DS-UA 111 HW4

August 17, 2023

1 Question 1

a) False

b) True

c) True

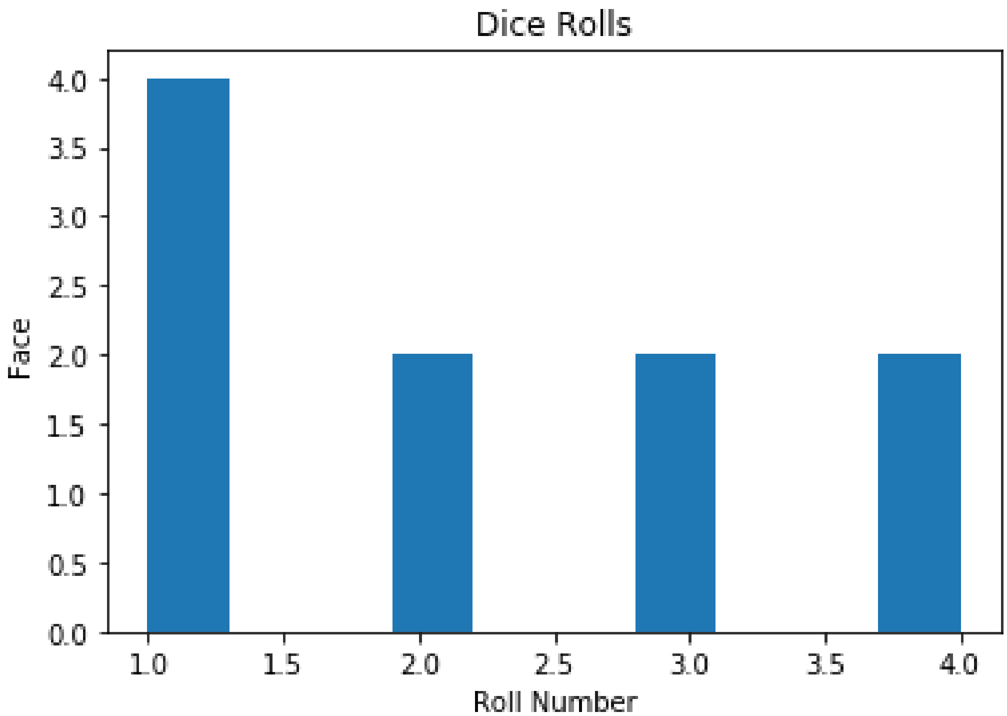
d) True

e) False

2 Question 2

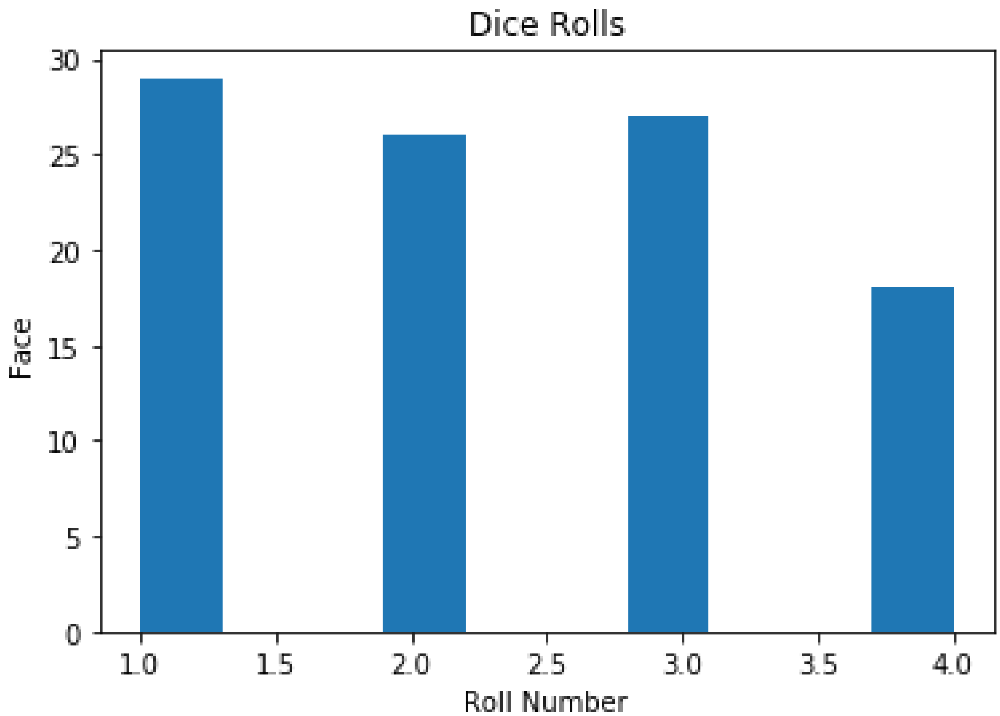
[1]:
```
# part a
import numpy as np
import matplotlib.pyplot as plt

faces = np.random.choice(range(1,5), 10)
faces
plt.hist(faces)
plt.title("Dice Rolls")
plt.xlabel("Roll Number")
plt.ylabel("Face")
plt.show()
```

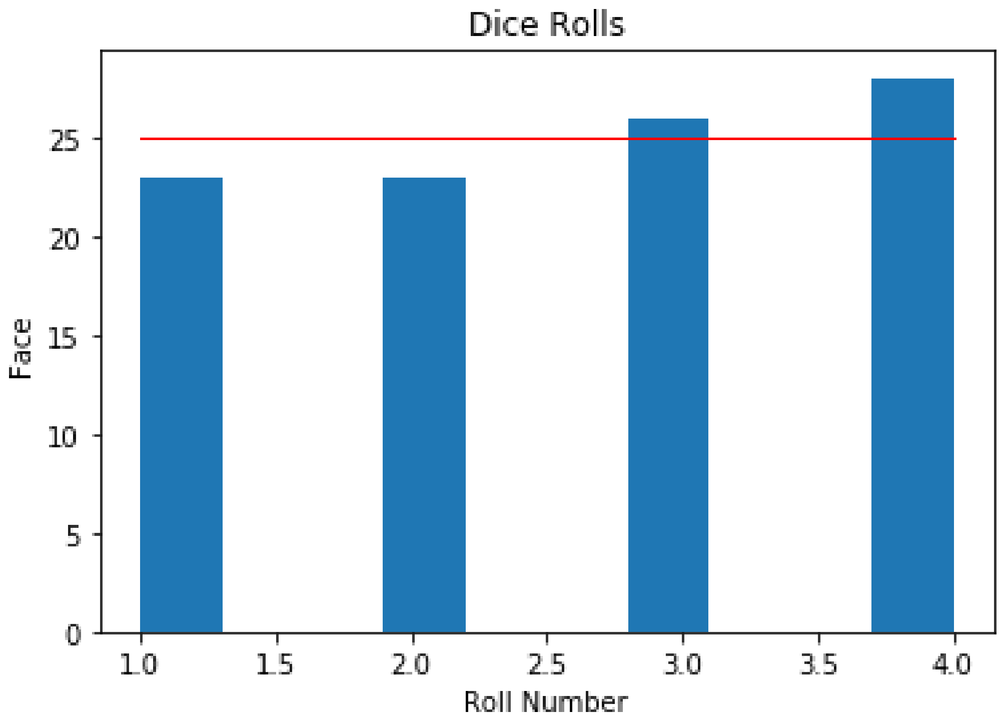


b) 0.2 ; 0.25

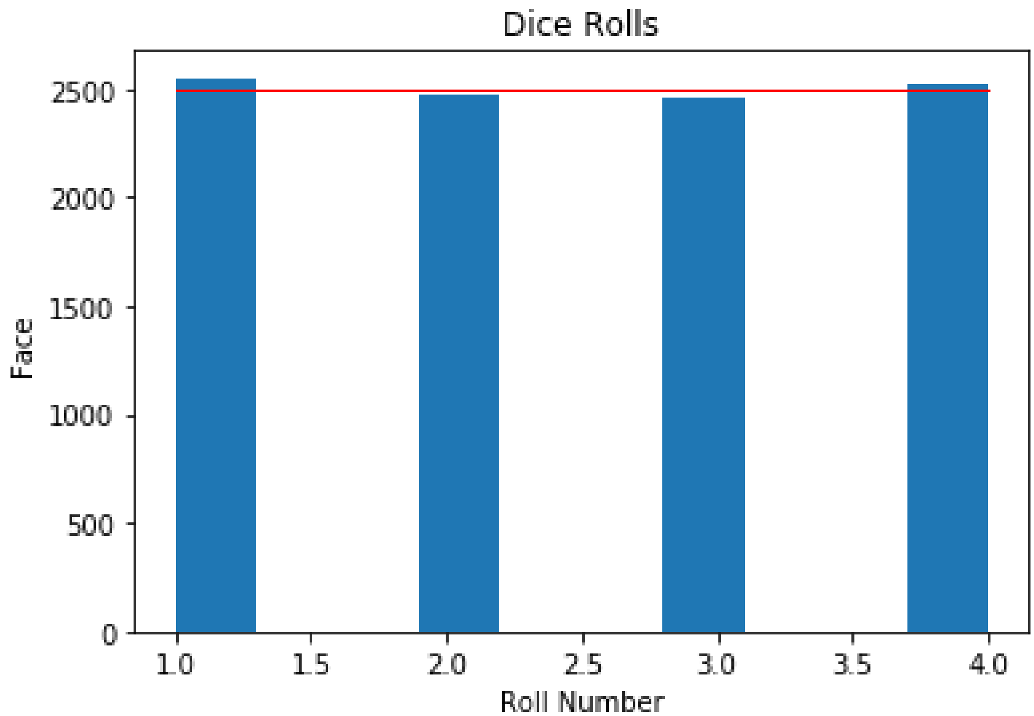
[2]:
```
# part c
faces = np.random.choice(range(1,5), 100)
faces
plt.hist(faces)
plt.title("Dice Rolls")
plt.xlabel("Roll Number")
plt.ylabel("Face")
plt.show()
```



[3]:
```
# part d
faces = np.random.choice(range(1,5), 100)
faces
plt.hist(faces)
plt.title("Dice Rolls")
plt.xlabel("Roll Number")
plt.ylabel("Face")
x_coord = [1,4]
y_coord = [25,25]
plt.plot(x_coord, y_coord, color="red", linewidth=1)
plt.show()
```



[4]:
```
# part e
faces = np.random.choice(range(1,5), 10000)
faces
plt.hist(faces)
plt.title("Dice Rolls")
plt.xlabel("Roll Number")
plt.ylabel("Face")
x_coord = [1,4]
y_coord = [2500,2500]
plt.plot(x_coord, y_coord, color="red", linewidth=1)
plt.show()
```



f) This happens because of the law of large numbers. The law of large numbers states that the higher the sample size of an experiment, the closer our empirical distribution matches the theoretical distribution.

3 Question 3

[5]:
```
# part a
import pandas as pd
population = pd.read_csv("population_data.csv", squeeze = True)
pd.DataFrame(population).describe()
```

[5]:
```
                   0
count  100000.000000
mean        8.330766
std         6.291950
min         0.000415
25%         3.314054
50%         7.047814
75%        11.993263
max        46.983332
```

[6]:
```
# part b
population_sample = pd.DataFrame(population).sample(n=25)
```

```
population_sample.describe()
```

[6]:

```
               0
count  25.000000
mean    8.542476
std     4.696871
min     0.424520
25%     4.647740
50%     8.547478
75%    11.374711
max    17.016525
```

[7]:

```
# part c
population_sample = pd.DataFrame(population.sample(n=1000))
population_sample.describe()
```

[7]:

```
                 0
count  1000.000000
mean      8.354364
std       6.477645
min       0.003483
25%       3.345058
50%       6.906624
75%      11.602347
max      36.584645
```

d) Yes; this is because of the law of large number.

[8]:

```
# part e
def percentile(input_data):
    y1 = np.power(input_data,0.25)
    y2 = np.sqrt(y1) * -1
    y3 = np.exp(y2)
    y4 = np.sin(y3)
    return np.percentile(y4, 80)
```

[9]:

```
# part f
percentile(population)
```

[9]: 0.3177257855396434

[10]:

```
# part g
population_sample_100 = pd.DataFrame(population.sample(n=100))
percentile(population_sample_100)
```

[10]: 0.3229746322037541

[11]:
```
# part h
population_sample_1000 = pd.DataFrame(population.sample(n=1000))
percentile(population_sample_1000)
```

[11]: 0.31977613407151495

The complex statistic on a randomly chosen sample of size 1000 from the population is closer; LLN seems to work well when the statistics are complex.

4 Question 4

[12]:
```
# part a
from scipy.stats import norm
housing = pd.read_csv('house.csv')
sale_price = housing['SalePrice']
pd.DataFrame(sale_price.describe())
```

[12]:

	SalePrice
count	2930.000000
mean	180796.060068
std	79886.692357
min	12789.000000
25%	129500.000000
50%	160000.000000
75%	213500.000000
max	755000.000000

[13]:
```
# part b
p = 0.01
crit_val_two_tail = norm(0,1).ppf(1 - p/2)
crit_val_one_tail = norm(0,1).ppf(1 - p)
print(crit_val_two_tail)
print(crit_val_one_tail)
```

```
2.5758293035489004
2.3263478740408408
```

[14]:
```
# part c
h0 = 150000
z = (np.mean(sale_price) - h0)/ (np.std(sale_price) / np.sqrt(len(sale_price)))
print(abs(crit_val_two_tail) < abs(z))
```

```
True
```

I reject the null hypothesis.

[15]:
```python
# part d
print(crit_val_one_tail < z)
```

```
True
```

I reject the null hypothesis.

[16]:
```python
# part e
h0 = 200000
z = (np.mean(sale_price) - h0)/ (np.std(sale_price) / np.sqrt(len(sale_price)))
p_value = 1 - norm(0,1).cdf(z)
print(p_value)
```

```
1.0
```

I don't reject the null hypothesis.

[17]:
```python
# part f
p_value = norm(0,1).cdf(z)
print(p_value)
```

```
5.067385832657905e-39
```

I reject the null hypothesis.

[18]:
```python
# part g
p = 0.01
alpha = abs(norm(0,1).ppf(p/2))
a = np.mean(sale_price) - alpha * np.std(sale_price)/np.sqrt(len(sale_price))
b = np.mean(sale_price) + alpha * np.std(sale_price)/np.sqrt(len(sale_price))
a,b
```

[18]:
```
(176995.18508695194, 184596.93504956685)
```

5 Question 5

[19]:
```python
# part a
import statsmodels.formula.api as smf
world = pd.read_csv("World.csv")
X = world[['gdp_10_thou', 'pop_total', 'spendhealth', 'fertility']]
y = world['lifeex_total']
life_expectancy_regression = smf.ols('lifeex_total ~ gdp_10_thou + pop_total +␣
↪spendhealth + fertility',
                                                                                ␣
↪     data = world).fit()
life_expectancy_regression.summary()
```

[19]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:            lifeex_total   R-squared:                       0.712
Model:                             OLS   Adj. R-squared:                  0.704
Method:                  Least Squares   F-statistic:                     88.53
Date:                 Thu, 17 Aug 2023   Prob (F-statistic):           1.06e-37
Time:                         09:50:01   Log-Likelihood:                -456.37
No. Observations:                  148   AIC:                             922.7
Df Residuals:                      143   BIC:                             937.7
Df Model:                            4
Covariance Type:             nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept      80.5665      1.666     48.366      0.000      77.274      83.859
gdp_10_thou     2.7797      0.598      4.652      0.000       1.599       3.961
pop_total      -0.0005      0.003     -0.171      0.864      -0.006       0.005
spendhealth     0.0375      0.310      0.121      0.904      -0.575       0.650
fertility      -4.7158      0.346    -13.616      0.000      -5.400      -4.031
==============================================================================
Omnibus:                        38.137   Durbin-Watson:                   2.158
Prob(Omnibus):                   0.000   Jarque-Bera (JB):               76.368
Skew:                           -1.144   Prob(JB):                     2.61e-17
Kurtosis:                        5.674   Cond. No.                         613.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[20]:

```python
# part b
health_spending_increase_coeff = 1000000
lifeex_increase = health_spending_increase_coeff * 0.0375
print(lifeex_increase)
```

```
37500.0
```

c) 71.2%

d) 80.5665

e) Yes, we can reject the null hypothesis. The p-value for GDP is 0.000, which is less than our p-value threshold of 0.05.

