Assignment 4

Importing necessary python libraries

In [1]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import seaborn as sns
```

Correlation

Correlation of independent variables

In [2]:
```
rng = np.random.default_rng(24102021)
```

Drawing 100 normals for each variable in a single iteration and the total iteration is 1000

In [3]:
```
cor_array = np.zeros(1000)
for i in range(1000):
    xs = pd.Series(rng.standard_normal(100))
    ys = pd.Series(rng.standard_normal(100))
    cor_array[i] = xs.corr(ys)

print("Mean: ", np.mean(cor_array))
print("Variance: ", np.var(cor_array))
```

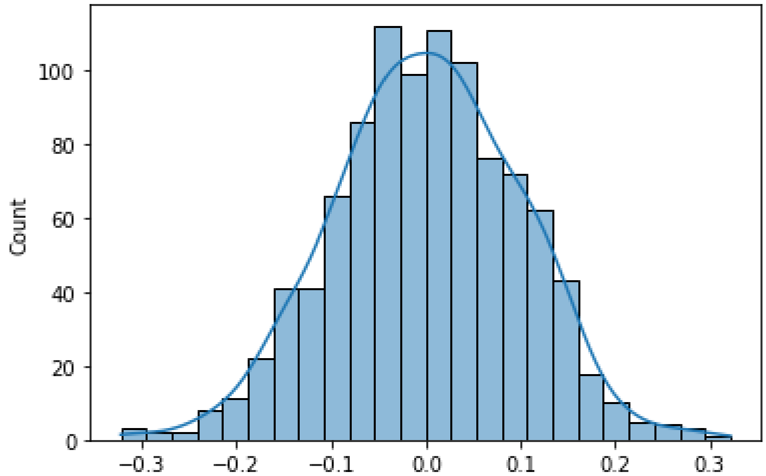
```
Mean:  0.0007069468817587299
Variance:  0.009592948238514936
```

The mean and variance of this correlation coefficient is very close to zero which indicates that there is no relationship between x and y. Thus, they are independent/uncorrelated.

Plotting the distribution

In [4]:
```
sns.histplot(cor_array, kde=True)
```

Out[4]:
```
<AxesSubplot:ylabel='Count'>
```



Repeating the previous simulation by drawing 1000 normals for each variable in a single iteration

In [5]:

```
cor_array = np.zeros(1000)
for i in range(1000):
    xs = pd.Series(rng.standard_normal(1000))
    ys = pd.Series(rng.standard_normal(1000))
    cor_array[i] = xs.corr(ys)

print("Mean: ", np.mean(cor_array))
print("Variance: ", np.var(cor_array))
```

```
Mean:  -0.0021920021758015194
Variance:  0.0009517476178547055
```

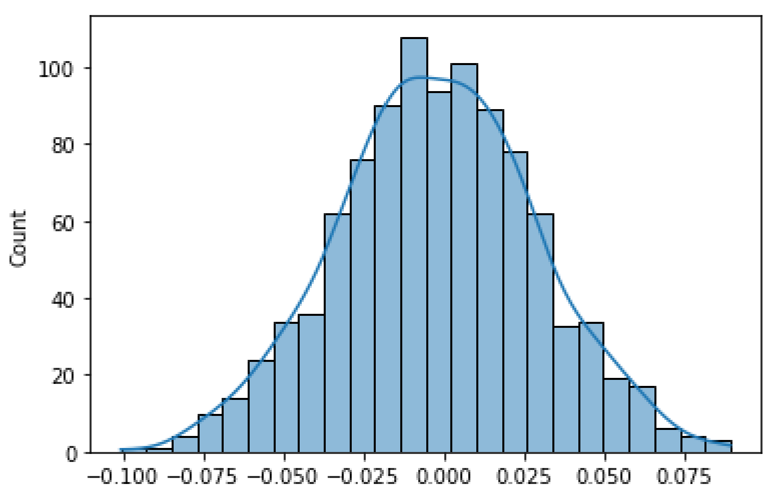
Here also the mean and variance of this correlation coefficient is close to zero which indicates that there is no relationship between x and y. Thus, they are independant/uncorrelated.

For 1000 draws instead of 100 in per iteration of 1000, the mean of correlation coefficient decreases and it goes more closer to zero, also the variance decreases slighty.

In [6]:

```
sns.histplot(cor_array, kde=True)
```

Out[6]: <AxesSubplot:ylabel='Count'>



100 draws of each variable per iteration to compute 1000 correlation coefficients between

correlated variables

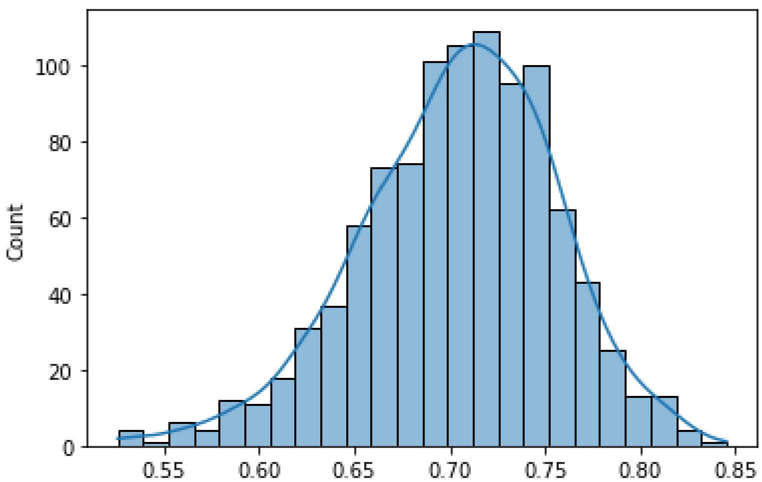
In [7]:
```
cor_array = np.zeros(1000)
for i in range(1000):
    xs = pd.Series(rng.standard_normal(100))
    ys = pd.Series(rng.standard_normal(100))
    zs = xs + ys
    cor_array[i] = xs.corr(zs)

print("Mean: ", np.mean(cor_array))
print("Variance: ", np.var(cor_array))
```

```
Mean:  0.7041384007528332
Variance:  0.002625132168986839
```

In [8]:
```
sns.histplot(cor_array, kde=True)
```

Out[8]: <AxesSubplot:ylabel='Count'>



1000 draws of each variable per iteration to compute 1000 correlation coefficients between correlated variables

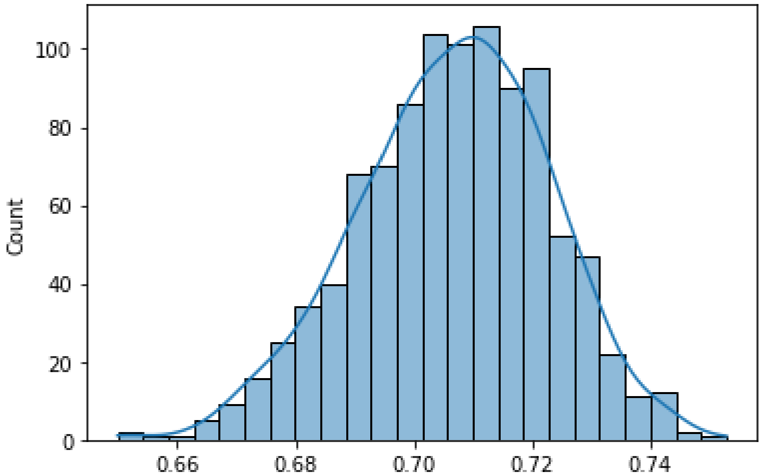
In [9]:
```
cor_array = np.zeros(1000)
for i in range(1000):
    xs = pd.Series(rng.standard_normal(1000))
    ys = pd.Series(rng.standard_normal(1000))
    zs = xs + ys
    cor_array[i] = xs.corr(zs)

print("Mean: ", np.mean(cor_array))
print("Variance: ", np.var(cor_array))
```

```
Mean:  0.706561559351059
Variance:  0.00026027859693015306
```

In [10]:
```
sns.histplot(cor_array, kde=True)
```

Out[10]: <AxesSubplot:ylabel='Count'>



10000 draws of each variable per iteration to compute 1000 correlation coefficients between correlated variables

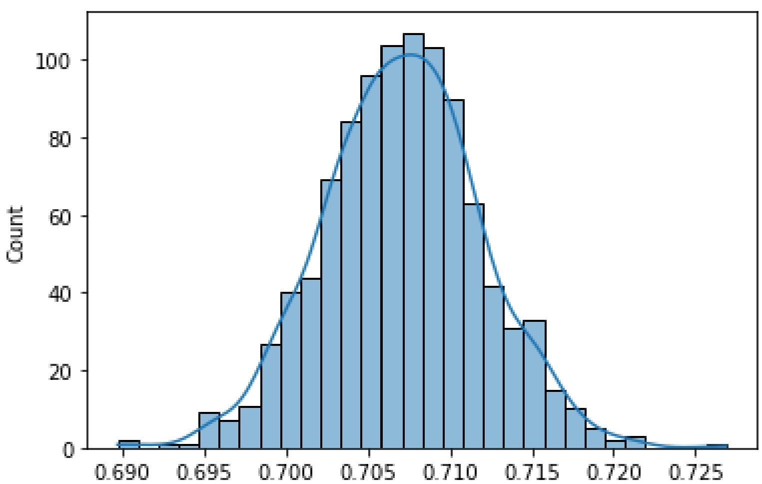
In [11]:
```
cor_array = np.zeros(1000)
for i in range(1000):
    xs = pd.Series(rng.standard_normal(10000))
    ys = pd.Series(rng.standard_normal(10000))
    zs = xs + ys
    cor_array[i] = xs.corr(zs)

print("Mean: ", np.mean(cor_array))
print("Variance: ", np.var(cor_array))
```

```
Mean:  0.707102053368046
Variance:  2.3133264623538096e-05
```

In [12]:
```
sns.histplot(cor_array, kde=True)
```

Out[12]: <AxesSubplot:ylabel='Count'>



For 100 draws, the mean of the corelation coefficient is 0.704 which is smaller than 0.707, distribution is left skewed. For 1000 draws, the mean of the corelation coefficient is approximately equal to 0.706, distribution contains outlier and almost symmetrical and for 10000 draws, the mean of the corelation coefficient is exactly 0.707, distribution is right skewed.

Linear Regression

Predicting Y with X for the given distribution and fit a linear model to this data

In [13]:
```
xs = rng.standard_normal(1000)
errs = rng.standard_normal(1000)
ys = 0 + 1 * xs + errs
data = pd.DataFrame({
    'X': xs,
    'Y': ys
})
data.head()
```

Out[13]:

	X	Y
0	1.129827	-0.581514
1	-0.865761	-1.450910
2	1.397495	2.522028
3	2.034763	0.923421
4	0.657732	0.981118

In [14]:
```
model = smf.ols('Y ~ X', data=data).fit()
model.summary()
```

Out[14]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.505
Model:	OLS	**Adj. R-squared:**	0.504
Method:	Least Squares	**F-statistic:**	1016.
Date:	Sun, 24 Oct 2021	**Prob (F-statistic):**	2.20e-154
Time:	20:56:39	**Log-Likelihood:**	-1390.1
No. Observations:	1000	**AIC:**	2784.
Df Residuals:	998	**BIC:**	2794.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.0251	0.031	-0.815	0.415	-0.085	0.035
X	0.9922	0.031	31.882	0.000	0.931	1.053

Omnibus:	2.108	**Durbin-Watson:**	2.027
Prob(Omnibus):	0.348	**Jarque-Bera (JB):**	2.049
Skew:	0.060	**Prob(JB):**	0.359
Kurtosis:	2.814	**Cond. No.**	1.01

Notes:

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

In [15]:

```
print("Intercept is ", model.params['Intercept'])
print("Slope is ", model.params['X'])
print("R^2 is ", model.rsquared )
```

```
Intercept is  -0.025069257711995425
Slope is  0.9921757747011009
R^2 is  0.5045802167978103
```

From the model, we get the intercept -0.025 which is closely equal to the given intercept of 0 and we get the slope 0.992 which also approximately equal to the given slope 1. The R^2 values is 0.505 which indicates that 50.5% of the variance of the outcome variable(Y) is explained by the predictor variable(X). This means the model is almost a good fit for the data.

Residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions

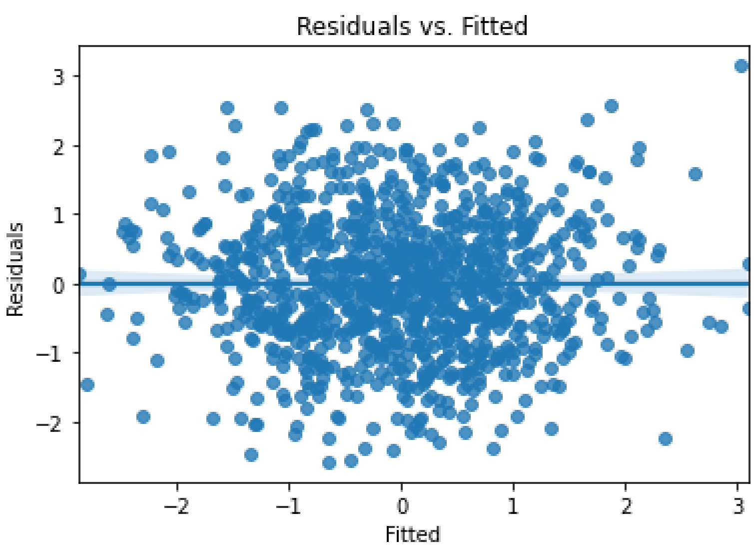
In [16]:

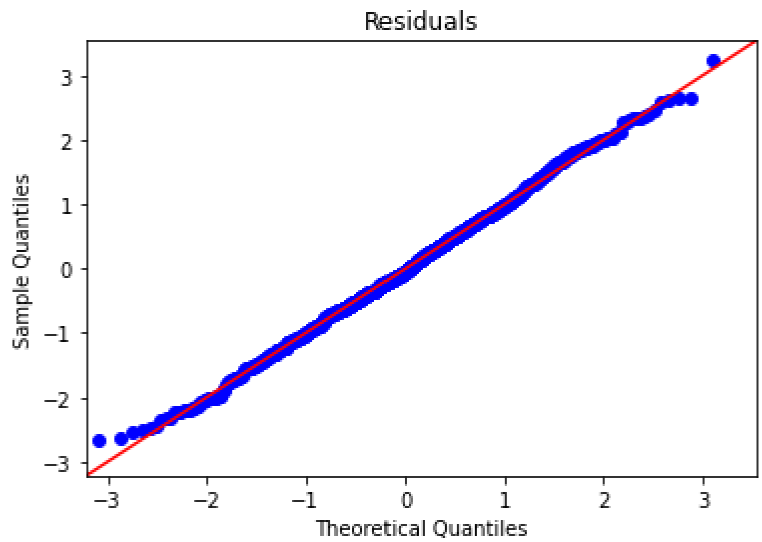
```
def plot_lm_diag(fit):
    "Plot linear fit diagnostics"
    sns.regplot(x=fit.fittedvalues, y=fit.resid)
    plt.xlabel('Fitted')
    plt.ylabel('Residuals')
    plt.title('Residuals vs. Fitted')
    plt.show()

    sm.qqplot(fit.resid, fit=True, line='45')
    plt.title('Residuals')
    plt.show()
```

In [17]:

```
plot_lm_diag(model)
```





From residuals vs. fitted plot, we can find that the pattern approximately looks homoscedastic since the variance is not sacttered too much and linearity exists between x and y. From qqplot we see that, residulas are pretty normally distributed except there are some outliers.

Repeating the above simulation 1000 times, fitting a linear model each time. Showing the mean, variance, and a distribution plot of the intercept, slope, and R^2 from these simulations.

In [18]:

```
intercepts = np.zeros(1000)
slopes = np.zeros(1000)
rsq = np.zeros(1000)

for i in range(1000):
    xs = rng.standard_normal(1000)
    errs = rng.standard_normal(1000)
    ys = 0 + 1 * xs + errs
    data = pd.DataFrame({
        'X': xs,
        'Y': ys
    })
    model = smf.ols('Y ~ X', data=data).fit()
    intercepts[i] = model.params['Intercept']
    slopes[i] = model.params['X']
    rsq[i] = model.rsquared

print("Intercept->", "Mean: " + str(np.mean(intercepts)), "and Variance: " + str
print("Slope->", "Mean: " + str(np.mean(slopes)), "and Variance: " + str(np.var(
print("R^2->", "Mean: " + str(np.mean(rsq)), " and Variance: " + str(np.var(rsq)
```

```
Intercept-> Mean: -0.00025273559588850035 and Variance: 0.0009718187635945533
Slope-> Mean: 1.0016881372870643 and Variance: 0.0010348570641509403
R^2-> Mean: 0.5004886028098166  and Variance: 0.0004980086667729802
```

In [19]:

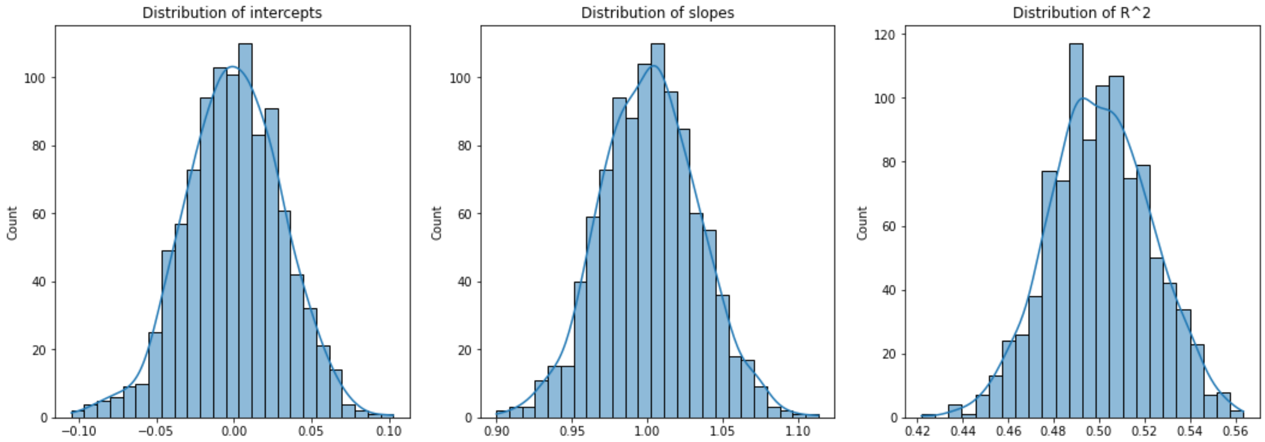
```
plt.figure(figsize=(18,6))
plt.subplot(1,3,1)
sns.histplot(intercepts, kde=True)
plt.title("Distribution of intercepts")

plt.subplot(1,3,2)
sns.histplot(slopes, kde=True)
```

```
plt.title("Distribution of slopes")


plt.subplot(1,3,3)
sns.histplot(rsq, kde=True)
plt.title("Distribution of R^2")

plt.show()
```



The distribution of the intercepts, slopes and R^2 all are symmetric/normal which implies that mean and median are almost equal.

Fitting a model to data with α=1,and β=4

In [20]:

```
xs = rng.standard_normal(1000)
errs = rng.standard_normal(1000)
ys = 1 + 4 * xs + errs
data = pd.DataFrame({
    'X': xs,
    'Y': ys
})
data.head()
```

Out[20]:

	X	Y
0	1.287273	5.846553
1	0.995769	4.921597
2	-1.927833	-5.417060
3	-1.112076	-3.263086
4	0.703960	4.523538

In [21]:

```
model = smf.ols('Y ~ X', data=data).fit()
model.summary()
```

Out[21]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.946
Model:	OLS	**Adj. R-squared:**	0.946
Method:	Least Squares	**F-statistic:**	1.756e+04

Date:	Sun, 24 Oct 2021	**Prob (F-statistic):**	0.00
Time:	20:56:46	**Log-Likelihood:**	-1395.9
No. Observations:	1000	**AIC:**	2796.
Df Residuals:	998	**BIC:**	2806.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1.0090	0.031	32.620	0.000	0.948	1.070
X	4.0272	0.030	132.504	0.000	3.968	4.087

Omnibus:	1.361	**Durbin-Watson:**	1.886
Prob(Omnibus):	0.506	**Jarque-Bera (JB):**	1.440
Skew:	0.076	**Prob(JB):**	0.487
Kurtosis:	2.894	**Cond. No.**	1.02

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

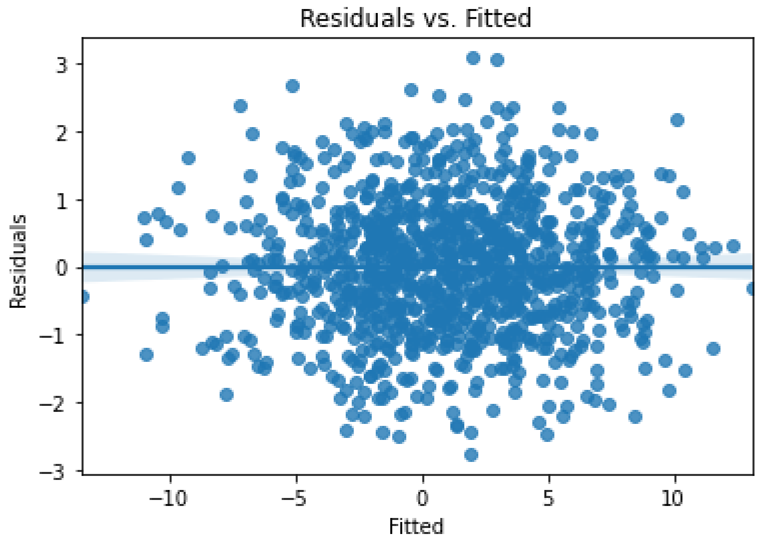
In [22]:
```
print("Intercept is ", model.params['Intercept'])
print("Slope is ", model.params['X'])
print("R^2 is ", model.rsquared )
```

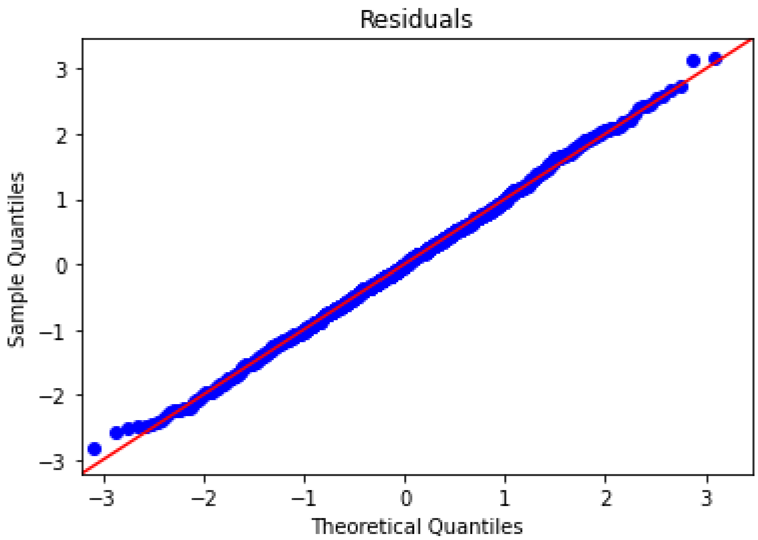
```
Intercept is  1.0090222946011944
Slope is  4.027170212904327
R^2 is  0.9462152276382634
```

From the model, we get the intercept 1.009 which is approximately equal to 1 and we get the slope 4.027 which also approximately equal to our given slope 4. The R^2 values is 0.946 which indicates that 94.6% of the variance of the outcome variable(Y) is explained by the predictor variable(X). And this shows that the model is definitely a proper fit for our data.

Residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions

In [23]:
```
plot_lm_diag(model)
```





From residuals vs. fitted plot, we can find that it looks like homoskedastic. Linearity exists between x and y. From qqplot we can see that, residulas are pretty normally distributed except for some outliers.

Repeating the above simulation for 1000 times

In [24]:

```
intercepts = np.zeros(1000)
slopes = np.zeros(1000)
rsq = np.zeros(1000)

for i in range(1000):
    xs = rng.standard_normal(1000)
    errs = rng.standard_normal(1000)
    ys = 1 + 4 * xs + errs
    data = pd.DataFrame({
        'X': xs,
        'Y': ys
    })
    model = smf.ols('Y ~ X', data=data).fit()
    intercepts[i] = model.params['Intercept']
    slopes[i] = model.params['X']
    rsq[i] = model.rsquared

print("Intercept->", "Mean: " + str(np.mean(intercepts)), "and Variance: " + str
```

```
print("Slope->", "Mean: " + str(np.mean(slopes)), "and Variance: " + str(np.var(
print("R^2->", "Mean: " + str(np.mean(rsq)), " and Variance: " + str(np.var(rsq)
```

```
Intercept-> Mean: 0.9987096210896318 and Variance: 0.0009754962880677402
Slope-> Mean: 3.999547615593469 and Variance: 0.0009807011689235276
R^2-> Mean: 0.941084339009865  and Variance: 1.4063261218571244e-05
```

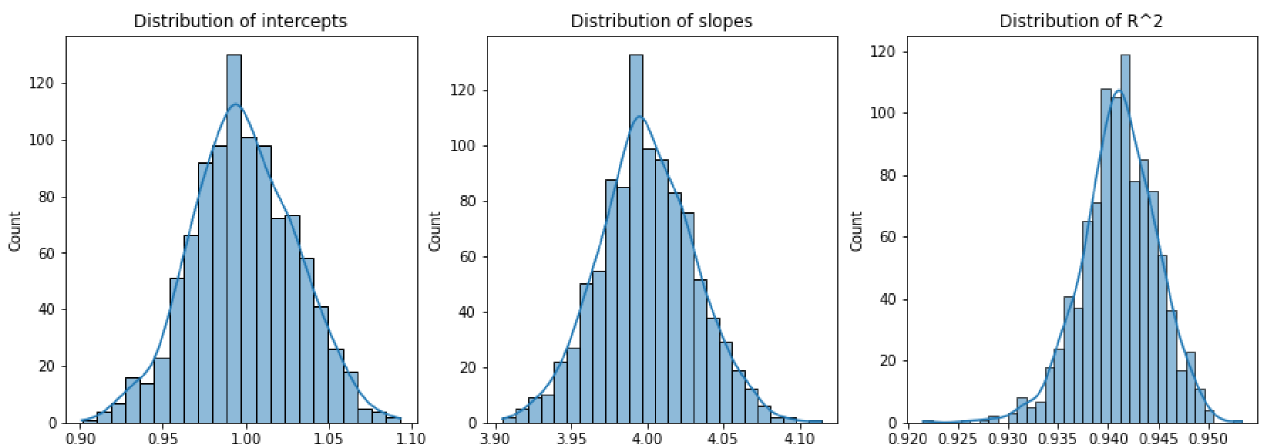
In [25]:

```
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
sns.histplot(intercepts, kde=True)
plt.title("Distribution of intercepts")

plt.subplot(1,3,2)
sns.histplot(slopes, kde=True)
plt.title("Distribution of slopes")


plt.subplot(1,3,3)
sns.histplot(rsq, kde=True)
plt.title("Distribution of R^2")

plt.show()
```



The distribution of the intercepts and slopes are symmetric but the distribution of R^2 is slightly left skewed.

Nonlinear Data

Generating 1000 data points for xs and errs(both are normally distributed) and calculate ys = 10+5*np.exp(xs)+errs

In [26]:

```
xs = rng.standard_normal(1000)
errs = np.random.normal(0, 5, 1000)
ys = 10 + 5*np.exp(xs) + errs
data = pd.DataFrame({
    'X': xs,
    'Y': ys
})
data.head()
```

Out[26]:

	X	Y

	X	Y
0	-0.403976	7.477277
1	1.324757	25.316034
2	0.904153	30.712616
3	-0.456311	9.627885
4	-0.982888	10.705041

Fit a linear model predicting y with x

In [27]:
```
model = smf.ols('Y ~ X', data=data).fit()
model.summary()
```

Out[27]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.446
Model:	OLS	**Adj. R-squared:**	0.445
Method:	Least Squares	**F-statistic:**	803.1
Date:	Sun, 24 Oct 2021	**Prob (F-statistic):**	4.24e-130
Time:	20:56:53	**Log-Likelihood:**	-3731.7
No. Observations:	1000	**AIC:**	7467.
Df Residuals:	998	**BIC:**	7477.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	18.7396	0.320	58.597	0.000	18.112	19.367
X	8.9468	0.316	28.339	0.000	8.327	9.566

Omnibus:	1252.304	**Durbin-Watson:**	1.970
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	246941.142
Skew:	6.282	**Prob(JB):**	0.00
Kurtosis:	78.952	**Cond. No.**	1.02

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

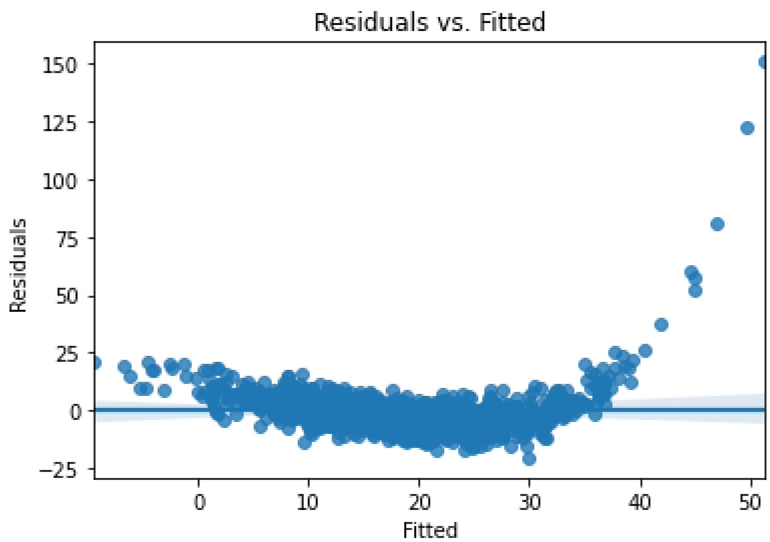
In [28]:
```
print("Intercept is ", model.params['Intercept'])
print("Slope is ", model.params['X'])
print("R^2 is ", model.rsquared )
```

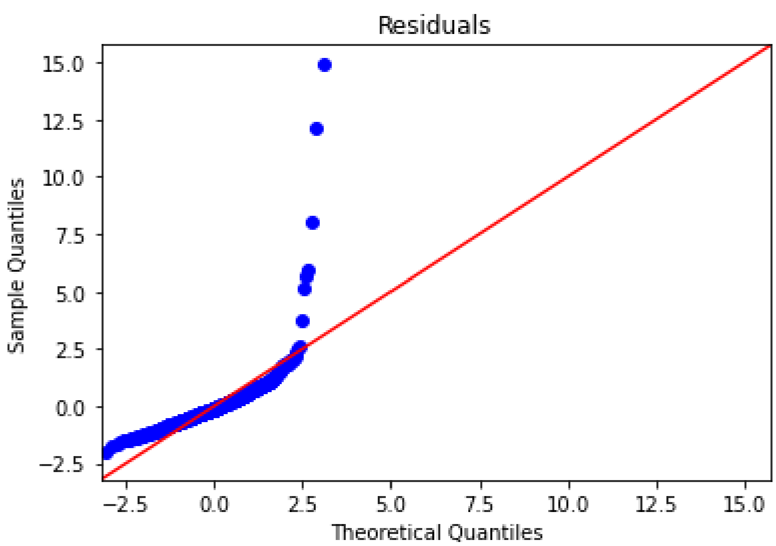
```
Intercept is  18.739629699945258
Slope is  8.946817867870593
R^2 is  0.44590146578859247
```

From the model, we get the intercept 18.739 which is not equal to the given intercept 10 and we get the slope 8.94. This happens because the data is non-linear. The R^2 values is 0.446 which indicates that 44.6% of the variance of the outcome variable(Y) is explained by the predictor variable(X).

Residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions

In [29]:
```
plot_lm_diag(model)
```

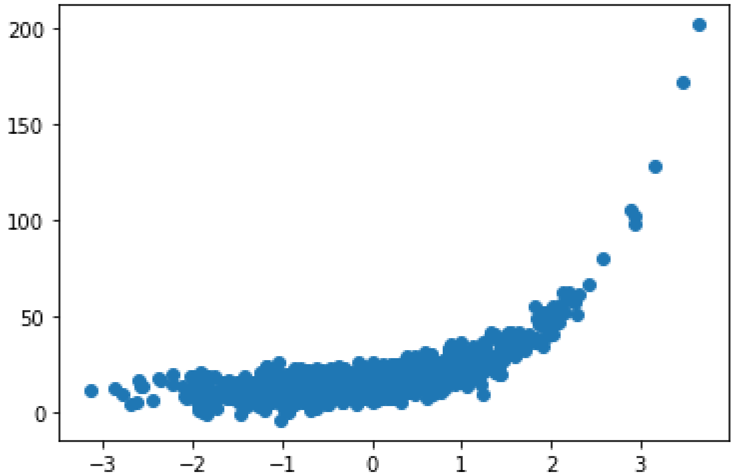




The residuals plot has a clear curve. Residuals has positive values for larger or smaller fitted values and negative values in the middle. Points are not scattered randomly around zero line from left to right although there are some outliers. That is why Homoscedasticity is present. Both in residuals vs fitted plot and qqplot, data have outliers and residuals is not normally distributed. Linearity property is violated for this data.

Drawing a scatter plot for X and Y.

In [30]:
```
plt.scatter(data['X'], data['Y'])
plt.show()
```



Generating 1000 data points for xs and errs(both are normally distributed) and calculating ys = -2+3 * xs^3 + errs

In [31]:

```
xs = rng.standard_normal(1000)
errs = np.random.normal(0, 5, 1000)
ys = -2 + 3 * xs**3 + errs
data = pd.DataFrame({
    'X': xs,
    'Y': ys
})
data.head()
```

Out[31]:

	X	Y
0	0.074450	-1.632348
1	2.527609	52.365540
2	0.678259	-0.157975
3	0.208523	-5.957724
4	1.402596	10.298770

In [32]:

```
model = smf.ols('Y ~ X', data=data).fit()
model.summary()
```

Out[32]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.545
Model:	OLS	**Adj. R-squared:**	0.544
Method:	Least Squares	**F-statistic:**	1194.
Date:	Sun, 24 Oct 2021	**Prob (F-statistic):**	1.08e-172
Time:	20:56:54	**Log-Likelihood:**	-3659.9
No. Observations:	1000	**AIC:**	7324.
Df Residuals:	998	**BIC:**	7334.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.3006	0.298	-7.729	0.000	-2.885	-1.716
X	10.1029	0.292	34.552	0.000	9.529	10.677

Omnibus:	315.097	**Durbin-Watson:**	1.983
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	15597.047
Skew:	0.636	**Prob(JB):**	0.00
Kurtosis:	22.306	**Cond. No.**	1.02

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

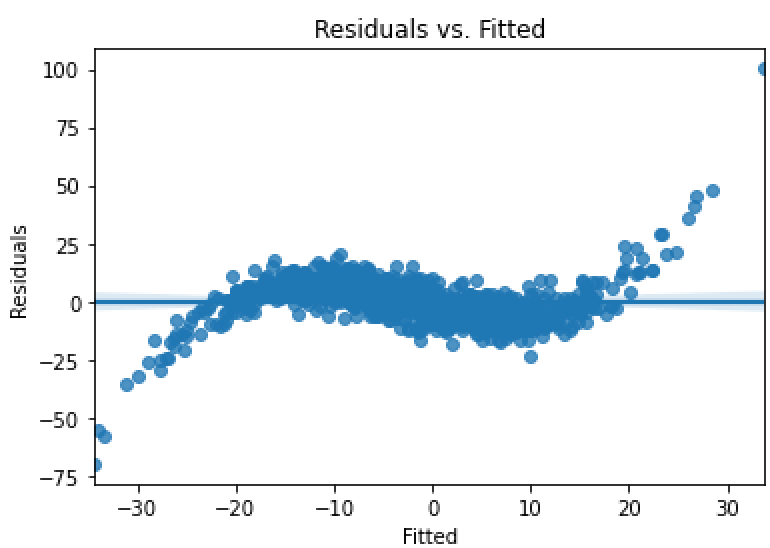
In [33]:
```
print("Intercept is ", model.params['Intercept'])
print("Slope is ", model.params['X'])
print("R^2 is ", model.rsquared )
```

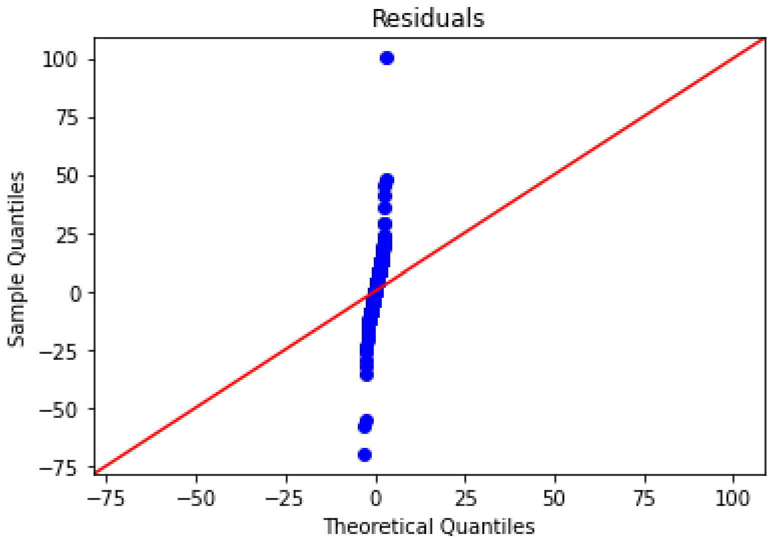
```
Intercept is  -2.3005718511644275
Slope is  10.10289787946856
R^2 is  0.5446795744882118
```

From the model, we get the intercept -2.3 which is closely equal to the given intercept -2 and we get the slope 10.1, this happens because the data is non-linear. The R^2 values is 0.5446 which indicates that 54.46% of the variance of the outcome variable(Y) is explained by the predictor variable(X). This model is a good fit for the data.

Residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions

In [34]:
```
plot_lm_diag(model)
```

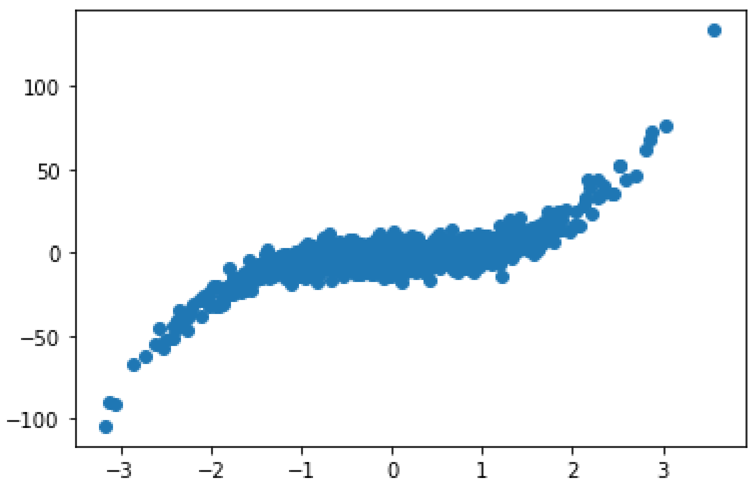




Homoscedasticity is present. Equal variance assumption is satisfied. Both residuals vs fitted plot and qqplot have outliers and residuals is not normally distributed. Here, linear model assumption of equal error is violated.

In [35]:

```
plt.scatter(data['X'], data['Y'])
plt.show()
```



Non-Normal Covariates

Generating 1000 data points for xs (Gamma Distributed) and errs(Normally distributed) and calculating ys = 10 + 0.3 * xs + errs

In [36]:

```
xs = rng.gamma(2, 1, 1000)
errs = rng.standard_normal(1000)
ys = 10 + 0.3 * xs + errs
data = pd.DataFrame({
    'X': xs,
    'Y': ys
})
data.head()
```

Out[36]:

	X	Y

	X	Y
0	2.796342	9.885310
1	0.894990	9.338713
2	0.224367	11.458785
3	4.131458	11.979442
4	1.441539	11.206950

Plotting the distribution of X and Y

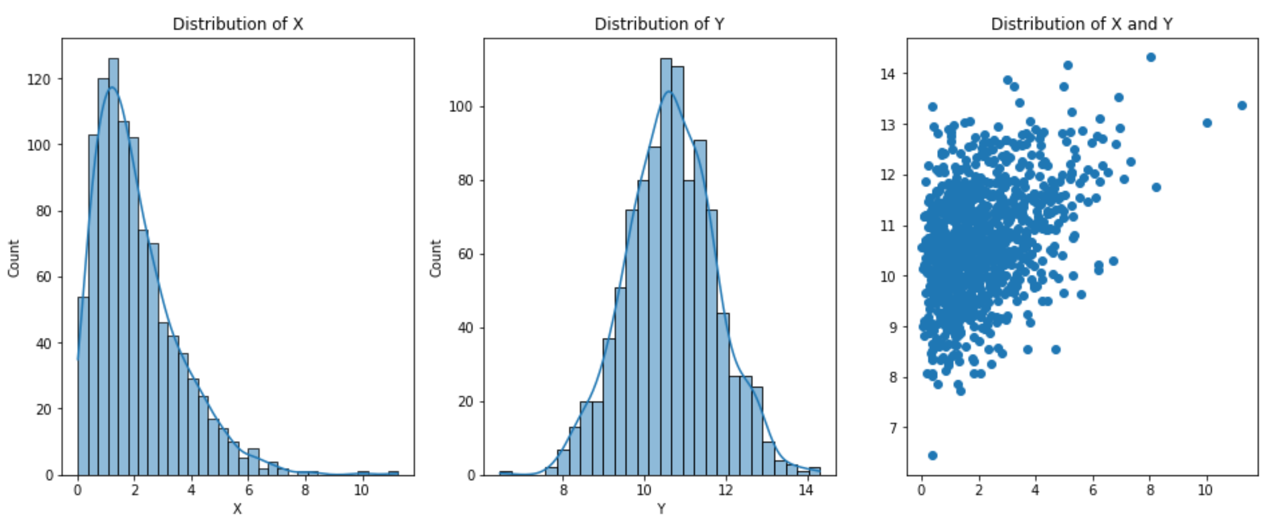
In [37]:
```
plt.figure(figsize=(16,6))

plt.subplot(1,3,1)
sns.histplot(data['X'], kde=True)
plt.title("Distribution of X")

plt.subplot(1,3,2)
sns.histplot(data['Y'], kde=True)
plt.title("Distribution of Y")


plt.subplot(1,3,3)
plt.scatter(data['X'],data['Y'])
plt.title("Distribution of X and Y")

plt.show()
```



The distribution of Y is normal but the distribution of X is right skewed.

Fitting a linear model for predicting y with x

In [38]:
```
model = smf.ols('Y ~ X', data=data).fit()
model.summary()
```

Out[38]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.154
Model:	OLS	**Adj. R-squared:**	0.153

Method:	Least Squares	**F-statistic:**	182.0
Date:	Sun, 24 Oct 2021	**Prob (F-statistic):**	3.16e-38
Time:	20:56:55	**Log-Likelihood:**	-1425.3
No. Observations:	1000	**AIC:**	2855.
Df Residuals:	998	**BIC:**	2864.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	10.0527	0.055	182.294	0.000	9.944	10.161
X	0.2924	0.022	13.491	0.000	0.250	0.335

Omnibus:	0.322	**Durbin-Watson:**	1.884
Prob(Omnibus):	0.851	**Jarque-Bera (JB):**	0.227
Skew:	-0.021	**Prob(JB):**	0.893
Kurtosis:	3.060	**Cond. No.**	4.88

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

In [39]:

```
print("Intercept is ", model.params['Intercept'])
print("Slope is ", model.params['X'])
print("R^2 is ", model.rsquared )
```

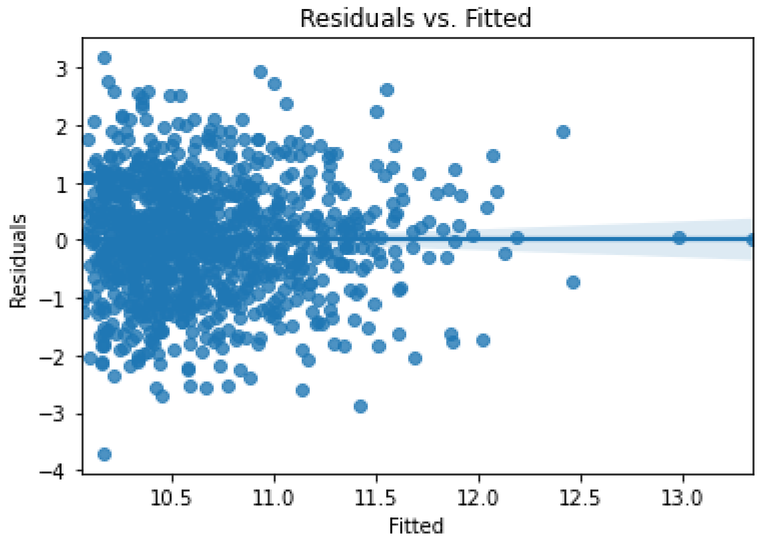
```
Intercept is  10.052691416378746
Slope is  0.29243423488262305
R^2 is  0.15424853806613037
```

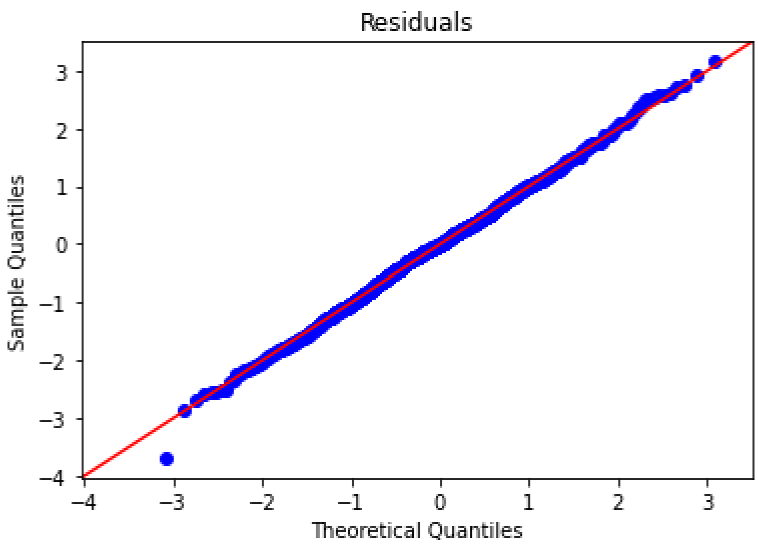
From the model, we get the intercept 10.05 which is approximately equal to the given intercept 10 and we get the slope 0.292 which also approximately equal to our given slope 0.3. The R^2 values is 0.1542 which indicates that only 15.42% of the variance of the outcome variable(Y) is explained by the predictor variable(X) which implies not a good model to fit for the data.

Residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions

In [40]:

```
plot_lm_diag(model)
```





From the Q-Q plot we see that normal distribution property is hold nicely. But from the Residuals vs. Fitted figure, we can see that the funnel shape which indicates the heteroscedasticity. Normal error assumption is satiesfied. Of course, it shows that the data fits on linear model.

Multiple Regression

Generating 1000 data points for xs1, xs2, errs (all are normally distributed) and calculating ys = 1 + 0.5*xs1* + 3xs2 + errs

In [41]:

```
xs1 = np.random.normal(10, 2, 1000)
xs2 = np.random.normal(-2, 5, 1000)
errs = rng.standard_normal(1000)
ys = 1 + 0.5*xs1 + 3*xs2 + errs
data = pd.DataFrame({
    'X1': xs1,
    'X2': xs2,
    'Y': ys
})
data.head()
```

Out[41]:

	X1	X2	Y

	X1	X2	Y
0	9.419430	-6.267051	-13.395866
1	5.922397	-8.159583	-19.234317
2	7.891919	-4.906548	-11.229522
3	9.296712	-5.173241	-8.566046
4	8.584609	-3.560265	-6.311484

In [42]:

```
model = smf.ols('Y ~ X1 + X2', data=data).fit()
model.summary()
```

Out[42]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.995
Model:	OLS	**Adj. R-squared:**	0.995
Method:	Least Squares	**F-statistic:**	1.029e+05
Date:	Sun, 24 Oct 2021	**Prob (F-statistic):**	0.00
Time:	20:56:56	**Log-Likelihood:**	-1429.2
No. Observations:	1000	**AIC:**	2864.
Df Residuals:	997	**BIC:**	2879.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.7529	0.166	4.537	0.000	0.427	1.079
X1	0.5241	0.016	32.257	0.000	0.492	0.556
X2	2.9949	0.007	450.848	0.000	2.982	3.008

Omnibus:	2.321	**Durbin-Watson:**	1.952
Prob(Omnibus):	0.313	**Jarque-Bera (JB):**	2.379
Skew:	0.114	**Prob(JB):**	0.304
Kurtosis:	2.929	**Cond. No.**	54.2

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

In [43]:

