SMP exam 2019/2020

In [ ]:

```
# Student number: 237294
```

In [131]:

```
# Import relevant libraries here
from scipy.stats import binom
from scipy.stats import geom
from scipy.stats import hypergeom
from scipy.stats import poisson
from scipy.stats import nbinom
from scipy.stats import expon
from scipy.stats import chi2_contingency
from scipy.stats import ttest_rel
from scipy.stats import norm
from scipy.stats import normaltest
from scipy.stats import ttest_ind
from scipy.stats import uniform
from scipy.stats import probplot
from scipy.stats import t
import statsmodels.api as sm
import pandas as pd
import sympy as sp
from sympy import *
sp.init_printing()
import matplotlib.pyplot as plt
import scipy.special as special
from scipy.integrate import quad
import numpy as np
from scipy.misc import derivative
import math
from math import *
from fractions import Fraction
import fractions
from IPython.display import display, Math, Latex
%matplotlib inline
from sympy.interactive import printing
printing.init_printing()
```

Feel free to add cells if you need to. The easiest way to convert to pdf is to save this notebook as .html (File-->Download as-->HTML) and then convert this html file to pdf.

Assignment 1 (15%)

In [20]:

```
# a)
x = sp.Symbol('x')
c = sp.Symbol('c')
f = 2 * x + 1
f
```

Out[20]:

$$2x + 1$$

In [211]:

```
F = integrate(f, x)
display(Latex("We see that the integration of f(x) is F(x) plus some constant, c:"))
display(F + c)
display(Latex("The C constant is the normal integration constant, let's find its valu
e:"))
```

We see that the integration of f(x) is F(x) plus some constant, c:

$$c + x^2 + x$$

The C constant is the normal integration constant, let's find its value:

In [35]:

```
F = F + c
# finding the Area by F(xEnd) - F(xStart), which needs to be equal to 1,
# then we can find the constant value, C
solve(F.subs({x: 1/2}) - F.subs({x: -1/2}) - 1, c)
display(Math("C = 0"))
display(Math("=>"))
display(Math("\\frac{d f(x)}{dx} = %s" % latex(F.subs({c: 0}))))
```

$$C = 0$$

$$=>$$

$$\frac{df(x)}{dx} = x^2 + x$$

In [36]:

```
# b)
prob = integrate(f, (x, -1/2, 0))
prob2 = 1 - integrate(f, (x, -1/2, 1/4))
display(Math("P(X < 0) = %s" % round(prob, 4)))
display(Math("P(X > \\frac{1}{4}) = %s" % round(prob2, 4)))
```

$$P(X < 0) = 0.25$$

$$P(X > \frac{1}{4}) = 0.4375$$

In [37]:

```
# c)
E = integrate(x * f, (x, -1/2, 1/2))
display(Math("E[X] = %s" % round(E, 4)))
V = integrate(x**2 * f, (x, -1/2, 1/2)) - E ** 2
display(Math("Var(X) = %s" % round(V, 4)))
```

$E[X] = 0.1667$

$Var(X) = 0.0556$

Assignment 2 (20%)

In [47]:

```
# a)
# Let B denote the event that a hardware breaks.
rate = 3
prob = poisson.pmf(0, rate)
display(Math("P(B = 0) = %s" % round(prob, 4)))
```

$P(B = 0) = 0.0498$

In [48]:

```
# b)
prob = poisson.sf(10, 3) # SF is exclusive, and since we want greater than 10 events, w
e use 10
display(Math("P(B > 10) = %s" % round(prob, 4)))
```

$P(B > 10) = 0.0003$

In [49]:

```
# c)
# Since avg 3 breakdowns/day, its 24hr / 3
display(Math("Avg(K) = %s hours" % (24 / 3)))
```

$Avg(K) = 8.0 hours$

In [50]:

```
# d)
# The exponential distribution models the time between events, K
```

In [51]:

```
# e)
# half a day = 12hrs
rate = 1.5 # 1½ breakdowns every 12 hours
prob = poisson.pmf(0, rate)

# note: this P(B = 0) should not be confused with the one from a).
# This is for ½day, the other was for 1 day.
display(Math("P(B = 0) = %s" % round(prob, 4)))
```

$P(B = 0) = 0.2231$

Assignment 3 (25%)

