In [ ]:
```python
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

import scipy.stats as st
import statsmodels.formula.api as smf
import statsmodels.api as sm
```

1.

(a)

In [ ]:
```python
indicators = pd.read_table("/Users/user/Desktop/Yonsei/Junior/3-2/Introduction to Data Analysis and Regression/Homework/indicators2.txt", sep='\t')
```

In [ ]:
```python
indicators2 = indicators[['PriceChange','LoanPaymentsOverdue']]
indicators2
```

Out[ ]:

	PriceChange	LoanPaymentsOverdue
Atlanta	1.2	4.55
Boston	-3.4	3.31
Chicago	-0.9	2.99
Dallas	0.8	4.26
Denver	-0.7	3.56
Detroit	-9.7	4.71
LasVegas	-6.1	4.90
LosAngeles	-4.8	3.05
MiamiFt.Lauderdale	-6.4	5.63
MinneapolisStPaul	-3.4	3.01
NewYork	-3.8	3.29
Phoenix	-7.3	3.26
Portland	3.8	1.93
SanDiego	-7.8	3.45
SanFrancisco	-4.1	2.29
Seattle	6.9	1.65
Tampa	-8.8	4.60
WashingtonDC	-7.2	3.14

In [ ]:
```python
indicators2['PriceChange'].mean()
```

Out[ ]:
```
-3.427777777777778
```

In [ ]:
```python
indicators2['LoanPaymentsOverdue'].mean()
```

Out[ ]:
```
3.532222222222222
```

In [ ]:
```python
len(indicators2.index)
```

Out[ ]:
```
18
```

In [ ]:
```python
SXX = ((indicators2['LoanPaymentsOverdue'] - indicators2['LoanPaymentsOverdue'].mean())**2).sum()
SXX
```

Out[ ]:
```
19.16011111111111
```

In [ ]:
```python
SYY = ((indicators2['PriceChange'] - indicators2['PriceChange'].mean())**2).sum()
SYY
```

Out[ ]:
```
347.0161111111111
```

In [ ]:
```python
SXY = ((indicators2['LoanPaymentsOverdue']
       - indicators2['LoanPaymentsOverdue'].mean())*(indicators2['PriceChange']
                                                     - indicators2['PriceChange'].mean())).sum()
SXY
```

Out[ ]:
```
-43.0818888888889
```

In [ ]:
```python
beta1 = SXY / SXX
beta1
```

Out[ ]:
```
-2.248519783578152
```

In [ ]:
```python
beta0 = indicators2['PriceChange'].mean() - beta1 * indicators2['LoanPaymentsOverdue'].mean()
beta0
```

Out[ ]:
```
4.514493768883272
```

In [ ]:
```python
s = (SYY / (len(indicators2.index)-2))**(1/2)
s
```

Out[ ]:
```
4.657092112514465
```

In [ ]:
```python
st.t.ppf(0.975, len(indicators2.index)-2)
```

Out[ ]:
```
2.1199052992210112
```

In [ ]:
```python
beta1 - st.t.ppf(0.975, len(indicators2.index)-2) * s / (SXX)**(1/2)
```

Out[ ]:
```
-4.503964968807471
```

In [ ]:
```python
beta1 + st.t.ppf(0.975, len(indicators2.index)-2) * s / (SXX)**(1/2)
```

Out[ ]:
```
0.00692540165116684
```

Thus, the 95% confidence interval is (-4.503964968807471, 0.00692540165116684).

In [ ]:
```python
Estimate = beta0 + beta1 * 4
Estimate
```

Out[ ]:
```
-4.479585365429337
```

In [ ]:
```python
lm_model = smf.ols(formula='PriceChange ~ LoanPaymentsOverdue', data=indicators2).fit()
```

