Homework 1

Introduction to Data Science EN.553.436/EN.553.636 - Fall 2021

Due date: Wednesday, September 22 at midnight.

Guidelines

- Answer in the cells immediately below the problem statements. If a problem calls for code, a code cell will follow. If a problem calls for a text response, a Markdown cell will follow.
- Your code should include some comments. Insufficient commentary may result in loss of points. But you do not necessarily need to comment every line or problem. Further guidelines:
  - If the the meaning of some line of code would be obvious to the average Python novice, you do not need to comment (e.g., you do not need to comment *import numpy*).
  - If your code is moderately complex, you should comment (e.g., if you nest several functions in one line, you should comment).
  - It may be proper to comment before a code block to describe generally what you are doing (e.g., you should comment before a function definition to explain the function and its parameters).

Problem 1

1.1

Load the *lowbwt* dataset from the OpenML repository as a Pandas DataFrame from following URL: https://www.openml.org/data/get_csv/3640/dataset_2189_lowbwt.arff. Use a function that is able to handle loading the data directly into Jupyter from the URL. The function should take the URL as an argument. **Do not load the data using a filepath on your hard drive:** again, load the data directly into Jupyter using the URL.

Print the loaded DataFrame. Read the description of the dataset to better understand it. Check the column names and values to see if they match the variables discussed in the description. One or more variables may have been renamed.

In [94]:
```
# Module for arrays.
import numpy as np
# Module for dataframe manipulation.
import pandas as pd
# Function for downloading from URLs.
from urllib import request
```

In [95]:
```
data1=pd.read_csv('https://www.openml.org/data/get_csv/3640/dataset_2189_lowb
print(data1)
#birth weight is class in the dataset
```

```
     LOW  AGE  LWT  RACE  SMOKE  PTL  HT  UI  FTV  class
0      0   19  182     2      0    0   0   1    0   2523
1      0   33  155     3      0    0   0   0    3   2551
2      0   20  105     1      1    0   0   0    1   2557
3      0   21  108     1      1    0   0   1    2   2594
4      0   18  107     1      1    0   0   1    0   2600
..   ...  ...  ...   ...    ...  ...  ..  ..  ...    ...
184    1   28   95     1      1    0   0   0    2   2466
185    1   14  100     3      0    0   0   0    2   2495
186    1   23   94     3      1    0   0   0    0   2495
187    1   17  142     2      0    0   1   0    0   2495
188    1   21  130     1      1    0   1   0    3   2495

[189 rows x 10 columns]
```

1.2

From the full DataFrame, extract and print a DataFrame with the birthweight column (and only the birthweight column) for mothers who smoked during pregnancy and had low-birthweight deliveries.

In [96]:
```
partial = (data1["LOW"]==1)&(data1["SMOKE"]==1)
output1 = data1[partial]["class"]
#the output is for low birthweight babies with mother who smoked during pregn
output1
```

Out[96]:
```
130     709
132    1135
139    1790
140    1818
141    1885
144    1928
145    1928
147    1936
152    2084
153    2084
155    2125
156    2126
157    2187
159    2211
160    2225
164    2296
165    2296
168    2353
170    2367
171    2381
172    2381
175    2410
176    2410
177    2414
178    2424
182    2466
183    2466
184    2466
186    2495
188    2495
Name: class, dtype: int64
```

1.3

Print the following statistics for the birthweights in the original full dataset:

- Standard deviation
- 0.16 Quantile
- Mean
- Median
- 0.84 Quantile

Afterwards, print the same statistics for the birthweights in the subset you selected in 1.2.

In [100…

```
# def a function to calculate all the statistics
def statistics(dataframe):
    std = dataframe.std()
    mean = dataframe.mean()
    quantiles = dataframe.quantile([0.16,0.5,0.84])
    print("std: {}, 0.16 quantile: {}, mean: {}, median: {}, 0.84 quantile: {
          format(std, quantiles[0.16], mean, quantiles[0.5], quantiles[0.84])
origin_birthweight = data["class"]
print("The statistics for the birthweights in the original full dataset\n")
statistics(origin_birthweight)
```

```
The statistics for the birthweights in the original full dataset

std: 729.0224168601321, 0.16 quantile: 2226.2, mean: 2944.6560846560847, media
n: 2977.0, 0.84 quantile: 3695.1599999999994
```

In [101…

```
#Statistics for low birthweight babies with mother who smoked during pregnanc
print("The statistics for the birthweights in the original full dataset\n")
statistics(output1)
```

```
The statistics for the birthweights in the original full dataset

std: 399.8074090956996, 0.16 quantile: 1912.52, mean: 2143.0333333333333, medi
an: 2260.5, 0.84 quantile: 2439.12
```

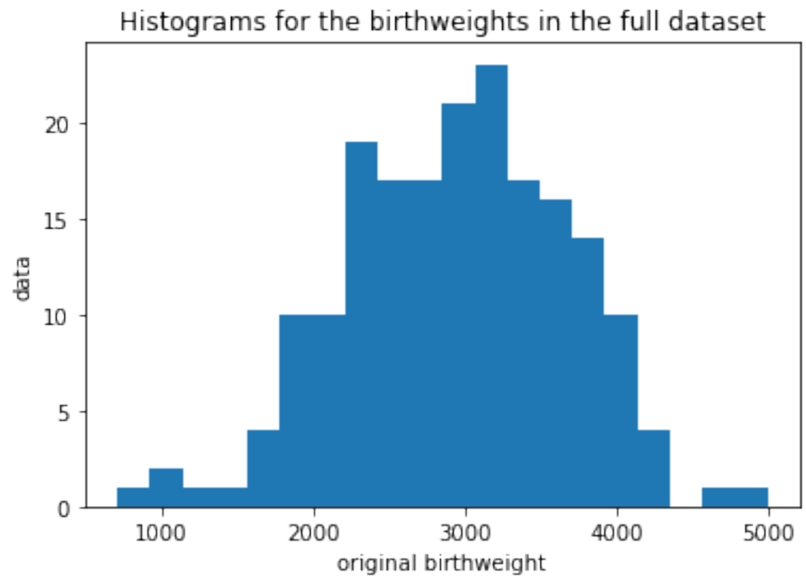
1.4

Plot two density histograms: one for the birthweights in the full dataset, and one for the birthweights in the subset you selected in 1.2. Label the histograms.

In [124…

```
plt.hist(origin_birthweight,bins=20)
plt.title('Histograms for the birthweights in the full dataset')
plt.xlabel("original birthweight")
plt.ylabel("data")
```

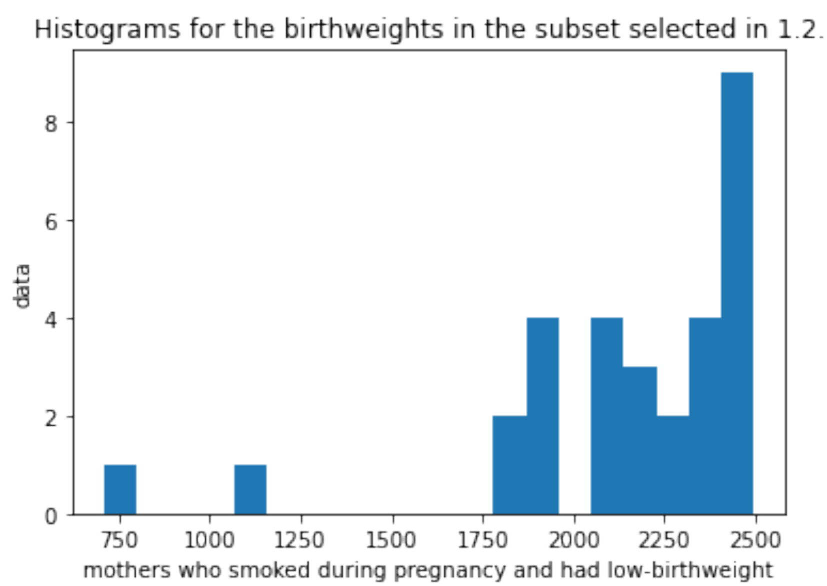
Out[124…: Text(0, 0.5, 'data')



In [123…:

```
plt.hist(output1,bins=20)
plt.title('Histograms for the birthweights in the subset selected in 1.2.')
plt.xlabel("mothers who smoked during pregnancy and had low-birthweight")
plt.ylabel("data")
```

Out[123…: Text(0, 0.5, 'data')



1.5

Is a normal distribution a plausible model for birthweight in either of the two datasets? Back up your answer using the previous results. This image of a normal PDF may be useful:

Normal PDF

_Answer: The normal distribution is plausible model for birthweight with original dataset, since symmetric about the mean around 2944. While for the subset of mothers who smoked during pregnancy and had low-birthweight, we can see from the histogram that it does not follow a normal distribution. The distribution of birthweight is this case is kind of scattered and left skewed.

Problem 2

In this exercise, we will proceed in steps to perform rejection sampling of a beta random variable using a triangular random variable as candidate.

2.1

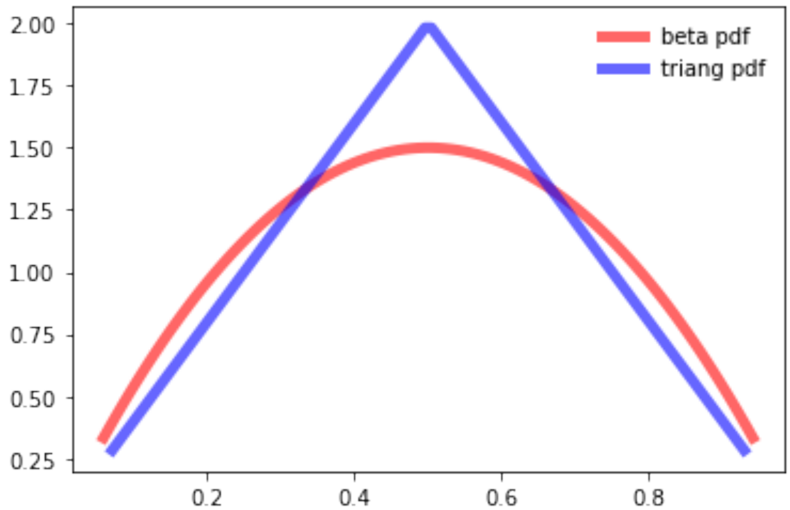
Plot an overlay of a beta PDF and a triangular PDF with the following parameters:

- For the beta PDF, a=2, b=2, loc=0, scale=1.
- For the triangular PDF, c=0.50, loc=0, scale=1.

In [153…

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
from scipy.stats import beta
from scipy.stats import triang
```

In [34]:
```
fig, ax = plt.subplots(1, 1)

#beta PDF
a,b,loc,scale=2,2,0,1
x = np.linspace(beta.ppf(0.01, a, b),beta.ppf(0.99, a, b), 100)
ax.plot(x, beta.pdf(x, a, b),'r-', lw=5, alpha=0.6, label='beta pdf')

#triangular PDF
c,loc,scale=0.5,0,1
x = np.linspace(triang.ppf(0.01, c),triang.ppf(0.99, c), 100)
ax.plot(x, triang.pdf(x, c),'b-', lw=5, alpha=0.6, label='triang pdf')

ax.legend(loc='best', frameon=False)
plt.show()
```



2.2

We will perform 10,000 trials of the rejection sampling procedure. Simulate and store 10,000 random variables distributed as $\text{Uniform}[0, 1]$ using random state 436. Simulate and store 10,000 triangular random variables from the specified triangular distribution using random state 636.

In [126…
```
# 10,000 random variables distributed as  Uniform[0,1]
np.random.seed(436)
uniforms = np.random.uniform(low=0.0, high=1.0, size=10000)

