In [1]:
```
from typing import List
from typing import Tuple
from typing import Union

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm

from sklearn.linear_model import LinearRegression
from scipy.stats import binom
from scipy.stats import norm
from scipy.stats import ttest_ind
from tqdm import tqdm

sns.set(font_scale=1.5)
sns.set_style("whitegrid", {'grid.linestyle':'--'})
```

In [2]:
```
#import the data
data = pd.read_csv("https://raw.githubusercontent.com/changyaochen/MECE4520/master/data/simple_linear_re

#create header row
col_Names=["X", "Y"]
data = pd.read_csv("https://raw.githubusercontent.com/changyaochen/MECE4520/master/data/simple_linear_re
data #output the table
```

Out[2]:

	X	Y
0	-15.000	-475.672
1	-14.824	-309.553
2	-14.648	-353.402
3	-14.472	-478.037
4	-14.296	-399.064
...	...	...
195	19.296	92.122
196	19.472	322.165
197	19.648	342.109
198	19.824	333.040
199	20.000	350.654

200 rows × 2 columns

In [3]:
```
#simple linear regression
```

In [4]:
```
#define data

X = np.append(np.ones(shape=(data.shape[0], 1)), data[["X"]].values, axis=1)
Y = np.append(np.ones(shape=(data.shape[0], 1)), data[["Y"]].values, axis=1)

print(X.shape)
print(Y.shape)
```

```
(200, 2)
(200, 2)
```

In [5]:
```
#calculate point estimates (coefficients) - much quicker than the previous way

betas = np.linalg.inv(X.T @ X) @ X.T @ Y
beta0 = betas[0,1]
beta1 = betas[1,1]

print(f"Beta_0 is {beta0}")
print(f"Beta_1 is {beta1}")
```

```
Beta_0 is -26.402462862372595
Beta_1 is 15.38160914494905
```

In [6]:
```
#calculate the standard errors

Y_hat = X @ betas
residual = Y - Y_hat
var = np.var(residual, ddof = X.shape[0])

se = np.sqrt(var * np.linalg.inv(X.T @ X))

SE_beta_0 = se[0,0]
SE_beta_1 = se[1,1]

print(f"The standard error for beta_0 is: {SE_beta_0:5.4f}")
print(f"The standard error for beta_1 is: {SE_beta_1:5.4f}")
```

```
The standard error for beta_0 is: 6.5029
The standard error for beta_1 is: 0.6218

/var/folders/3v/gxl3z6yn3fd_gv2bypr81hl80000gn/T/ipykernel_93389/1796252573.py:7: RuntimeWarning: inva
lid value encountered in sqrt
  se = np.sqrt(var * np.linalg.inv(X.T @ X))
```

In [7]:
```
#calculate R2

r2 = np.power(Y_hat - np.mean(Y), 2).sum() / np.power(Y - np.mean(Y), 2).sum()

print(f"The value of R squared is {r2}")
```

```
The value of R squared is 0.7541128378551109
```

In [8]:

```python
#Plot the data using the 'statsmodels' library
model_1 = smf.ols(formula='Y ~ X', data=data)
result_1 = model_1.fit()
print(result_1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.754
Model:                            OLS   Adj. R-squared:                  0.752
Method:                 Least Squares   F-statistic:                     605.7
Date:                Mon, 23 Oct 2023   Prob (F-statistic):           3.78e-62
Time:                        00:24:56   Log-Likelihood:                -1182.2
No. Observations:                 200   AIC:                             2368.
Df Residuals:                     198   BIC:                             2375.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -26.4025      6.536     -4.040      0.000     -39.291     -13.514
X             15.3816      0.625     24.612      0.000      14.149      16.614
==============================================================================
Omnibus:                        0.509   Durbin-Watson:                   1.513
Prob(Omnibus):                  0.775   Jarque-Bera (JB):                0.387
```

In [9]:

```python
#calculate the confidence interval for a new x value of 10 and the predicted y

x_new = 10
var = np.var(residual, ddof=X.shape[0])
multiplier = 1.96   # multiplier = t.ppf(q=0.975, df=X.shape[0] - X.shape[1])

y_hat_new = (beta0 + beta1 * x_new)
print(f"For a new x value of {x_new}, the predicted y value is {y_hat_new}")

delta = np.sqrt(var) * np.sqrt(1 / X.shape[0] + (x_new - np.mean(X[:, 1]))**2 / np.sum((X[:, 1] - np.mea

print(f"The lower bound of the 95% CI is: {y_hat_new - multiplier * delta:5.3f}")
print(f"The upper bound of the 95% CI is: {y_hat_new + multiplier * delta:5.3f}")
```

```
For a new x value of 10, the predicted y value is 127.4136285871179
The lower bound of the 95% CI is: 112.028
The upper bound of the 95% CI is: 142.800
```

In [23]:
```
#calculate if the result is "significant" - equivalently, does the (95%) confidence interval of Beta_1 i
CI_lower = y_hat_new - multiplier * delta
CI_higher = y_hat_new + multiplier * delta

def confidence_interval(number, intervals):
    for interval in intervals:
        if interval[0] <= number <= interval[1]:
            return True
    return False

intervals = [(CI_lower, CI_higher)]
number = 0

if confidence_interval(number, intervals):
    print("The result is significant")
else:
    print("The result is not significant")
```

```
The result is significant
```

In [11]:
```
#EXTRA - calculate the prediction interval (PI)
delta = np.sqrt(var) * np.sqrt(1 + 1 / X.shape[0] + (x_new - np.mean(X[:, 1]))**2 / np.sum((X[:, 1] - np

print(f"The lower bound of the 95% PI is: {y_hat_new - multiplier * delta:5.3f}")
print(f"The upper bound of the 95% PI is: {y_hat_new + multiplier * delta:5.3f}")
```

```
The lower bound of the 95% PI is: -48.286
The upper bound of the 95% PI is: 303.113
```

In [12]:
```
#multiple linear regression
```

In [13]:
```
#import the data
data = pd.read_csv("https://raw.githubusercontent.com/changyaochen/MECE4520/master/data/simple_linear_re

col_Names=["X", "Y", "X_squared"]
data = pd.read_csv("https://raw.githubusercontent.com/changyaochen/MECE4520/master/data/simple_linear_re
data
```

Out[13]:

	X	Y	X_squared
0	-15.000	-475.672	NaN
1	-14.824	-309.553	NaN
2	-14.648	-353.402	NaN
3	-14.472	-478.037	NaN
4	-14.296	-399.064	NaN
...	...	...	...
195	19.296	92.122	NaN
196	19.472	322.165	NaN
197	19.648	342.109	NaN
198	19.824	333.040	NaN
199	20.000	350.654	NaN

200 rows × 3 columns

In [14]:
```
#add X_squared to the data set
df = pd.DataFrame(data)

df['X_squared'] = df['X'] ** 2

print(df)
```

```
           X        Y   X_squared
0   -15.000 -475.672  225.000000
1   -14.824 -309.553  219.750976
2   -14.648 -353.402  214.563904
3   -14.472 -478.037  209.438784
4   -14.296 -399.064  204.375616
..      ...      ...         ...
195  19.296   92.122  372.335616
196  19.472  322.165  379.158784
197  19.648  342.109  386.043904
198  19.824  333.040  392.990976
199  20.000  350.654  400.000000

[200 rows x 3 columns]
```

In [15]:
```
X = data[["X", "X_squared"]].values
X = np.append(np.ones((X.shape[0], 1)), X, axis=1)
print(X.shape)
```

```
(200, 3)
```

In [16]:
```
#calculate the point estimates
betas_2 = np.linalg.inv(X.T @ X) @ X.T @ Y

beta0_2 = betas_2[0,1]
beta1_2 = betas_2[1,1]
beta2_2 = betas_2[2,1]

print(f"Beta_0 is {beta0_2}")
print(f"Beta_1 is {beta1_2}")
print(f"Beta_2 is {beta2_2}")
```

```
Beta_0 is -7.774602003296486
Beta_1 is 16.343206107149058
Beta_2 is -0.1923193924400019
```

In [17]:
```
#calculate the standard errors
Y_hat_2 = X @ betas_2
residual_2 = Y - Y_hat_2
var_2 = np.var(residual_2, ddof=X.shape[0])

se = np.sqrt(var * np.linalg.inv(X.T @ X))

SE_beta_0 = se[0,0]
SE_beta_1 = se[1,1]
SE_beta_2 = se[2,2]

print(f"The standard error for beta_0 is: {SE_beta_0:5.4f}")
print(f"The standard error for beta_1 is: {SE_beta_1:5.4f}")
print(f"The standard error for beta_2 is: {SE_beta_2:5.4f}")
```

```
The standard error for beta_0 is: 9.2882
The standard error for beta_1 is: 0.7099
The standard error for beta_2 is: 0.0685

/var/folders/3v/gxl3z6yn3fd_gv2bypr81hl80000gn/T/ipykernel_93389/3309654660.py:6: RuntimeWarning: inva
lid value encountered in sqrt
  se = np.sqrt(var * np.linalg.inv(X.T @ X))
```

In [18]:
```
#calculate R2
r2_2 = np.power(Y_hat_2 - np.mean(Y), 2).sum() / np.power(Y - np.mean(Y), 2).sum()

print(f"The value of R squared is {r2_2}")
```

```
The value of R squared is 0.7638122352031607
```

In [19]:
```
#Plot the data using the 'statsmodels' library
model_2 = smf.ols(formula = 'Y ~ X + X_squared', data=data)
result_2 = model_2.fit()
print(result_2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.763
Model:                            OLS   Adj. R-squared:                  0.761
Method:                 Least Squares   F-statistic:                     317.8
Date:                Mon, 23 Oct 2023   Prob (F-statistic):           2.22e-62
Time:                        00:25:02   Log-Likelihood:                -1178.2
No. Observations:                 200   AIC:                             2362.
Df Residuals:                     197   BIC:                             2372.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -7.7746      9.172     -0.848      0.398     -25.863      10.314
X             16.3432      0.701     23.314      0.000      14.961      17.726
X_squared     -0.1923      0.068     -2.844      0.005      -0.326      -0.059
==============================================================================
Omnibus:                        0.987   Durbin-Watson:                   1.575
Prob(Omnibus):                  0.610   Jarque-Bera (JB):                0.993
Skew:                          -0.028   Prob(JB):                        0.609
Kurtosis:                       2.659   Cond. No.                         223.
==============================================================================

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

In [20]:
```
#calculate if the result is "significant" - equivalently, does the (95%) confidence interval of Beta_1 i
CI_lower = 14.961
CI_higher = 17.726
#from the table - these are the values of the confidence interval for Beta_1

def confidence_interval(number, intervals):
    for interval in intervals:
        if interval[0] <= number <= interval[1]:
            return True
    return False

intervals = [(CI_lower, CI_higher)]
number = 0

if confidence_interval(number, intervals):
    print("The result is significant")
else:
    print("The result is not significant")
```

```
The result is not significant
```

In [21]:
```
#Problem 3 - prove that maximizing the likelihood leads to minimizing the mean squared error

#this should be true since the "best" coefficients are the ones that maximize the likelihood of the data
#minimizing the mean squared error occurs when the point estimates of beta_0 and beta_1 maximize the lik
#minimizing the mean squared error = maximizing the likelihood - when the errors are normally distribute
```

In [22]:
```
#random data - can mix up the numbers for different results

np.random.seed(5)
x_3 = np.random.rand(100)
error_3 = np.random.normal(7, 13, size = 100) #linear data sample
y_3 = 2 * x_3 + 1 + error_3

X_3 = sm.add_constant(x_3) #add number from within the parameters

model_3 = sm.OLS(y_3, X_3)
result_3 = model_3.fit()

print(result_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                 -0.010
Method:                 Least Squares   F-statistic:                   0.01782
Date:                Mon, 23 Oct 2023   Prob (F-statistic):              0.894
Time:                        00:25:04   Log-Likelihood:                -401.93
No. Observations:                 100   AIC:                             807.9
Df Residuals:                      98   BIC:                             813.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          9.2121      2.597      3.547      0.001       4.059      14.366
x1             0.6001      4.495      0.134      0.894      -8.320       9.520
==============================================================================
Omnibus:                        0.677   Durbin-Watson:                   1.660
Prob(Omnibus):                  0.713   Jarque-Bera (JB):                0.802
Skew:                           0.121   Prob(JB):                        0.670
Kurtosis:                       2.635   Cond. No.                         4.17
==============================================================================

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