In [59]:

```
# a)
# Let L denote the event that a person has lactose intolerance
# Let M denote the event that a person is male.
PLM = 20/120
display(Math("P(L|M) = %s" % round(PLM, 4)))
```

$$P(L|M) = 0.1667$$

In [199]:

```
# b)
PL = (20 + 55) / (120 + 215)
display(Math("P(L) = %s" % round(PL, 4)))
```

$$P(L) = 0.2239$$

In [61]:

```
# c)
display(Math("P(M|L) = \\frac{P(L|M) * P(M)}{P(L)}"))
PM = 120/(120 + 215)
display(Math("P(M) = \\frac{120}{120 + 215} = %s" % round(PM, 4)))
display(Math("P(M|L) = %s" % round(PLM * PM / PL, 4)))
```

$$P(M|L) = \frac{P(L|M) * P(M)}{P(L)}$$

$$P(M) = \frac{120}{120 + 215} = 0.3582$$

$$P(M|L) = 0.2667$$

In [82]:

```
# d)
df = pd.DataFrame({
    "Male": [20, 100, 120],
    "Female": [55, (215-55), 215],
    "Sum": [20+55, 100 + (215-55), 120 + 215]
})
df = df.rename({0: "Lactose intolerance", 1: "No intolerance", 2: "Sum"})
df.head()
```

Out[82]:

	Male	Female	Sum
Lactose intolerance	20	55	75
No intolerance	100	160	260
Sum	120	215	335

In [201]:

```
# e)
alpha = 0.05
data = pd.DataFrame({
    "Male": [20, 100],
    "Female": [55, (215-55)],
}).rename({0: "Lactose intolerance", 1: "No intolerance"})
stat, pvalue, dof, ex = chi2_contingency(data)

if pvalue < alpha:
    print("Reject since p-value = " + repr(round(pvalue, 4)) + ' < ' + repr(alpha))
else:
    print("Fail to reject since p-value = " + repr(round(pvalue, 4)) + ' > ' + repr(alp
ha))
display(data)
print('Expected values:')
print('________________')
expected = pd.DataFrame(ex.astype(int), columns=data.columns)
expected = expected.rename({0: "Lactose intolerance", 1: "No intolerance"})
display(expected)
```

```
Fail to reject since p-value = 0.0818 > 0.05
```

	Male	Female
Lactose intolerance	20	55
No intolerance	100	160

```
Expected values:
________________
```

	Male	Female
Lactose intolerance	26	48
No intolerance	93	166

Assignment 4 (20%)

In [96]:

```
# a)
Volume = pd.DataFrame({ "Beer Vol.":[0.44, 0.43, 0.39, 0.49, 0.49, 0.63, 0.53, 0.47, 0.
56, 0.42, 0.53, 0.54,
0.55, 0.53, 0.47, 0.50, 0.41, 0.49, 0.53, 0.38, 0.60, 0.50, 0.44, 0.45, 0.40, 0.58,
0.41, 0.48, 0.51, 0.47, 0.4, 0.45, 0.54, 0.51, 0.47, 0.35, 0.40, 0.49, 0.47, 0.48,
0.57, 0.44, 0.52, 0.39, 0.43] })
E = Volume.mean()[0]
display(Math("Avg(Volume) = %s" % round(E, 4)))
```

$$Avg(Volume) = 0.4784$$

In [99]:

```
# b)
std = Volume.std()[0]
display(Math("Std(Volume) = %s" % round(std, 4)))
```

$$Std(Volume) = 0.0632$$

In [113]:

```
# c)
# One-tailed test
# H0 = Barman pours the fair amount of beer in each drink
alpha = 0.05
expected_vol = 0.5
n = Volume.count()[0]
t_val = (E - expected_vol) / (std / sqrt(n))
prob = t.cdf(t_val, df=n-1)
p_val = min(prob,1-prob)
print("p-value = %s" % round(p_val, 5))
print("Reject since ", round(p_val, 5), ' < ', alpha)
```

```
p-value = 0.01351
Reject since  0.01351  <  0.05
```

In [202]:

```
# d)
# H0: There is no difference in volume poured at each bar.
NBV = pd.DataFrame({ "Beer Vol.":[0.50, 0.42, 0.36, 0.52, 0.45, 0.50, 0.50, 0.51, 0.44, 0.37,
0.45, 0.44, 0.49, 0.43, 0.48, 0.50, 0.33, 0.40, 0.44, 0.42]})

val = ttest_rel(Volume["Beer Vol."][:20], NBV["Beer Vol."])
alpha = 0.05

if val[0] < 0:
    pvalue = 1-val[1]/2
else:
    pvalue = val[1]*0.5

if pvalue < alpha:
    print("Reject since ", round(pvalue, 4), ' < ', alpha)
else:
    print("Fail to reject since ", round(pvalue, 4) , '\u2265' , alpha)
```

```
Reject since  0.0023  <  0.05
```

Assignment 5 (20%)

In [127]:

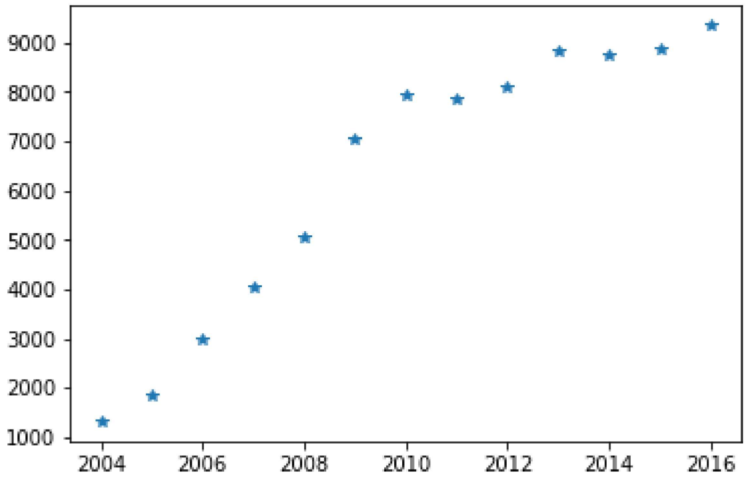
```
df = pd.DataFrame({
    "Year": [2004,2005,2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016],
    "Number": [1319,1861,2980,4048,5065,7048,7935,7844,8081,8845,8731,8882,9350]
})
X = df["Year"]
Y = df["Number"]
df.head()
```

Out[127]:

	Year	Number
0	2004	1319
1	2005	1861
2	2006	2980
3	2007	4048
4	2008	5065

In [128]:

```
plt.plot(X, Y, '*')
plt.show()
```



In [172]:

```
X1 = sm.add_constant(X)
model = sm.OLS(Y, X1).fit()
res = model.resid
yhat = model.fittedvalues
display(model.summary())

#Slope
beta = model.params["Year"]
print('beta = ', beta)

#Intercept:
alfa = model.params["const"]
print('alfa = ', alfa)
```

```
C:\Users\Amavin\Anaconda3\lib\site-packages\scipy\stats\stats.py:1394: Use
rWarning: kurtosistest only valid for n>=20 ... continuing anyway, n=13
  "anyway, n=%i" % int(n))
```

OLS Regression Results

Dep. Variable:	Number	**R-squared:**	0.900
Model:	OLS	**Adj. R-squared:**	0.891
Method:	Least Squares	**F-statistic:**	98.88
Date:	Fri, 10 Jan 2020	**Prob (F-statistic):**	7.82e-07
Time:	11:18:41	**Log-Likelihood:**	-106.50
No. Observations:	13	**AIC:**	217.0
Df Residuals:	11	**BIC:**	218.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.402e+06	1.42e+05	-9.899	0.000	-1.71e+06	-1.09e+06
Year	700.6264	70.458	9.944	0.000	545.550	855.703

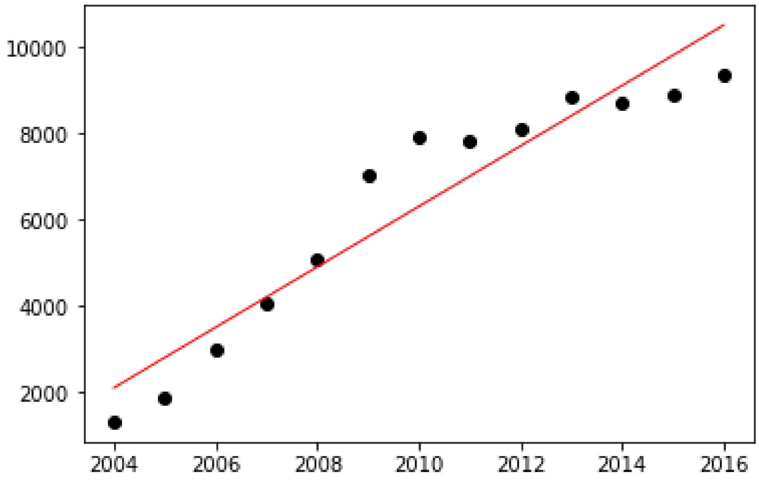
Omnibus:	1.198	**Durbin-Watson:**	0.400
Prob(Omnibus):	0.549	**Jarque-Bera (JB):**	0.965
Skew:	0.478	**Prob(JB):**	0.617
Kurtosis:	2.068	**Cond. No.**	1.08e+06