# 10,000 triangular random variables
np.random.seed(636)
triangulars = np.random.triangular(0, 0.5, 1, size = 10000)
```

2.3

Let f be the beta PDF and g the triangular PDF. Using 1.50 as an estimate of $\sup f/g$, generate samples from the beta distribution by rejection sampling. Store your samples. Print the number of samples you obtain.

In [127…

```
# rejection sampling

saved_beta_samples = []
for i in range(10000):
    # sample from the triangular distribution
    sample_tri = triangulars[i]
    # uniform sampling from [0, 1.5 * g(sample_tri)]
    sample_uniform = 1.5 * triang.pdf(sample_tri, c, loc = 0, scale = 1) * un
    if sample_uniform < beta.pdf(sample_tri, a, b, loc = 0, scale = 1):
        saved_beta_samples.append(sample_tri)

print("number of samples saved for beta: {}".format(len(saved_beta_samples)))
```

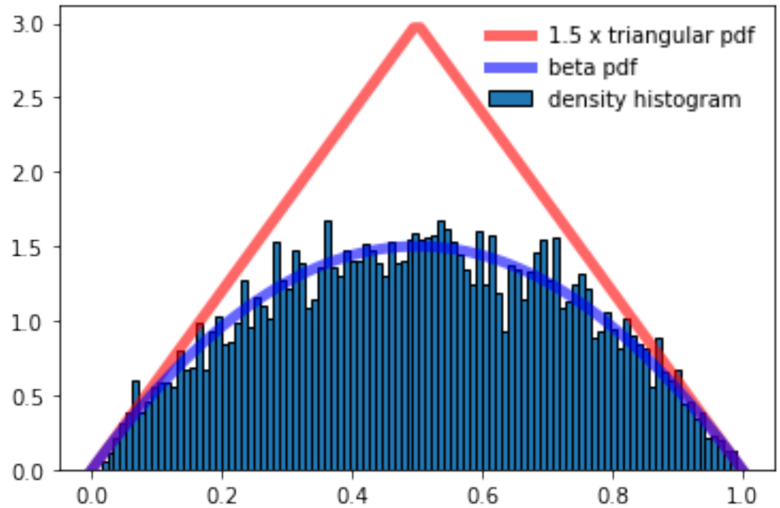
```
number of samples saved for beta: 6674
```

2.4

Plot a density histogram of your samples overlaid with the beta and triangular PDFs. Use 100 bins.

In [151…

```
fig, ax = plt.subplots(1, 1)
# for triangular pdf
c = 0.50
x = np.linspace(0, 1, 100)
ax.plot(x, 1.5 * triang.pdf(x, c, loc = 0, scale = 1), 'r-', lw=5, alpha=0.6,

# for beta pdf
a, b = 2, 2
ax.plot(x, beta.pdf(x, a, b, loc = 0, scale = 1), 'b-', lw=5, alpha=0.6, labe

plt.hist(saved_beta_samples,bins=100, density=True, edgecolor='black', label =

ax.legend(loc='best', frameon=False)
plt.show()
```



Problem 3

3.1

The Epanechnikov kernel is defined by

$$K(u) = \frac{3}{4}(1 - u^2) \qquad \text{for } |u| \leq 1$$

Perform Epanechnikov kernel density estimation on 1,000 simulated samples from a lognormal distribution with s=1 and random state 636. Use a bandwidth of 2.0. Plot the density estimate over the support of the lognormal distribution. (You can use 2 times the maximum of your samples as an upper bound for the support.)

In [154…

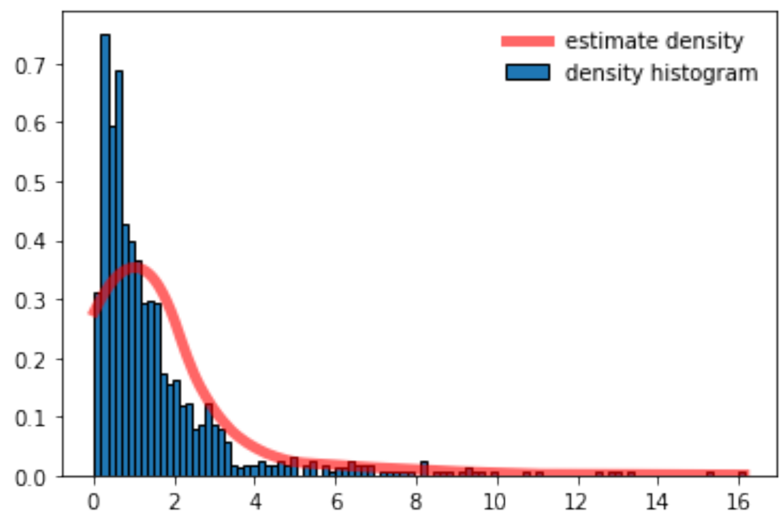
```
# 1,000 simulated samples from a lognormal distribution
np.random.seed(636)
lognormals = np.random.lognormal(mean = 0, sigma = 1, size = 1000)

def Epanechnikov_kernel_density(x0):
    bias = (lognormals - x0) / 2.0
    bias_cliped = np.clip(bias, -1, 1)
    kernel = 0.75 * (1 - bias_cliped ** 2)
    return kernel.sum() / (2 * len(lognormals))
```

In [154…

```
support = np.sort(lognormals)
predict_density = np.zeros(1000)
for idx in range(1000):
    predict_density[idx] = Epanechnikov_kernel_density(support[idx])

# the PDF must be normalized
predict_density /= scipy.integrate.trapz(predict_density, support)
```

In [154…

```
fig, ax = plt.subplots(1, 1)

ax.plot(support, predict_density, 'r-', lw=5, alpha=0.6, label='estimate dens
plt.hist(lognormals,bins=100, density=True, edgecolor='black', label = "densi

ax.legend(loc='best', frameon=False)
plt.show()
```



3.2

Test whether the estimate integrates to unity over the support of the lognormal distribution.

In [155…

```
scipy.integrate.trapz(predict_density, support)
```

Out[155… 1.0

3.3

Explain the results of your integration.

_Answer:_As the density was estimated on finite sampled data, the integration will be smaller than 1. Thus we have to normalize it by scipy.integrate.trapz

Problem 4

4.1

Below we load the Boston house prices dataset. We also store the labels of the predictor variables for you.

Our goal will be to predict house price (MEDV) by regression. Split the dataset into a training and test set using 1/3 as the test size and a random state of 553. Use the function _train_test*split* from _sklearn.model*selection* for this purpose.

In [157…

```
# Loading data:

# Import function for loading the 'boston' dataset.
from sklearn.datasets import load_boston
# Load a 'bunch' containing data and descriptions.
boston_bunch = load_boston()
# Extract and store predictor variables.
X = boston_bunch.data
# Extract and store the variable that is the target for prediction.
y = boston_bunch.target
# Extract and store labels of predictor variables.
labels = boston_bunch.feature_names
```

In [159…

```
# Your code:
labels
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1 / 3, ra
```

4.2

Fit three different linear models on the training set by ordinary least squares (OLS):

- A model using all predictor variables.
- A model using only AGE, NOX, DIS, and RAD as predictor variables.
- A model using all polynomial combinations of degree ≤ 2 of the original thirteen predictor variables.

In [160…
```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

In [161…
```
model_all_variables = linear_model.LinearRegression()
model_all_variables.fit(X_train, y_train)
```

Out[161…
```
LinearRegression()
```

In [162…
```
# get  AGE, NOX, DIS, and RAD as sub variables.
X_train_sub = X_train[:, [6, 4, 7, 8]]
model_four_variables = linear_model.LinearRegression()
model_four_variables.fit(X_train_sub, y_train)
```

Out[162…
```
LinearRegression()
```

In [163…
```
from sklearn.preprocessing import PolynomialFeatures
# augment the features with PolynomialFeatures
transformed_data = PolynomialFeatures(2).fit_transform(X_train)
```

In [164…
```
model_poly = linear_model.LinearRegression()
model_poly.fit(transformed_data, y_train)
```

Out[164…
```
LinearRegression()
```

4.3

For model assessment, print the following for each of the three models:

- The R^2 of the predictions on the training set.
- The R^2 of the predictions on the test set.
- Predicted MEDV for the first five sample points in the test set.
- True MEDV for the first five sample points in the test set.

In [165…
```
print("The R_square of the predictions on the training set for each of the th
print("model_all_variables: {}, model_four_variables: {}, model_poly: {}".for
      (r2_score(model_all_variables.predict(X_train), y_train),
     r2_score(model_four_variables.predict(X_train_sub), y_train),
       r2_score(model_poly.predict(transformed_data), y_train)))
```

```
The R_square of the predictions on the training set for each of the three mode
ls

model_all_variables: 0.6957079798569403, model_four_variables: -1.552296221579
7982, model_poly: 0.9050534733715457
```

In [166…

```
X_test_sub = X_test[:, [6, 4, 7, 8]]
transformed_test_data = PolynomialFeatures(2).fit_transform(X_test)
print("The R_square of the predictions on the test set for each of the three m
print("model_all_variables: {}, model_four_variables: {}, model_poly: {}".form
      (r2_score(model_all_variables.predict(X_test), y_test),
     r2_score(model_four_variables.predict(X_test_sub), y_test),
       r2_score(model_poly.predict(transformed_test_data), y_test)))
```

```
The R_square of the predictions on the test set for each of the three models

model_all_variables: 0.564687499531662, model_four_variables: -2.0486276859662
476, model_poly: 0.7530719079627546
```

In [167…

```
print("The Predicted MEDV for the first five sample points in the test set.\n
print("model_all_variables: {}\nmodel_four_variables: {}\nmodel_poly: {}".form
      (model_all_variables.predict(X_test)[:5],
       model_four_variables.predict(X_test_sub)[:5],
       model_poly.predict(transformed_test_data)[:5]))
```

```
The Predicted MEDV for the first five sample points in the test set.

model_all_variables: [24.2652595  12.11746393 27.67012303 24.11419114 21.83525
384]
model_four_variables: [27.88749751 15.41806533 25.36918791 24.09219392 26.3778
3049]
model_poly: [25.04007101  9.51745439 31.16530704 17.5032692  22.02355266]
```

In [168…

```
print("True MEDV for the first five sample points in the test set: {}".format
```

```
True MEDV for the first five sample points in the test set: [24.6  5.6 27.1 21
.9 20. ]
```

4.4

Comment on your results in 4.3, which model do you think is the best? Explain you answer.

_Answer:_As the relationship between the target and predictor variables is not linear, poly regression performs best. Besides, using only a few features for linear regression is not enough.

4.5

Consider the linear regression model using all original features you built above. Holding all other variables equal, what effect does the model predict that an increase in 0.1 parts per 10 million nitric oxide concentration in a place will have on the median value of of owner-occupied homes in that place? Write code that will return and print the answer.

In [169…

```
model_all_variables.coef_
```

Out[169…

```
array([-1.10039735e-01,  4.87028995e-02,  6.02131092e-02,  2.40773435e+00,
       -1.64963662e+01,  4.06242179e+00,  1.51806513e-02, -1.24182217e+00,
        3.73125343e-01, -1.53000833e-02, -9.39645782e-01,  1.01952518e-02,
       -5.42301337e-01])
```

In [170…

```
print("It will increase by value {}".format(model_all_variables.coef_[4] * 0.
```

```
It will increase by value -1.6496366216182925
```

In [ ]:

```
```