In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics

plt.style.use('seaborn')
pd.set_option('display.max_columns', 100)
```

In [3]:
```
# Loading dataset
```

In [4]:
```
data_1 = pd.read_csv('data_1.csv')
```

In [5]:
```
data_1 = data_1[['V1','V2']]
```

In [6]:
```
data = data_1.iloc[1:,:]# =
data.columns = data_1.iloc[0]
```

In [7]:
```
data = data.astype(float)
```

In [8]:
```
#Observing data
```

In [9]:
```
data.head(10)
```

Out[9]:

	density	gain
1	0.686	17.6
2	0.686	17.3
3	0.686	16.9
4	0.686	16.2
5	0.686	17.1
6	0.686	18.5
7	0.686	18.7
8	0.686	17.4
9	0.686	18.6
10	0.686	16.8

In [10]:
```
# Scenario 1, Fitting
```

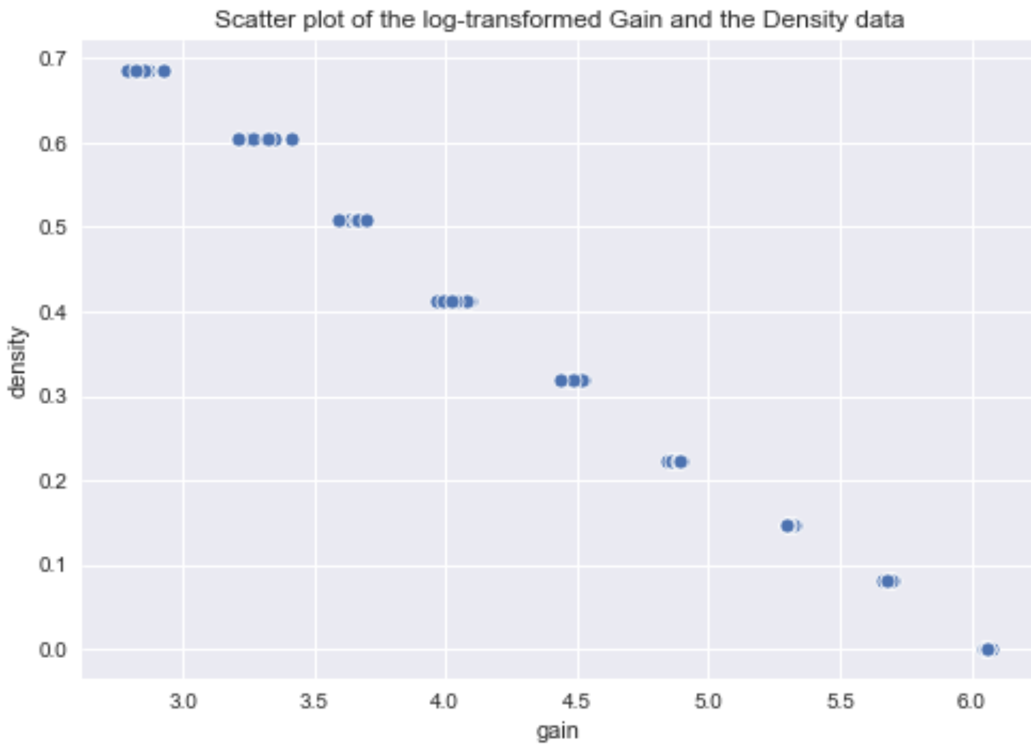
In [11]:
```
#Use the data to fit the gain,or a transformation of gain,to density.

#Try sketching the least squares line on a scatter plot.
```

In [12]:
```
sns.scatterplot(data.gain, data.density)
plt.title("Scatter plot of the Gain and the Density data")
```

Out[12]: `Text(0.5,1,'Scatter plot of the Gain and the Density data')`

In [13]:
```
sns.scatterplot(np.log(data.gain), data.density)
plt.title("Scatter plot of the log-transformed Gain and the Density data
```

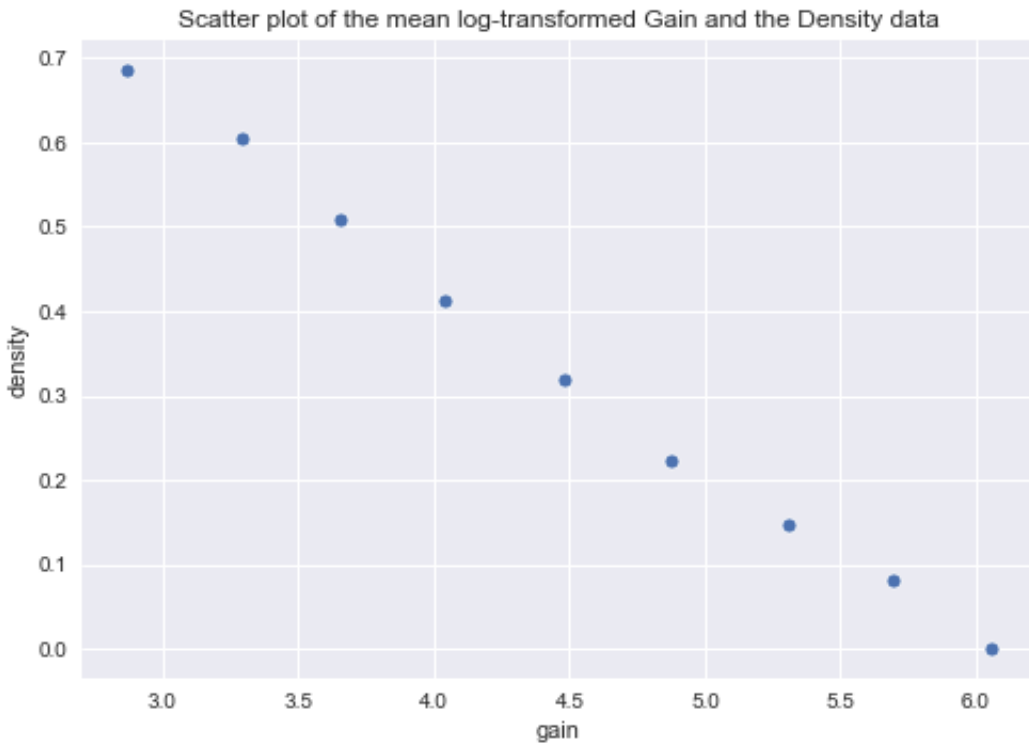
Out[13]:
```
Text(0.5,1,'Scatter plot of the log-transformed Gain and the Density d
ata')
```



In [14]:
```
temp = data.groupby('density').gain.apply(lambda x: np.log(x.mean())).to
```

In [15]:
```
sns.scatterplot(temp.gain, temp.density)
plt.title("Scatter plot of the mean log-transformed Gain and the Density
```

Out[15]:
```
Text(0.5,1,'Scatter plot of the mean log-transformed Gain and the Dens
ity data')
```



In [16]:
```
# least square line and scatter plot of original data of log transformat
```

In [17]:
```
from statsmodels.regression.quantile_regression import QuantReg
import statsmodels.formula.api as smf
```

In [18]:
```
# Least Absolute Devidations Regression Line
```

In [19]:
```
data = data.assign(logy = np.log(data.gain))
```

In [20]:
```
data.head()
```

Out[20]:

	density	gain	logy
1	0.686	17.6	2.867899
2	0.686	17.3	2.850707
3	0.686	16.9	2.827314
4	0.686	16.2	2.785011
5	0.686	17.1	2.839078

In [21]:
```
mod = smf.quantreg('density ~ logy', data)
res = mod.fit(q=.5)
```

In [22]:
```
res.params
```

Out[22]:
```
Intercept    1.295485
logy        -0.215571
dtype: float64
```

In [23]:
```
ladr_slope,ladr_intercept  = res.params['logy'], res.params['Intercept']
```

In [24]:
```
ladr_slope
```

Out[24]:
```
-0.21557061687315826
```

In [ ]:
```

```

In [25]:
```
minimum = (min(data.logy) - .1)* ladr_slope + ladr_intercept
```

In [26]:
```
maximum = (max(data.logy) + .1)* ladr_slope + ladr_intercept
```

In [27]:
```
temp = data.groupby('density').gain.apply(lambda x: np.log(x.mean())).to
```

In [28]:
```
temp
```

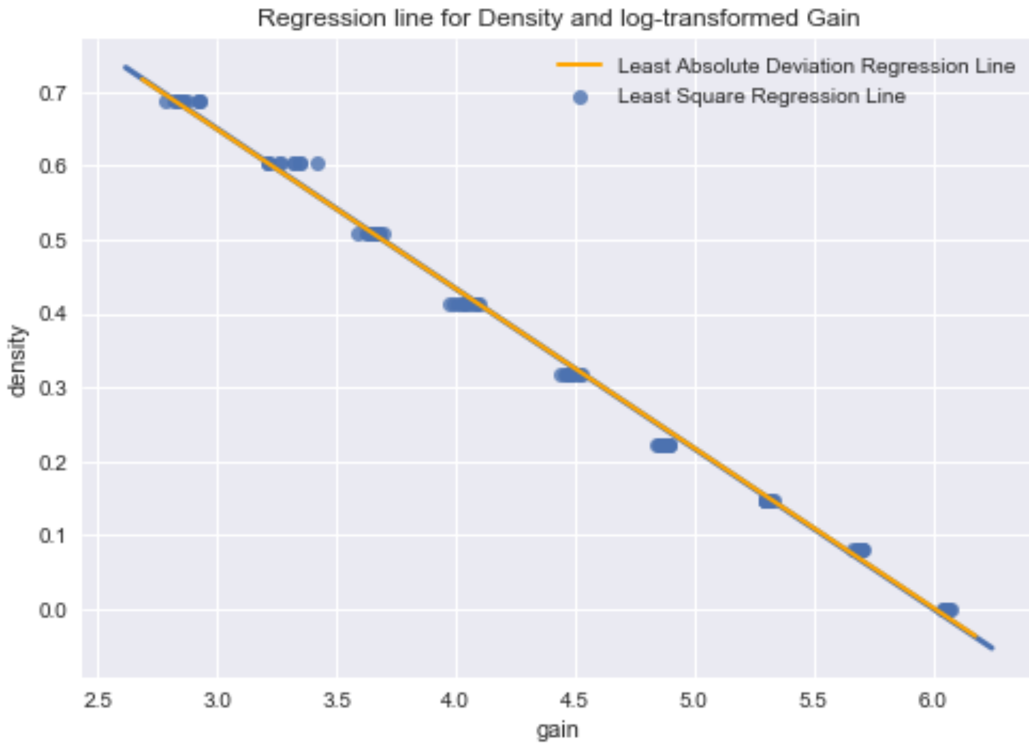
Out[28]:

	density	gain
0	0.001	6.056081
1	0.080	5.690697
2	0.148	5.305293
3	0.223	4.872139
4	0.318	4.482890
5	0.412	4.039888
6	0.508	3.651956
7	0.604	3.293241
8	0.686	2.862772

In [ ]:

In [29]:
```
sns.regplot(np.log(data.gain), data.density, ci = None, label = 'Least S
plt.plot([min(data.logy) - .1, max(data.logy) + .1],[minimum, maximum],l
plt.legend()
plt.title('Regression line for Density and log-transformed Gain')
```

Out[29]: Text(0.5,1,'Regression line for Density and log-transformed Gain')



In [30]:
```
# Get the parameter of least square line
from scipy import stats
```

In [31]:
```
#least square line of original data
stats.linregress(data.gain, data.density)
```

Out[31]: LinregressResult(slope=-0.0015334078316468012, intercept=0.5497239543095568, rvalue=-0.9031596703485592, pvalue=4.518580918277026e-34, stderr=7.769937888653647e-05)

In [32]:
```
#least square line of log-transformed data
stats.linregress(np.log(data.gain), data.density)
```

Out[32]: LinregressResult(slope=-0.21620320109581187, intercept=1.2980126052584207, rvalue=-0.9979069608362692, pvalue=1.8572471194543776e-106, stderr=0.0014935062316818492)

In [33]:
```
#Least Absolute Deviations Regression Line
print("slope = {}".format(ladr_slope))
print("intercept = {}".format(ladr_intercept))
```

```
slope = -0.21557061687315826
intercept = 1.2954851423146105
```

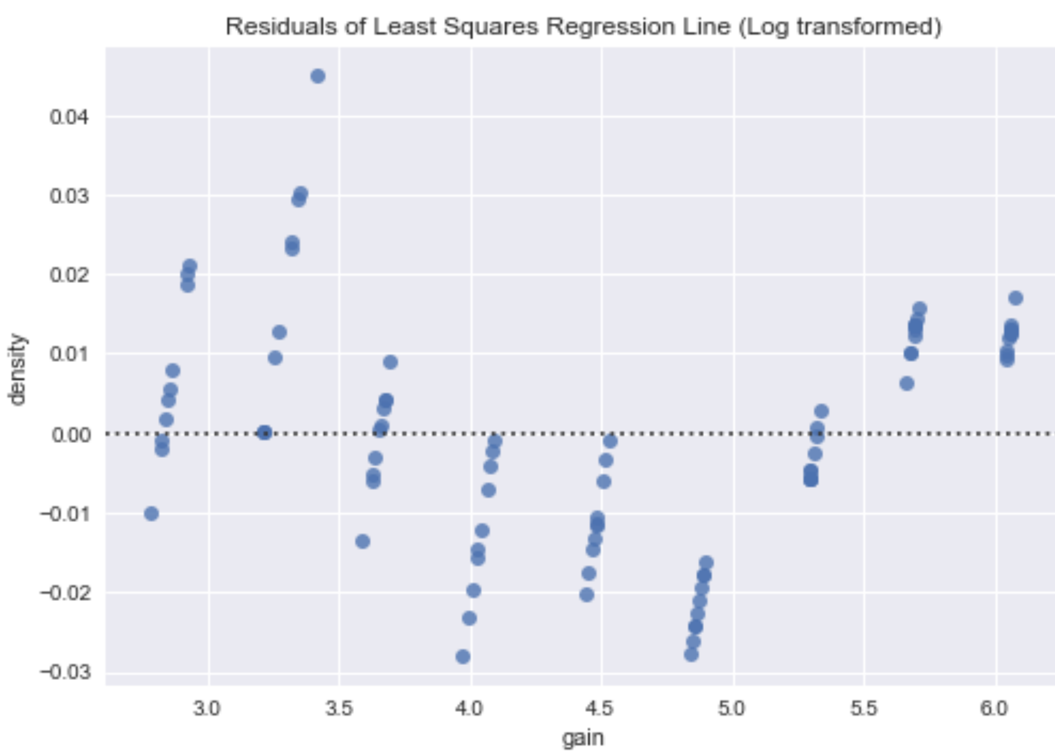
In [34]:
```
slope = stats.linregress(np.log(data.gain), data.density)[0]
intercept = stats.linregress(np.log(data.gain), data.density)[1]
```

In [35]:
```
pred = np.log(data.gain) * slope + intercept
```

In [36]:
```
residuals = data.density - pred
#residuals = np.exp(residuals)
```

In [37]:
```
sns.residplot(np.log(data.gain), data.density)
plt.title('Residuals of Least Squares Regression Line (Log transformed)'
```

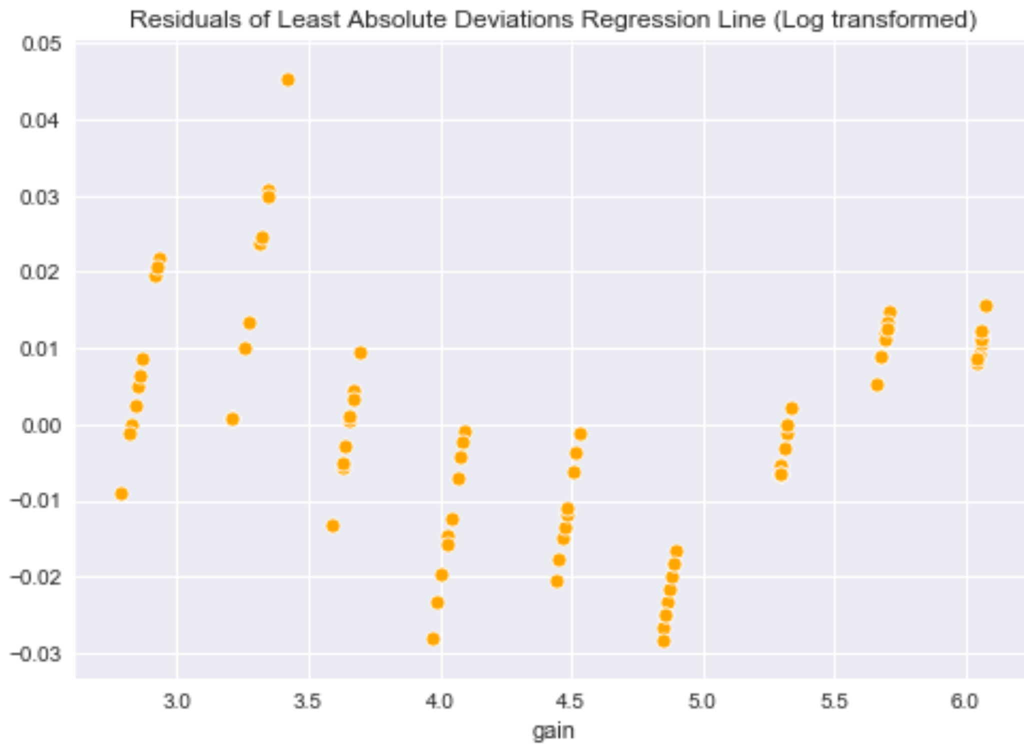
Out[37]:
```
Text(0.5,1,'Residuals of Least Squares Regression Line (Log transforme
d)')
```



In [38]:
```
pred_ladr = np.log(data.gain) *ladr_slope + ladr_intercept
residuals_ladr = data.density - pred_ladr
```

In [39]:
```
sns.scatterplot(np.log(data.gain), residuals_ladr, color = 'orange')
plt.title('Residuals of Least Absolute Deviations Regression Line (Log t
```

Out[39]:
```
Text(0.5,1,'Residuals of Least Absolute Deviations Regression Line (Lo
g transformed)')
```

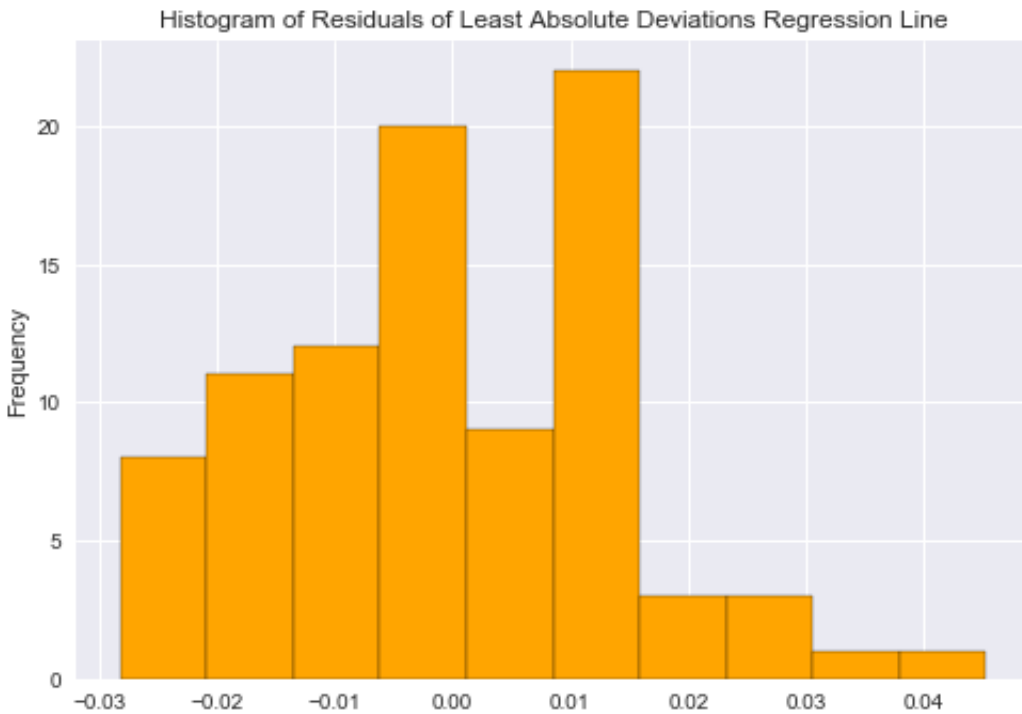


In [40]:
```
residuals.plot(kind = 'hist', edgecolor = 'k')
plt.title('Histogram of Residuals of Least Squares Regresion Line')
```

Out[40]:
```
Text(0.5,1,'Histogram of Residuals of Least Squares Regresion Line')
```

In [41]:
```
residuals_ladr.plot(kind = 'hist', edgecolor = 'k', color = 'orange')
plt.title('Histogram of Residuals of Least Absolute Deviations Regressio
```

Out[41]: Text(0.5,1,'Histogram of Residuals of Least Absolute Deviations Regres
sion Line')

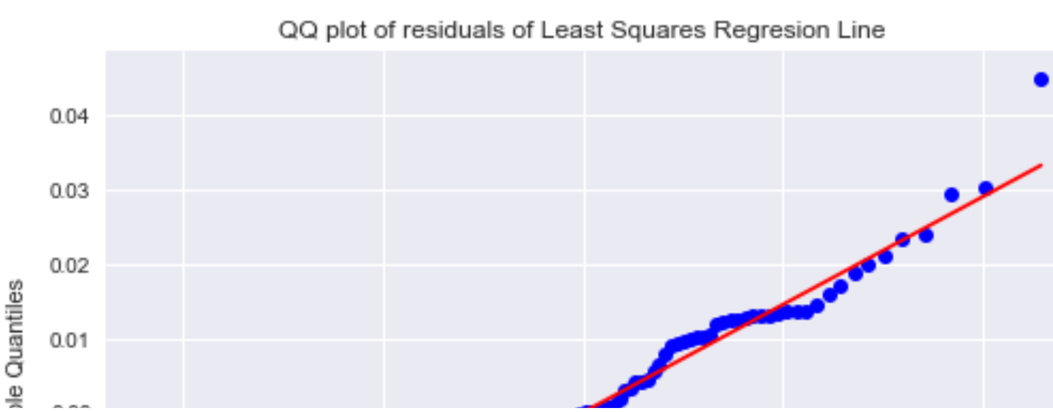


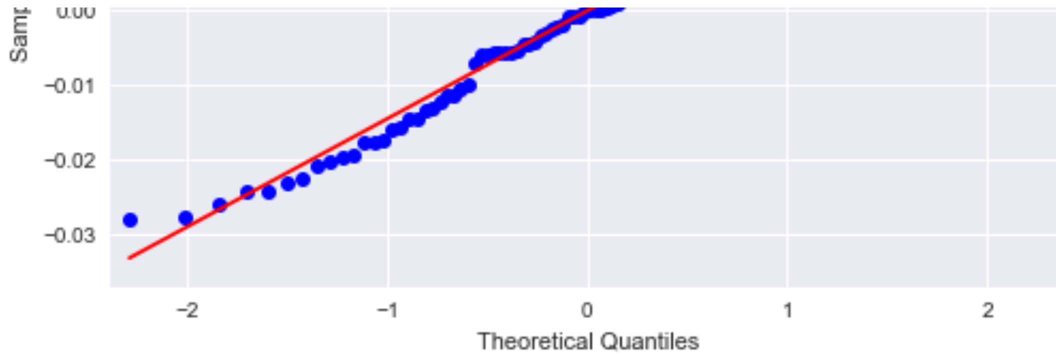
In [42]:
```
import statsmodels.api as sm
```

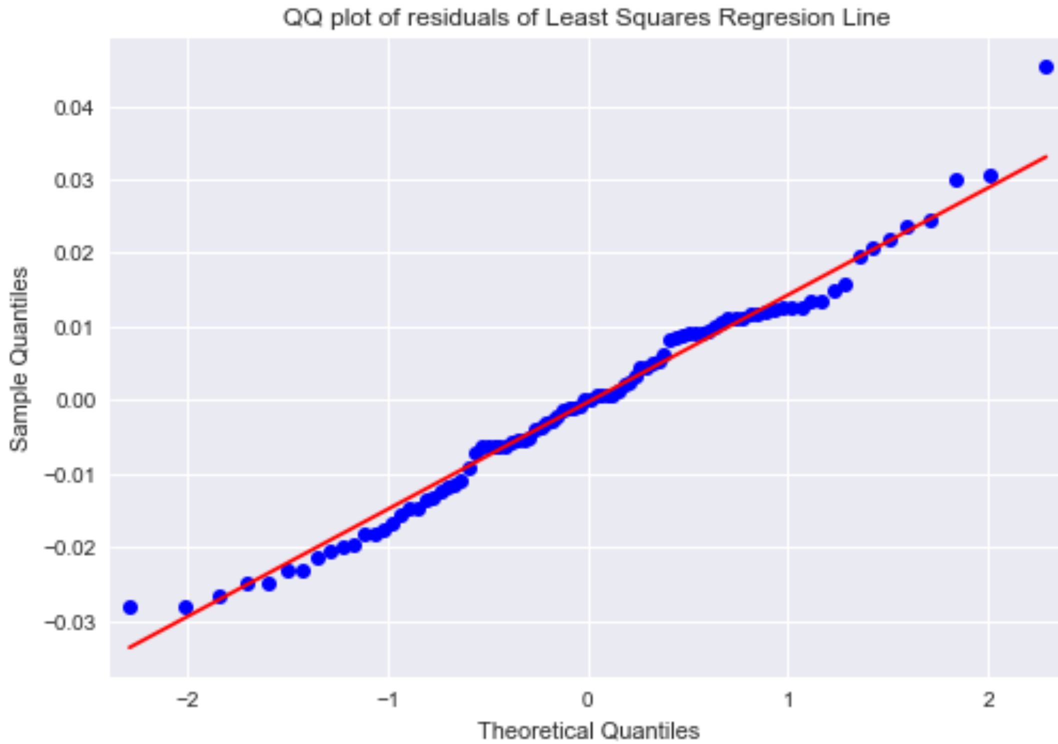
In [43]:
```
# QQ plot of residuals of Least Squares Regresion Line
```

In [46]:
```
sm.qqplot(residuals, line="s")
plt.title('QQ plot of residuals of Least Squares Regresion Line')
sm.qqplot(residuals_ladr, line="s")
plt.title('QQ plot of residuals of Least Squares Regresion Line')
```

Out[46]: Text(0.5,1,'QQ plot of residuals of Least Squares Regresion Line')







Scenario 2, Predicting

In [47]:
```
#Scenario 2, Predicting
a = 38.6
b = 426.7
```

In [40]:
```
# plot for ls of log transformed c.i & prediction interval
```

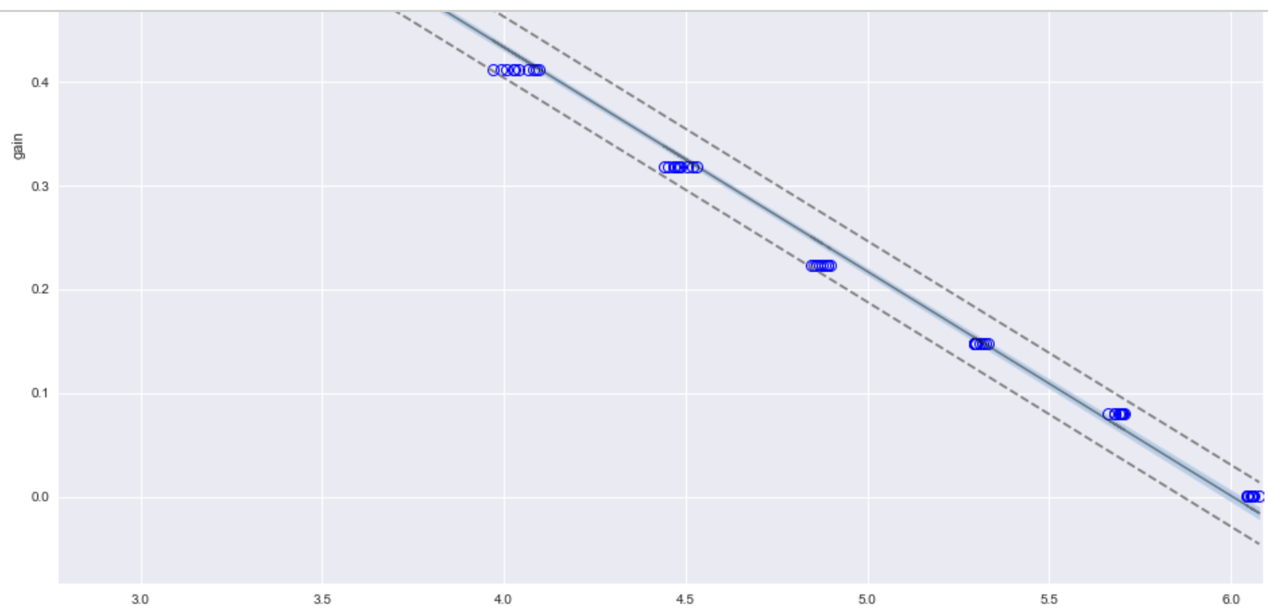
In [48]:
```
def plot_ci_manual(t, s_err, n, x, x2, y2, ax=None):
    """Return an axes of confidence bands using a simple approach.

    Notes
    -----
    .. math:: \left| \: \hat{\mu}_{y|x0} - \mu_{y|x0} \: \right| \; \leq
       math:: \hat{\sigma} = \sqrt{\sum_{i=1}^n{\frac{(y_i-\hat{y})^2}{n
```

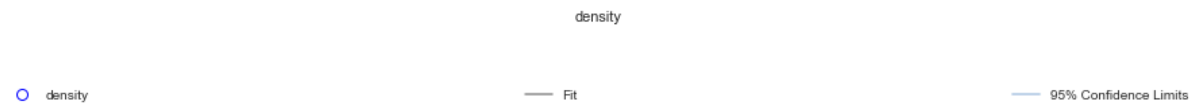
```
    .. math:: \hat{\sigma} = \sqrt{\sum_{i=1}^n{\frac{(y_i-\hat{y})^2}{n
    
    References
    ----------
    .. [1]: M. Duarte.  "Curve fitting," JUpyter Notebook.
       http://nbviewer.ipython.org/github/demotu/BMC/blob/master/noteboo

    """
    if ax is None:
        ax = plt.gca()

    ci = t*s_err*np.sqrt(1/n + (x2-np.mean(x))**2/np.sum((x-np.mean(x))*
    ax.fill_between(x2, y2+ci, y2-ci, color="#b9cfe7", edgecolor="")

    return ax


def plot_ci_bootstrap(xs, ys, resid, nboot=500, ax=None):
    """Return an axes of confidence bands using a bootstrap approach.

    Notes
    -----
    The bootstrap approach iteratively resampling residuals.
    It plots `nboot` number of straight lines and outlines the shape of
    The density of overlapping lines indicates improved confidence.

    Returns
    -------
    ax : axes
        - Cluster of lines
        - Upper and Lower bounds (high and low) (optional)  Note: sensit

    References
    ----------
    .. [1] J. Stults. "Visualizing Confidence Intervals", Various Conseq
       http://www.variousconsequences.com/2010/02/visualizing-confidence

    """
    if ax is None:
        ax = plt.gca()

    bootindex = sp.random.randint

    for _ in range(nboot):
        resamp_resid = resid[bootindex(0, len(resid)-1, len(resid))]
        # Make coeffs of for polys
        pc = sp.polyfit(xs, ys + resamp_resid, 1)
        # Plot bootstrap cluster
        ax.plot(xs, sp.polyval(pc, xs), "b-", linewidth=2, alpha=3.0/flo

    return ax
```

In [49]:
```
x = np.log(data.gain)
y = data.density
```

In [50]:
```
def equation(a, b):
    """Return a 1D polynomial."""
    return np.polyval(a, b)

p, cov = np.polyfit(x, y, 1, cov=True)                          # parameters
y_model = equation(p, x)                                        # model using

# Statistics
n = len(y)                                                   # number of observat
m = p.size                                                      # number of p
dof = n - m                                                     # degrees of
t = stats.t.ppf(0.975, n - m)                                   # used for CI

# Estimates of Error in Data/Model
resid = y - y_model
chi2 = np.sum((resid/y_model)**2)                               # chi-squared
chi2_red = chi2/(dof)                                           # reduced chi
s_err = np.sqrt(np.sum(resid**2)/(dof))                         # standard de


# Plotting -------------------------------------------------------------
fig, ax = plt.subplots(figsize=(16, 12))

# Data
ax.plot(
    x, y, "o", color="#b9cfe7", markersize=8,
    markeredgewidth=1, markeredgecolor="b", markerfacecolor="None"
)

# Fit
ax.plot(x,y_model,"-", color="0.1", linewidth=1.5, alpha=0.5, label="Fit

x2 = np.linspace(np.min(x), np.max(x), 100)
y2 = equation(p, x2)

# Confidence Interval (select one)
plot_ci_manual(t, s_err, n, x, x2, y2, ax=ax)
#plot_ci_bootstrap(x, y, resid, ax=ax)

# Prediction Interval
pi = t*s_err*np.sqrt(1+1/n+(x2-np.mean(x))**2/np.sum((x-np.mean(x))**2))
ax.fill_between(x2, y2+pi, y2-pi, color="None", linestyle="--")
ax.plot(x2, y2-pi, "--", color="0.5", label="95% Prediction Limits")
ax.plot(x2, y2+pi, "--", color="0.5")
```

```
# Figure Modifications -----------------------------------------------
# Borders
ax.spines["top"].set_color("0.5")
ax.spines["bottom"].set_color("0.5")
ax.spines["left"].set_color("0.5")
ax.spines["right"].set_color("0.5")
ax.get_xaxis().set_tick_params(direction="out")
ax.get_yaxis().set_tick_params(direction="out")
ax.xaxis.tick_bottom()
ax.yaxis.tick_left()

# Labels
plt.title("Fit Plot for density vs. gain", fontsize="14", fontweight="bo
plt.xlabel("density")
plt.ylabel("gain")
plt.xlim(np.min(x)-.01,np.max(x)+.01)

# Custom legend
handles, labels = ax.get_legend_handles_labels()
display = (0, 1)
anyArtist = plt.Line2D((0,1), (0,0), color="#b9cfe7")      # create cust
legend = plt.legend(
    [handle for i, handle in enumerate(handles) if i in display] + [anyA
    [label for i, label in enumerate(labels) if i in display] + ["95% Co
    loc=9, bbox_to_anchor=(0, -0.21, 1., .102), ncol=3, mode="expand"
)
frame = legend.get_frame().set_edgecolor("0.5")

# Save Figure
#plt.tight_layout()
#plt.savefig("filename.png", bbox_extra_artists=(legend,), bbox_inches="

plt.show()
```





In [160]:
```
0.508, 0.001
```
Out[160]: (0.508, 0.001)

In [44]:
```
#Least Square
```

In [51]:
```
lse = stats.linregress(np.log(data.gain),data.density)
```

In [52]:
```
lse
```
Out[52]:
```
LinregressResult(slope=-0.21620320109581187, intercept=1.2980126052584
207, rvalue=-0.9979069608362692, pvalue=1.8572471194543776e-106, stder
r=0.0014935062316818492)
```

In [53]:
```
mod = smf.quantreg('density ~ logy', data)
res = mod.fit(q=.5)
```

In [54]:
```
#ls pred of a
pred = lse[0] * np.log(a) + lse[1]
pred
```
Out[54]: 0.508167768674875

In [55]:
```
#ladr pred
pred_ladr = res.params['logy'] * np.log(a) + res.params['Intercept']
pred_ladr
```
Out[55]: 0.5079512954825337

In [56]:
```
#pi = t*lse[4]*np.sqrt(1/n+(np.log(38.6)-np.mean(x))**2/np.sum((np.log(3
pi = t*lse[4]*np.sqrt(1/n+(np.log(38.6)-np.mean(x))**2/np.sum((x-np.mean
```

In [57]:
```
#CI of ls pred, assume known variance
me = pi
pred - me, pred + me
```
Out[57]: (0.5077693465097162, 0.5085661908400338)

In [58]:
```
#pi = t*lse[4]*np.sqrt(1+1/n+(np.log(38.6)-np.mean(x))**2/np.sum((np.log
pi = t*lse[4]*np.sqrt(1+1/n+(np.log(38.6)-np.mean(x))**2/np.sum((x-np.me
```

In [59]:
```
#confidence this interval contains the density of the next data point wi
me = pi
pred - me, pred + me
```

Out[59]: (0.5051731165919096, 0.5111624207578405)

In [60]:
```
#ls pred of b
pred = lse[0] * np.log(b) + lse[1]
pred
```

Out[60]: -0.011331534157646539

In [61]:
```
#ladr pred of b
pred_ladr = res.params['logy'] * np.log(b) + res.params['Intercept']
pred_ladr
```

Out[61]: -0.010028015689430791

In [62]:
```
#pi = t*lse[4]*np.sqrt(1/n+(np.log(b)-np.mean(x))**2/np.sum((np.log(b)-n
pi = t*lse[4]*np.sqrt(1/n+(np.log(b)-np.mean(x))**2/np.sum((x-np.mean(x)
```

In [63]:
```
#CI of ls pred of b
me = pi
pred - me, pred + me
```

Out[63]: (-0.011902094325343738, -0.010760973989949934)

In [64]:
```
#pi = t*lse[4]*np.sqrt(1+1/n+(np.log(b)-np.mean(x))**2/np.sum((np.log(b)
pi = t*lse[4]*np.sqrt(1+1/n+(np.log(b)-np.mean(x))**2/np.sum((x-np.mean(
```

In [65]:
```
#prediction interval
me = pi
pred - me, pred + me
```

Out[65]: (-0.014353907361154397, -0.008309160954138681)

Scenario 3

In [67]:
```
a = 38.6
y = 0.508
b = 426.7
```

In [68]:
```
data = data[['density','gain']]
```

In [69]:
```
train = data[data.density!=0.508]
test = data[data.density == 0.508]
```

In [70]:
```
x = np.log(train.gain)
n = len(train)
```

In [71]:
```
lse_inverse = stats.linregress(np.log(train.gain),train.density)
```

In [73]:
```
lse_inverse
```

Out[73]:
```
LinregressResult(slope=-0.21627808679336208, intercept=1.2984221228853
376, rvalue=-0.9977772794097103, pvalue=1.6043117188628408e-93, stderr
=0.0016354888159116026)
```

In [72]:
```
slope, intercept = lse_inverse[0], lse_inverse[1]
```

In [74]:
```
#ls
pred = np.log(a) * slope + intercept
pred
```

Out[74]:
```
0.5083037099567416
```

In [75]:
```
train = train.assign(logy = np.log(train.gain))
```

In [76]:
```
#ladr
mod = smf.quantreg('density ~ logy', train)
res = mod.fit(q=.5)
```

In [77]:
```
pred_ladr = res.params['logy'] * np.log(a) + res.params['Intercept']
pred_ladr
```

Out[77]:
```
0.5070976230054153
```

In [78]:
```
#pi = t*lse_inverse[4]*np.sqrt(1/n+(np.log(38.6)-np.mean(x))**2/np.sum((
pi = t*lse_inverse[4]*np.sqrt(1/n+(np.log(38.6)-np.mean(x))**2/np.sum((x
```

In [79]:
```
me = pi
pred - me, pred + me
```

Out[79]:
```
(0.507821746432322, 0.5087856734811611)
```

In [80]:
```
#pi = t*lse_inverse[4]*np.sqrt(1 + 1/n+(np.log(38.6)-np.mean(x))**2/np.s
pi = t*lse_inverse[4]*np.sqrt(1 + 1/n+(np.log(38.6)-np.mean(x))**2/np.su
```

In [81]:
```
me = pi
pred - me, pred + me
```

Out[81]: (0.5050179792716427, 0.5115894406418404)

In [82]:
```
train = data[data.density!=0.001]
test = data[data.density == 0.001]
```

In [83]:
```
x = np.log(train.gain)
n= len(train)
```

In [84]:
```
lse_inverse = stats.linregress(np.log(train.gain),train.density)
```

In [85]:
```
slope, intercept = lse_inverse[0], lse_inverse[1]
```

In [86]:
```
pred = np.log(b) * slope + intercept
pred
```

Out[86]: -0.018558799389517766

In [87]:
```
pred_ladr = res.params['logy'] * np.log(b) + res.params['Intercept']
pred_ladr
```

Out[87]: -0.009654316997500079

In [88]:
```
pi = t*lse_inverse[4]*np.sqrt(1/n+(np.log(b)-np.mean(x))**2/np.sum((x-np
```

In [89]:
```
me = pi
pred - me, pred + me
```

Out[89]: (-0.019404774100666668, -0.017712824678368853)

In [90]:
```
pi = t*lse_inverse[4]*np.sqrt(1 + 1/n+(np.log(b)-np.mean(x))**2/np.sum((
```

In [91]:
```
me = pi
pred - me, pred + me
```

Out[91]: (-0.022154001258395405, -0.014963597520640127)

In [ ]:

Additional Temperature and Latitude

In [95]:
```
addata = pd.read_csv('Full Resolution Data/64503600.csv')
addata = addata.astype(float)
addata.head()
```

Out[95]:

	BuoyID	Year	Hour	Min	DOY	POS_DOY	Lat	Lon	BP	Ts
0	64503600.0	2017.0	12.0	0.0	7.500	7.500	63.0656	-5.0590	1018.6	6.19
1	64503600.0	2017.0	13.0	0.0	7.542	7.542	63.0654	-5.0358	1017.8	6.13
2	64503600.0	2017.0	14.0	0.0	7.583	7.583	63.0646	-5.0112	1017.2	6.11
3	64503600.0	2017.0	15.0	0.0	7.625	7.625	63.0654	-4.9860	1016.2	6.10
4	64503600.0	2017.0	16.0	0.0	7.667	7.667	63.0692	-4.9632	1015.6	6.08

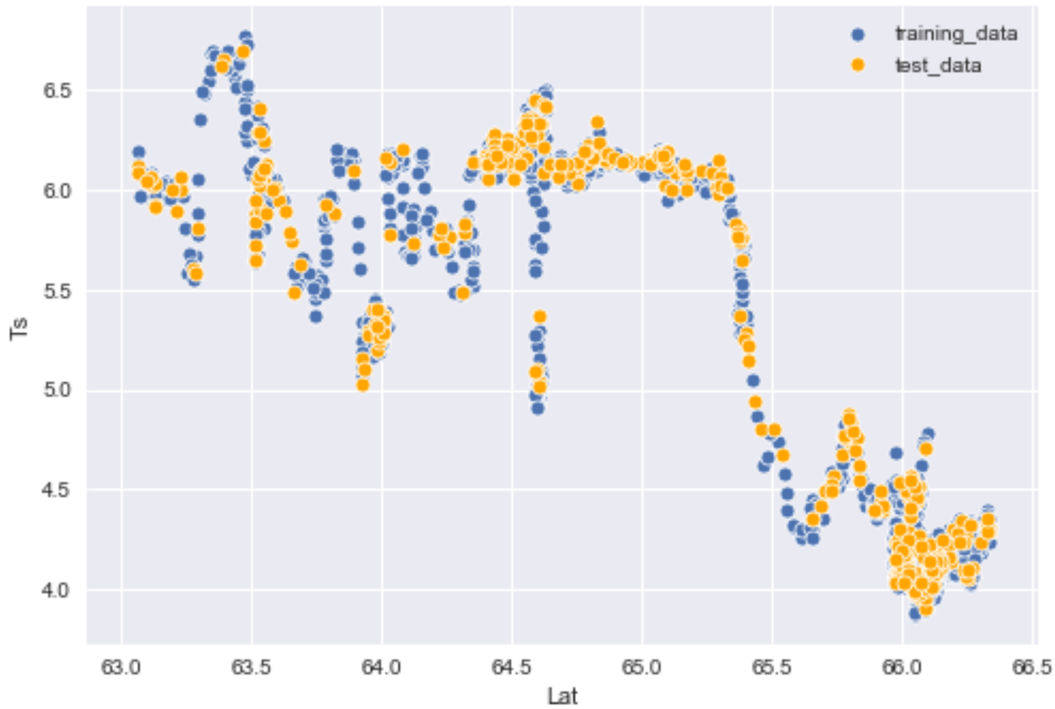
In [96]:
```
X = addata.drop('Ts',axis = 1)
y = addata.Ts
```

In [97]:
```
from sklearn.model_selection import train_test_split
```

In [98]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [99]:
```
sns.scatterplot(x = X_train.Lat, y = y_train, label = 'training_data')
sns.scatterplot(x = X_test.Lat, y = y_test, color = 'orange', label = 't
plt.legend()
```

Out[99]: <matplotlib.legend.Legend at 0x1c1d1276a0>



In [102]:
```
reg = stats.linregress(X_train['Lat'],y_train)
```

In [103]:
```
reg
```

Out[103]:
```
LinregressResult(slope=-0.8317229112792943, intercept=59.3866888687432
9, rvalue=-0.8389063819500947, pvalue=0.0, stderr=0.013272346699615703
)
```

In [107]:
```
slope = reg[0]
slope
```

Out[107]: -0.8317229112792943

In [108]:
```
intercept = reg[1]
intercept
```

Out[108]: 59.38668886874329

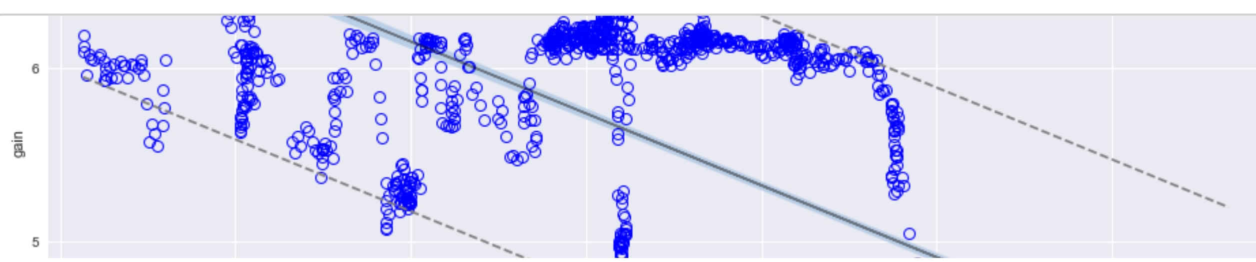
In [109]:
```
rsquared = reg[2]**2
rsquared
```
Out[109]: 0.7037639176765982

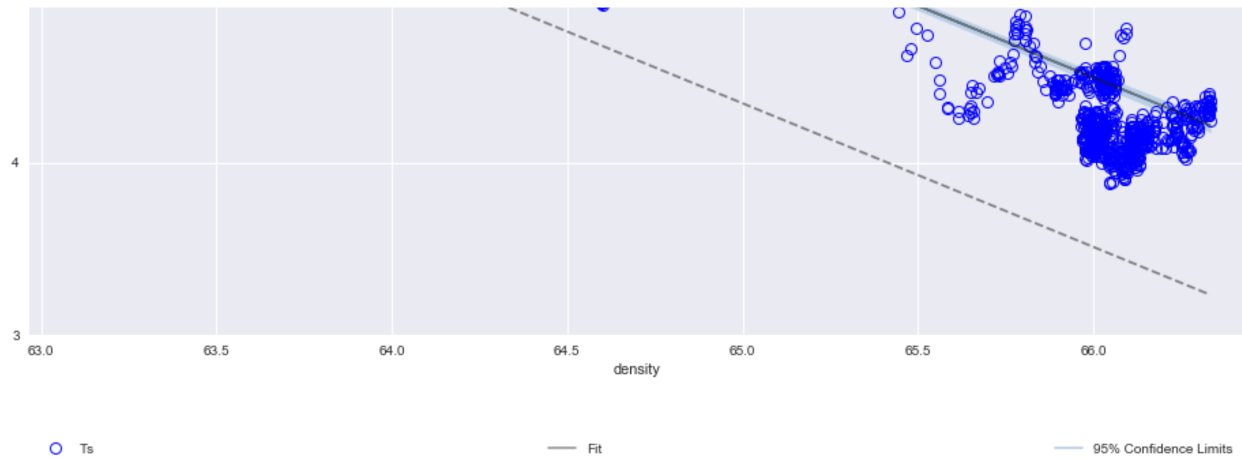
In [111]:
```
pval = reg[3]
pval
```
Out[111]: 0.0

In [112]:
```
x = X_train['Lat']
y = y_train
```

In [116]:
```
def equation(a, b):
    """Return a 1D polynomial."""
    return np.polyval(a, b)

p, cov = np.polyfit(x, y, 1, cov=True)                        # parameters
y_model = equation(p, x)                                      # model using

# Statistics
n = len(y)                                                 # number of observat
m = p.size                                                    # number of p
dof = n - m                                                   # degrees of
t = stats.t.ppf(0.975, n - m)                                 # used for CI

# Estimates of Error in Data/Model
resid = y - y_model
chi2 = np.sum((resid/y_model)**2)                             # chi-squared
chi2_red = chi2/(dof)                                         # reduced chi
s_err = np.sqrt(np.sum(resid**2)/(dof))                       # standard de


# Plotting -------------------------------------------------------------
fig, ax = plt.subplots(figsize=(16, 12))

# Data
ax.plot(
    x, y, "o", color="#b9cfe7", markersize=8,
    markeredgewidth=1, markeredgecolor="b", markerfacecolor="None"
)

# Fit
ax.plot(x,y_model,"-", color="0.1", linewidth=1.5, alpha=0.5, label="Fit

x2 = np.linspace(np.min(x), np.max(x), 100)
y2 = equation(p, x2)

# Confidence Interval (select one)
```

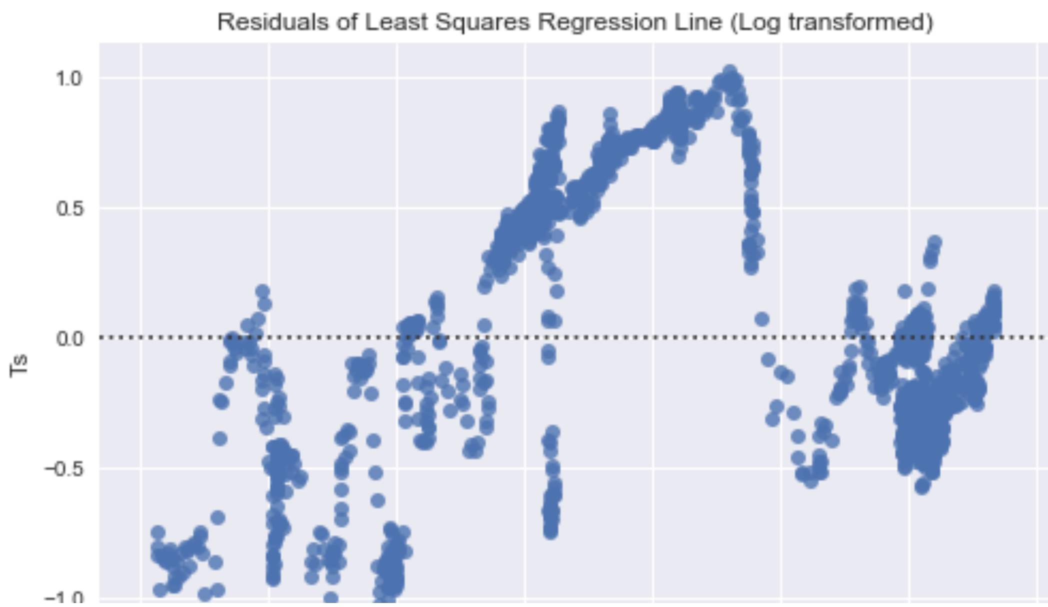
```
# Confidence Interval (select one)
plot_ci_manual(t, s_err, n, x, x2, y2, ax=ax)
#plot_ci_bootstrap(x, y, resid, ax=ax)

# Prediction Interval
pi = t*s_err*np.sqrt(1+1/n+(x2-np.mean(x))**2/np.sum((x-np.mean(x))**2))
ax.fill_between(x2, y2+pi, y2-pi, color="None", linestyle="--")
ax.plot(x2, y2-pi, "--", color="0.5", label="95% Prediction Limits")
ax.plot(x2, y2+pi, "--", color="0.5")


# Figure Modifications -----------------------------------------------------
# Borders
ax.spines["top"].set_color("0.5")
ax.spines["bottom"].set_color("0.5")
ax.spines["left"].set_color("0.5")
ax.spines["right"].set_color("0.5")
ax.get_xaxis().set_tick_params(direction="out")
ax.get_yaxis().set_tick_params(direction="out")
ax.xaxis.tick_bottom()
ax.yaxis.tick_left()

# Labels
plt.title("Fit Plot for Latitude vs. Temperature", fontsize="14", fontwe
plt.xlabel("density")
plt.ylabel("gain")
plt.xlim(np.min(x)-.1,np.max(x)+.1)

# Custom legend
handles, labels = ax.get_legend_handles_labels()
display = (0, 1)
anyArtist = plt.Line2D((0,1), (0,0), color="#b9cfe7")      # create cust
legend = plt.legend(
    [handle for i, handle in enumerate(handles) if i in display] + [anyA
    [label for i, label in enumerate(labels) if i in display] + ["95% Co
    loc=9, bbox_to_anchor=(0, -0.21, 1., .102), ncol=3, mode="expand"
)
frame = legend.get_frame().set_edgecolor("0.5")

# Save Figure
#plt.tight_layout()
#plt.savefig("filename.png", bbox_extra_artists=(legend,), bbox_inches="

plt.show()
```





In [118]:
```
sns.residplot(x,y)
plt.title('Residuals of Least Squares Regression Line (Log transformed)'
```

Out[118]:
```
Text(0.5,1,'Residuals of Least Squares Regression Line (Log transforme
d)')
```

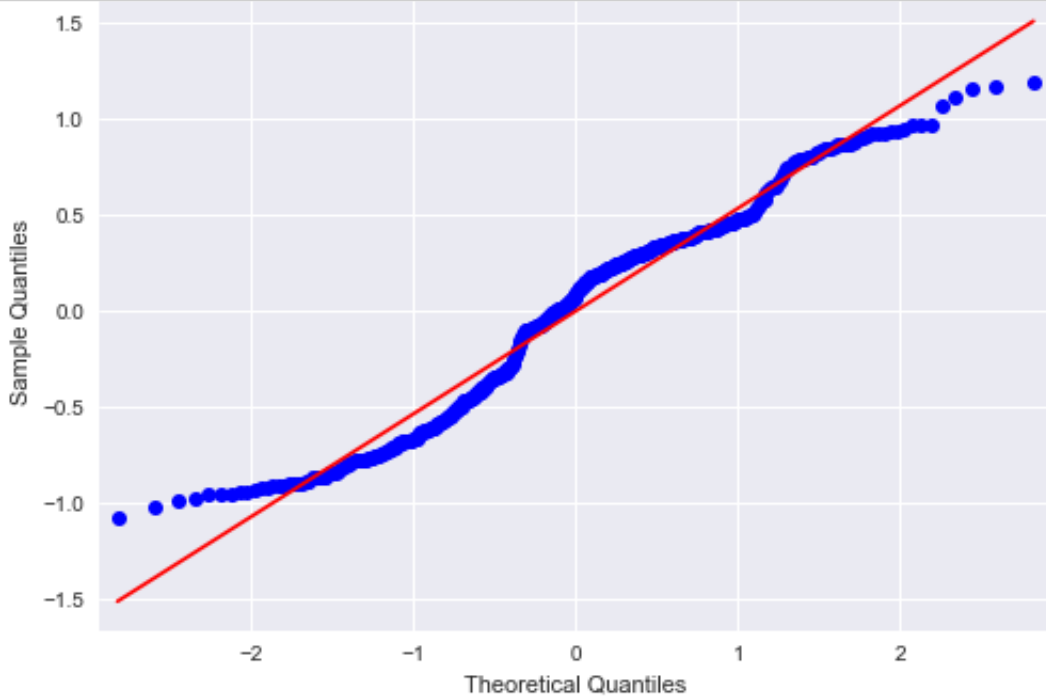


In [119]:
```
model = lambda x: x*slope + intercept
```

In [ ]:

In [125]:
```
df = X_test['Lat'].apply(model).to_frame().merge(y_test.to_frame(), left_
resid = df['Lat'] - df['Ts']
```

In [126]:
```
sm.qqplot(resid, line="s")
plt.title('QQ plot of residuals of Least Squares Regresion Line')
```



In [134]:
```
df2 = X_test['Lat'].to_frame().merge(y_test.to_frame(), left_index = Tru
df2
```

Out[134]:

	Lat	Ts
1504	66.2718	4.11
957	65.0428	6.14
1034	65.1746	6.07
240	63.9292	5.03
1555	66.3238	4.32
1935	66.0002	4.12
1636	66.0912	4.06
529	64.6012	6.29
217	63.8220	5.88
513	64.5466	6.33
292	63.9904	5.40

In [148]:
```
n = len(X_train['Lat'])
```

In [149]:
```
me = t*reg[4]*np.sqrt(1 + 1/n+(np.log(df2.loc[1504, 'Lat'])-np.mean(x))*
actual = df2.loc[1504, 'Ts']
pred = df2.loc[1504, 'Lat'] * slope + intercept
#pi
pred-me, pred+me
```

Out[149]: (4.217405301281697, 4.316423572766614)

In [ ]:

In [ ]:

In [150]:
```
actual
```

Out[150]: 4.11

In [151]:
```
me = t*reg[4]*np.sqrt(1/n+(np.log(df2.loc[1504, 'Lat'])-np.mean(x))**2/n
actual = df2.loc[1504, 'Ts']
pred = df2.loc[1504, 'Lat'] * slope + intercept
#ci
pred-me, pred+me
```

Out[151]: (4.224801854960137, 4.309027019088174)

In [144]:
```
me = t*reg[4]*np.sqrt(1 + 1/n+(np.log(df2.loc[648, 'Lat'])-np.mean(x))**
actual = df2.loc[648, 'Ts']
pred = df2.loc[648, 'Lat'] * slope + intercept
#pi
pred-me, pred+me
```

Out[144]: (5.757406797194512, 5.856458361903287)

In [145]:
```
actual
```

Out[145]: 6.16

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: