Statistics

In this assessment, we will use the Python libraries `Pandas` and `NumPy` to perform basic operations on the data relating to food hygiene in Wandsworth. In addition we will also perform linear regression on data about vehicle mileage and price.

Run the cell below each time you load the page to make sure that all the imports are done correctly

In [12]:

```
import numpy as np
import pandas as pd

from bokeh.io import output_notebook, show
from bokeh.charts import *
output_notebook()

from nose.tools import *

import pymongo
from pymongo import MongoClient
client = MongoClient('mongodb://cpduser:M13pV5woDW@mongodb/health_data', 27017)
db = client.health_data

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

print('All libraries successfully loaded')
```

(http://bokeh.pydata.org) Loading BokehJS ...

```
All libraries successfully loaded
```

Question 1: Statistics

Question 1(a) [4 marks]

Create a **function** `evaluate_random_data` which has the following **parameters**:

- `n` the size of the Pandas `Series` you are to create
- `evaluation` a string should say which task to perform on the `Series` object.
- `seed` an integer which should instantiate a numpy `RandomState` instance, with a default value of 543210

The function should do the following:

- Create an array of size `n` which is full of random numbers integers between 0 and 100 (inclusive)
- The `evaluation` should be able to perform mean, median and standard deviation functions on the created array
- The function should **return** the output of the evaluation on the array. It should be able to cope with `"mean"`, `"median"` and `"std"` (standard deviation).

Hint: Week 4, Guided Exercise 2, Randomness - Importing Data - Bokeh Charts

In [68]:

```
def evaluate_random_data(n, evaluation, seed=543210):
    # YOUR CODE HERE
    low = 0
    high = 101
    rs = np.random.RandomState(seed)
    j = rs.randint(low, high, size = n )
    listy = list(j)
    s = pd.Series(listy)
    if evaluation is 'mean':
        evaluation = s.mean()
    if evaluation is 'median':
        evaluation = s.median()
    if evaluation is 'std':
        evaluation = s.std()

    return evaluation

    raise NotImplementedError()
```

In [69]:

```
# Check a RandomState object is instantiated
tmp_rs = np.random.RandomState
del np.random.RandomState

try:
    evaluate_random_data(100, 'mean')
except AttributeError:
    print('The function correctly uses RandomState')
else:
    raise AssertionError('The function does not use RandomState')
finally:
    np.random.RandomState = tmp_rs


assert_equal(evaluate_random_data(100, 'mean', 511419), 47.39)
assert_equal(evaluate_random_data(100, 'mean', 364235), 48.45)

assert_equal(evaluate_random_data(100, 'median', 511419), 46)
assert_equal(evaluate_random_data(100, 'median', 364235), 44)

output = evaluate_random_data(100, 'std', 511419)
"{0:.4f}".format(round(output,4))
assert_equal("{0:.4f}".format(round(output,4)), '29.2968')

output = evaluate_random_data(100, 'std', 364235)
"{0:.4f}".format(round(output,4))
assert_equal("{0:.4f}".format(round(output,4)), '30.5159')

print('All tests successfully passed')
```

```
The function correctly uses RandomState
All tests successfully passed
```

Question 1(b) [3 marks]

The function `get_data` (provided) returns a Pandas `Series` for the given collection. Using data from this function, and the PyMongo `collection_names` function, write a function `get_means` which takes the mean `RatingValue` from the first `n` collections in the `collection_names()` function.

- You should **sort** `collection_names` in ascending alphabetical order. Note that this is a Python `list` rather than a `Cursor`, so use **this guide to sorting lists (https://wiki.python.org/moin/HowTo/Sorting)** to help you
- The function should **return** a **Pandas `Series` object**, with a **name** of **`RatingValueMeans`**.
- You should use the function from Question 1(a) to get a `Series` object of the `RatingValue`, and **obtain the mean** from that object
- N.B. Be very careful you properly exit your loop!

