**QF634 APPLIED QUANTITATIVE RESEARCH METHODS**
**LECTURE 2**

Lecturer: Prof Lim Kian Guan

# Regularizations

Ridge and Lasso Regressions

## Regression and Regularization

Let $Y_k$ be the $k^{th}$ target variable observed and ($X_{1k}$ , $X_{2k}$ , $X_{3k}$ , ….. , $X_{pk}$) be the set of p features related to the $k^{th}$ target variable. The linear regression is written as

$$Y_k = b_0 + b_1 X_{1k} + b_2 X_{2k} + b_3 X_{3k} + \ldots\ldots + b_p X_{pk} + \varepsilon_k \qquad (2.1)$$

where $\varepsilon_k$ is a residual variable that is unobserved.

## Regression and Regularization

- Unlike classical setup in linear regression where the added residual noise or innovation may take specific distributional assumptions to enable testing, the residual noise in the machine learning method takes a low profile.
- The linear regression under supervised machine learning in Eq. (2.1) typically aims to find optimal parameter (estimates) $\hat{b}_0$, $\hat{b}_1$, $\hat{b}_2$, …., $\hat{b}_p$ so that a loss criterion such as sum of squared errors, $\sum_{k=1}^{0.8n}\left(Y_k - \hat{Y}_k\right)^2$ is minimized, where fitted $\hat{Y}_k = \hat{b}_0 + \hat{b}_1 X_{1k} + \hat{b}_2 X_{2k} + \cdots + \hat{b}_p X_{pk}$. 0.8n is 80% sample as training set.
- $R^2$-score for the training data set is $1 - \sum_{k=1}^{0.8n}\left(Y_k - \hat{Y}_k\right)^2 / \sum_{k=1}^{0.8n}(Y_k - \overline{Y}_k)^2$. The second term is sum of squared errors over total sum of squares in standard least squares regression terminology, and $R^2$ lies between 0 and +1.

## Regression and Regularization

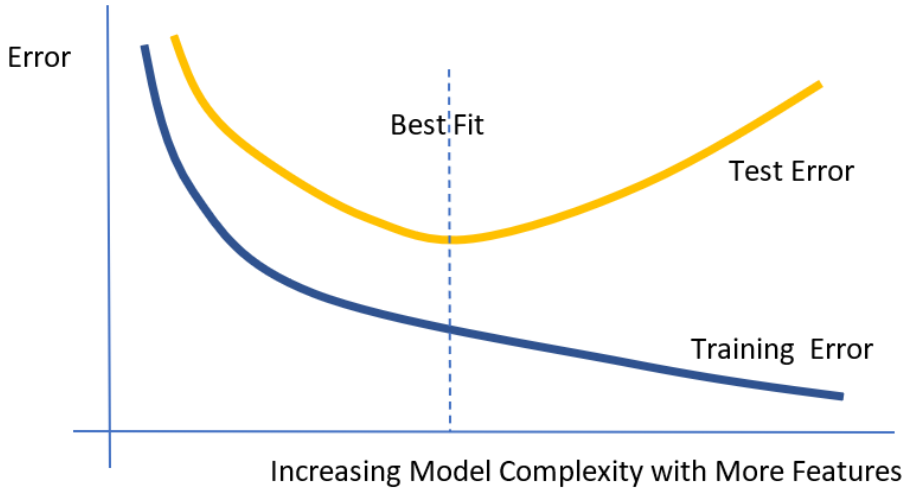
- When the trained or fitted coefficients $\hat{b}_0, \hat{b}_1, \hat{b}_2, \ldots., \hat{b}_p$ are employed in the prediction of the test data set of 0.2n number of the remaining $Y_k$'s, using the associated test set data of $X_{jk}$'s (for j =1,2,…,p), the prediction yields $Y_k^* = \hat{b}_0 + \hat{b}_1 X_{1k} + \hat{b}_2 X_{2k} + \cdots + \hat{b}_p X_{pk}$.
- The test score is $R^2 = 1 - \sum_{k=1}^{0.2n}(Y_k - Y_k^*)^2 / \sum_{k=1}^{0.2n}(Y_k - \overline{Y}_k)^2$. In this case, the score $R^2$ may be sometimes negative if the actual model of the test data is different so that $\sum_{k=1}^{0.2n}(Y_k - Y_k^*)^2$ is very large.
- The $R^2$ score in the test set is a useful measure of accuracy of the machine learning method when coefficients (or model parameters) trained by a training data set is applied to new data in a test data set, assuming both data sets come from the same data generating model. The use of training and test data sets is similar to the idea of in-sample and out-of-sample fits.

## Regression and Regularization

- If $\{Y_k, X_{1k}, X_{2k}, X_{3k}, \ldots\ldots, X_{pk}\}_{k=1,2,\ldots n}$ is not a time series, but a cross-section, then splitting the set into training and testing subsets can be random.
- If it is a time series, then typically the training set data would precede those of the test set. Moreover, in the strict sense of prediction based on available information, the features would have time stamps prior to that of the target variable.
- If a training set $R^2$ or accuracy of fit by the model is high and if a test set $R^2$ score is also high or having accurate prediction based on the trained or fitted coefficients, then the model with its trained coefficients may be usable for future prediction of the target variable when a new set of features arrives.

# Overfitting and Underfitting in Machine Learning

- Regressions based on available features can run into overfitting problem when too many features (or explanatory variables) are utilized. Good fit, e.g., high $R^2$ in the training data set, but inaccurate prediction using the test data set.
- Overfitting could occur with some outlier features that attract coefficients with high fitted values, and could also occur with highly multi-collinear features.
- Underfitting occurs when both the $R^2$'s in the training data set fitting and in the test data set prediction are low.
- To reduce overfitting and enable good prediction, i.e., high $R^2$'s in the training data set fitting and in the test data set prediction, some constraints are added to the fitted coefficients.



## Overfitting and Underfitting in Machine Learning

- Prediction error or deviation of actual test data value from predicted value could typically be produced by too large values of some fitted coefficients that were too eager to overfit the training data set or due to multi-collinearity.
- A penalty function is added to reduce the tendencies to yield large-fitted coefficients – this technique is called regularization.
- There are two common regularized linear regression methods – Ridge regression and Lasso (Least absolute shrinkage and selection operator) regression.
- Regularization could shrink some fitted coefficient or even reduce them toward zero, effectively reducing the dimension space of the features since those affected features become impactless if their fitted coefficients are close to zero.

## Ridge Regression

- For Ridge regression using the module sklearn.linear_model.Ridge, see documentation in https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

$$\text{Min } \sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{p} b_j X_{jk}\right)^2 + \alpha\left(\sum_{j=0}^{p} b_j^2\right) \qquad (2.2)$$

  where $X_{0k} = 1$. The second term is in $L^2$-norm, so Eq. (2.2) is also called a $L^2$ regularized regression. The L2 norm calculates the distance of the vector coordinate from the origin of the vector space. It is also known as the Euclidean norm as it is calculated as the Euclidean distance from the origin.

