statmodel

May 31, 2023

[ ]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import statistics as stat
import scipy.stats as scip
from numpy import random as rand
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures as polyfeat
from statsmodels.graphics.gofplots import qqplot
import matplotlib as mpl

#makes the plot come out in sns format
sns.set()
#time between readings
increment=300
#read table into python and duration coloumn
table= pd.read_csv('/Users/Windows/Documents/GitHub/Moisture-Sensor_val/07.03 0 moisture/Mall.csv')
Sensor_val=np.array([table.loc[:,'M0'],table.loc[:,'M1'],table.loc[:,'M2']])
n=len(Sensor_val[0])
#time array based on time between readings
Time= np.linspace(0,increment*n)
print(n)
```

[ ]:

```python
#  Boxplot of Sensor_vals
tablemelt=table.melt()
sns.boxplot(x=tablemelt['variable'], y=tablemelt['value'], data=tablemelt)
plt.title('Boxplot of SENSOR VALUES (nm)')
plt.show()
```

[ ]:

```python
TrimMean=np.array([0.0,0.0,0.0])
StdDev=np.array([0.0,0.0,0.0])
Mean=np.array([0.0,0.0,0.0])
CI=np.array([(0.,0.),((0.,0.)),(0.,0.)])
#80% confidence interval
alpha=0.2
Z= scip.norm.ppf(1-alpha/2)
```

```
for i in range(0,3):
    TrimMean[i]= scip.trim_mean(Sensor_val[i],0.1)
    StdDev[i]= stat.stdev(Sensor_val[i])
    Mean[i]=Sensor_val[i].mean()
    #Confidence interval
    CI[i] = scip.t.interval(confidence=1-alpha, df=len(Sensor_val[i])-1,␣
 ↪loc=Mean[i], scale=StdDev[i]/np.sqrt(len(Sensor_val[i])))
   # CI[i] = t_interval[1] # we assign the upper bound of the interval to the␣
 ↪array element
print(CI)
```

[ ]:

```
print(Sensor_val[1])
```

[ ]:

```
# %% 2b Polynomial Linear Regression Model
def PolyRegress(x,y,n,plot):
    ##set bias to false so it is not automatically 0
    Poly = polyfeat(degree=n,include_bias= False)
    #using Polynomial Features as required
    PolyFeatures = Poly.fit_transform(x.reshape(-1,1))
    #fit polynomial regression model
    polymodel = LinearRegression()
    polymodel.fit(PolyFeatures, y)
    Pred=polymodel.predict(PolyFeatures)
    ##display model coefficients
    ##print(polymodel.intercept_, polymodel.coef_)
    #Number of parameters being estimated
    q=n+1
    StdDev= stat.stdev(Sensor_val)
    #Residual sum of squares
    RSS= (np.sum((y-Pred)**2))
    #Log likelihood function with Eqn from Stats book pg 150
    LL=-(len(y)*0.5*np.log(2*np.pi*(StdDev**2)))-(1/(2*StdDev**2))*RSS
    #AIC information
    AIC= 2*q-2*LL
    #Only plot if required
    if plot==True:
        plt.figure()
        plt.scatter(x,y)
        plt.plot(x,Pred,c='r',label='AIC='+str(np.round(AIC,decimals=2))+'␣
 ↪RSS='+str(np.round(RSS,decimals=4)))
        plt.xlabel("Standardised Time Index")
        plt.ylabel("Sensor_val (nm)")
        plt.ticklabel_format(useOffset=False)
        plt.title("Sensor_val against Time with a Polynomial regression of order␣
 ↪" + str(n))
        plt.legend()
        plt.show()
        print(n, AIC)
    #Return prediction array
    return Pred
```

[ ]:

```
# %% 2c Run Polynomial linear regress for different orders and standardise X
## Standardise the X variable, reason in report
StdTime= (Time-np.mean(Time))/stat.stdev(Time)
for i in range(1,7):
    MoisturePred=PolyRegress(StdTime,Sensor_val,i,True)
```

[ ]:

```
##2d checking validity of linear regression assumptions
#chosen model  based on AIC
MoisturePred= PolyRegress(Time,Sensor_val,6,False)
Residuals=Sensor_val-MoisturePred
plt.scatter(StdTime,Residuals)
plt.xlabel("Standardised Time Index")
plt.ylabel("Residual (nm)")
plt.ticklabel_format(useOffset=False)
plt.title("Residuals against Time for a Polynomial regression of order 6")
plt.show()
#do a histogram of residuals to see whether it is normally distributed
sns.histplot(Residuals,bins=20,kde=True, stat="probability")
plt.ylabel("Relative Frequency")
plt.xlabel("Residual (nm)")
plt.title('Histogram plot of Residuals')
plt.show()

#Standardise the residuals
Residuals= (Residuals-np.mean(Residuals))/stat.stdev(Residuals)
#qqPLot of the distribution of the residuals versus the Standard normal␣
 ↪distribution
qqplot(Residuals,line='45')
plt.xlabel('Theoretical')
plt.ylabel('Sample')
plt.title('Normal QQ plot of Residuals')
plt.show()
```