[21]:

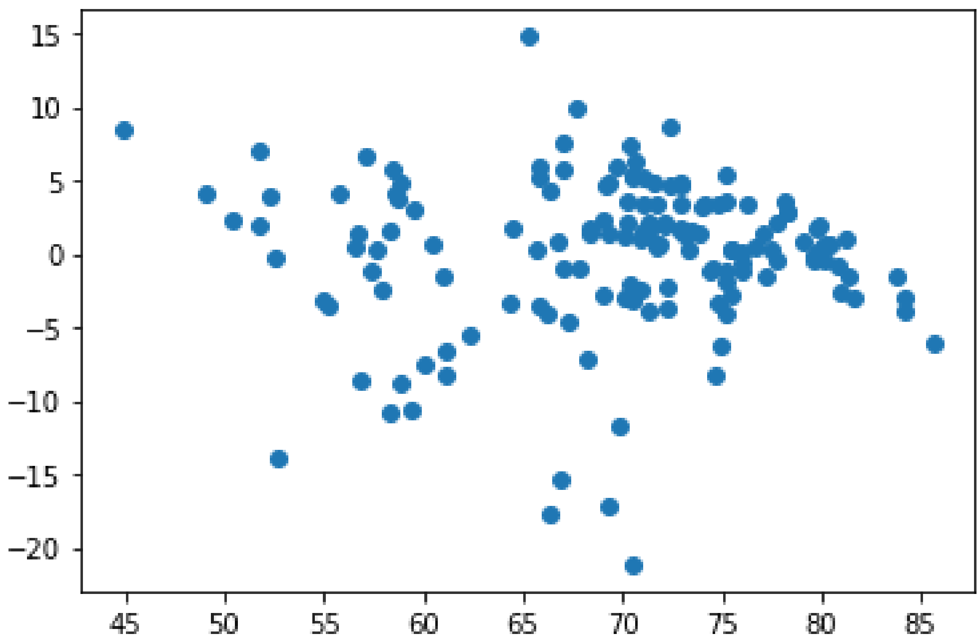
```python
# part f
p = 0.05
```

```
alpha = abs(norm(0,1).ppf(p/2))
a = np.mean(world['fertility']) - alpha * np.std(world['fertility'])/np.
↪sqrt(len(world['fertility']))
b = np.mean(world['fertility']) + alpha * np.std(world['fertility'])/np.
↪sqrt(len(world['fertility']))
print(a,"and",b)
```

```
2.5367014919366944 and 2.9772009470876957
```

g) Yes, because the model assumes a linear relationship.

[22]:
```
# part h
from sklearn import linear_model
pred_y = life_expectancy_regression.predict(X)
residual = world['lifeex_total'] - pred_y
plt.scatter(pred_y, residual)
plt.show()
```



There doesn't seem to be a clear trend.

[23]:
```
# part i
df = {"prediction": pred_y, "residual": residual}
pred_residual_corr = pd.DataFrame(df)
print(pred_residual_corr.corr())
```

```
              prediction      residual
prediction  1.000000e+00 -1.590993e-15
```

```
residual   -1.590993e-15  1.000000e+00
```

It suggests the presence of potential problems with the models doesn't depend on the prediction of life expectancy.

```
[24]: # part j
df = {"gdp_10_thou": world['gdp_10_thou'], "pop_total": world['pop_total'],␣
 ↪"spendhealth": world['spendhealth'],
                                                                              ␣
 ↪  "fertility": world['fertility']}
pred_residual_corr = pd.DataFrame(df)
print(pred_residual_corr.corr())
```

```
             gdp_10_thou  pop_total  spendhealth  fertility
gdp_10_thou     1.000000  -0.036051     0.661569  -0.392485
pop_total      -0.036051   1.000000    -0.125908  -0.062368
spendhealth     0.661569  -0.125908     1.000000  -0.404251
fertility      -0.392485  -0.062368    -0.404251   1.000000
```

k) Health spending and the GDP seem to be highly correlated. This could be a problem in interpreting the model in (b), because it means GDP could be a confounding variable between health spending and life expectancy. Therefore, we can't determine whether health spending has a causal effect on life expectancy.

6 Citations

1) https://numpy.org/doc/stable/reference/routines.math.html; used this source to find various numpy functions to help work on Q3
2) https://numpy.org/doc/stable/reference/generated/numpy.percentile.html#numpy.percentile; used this source to find the percentile method in numpy
3) https://www.geeksforgeeks.org/python-pandas-dataframe-corr/; used this source to learn how to use the corr() method in pandas for Q5.