- The second term is the penalty term for the regression. When we use the computed value of linear_model. Ridge ( ), the default parameters within ( ) includes setting hyperparameter alpha, e.g., $\alpha$ =0.1.

## Lasso Regression

- For Ridge regression using the module sklearn.linear_model.Lasso, see documentation in https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html.

$$\text{Min } \sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{p} b_j X_{jk}\right)^2 + \alpha \sum_{j=0}^{p}|b_j| \qquad (2.3)$$

  where $X_{0k} = 1$. The second term is in $L^1$-norm, so Eq. (2.3) is also called a $L^1$ regularized regression. The L1 norm calculates the sum of absolute vector values, where the absolute value of a scalar is $|.|$.

- The second term is the penalty term for the regression. When we use the computed value of linear_model.Lasso ( ), the default parameters within ( ) includes setting hyperparameter alpha, e.g., $\alpha$ =0.1.

## Worked Example – Data

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

- We explain the method behind one of the popular problems posted on public websites – that of predicting California house prices. This method and other related methods in machine learning can be applied to predicting housing prices anywhere as long as there are adequate data pertaining to the house prices and features of each house.
- This publicly accessible dataset comprises only 8 features (characteristics) of houses in the various areas of the state of California in a 1990 US Census survey. 20,640 samples (sample points) are available in the data set California_housing.frame.csv, each of which is a residential block in California.

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

The target variable is MedHouseVal -- median house value in a block, in units of US$100,000. The features are:

(1) MedInc -- median of household incomes in block, in units of $10,000,

(2) HouseAge – median age of house in block in years,

(3) AveRooms -- average number of rooms per household in block,

(4) AveBedrms -- average number of bedrooms per household in block,

(5) Population -- total number of occupants in block,

(6) AveOccup -- average number of occupants per household in block,

(7) Latitude, and

(8) Longitude.

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

As the 'MedHouseVal' price data might have been right-censored at 5 (representing $500,000), any number above 5 is entered as 5. The censored data may not carry the correct information for purpose of analyses since a $10 million median house value is recorded as $500,000 and may distort the fitting and testing results. We remove such sample points as they do not carry the correct price information. We also remove incorrect data of 'AveOccup' > 10. This leaves 19,615 sample points.

In [2]:
```
### We can import the data set in another way via a downloaded csv file in the same Dir as this Jupyter kernel (program file)
### The dataframe name is now df
df = pd.read_csv('california_housing.frame.csv')

### There is an aberration in the large spikes of up to 1200 occupants per household in the original data set.
### These could be incorrectly notated. We delete 37 such records with AveOccup > 10
df = df[~(df['AveOccup'] >= 10.0)]

df=df.iloc[:,1:10]
```

In [3]:
```
df.head()
### This returns the first 5 rows of the dataframe df
```

Out[3]:

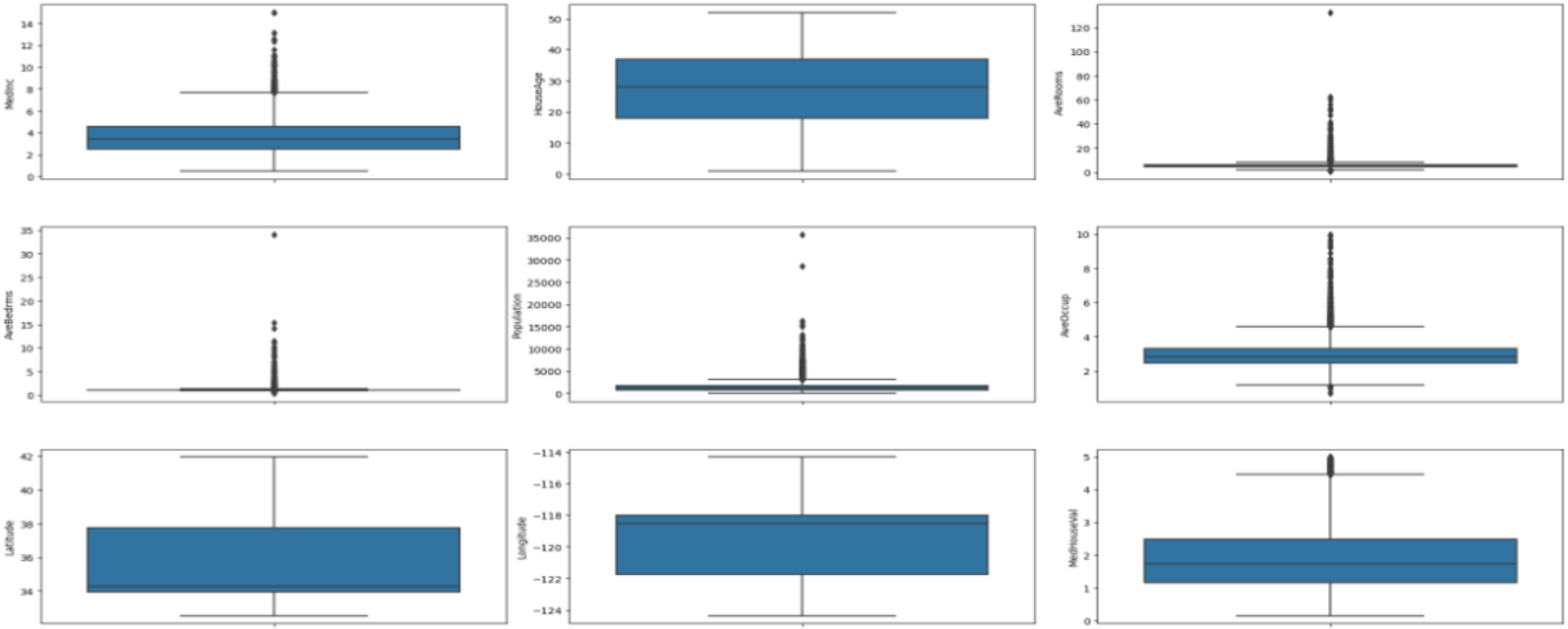
| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Longitude | MedHouseVal |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 8.3252 | 41.0 | 6.984127 | 1.023810 | 322.0 | 2.555556 | 37.88 | -122.23 | 4.526 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 | 2.109842 | 37.86 | -122.22 | 3.585 |
| 2 | 7.2574 | 52.0 | 8.288136 | 1.073446 | 496.0 | 2.802260 | 37.85 | -122.24 | 3.521 |
| 3 | 5.6431 | 52.0 | 5.817352 | 1.073059 | 558.0 | 2.547945 | 37.85 | -122.25 | 3.413 |
| 4 | 3.8462 | 52.0 | 6.281853 | 1.081081 | 565.0 | 2.181467 | 37.85 | -122.25 | 3.422 |

In [4]:
```
np.shape(df)
### This returns the dimension of the tabular dataframe, i.e. 506 rows of data without variable names, 15 columns excepting index
```

Out[4]: (20603, 9)

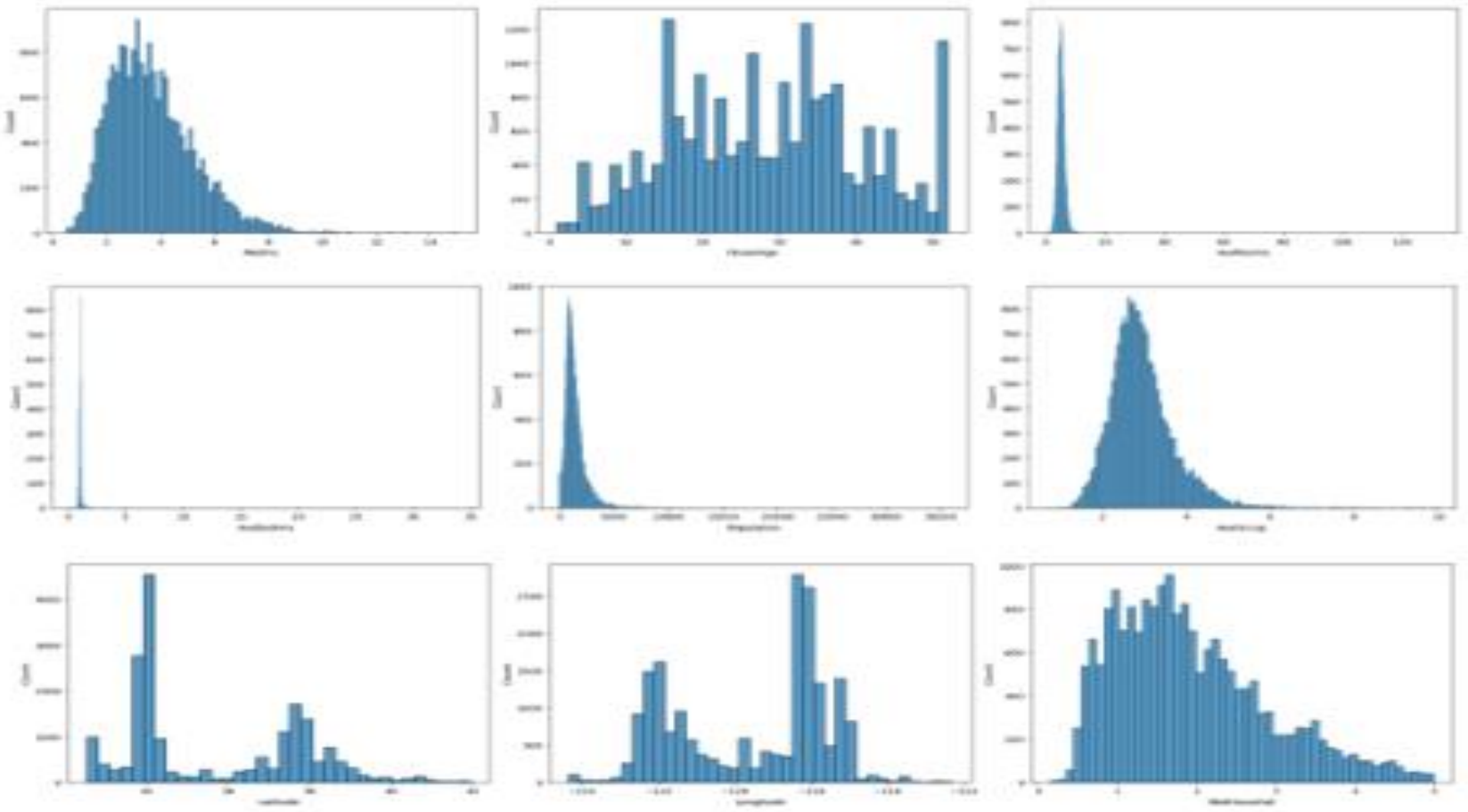
## Box Plots

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook



## Histograms

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook



## Pairwise Correlations of all Features

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

Code line [13] shows the correlation heat map amongst the features, including the target variable. It is seen that MedInc has reasonably high correlation with MedHouseVal, followed by AveOccup, Latitude and AveRooms. The other features have low pairwise correlations with MedHouseVal, below 0.1.

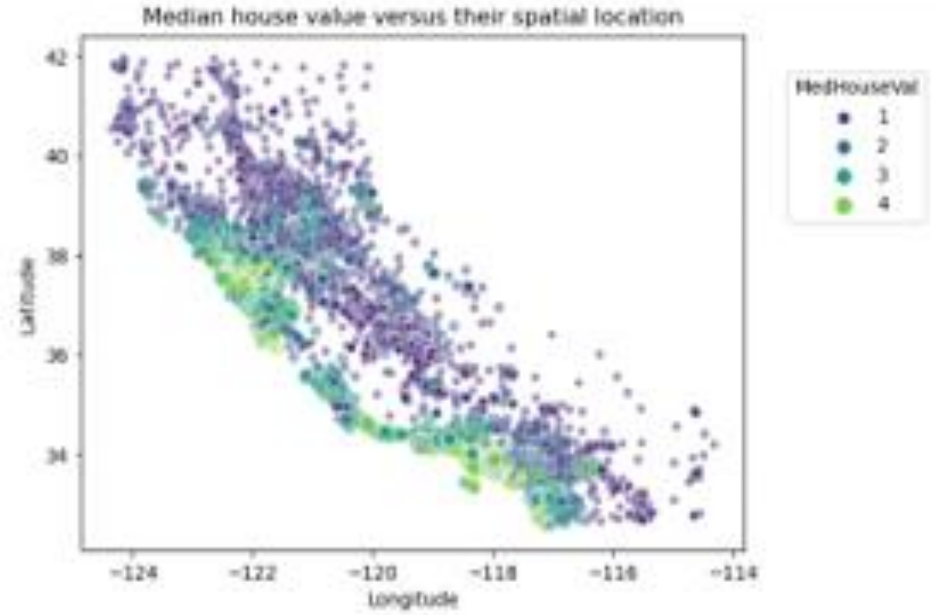
```
In [13]: ### pairwise correlation on the features including medv
         plt.figure(figsize=(20, 10))
         sns.heatmap(df.corr().abs(),  annot=True)
         ### In heat map, clearly tax and rad are highly correlated
         ### For more options - see https://seaborn.pydata.org/generated/seaborn.heatmap.html
```

```
Out[13]: <AxesSubplot:>
```

## Plot of Dep Var Latitude and Longitude

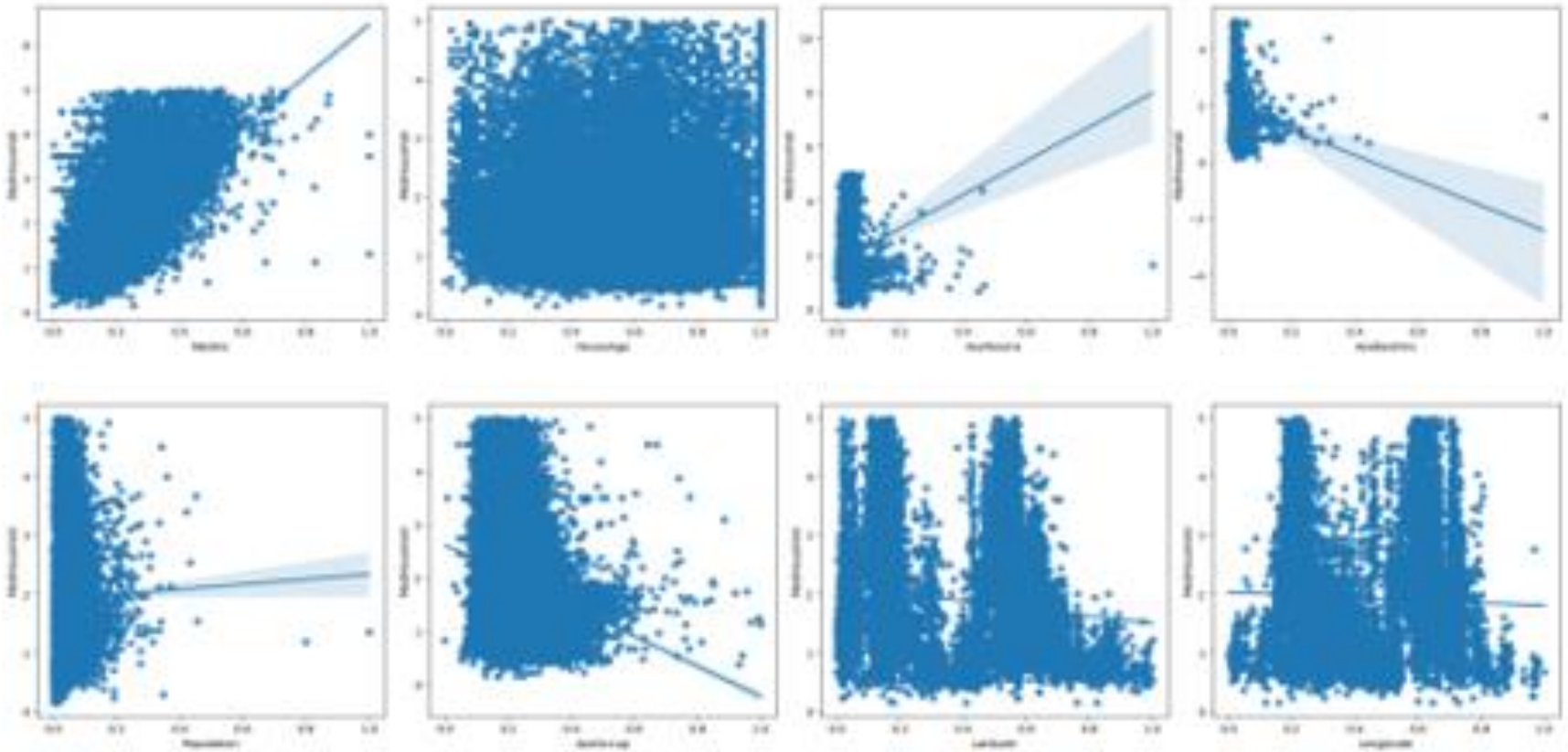
Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

```
In [14]: import seaborn as sns
sns.scatterplot(data=df, x="Longitude", y="Latitude",
                size="MedHouseVal", hue="MedHouseVal",  ###size is grouping variable, hue is order of processing
                palette="viridis", alpha=0.5)
plt.legend(title="MedHouseVal", bbox_to_anchor=(1.05, 0.95),
           loc="upper left")
_ = plt.title("Median house value versus their spatial location")
### Key shows MedHouseVal in ranges (1) 0-1, (2) 1-2, (3) 2-3, (4), > 3
```



**Plot of Dep Var against Features (Scaled to (0,1))**

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook



- A linear regression model will be used to predict the housing prices in subsection 2.2. For this, a standard procedure would be to split the 19,615 data points (each point referring to a set of $\{Y_k, X_{1k}, X_{2k}, X_{3k}, \ldots\ldots, X_{pk}\}$ values for a given k) into a training data set, a validation data set, and a test data set. This could be in proportions 60%, 20%, and 20% respectively.
- Regressions are done on the training data set using several models, each of which may involve a different set of features or explanatory variables. The best fitted model is usually chosen in terms of the highest $R^2$ or coefficient of determination. In linear regression, $R^2 = 1 -$ residual sum of squared errors/total sum of squared errors.
- The fitted models are then used to predict the y values in the validation data set. This allows optimizing or fine-tuning of hyperparameters in the model if any to find the model with highest $R^2$ or best fit. Overfitted models in the sense of poor $R^2$ in the validation data set are to be removed, and the best model in terms of highest $R^2$ would be used in testing.
- Before touching the test data set or the hold-out data set, the original training and validation data sets could be combined to do training once more for the chosen model. The latest fitted model is then used to check out prediction score or $R^2$ in the test data set.
- If the latter test score is reasonably high (close to the training $R^2$), then the model is ready for use to predict the next unseen set of new features or generalized data.

# Splitting Data Set given the Linear Model

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

We shall assume that the model with all the 8 features is the optimal one after validation, and that the model is going to be estimated or fitted next using the 60%+20% data sets (step before applying testing data). In Scikit-learn (or simply sklearn) module, there is a useful train_test_split function that splits the total data set into just a training set and a test set. We use this function to split into the 80% training and 20% test accordingly as explained above. See code line [18]. There is a random selection of the 80% versus 20%. However, if it is a time series, then random selection is not suitable.

In [18]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 50)
### test_size here is 20%; random state number is a seed number -- diff seed produces diff random draws
### note 0.2*19615 = 3923 (or closest to an integer). 0.8*19615 = 15,692.
### X_train, y_train each has 15,692 rows.
### X_test, y_test each has 3923 rows
### Latter can be checked using print(len(y_test))
print(len(X_train),len(y_train),len(X_test),len(y_test))
```

```
15692 15692 3923 3923
```

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

In code line [19], y_train is regressed linearly on X_train and the estimated intercept (train) and slopes (train) coefficients are seen in the outputs. Training set R2_score (train) is 0.62035, i.e. $1 - \sum_{k=1}^{0.8n}\left(Y_k - \hat{Y}_k\right)^2 / \sum_{k=1}^{0.8n}(Y_k - \overline{Y}_k)^2$. $\hat{Y}_k$ is fitted $Y_k$ using the estimated intercept (train) and slopes (train) coefficients multiplied by the training set $\{X_{1k}, X_{2k}, X_{3k}, \ldots\ldots, X_{pk}\}$ values.

```
In [19]: ### see module documentation in https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
         ### LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
         ### copy_X=True -- X will be copied; else, it may be overwritten.
         ### if fit_intercept = False, intercept will be set to 0.0
         ### n_jobs=None (same as  = 1), i.e. no use of additional parallel processors
         ### normalize=False: Ignored when fit_intercept is set to False.
         ### If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm.
         ### If you wish to standardize, one could use StandardScaler before calling fit on an estimator with normalize=False.

         from sklearn.linear_model import LinearRegression
         Linreg = LinearRegression()
         Linreg.fit(X_train, y_train) ###training the algorithm, note regression is done without normalizing the X, y
         ### To retrieve the intercept:
         print('Intercept (train):', Linreg.intercept_)
         ### To retrieving the slope:
         print('Slopes (train):', Linreg.coef_)

         from sklearn.metrics import r2_score
         y_pred_Linreg_train = Linreg.predict(X_train)
         ### Fitting y using the x_train data
         r2_score_Linreg_train = r2_score(y_train, y_pred_Linreg_train)

         print('R2_score (train): ', r2_score_Linreg_train)
         ### R2_score (train) is the R-square in the linear regression involving only the training data set

Intercept (train): 4.077720011109183
Slopes (train): [  5.91059822   0.46819653 -10.58017655  15.28770265   1.38115905
  -2.65806525  -3.69542188  -3.92228473]
R2_score (train):  0.6203540883823073
```

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

In code line [22], $Y_k^*$ is predicted value of $Y_k$ in the test data set using $Y_k^* = \hat{b}_0 + \hat{b}_1 X_{1k} + \hat{b}_2 X_{2k} + \cdots + \hat{b}_p X_{pk}$ where $\hat{b}_0$ is intercept (train) and $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_p$ are the slopes coefficients from the fitting in the training set. $X_{jk}$'s (for j =1,2,…,p) are from the test data set. The test score is $R^2 = 1 - \sum_{k=1}^{0.2n}(Y_k - Y_k^*)^2 / \sum_{k=1}^{0.2n}(Y_k - \overline{Y}_k)^2$ where $Y_k$ is y_test and $Y_k^*$ is y_pred. Test score $R^2$ is seen as 0.62137 which is slightly higher than the training $R^2$ shown in [19].

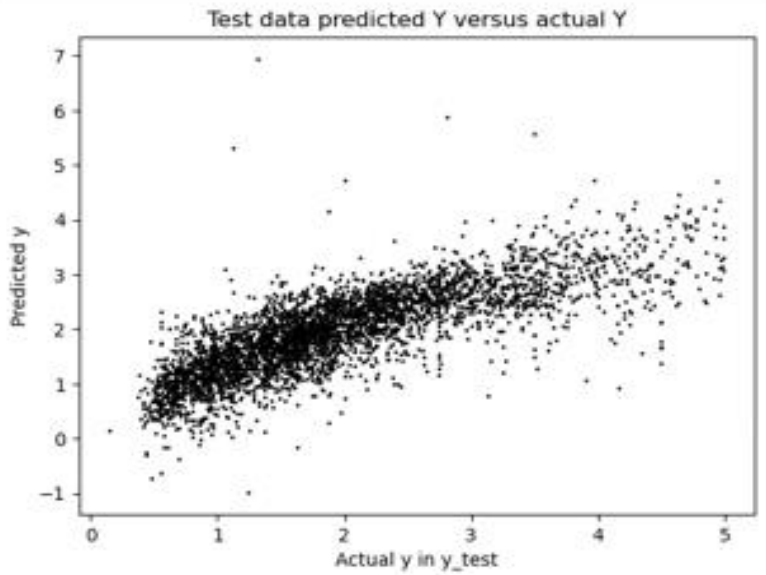
```
In [22]: ### Predicting R2 Score using Test Set but Intercept(train) and Slopes(train) from Training results
         y_pred = Linreg.intercept_ + np.dot(X_test,Linreg.coef_.T)
         rmse_y_pred = (np.sqrt(mean_squared_error(y_test, y_pred)))
         def tss(y_test):
             return ((y_test - np.mean(y_test))**2).sum()
         R2_pred=1-(3923*(rmse_y_pred)**2)/tss(y_test)
         print("Pred_RMSE:",rmse_y_pred)
         print("R2_pred:",R2_pred)

         Pred_RMSE: 0.6012841104200348
         R2_pred: 0.6213764009165903
```

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

The graph of the test data predicted y values versus the actual y (y_test values) is shown in the scatterplot of [24]. The prediction is seen to be reasonably good.

In [24]:
```
### A scatterplot may be clearer - here matplotlib
plt.scatter(y_test, y_pred,s=1,color='black')
plt.title("Test data predicted Y versus actual Y")
plt.xlabel("Actual y in y_test")
plt.ylabel("Predicted y")
plt.show()
```



Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

- Next, we apply Ridge regression and Lasso regression. These are shown in code lines [25] to [28]. In both regularized regressions, the regularization penalty term $\alpha > 0$ is a value to be exogenously fixed for training the model. Given $\alpha$, the regression model with penalty constraint is then optimized.
- In the regularized regression, $\alpha$ is called a hyperparameter of the regression model. Its value is external to the regression model that optimizes the parameters $\hat{b}_0, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_p$. Hyperparameters arise in many machine learning models such as the number of layers in neural network, the number of branches in a decision tree, the number of trees in a random forest, the number of clusters in a clustering algorithm, and so on.

- Given the hyperparameters, the model can then be optimized. Optimizing the model, such as maximizing $R^2$ or minimizing RMSE etc., can then be done in a second stage by fine-tuning (adjusting) the hyperparameter value to achieve a higher validation $R^2$ score – this could be done manually or in a programmed grid-step approach, e.g. sklearn.model_selection.GridSearchCV, using the validation data set.
- Once the parameters (and chosen or fine-tuned hyperparameter) are determined using the training and validation data sets, the optimization could be repeated given the selected hyperparameter over the combined training and validation data sets. This optimized model with optimal fitted parameters are then used to make predictions in the test data set. High test score $R^2$ means the model is ready for use in predicting targets based on new cases with given features. If test score $R^2$ is low, then the model should be improved or changed before using for generalized (new cases) prediction.

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

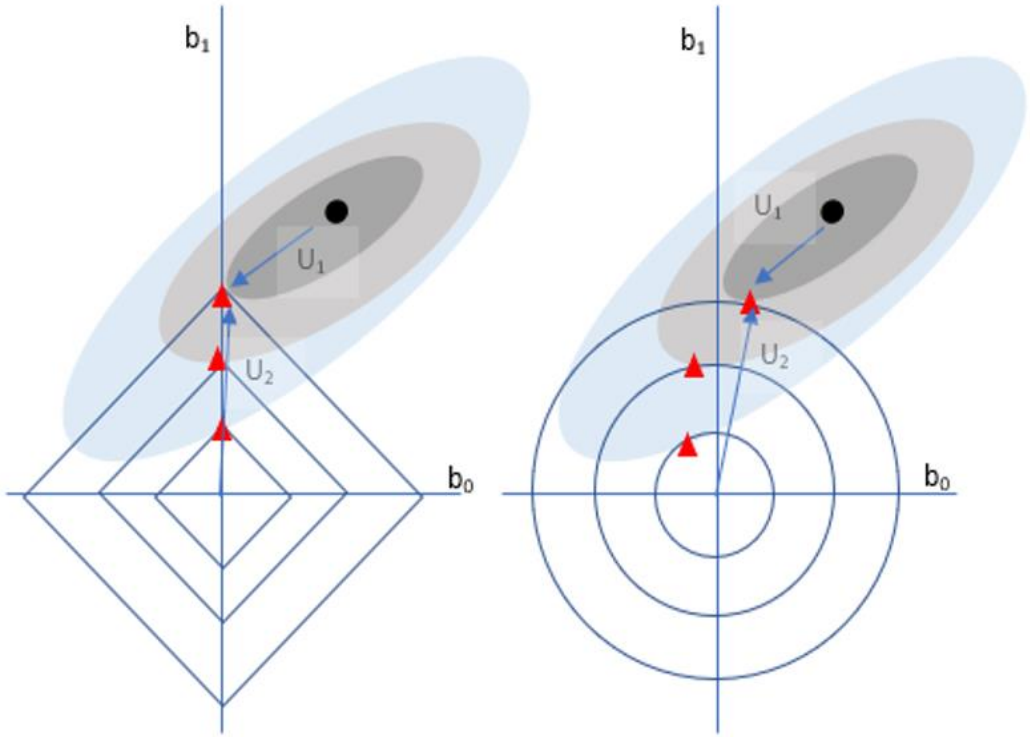
The training $R^2$ scores and test $R^2$ scores for the linear regression with no constraints, and the Ridge and Lasso regressions are tabulated below.

| | Training Score $R^2$ | Test Score $R^2$ |
|---|---|---|
| Linear Regression (no constraints) | 0.62035 | 0.62137 |
| Ridge Regression ($\alpha = 0.05$) | 0.62019 | 0.62123 |
| Lasso Regression ($\alpha = 0.005$) | 0.59152 | 0.59486 |

In this case study, the training and test scores for the regularized regressions are a little lower than that of regression without constraints – indicating underfitting when the $L^1$ , $L^2$ constraints are imposed on the parameters. Another possible regularized regression is the 'elastic net' regression that is the minimization of objective function as follows, where there are two hyperparameters $\alpha_1$ and $\alpha_2$.

$$\sum_{k=1}^{n}\left(Y_k - \sum_{j=0}^{p} b_j X_{jk}\right)^2 + \alpha_1\left(\sum_{j=0}^{p} b_j^2\right) + \alpha_2 \sum_{j=0}^{p}|b_j|.$$

In general, Lasso models are adept at feature selection – i.e., selecting a reduced/smaller set of features from a large set. It may therefore ignore or remove some features that may, nevertheless, be interesting or important. In contrast, the ridge regression penalty reduces correlated features together without quickly removing one of them. Thus, Ridge is not as useful when there is large number of possibly redundant features to prune instead of just shrinking. Their difference can be illustrated geometrically as follows using a two-dimensional case of minimizing $\sum_{k=1}^{n}(Y_k - b_0 - b_1 X_{1k})^2$



## Cross-Validation

- Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. K-fold cross validation refers to the number of groups that a given data sample less the hold-out sub-sample is split into. This procedure improves the simple split between training and test sets of the data.
- k approximately equal subsets of the data sample after leaving out the hold-out set are drawn randomly. Pick a subset as the validation sample. Pick the rest of subsets as training sample. Fit model on training sample and evaluate on the validation/test sample. Pick the next subset as the validation sample, and the rest as training sample, and so on. There will be k number of such fittings and testing using a validation subset. Thus, there will be k number of test $R^2$ scores using this k-fold cross-validation approach. The mean and variance of the scores are computed.
- Repeated k-fold cross-validation occurs when the resampling is done again by randomly splitting the sample into the hold-out and remainder and randomly selecting the k subsets again. Stratified cross-validation occurs when it is ensured that each subset of the data sample contains the same proportion of observations with a given categorical value, e.g., stratification by gender means each subset should have the same proportion of gender mix. Generally, increasing k and/or increasing the number of repetitions will reduce the bias (mean of the scores departure from true mean) but increase the variance or noise of the scores.

# Re-sampling

- There is some similarity of the concept of cross-validation in ML with resampling schemes in classical statistics
- The Jackknife is a method when sequentially one observation is left out and the required statistic is computed based on the remaining data. Sample of N requires N number of such computations. Mean and variance of the various computed statistics provide idea of closeness to true statistic and its variance. Similar idea to cross-validation。
- Bootstrapping is a statistical procedure that resamples a single dataset to create many simulated samples. This process allows you to calculate standard errors, construct confidence intervals, and perform hypothesis testing for numerous types of the sample statistics. Similar in concept to repeated cross-validation
- While Jackknife and Bootstrapping are resampling procedures to compute sample statistics, cross-validation and repeated cross-validation are procedures to compute validation scores。

## Cross-Validation

- Fine-tune hyperparameters
- Select optimal model

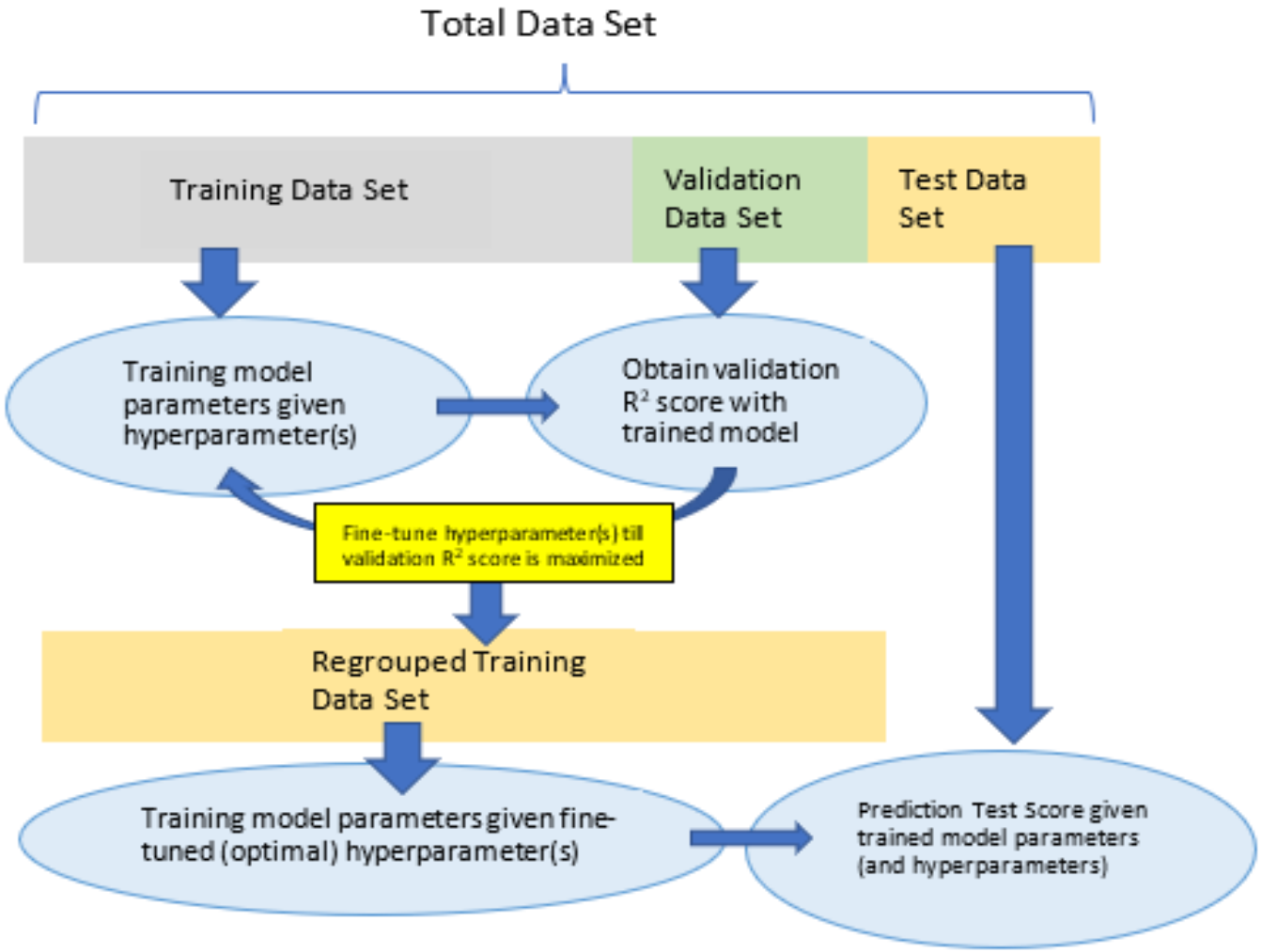



Figure 2.1 (original)

# k-Fold Cross-Validation

- The k-fold cross-validation is a common approach or strategy to improve validation effectiveness and thus come up with an optimal model (with optimized model parameters as well as hyperparameters) .
- This is done particularly with smaller sample size (smaller number of sample points) in the total data set.
- Usually not applicable when the data are in time series form as time ordering cannot be randomly re-arranged.
- Provides k **validation** $R^2$ scores. Computes mean and standard deviation of scores.

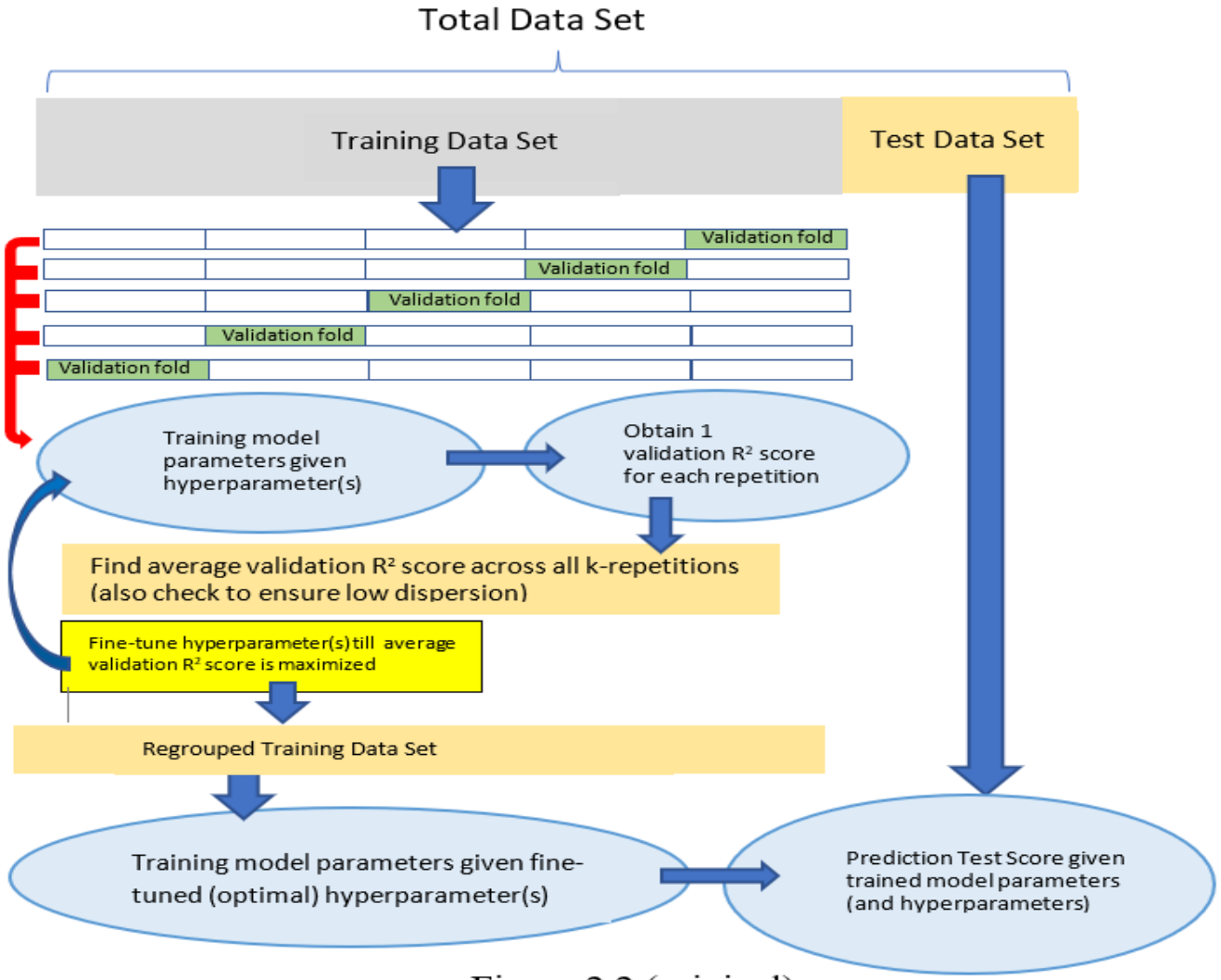



Figure 2.2 (original)

# k-Fold Cross-Validation

Code line [29] shows that the data set is reshuffled anew with a fresh X_train and y_train training set of 15,692 points. Code lines [30] to [32] computes the cross-validation $R^2$ results of using the process and procedure outlined in Figure 2.2 with k=5 and involving only the original training set of 15,692 points.

```
In [29]: ### we can try reshuffling the X_train,y_train point positions here if we wish
         from sklearn.utils import shuffle
         X_shuffle, y_shuffle = shuffle(X_train,y_train, random_state=40)
         ### Rename X_Shuffle y_shuffle
         X_train = X_shuffle
         y_train = y_shuffle
         ### shuffle just randomly permutates the rows

In [30]: ### Now k-fold cross validation is to be performed on X_train y_train reshuffled dataset as in [29], leaving test set intact
         ### The training set here de facto combines a training and validation data sets
         ### Predicting Cross Validation Score
         ### By default cross_val_score uses the scoring provided in the given estimator, which is r2 score here
         ### Details of other scoring methods and metrics can be found in https://scikit-learn.org/stable/modules/model_evaluation.html

         from sklearn.model_selection import cross_val_score
         scoreslinreg = cross_val_score(estimator = Linreg, X = X_train, y = y_train, cv = 5)
         ### Note the combined training set is 80% as in [18], randomized by seed = random_state number
         ### This combined training set X_train, y_train is split into k=5 (cv=5) folds for each of k=1,2,...,5 repetitions
         print(scoreslinreg)
         ### Score is R2 measure, there are 5 scores since k=cv=5, one for each repetition
         print("%0.4f mean R2 with a standard deviation of %0.4f" % (scoreslinreg.mean(), scoreslinreg.std()))

[0.6103278  0.61590082 0.6233678  0.61861227 0.62083429]
0.6178 mean R2 with a standard deviation of 0.0045
```

# k-Fold Cross-Validation

Ridge Regression

In [31]:
```
### Now k-fold cross validation is to be performed on X_train y_train dataset in [29], leaving test set intact
### The training set here de facto combines a training and validation data sets
### Predicting Cross Validation Score
### By default cross_val_score uses the scoring provided in the given estimator, which is r2 score here

from sklearn import linear_model
Ridge=linear_model.Ridge(alpha=0.05,)
### Do not need to specify Ridge.fit(X_train, y_train)
### training the algorithm, note regression is done without normalizing the X, y (X already scaled in [17])
### Above steps may be repeated in order to try different hyperparameter of alpha and select one with highest ave R2

from sklearn.model_selection import cross_val_score
scoresRidge = cross_val_score(estimator = Ridge, X = X_train, y = y_train, cv = 5)
### Note the combined training set is 80% as in [18], randomized by seed = random_state number
### This combined training set X_train, y_train is split into K=5 (cv=5) folds for each of R=1,2,...,3 repetitions
print(scoresRidge)
### Score is R2 measure, there are 5 scores since R=cv=5, one for each repetition
print("%0.4f mean R2 with a standard deviation of %0.4f" % (scoresRidge.mean(), scoresRidge.std()))
```

```
[0.61003575 0.61517579 0.62265187 0.62204011 0.61909958]
0.6158 mean R2 with a standard deviation of 0.0045
```

# k-Fold Cross-Validation

Lasso Regression

```
In [12]: ### Now k-fold cross validation is to be performed on X_train y_train reshuffled dataset in [20], leaving test set intact
         ### The training set here de facto combines a training and validation data sets
         ### Predicting Cross Validation Score
         ### By default cross_val_score uses the scoring provided in the given estimator, which is r2 score here

         from sklearn import linear_model
         Lasso=linear_model.Lasso(alpha=0.001,)
         ### Do not need to specify Lasso.fit(X_train, y_train)
         ### training the algorithm, note regression is done without normalizing the X, y
         ### Above steps may be repeated in order to try different hyperparameter of alpha and select one with highest ave R2

         from sklearn.model_selection import cross_val_score
         scoresLasso = cross_val_score(estimator = Lasso, X = X_train, y = y_train, cv = 5)
         ### Note the combined training set is 80% as in [18], randomized by seed = random_state number
         ### This combined training set X_train, y_train is split into k=1 (cv=1) folds for each of k=1,2,...,5 repetitions
         print(scoresLasso)
         ### Score is R2 measure, there are 5 scores since k=cv=5, one for each repetition
         print("%0.4f mean R2 with a standard deviation of %0.4f" % (scoresLasso.mean(), scoresLasso.std()))

[0.60127122 0.60612945 0.61339057 0.61278361 0.60809822]
0.6103 mean R2 with a standard deviation of 0.0073
```

## k-fold Cross-Validation Scores

Please upload Chapter2-1.ipynb and follow the computing steps in Jupyter Notebook

Figure 2.2 with k=5 and involving only the original training set of 15,692 points. There are 5 $R^2$'s for each fold. Their average or mean as well as their standard deviation are computed. The validation $R^2$ score mean and standard deviation (these are strictly training scores) are reported below the linear regression with no constraints, and the Ridge and Lasso regressions.

| | Mean Validation Scores $R^2$ | Standard Deviation of Validation Scores $R^2$ |
|---|---|---|
| Linear Regression (no constraints) | 0.6178 | 0.0045 |
| Ridge Regression ($\alpha = 0.05$) | 0.6181 | 0.0045 |
| Lasso Regression ($\alpha = 0.001$) | 0.6103 | 0.0073 |

# Summary

- There are advantages in the use of regularized regressions as a comparison to the linear regression model without constraints when the model may have too many features and can be overfitted using the training data.
- Cross-validation is a useful strategy to accompany regularized regressions as there is a hyperparameter in the penalty term in the objective function. However, one disadvantage of cross-validation is that it may take up more computing time with the number of repetitions.
- Another aspect in machine learning is the scaling of the features to make features more comparable.
- When the prediction problem does not have too many features, then using regularizations, particularly Lasso, may yield under-fitting when coefficients are forced to be less impactful. Thus, regularizations should be done only when it improves the fitting and testing accuracies.
- There are many studies showing advantages of regularizations – see Jorge Chan-Lau (2017), and Xin and Khalid (2018). As another example, using Kaggle data set on cruise_ship_info, a high predictive accuracy is obtained and improved marginally with regularization.
- By today's data standards, for predicting real estate values professionally, a lot more features can and should be added such as: recently transacted values of similar neighboring plots or blocks or units, distances/connections to transport nodes such as highways, train/bus stations, distances/connections to markets/shops/hospitals/entertainment centers, distances to childcare facilities, distances to seaside or vista points, neighborhood crime rates/theft rates, parking facilities and estate road and amenities conditions, electricity/gas supply conditions, weather conditions including flooding, hurricane situations, and so on. These additional features are likely to produce better regularized regression predictions.

## Homework 1 Graded Exercise:

### Chapter2-2.ipynb (this file is provided only after grading)

Use data set 'cruise_ship_info.csv' to predict the number of crews required to man a cruise ship given certain features/attributes of the ship. Use the features Age, TonnageGTx1000 (30.0 means 30,000 gross tons), passengersx100 (6.94 means 694 passengers), lengthx100ft (5.94 means 594 feet length of ship), cabinsx100 (3.55 means a total of 355 cabins on the ship), and spaceratio (42.64 means number of gross tonnage divided by number of passengers on ship). Target variable is Crewx100 (3.55 means 355 crews on the ship).

Use StandardScaler to standardize all the features (subtracting by mean and dividing by standard deviation).

Split the 158 rows of sample into 70% training and 30% test. In train_test_split, use random_state = 0.

Q1. Using LinearRegression in sklearn.linear.model, find $R^2$ for training data set, and $R^2$ score for the test data set.

## Homework 1 Graded Exercise:

Q2. Now use a Ridge Regression with alpha = 0.05 and report the training and test $R^2$ scores.

Q3. Now use a LASSO Regression with alpha = 0.01 and report the training and test $R^2$ scores.

Q4. Now use 4-fold cross validation (without reshuffling), on the X_train and y_train. Use Linreg and report the mean $R^2$ scores and standard deviation of $R^2$ scores in the 4 cross-validation cases.

Q5. Now use 4-fold cross validation (without reshuffling), on the X_train and y_train. Use Ridge regression (alpha = 0.05) and report the mean $R^2$ scores and standard deviation of $R^2$ scores in the 4 cross-validation cases.

# End of Class