```
print("Intercept is ", model.params['Intercept'])
print("Slope1 is ", model.params['X1'])
print("Slope2 is ", model.params['X2'])
print("R^2 is ", model.rsquared )
```

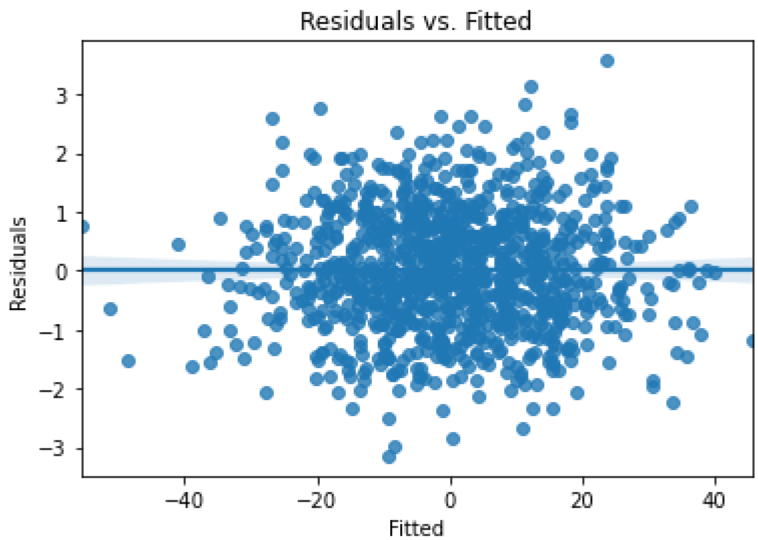
```
Intercept is  0.7529329159810596
Slope1 is  0.5240831009256066
```

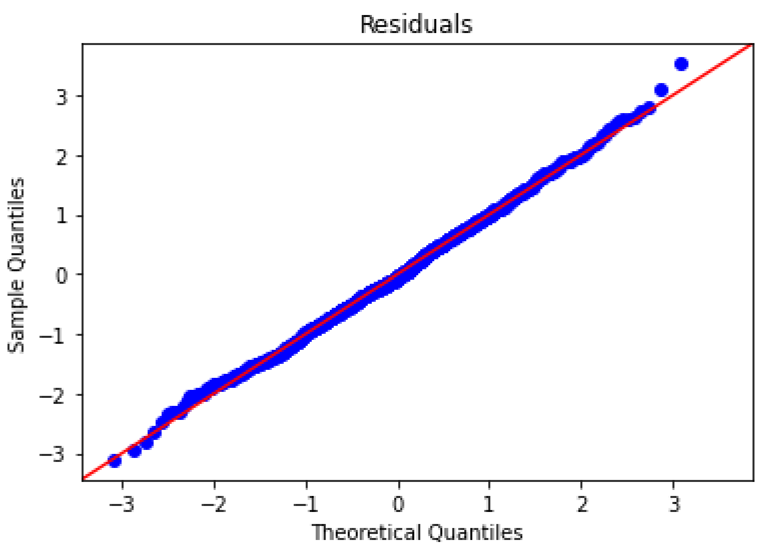
```
Slope2 is  2.994903282270229
R^2 is  0.995177582373531
```

From the model, we get the intercept 0.752 which is approximately equal to the given intercept 1 and we get the slope1 0.524 and slope2 2.995 which are also approximately equal to our given slopes 0.5 and 3 respectively. The R^2 values is 0.9951 which indicates that 99.51% of the variance of the outcome variable(Y) is explained by the predictor variables(X1 and X2 combinedly) which implies a good model to fit for the data.

Residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions

In [44]:
```
plot_lm_diag(model)
```





Homoscedasticity is present since constant variance assumptions is satisfied. The residuals is pretty much normally distributed. Linear model assumptions of normal error is also satisfied.

Correlated Predictors

Drawing 1000 samples of variables X1 and X2 from a multivariate normal with means <1,3>, variances of 1, and a covariance Cov(X1, X2) = 1

In [45]:
```
xs = rng.multivariate_normal([1, 3], [[1, 0.85], [0.85, 1]], 1000)
```

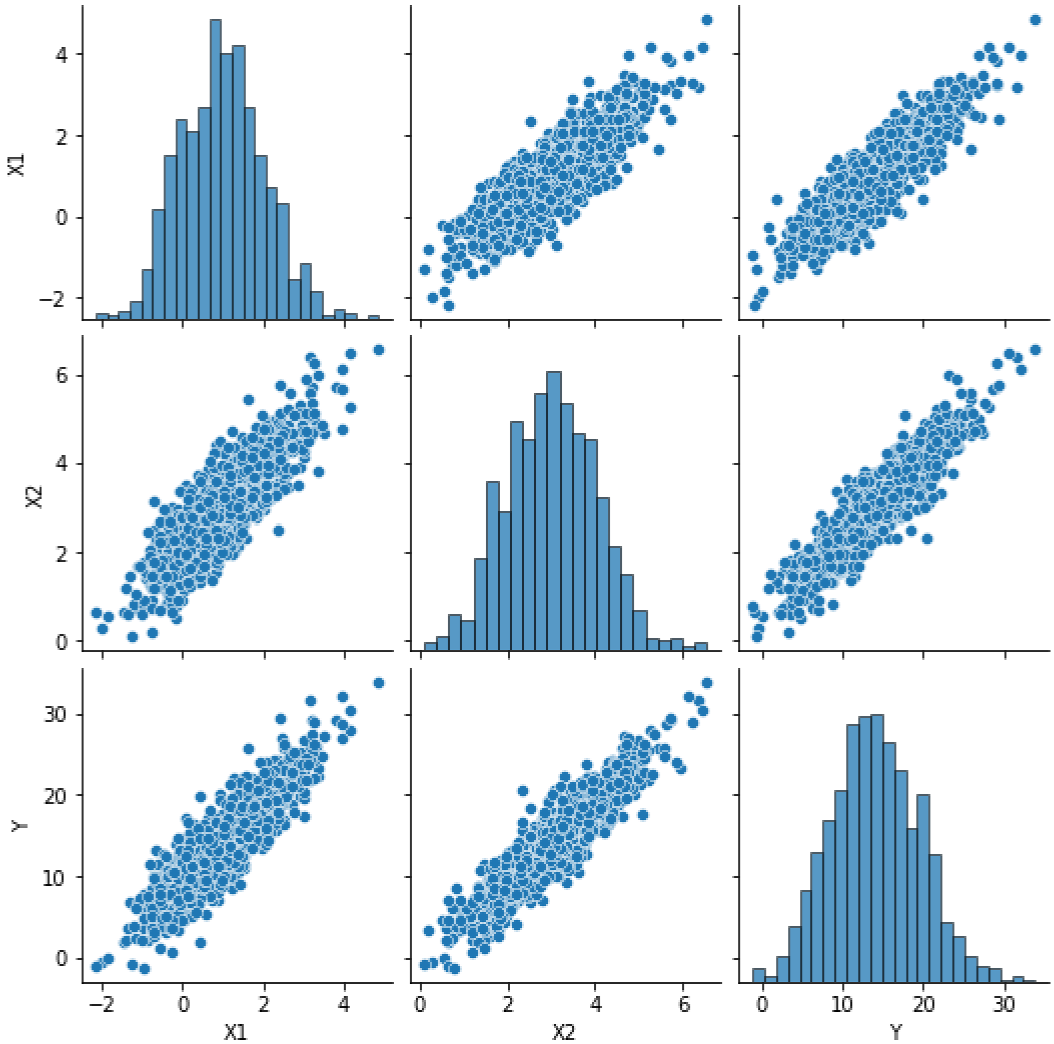
In [46]:
```
xs1 = xs[:, 0]
xs2 = xs[:, 1]
errs = np.random.normal(0, 2, 1000)
ys = 3 + 2*xs1 + 3*xs2 + errs
data = pd.DataFrame({
    'X1': xs1,
    'X2': xs2,
    'Y': ys
})
data.head()
```

Out[46]:

	X1	X2	Y
0	-0.411783	1.125076	4.959317
1	1.235898	3.516311	13.242781
2	1.164601	2.736442	11.966025
3	0.897190	3.102262	12.536747
4	0.416133	2.729115	11.133105

Showing a pairplot of our variables X1, X2 and Y

In [47]:
```
sns.pairplot(data[['X1', 'X2', 'Y']])
plt.show()
```



Each of the variable is normally distributed and dependent variable has linear relationship with independent variables.

In [48]:

```
model = smf.ols('Y ~ X1 + X2', data=data).fit()
model.summary()
```

Out[48]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.876
Model:	OLS	**Adj. R-squared:**	0.875
Method:	Least Squares	**F-statistic:**	3508.
Date:	Sun, 24 Oct 2021	**Prob (F-statistic):**	0.00
Time:	20:56:58	**Log-Likelihood:**	-2093.9
No. Observations:	1000	**AIC:**	4194.
Df Residuals:	997	**BIC:**	4209.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	2.5202	0.263	9.574	0.000	2.004	3.037

X1	2.0045	0.117	17.115	0.000	1.775	2.234
X2	3.1672	0.115	27.453	0.000	2.941	3.394

Omnibus:	0.129	**Durbin-Watson:**	2.049
Prob(Omnibus):	0.938	**Jarque-Bera (JB):**	0.062
Skew:	-0.004	**Prob(JB):**	0.970
Kurtosis:	3.038	**Cond. No.**	17.0

Notes:

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

In [49]:

```
print("Intercept is ", model.params['Intercept'])
print("Slope is ", model.params['X1'])
print("Slope is ", model.params['X2'])
print("R^2 is ", model.rsquared )
```

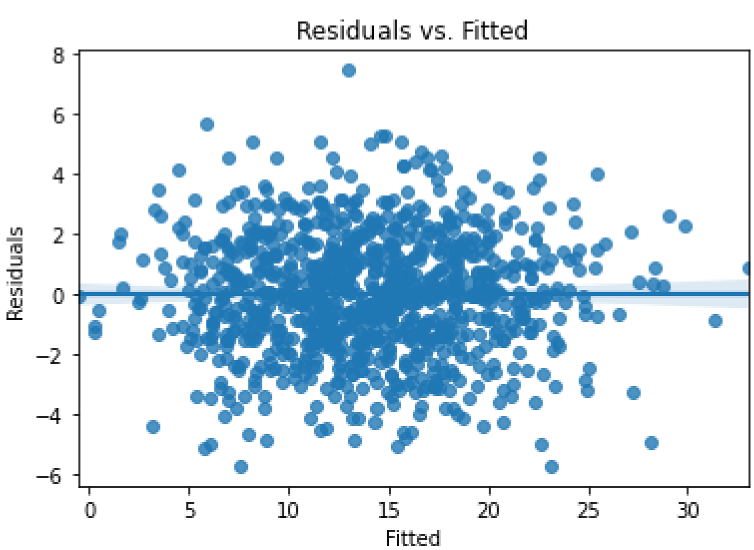
```
Intercept is  2.52020485514411
Slope is  2.0045071007860606
Slope is  3.1672440871106127
R^2 is  0.8755920994517906
```

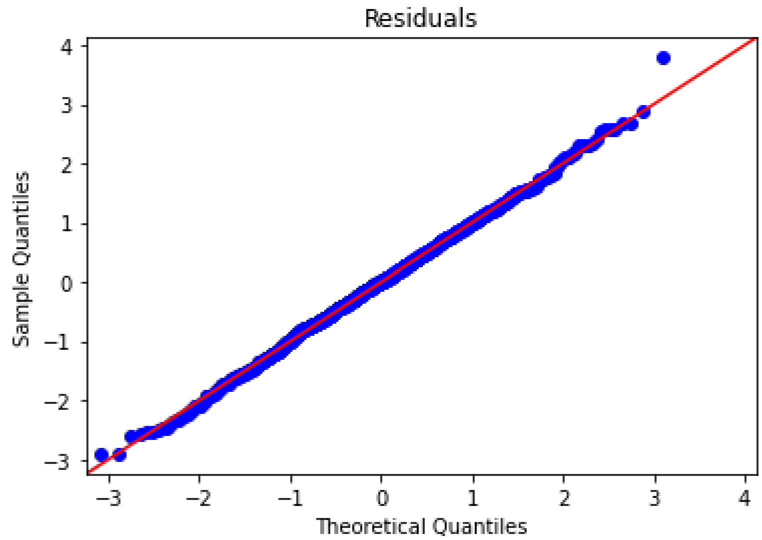
From the model, we get the intercept 2.52 which is approximately equal to the given intercept 3 and we get the slope1 2.004 and slope2 3.167 which are also approximately equal to our given slopes 2 and 3 respectively. The R^2 values is 0.8755 which indicates that 87.55% of the variance of the outcome variable(Y) is explained by the predictor variables(X1 and X2 combinedly) which implies a good model to fit for the data.

Residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions

In [50]:

```
plot_lm_diag(model)
```





From the above figures, we see that there is linear relationship between the outcome variable and the independent variables. Variance are not scattered so much that is why Homoscedasticity is present. The residuals is pretty much normally distributed.

In [51]:

```
intercepts = np.zeros(100)
slope1 = np.zeros(100)
slope2 = np.zeros(100)
for i in range(100):
    xs = rng.multivariate_normal([1, 3], [[1, 0.85], [0.85, 1]], 1000)
    xs1 = xs[:, 0]
    xs2 = xs[:, 1]
    errs = np.random.normal(0, 2, 1000)
    ys = 3 + 2*xs1 + 3*xs2 + errs
    data = pd.DataFrame({
    'X1': xs1,
    'X2': xs2,
    'Y': ys
    })
    model = smf.ols('Y ~ X1 + X2', data=data).fit()
    intercepts[i] = model.params['Intercept']
    slope1[i] = model.params['X1']
    slope2[i] = model.params['X2']
print("Intercept->", "Mean: " + str(np.mean(intercepts)), "and Variance: " + str
print("Slope1->", "Mean: " + str(np.mean(slopes)), "and Variance: " + str(np.var
print("Slope2->", "Mean: " + str(np.mean(rsq)), " and Variance: " + str(np.var(s
```

```
Intercept-> Mean: 2.9893452721138054 and Variance: 0.06608658998047688
Slope1-> Mean: 3.999547615593469 and Variance: 0.014157376439777719
Slope2-> Mean: 0.941084339009865  and Variance: 0.012646188139296579
```

Repeating the above simulation (drawing 1000 variables and fitting a linear model) 100 times. Showing the mean, variance, and appropriate distribution plots of the estimated intercepts and coefficients (for x1 and x2).

In [52]:

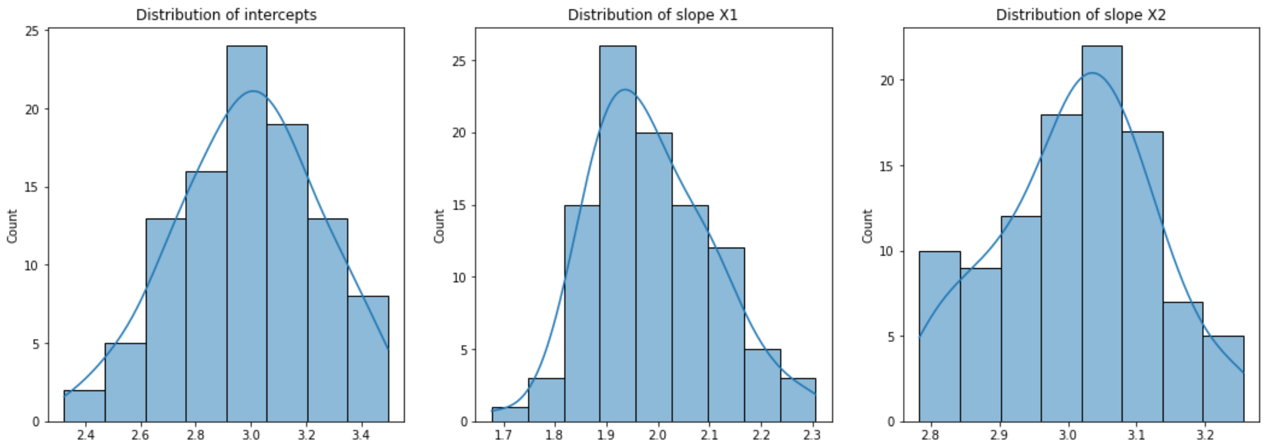
```
plt.figure(figsize=(18,6))

plt.subplot(1,3,1)
sns.histplot(intercepts, kde=True)
plt.title("Distribution of intercepts")
```

```
plt.subplot(1,3,2)
sns.histplot(slope1, kde=True)
plt.title("Distribution of slope X1")


plt.subplot(1,3,3)
sns.histplot(slope2, kde=True)
plt.title("Distribution of slope X2")


plt.show()
```



Repeating the repeated simulation for a variety of different covariances from 0 to 1 (including at least 0, 0.9, 0.99, and 0.999)

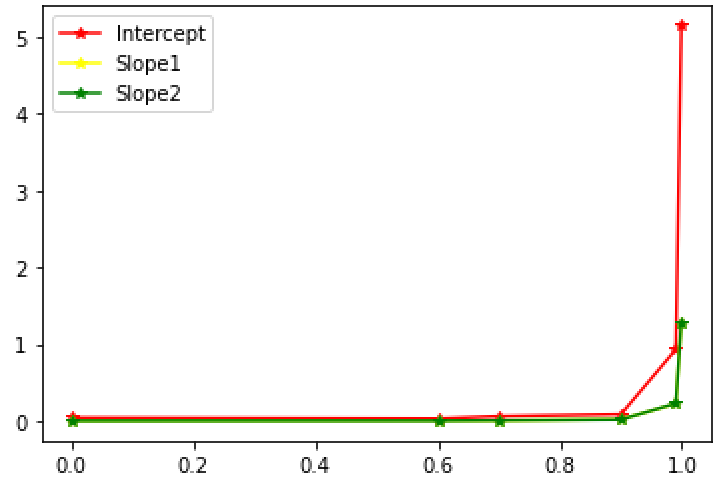
In [53]:

```
var_list = [0, 0.6, 0.7, 0.9, 0.99, 0.999]
length = len(var_list)
var_intercepts = np.zeros(length)
var_slope1 = np.zeros(length)
var_slope2 = np.zeros(length)
for i in range (length):
    intercepts, slopes1, slopes2 = np.zeros(100), np.zeros(100), np.zeros(100)
    for j in range(100):
        xs = rng.multivariate_normal([1, 3], [[1, var_list[i]], [var_list[i], 1]
        xs1 = xs[:, 0]
        xs2 = xs[:, 1]
        errs = np.random.normal(0, 2, 1000)
        ys = 3 + 2*xs1 + 3*xs2 + errs
        data = pd.DataFrame({
        'X1': xs1,
        'X2': xs2,
        'Y': ys
        })
        model = smf.ols('Y ~ X1 + X2', data=data).fit()
        intercepts[j] = model.params['Intercept']
        slopes1[j] = model.params['X1']
        slopes2[j] = model.params['X2']
    var_intercepts[i] = np.var(intercepts)
    var_slope1[i] = np.var(slopes1)
    var_slope2[i] = np.var(slopes2)
```

Creating line plots that show how the variance of the estimated regression parameters change with the increase of the correlation (covariance) between X1 and X2

In [54]:
```
plt.plot(var_list, var_intercepts, '*-' , label = "Intercept", color = "red")
plt.plot(var_list, var_slope1, '*-', label = "Slope1", color = "yellow")
plt.plot(var_list, var_slope2, '*-', label = "Slope2", color = "green")
plt.legend()
```

Out[54]: <matplotlib.legend.Legend at 0x7fe3f7bf5d60>



From the above figure, we can say that with the increase of covarience, it will also increase the intercept as well as the slopes. And at the maximum limit of covariance which is 1, intecept and solpes increases exponentially.

Reflection

In this assignment, we get to experience linear regression and simulation to study the behavior of statistical techniques. We use synthetic data instead of real data in this assignment. And we tried to fit those data into a defined model and got to know the simulation process. While creating the data, we followed normal, gamma distribution, linear, nonlinear data based on given slope, intercept, and standard errors. Also, we take the different number of data points for the same number of iterations and try to observe the correlation of correlated and independent variables.

We use linear-single variable regression, multiple regression in this assignment. We also use nonlinear, non-normal covariate data in this assignment. Given x and errors, we calculate y using given equations and find whether the model properly fits or not for the data. Then we tried to fit data to a linear model. We used linear, nonlinear data, correlated data, plot residuals, and Q-Q plot and checked if the model fulfills the assumptions of the linear model, normal distribution, and scedasticity (homoscedasticity or heteroscedasticity). So, the linear model's assumption gets clearer, and we get to know how to interpret the idea of equal variance and normal errors from the representations. Lastly, we used different covariances for our outcome and predictor variables and checked how they changed when the covariance changed. We find that, with the increase of covariance, the variance of intercept and coefficient increase.