Scenario

A hospital is measuring the heights of admitted patients.

The data of such measurements is located in data.csv. There are 3 columns. The first column is the date. Second column contains the number of patients admitted on such date. The third column is the average height of the admitted patients for that day.

Task

1) How can one retrospectively identify outliers? In other words, having seen the data for the past ten months, can we identify erroneous values? Please create code to test such.

Answer: Yes

2) How can we proactively identify outliers? For example, pretend today is 2020-02-01, and you have observed data for a month. How can you check whether the values computed on 2020-02-01 are erroneous? Please write code that can work in a proactive manner.

Answer: Yes

Answer Explained:

Background:
The given dataset is a time-series dfataset with two main features:

1. "average_height"
2. "num_admissions"
   The "average_height" feature was ruled out of the prediction for outliers for the following reason: Although a patient's individual height is from a normal distribution, an "average_height" was not correlated in any way to the number of admissions of the patients or to the day on which they were admitted. Just for information, I studied the SD of the average_heights and it was also a narrow distribution: 0.9.

We know that the "num_admissions" was drawn from a linear distribution of slope, "3" and so is correlated linearly to the day of the admission calculated from the start of the year. That is to say, the patients admitted are inclreasing linearly as the year progresses.

Methodology used:
Step 1: Calculate the "delta" number of days from the first entry data
Step 2: Harvest X as the delta days, and y as the num_admissions
Step 3: Fit a linear regression model through the given data (X and y)
Step 4: Clean outliers enough to get as close a match to slope, '3' - done by calculating the squared errors and eliminating the top 7% of the data
Step 5: Visualize the results for information
Step 6: Predict outliers: any point predicted one RMSD above or below the fitted line
Step 6a. Calcute the SD of the residuals
Step 6b. Check if the given date and "num_admissions" reading:
is an outiler - predicted value of instance falls outside 1 RMSD of the line
not an outlier - predicted value of instance falls within 1 RMSD of the line

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.dates import (MONTHLY, DateFormatter,
                              rrulewrapper, RRuleLocator)
import seaborn as sns
%matplotlib widget
```

In [4]:
```
data = pd.read_csv("quality_data.csv", parse_dates = ["date"])
# Sort data
data.sort_values("date", inplace=True)# pos = index position of the 'd
ate' column
#### Use timedelta to calculate time from start of file
pos = data.columns.get_loc('date')
data['delta'] = (data.iloc[1:, pos] - data.iat[0,pos])
data.loc[0,'delta'] = pd.Timedelta(days = 0)
# Convert 'delta' to integer # days
data['delta'] = data['delta'] / np.timedelta64(1, 'D')
data['delta'] = data['delta'].astype('int')
```

In [40]:
```
# fig, ax = plt.subplots()
# rule = rrulewrapper(MONTHLY)
# loc = RRuleLocator(rule)
# formatter = DateFormatter('%Y-%m-%d')
# ax.xaxis.set_major_locator(loc)
# ax.xaxis.set_major_formatter(formatter)
# ax.tick_params(axis='x', labelrotation=45)
# plt.plot(data['date'], data['num_admissions'])
```

Study "num_admissions"

outliers for num_admissions using scikit learn:

In [6]:
```
def clean_outliers(predictions, delta, num_admissions):
    """
        clean away the 7% of points that have the largest
        residual errors (different between the prediction
        and the actual net worth)

        return a two tuples named outliers and cleaned_data where:
        outliers includes the top 7% of predictions with largest squar
e-errors, and
        cleaned_data includes tuples of the form (delta,num_admission
s,errors)
    """

    # calculate the residual error, descend sort, and harvest 93% of t
he data
    # for the best fit to achieving a slope closest to '3'

    errors = (num_admissions-predictions)**2
    cleaned_data = zip(delta,num_admissions,errors)
    cleaned_data = sorted(cleaned_data,key=lambda x:x[2], reverse=Tru
e)
    limit = int(len(num_admissions)*0.07)
    outliers = cleaned_data[:limit]
    return outliers, cleaned_data[limit:]
```

In [7]:
```
# Linear Regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import mean_absolute_error, mean_squared_error
lreg = LinearRegression()
plt.close("all")
# split into input and output elements
# Trying feature X as # days as stored in 'delta' column
# Trying target, y, as num_admissions
delta_pos = data.columns.get_loc("delta")
adm_pos = data.columns.get_loc("num_admissions")
X, y = data.iloc[:,delta_pos].values, data.iloc[:, adm_pos].values

lreg.fit(X.reshape(-1,1),y)
predictions = lreg.predict(X.reshape(-1,1))
plt.figure()
# plot the predicted line
try:
    plt.plot(X, predictions, color="blue")
except Exception as e:
    pass
# Scatter plot of original num_admission values
plt.scatter(X, y)
plt.show()
# evaluate predictions
mae = mean_absolute_error(y, predictions)
print(f"With Outliers MAE: {mae:.3f}")
mse = mean_squared_error(y, predictions)
print(f'With Outliers  MSE:{mse:.3f}')
print(f"With Outliers linear model coeff : {lreg.coef_}")
print(f"With Outliers linear model intercept: {lreg.intercept_}")
print('With Outliers R-squared score (training): {:.3f}'
     .format(lreg.score(X.reshape(-1,1), y)))

# # Find outliers #
# ### identify and remove the most outlier-y points
cleaned_data = []
(outliers, cleaned_data) = clean_outliers( predictions, X, y )
### Check regression with the new cleaned data
if len(cleaned_data) > 0:
    delta, num_admissions, errors = zip(*cleaned_data)
    delta = np.reshape( np.array(delta), (len(delta), 1))
    num_admissions = np.reshape( np.array(num_admissions), (len(num_ad
missions), 1))

    ## refit the cleaned data!
    lreg.fit(delta.reshape(-1,1), num_admissions.reshape(-1,1))
    clean_pred = lreg.predict(delta)
    plt.plot(delta, clean_pred, color="orange")
    plt.scatter(delta, num_admissions, color="orange")
    #plt.scatter(delta, num_admissions)
    plt.xlabel("Delta Days")
    plt.ylabel("# Admissions")
    plt.show()
```

```
    #print(f"\n# Outliers: {len(X) - len(cleaned_data)}\n")
    print(f"\n# Outliers: {len(list(outliers))}\n")
    print(f"{outliers}\n")
    # evaluate predictions
    mae = mean_absolute_error(num_admissions, clean_pred)
    print(f"Without Outliers MAE: {mae:.3f}")
    mse = mean_squared_error(num_admissions, clean_pred)
    print(f'Without Outliers MSE: {mse:.3f}')
    print(f"Without Outliers linear model coeff : {lreg.coef_}")
    print(f"Without Outliers linear model intercept: {lreg.intercept
_}")
    print(f'Without Outliers R-squared score: {lreg.score(delta, num_a
dmissions):.3f}')
else:
    print("no outliers, no refitting to be done")
```

```
With Outliers MAE: 73.197
With Outliers  MSE:8374.497
With Outliers linear model coeff : [2.90930478]
With Outliers linear model intercept: 15.034200743494239
With Outliers R-squared score (training): 0.859

# Outliers: 18

[(160, 731, 62738.745012220745), (170, 760, 62692.14100989372), (150,
207, 59745.98448490394), (242, 480, 57162.094754720456), (199, 827, 5
4295.593442151076), (9, 262, 48744.71636703137), (79, 29, 46599.54524
6667636), (134, 618, 45419.690731194976), (115, 143, 42685.3161696476
4), (54, 376, 41560.26192034695), (86, 62, 41304.22605238111), (151,
255, 39736.12546946159), (175, 339, 34285.16501887485), (241, 898, 33
059.72985767152), (226, 497, 30813.266594397737), (46, 322, 29976.690
69156439), (184, 381, 28678.162466400885), (51, 331, 28086.8289595580
83)]

Without Outliers MAE: 63.515
Without Outliers MSE: 5827.038
Without Outliers linear model coeff : [[2.95008539]]
Without Outliers linear model intercept: [9.3579011]
Without Outliers R-squared score: 0.901
```

Print outliers

In [21]:
```
# calculate RMSD ot SD of residuals
import math
# use the fitted lreg model from above
deltas = list(zip(*outliers))[0]
res_sq = list(zip(*cleaned_data))[2]
clean_df = data[~data['delta'].isin(list(deltas))]
outliers = data[data['delta'].isin(list(deltas))]
sd_res = math.sqrt(sum(res_sq)/(len(clean_df) - 2))
```

In [42]:
```
def proactive_outlier_test(x, y, w, b, sd_res):
    yhat = w  * x + b
    res = abs(y - yhat)
    if res > sd_res:
        print("Outlier")
        return True
    else:
        print("Not an outlier")
        return False
```

In [43]:
```
# 'new_num_admissions' are the patients admitted for the given day, 'n
ew_date'
new_date = pd.Timestamp(2020, 11, 8)
new_delta = (new_date - pd.Timestamp(2020, 1, 1)) / np.timedelta64(1,
'D')
new_num_admissions = 300
w = lreg.coef_
b = lreg.intercept_
proactive_outlier_test(x =  new_delta, y = new_num_admissions, w = w,
b = b, sd_res = sd_res)
```

```
Outlier
```

Out[43]:
```
True
```

In [44]:
```
# 'new_num_admissions' are the patients admitted for the given day, 'n
ew_date'
new_date = pd.Timestamp(2020, 2, 1)
new_delta = (new_date - pd.Timestamp(2020, 1, 1)) / np.timedelta64(1,
'D')
new_num_admissions = 53
w = lreg.coef_
b = lreg.intercept_
proactive_outlier_test(x =  new_delta, y = new_num_admissions, w = w,
b = b, sd_res = sd_res)
```

```
Not an outlier
```

Out[44]:
```
False
```

In [45]:
```
# 'new_num_admissions' are the patients admitted for the given day, 'n
ew_date'
new_date = pd.Timestamp(2020, 2, 21)
new_delta = (new_date - pd.Timestamp(2020, 1, 1)) / np.timedelta64(1,
'D')
new_num_admissions = 331
w = lreg.coef_
b = lreg.intercept_
proactive_outlier_test(x =  new_delta, y = new_num_admissions, w = w,
b = b, sd_res = sd_res)
```

```
Outlier
```

Out[45]:
```
True
```

In [ ]: