Homework 3 - CS348 Spring 2019

Description - This assignment is intended to teach you about linear models and naive bayes using python.

Getting Started - You should complete the assignment using your own installation of Python 3 and the packages numpy, pandas, matplotlib, and seaborn. Download the assignment from Moodle and unzip the file. This will create a directory with this file, 'HW03.ipynb', and a 'data' directory. The data files for each data set are in the 'data' directory.

Deliverables - The assignment has a single deliverable: this jupyter notebook file saved as a pdf. Please answer all coding and writing questions in the body of this file. Once all of the answers are complete, download the file by navigating the following menus: File -> Download as -> PDF via LaTeX. Submit the downloaded pdf file on gradescope. Alternatively, you can save the file as a pdf via the following: File -> Print Preview -> Print as pdf.

Note: You will be writing the written responses in the same cell block as the coding solution, so make sure to comment out the written responses.

Data Sets - In this assignment, you will run and analyze linear regression on a single synthetic dataset

Academic Honesty Statement — Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is considered cheating. Posting your code to public repositories such as GitHub is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

This academic honesty statement does not restrict you from reading official documentation or using other web resources for understanding the syntax of python, related data science libraries, or properties of distributions.

In [26]:
```
# Do not import any other libraries other than those listed here.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as skl
import seaborn as sns

from sklearn.linear_model import LinearRegression
```

Problem 1 - Regression

In this problem you'll construct a linear model to analyze a synthetic dataset.

In [27]:
```
# Load the data
x_train = np.loadtxt('data/x_train.csv', delimiter=',')
x_val = np.loadtxt('data/x_val.csv', delimiter=',')
y_train = np.loadtxt('data/y_train.csv', delimiter=',')
y_val = np.loadtxt('data/y_val.csv', delimiter=',')
```

Part 1 (16 points)
Using sci-kit learn's `LinearRegression` class, fit a linear regression model between features `x_train` and targets `y_train` . Using this model, produce a new array `y_train_res` which contains the residuals of the learned linear model corresponding to each row in the training dataset.

In [28]:
```
# Part 1 Solution

# --- write code here ---
reg=LinearRegression().fit(x_train, y_train)
y_pre=reg.predict(x_train)
y_train_res=[0]*100
for index in range(100):
    y_train_res[index]=y_train[index]-y_pre[index]
```

Part 2 (16 points)
Without using `LinearRegression` 's `score()` method or `sklearn.metrics.r2_score` , compute the R-squared value for the trained model. Does this result imply that the linear regression assumptions are satisfied? Why or why not? Would your answer change if the R-squared were larger?

In [29]:
```
# Part 2 Solution

# --- write code here ---
y_train_tss=0
mean=np.mean(y_train)
for i in range(100):
    y_train_tss += (y_train[i]-mean) ** 2
y_train_rss=0
for j in range(100):
    y_train_rss += (y_train_res[j])**2
r_squared = 1-(y_train_rss/y_train_tss)
print(r_squared)

# --- written response here ---
# The result doesn't imply that the linear regression assumptions are sa
tisfied
# The R-squared value is around 0.66, which is still far away from 1
# If the R-squared value is larger and close to 1, it implies that the l
inear regression assumptions are satisfied
```

```
0.6612116191209516
```

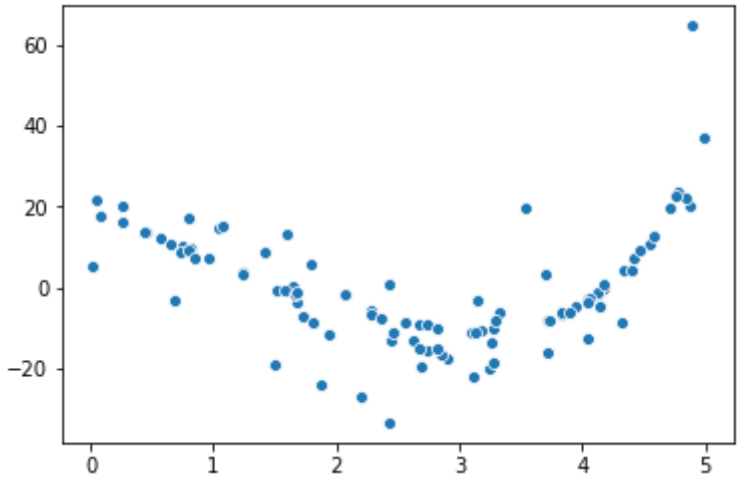
Based on their knowledge of the data generating process, your colleague suggests that a multiple linear regression may be an inappropriate method for analyzing this data. They instead suggest the conditional model `y = B_1 * x_1 + B_2 * f(x_2) + error`, where `y` is the outcome, `B_1` and `B_2` are parameters of the DGP, `x_1` and `x_2` are the first and second column of the dataset, `f()` is some non-linear function, and `error` is some random variable with 0 mean and constant variance.

Part 3 (10 points)
Make a residual plot with x_2 on the horizontal axis. Using this residual plot, describe any supporting evidence for your colleague's assumptions about the DGP.

Note: Recall the python/pandas is 0-indexed. Therefore, x_2 corresponds to `x_train[:, 1]`.

In [30]:
```
# Part 3 Solution

# --- write code here ---
sns.scatterplot(x=x_train[:,1], y=y_train_res)

# --- written response here ---
# The plot pattern is not random, thus it's a better fit for non-linear
 model.
# The pattern is not symmetrically distributed, which menas the linear r
egression is an
# inappropriate method.
```

Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2085a3c8>
```



Part 4 (30 points)
Using your analysis in parts 1-3 as a guide, determine which of the following functions best represent `f()` in your colleague's conditional model.

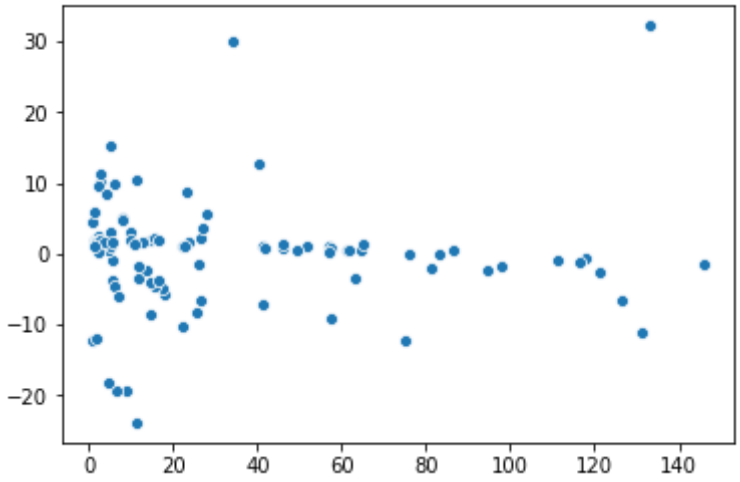
```
f(x) = ln(x)
f(x) = exp(x)
f(x) = x**2
f(x) = x**0.5
```

In your code solution, you should show the R-squared value on your training data for a simple linear regression after using each of these non-linear transformations, as well as the residual plot for the one function you have selected. Using these sources of evidence, explain why you made your selection between the 4 transformations.

In [31]:
```
# Part 4 Solution

# --- write code here ---
copy1=np.copy(x_train)
copy2=np.copy(x_train)
copy3=np.copy(x_train)
copy4=np.copy(x_train)

# apply function
copy1[:,1]=np.log(copy1[:,1])
copy2[:,1]=np.exp(copy2[:,1])
copy3[:,1]=copy3[:,1] ** 2
copy4[:,1]=copy4[:,1] ** 0.5

# fit new models
model1=LinearRegression().fit(copy1, y_train)
model2=LinearRegression().fit(copy2, y_train)
model3=LinearRegression().fit(copy3, y_train)
model4=LinearRegression().fit(copy4, y_train)

# get score
score1=model1.score(copy1, y_train)
score2=model2.score(copy2, y_train)
score3=model3.score(copy3, y_train)
score4=model4.score(copy4, y_train)

print(score1,score2,score3,score4)

# calculate residual
pred=model2.predict(copy2)
res=[0]*len(y_train)
for i in range(len(y_train)):
    res[i]=y_train[i]-pred[i]
sns.scatterplot(x=copy2[:,1], y=res)

# --- written response here ---
# I choose f(x)=exp(x) because its R-squared value is closer to 1 among
 the four results.
#
```

```
0.3481836559230782 0.9020636230233146 0.8078331499104046 0.534553879382
7771
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2099bef0>



Part 5 (10 points)
Evaluate the R-squared values on `x_val` and `y_val` for both the original linear model in part 1 and the selected model in part 4. Describe how and why these results differ from the evaluation results in part 1 and part 4.

Note: Make sure that whatever transformation you apply to the training data in part 4 is also applied to the validation data before evaluating the linear regression model. These models should be trained using only the training data, and then evaluated using only the validation data.

In [32]:
```
# Part 5 Solution

# --- write code here ---

print(reg.score(x_val, y_val))
x_val[:,1]=np.exp(x_val[:,1])
print(model2.score(x_val, y_val))

# --- written response here ---
# The R-squared values of part1 model are 0.66 and 0.53; of part4 are 0.
90 and 0.83.
# The reason the R-squared values differ from training data and testing
 data is that the model doesn't
# fit well for the testing data.
```

```
0.5296041983308444
0.8323371024157278
```

Part 6 (18 points) Make a new residual plot using `x_1` (i.e. `x_train[:, 0]`) with your selected model from part 4. What does this imply about the assumptions in our regression model? Write and describe the equation for a conditional model similar to your colleagues that adjusts for this new finding.

In [33]:
```
# Part 6 Solution

# --- write code here ---
sns.scatterplot(x=copy2[:,0], y=res)

# --- written response here ---
# Compared with the pattern of regression model, this pattern has more d
ots scatter close to 0.
# The difference between two models is that our regression model is trai
ned wihtout applying exp() on x_2
# and this model has exp() applied on x_2.
# Thus it means that the a multiple linear regression is inappropriate f
or this data.
# Instead, conditional model is a better fit for this data.

# The conditional model is y = B_1*x_1 + B_2*f(x_2) + error, where y is
 the outcome, B_1 and B_2 are the parameters
# of the DGP, x_1 and x_2 are the first and second column of the datase
t, f() is non-linear exponential function exp()
# and error is some random variable with 0 mean and constant variance.
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1a209ec748>