COMP47350: Data Analytics (Conv)

Dr. Georgiana Ifrim
georgiana.ifrim@ucd.ie

Insight Centre for Data Analytics
School of Computer Science
University College Dublin

2018/19

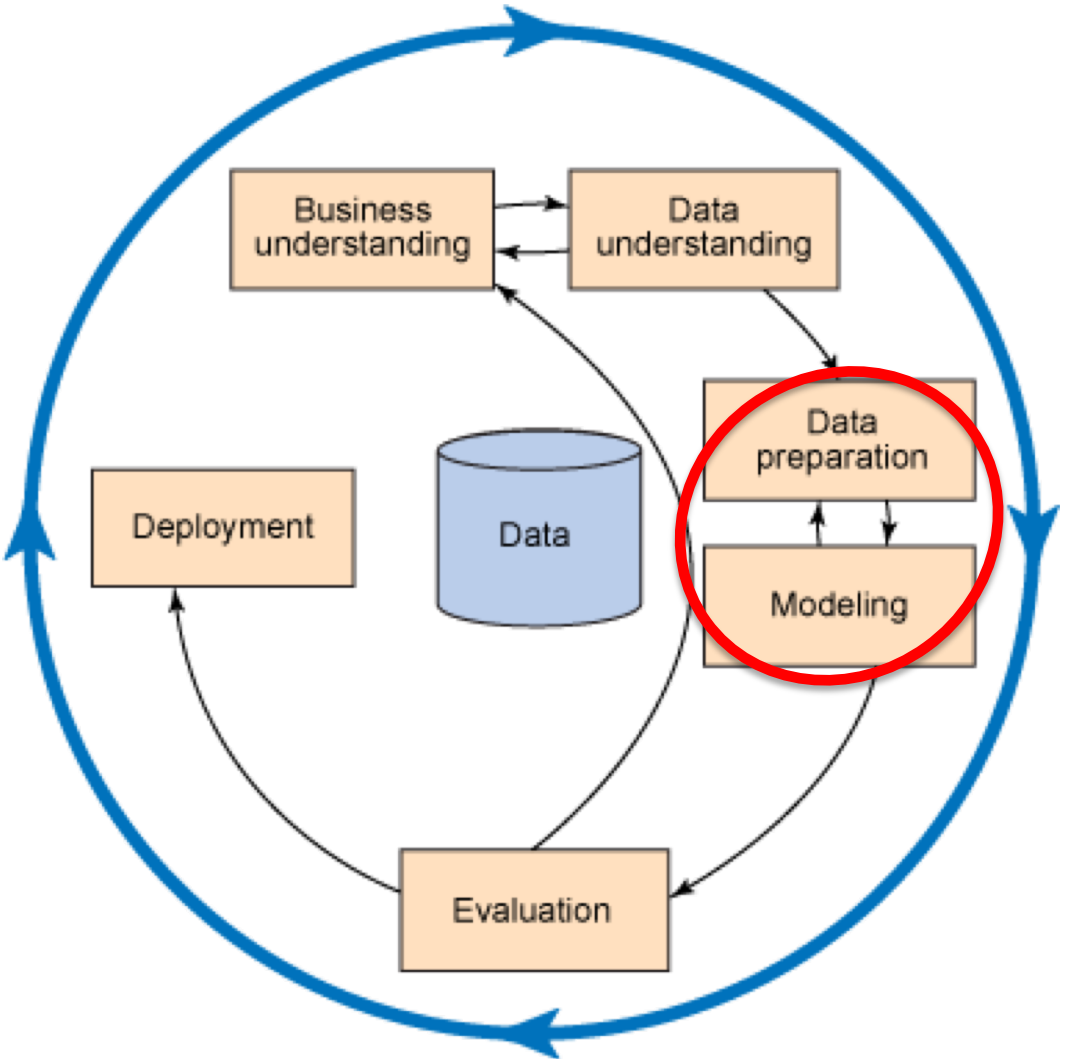
Module Topics

- **Python Environment** (Anaconda, Jupyter Notebook)
- **Getting Data** (Web scrapping, APIs, DBs)
- **Understanding Data** (slicing, visualisation)
- **Preparing Data** (cleaning, transformation)
- **Modeling & Evaluation** (machine learning)

Data Analytics Project Lifecycle:

CRISP-DM

CRISP-DM: **CR**oss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining



Today's Lecture & Lab

Linear regression hands-on

- Reading data to into a data frame; data understanding/prep (**pandas, matplotlib**)
- Training/fitting a model using Python (**scikit-learn:** **https://scikit-learn.org/stable/tutorial/basic/tutorial.html**)
- Interpreting the fitted model
- Using the trained model to predict on new data