In [ ]:
```python
lm_model.summary()
```

```
/Users/user/opt/anaconda3/lib/python3.9/site-packages/scipy/stats/stats.py:1541: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n=18
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[ ]:

OLS Regression Results

Dep. Variable:	PriceChange	**R-squared:**	0.279
Model:	OLS	**Adj. R-squared:**	0.234
Method:	Least Squares	**F-statistic:**	6.196
Date:	Sat, 21 Jan 2023	**Prob (F-statistic):**	0.0242
Time:	12:30:21	**Log-Likelihood:**	-49.226
No. Observations:	18	**AIC:**	102.5
Df Residuals:	16	**BIC:**	104.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4.5145	3.324	1.358	0.193	-2.532	11.561
LoanPaymentsOverdue	-2.2485	0.903	-2.489	0.024	-4.163	-0.334

Omnibus:	2.121	**Durbin-Watson:**	2.009
Prob(Omnibus):	0.346	**Jarque-Bera (JB):**	1.403
Skew:	0.448	**Prob(JB):**	0.496
Kurtosis:	1.966	**Cond. No.**	14.0

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

In [ ]:
```python
lm_model.resid.std() # = residual standard error.
```

Out[ ]:
```
3.8359417252407346
```

In [ ]:
```python
(SYY / (len(indicators2.index)-2))**(1/2) / SXX**(1/2) # why is it different to 0.903?
```

Out[ ]:
```
1.0639367645611877
```

Anyway, the confidence interval of beta1 for alpha=0.05 is (-4.163, -0.334),
so it has an evidence of significant negative linear regression,
whereas R says that the same interval is (-4.504445, 0.006445185),
so it doesn't have an evidence of significant negative linear regression.

(b)

In [ ]:
```python
lm_model.params
```

Out[ ]:
```
Intercept              4.514494
LoanPaymentsOverdue   -2.248520
dtype: float64
```

In [ ]:
```python
lm_model.predict(DataFrame({"LoanPaymentsOverdue": [4]}))
```

Out[ ]:
```
0   -4.479585
dtype: float64
```

In [ ]:
```python
lm_model.predict(DataFrame({"LoanPaymentsOverdue": [4]})) - st.t.ppf(0.975, len(indicators2.index)-2) * 0.903
```

Out[ ]:
```
0   -6.39386
dtype: float64
```

In [ ]:
```python
lm_model.predict(DataFrame({"LoanPaymentsOverdue": [4]})) + st.t.ppf(0.975, len(indicators2.index)-2) * 0.903
```

Out[ ]:
```
0   -2.565311
dtype: float64
```

Thus, the 95% confidence interval for beta1 where x=4 is (-6.39386, -2.565311).
Therefore, 0% = 0.00 is not a feasible value for E(Y|X=4).

3.

(a) Find a 95% confidence interval for the start-up time, i.e. beta0.

In [ ]:
```python
invoices = pd.read_table("/Users/user/Desktop/Yonsei/Junior/3-2/Introduction to Data Analysis and Regression/Homework/invoices.txt", sep="\t")
```

In [ ]:
```python
invoices
```

Out[ ]:

	Day	Invoices	Time
0	1	149	2.1
1	2	60	1.8
2	3	188	2.3
3	4	23	0.8
4	5	201	2.7
5	6	58	1.0
6	7	77	1.7
7	8	222	3.1
8	9	181	2.8
9	10	30	1.0
10	11	110	1.5
11	12	83	1.2
12	13	60	0.8
13	14	25	1.0
14	15	173	2.0
15	16	169	2.5
16	17	190	2.9
17	18	233	3.4
18	19	289	4.1
19	20	45	1.2
20	21	193	2.5
21	22	70	1.8
22	23	241	3.8
23	24	103	1.5
24	25	163	2.8
25	26	120	2.5
26	27	201	3.3
27	28	135	2.0
28	29	80	1.7
29	30	29	1.5

In [ ]:
```python
lm_model_hw2_3 = smf.ols('Time~Invoices', data=invoices).fit()
```

In [ ]:
```python
lm_model_hw2_3.summary()
```

Out[ ]:

OLS Regression Results

Dep. Variable:	Time	**R-squared:**	0.872
Model:	OLS	**Adj. R-squared:**	0.867
Method:	Least Squares	**F-statistic:**	190.4
Date:	Sat, 21 Jan 2023	**Prob (F-statistic):**	5.17e-14
Time:	12:35:12	**Log-Likelihood:**	-8.2528
No. Observations:	30	**AIC:**	20.51
Df Residuals:	28	**BIC:**	23.31
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.6417	0.122	5.248	0.000	0.391	0.892
Invoices	0.0113	0.001	13.797	0.000	0.010	0.013

Omnibus:	2.815	**Durbin-Watson:**	1.760
Prob(Omnibus):	0.245	**Jarque-Bera (JB):**	1.341
Skew:	-0.042	**Prob(JB):**	0.511
Kurtosis:	1.968	**Cond. No.**	303.

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

Thus, the 95% confidence interval for beta0 = (0.391, 0.892).

(b) Suppose that $\beta_1 = 0.01$. Test the null hypothesis $H_0 : \beta_1 = 0.01$ against a two-sided alternative.

In [ ]:
```python
lm_model_hw2_3.HC3_se.Invoices
```

Out[ ]:
```
0.0007872634462536831
```

In [ ]:
```python
(lm_model_hw2_3.params.Invoices - 0.01) / lm_model_hw2_3.HC3_se.Invoices
```

Out[ ]:
```
1.640675810336413
```

In [ ]:
```python
st.t.ppf(0.975, len(invoices.index)-2)
```

Out[ ]:
```
2.048407141795244
```

Thus, 1.64 < 2.04, so that we cannot reject the null.
Therefore, we cannot say that $\beta_1 = 0.01$.

(c) Find a point estimate and a 95% prediction interval for the time taken to process 130 invoices.

In [ ]:
```python
lm_model_hw2_3.predict(DataFrame({"Invoices":[130]}))
```

Out[ ]:
```
0    2.109624
dtype: float64
```

In [ ]:
```python
sxx =((invoices['Invoices'] - invoices['Invoices'].mean())**2).sum()
sxx
```

Out[ ]:
```
162366.96666666662
```

In [ ]:
```python
s_hw2_3 = (sum((lm_model_hw2_3.resid)**2) / (len(invoices.index)-2))**(1/2)
s_hw2_3
```

Out[ ]:
```
0.3297733332485559
```

In [ ]:
```python
lm_model_hw2_3.resid.std() # a little bit different from residual standard error.
```

Out[ ]:
```
0.3240377073104922
```

In [ ]:
```python
se130_hw2_3 = s_hw2_3 * (1 + (1/len(invoices.index)) + (((130 - invoices['Invoices'].mean())**2) / sxx))**(1/2)
se130_hw2_3
```

Out[ ]:
```
0.3352245025503466
```

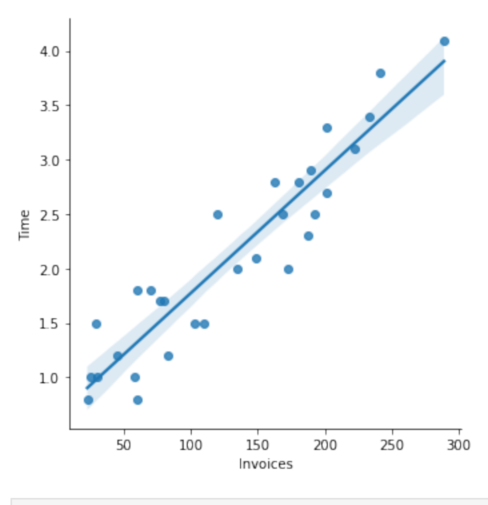
In [ ]:
```python
print(lm_model_hw2_3.predict(DataFrame({"Invoices":[130]})) - st.t.ppf(0.975, len(invoices.index)-2) * se130_hw2_3)
print(lm_model_hw2_3.predict(DataFrame({"Invoices":[130]})) + st.t.ppf(0.975, len(invoices.index)-2) * se130_hw2_3)
```

```
0    1.422947
dtype: float64
0    2.7963
dtype: float64
```

Thus, the point estimate is 2.109624, and 95% prediction interval is (1.422947, 2.7963).

In [ ]:
```python
sns.lmplot(x='Invoices', y='Time', data=invoices)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7fb7aa108460>
```



In [ ]: