample_weight])
- set_params(**params)

As well as a few attributes:

- coef_
- rank_
- singular_
- intercept_
- n_features*in*
- feature_names*in*

There are also parameters you can modify when creating your object from class LinearRegression.

2)a) Use your training dataset to fit them into a LinearRegression (with scikit-learn) and print out the coeficients of your model.

In [8]:
```
lm = LinearRegression(normalize=True)
lm.fit(housing_X_train,housing_y_train)
print(lm.coef_)
```

```
[ 4.08845023e+04 -3.53662975e+04  2.02030891e+04  3.04208735e+04
  1.19225916e+05  7.51326399e+01  8.07150245e+01  8.05312427e+01
  2.39235724e+01  2.03182891e-03 -5.15883752e-01  4.02406063e+04
  5.69730425e+05 -3.46608156e+03  1.39022598e+01  2.77649140e+01]
```

2)b) You can calculate the R^2 of your model by using the method `score` from LinearRegression. Use your model from 2)a) to print out the R^2 of your training set and the R^2 of your testing set.

In [9]:
```
print("Training R-squared: ", lm.score(housing_X_train,housing_y_train))
print("Testing R-squared: ", lm.score(housing_X_test,housing_y_test))
```

```
Training R-squared:  0.6538332945773833
Testing R-squared:  0.6516710914367817
```

2)c) How good is your model based on your answer from 2)b)?

Better than a coin flip! But could be better.

2)d) Use the [`mean_squared_error` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) method (package already imported) to calculate your training and testing MSE.

**Hint: You'll have to calculate the predicted values of your model, both with your train and test datasets, and then calcualte their corresponding MSEs.

In [10]:
```
housing_train_pred = lm.predict(housing_X_train)
housing_test_pred = lm.predict(housing_X_test)

print('Training Mean squared error: %.2f'% mean_squared_error(housing_y_
train, housing_train_pred))
print('Testing Mean squared error: %.2f'% mean_squared_error(housing_y_t
est, housing_test_pred))
```

```
Training Mean squared error: 43720093297.38
Testing Mean squared error: 55774910415.01
```

2)e) Let's now try a polynomial regression with scikit-learn. We must first transform our X's into [polynomial features (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html). Follow the instructions below and print the coeficients and the test MSE for your polynomial regression (degree=3) model.

Polynomial features is a python class, therefore we must create an object of this class. When creating this object we must specify the degrees of our polynomial.

```
poly_model = PolynomialFeatures(degree=your_degree)
```

Once we have our model, we must transform our X's into polynomial form

```
X_poly_train = poly_model.fit_transform(diabetes_X_train)
```

After this step you can follow the same steps as your LinearRegression model. The difference is that you'll plug in your transformed polynomial features instead of your X_train or X_test.

In [11]:
```
poly_model = PolynomialFeatures(degree=3)

X_poly_train = poly_model.fit_transform(housing_X_train)
X_poly_test = poly_model.fit_transform(housing_X_test)

lm_poly = LinearRegression(normalize=True)
lm_poly.fit(X_poly_train, housing_y_train)
housing_y_pred_poly = lm_poly.predict(X_poly_test)

print('Coefficients:', lm_poly.coef_)
print('Mean squared error: %.2f'% mean_squared_error(housing_y_test, hou
sing_y_pred_poly))
```

```
Coefficients: [-5.38347702e+02 -1.08679097e+10  7.78243059e+09  1.57740
193e+10
 -4.19997810e+10  2.62887591e+10  1.02968423e+11  1.02951008e+11
 -1.02993593e+11  1.14633960e+08 -2.92619835e+06 -2.17784844e+06
 -3.54756203e+10  2.48615704e+14  1.86264616e+08  3.70040260e+07
 -7.19721948e+08 -1.53474977e+07  8.19569953e+06  1.92720254e+07
 -8.42910936e+05 -4.47927866e+06 -2.59945905e+10 -2.59946167e+10
  2.59946254e+10 -2.55987070e+04  7.19106417e+02 -1.21348870e+03
 -5.88148670e+06  9.30876317e+13  1.26170606e+06  6.71377024e+03
  1.96186936e+05 -3.47504210e+06 -1.43030727e+07  1.21125229e+07
  1.08547031e+07  6.47502565e+08  6.47513125e+08 -6.47506838e+08
  1.23634766e+04 -2.22367968e+03  3.09962343e+03  7.67820833e+06
  4.41365100e+13 -6.38199649e+05 -2.76907193e+04 -1.46236522e+05
  7.84030612e+05 -1.14718727e+07 -9.62646326e+06  2.81227301e+09
  2.81224322e+09 -2.81224126e+09  1.02374446e+04 -4.49472661e+02
  1.25034129e+02 -4.93470244e+06  1.23145106e+13  1.26199656e+05
 -2.78976637e+04 -3.23750544e+05  7.04986178e+07  1.58414863e+07
 -1.88905600e+10 -1.88903922e+10  1.88904091e+10 -2.95886711e+04
 -3.62836961e+02  2.80393991e+03  5.90600798e+07 -1.33982366e+13
 -5.08828076e+05 -5.37975572e+04  8.65699635e+05 -1.43215499e+07
 -3.24396652e+09 -3.24402338e+09  3.24399373e+09  1.14253345e+04
 -8.61971949e+02 -2.43206394e+01 -2.90468873e+06  9.48438627e+12
  8.63374491e+05  5.35837994e+02 -5.51763337e+05 -1.36603250e+07
 -8.89758082e+06  1.09070326e+07 -5.92050469e+06  4.67259158e+04
  8.34707493e+04 -2.42979515e+09  6.66862590e+09  9.88216399e+06
  1.11456088e+07 -3.65373991e+04  4.76293454e+06 -7.51622131e+06
 -5.92047120e+06  4.67258977e+04  8.34732526e+04 -2.42974640e+09
 -1.65747968e+11  9.88220499e+06  1.11455977e+07 -3.61791080e+04
  2.75333991e+06  5.92052925e+06 -4.67247203e+04 -8.34731824e+04
  2.42973124e+09 -1.06372462e+10 -9.88239261e+06 -1.11455188e+07
  3.70517335e+04 -5.44898276e+01  1.46685487e-01 -3.66500378e-01
  5.15192973e+04  3.13496742e+08 -2.02489023e+03 -7.53281107e+00
 -2.29554469e+03  5.84416982e-03  4.69555461e-04  8.34431312e+02
  1.46544739e+09 -4.45033957e+00 -4.59030410e-01  5.98672425e+01
 -3.34806265e-03 -6.43910074e+02  2.99710327e+09  2.24681957e+01
 -3.68134641e+00  4.39327860e+01 -7.11134992e+06 -8.97384942e+12
 -4.56998608e+05  3.32056901e+03  7.32127041e+05 -1.16830780e+14
  1.98012858e+09 -6.36721860e+10 -4.46821079e+07 -1.17763607e+04
  2.31945159e+02 -3.35811058e+03 -4.78632149e+02 -7.47932828e+02
  7.40773902e+03 -6.44475288e+03  3.67062276e+03  1.34228439e+04
 -9.50308194e+03 -9.27377666e+03 -1.78205013e+09 -1.78205019e+09
  1.78205014e+09  6.73830799e+00  4.15930500e-01 -7.39539896e-01
 -7.97843094e+03 -1.02664161e+05  1.66929409e+02  3.43284590e+01
  1.53608674e+02 -5.91554575e+02 -9.70104942e+03 -2.79346082e+03
  1.08385022e+04 -2.49943205e+09 -2.49943206e+09  2.49943207e+09
 -2.29088635e+01 -2.12065332e-01  2.68265237e-01  3.60586463e+03
  5.58583719e+04  1.08123558e+02 -7.13139596e+00 -8.62144031e+01
  4.14060587e+01 -1.87905453e+04 -6.28459184e+03 -6.01146438e+09
 -6.01146439e+09  6.01146437e+09  2.83946086e+00 -1.07419506e-02
 -9.55523560e-01 -7.30426890e+03 -3.59785045e+05  3.57261016e+02
  3.58237561e+01 -2.02801803e+02  6.60813398e+03 -2.60962288e+04
  2.10178896e+08  2.10179011e+08 -2.10178884e+08 -4.95718952e+00
  1.42567861e+00 -7.25153822e-01 -1.73790684e+04  2.73240786e+05
  6.52201888e+02 -7.06393888e+00 -2.30632395e+00  3.82479354e+03
 -7.77476687e+08 -7.77476744e+08  7.77476733e+08 -3.16353760e+01
 -6.96931229e-01  6.55035800e-01 -1.45761609e+04  6.27998659e+04
  4.99666991e+02 -5.82207659e+00  3.56871846e+01  1.25838283e+06
```

```
 -7.26624686e+06 -1.17360224e+06 -1.95814312e+06  3.36879164e+02
  6.12319883e+03 -2.38952644e+09 -8.73501580e+09  2.29875911e+05
  7.88799155e+06 -5.78207607e+04 -8.52462965e+06  8.60941037e+06
 -1.95814312e+06  3.36879564e+02  6.12319862e+03 -2.38952647e+09
 -8.73501567e+09  2.29877058e+05  7.88799160e+06 -5.78205129e+04
 -8.47806464e+04  1.95814314e+06 -3.36879223e+02 -6.12319875e+03
  2.38952644e+09  8.73501618e+09 -2.29876994e+05 -7.88799163e+06
  5.78204237e+04  6.05401522e-03  9.78137556e-04 -8.01789817e-04
  3.10906162e+01 -7.65000997e+02 -7.85745591e-01 -2.61447174e-02
  2.78822376e-01 -3.18510432e-06  2.51946688e-06  4.69657574e-01
  1.52428037e-01 -1.32370156e-02 -4.96094126e-04 -7.06064954e-03
  3.04624298e-06 -4.19995413e-01  2.31223340e+01 -6.75775079e-03
  9.24396369e-04  1.25328556e-02  8.33131377e+03 -4.65747418e+05
  3.57920395e+02  1.37399571e+01  5.39008709e+01 -9.30869767e+13
  7.11546204e+03  5.97141480e+02 -6.79405648e+03 -1.98257695e+00
  3.59565418e-01 -1.28197725e+01  1.11124346e+00 -9.81235941e-02
 -8.70648944e-01  3.38664243e+02 -6.97898030e+03 -7.79324295e+03
 -1.19011328e+03  1.17209456e+09  1.17209457e+09 -1.17209456e+09
  1.44669815e+00  3.28666533e-01 -3.51363906e-01  1.54850673e+03
  4.24127372e+04  1.12774647e+00 -3.75351627e+00  3.57096521e+01
 -2.92317071e+03 -2.03135722e+03  3.51982978e+03 -6.94942980e+08
 -6.94943009e+08  6.94943004e+08 -2.68726877e+00 -8.40098935e-02
  4.70327133e-01  2.99661908e+03  4.13839573e+05 -1.24814253e+02
 -5.92097831e+00  1.48600584e+02 -4.34897336e+03  2.18137583e+03
 -1.01979860e+09 -1.01979865e+09  1.01979862e+09 -3.77081075e+00
  9.07980166e-01 -9.49498542e-01 -1.39530950e+04 -1.62240761e+04
 -2.47202975e+02  1.57505848e+01 -1.18028757e+02  4.36118053e+02
 -6.90888367e+08 -6.90888344e+08  6.90888347e+08  8.00380738e+00
  1.09412584e-01 -2.47842609e-01 -3.62958988e+03 -2.76839448e+05
 -2.51420482e+02  1.10875110e-02 -1.05774062e+02  1.96003052e+06
  2.86494279e+06 -1.16961270e+06  4.03185647e+06 -1.82580883e+04
  2.69311744e+04  2.22813132e+09  2.40410957e+09 -9.84209060e+05
 -5.36581592e+05 -3.52652197e+04  9.04912276e+05 -1.14494499e+05
  4.03185651e+06 -1.82580879e+04  2.69311745e+04  2.22813129e+09
  2.40410894e+09 -9.84209576e+05 -5.36581582e+05 -3.52653168e+04
 -7.90417815e+05 -4.03185651e+06  1.82580876e+04 -2.69311738e+04
 -2.22813129e+09 -2.40410988e+09  9.84209292e+05  5.36581602e+05
  3.52652587e+04  3.35739658e-02 -4.19775459e-04  7.26438257e-04
 -1.54799832e+01  5.52776349e+02  3.71439463e-01 -3.72383883e-03
 -1.34396882e-01  1.22177272e-06  6.87365526e-07 -2.89324232e-01
  5.25727519e+01  1.76037054e-02  3.56407343e-04  2.23095719e-02
 -6.18185906e-06  3.49027675e-01 -2.77954421e+01  7.44095270e-04
 -1.09793696e-03 -3.16179233e-02  8.41348129e+02  2.62329064e+06
 -1.43268242e+02 -3.62466295e+00 -7.53480875e+01 -4.41366091e+13
  1.32850057e+04 -1.38307452e+02  6.38845843e+02 -1.05443894e+01
  2.70144594e-01  6.93567545e+00 -5.65263295e-01  2.88556558e-01
  6.77621936e-01  3.60546124e+03 -8.05313756e+02  4.11984119e+02
  4.42842015e+09  4.42842015e+09 -4.42842014e+09 -5.63270818e+00
  1.21166416e-01 -2.52643751e-02 -8.04406780e+03  7.90224538e+04
  1.77492229e+02 -2.60970232e+01 -1.19480496e+01  3.13591818e+04
  1.51700054e+04  3.02120298e+06  3.02112966e+06 -3.02113484e+06
 -2.21679087e+01 -4.51394362e-01  8.99890693e-01 -8.19993303e+02
 -1.44722998e+05  2.68455387e+02 -6.35137041e+01  1.09479626e+02
 -5.89333644e+03 -1.33711159e+09 -1.33711156e+09  1.33711159e+09
  7.32186470e+00 -9.82273051e-01  1.02159843e+00 -3.81816876e+03
  3.93142823e+04  1.68961769e+02 -2.58633740e+01  9.52174548e+01
 -2.13335912e+05  2.40152352e+06  3.07354231e+05  2.93336208e+06
```

```
  3.12912063e+04  3.79930154e+04 -1.31368741e+09  3.89311925e+09
  3.41614747e+06 -1.66202322e+06 -9.86533416e+04  2.61485937e+06
 -2.52084102e+06  2.93336205e+06  3.12912052e+04  3.79930162e+04
 -1.31368741e+09  3.89311877e+09  3.41614762e+06 -1.66202326e+06
 -9.86530408e+04 -9.40183457e+04 -2.93336209e+06 -3.12912055e+04
 -3.79930163e+04  1.31368744e+09 -3.89311868e+09 -3.41614763e+06
  1.66202324e+06  9.86530195e+04  3.38776192e-02  1.48776324e-05
 -8.20440954e-05 -1.52836748e+01  8.48800655e+00 -2.34543753e-01
  3.26324028e-02 -1.00350957e-01  7.57195250e-09 -1.64757187e-06
 -3.54658102e-01  2.13169394e+01  3.01005608e-02  4.48725432e-04
  4.03996312e-03  1.63284860e-06  1.63192781e-01 -8.31009189e+00
 -9.69135411e-03  8.53289792e-04 -1.14716494e-03  8.74007119e+02
  2.63367051e+05  3.34953053e+02  1.49553377e+01  4.43104115e+01
 -1.23149861e+13 -2.51481800e+03 -3.22215346e+02  4.86136909e+03
 -2.03585029e+01 -9.24104301e-01 -5.08384177e-01  1.13205789e+00
  2.82918306e-01  1.65371189e+00 -6.10825626e+04 -2.24437059e+04
  1.15572371e+10  1.15572370e+10 -1.15572371e+10  6.21130426e+01
 -1.81203986e+00 -1.08766101e-01  2.41141197e+03 -4.92780083e+04
  7.50385705e+02 -1.66571679e+01 -7.30089153e+02  1.30030275e+04
  1.89259492e+08  1.89259550e+08 -1.89259530e+08 -1.24446150e+01
  1.19105676e-01  4.37531795e-02  2.15622886e+04 -1.26135450e+05
  9.27305244e+02  1.48144147e+01 -1.80618618e+02 -2.61093247e+06
 -1.21514322e+07 -4.42901508e+06  3.39134225e+06 -3.81129794e+03
 -1.02728138e+05 -5.04117308e+09  4.40372236e+10 -6.11547243e+06
  1.47840854e+07 -2.94598712e+05 -9.54049963e+06  2.50055208e+06
  3.39134223e+06 -3.81129775e+03 -1.02728139e+05 -5.04117305e+09
  4.40372241e+10 -6.11547431e+06  1.47840853e+07 -2.94600380e+05
  7.03994758e+06 -3.39134231e+06  3.81129782e+03  1.02728139e+05
  5.04117310e+09 -4.40372239e+10  6.11547291e+06 -1.47840854e+07
  2.94600239e+05  1.57500940e-02 -8.12073582e-04  6.79420957e-04
 -3.42774901e+01  4.89607340e+02  1.31961589e+00 -4.52075605e-02
  2.75950136e-01 -4.18942442e-07  1.09159569e-05 -5.37052690e-01
 -5.34325969e+01 -1.83661699e-02  2.96411944e-03  4.08158120e-03
 -9.69174640e-06  7.09664094e-01 -3.66589739e+01  1.84847996e-02
 -2.36460930e-03 -2.89832408e-02 -9.31073417e+03  1.43196596e+06
 -2.62298268e+02  1.81437503e+00 -5.97368119e+02  1.33970912e+13
 -1.74379337e+03 -8.37848186e+02  1.16699884e+04 -3.89818540e+01
 -6.97841483e-01  6.61979097e+00  7.24262231e-01  5.48199551e-01
 -4.47352593e+00 -3.67967359e+03  3.28400755e+08  3.28400759e+08
 -3.28400753e+08  8.97861746e+00  4.72167191e-01 -5.61153869e-01
  1.03685737e+04  6.59002989e+04  5.37390009e+01  1.51392406e+00
  1.45513426e+02 -1.20894598e+06 -2.35441365e+06  3.52230201e+05
 -8.50520932e+05  8.95645909e+03  5.70673772e+03 -6.54312617e+08
  1.47657648e+09  6.20655124e+05  1.98716914e+06  6.71704513e+04
 -1.14546762e+06  2.88751867e+05 -8.50520916e+05  8.95645972e+03
  5.70673704e+03 -6.54312610e+08  1.47657612e+09  6.20656427e+05
  1.98716913e+06  6.71710025e+04  8.56715778e+05  8.50520939e+05
 -8.95645973e+03 -5.70673714e+03  6.54312615e+08 -1.47657604e+09
 -6.20655263e+05 -1.98716915e+06 -6.71707257e+04 -1.74822435e-02
  3.85500129e-05  1.28263634e-04 -1.19137880e+01  6.08615834e+02
 -4.74935838e-01  3.93890628e-02 -1.07452305e-01  6.06059048e-07
  7.16390114e-06  5.07722561e-01 -2.64856766e+01 -3.05577008e-02
 -1.24139936e-03  9.38269917e-03 -3.18828125e-06 -4.65986405e-01
 -2.33583410e+00  3.18092009e-02  8.62254614e-04 -3.69558670e-04
 -1.05075181e+04  5.43405512e+05 -1.24457241e+02 -7.46708857e+00
  3.12751226e+01 -9.48515084e+12 -2.01844677e+04  3.26277186e+02
  8.15026630e+03  2.41533630e+00 -1.26862479e-01 -8.92709821e+00
```

```
  6.74125985e-01 -1.64083022e-02  2.89431838e+00  1.36303443e+03
 -4.61156048e+03  1.17331181e+03 -1.24829047e+03  9.25602867e+00
 -9.73605906e+00 -4.73908442e+06 -8.11671025e+06  7.52033505e+02
  9.36123116e+03 -6.88734063e+01 -5.29626332e+03  9.33700128e+03
  1.04389517e+03  1.61255768e+01  5.41467281e+01 -2.78875977e+06
 -4.91514217e+07  7.41318702e+03  2.01889029e+04 -3.16916312e+01
 -2.39392400e+03  3.66731448e+03 -2.64444893e+01 -1.54235854e+01
  2.26936249e+06  1.79917736e+07 -2.10976034e+02 -1.27816837e+04
  1.65904449e+02 -4.15146074e+02 -5.72293193e+01  7.07109040e+00
  3.35825269e+06  8.78790314e+06 -4.05222242e+03 -1.28901280e+03
  4.33104178e+01 -1.41322981e+00 -8.14362076e-01  6.00200337e+04
 -1.43351491e+05  9.17222721e+00 -8.63962991e+00  1.15847296e+01
 -3.26419546e-01 -1.87165190e+04 -1.84535935e+06 -1.27841552e+02
 -4.64393690e+02  1.70881581e+00 -7.29977735e+08  2.55147258e+10
  5.77136136e+05 -2.04596333e+06 -9.68478264e+04 -7.29864843e+09
 -1.06114054e+07 -8.17933303e+07 -4.39642241e+05  5.76820104e+03
  6.26672040e+03  3.17332524e+02 -1.52710975e+03 -1.94973134e+01
 -4.72267866e+02  6.78331822e+02  1.47728289e+02  2.29218554e+03
  6.86954849e+00  6.38827876e+01  1.95032468e+06 -4.10347121e+07
  6.66115380e+03  1.08276717e+04  3.71798265e+01 -6.83637819e+02
  1.26838370e+02 -2.40580094e+01 -8.90424315e+01 -4.42004661e+06
  5.09097760e+07 -6.12009724e+03 -1.42481242e+04  5.98511750e+01
 -4.15146080e+02 -5.72293208e+01  7.07109247e+00  3.35825274e+06
  8.78790285e+06 -4.05222168e+03 -1.28901284e+03  4.33100610e+01
 -1.41322981e+00 -8.14362062e-01  6.00200346e+04 -1.43351603e+05
  9.17217085e+00 -8.63963060e+00  1.15847309e+01 -3.26419558e-01
 -1.87165197e+04 -1.84535928e+06 -1.27841462e+02 -4.64393690e+02
  1.70878846e+00 -7.29977740e+08  2.55147266e+10  5.77136860e+05
 -2.04596327e+06 -9.68483382e+04  1.65117705e+11 -1.06114367e+07
 -8.17933321e+07 -4.39639146e+05  5.76816924e+03  6.26671868e+03
  3.17333321e+02 -1.52711002e+03 -1.94971594e+01 -4.72269721e+02
 -1.42422229e+02 -2.41902395e+03  1.71884610e+01  2.51596437e+01
  2.46972192e+06 -9.87506404e+06 -5.41056722e+02  3.42045258e+03
 -9.70315422e+01  4.15145995e+02  5.72293193e+01 -7.07109041e+00
 -3.35825269e+06 -8.78790343e+06  4.05222054e+03  1.28901283e+03
 -4.33106279e+01  1.41322981e+00  8.14362069e-01 -6.00200344e+04
  1.43351546e+05 -9.17219675e+00  8.63962842e+00 -1.15847423e+01
  3.26419555e-01  1.87165197e+04  1.84535932e+06  1.27841517e+02
  4.64393692e+02 -1.70879032e+00  7.29977732e+08 -2.55147265e+10
 -5.77136283e+05  2.04596330e+06  9.68484793e+04  1.12678668e+10
  1.06114247e+07  8.17933310e+07  4.39635767e+05 -5.76815359e+03
 -6.26672071e+03 -3.17332058e+02  1.52710992e+03  1.94963974e+01
  4.72265257e+02  2.77632125e-05  6.57421829e-08  3.07154268e-08
 -1.52771856e-03 -2.63080532e-01  1.84075856e-03  2.61094468e-06
  5.17490107e-04  1.69523846e-09 -5.30095626e-09 -7.16875825e-04
  1.89560822e-03 -1.33080170e-05 -1.35550924e-06 -1.22842829e-06
  9.66374223e-11  2.92806140e-04  3.44307760e-02  1.53725928e-05
 -1.87391860e-06  3.40985844e-06  1.90889022e+00 -2.40544080e+03
 -1.94308113e-02 -1.58833006e-02 -5.23100340e-01 -3.13956550e+08
 -1.13172439e+01 -2.62160107e-01  4.96936797e+00 -1.08199091e-02
  7.76356721e-04  2.10384699e-02 -7.92530364e-04  7.43889822e-05
  1.14846978e-02  1.45355203e-12 -1.10980604e-11  1.05879076e-06
  1.01525239e-04  4.26488943e-08  1.21825968e-08 -6.05342364e-08
 -2.86193847e-11 -7.77603892e-07 -1.49150666e-03 -3.78402952e-07
 -6.80944203e-09  2.52604810e-09  5.33140126e-02  4.55023297e+01
  1.55850004e-03  1.96318952e-03 -8.54273649e-03 -1.46544039e+09
  5.55493528e-01 -1.31533446e-02 -8.43734728e-02  8.36952514e-04
```

```
  6.72345579e-05  1.28958532e-05 -6.76710253e-05  4.79501810e-06
 -3.05889272e-04  1.05038496e-11  8.86540244e-07  9.95165184e-04
  5.15778393e-08  3.57304928e-09  3.33383045e-08 -3.05927910e-01
 -3.46487670e+01 -1.76328114e-02 -1.63784213e-03  6.93350319e-03
 -2.99706879e+09  9.63314783e-01  3.95267625e-02 -3.68629080e-01
 -7.29084277e-04 -2.76237949e-05 -2.01006607e-04 -1.99820551e-05
  3.84598376e-05 -2.21801974e-04  2.05250104e+04  3.61976848e+06
  2.88945849e+02  7.56603262e+00  6.64772866e+01  8.97341588e+12
  6.95943366e+03 -3.80094811e+02  3.97027687e+03  4.54879661e+00
  7.29228739e-01  4.47054463e+00 -2.41812675e-01 -4.28592379e-02
 -3.77480328e+00 -1.31681553e+14 -1.95287856e+09  6.36719227e+10
  4.21306170e+07  1.72642776e+02 -1.29832203e+01 -2.83872099e+02
 -1.45822274e+01  3.23195882e+00  1.53290055e+01  8.55127052e-01
 -1.28260448e-02  6.98940123e-02  5.29033369e-02 -2.90605553e-03
  1.58595691e-02 -2.63845683e-02  4.83639834e-03  3.78524159e-03
 -2.54048226e-02]
Mean squared error: 187039770190.87
```

2)f) Is your test MSE from 2)e) any better than the one obtained in 2)d)?

Not at all, this MSE is significantly higher.

---

3. Ridge and Lasso Regression

Now that you have warmed up let the fun begin! We will start by looking at Ridge and Lasso Regression. In scikit-learn there are a few ways to compute Lasso Regression and Ridge Regression. For this exercise focus on the following:

- [Lasso (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html)
- [LassoCV (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV)
- [Ridge (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)
- [RidgeCV (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html)

The `CV` at the end of Ridge or Lasso means that this model has cross-validation incorporated into its model objects. Therefore the alphas will be internally computed through cross-validation in these classes.

Note: For section you may take the boston dataset or the diabetes dataset.

In [12]:
```python
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
```

3)a) For this first exercise with Ridge regression we are going to apply Ridge() model to the Diabetes dataset. Please complete the code below to iterate through different values of `alpha` and store the values of the errors and coefficients for each alpha.

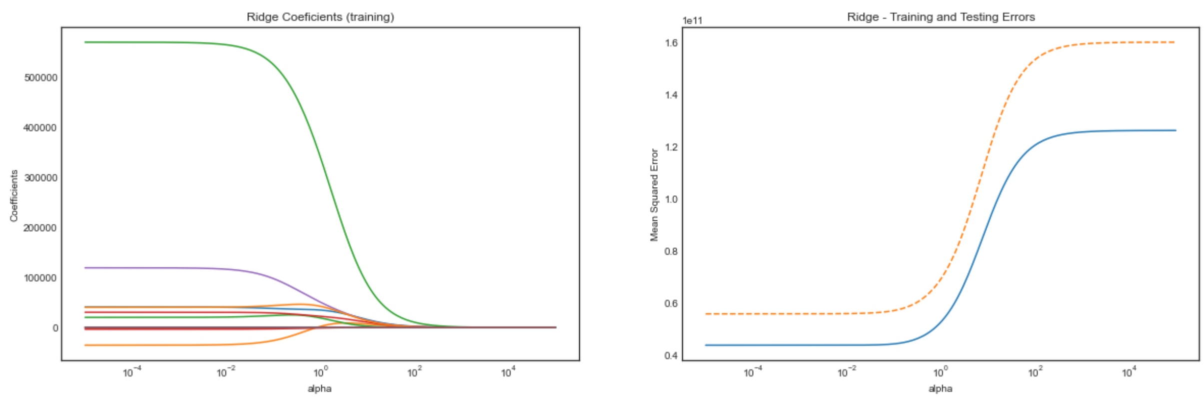
Steps:

- Declare a model with Ridge(). This is the same way we would do it with LinearRegression() model
- Fit the model
- Use the model to make predictions
- Store values of model.coef_ for coefficients
- Store the MSE in errors by using the metric `mean_square_error(,)`

In [13]:
```
model_ridge = Ridge(normalize=True)
coeficents = []
errors_train = []
errors_test = []

alphas = np.logspace(-5, 5, 200)
for a in alphas:
    model_ridge.set_params(alpha=a)
    model_ridge.fit(housing_X_train, housing_y_train)
    coeficents.append(model_ridge.coef_)
    y_pred = model_ridge.predict(housing_X_train)
    errors_train.append(mean_squared_error(housing_y_train,y_pred))
    y_pred_test = model_ridge.predict(housing_X_test)
    errors_test.append(mean_squared_error(housing_y_test,y_pred_test))
```

3)b) Plot the coefficients and errors you collected in the previous question. You should generate two plots, one of coefficients vs alphas, and another one of error vs alphas. You can take inspiration from this [example (https://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_coeffs.html#sphx-glr-auto-examples-linear-model-plot-ridge-coeffs-py)](https://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_coeffs.html#sphx-glr-auto-examples-linear-model-plot-ridge-coeffs-py).

In [14]:
```
plt.figure(figsize=(20, 6))

plt.subplot(121)
ax = plt.gca()
ax.plot(alphas, coeficents)
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('Coefficients')
plt.title('Ridge Coeficients (training)')
plt.axis('tight')

plt.subplot(122)
ax = plt.gca()
ax.plot(alphas, errors_train,linestyle="-", label="Train")
ax.plot(alphas, errors_test,linestyle="--", label="Test")
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('Mean Squared Error')
plt.title('Ridge - Training and Testing Errors')
plt.axis('tight')

plt.show()
```

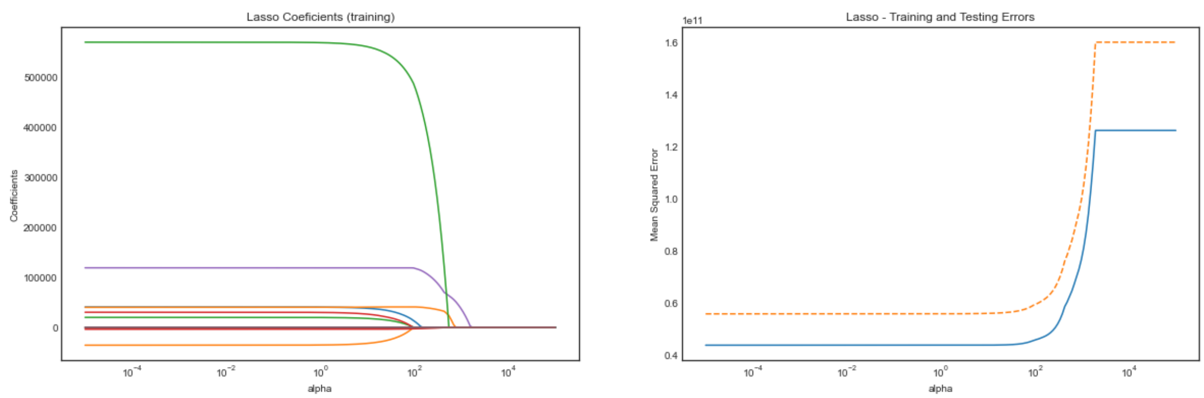


3)c) The graphs resemble the ones we talked about in class. What can you comment about these graphs? What seems to be a reasonable value for alpha?

According to the right hand side, the alpha with the lowest error is around 10^{-1} (0.1). Taking an alpha=0.1 to the left hand side we can draw a vertical line and see the coefficients for the model regresenting that alpha.

3)d) Repeat a),b) and c) for Lasso(). Do you see any differences (compared to Ridge) when looking at the graphs?

In [15]:
```
model_lasso = Lasso(normalize=True)
coeficents = []
errors_train = []
errors_test = []

alphas = np.logspace(-5, 5, 200)
for a in alphas:
    model_lasso.set_params(alpha=a)
    model_lasso.fit(housing_X_train, housing_y_train)
    coeficents.append(model_lasso.coef_)
    errors_train.append(mean_squared_error(housing_y_train,model_lasso.p
redict(housing_X_train)))
    errors_test.append(mean_squared_error(housing_y_test,model_lasso.pre
dict(housing_X_test)))

plt.figure(figsize=(20, 6))

plt.subplot(121)
ax = plt.gca()
ax.plot(alphas, coeficents)
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('Coefficients')
plt.title('Lasso Coeficients (training)')
plt.axis('tight')

plt.subplot(122)
ax = plt.gca()
ax.plot(alphas, errors_train,linestyle="-", label="Train")
ax.plot(alphas, errors_test,linestyle="--", label="Test")
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('Mean Squared Error')
plt.title('Lasso - Training and Testing Errors')
plt.axis('tight')

plt.show()
```



According to the right hand side, the alpha with the lowest error is around 10^1. Taking an alpha=10 to the left hand side we can draw a vertical line and see the coefficients for the model regresenting that alpha. Different to Ridge, we needed a smaller alpha for Lasso and we had to update the rande to include smaller alphas.

3)e) For this question use the CV version of Ridge (with a cv=10) to model your same dataset. How good is this model for your dataset?

In [16]:
```
model_ridgeCV = RidgeCV(cv=10,normalize=True).fit(housing_X_train, housi
ng_y_train)


coefficients=pd.DataFrame(housing_X_train.columns,columns=["feature"])
coefficients["values"] = model_ridgeCV.coef_

print("Ridge - best_score: \n",model_ridgeCV.best_score_)
print("Ridge - best alpha: "+str(model_ridgeCV.alpha_))
print("Ridge - intercept: "+str(model_ridgeCV.intercept_))
print("Ridge - coefficients:\n ", coefficients)
```

```
Ridge - best_score:
 0.6481587100698838
Ridge - best alpha: 0.1
Ridge - intercept: -6924275.449746476
Ridge - coefficients:
             feature         values
0        bathrooms   37984.164749
1         bedrooms  -26149.963715
2        condition   24112.951250
3           floors   28999.330013
4            grade   97902.440693
5       sqft_above      71.317974
6    sqft_basement      77.971226
7      sqft_living      76.765596
8    sqft_living15      41.601694
9         sqft_lot      -0.001235
10      sqft_lot15      -0.431604
11            view   43743.366621
12      waterfront  525483.711373
13        yr_built   -2744.120654
14    yr_renovated      24.669167
15         zipcode     118.274171
```

These results make sense, given our previous results of Ridge. Ridge is not doing a good job (R^2 = 0.65) predicting the target variable. We might need to do some feature engineering or try more complex models that better capture the features of the diabetes dataset.