Hint: Week 3, Guided Exercise 4, Cursors
Hint: Week 4, Guided Exercise 2, Importing Data - Bokeh Charts

In [13]:

```
def get_data(collection):
    cursor = collection.find()
    rating_values = pd.DataFrame(list(cursor))['RatingValue']
    return rating_values

def get_means(n):

    # YOUR CODE HERE
    i = 0
    j = []
    while i < n:
        cn = sorted(db.collection_names(), key=str.lower)[i]
        RV_Means = get_data(db[cn]).mean()
        j.append(RV_Means)

        i += 1

    RatingValueMeans = pd.Series(j)

    return RatingValueMeans

    raise NotImplementedError()
```

In [5]:

```
# You don't need to write anything here
means = get_means(10)

assert_equal(type(means), pd.Series)
assert_equal(len(means), 10)
assert_equal(round(means.sum(), 4), 27.2436)
means = get_means(12)
assert_equal(len(means), 12)
assert_equal(round(means.sum(), 4), 36.2604)
print('All tests passed successfully')
```

```
All tests passed successfully
```

Question 1(c) [4 marks]

Create a function `get_sample_mean_distribution` which **returns** a series of the distribution of sample means of the `RatingValue` of the data from a given series. The function should be defined as follows:

- `data`: A `Series` object of the mean ratings of establishments
- `n`: integer, the size of the sample
- `m=1000`: integer, the amount of times to repeat the procedure
- `seed=543210`: integer, default value 543210. This should set a `RandomState` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.RandomState.html) instance to be initialised with this value as an argument.
- You should use the Pandas `sample` (http://pandas.pydata.org/pandas-docs/version/0.19.2/generated/pandas.DataFrame.sample.html) function to generate your samples.

The function should **return** the output as a **Pandas `Series` object**.

Hint: Week 4, Guided Exercise 2, Randomness

Type *Markdown* and LaTeX: α^2

In [90]:

```
def get_sample_mean_distribution(data, n, m=1000, seed=543210):

    # YOUR CODE HERE
    j = []
    ds = data.sample(n = n, random_state = seed) #시리즈에서 랜덤 샘플을 뽑고
    seed = np.random.RandomState(seed=seed)
    for ri in range(0, m):
        ds = data.sample(n = n, random_state = seed)
        want = ds.mean()
        j.append(want)

    Rwant = pd.Series(j)
    return Rwant
    raise NotImplementedError()
```

In [91]:

```
# You don't need to write anything here
m = 1000
n = 20
data = pd.Series([5, 5, 5, 5, 5, 4, 5, 4, 5, 4, 5, 5, 3, 4, 4, 5, 5, 5, 4, 5, 5, 5,
seed = 123456
actual_output = get_sample_mean_distribution(data, n, m, seed)

assert_equal(len(actual_output), 1000)
assert_equal(round(actual_output.sum(),4), 4528.6)

# Check randomness is working:
# The same seed should lead to the same result
seed_test_equal = get_sample_mean_distribution(data, n, m, seed)
assert_equal(actual_output.mean(), seed_test_equal.mean())

# A different seed should not be equal
seed_test_not_equal = get_sample_mean_distribution(data, n, m, 54321)
assert_not_equal(actual_output.mean(), seed_test_not_equal.mean())

print('All tests successfully passed')
```

```
All tests successfully passed
```

Question 1(d) [2 marks]

Using the output from Question 1(c), **plot a histogram** illustrating the distribution, and **return** the histogram as the output of the function `plot_sample_mean_distribution`.

This function should take the same parameters as the `get_sample_mean_distribution`, so that it can call that function to obtain the data. The histogram should have **10 bins**.

Hint: Week 1, Guided Exercise 1, Python Primer
Hint: Week 2, Guided Exercise 2, Python for Data Science
Hint: Week 4, Guided Exercise 2, IQR & Outliers - Bokeh Charts
Hint: Week 4, Guided Exercise 3, Residual Analysis

In [109]:

```
def plot_sample_mean_distribution(data, n, m=1000, seed=543210):

    # YOUR CODE HERE
    k = get_sample_mean_distribution(data, 20, 1000, 123456)
    histo = pd.DataFrame({'RatingValue': k})
    from bokeh.plotting import figure, show, output_file
    output_notebook()
    hist = Histogram(histo, bins=10)
    # Show absolute number on axis rather than E notation:
    show(hist)

    return hist
    raise NotImplementedError()
```

In [110]:

```
# You don't need to write anything here
data = pd.Series([5, 5, 5, 5, 5, 4, 5, 4, 5, 4, 5, 5, 3, 4, 4, 5, 5, 5, 4, 5, 5, 5,

# assert_equal())
assert_equal(type(plot_sample_mean_distribution(data, 20)), Chart)
# Check Histogram
old_hist = Histogram
del Histogram

try:
    plot_sample_mean_distribution(data, 20)
except NameError:
    pass
else:
    raise AssertionError("The chart does not appear to be a Histogram")
finally:
    Histogram = old_hist
    del old_hist

print('All tests successfully passed')
```

(http://bokeh.pydata.org) Loading BokehJS ...

(http://bokeh.pydata.org) Loading BokehJS ...

```
All tests successfully passed
```

Question 2: Linear Regression

This question uses a different dataset in order to illustrate linear regression. We'll use a dataset similar to the cars dataset given as an example in the videos.

Question 2(a) [1 mark]

Create a function `get_car_data` which uses the Pandas `read_csv` to import the data in the `cars.csv` file and **return** a `DataFrame` of the data (The `cars.csv` file is in the same directory in the notebook)

Hint: Week 4, Guided Exercise 3, Classification - Importing Data

In [15]:

```
import numpy as np
import pandas as pd
import sklearn
import sklearn.naive_bayes as nb
import sklearn.feature_extraction.text as text
import sklearn.model_selection as cv



def get_car_data():
    # YOUR CODE HERE
    cars_data = pd.read_csv('cars.csv')


    return cars_data

    raise NotImplementedError()
```

In [167]:

```
# You don't need to write anything here
assert_equal(len(get_car_data()), 1155)
assert_equal(type(get_car_data()), pd.DataFrame)
print('All tests successfully passed')
```

```
All tests successfully passed
```

Question 2(b) [3 marks]

Create a **function** `remove_outliers` which removes the outliers from the dataset according to the $lower - (1.5 * IQR)$ and $upper + (1.5 * IQR)$ rule. The function should have the following parameters:

- `data` a Pandas `DataFrame`
- `field` a string, which will say for which of the fields in the `DataFrame` to locate and remove outliers

The function should **return** the `DataFrame` without the outliers.

Hint: Week 4, Guided Exercise 2, IQR
Hint: Week 4, Guided Exercise 3, Residual Analysis

In [34]:

```
def remove_outliers(data, field):
    # YOUR CODE HERE
    #gc = get_car_data()

    X = data[field]

    upper = float(X.quantile(0.75)) #상위 사분위수
    lower = float(X.quantile(0.25)) #하위 사분위수
    iqr = upper - lower #사분범위 IQR
    data = data[(X > (float(lower) - (iqr * 1.5)))
                & (X < (float(upper) + (iqr * 1.5)))]

    return data
    raise NotImplementedError()
```

In [35]:

```
# You don't have to write anything here
data = get_car_data()

data = remove_outliers(data, 'Price')
assert_equal(data.count()['Price'], 1101)


data = remove_outliers(data, 'Mileage')
assert_equal(data['Mileage'].count(), 1100)

data = remove_outliers(data, 'Year')
assert_equal(data['Year'].count(), 1092)
print('All tests successfully passed')
```

```
All tests successfully passed
```

Question 2(c) [2 marks]

Create a function `scatter_feature`, which generates a **scatter** plot of data. The function should take two parameters: `data`, a `DataFrame` of the regression data, and `feature_name`, the column name of the feature to be used as a predictor variable. The following should also be applied:

- The **response** variable is the `Price` columnn
- The function should return a **`Chart`** instance of the scatter plot.

Hint: Week 4, Guided Exercise 2, Bokeh Charts

In [455]:

```
def scatter_feature(data, feature_name):
    """
    This function returns a scatter plot for the particular `feature_name` v Price
    `data` should be a `DataFrame`
    `feature_name` should be a string
    """
    # YOUR CODE HERE
    data = get_car_data()

    X = data[[feature_name]] #꽃잎 길이 = 예측 변수 (예측해서 사용을 하면)
    #한개 이상의 열이 있을지도 모르기에 두 쌍의 괄호를 사용했음
    y = data['Price'] #꽃받침 길이 = 반응 변수 (그에 따라 반응하는 변수)

    fig = Figure() #시각화 프로그램인 보케를 이용하여 차트를 만들거나 산포 그래프를
    #무언가에 할당해야하므로 Figure()를 fig에 저장 !
    fig.circle(X[feature_name],y)
    return fig

    raise NotImplementedError()
show(scatter_feature(data, 'Mileage'))
show(scatter_feature(data, 'Year'))
```

Question 2(d) [3 marks]

Using the `scikit-learn` libarary, write a function `split_data` which generates a **training** and a **test** dataset, based on features passed in as arguments. The function should have three parameters:

- **`data`** a `DataFrame` containing the data to be evaluated
- **`features`** a list of strings which will be used to select the features from the dataset
- **`seed`** an integer with default value `543210`, which should be used to initialise a `RandomState` (as with Question 1(a),
- The function should return the output of **`train_test_split` (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)**, using that `RandomState`, and data with the selected features from the `features` parameter.
- The **test size** of the returned data should be **20%**
- As with question 2c the **response** variable shuld be the **'Price'** columnn

Hint: Week 4, Guided Exercise 3, Training and Testing Data

In [20]:

```
def split_data(data, features, seed=543210):
    # YOUR CODE HERE
    X = data[features]
    y = data['Price']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
    #총 네가지 값이 있음. 이 결과값이 각각에 할당 됨

    return X_train, X_test, y_train, y_test
    raise NotImplementedError()
```

In [21]:

```
# You don't need to write anything here
data = get_car_data()

x1, x2, y1, y2 = split_data(data, ['Mileage'])
assert_equal(len(x1), 924)
assert_equal(len(x2), 231)

assert_equal(x1.index[0], 607)
x1, x2, y1, y2 = split_data(data, ['Mileage'], 123456)
assert_equal(x1.index[0], 642)
print("All tests successfully passed")
```

```
All tests successfully passed
```

Question 2(e) [4 marks]

Create a function `evaluate_model` to **fit** and **evaluate** the linear regression model using the test the accuracy of the model with different features using R^2. The function should contain the same parameters as `split_data`, so it can call that function.

The model should be evaluated based on the test data.

Hint: Week 2, Guided Exercise 2, Exercise 3a
Hint: Week 4, Guided Exercise 3, Lineaar Regression - Fitting The Model - Residual Analysis

In [75]:

```
def evaluate_model(data, features, seed=543210):
    # YOUR CODE HERE
    x11, x22, y11, y22 = split_data(data, features, seed)
    lm = LinearRegression()
    lm.fit(x11, y11)
    m = lm.coef_[0] #coef_, intercept_ 둘 다 선형 회귀 라이브러리의 함수임 !
    c = lm.intercept_


    return lm.score(x22, y22)

    raise NotImplementedError()


evaluate_model(get_car_data(), ['Year'])
```

Out[75]:

```
0.35490873947097612
```

In [76]:

```
## You don't need to write anything here

data = get_car_data()

assert_equal(round(evaluate_model(data, ['Mileage'], 511419), 4),0.4742)
assert_equal(round(evaluate_model(data, ['Year'], 511419), 4), 0.2526)
assert_equal(round(evaluate_model(data, ['Year', 'Mileage'], 511419), 4), 0.4754)
print("All tests successfully passed")
```

```
All tests successfully passed
```

In [ ]:

```

```

In [ ]:

```

```