Linear Regression

Estimates a **linear relationship** between the descriptive features and the target feature (trained model is a set of weights: w_0, w_1, …,w_n, where n is the number of features)

target_feature = w_0 + w_1 *feature_1 + w_2*feature_2 + + … + w_n*feature_n

Linear Regression: Simple vs Multivariable Regression

- Simple linear regression model can be written:

predicted_value = **w_0** + **w_1** * feature_value

- We can extend the model to use more features to train multivariable regression:

predicted_value = **w_0** + **w_1** * feature_1 + **w_2** * feature_2 + ….+ **w_n** * feature_n

Linear Regression: Evaluation

- Error between prediction and actual value:

  predicted_value = w_0 + w_1 * feature_1

- Error on single example:

  error = actual_value – predicted_value

- Sum of squared errors: sum the error over all training samples

  Total error = (error_on_example_1)^2 + (error_on_example_2)^2 + …+ (error_on_example_m)^2

Linear Regression: Measuring Error

Table: Calculating the sum of squared errors for the candidate model (with $\mathbf{w}[0] = 6.47$ and $\mathbf{w}[1] = 0.62$) making predictions for the the office rentals dataset.

ID	Size	Rental Price
1	500	320
2	550	380
3	620	400
4	630	390
5	665	385
6	700	410
7	770	480
8	880	600
9	920	570
10	1,000	620

ID	Rental Price	Model Prediction	Error Error	Squared Error
1	320	316.79	3.21	10.32
2	380	347.82	32.18	1,035.62
3	400	391.26	8.74	76.32
4	390	397.47	-7.47	55.80
5	385	419.19	-34.19	1,169.13
6	410	440.91	-30.91	955.73
7	480	484.36	-4.36	19.01
8	600	552.63	47.37	2,243.90
9	570	577.46	-7.46	55.59
10	620	627.11	-7.11	50.51
			Sum	**5,671.64**
		Sum of squared errors (Sum/2)		**2,835.82**

Linear Regression Lines

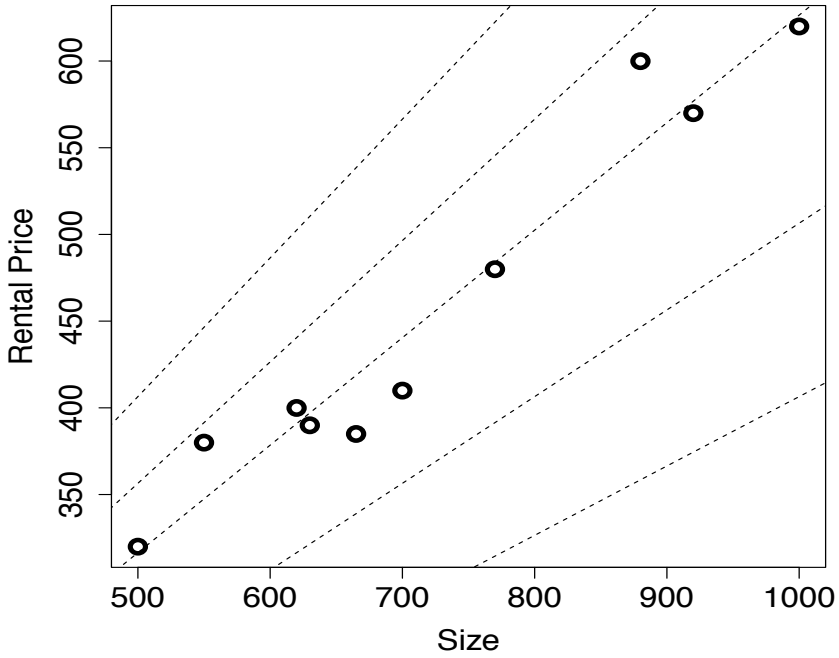


Figure: A scatter plot of the SIZE and RENTAL PRICE features from the office rentals dataset. A collection of possible simple linear regression models capturing the relationship between these two features are also shown. For all models **w**[0] is set to 6.47. From top to bottom the models use 0.4, 0.5, 0.62, 0.7 and 0.8 respectively for **w**[1].

Sum of Squared Errors

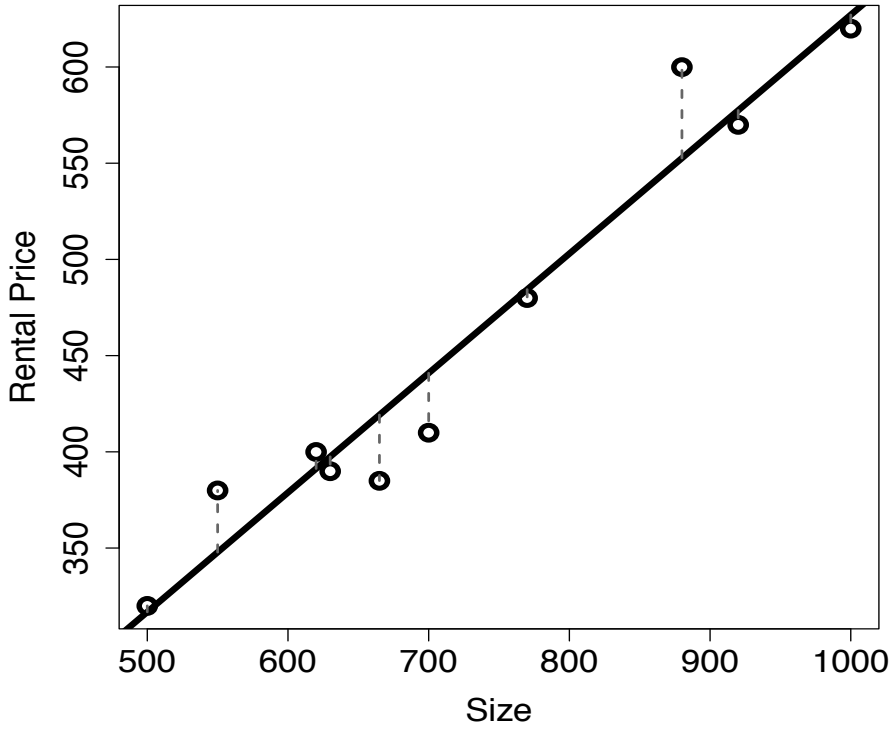


Figure: A scatter plot of the SIZE and RENTAL PRICE features from the office rentals dataset showing a candidate prediction model (with $\mathbf{w}[0] = 6.47$ and $\mathbf{w}[1] = 0.62$) and the resulting errors.

Linear Regression: Example

Table: The **office rentals dataset**: a dataset that includes office rental prices and a number of descriptive features for 10 Dublin city-centre offices.

ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

Can we predict the **rental price**, given the descriptive features (**size, floor, broadband rate, energy rating**) for an office?

Simple Linear Regression (1 feature)

Simple Regression: Predict a target feature using a single input feature.

ID	Size	Rental Price
1	500	320
2	550	380
3	620	400
4	630	390
5	665	385
6	700	410
7	770	480
8	880	600
9	920	570
10	1,000	620

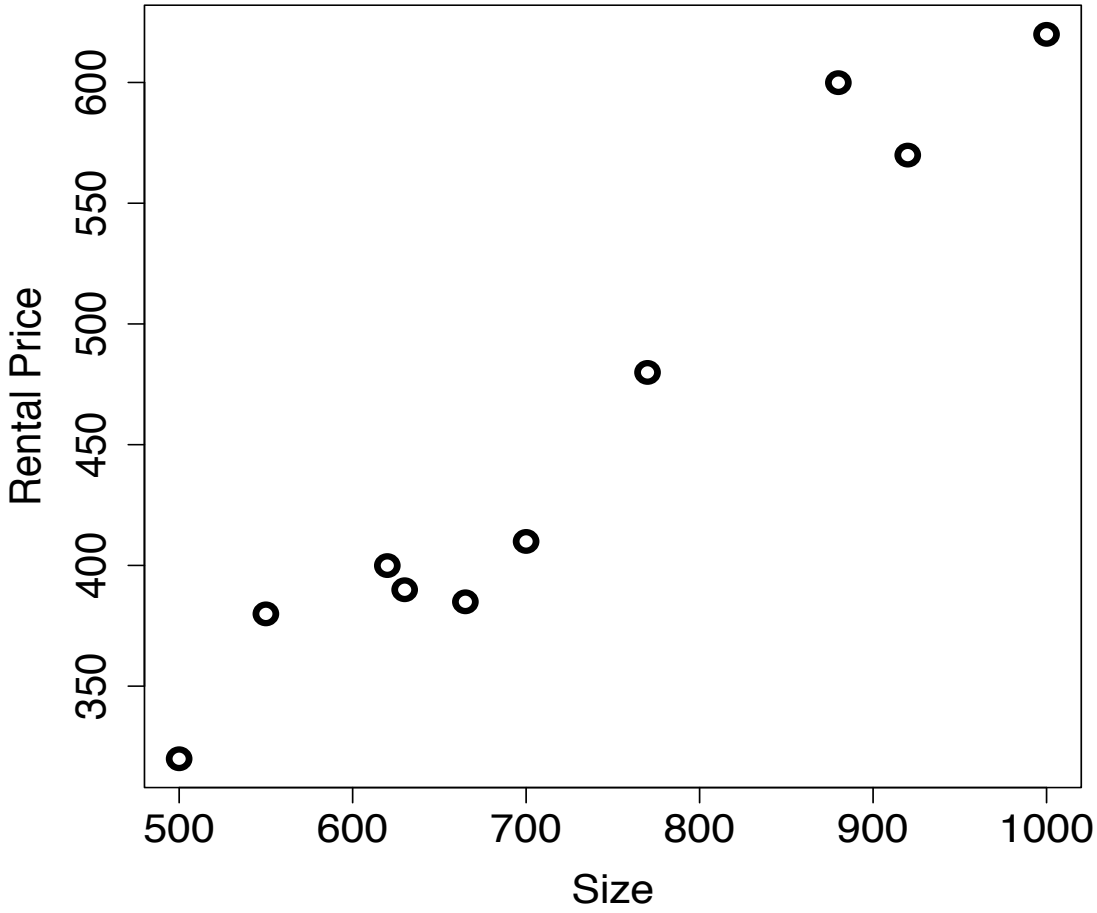
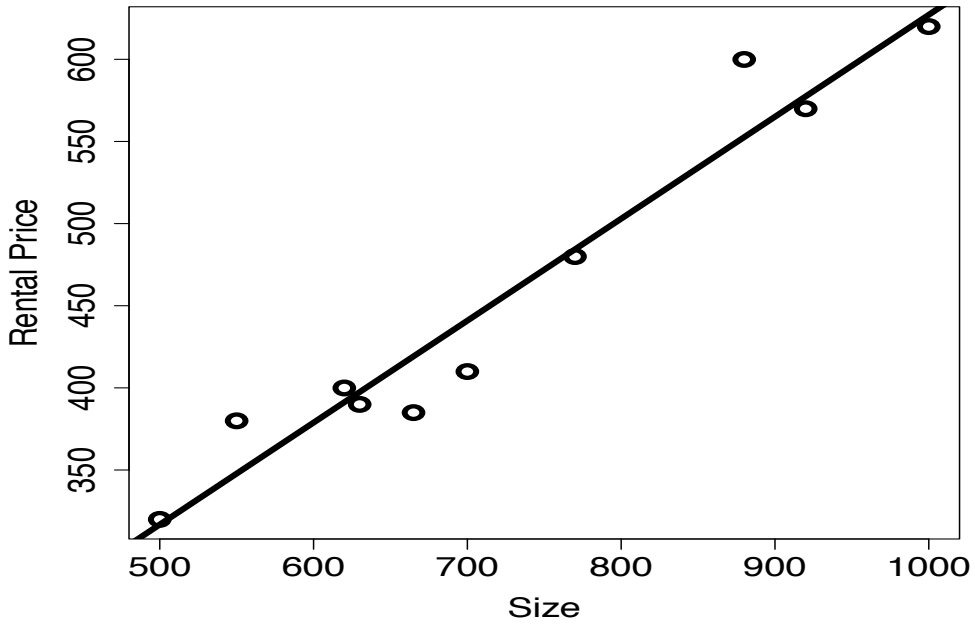


Figure: A scatter plot of the SIZE and RENTAL PRICE features from the office rentals dataset

Simple Linear Regression

- **Regression line** estimates relationship between SIZE and RENTAL PRICE
- The learned model using our training set with 10 examples is: w_0 = 6.47, w_1 = 0.62

$$\text{RENTAL PRICE} = 6.47 + 0.62 \times \text{SIZE}$$



Reading data

```
# Library Imports.
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Allows plots to appear directly in the notebook.
%matplotlib inline

from patsy import dmatrices
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
# Read a CSV dataset with 10 example offices into a dataframe.
# The data is described by 5 features (4 descriptive features: Size, Floor, BroadbandRate, EnergyRating;
# the target feature: RentalPrice).


# Read csv file into a dataframe.
df = pd.read_csv('Offices.csv')
df.head(10)
```

Out[64]:

	ID	Size	Floor	BroadbandRate	EnergyRating	RentalPrice
0	1	500	4	8	C	320
1	2	550	7	50	A	380
2	3	620	9	7	A	400
3	4	630	5	24	B	390
4	5	665	8	100	C	385
5	6	700	4	8	B	410
6	7	770	10	7	B	480
7	8	880	12	50	A	600
8	9	920	14	8	C	570

Understanding data

In [2]:
```
# Look at correlations for all the continuous features.
df[['Size', 'Floor', 'BroadbandRate', 'RentalPrice']].corr()
```

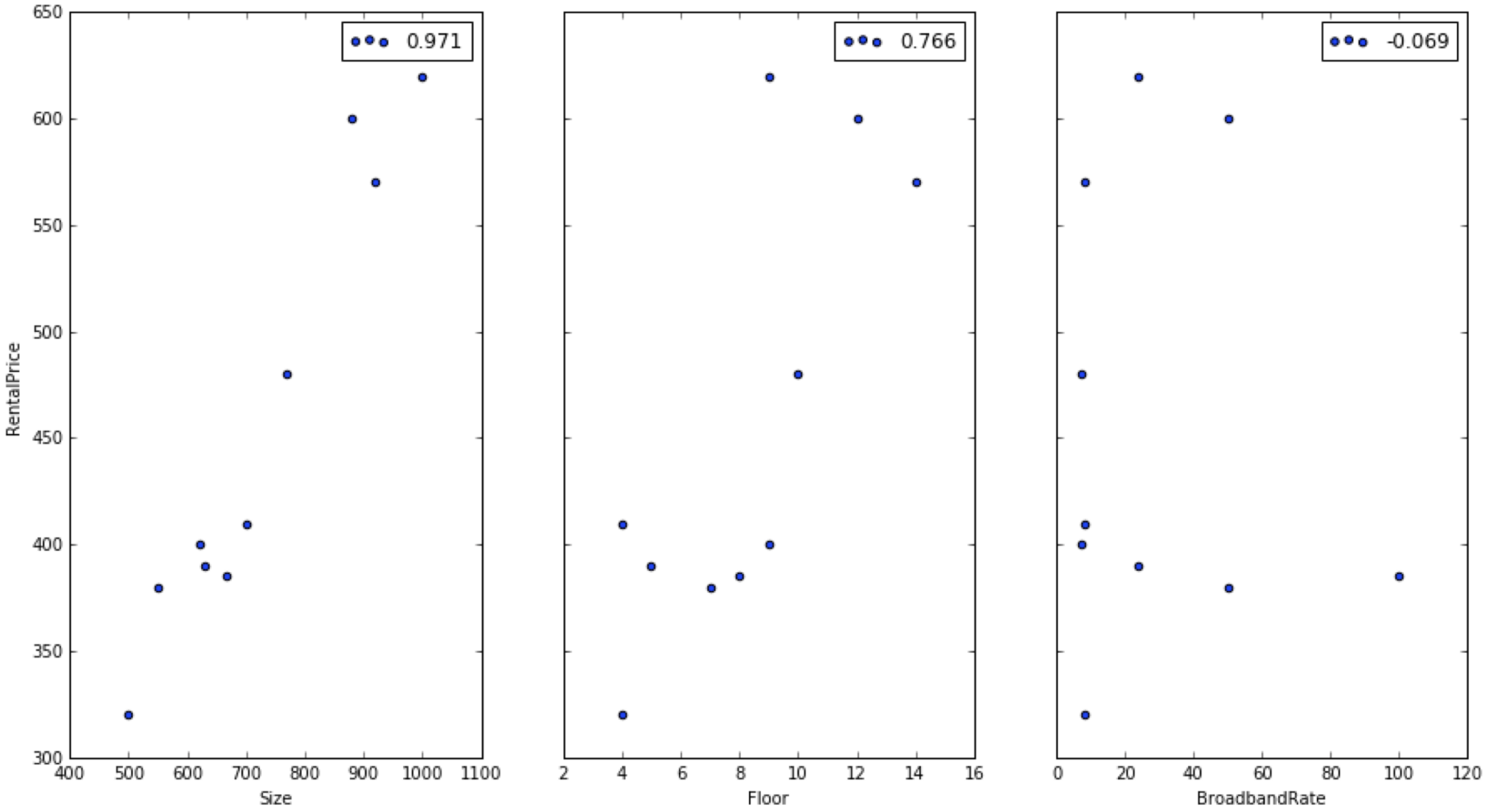
Out[2]:

	Size	Floor	BroadbandRate	RentalPrice
Size	1.000000	0.710863	-0.069117	0.971262
Floor	0.710863	1.000000	0.054897	0.766020
BroadbandRate	-0.069117	0.054897	1.000000	-0.068597
RentalPrice	0.971262	0.766020	-0.068597	1.000000

In [3]:
```
# The correlation for a given pair of features
df[['Size', 'RentalPrice']].corr().as_matrix()[0,1]
```

Out[3]:
```
0.971262060862957 3
```

```
# Scatterplots for each descriptive feature and target feature.
# Show the correlation value in the plot.
fig, axs = plt.subplots(1, 3, sharey=True)
df.plot(kind='scatter', x='Size', y='RentalPrice', label="%.3f" % df[['Size', 'RentalPrice']].corr().as_matrix()[0,1],
df.plot(kind='scatter', x='Floor', y='RentalPrice', label="%.3f" % df[['Floor', 'RentalPrice']].corr().as_matrix()[0,1
df.plot(kind='scatter', x='BroadbandRate', y='RentalPrice', label="%.3f" % df[['BroadbandRate', 'RentalPrice']].corr()
```



Linear regression with package scikit-learn

Simple linear regression (one descriptive feature)

Preparing the data

```
# Prepare the descriptive features
df
X = df[['Size', 'Floor', 'BroadbandRate']]
y = df.RentalPrice
print("Full data frame: ", df)
print("\nDescriptive features:\n", X)
print("\nTarget feature:\n", y)
```

```
Full data frame:      ID  Size  Floor  BroadbandRate EnergyRating  RentalPrice
0    1   500      4              8              C           320
1    2   550      7             50              A           380
2    3   620      9              7              A           400
3    4   630      5             24              B           390
4    5   665      8            100              C           385
5    6   700      4              8              B           410
6    7   770     10              7              B           480
7    8   880     12             50              A           600
8    9   920     14              8              C           570
9   10  1000      9             24              B           620

Descriptive features:
     Size  Floor  BroadbandRate
0    500      4              8
1    550      7             50
2    620      9              7
3    630      5             24
4    665      8            100
5    700      4              8
6    770     10              7
7    880     12             50
8    920     14              8
9   1000      9             24

Target feature:
  0     320
1     380
2     400
3     390
4     385
5     410
6     480
```

Testing the model

Using the trained model to predict the target feature RentalPrice, given the descriptive feature Size.

```
# Predicted scores for each example.
predicted_scores = linreg.predict(X[['Size']])
print("Predicted scores:", predicted_scores)
print("\nPrediction for first example (Size, PredictedScore): ", X['Size'].values[0], predicted_scores[0])
```

```
Predicted scores: [316.78694141 347.81894557 391.26375139 397.47015222 419.19255513
 440.91495804 484.35976387 552.63017302 577.45577635 627.106983  ]

Prediction for first example (Size, PredictedScore):  500 316.78694140557286
```

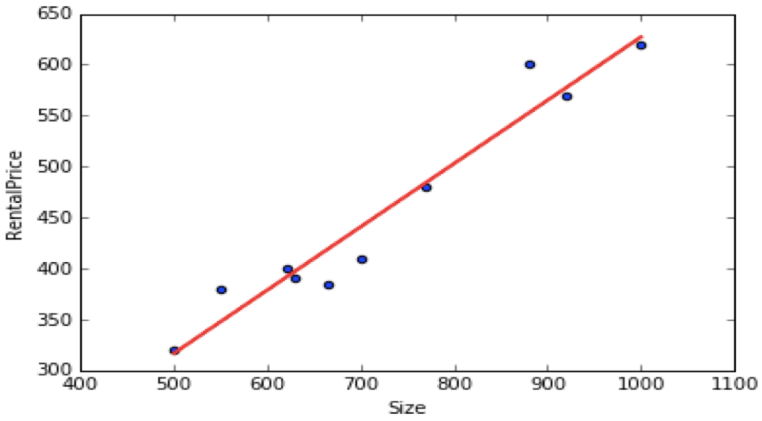
```
# The model learned is: RentalPrice = 6.46690 + 0.62064 * Size
# This means that for a unit increase in Size, we have a 0.62064 increase in RentalPrice.
# We can use the learned model to predict the RentalPrice for a new office Size.

# The Statsmodels predict() method expects a dataframe object.
# Create a new dataframe with a new test example.
X_new = pd.DataFrame({'Size': [730]})
X_new.head()
```

	Size
0	730

Plotting the trained model

```
# We can plot the regression line that was estimated from our trainign set.
# First we use the trained model to predict prices for the min and max Size
X_minmax = pd.DataFrame({'Size': [df.Size.min(), df.Size.max()]})
X_minmax.head()
```

	Size
0	500
1	1000




```
# Make predictions for the min and max Size values and store them.
predictions = lm.predict(X_minmax)
predictions
```

```
array([ 316.78694141,  627.106983  ])
```

```
# First, plot the observed data
df.plot(kind='scatter', x='Size', y='RentalPrice')

# Next, plot the regression line, in red.
plt.plot(X_minmax, predictions, c='red', linewidth=2)
```

Multiple linear regression (using more than one feature)

Training the model

```
# Use more features for training
# Train aka fit, a model using all continuous features.
multiple_linreg = LinearRegression().fit(X, y)

# Print the weights learned for each feature.
print(multiple_linreg.coef_)
```

```
[ 0.54873985  4.96354677 -0.06209515]
```

Testing the model

Using the trained model to predict the target feature RentalPrice, given the descriptive features Size, Floor BroadbandRate.

```
multiple_linreg_predictions = multiple_linreg.predict(X)

print("Predictions with single linear regression: " + str(predicted_scores))
print("\nPredictions with multiple linear regression: " + str(multiple_linreg_predictions))
```

```
Predictions with single linear regression: [316.78694141 347.81894557 391.26375139 397.47015222 419.19255513
 440.91495804 484.35976387 552.63017302 577.45577635 627.106983  ]

Predictions with multiple linear regression: [313.28890799 353.00854431 404.01751854 388.5951124  417.97241593
 423.03687728 491.29204228 558.91042747 593.39511116 611.48304265]
```

Sampling the training and test data.

```
# Split the data into train and test sets
# Take a third (random) data samples as test data, rest as training data
# Note that this training set if very small and the model will not be very reliable due to this sample size problem
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

print("Training data:\n", pd.concat([X_train, y_train], axis=1))
print("\nTest data:\n", pd.concat([X_test, y_test], axis=1))
```

```
Training data:
     Size  Floor  BroadbandRate  RentalPrice
3    630      5             24          390
0    500      4              8          320
8    920     14              8          570
7    880     12             50          600
2    620      9              7          400
4    665      8            100          385
5    700      4              8          410

Test data:
     Size  Floor  BroadbandRate  RentalPrice
6    770     10              7          480
9   1000      9             24          620
1    550      7             50          380
```

```
# Train on the training sample and test on the test sample.
linreg_train = LinearRegression().fit(X_train, y_train)
# Print the weights learned for each feature.
print(linreg_train.coef_)
```

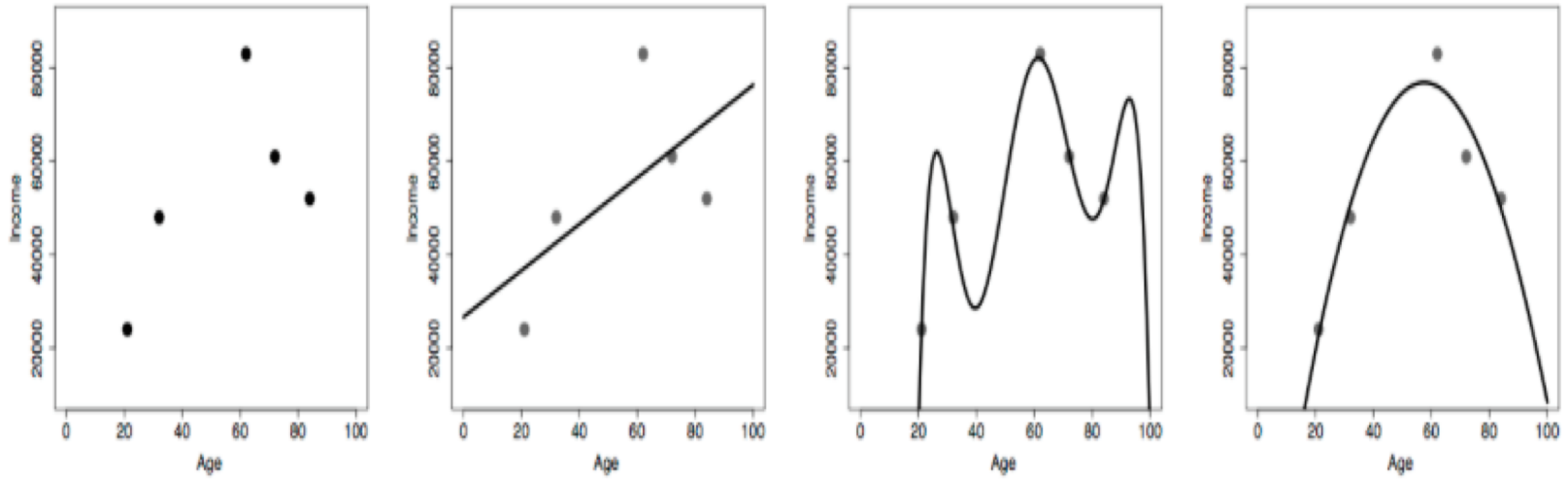
```
[ 0.57286399  4.7793226  -0.17363329]
```

```
# Estimated classes  on test set
y_predicted = linreg_train.predict(X_test)
print(y_predicted)
```

```
[491.58681475 615.61444285 343.75253853]
```

Model evaluation: Training versus test error

Evaluating the model quality only on the training data is misleading. This happens due to the fact that more complex models (e.g., more features) can fit the training data better, but do not generalise to new data, since they pick up too many unimportant details from the training data. The more complexity we add to the model, the lower the training error tends to get, and the higher the test error. This is called over-fitting the data. An example is shown in the picture below.



(a) Dataset (b) Underfitting (c) Overfitting (d) Just right

Figure: Striking a balance between overfitting and underfitting when trying to predict age from income.

Optimism of Training Error

[http://statweb.stanford.edu/~tibs/ElemStatLearn/, Chapter 7]

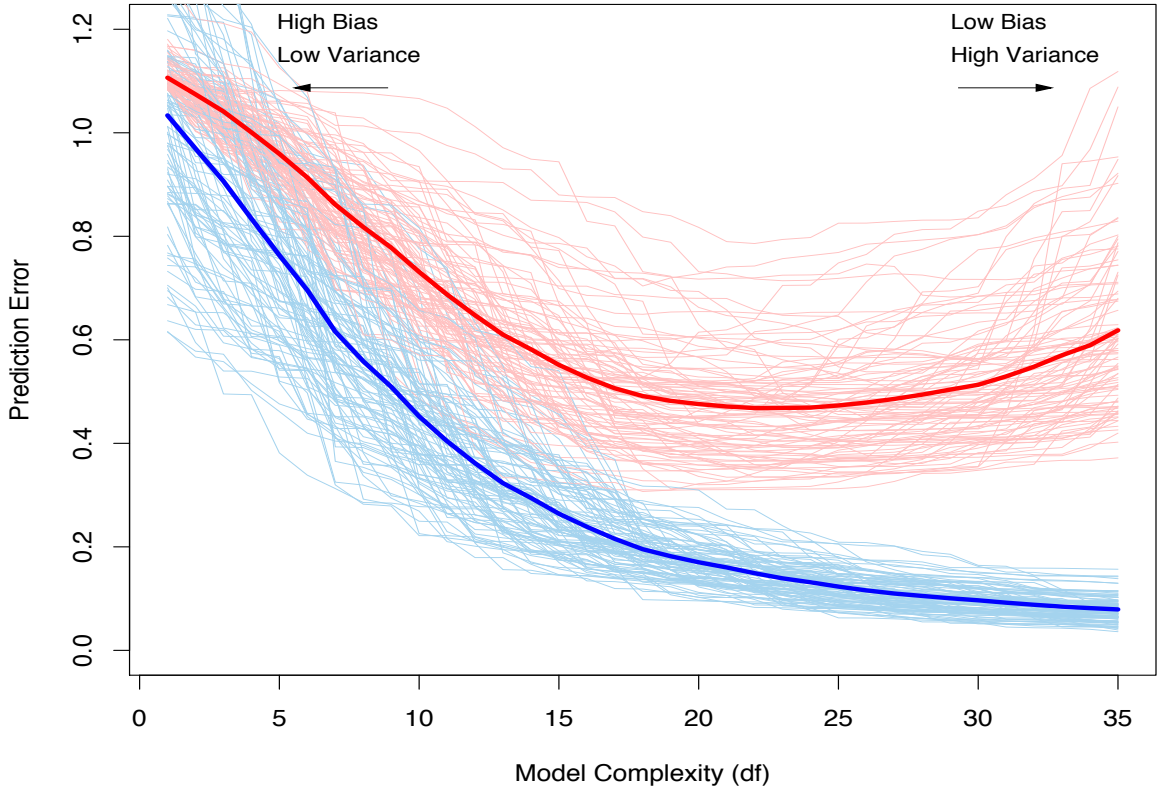


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$*.*

References

- Chapter7 from **FMLPDA Book: Fundamentals of Machine Learning for Predictive Data Analytics**, by J. Kelleher, B. Mac Namee and A. D'Arcy, MIT Press, 2015 (machinelearningbook.com)
- Chapter3 from **An Introduction to Statistical Learning**, by G. James, D. Witten, T. Hastie, R. Tibshirani, Springer, 2016 (free book: http://www-bcf.usc.edu/~gareth/ISL/)
- A friendly introduction to linear regression (using Python): http://www.dataschool.io/linear-regression-in-python/
- http://ipython-books.github.io/featured-04/
- http://statweb.stanford.edu/~tibs/ElemStatLearn/