3)f) Repeat e) for LassoCV(). How good is this model for your dataset? Was it better or worse than Ridge?

In [17]:
```
model_lassoCV= LassoCV(cv=10, normalize=True).fit(housing_X_train, housi
ng_y_train)


coefficients=pd.DataFrame(housing_X_train.columns,columns=["feature"])
coefficients["values"] = model_lassoCV.coef_

print("Lasso - R-squared: "+str(model_lassoCV.score(housing_X_train, hou
sing_y_train)))
print("Lasso - best alpha: "+str(model_lassoCV.alpha_))
print("Lasso - intercept: "+str(model_lassoCV.intercept_))
print("Lasso - coefficients:\n ", coefficients)
```

```
Lasso - R-squared: 0.6538217014066989
Lasso - best alpha: 2.578760187662181
Lasso - intercept: 3590231.633663503
Lasso - coefficients:
             feature         values
0       bathrooms   40211.378761
1        bedrooms  -34439.666873
2       condition   19658.186174
3          floors   29540.908072
4           grade  119222.916289
5      sqft_above      13.440126
6   sqft_basement      18.601824
7     sqft_living     142.023282
8   sqft_living15      23.311050
9        sqft_lot      -0.000000
10     sqft_lot15      -0.499649
11           view   40317.297607
12     waterfront  567455.493269
13       yr_built   -3445.644723
14   yr_renovated      13.523823
15        zipcode      24.126693
```

Again, these results make sense, given our previous results of Lasso. Lasso is not doing a good job (R^2 = 0.65) with this diabetes dataset. We might need to do some feature engineering or try more complex models that better capture the features of the diabetes dataset.

If you got to this point you will receive full marks for your Lab 2 (considering that you also attending this Lab session). Nonetheless, we recommend you keep going so that HW2 is easier for you.

---

4. Cross Validation

For this question we will look at K-fold cross validation and LOOCV.

In [18]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.preprocessing import PolynomialFeatures
import time
```

4)a) For this question you are going to apply cross validation to your dataset, while iterating from polynomial degree=1 up to degree=2. Look at the requirements below. How much did this operation take to compute?

Use [KFold() (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html) with the following parameters:

- k of 10.
- randome_state=None
- shuffle = False

Uncomment the lines of code and fill out the missing code.

In [20]:
```
lm = LinearRegression(normalize=True)
#characteristics of our CV (as listed above)
cross_val = KFold(n_splits=10, random_state=None, shuffle=False)

#start timer
start = time.time()
for i in range(0,3):
    poly = PolynomialFeatures(degree=i)
    if i==1:
        X_current = poly.fit_transform(housing_X_train)
    else:
        X_current = housing_X_train.copy()
    model = lm.fit(X_current, housing_y_train)
    scores = cross_val_score(model, X_current, housing_y_train,
                             scoring="neg_mean_squared_error", cv=cross_
val, n_jobs=-10)
    print("Degree-"+str(i)+" polynomial MSE: " + str(np.mean(np.abs(scor
es))) + ", standard_dev: " + str(np.std(scores)))
computation_time = (time.time()-start)
print("Computation time: %5.3f"%computation_time)
```

```
Degree-0 polynomial MSE: 43943513218.61914, standard_dev: 4819468328.93
5384
Degree-1 polynomial MSE: 43943513218.61915, standard_dev: 4819468328.93
5347
Degree-2 polynomial MSE: 43943513218.61914, standard_dev: 4819468328.93
5384
Computation time: 0.199
```

This operation took 0.199 seconds for me.

This can be different for you, depending on the characteristics of your computer.

4)b) Repeat the steps and code from 4)a) but this time use LOOCV instead of Kfolds. How much longer did your LOOCV take compared to your k-fold cross validation?

In [21]:
```
loo_cv = LeaveOneOut()
loo_cv.get_n_splits(housing_X_train)

#we are doing the same as before but now our splits/k = n
start = time.time()
loocv = KFold(n_splits=loo_cv.get_n_splits(housing_X_train), random_stat
e=None, shuffle=False)

for i in range(0,3):
    poly = PolynomialFeatures(degree=i)
    if i==1:
        X_current = poly.fit_transform(housing_X_train)
    else:
        X_current = housing_X_train.copy()
    model = lm.fit(X_current, housing_y_train)
    scores = cross_val_score(model, X_current, housing_y_train,
                             scoring="neg_mean_squared_error", cv=loocv,
n_jobs=-10)
    print("Degree-"+str(i)+" polynomial MSE: " + str(np.mean(np.abs(scor
es))) + ", standard_dev: " + str(np.std(scores)))

    computation_time = (time.time()-start)
print("Computation time: %5.3f"%computation_time)
```

```
Degree-0 polynomial MSE: 43963987132.36465, standard_dev: 215046165333.
95865
Degree-1 polynomial MSE: 43955281968.844215, standard_dev: 21475498166
4.15472
Degree-2 polynomial MSE: 43963987132.36465, standard_dev: 215046165333.
95865
Computation time: 290.331
```

My computation took 290.331 seconds. This is nearly 290 times slower than the k-fold calculation.

END OF LAB 2!