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.08e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

```
beta =  700.6263736264116
alfa =  -1401952.1648352416
```

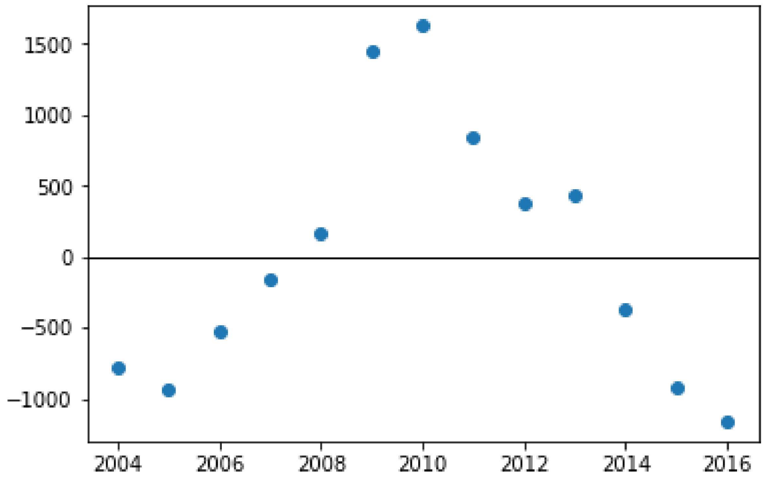
In [171]:

```
plt.scatter(X, Y,  color='black')
plt.plot(X, yhat, color='red', linewidth=1);
```



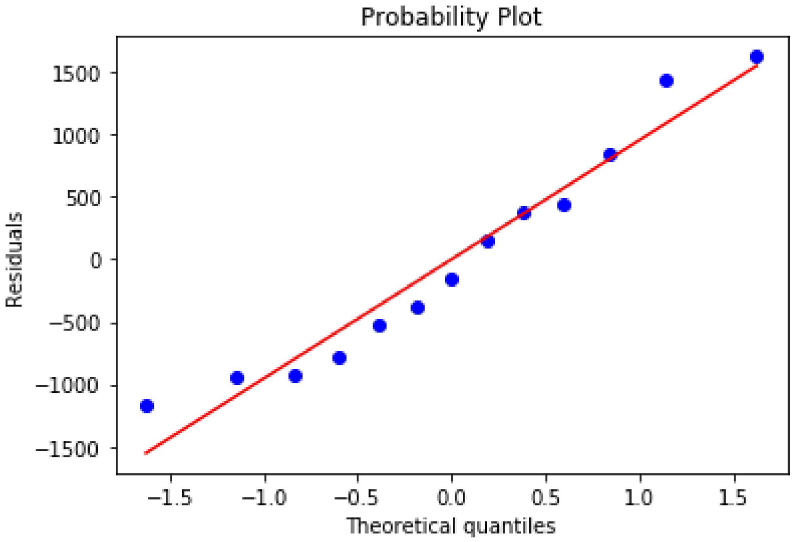
In [142]:

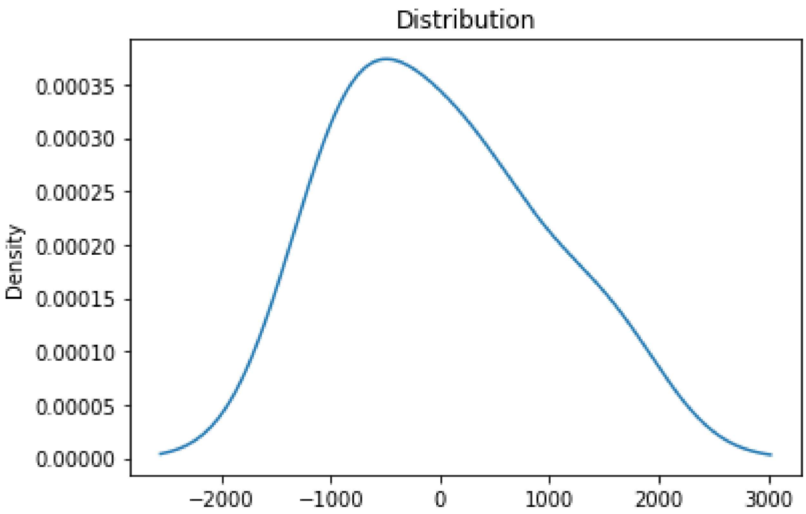
```
# Plotting the errors:
plt.scatter(X, res)
plt.axhline(y=0, color='k', linewidth=1)
plt.show()
```



In [150]:

```
probplot(res, plot=plt)
plt.ylabel('Residuals')
plt.show()
df = pd.DataFrame(res)
fig, ax = plt.subplots()
df.plot.kde(ax=ax, legend=False, title='Distribution')
plt.show();
display(Latex("The error seems to be normally distributed"))
```





The error seems to be normally distributed

In [188]:

```
#5.Test the adequacy of the model
#5.1.Could the Slope ß = 0?  Calculate the P-value for that

#We make a hypothesis to check that:
# H0 : ß = 0
# H1 : ß != 0

#We calculate the p-value:
n = len(X)
SSE = sum(res**2) #sum of square error
s2  = SSE/(n-2) # the Estimated variance of the error
Sxx = sum((X-X.mean())**2)

#p-value:
p_val = 2*t.cdf(0, df=n-2, loc=beta, scale=np.sqrt(s2/Sxx))

print("p_val = %s" % round(p_val, 5))

display(Latex("The p-value is lower than 0.05, so we reject the null hypothesis"))
```

```
p_val = 0.0
```

The p-value is lower than 0.05, so we reject the null hypothesis

In [184]:

```
# Is the model a good fit?
display(Math("R^2 = %s" % round(model.rsquared,4)))
display(Latex("The $R^2$ value shows a high correlation"))
```

$$R^2 = 0.8999$$

The R^2 value shows a high correlation

In [192]:

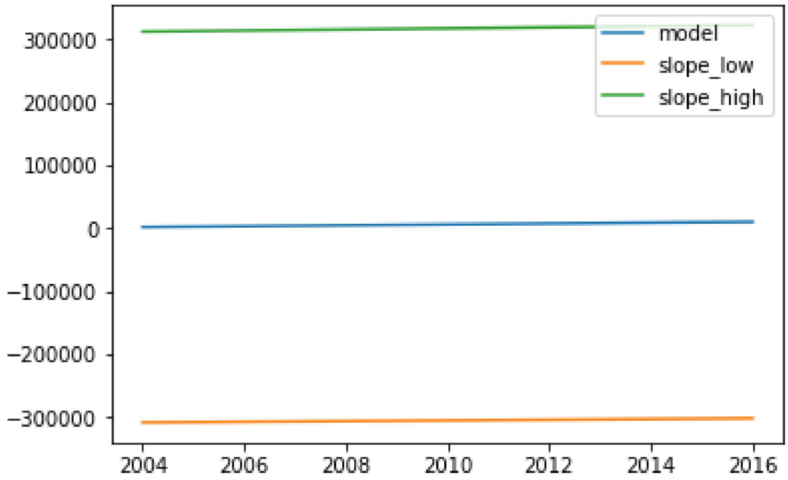
```
#6.Calculating the confidence interval for the slope and the intersection:
# (Which is also given in the OLS summary, but I couldn't figure out how to extract it)

s  = np.sqrt(s2)
t0 = t.ppf(0.975,n-2)
beta_low = beta - t0 * s / np.sqrt(Sxx)
beta_high = beta + t0 * s / np.sqrt(Sxx)
print('beta_low = ', beta_low, '\nbeta_high = ', beta_high)
alfa_low  = alfa - t0 * s * np.sqrt(1/n + X.mean()**2/Sxx)
alfa_high = alfa + t0 * s * np.sqrt(1/n + X.mean()**2/Sxx)
print('alfa_low = ', alfa_low, '\nalfa_high = ', alfa_high)
```

```
beta_low =  545.5495320901293
beta_high =  855.7032151626938
alfa_low =  -1713657.1563913033
alfa_high =  -1090247.17327918
```

In [207]:

```
# Plotting the different lines:
plt.plot(X, alfa+beta*X, label = 'model')
plt.plot(X, alfa+beta_low*X, label = 'slope_low')
plt.plot(X, alfa+beta_high*X, label = 'slope_high')
plt.legend(loc=1)
plt.show()
```



In [ ]: