In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


# from google.colab import files
# uploaded=files.upload()
# import io
```

In [2]:

```
# data=pd.read_csv(io.StringIO(uploaded['gene_data.csv'].decode('utf-8')),header=None)
# data.head()

data= pd.read_csv('gene_data.csv', header=None, sep='\t')
data.head()
```

Out[2]:

	0	1	2	3	4	5
0	0.0	1.050778	0.565836	0.970966	0.564797	0.482205
1	0.1	0.927415	0.314328	0.926647	0.644547	0.622073
2	0.2	0.920302	0.322418	0.912583	0.807286	0.680840
3	0.3	0.780651	0.456490	0.868285	0.882913	0.877914
4	0.4	0.804124	0.586241	0.915385	0.955537	0.921647

In [3]:

```
data.columns= [ 'time', 'x1', 'x2', 'x3','x4','x5']
data.head(2)
```

Out[3]:

	time	x1	x2	x3	x4	x5
0	0.0	1.050778	0.565836	0.970966	0.564797	0.482205
1	0.1	0.927415	0.314328	0.926647	0.644547	0.622073

In [4]:

```
print(data.shape)
print(data.isnull().sum())
```

```
(301, 6)
time    0
x1      0
x2      0
x3      0
x4      0
x5      0
dtype: int64
```

Task 1: Preliminary data analysis

You should first perform an initial exploratory data analysis, by investigating:

• Time series plots (of each gene against sampling time)

• Distribution for each gene (time-series)

• Correlation and scatter plots (between different combination of two genes) to examine their dependencies

In [5]:

```
data1= data.copy()
data1.head(2)
```

Out[5]:

	time	x1	x2	x3	x4	x5
0	0.0	1.050778	0.565836	0.970966	0.564797	0.482205
1	0.1	0.927415	0.314328	0.926647	0.644547	0.622073

In [6]:

```
data1.set_index(data1['time'], inplace=True)
```

In [7]:

```
data1.head(2)
```

Out[7]:

time	time	x1	x2	x3	x4	x5
0.0	0.0	1.050778	0.565836	0.970966	0.564797	0.482205
0.1	0.1	0.927415	0.314328	0.926647	0.644547	0.622073

In [8]:

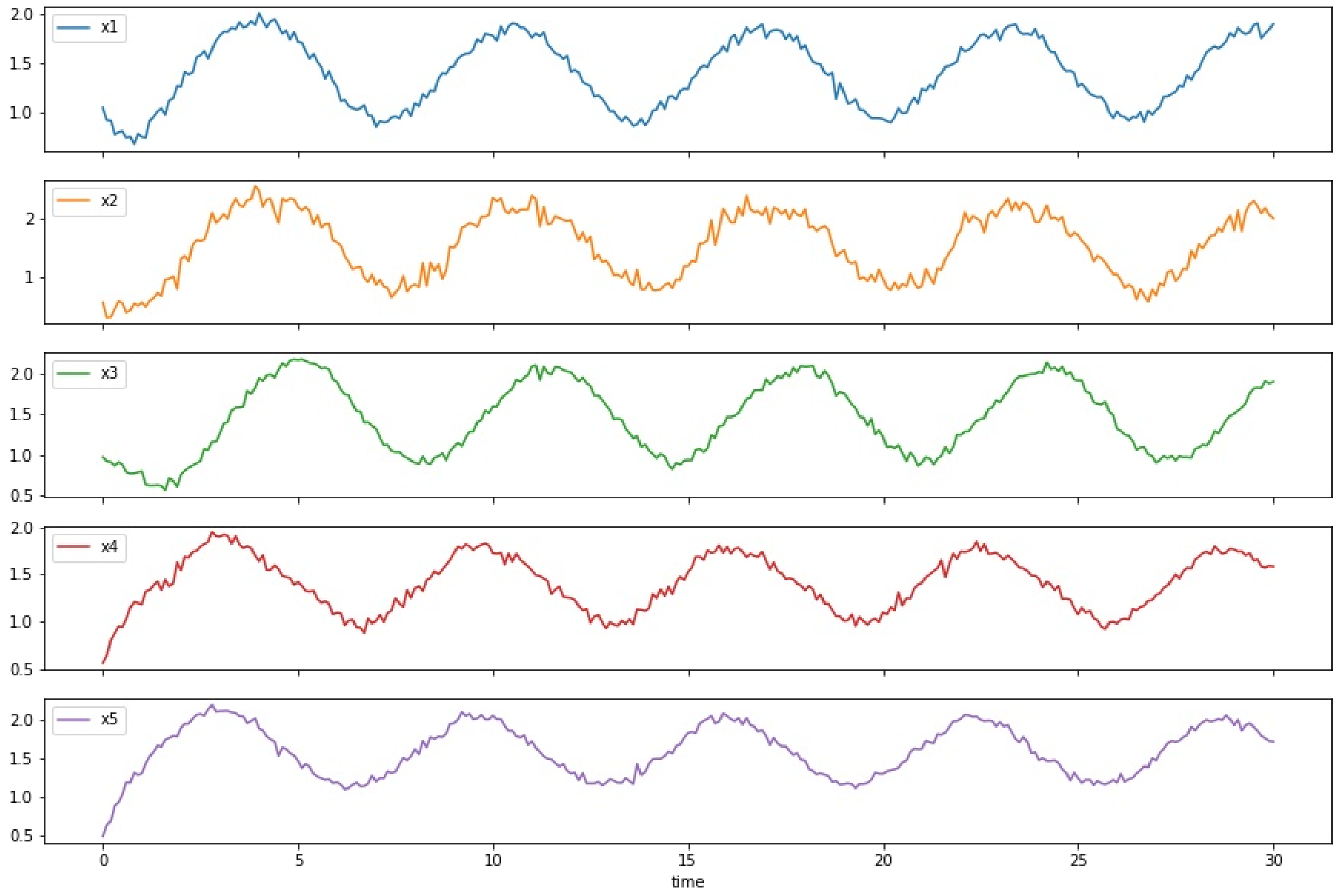
```
data1.describe()
```

Out[8]:

	time	x1	x2	x3	x4	x5
count	301.000000	301.000000	301.000000	301.000000	301.000000	301.000000
mean	15.000000	1.408540	1.523716	1.447331	1.410121	1.595587
std	8.703543	0.350848	0.563047	0.439728	0.290399	0.335002
min	0.000000	0.681297	0.314328	0.569043	0.564797	0.482205
25%	7.500000	1.075980	0.994929	1.036931	1.156419	1.301543
50%	15.000000	1.424517	1.586479	1.446317	1.424729	1.617133
75%	22.500000	1.756039	2.041698	1.880349	1.683586	1.912497
max	30.000000	2.011843	2.540164	2.173947	1.946802	2.190523

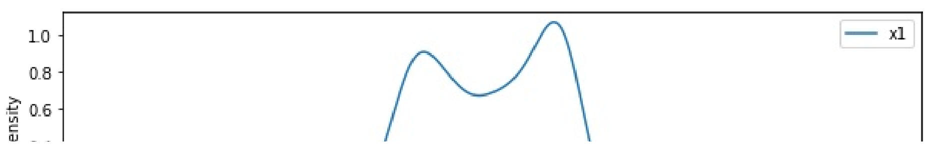
In [9]:

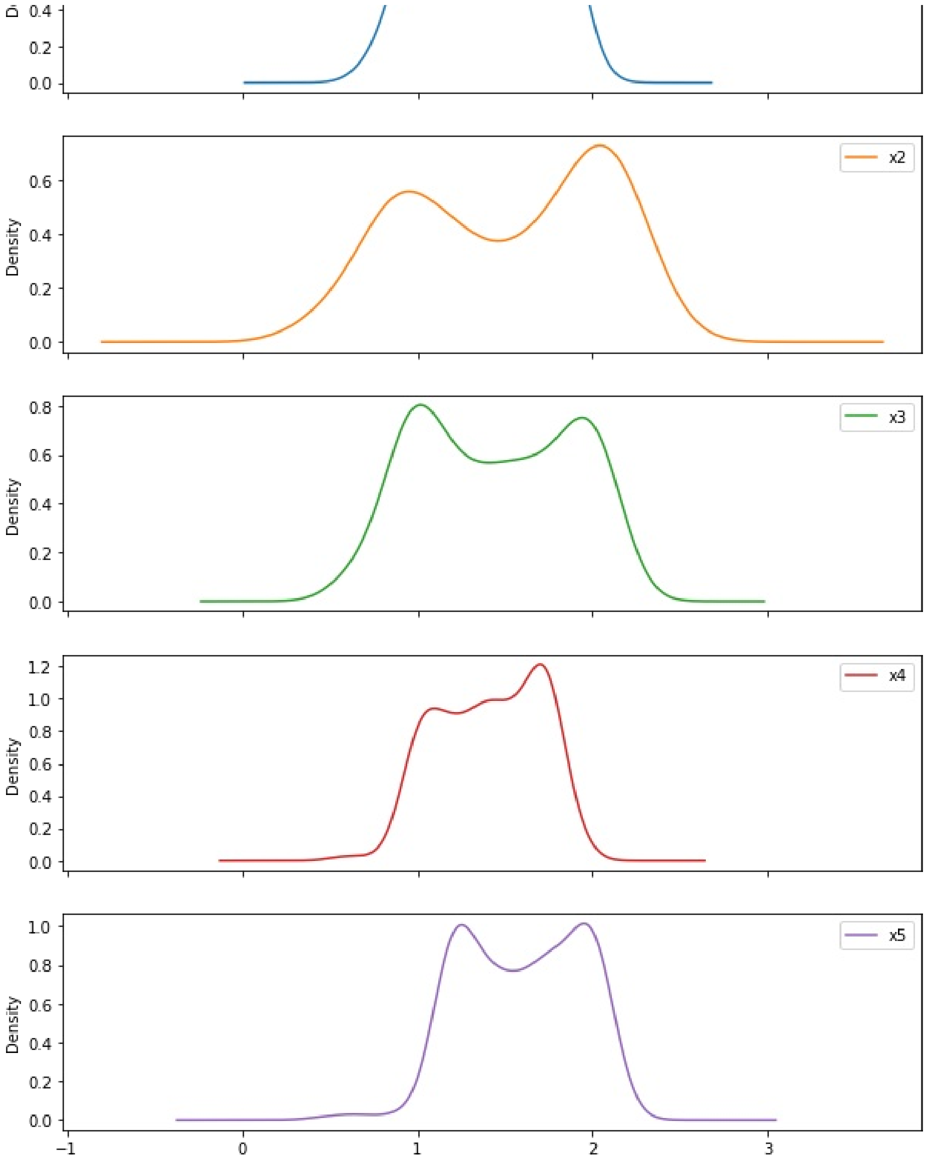
```
#Time series plots (of each gene against sampling time)
data1.drop('time',axis=1).plot(subplots=True, figsize=(15,10))
plt.show()
```



In [10]:

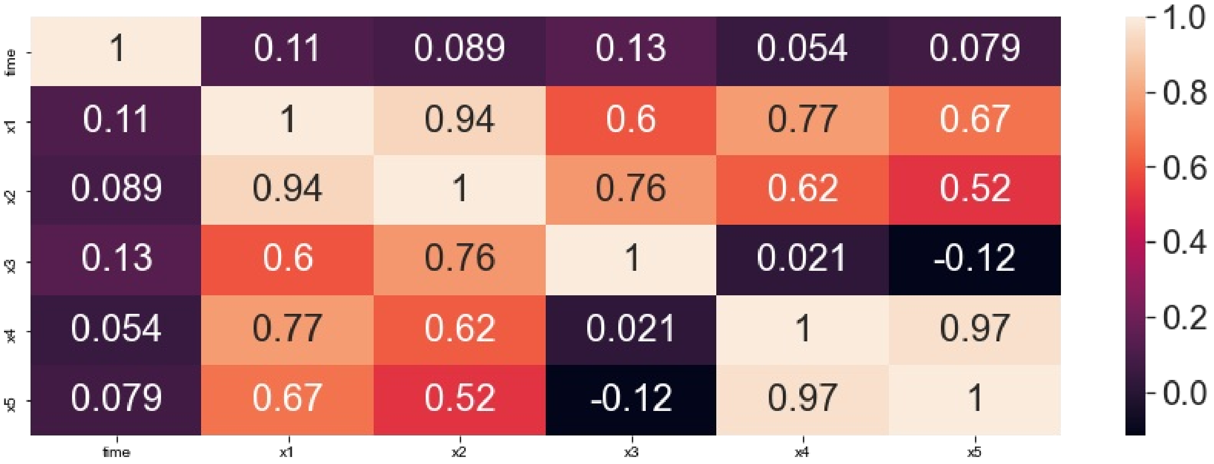
```
#Distribution for each gene (time-series)
data1.drop('time',axis=1).plot(subplots=True, kind='kde',figsize=(10,15))
plt.show()
```





In [11]:

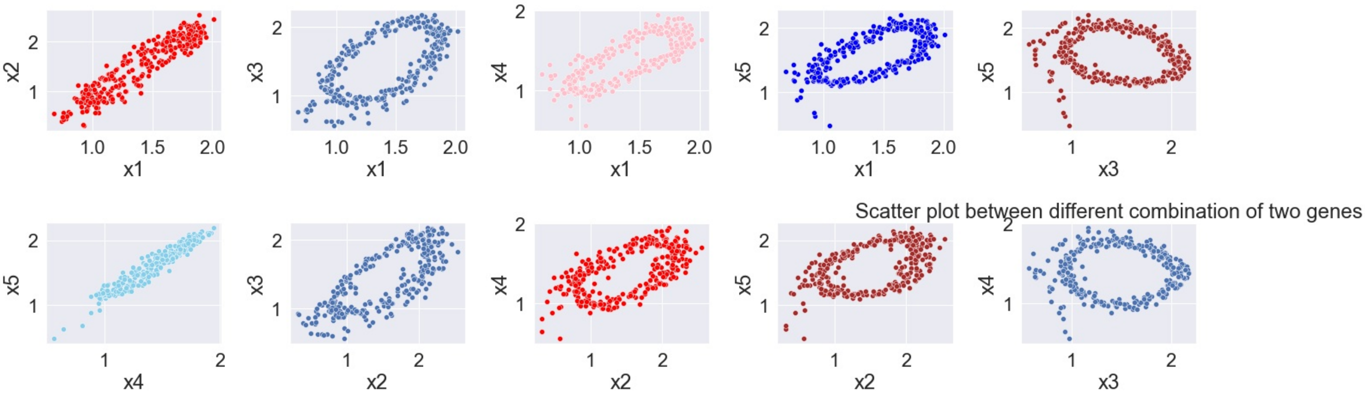
```
#Correlation and scatter plots (between different combination of two genes) to examine their dependencies

##Correlation plot
plt.subplots(figsize=(15,5))
sns.set(font_scale=2)
sns.heatmap(data1.corr(), annot=True)
plt.show()
```



In [12]:

```
## Scatter plot
fig, axs = plt.subplots(nrows = 2, ncols = 5, figsize = (20, 7))
plt.title('Scatter plot between different combination of two genes')

sns.scatterplot(data1['x1'],data1['x2'],ax=axs[0,0],color='red')
sns.scatterplot(data1['x1'],data1['x3'],ax=axs[0,1])
sns.scatterplot(data1['x1'],data1['x4'],ax=axs[0,2],color='pink')
```

```
sns.scatterplot(data1['x1'],data1['x5'],ax=axs[0,3],color='blue')
sns.scatterplot(data1['x3'],data1['x5'],ax=axs[0,4],color='brown')

sns.scatterplot(data1['x4'],data1['x5'],ax=axs[1,0],color='skyblue')
sns.scatterplot(data1['x2'],data1['x3'],ax=axs[1,1])
sns.scatterplot(data1['x2'],data1['x4'],ax=axs[1,2],color='red')
sns.scatterplot(data1['x2'],data1['x5'],ax=axs[1,3],color='brown')

sns.scatterplot(data1['x3'],data1['x4'],ax=axs[1,4])


plt.title('Scatter plot between different combination of two genes')
plt.subplots_adjust(wspace=1)
plt.tight_layout()
plt.show()
```



In [ ]:

Task 2: Regression – modelling the relationship between gene expression We would like to determine a suitable mathematical model in explaining the relationship between the output gene $\mathbf{y}$ = $\mathbf{x}$2 with other input genes (i.e. $\mathbf{x}$1, $\mathbf{x}$3, $\mathbf{x}$4, $\mathbf{x}$5) that actually 'regulate' its expression, which we assume can be described by a polynomial regression model. Below are 5 candidate nonlinear polynomial regression models, and only one of them can 'truly' describe such a relationship. The objective is to identify this 'true' model from those candidate models following Tasks 2.1 – 2.6.

Candidate models are with the following structures:

Model 1: $y = \theta 1\ x4 + \theta 2\ x3\ 2 + \theta bias$

Model 2: $y = \theta 1\ x4 + \theta 2\ x3\ 2 + \theta 3\ x5 + \theta bias$

Model 3: $y = \theta 1\ x3 + \theta 2\ x4 + \theta 3\ x5\ 3$

Model 4: $y = \theta 1\ x4 + \theta 2\ x3\ 2 + \theta 3\ x5\ 3 + \theta bias$

Model 5: $y = \theta 1\ x4 + \theta 2\ x1\ 2 + \theta 3\ x3\ 2 + \theta bias$

Task 2.1: Estimate model parameters $\boldsymbol{\theta} = \{\theta 1, \theta 2, \cdots, \theta bias\}\ T$

for every candidate model using Least Squares ($\boldsymbol{\theta}$ = ($\mathbf{X}\ T\mathbf{X}$) −1$\mathbf{X}\ T\mathbf{y}$), using the provided input and output gene datasets (use all the data for training).

Task 2.2: Based on the estimated model parameters, compute the model residual (error) sum of squared errors (RSS), for every candidate model. $RSS = \sum (yi - \mathbf{x}i\boldsymbol{\theta})\ 2\ n\ i{=}1$

Here $\mathbf{x}i$ denotes the $i\ th$ row ($i\ th$ data sample) in the input data matrix $\mathbf{X}$, $\boldsymbol{\theta}$ is a column vector.

Task 2.3: Compute the log-likelihood function for every candidate model: ln $p(D|\boldsymbol{\theta}) = -\ n\ 2$ ln$(2\pi) - n\ 2$ ln$(\hat{\sigma}\ 2\ ) - 1\ 2\hat{\sigma}\ 2$ RSS

Here, $\hat{\sigma}\ 2$ is the variance of a model's residuals (prediction errors) distributions $\hat{\sigma}\ 2$ = RSS/(n − 1) , with n the number of data samples. D denotes the input-output dataset $\{\mathbf{X}, \mathbf{y}\}$.

Task 2.4: Compute the Akaike information criterion (AIC) and Bayesian information criterion (BIC) for every candidate model: $AIC = 2k - 2$ ln $p(D|\boldsymbol{\theta})$

$BIC = k \cdot$ ln $(n) - 2$ ln $p(D|\boldsymbol{\theta})$

Here ln $p(D|\boldsymbol{\theta})$ is the log-likelihood function obtained from Task 2.3 for each model, k is the number of estimated parameters in each candidate model.

Task 2.1

In [13]:

```
import statsmodels.formula.api as smf
model1= smf.ols('time ~ x4+ np.square(x3)', data=data).fit()
```

In [14]:

```
model1.summary()
```

Out[14]:

OLS Regression Results

Dep. Variable:	time	**R-squared:**	0.011
Model:	OLS	**Adj. R-squared:**	0.004
Method:	Least Squares	**F-statistic:**	1.667
Date:	Wed, 20 Oct 2021	**Prob (F-statistic):**	0.191
Time:	23:07:45	**Log-Likelihood:**	-1076.2
No. Observations:	301	**AIC:**	2158.
Df Residuals:	298	**BIC:**	2170.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	11.3492	2.633	4.310	0.000	6.167	16.532
x4	1.5947	1.727	0.924	0.356	-1.803	4.993
np.square(x3)	0.6129	0.392	1.565	0.119	-0.158	1.384

Omnibus:	221.349	**Durbin-Watson:**	0.000
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	19.524
Skew:	0.035	**Prob(JB):**	5.76e-05
Kurtosis:	1.754	**Cond. No.**	18.9

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

In [15]:

```
model2= smf.ols('time ~ x4+ np.square(x3)+x5', data=data).fit()
```

In [16]:

```
model2.summary()
```

Out[16]:

OLS Regression Results

Dep. Variable:	time	**R-squared:**	0.047
Model:	OLS	**Adj. R-squared:**	0.038
Method:	Least Squares	**F-statistic:**	4.913
Date:	Wed, 20 Oct 2021	**Prob (F-statistic):**	0.00240
Time:	23:07:45	**Log-Likelihood:**	-1070.6
No. Observations:	301	**AIC:**	2149.
Df Residuals:	297	**BIC:**	2164.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	8.8400	2.695	3.281	0.001	3.537	14.143
x4	-24.5754	7.972	-3.083	0.002	-40.264	-8.887
np.square(x3)	1.4997	0.467	3.213	0.001	0.581	2.418
x5	23.4295	6.973	3.360	0.001	9.706	37.153

Omnibus:	152.674	**Durbin-Watson:**	0.053
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	18.068
Skew:	0.080	**Prob(JB):**	0.000119
Kurtosis:	1.811	**Cond. No.**	73.2

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

In [17]:
```
model2.cov_params()
```

Out[17]:

	Intercept	x4	np.square(x3)	x5
Intercept	7.261084	1.769022	-0.526391	-5.207689
x4	1.769022	63.549482	-2.062583	-54.314436
np.square(x3)	-0.526391	-2.062583	0.217880	1.840380
x5	-5.207689	-54.314436	1.840380	48.626507

In [18]:
```
model3= smf.ols('time ~ x3+x4+ I(x5**3.0)', data=data).fit()
```

In [19]:
```
model3.summary()
```

Out[19]:

OLS Regression Results

Dep. Variable:	time	**R-squared:**	0.021
Model:	OLS	**Adj. R-squared:**	0.011
Method:	Least Squares	**F-statistic:**	2.108
Date:	Wed, 20 Oct 2021	**Prob (F-statistic):**	0.0993
Time:	23:07:46	**Log-Likelihood:**	-1074.7
No. Observations:	301	**AIC:**	2157.
Df Residuals:	297	**BIC:**	2172.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9.0460	5.187	1.744	0.082	-1.162	19.254
x3	2.6576	1.321	2.011	0.045	0.057	5.258
x4	1.4676	6.294	0.233	0.816	-10.919	13.854
I(x5 ** 3.0)	0.0083	0.718	0.012	0.991	-1.404	1.420

Omnibus:	219.232	**Durbin-Watson:**	0.001
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	19.609
Skew:	0.062	**Prob(JB):**	5.52e-05
Kurtosis:	1.756	**Cond. No.**	91.0

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

In [20]:
```
model4= smf.ols('time ~ x4+I(x3**2.0)+ I(x5**3.0)', data=data).fit()
```

In [21]:
```
model4.summary()
```

Out[21]:

OLS Regression Results

Dep. Variable:	time	**R-squared:**	0.011
Model:	OLS	**Adj. R-squared:**	0.001
Method:	Least Squares	**F-statistic:**	1.147
Date:	Wed, 20 Oct 2021	**Prob (F-statistic):**	0.330
Time:	23:07:46	**Log-Likelihood:**	-1076.1
No. Observations:	301	**AIC:**	2160.
Df Residuals:	297	**BIC:**	2175.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9.6809	5.550	1.744	0.082	-1.241	20.603
x4	3.7688	6.595	0.572	0.568	-9.209	16.747
I(x3 ** 2.0)	0.5198	0.478	1.088	0.277	-0.420	1.460
I(x5 ** 3.0)	-0.2581	0.755	-0.342	0.733	-1.745	1.229

Omnibus:	218.337	**Durbin-Watson:**	0.001
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	19.462
Skew:	0.034	**Prob(JB):**	5.94e-05
Kurtosis:	1.756	**Cond. No.**	100.

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

In [22]:
```
model5= smf.ols('time ~ x4+I(x1**2.0)+ I(x3**2.0)', data=data).fit()
```

In [23]:
```
model5.summary()
```

Out[23]:

OLS Regression Results

Dep. Variable:	time	**R-squared:**	0.011
Model:	OLS	**Adj. R-squared:**	0.001
Method:	Least Squares	**F-statistic:**	1.116
Date:	Wed, 20 Oct 2021	**Prob (F-statistic):**	0.343
Time:	23:07:46	**Log-Likelihood:**	-1076.2
No. Observations:	301	**AIC:**	2160.
Df Residuals:	297	**BIC:**	2175.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	10.6192	5.340	1.989	0.048	0.111	21.128
x4	2.3379	5.033	0.464	0.643	-7.567	12.243
I(x1 ** 2.0)	-0.2905	1.848	-0.157	0.875	-3.927	3.346
I(x3 ** 2.0)	0.7415	0.907	0.818	0.414	-1.043	2.526

Omnibus:	222.321	**Durbin-Watson:**	0.001
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	19.557
Skew:	0.039	**Prob(JB):**	5.67e-05
Kurtosis:	1.754	**Cond. No.**	56.9

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

In [24]:
```
print('Estimated parameters of model1 \n',model1.params)
print('\n')
print('Estimated parameters of model2 \n',model2.params)
print('\n')
print('Estimated parameters of model3 \n',model3.params)
print('\n')
print('Estimated parameters of model4 \n',model4.params)
print('\n')
print('Estimated parameters of model5 \n',model5.params)
```

```
Estimated parameters of model1
 Intercept        11.349158
x4               1.594710
np.square(x3)    0.612947
dtype: float64
```

```
Estimated parameters of model2
 Intercept         8.839955
x4              -24.575429
np.square(x3)     1.499691
x5               23.429543
dtype: float64


Estimated parameters of model3
 Intercept       9.045972
x3              2.657624
x4              1.467639
I(x5 ** 3.0)    0.008285
dtype: float64


Estimated parameters of model4
 Intercept       9.680892
x4              3.768793
I(x3 ** 2.0)    0.519813
I(x5 ** 3.0)   -0.258102
dtype: float64


Estimated parameters of model5
 Intercept       10.619207
x4               2.337869
I(x1 ** 2.0)    -0.290515
I(x3 ** 2.0)     0.741484
dtype: float64
```

In [25]:

```
# Task 2.2: Residual sum of square is also known as explained sum of square

print('Residual sum of square for model1: ',round(model1.ess,4))
print('Residual sum of square for model2: ',round(model2.ess,4))
print('Residual sum of square for model3: ',round(model3.ess,4))
print('Residual sum of square for model4: ',round(model4.ess,4))
print('Residual sum of square for model5: ',round(model5.ess,4))
```

```
Residual sum of square for model1:  251.4469
Residual sum of square for model2:  1074.4055
Residual sum of square for model3:  473.8122
Residual sum of square for model4:  260.2752
Residual sum of square for model5:  253.3173
```

In [26]:

```
## Task 2.3: Log Likelihood of models
print('Log Likelihood value for model1: ',round(model1.llf,4))
print('Log Likelihood value for model2: ',round(model2.llf,4))
print('Log Likelihood value for model3: ',round(model3.llf,4))
print('Log Likelihood value for model4: ',round(model4.llf,4))
print('Log Likelihood value for model5: ',round(model5.llf,4))
```

```
Log Likelihood value for model1:  -1076.208
Log Likelihood value for model2:  -1070.5935
Log Likelihood value for model3:  -1074.7115
Log Likelihood value for model4:  -1076.1488
Log Likelihood value for model5:  -1076.1954
```

In [27]:

```
#Task 2.4: AIC and BIC
print('AIC value for model1: ',round(model1.aic,4))
print('AIC value for model2: ',round(model2.aic,4))
print('AIC value for model3: ',round(model3.aic,4))
print('AIC value for model4: ',round(model4.aic,4))
print('AIC value for model5: ',round(model5.aic,4))
```

```
AIC value for model1:  2158.4159
AIC value for model2:  2149.187
AIC value for model3:  2157.4229
AIC value for model4:  2160.2977
AIC value for model5:  2160.3909
```

In [28]:

```
# BIC
print('BIC value for model1: ',round(model1.bic,4))
print('BIC value for model2: ',round(model2.bic,4))
print('BIC value for model3: ',round(model3.bic,4))
print('BIC value for model4: ',round(model4.bic,4))
print('BIC value for model5: ',round(model5.bic,4))
```

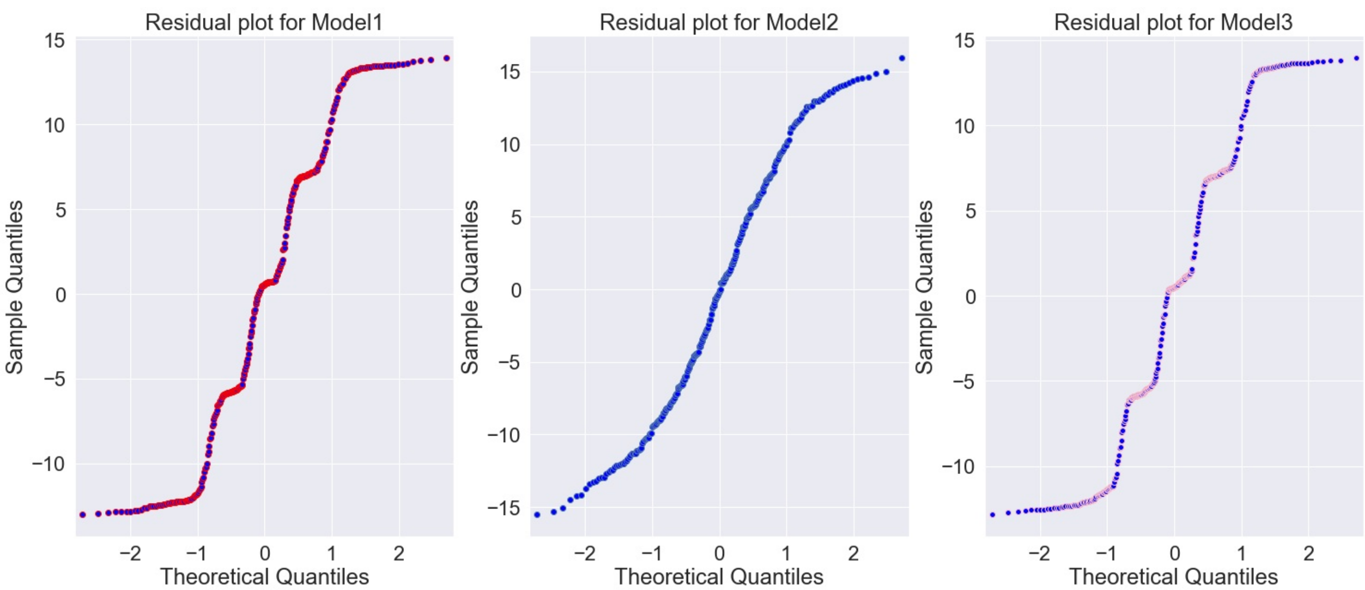
```
BIC value for model1:  2169.5373
BIC value for model2:  2164.0154
BIC value for model3:  2172.2514
BIC value for model4:  2175.1261
BIC value for model5:  2175.2193
```

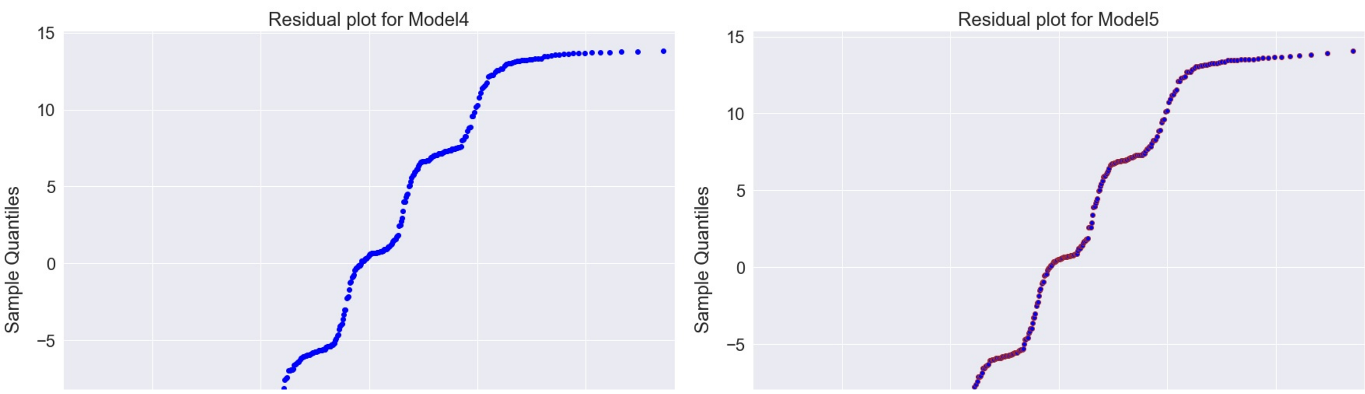
Task 2.5: Check the distribution of model prediction errors (residuals) for each candidate model. Plot the error distributions, and evaluate if those distributions are close to Normal/Gaussian (as the output gene has additive Gaussian noise), e.g. by using Q-Q plot.

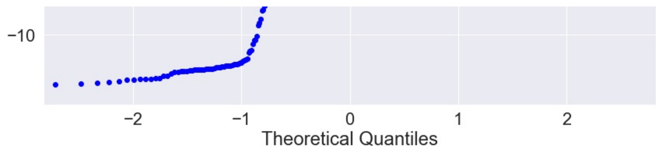
Task 2.6: Select 'best' regression model according to the AIC, BIC and distribution of model residuals from the 5 candidate models, and explain why you would like to choose this specific model.

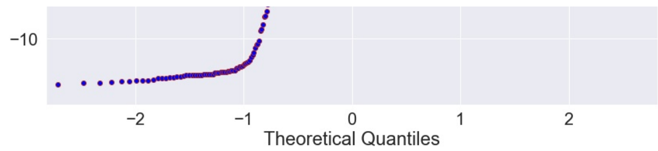
In [29]:

```
# Task 2.5: Residual plot by using QQplot.
import statsmodels.api as sm
fig, (ax1,ax2,ax3) = plt.subplots(nrows=1, ncols=3,figsize=(25,10))

sm.qqplot(model1.resid ,ax=ax1,color='red')
ax1.set_title("Residual plot for Model1")

sm.qqplot(model2.resid ,ax=ax2)
ax2.set_title("Residual plot for Model2")

sm.qqplot(model3.resid ,ax=ax3,color='pink')
ax3.set_title("Residual plot for Model3")

fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2,figsize=(25,10))

sm.qqplot(model4.resid ,ax=ax1,color='blue')
ax1.set_title("Residual plot for Model4")
sm.qqplot(model5.resid ,ax=ax2,color='brown')
ax2.set_title("Residual plot for Model5")


plt.subplots_adjust(wspace=1)
plt.tight_layout()
plt.show()
```









- Model 2 is close to Normal/ Gaussian distribution, so model-2 is the best model.

Task 2.7: Split the input and output gene dataset ($\mathbf{X}$ and $\mathbf{y}$) into two parts: one part used to train the model, the other used for testing (e.g. 70% for training, 30% for testing). For the selected 'best' model, 1) estimate model parameters use the training dataset; 2) compute the model's output/prediction on the testing data; and 3) also compute the 95% (model prediction) confidence intervals and plot them (with error bars) together with the model prediction, as well as the testing data samples.

In [30]:

```
#time ~ x4+I(x3**2.0)+ I(x5**3.0) ==> model-4

data['X3_2']= data.iloc[:,3]**2
data['X5_3']= data.iloc[:,5]**3
data.head()
```

Out[30]:

	time	x1	x2	x3	x4	x5	X3_2	X5_3
0	0.0	1.050778	0.565836	0.970966	0.564797	0.482205	0.942775	0.112123
1	0.1	0.927415	0.314328	0.926647	0.644547	0.622073	0.858675	0.240727
2	0.2	0.920302	0.322418	0.912583	0.807286	0.680840	0.832809	0.315598
3	0.3	0.780651	0.456490	0.868285	0.882913	0.877914	0.753918	0.676637
4	0.4	0.804124	0.586241	0.915385	0.955537	0.921647	0.837930	0.782877

In [31]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data[['x4','X3_2','X5_3']],data['time'],test_size = 0.3, ranc
```

In [32]:

```
X_train = sm.add_constant(X_train)
model = sm.OLS(y_train,X_train)
results = model.fit()
```

In [33]:

```
results.summary()
```

Out[33]:

OLS Regression Results

Dep. Variable:	time	**R-squared:**	0.008
Model:	OLS	**Adj. R-squared:**	-0.006
Method:	Least Squares	**F-statistic:**	0.5560
Date:	Wed, 20 Oct 2021	**Prob (F-statistic):**	0.645
Time:	23:07:51	**Log-Likelihood:**	-751.71
No. Observations:	210	**AIC:**	1511.
Df Residuals:	206	**BIC:**	1525.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	12.7637	6.587	1.938	0.054	-0.223	25.750
x4	1.7571	7.899	0.222	0.824	-13.816	17.330
X3_2	0.5094	0.570	0.894	0.372	-0.614	1.633
X5_3	-0.2650	0.919	-0.288	0.773	-2.076	1.546

Omnibus:	120.352	**Durbin-Watson:**	2.004
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	13.328
Skew:	-0.027	**Prob(JB):**	0.00128
Kurtosis:	1.767	**Cond. No.**	97.8

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

In [34]:
```
X_test.shape
```

Out[34]:
```
(91, 3)
```

In [35]:
```
x_pred = np.linspace(X_test.min(), X_test.max(),91)

x_matrix = sm.add_constant(x_pred)
y_pred = results.predict(x_matrix)
```

In [36]:
```
y_pred
```

Out[36]:
```
array([14.30721256, 14.32487572, 14.34253887, 14.36020203, 14.37786518,
       14.39552834, 14.4131915 , 14.43085465, 14.44851781, 14.46618096,
       14.48384412, 14.50150727, 14.51917043, 14.53683359, 14.55449674,
       14.5721599 , 14.58982305, 14.60748621, 14.62514936, 14.64281252,
       14.66047567, 14.67813883, 14.69580199, 14.71346514, 14.7311283 ,
       14.74879145, 14.76645461, 14.78411776, 14.80178092, 14.81944408,
       14.83710723, 14.85477039, 14.87243354, 14.8900967 , 14.90775985,
       14.92542301, 14.94308617, 14.96074932, 14.97841248, 14.99607563,
       15.01373879, 15.03140194, 15.0490651 , 15.06672826, 15.08439141,
       15.10205457, 15.11971772, 15.13738088, 15.15504403, 15.17270719,
       15.19037035, 15.2080335 , 15.22569666, 15.24335981, 15.26102297,
       15.27868612, 15.29634928, 15.31401243, 15.33167559, 15.34933875,
       15.3670019 , 15.38466506, 15.40232821, 15.41999137, 15.43765452,
       15.45531768, 15.47298084, 15.49064399, 15.50830715, 15.5259703 ,
       15.54363346, 15.56129661, 15.57895977, 15.59662293, 15.61428608,
       15.63194924, 15.64961239, 15.66727555, 15.6849387 , 15.70260186,
       15.72026502, 15.73792817, 15.75559133, 15.77325448, 15.79091764,
       15.80858079, 15.82624395, 15.84390711, 15.86157026, 15.87923342,
       15.89689657])
```

In [37]:
```
err= y_test-y_pred
```

In [38]:
```
err.head()
```

Out[38]:
```
177     3.392787
289    14.575124
228     8.457461
198     5.439798
60     -8.377865
Name: time, dtype: float64
```

In [39]:
```
CI_95=results.conf_int(alpha=0.05)
CI_95
```

Out[39]:

	0	1
const	-0.222775	25.750153
x4	-13.815717	17.329831
X3_2	-0.613726	1.632518
X5_3	-2.075924	1.545993

In [40]:
```
err.size
```

Out[40]:
```
91
```

In [41]:
```
c=results.get_prediction(x_matrix)
```

In [42]:
```
CI=c.conf_int(0.05)
```

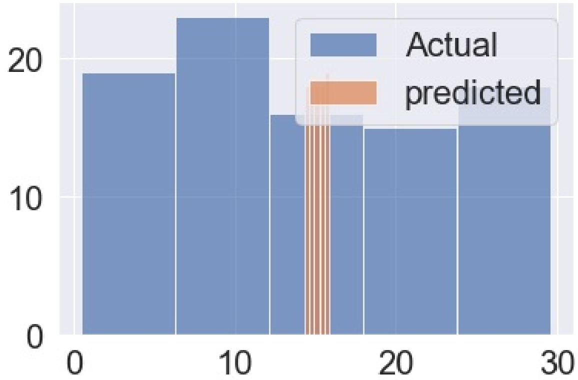
In [43]:
```
CI
```

Out[43]:
```
array([[-3.19551119, 31.80993632],
       [-3.16839289, 31.81814433],
       [-3.1415503 , 31.82662805],
       [-3.11498386, 31.83538791],
       [-3.08869398, 31.84442434],
       [-3.06268107, 31.85373775],
       [-3.03694554, 31.86332853],
       [-3.01148777, 31.87319708],
       [-2.98630814, 31.88334375],
       [-2.96140699, 31.89376891],
       [-2.93678468, 31.90447292],
       [-2.91244155, 31.91545609],
       [-2.88837791, 31.92671876],
       [-2.86459407, 31.93826124],
       [-2.84109033, 31.95008381],
       [-2.81786697, 31.96218676],
       [-2.79492426, 31.97457037],
       [-2.77226246, 31.98723487],
       [-2.7498818 , 32.00018053],
       [-2.72778252, 32.01340756],
       [-2.70596483, 32.02691618],
       [-2.68442894, 32.0407066 ],
       [-2.66317503, 32.054779  ],
       [-2.64220328, 32.06913356],
       [-2.62151385, 32.08377044],
       [-2.60110688, 32.09868979],
       [-2.58098252, 32.11389174],
       [-2.56114088, 32.12937641],
       [-2.54158206, 32.14514391],
       [-2.52230617, 32.16119432],
       [-2.50331328, 32.17752774],
       [-2.48460345, 32.19414423],
       [-2.46617674, 32.21104383],
       [-2.44803318, 32.22822658],
       [-2.4301728 , 32.24569251],
       [-2.41259561, 32.26344163],
       [-2.39530159, 32.28147392],
       [-2.37829075, 32.29978939],
       [-2.36156303, 32.31838799],
       [-2.3451184 , 32.33726967],
       [-2.3289568 , 32.35643438],
       [-2.31307816, 32.37588204],
       [-2.29748237, 32.39561257],
       [-2.28216936, 32.41562587],
       [-2.26713899, 32.43592181],
       [-2.25239115, 32.45650028],
       [-2.23792568, 32.47736113],
       [-2.22374244, 32.4985042 ],
       [-2.20984125, 32.51992932],
       [-2.19622193, 32.54163631],
       [-2.18288429, 32.56362498],
       [-2.16982811, 32.58589511],
       [-2.15705316, 32.60844648],
       [-2.14455922, 32.63127885],
       [-2.13234604, 32.65439197],
       [-2.12041333, 32.67778558],
       [-2.10876084, 32.7014594 ],
       [-2.09738827, 32.72541314],
       [-2.08629532, 32.7496465 ],
       [-2.07548167, 32.77415916],
       [-2.06494699, 32.79895079],
       [-2.05469094, 32.82402105],
       [-2.04471316, 32.84936959],
       [-2.03501329, 32.87499603],
       [-2.02559095, 32.9009    ],
       [-2.01644574, 32.92708111],
       [-2.00757727, 32.95353894],
       [-1.9989851 , 32.98027308],
       [-1.99066881, 33.00728311],
       [-1.98262797, 33.03456858],
       [-1.97486211, 33.06212903],
       [-1.96737077, 33.089964  ],
       [-1.96015348, 33.11807302],
       [-1.95320974, 33.14645559],
       [-1.94653905, 33.17511121],
       [-1.9401409 , 33.20403937],
       [-1.93401477, 33.23323955],
       [-1.92816011, 33.26271121],
```

```
       [-1.9225764 , 33.29245381],
       [-1.91726306, 33.32246678],
       [-1.91221953, 33.35274956],
       [-1.90744523, 33.38330157],
       [-1.90293958, 33.41412223],
       [-1.89870196, 33.44521093],
       [-1.89473178, 33.47656705],
       [-1.8910284 , 33.50818999],
       [-1.88759121, 33.5400791 ],
       [-1.88441955, 33.57223376],
       [-1.88151278, 33.6046533 ],
       [-1.87887024, 33.63733707],
       [-1.87649125, 33.6702844 ]])
```

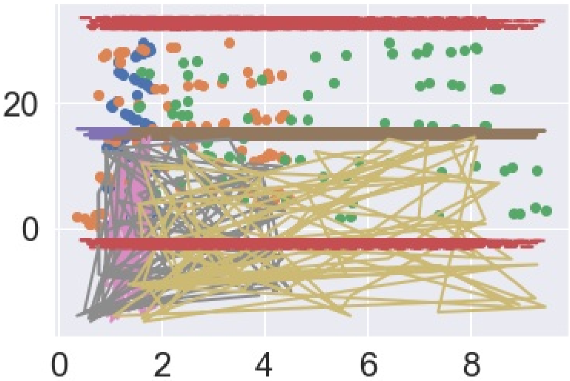
In [44]:

```
## Test data(Actual) and predicted data comparison.
plt.hist(y_test, bins = 5, label = "Actual", alpha = 0.7)
plt.hist(y_pred, bins = 5, label = "predicted", alpha = 0.7)

plt.legend()
plt.show()
```



In [45]:

```
plt.plot(X_test, y_test, 'o')
plt.plot(X_test, y_pred, '-', lw=2)
plt.plot(X_test, err, '-', lw=2)
plt.plot(X_test, CI[:,0], 'r--', lw=2)
plt.plot(X_test, CI[:,1], 'r--', lw=2)


plt.show()
```



Task 3: Approximate Bayesian Computation (ABC) Using 'rejection ABC' method to compute the posterior distributions of the 'selected' regression model parameters in Task 2.

1) You only need to compute 2 parameter posterior distributions -- the 2 parameters with largest absolute valuesin your least squares estimation (Task 2.1) of the selected model. Fix all the other parameters in your model as constant, by using the estimated values from Task 2.1.

2) Use a Uniform distribution as prior, around the estimated parameter values for those 2 parameters (from the Task 2.1). You will need to determine the range of the prior distribution.

3) Draw samples from the above Uniform prior, and perform rejection ABC for those 2 parameters.

4) Plot the joint and marginal posterior distribution for those 2 parameters.

5) Explain your results.

In [45]:
```
m_x4=data['x4'].mean()
m_x5=data['x5'].mean()
m_x32=data['X3_2'].mean()
```

In [46]:
```
m_x4=data['x4'].std()
m_x5=data['x5'].std()
m_x32=data['X3_2'].std()
```

In [47]:
```
data['X3_2'].max()
m_x32
```

Out[47]:
```
1.2804487880834328
```

In [48]:
```
import pymc3 as pm
```

In [49]:
```
with pm.Model() as normal_model:

    # The prior for the data likelihood is a Normal Distribution
    family = pm.glm.families.Normal()

    # Creating the model requires a formula and data (and optionally a family)
    pm.GLM.from_formula('time ~ x4+ np.square(x3)+x5', data=data, family = family)

    # Perform Markov Chain Monte Carlo sampling letting PyMC3 choose the algorithm
    normal_trace = pm.sample(draws=2000, chains = 2, tune = 500)
```

```
The glm module is deprecated and will be removed in version 4.0
We recommend to instead use Bambi https://bambinos.github.io/bambi/
WARNING (theano.tensor.blas): We did not find a dynamic library in the library_dir of the library we use for blas
. If you use ATLAS, make sure to compile it with dynamics library.
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [sd, x5, np.square(x3), x4, Intercept]
```

100.00% [2500/2500 00:28<00:00 Sampling chain 0, 0 divergences]

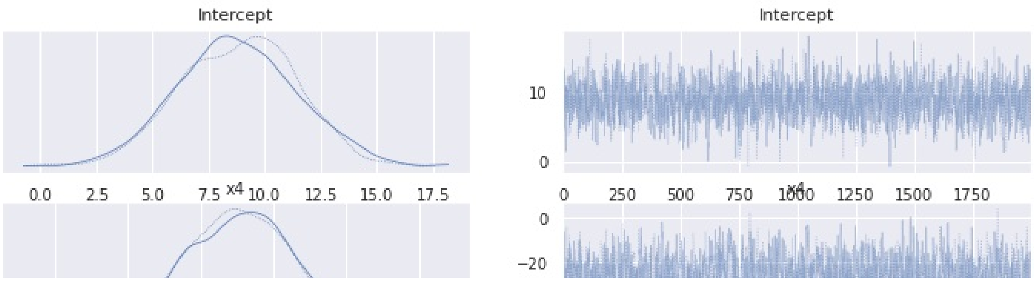
100.00% [2500/2500 00:31<00:00 Sampling chain 1, 0 divergences]

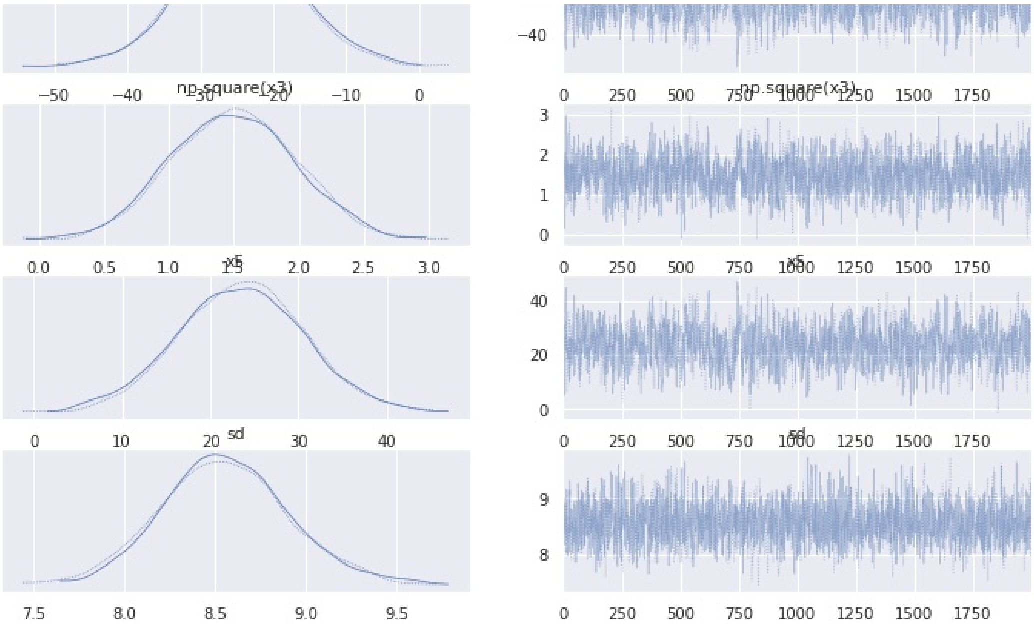
```
Sampling 2 chains for 500 tune and 2_000 draw iterations (1_000 + 4_000 draws total) took 60 seconds.
The acceptance probability does not match the target. It is 0.8915807221509819, but should be close to 0.8. Try t
o increase the number of tuning steps.
```

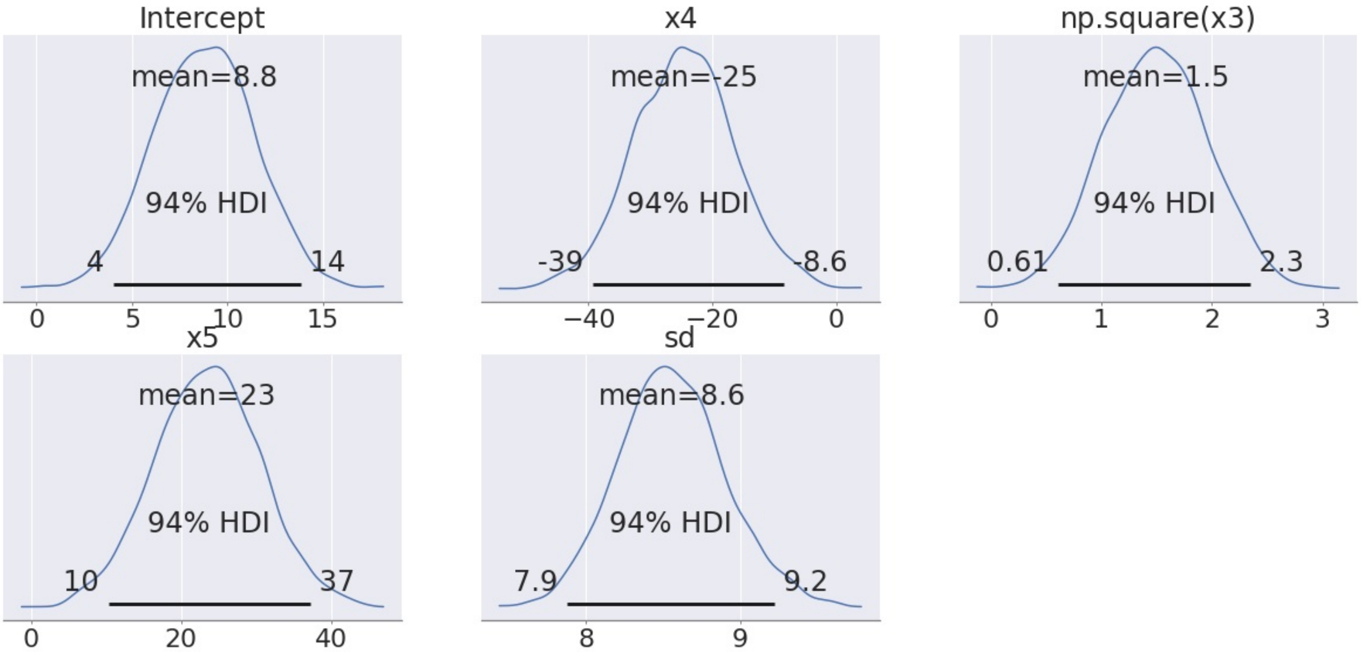
In [50]:
```
pm.traceplot(normal_trace)
pm.plot_posterior(normal_trace)
```

```
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

Out[50]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f460fe94850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f460fdf9810>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f460fd59950>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f460fd0ded0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f460fdec7d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f460fcf5610>]],
      dtype=object)
```




np.square(x3)
x5
sd


Intercept
mean=8.8
94% HDI
4
14
x4
mean=-25
94% HDI
-39
-8.6
np.square(x3)
mean=1.5
94% HDI
0.61
2.3
x5
mean=23
94% HDI
10
37
sd
mean=8.6
94% HDI
7.9
9.2

In [50]: