```
In [4]:  import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
```

```
In [5]:  ## change font for plots
         import matplotlib
         from matplotlib import rc
         matplotlib.rcParams['pdf.fonttype'] = 42 # to import text in illustrator
         matplotlib.rcParams['ps.fonttype'] = 42
         rc('font',**{'family':'serif','serif':['Arial']})

         SMALL_SIZE = 8
         MEDIUM_SIZE = 10

         plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
         plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
         plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
         plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
         plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
```

```
In [6]:  ## define cm2inch for plot size
         def cm2inch(*tupl): # cm to inch for figure size
             inch = 2.54
             if isinstance(tupl[0], tuple):
                 return tuple(i/inch for i in tupl[0])
             else:
                 return tuple(i/inch for i in tupl)
```

Time-Depth-Conversion

Here, we use multiple wells to calculate the time-depth conversion. For highest accuracy, we use different wells for sill S1 and S2.

```
In [7]:  ## import checkshot data from excel sheet
         input_path   = '/Users/jkop0001/Documents/PhD/4_1seismic/4_seismic_backup/wells/check

         toporoa      = pd.read_excel(input_path, sheet_name=0)
         briseis      = pd.read_excel(input_path, sheet_name=1)
         glencoe1     = pd.read_excel(input_path, sheet_name=2)
         chester1_st1 = pd.read_excel(input_path, sheet_name=4)
         makybe_diva1 = pd.read_excel(input_path, sheet_name=5)
         hijinx1      = pd.read_excel(input_path, sheet_name=6)
         lightfinger1 = pd.read_excel(input_path, sheet_name=7)
         nimblefoot   = pd.read_excel(input_path, sheet_name=8)
```

```
In [8]:  ## merge wells for specific areas
         # S1
         MD_S1  = -np.concatenate((chester1_st1['MD'].values, makybe_diva1['MD'].values,
                                  hijinx1['MD'].values, lightfinger1['MD'].values, nimblefoot['
         TWT_S1 = np.concatenate((chester1_st1['TWT'].values, makybe_diva1['TWT'].values,
                                  hijinx1['TWT'].values, lightfinger1['TWT'].values, nimblefoot

         # S2
         MD_S2  = -np.concatenate((toporoa['MD'].values, briseis['MD'].values, glencoe1['MD'].
         TWT_S2 = np.concatenate((toporoa['TWT'].values, briseis['TWT'].values, glencoe1['TWT'
```

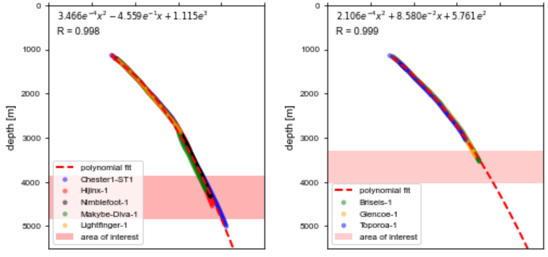
```
In [9]:  ## polynomial regression
         # S1
         p_S1  = np.polyfit(TWT_S1, MD_S1, 2, full=True)
         MD_S1_fit = np.polyval(p_S1[0], TWT_S1)

         # S2
         p_S2  = np.polyfit(TWT_S2, MD_S2, 2, full=True)
         MD_S2_fit = np.polyval(p_S2[0], TWT_S2)
```

```
In [10]: ## sort data
         # S1
         TWT_S1_sort = sorted(TWT_S1)
         MD_S1_fit_sort = sorted(MD_S1_fit, reverse=True)
         MD_S1_sort = sorted(MD_S1, reverse=True)

         # S2
         TWT_S2_sort = sorted(TWT_S2)
         MD_S2_fit_sort = sorted(MD_S2_fit, reverse=True)
         MD_S2_sort = sorted(MD_S2, reverse=True)
```

```
In [11]: ## fit accuracy
         # S1
         # correlation coefficient
         correlation_matrix = np.corrcoef(MD_S1_fit_sort, MD_S1_sort) # compare MD1_sort (dept
         correlation_xy = correlation_matrix[0,1]
         r_squared_S1 = correlation_xy**2

         # Root Mean Square Error (RMSE)
         RMSE_S1 = np.sqrt(((MD_S1 - MD_S1_fit) ** 2).mean())

         # S2
         # correlation coefficient
         correlation_matrix = np.corrcoef(MD_S2_fit_sort, MD_S2_sort) # compare MD1_sort (dept
         correlation_xy = correlation_matrix[0,1]
         r_squared_S2 = correlation_xy**2

         # Root Mean Square Error (RMSE)
         RMSE_S2 = np.sqrt(((MD_S2 - MD_S2_fit) ** 2).mean())
```

```
In [12]: ## calculate time-depth conversion from 1000-5500 [m] depth

         TWT      = np.linspace(1500, 8000, 1500)
         depth_S1 = np.polyval(p_S1[0], TWT)
         depth_S2 = np.polyval(p_S2[0], TWT)
```

```
In [13]: ## plot
         fig1, axs = plt.subplots(1,2, figsize=cm2inch(18.5,10))

         # S1
         ax=axs[0]
         ax.scatter(chester1_st1['TWT'].values, chester1_st1['MD'].values,
                    facecolor='b', marker = 'o', s=20, alpha=0.5, edgecolors='none', label='Che
         ax.scatter(hijinx1['TWT'].values, hijinx1['MD'].values,
                    facecolor='r', marker = 'o', s=20, alpha=0.5, edgecolors='none', label='Hi
         ax.scatter(nimblefoot['TWT'].values, nimblefoot['MD'].values,
                    facecolor='k', marker = 'o', s=20, alpha=0.5, edgecolors='none', label='Ni
         ax.scatter(makybe_diva1['TWT'].values, makybe_diva1['MD'].values,
                    facecolor='g', marker = 'o', s=20, alpha=0.5, edgecolors='none', label='Ma
         ax.scatter(lightfinger1['TWT'].values, lightfinger1['MD'].values,
                    facecolor='orange', marker = 'o', s=20, alpha=0.5, edgecolors='none', labe

         ax.plot(TWT, -depth_S1, 'r--', lw=2, label = 'polynomial fit')

         ax.fill_between(np.array((0,5000)), 3865, 4838,
                         color='r',linewidth=0, alpha=0.3, zorder=0, label='area of interest')


         # annotation
         ax.text(200, 350, '$ 3.466e^{-4}x^2 - 4.559e^{-1}x + 1.115e^3 $', fontsize=8)
         ax.text(200, 650, 'R = ' + str("%.3f" % r_squared_S1), fontsize=9)

         ax.set_ylim(0,5500)
         ax.set_xlim(0,5000)

         ax.set(ylabel='depth [m]')
         ax.set(xlabel='TWT [ms]')
         ax.set_title('a. Sill 1', size=12, x=0.1) # Title
         ax.legend(loc='lower left')

         ax.set_ylim(ax.get_ylim()[::-1])        # invert the axis
         ax.xaxis.tick_top()                     # and move the X-Axis
         ax.xaxis.set_label_position("top")




         # S2
         ax=axs[1]

         ax.scatter(briseis['TWT'].values, briseis['MD'].values,
                    facecolor='g', marker = 'o', s=20, alpha=0.5, edgecolors='none', label='Br
         ax.scatter(glencoe1['TWT'].values, glencoe1['MD'].values,
                    facecolor='orange', marker = 'o', s=20, alpha=0.5, edgecolors='none', labe
         ax.scatter(toporoa['TWT'].values, toporoa['MD'].values,
                    facecolor='b', marker = 'o', s=20, alpha=0.5, edgecolors='none', label='To

         ax.plot(TWT, -depth_S2, 'r--', lw=2, label = 'polynomial fit')

         ax.fill_between(np.array((0,5000)), 3303, 4029,
                         color='r',linewidth=0, alpha=0.2, zorder=0, label='area of interest')

         # annotation
         ax.text(200, 350, '$ 2.106e^{-4}x^2 + 8.580e^{-2}x + 5.761e^2 $', fontsize=8)
         ax.text(200, 650, 'R = ' + str("%.3f" % r_squared_S2), fontsize=9)

         ax.set_ylim(0,5500)
         ax.set_xlim(0,5000)

         ax.set(xlabel='TWT [ms]')
         ax.set(ylabel='depth [m]')
         ax.set_title('b. Sill 2', size=12, x=0.1) # Title
         ax.legend(loc='lower left')

         ax.set_ylim(ax.get_ylim()[::-1])        # invert the axis
         ax.xaxis.tick_top()                    # and move the X-Axis
         ax.xaxis.set_label_position("top")

         fig1.tight_layout()
```



Quantifying vertical offsets between neighbouring elements

Create arrays with vertical offsets:

[ID, max (or) min offset [m], element length at offset [m], max element length [m], length / max length]

Note that the element length is measured from the most inward part of the connector (0 m) towards the propagating tip (max element length).

In [14]:

```
import_path = '//Users/jkop0001/Documents/PhD/thesis-chapter/1_seismic/Geosphere/v2/su
save_path = '/Users/jkop0001/Documents/PhD/4_1seismic/4_seismic_backup/data_paper/inpu
save = 0  # save file? 0 = no, 1 = yes
```

In [15]:

```
## batch process

v_offs_max = np.zeros((1,8)) # [ID, max (or) min offset [m], error1, error2, elements
v_offs_min = np.zeros((1,8)) # [ID, max (or) min offset [m], error1, error2, elements

## import file
xl = pd.ExcelFile(import_path)
res = len(xl.sheet_names) # number of sheets

for k in range(0,res):

    ## import excel files
    data    = pd.read_excel(import_path, sheet_name=k)
    length  = data['length'].values # length [m]
    connect = data['connected'].values # are both elements connected? [0 = no; 1 = ye
    fault   = data['fault'].values # pre-existing fault between element? [0 = no; 1 =
    TWT_top = data['top_TWT'].values # top TWT measurement [ms]
    TWT_bot = data['bottom_TWT'].values # bottom TWT measurement [ms]

    ## calculate vertical offset TWT of upper and lower element (based on time-depth 
    E1      = np.zeros_like(TWT_top) # empty array --> depth [m] of upper element
    E1_err1 = np.zeros_like(E1) # empty array --> estimate error of E1 --> TWT-2ms
    E1_err2 = np.zeros_like(E1) # empty array --> estimate error of E1 --> TWT+2ms
    E2      = np.zeros_like(TWT_top) # empty array --> depth [m] of lower element
    E2_err1 = np.zeros_like(E2) # empty array --> estimate error of E2 --> TWT-2ms
    E2_err2 = np.zeros_like(E2) # empty array --> estimate error of E2 --> TWT+2ms
    v_offs  = np.zeros_like(TWT_top) # empty array to store vertical offset measureme

    err_TWT = 4 # [ms]; +- err_TWT to TWT measurements

    for j in range(0,len(TWT_top)):
        E1[j]      = np.abs(p_S1[0][0] * TWT_top[j]**2 + p_S1[0][1] * TWT_top[j] + p_S
        E1_err1[j] = np.abs(p_S1[0][0] * (TWT_top[j]-err_TWT)**2 + p_S1[0][1] * (TWT_t
        E1_err2[j] = np.abs(p_S1[0][0] * (TWT_top[j]+err_TWT)**2 + p_S1[0][1] * (TWT_t

        E2[j]      = np.abs(p_S1[0][0] * TWT_bot[j]**2 + p_S1[0][1] * TWT_bot[j] + p_S
        E2_err1[j] = np.abs(p_S1[0][0] * (TWT_bot[j]-err_TWT)**2 + p_S1[0][1] * (TWT_b
        E2_err2[j] = np.abs(p_S1[0][0] * (TWT_bot[j]+err_TWT)**2 + p_S1[0][1] * (TWT_b

    v_offs = E2-E1 # vertical offset measurements [m]


    ## sum up element's length
    length_tot = np.zeros(len(length))
    for i in range(0,len(length)):
        if i == 0:
            length_tot[i] = length[i]
        else:
            length_tot[i] = length_tot[i-1]+length[i]

    ## min & max offsets
    max_voffs = np.amax(v_offs)
    min_voffs = np.amin(v_offs)


    ## position of min & max offset
    ind1 = np.where(v_offs == max_voffs)    # ind max vertical offset
    ind2 = np.where(v_offs == min_voffs)    # ind min vertical offset


    ## estimate error
    # max offset
    max_err1 = E2_err1[ind1] - E1_err2[ind1] # lower boundary --> smaller offset
    max_err2 = E2_err2[ind1] - E1_err1[ind1] # upper boundary --> larger offset

    # min offset
    min_err1 = E2_err1[ind2] - E1_err2[ind2] # lower boundary --> smaller offset
    min_err2 = E2_err2[ind2] - E1_err1[ind2] # upper boundary --> larger offset



    ## max length at max & min offset
    max_voffs_l = length_tot[ind1]
    min_voffs_l = length_tot[ind2]

    max_voffs_rl = max_voffs_l/length_tot[-1]
    min_voffs_rl = min_voffs_l/length_tot[-1]



    ## create arrays with data

    # max vertical offset
    v_offs_max1 = np.ones((len(ind1[0]),8))

    for j in range(0,len(ind1[0])):
        v_offs_max1[j][0] = k
        v_offs_max1[j][1] = max_voffs                         # max vertical offset
        v_offs_max1[j][2] = np.abs(max_voffs-max_err1[j])     # estimated error; lower
        v_offs_max1[j][3] = np.abs(max_voffs-max_err2[j])     # estimated error; upper
        v_offs_max1[j][4] = max_voffs_l[j]                    # element length at max 
        v_offs_max1[j][5] = length_tot[-1]                    # max length of the elem
        v_offs_max1[j][6] = max_voffs_l[j] / length_tot[-1]   # length / max length
        v_offs_max1[j][7] = fault[0]                          # fault; 0 = no, 1 = yes

    v_offs_max = np.append(v_offs_max, v_offs_max1, axis=0)


    # min vertical offset
    v_offs_min1 = np.ones((len(ind2[0]),8))

    for j in range(0,len(ind2[0])):
        v_offs_min1[j][0] = k
        v_offs_min1[j][1] = min_voffs                         # min vertical offset
        v_offs_min1[j][2] = np.abs(min_voffs-min_err1[j])     # estimated error; lower
        v_offs_min1[j][3] = np.abs(min_voffs-min_err2[j])     # estimated error; upper
        v_offs_min1[j][4] = min_voffs_l[j]                    # segment length at min 
        v_offs_min1[j][5] = length_tot[-1]                    # total length of the seg
        v_offs_min1[j][6] = min_voffs_l[j] / length_tot[-1]   # length / total length
        v_offs_min1[j][7] = fault[0]                          # fault; 0 = no, 1 = yes

    v_offs_min = np.append(v_offs_min, v_offs_min1, axis=0)

v_offs_max = np.delete(v_offs_max, (0), axis=0)
v_offs_min = np.delete(v_offs_min, (0), axis=0)
```

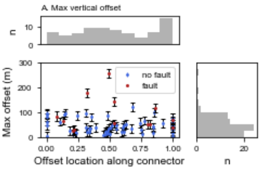
Fig. 7 - Plot maximum and minimum vertical offsets vs. the offset location along the connector

Note that a location of 0 represents the most inward part of the connector, whereas 1 represents the tip of the connector.

In [18]:

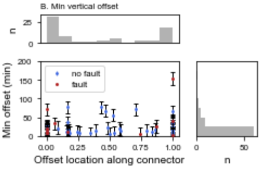
```
## maximum vertical offset
## separate dataset: fault/no fault
ind_f  = np.where(v_offs_max[:,7] == 1) # entries with fault
ind_nf = np.where(v_offs_max[:,7] == 0) # entries without fault

y_error_f  = [v_offs_max[ind_f,2][0],v_offs_max[ind_f,3][0]]   # error faults
y_error_nf = [v_offs_max[ind_nf,2][0],v_offs_max[ind_nf,3][0]] # error no faults

m = ['d', '.'] # marker
ms = [5, 15] # marker size
colour = ['royalblue', 'firebrick', 'gray'] # marker colour


## figure
fig = plt.figure(figsize=cm2inch(9,6))

scatter_axes = plt.subplot2grid((3, 3), (1, 0), rowspan=2, colspan=2)
x_hist_axes = plt.subplot2grid((3, 3), (0, 0), colspan=2)
y_hist_axes = plt.subplot2grid((3, 3), (1, 2), rowspan=2)

x_hist_axes.set_title('A. Max vertical offset', loc='left')

# plot scatter and error bars of connectors without faults
scatter_axes.scatter(v_offs_max[ind_nf,6],v_offs_max[ind_nf,1], zorder = 1, c=colour[
scatter_axes.errorbar(v_offs_max[ind_nf,6][0], v_offs_max[ind_nf,1][0], yerr=y_error_
            fmt='none', ecolor='k', elinewidth=1, capsize=2, capthick=1, barsabove=Fa

# plot scatter and error bars of connectors with faults
scatter_axes.scatter(v_offs_max[ind_f,6],v_offs_max[ind_f,1], zorder = 3, c=colour[1],
scatter_axes.errorbar(v_offs_max[ind_f,6][0], v_offs_max[ind_f,1][0], yerr=y_error_f,
            fmt='none', ecolor='k', elinewidth=1, capsize=2, capthick=1, barsabove=Fa

# histograms
x_hist_axes.hist(v_offs_max[:,6], bins=10, range=(0,1), color=colour[2], alpha=0.6)
y_hist_axes.hist(v_offs_max[:,1], bins=range(0, 300, 25), orientation='horizontal', co
```

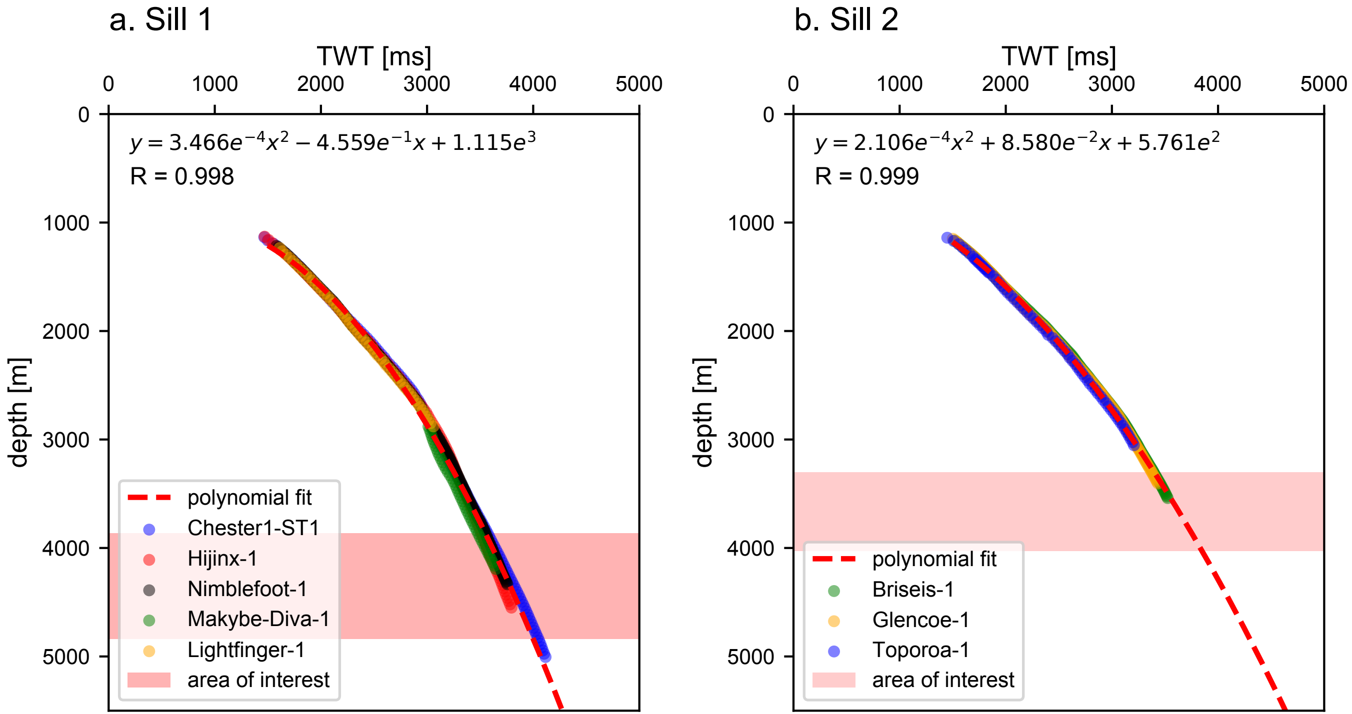
```
# set axes, labels, etc.
scatter_axes.set_ylim(0, 300)
y_hist_axes.set_ylim(0, 300)

scatter_axes.set(ylabel='Max offset (m)')
scatter_axes.set(xlabel='Offset location along connector')
x_hist_axes.set(ylabel='n')
y_hist_axes.set(xlabel='n')

x_hist_axes.axes.get_xaxis().set_ticks([])
y_hist_axes.axes.get_yaxis().set_ticks([])

scatter_axes.legend(loc='upper right')

fig.tight_layout()

if save == 1:
    plt.savefig(save_path + 'sill1_max_voffs4ms_err.pdf', format='pdf')
```



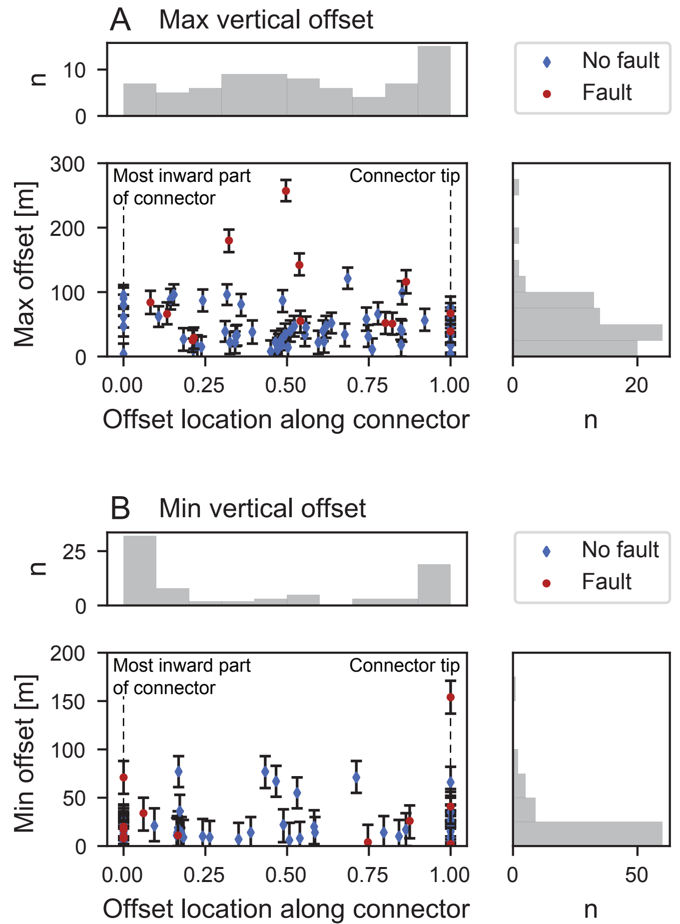
In [19]:

```
## minimum vertical offset
## separate dataset: fault/no fault
ind_f  = np.where(v_offs_min[:,7] == 1) # entries with fault
ind_nf = np.where(v_offs_min[:,7] == 0) # entries without fault

y_error_f  = [v_offs_min[ind_f,2][0],v_offs_min[ind_f,3][0]]   # error faults
y_error_nf = [v_offs_min[ind_nf,2][0],v_offs_min[ind_nf,3][0]] # error no faults

m = ['d', '.'] # marker
ms = [5, 15] # marker size
colour = ['royalblue', 'firebrick', 'gray'] # marker colour

## figure
fig = plt.figure(figsize=cm2inch(9,6))

scatter_axes = plt.subplot2grid((3, 3), (1, 0), rowspan=2, colspan=2)
x_hist_axes = plt.subplot2grid((3, 3), (0, 0), colspan=2)
y_hist_axes = plt.subplot2grid((3, 3), (1, 2), rowspan=2)

x_hist_axes.set_title('B. Min vertical offset', loc='left')

# plot scatter and error bars of connectors without faults
scatter_axes.scatter(v_offs_min[ind_nf,6],v_offs_min[ind_nf,1], zorder = 1, c=colour[(
scatter_axes.errorbar(v_offs_min[ind_nf,6][0], v_offs_min[ind_nf,1][0], yerr=y_error_r
            fmt='none', ecolor='k', elinewidth=1, capsize=2, capthick=1, barsabove=Fal

# plot scatter and error bars of connectors with faults
scatter_axes.scatter(v_offs_min[ind_f,6],v_offs_min[ind_f,1], zorder = 3, c=colour[1],
scatter_axes.errorbar(v_offs_min[ind_f,6][0], v_offs_min[ind_f,1][0], yerr=y_error_f,
            fmt='none', ecolor='k', elinewidth=1, capsize=2, capthick=1, barsabove=Fal

# histograms
x_hist_axes.hist(v_offs_min[:,6], bins=10, range=(0,1), color=colour[2], alpha=0.6)
y_hist_axes.hist(v_offs_min[:,1], bins=range(0, 200, 25), orientation='horizontal', cc

# set axes limits, labels, etc.
scatter_axes.set_ylim(0,200)
y_hist_axes.set_ylim(0, 200)

scatter_axes.set(ylabel='Min offset (min)')
scatter_axes.set(xlabel='Offset location along connector')
x_hist_axes.set(ylabel='n')
y_hist_axes.set(xlabel='n')

x_hist_axes.axes.get_xaxis().set_ticks([])
y_hist_axes.axes.get_yaxis().set_ticks([])

scatter_axes.legend(loc='upper left')

fig.tight_layout()

if save == 1:
    plt.savefig(save_path + 'sill1_min_voffs_4ms_err.pdf', format='pdf')
```





a. Sill 1
TWT [ms]
0
1000
2000
3000
4000
5000
0
1000
2000
3000
4000
5000
depth [m]
$y = 3.466e^{-4}x^2 - 4.559e^{-1}x + 1.115e^3$
R = 0.998
polynomial fit
Chester1-ST1
Hijinx-1
Nimblefoot-1
Makybe-Diva-1
Lightfinger-1
area of interest
b. Sill 2
TWT [ms]
0
1000
2000
3000
4000
5000
0
1000
2000
3000
4000
5000
depth [m]
$y = 2.106e^{-4}x^2 + 8.580e^{-2}x + 5.761e^2$
R = 0.999
polynomial fit
Briseis-1
Glencoe-1
Toporoa-1
area of interest



A
Max vertical offset
n
No fault
Fault
Max offset [m]
Most inward part of connector
Connector tip
Offset location along connector
n
0.00
0.25
0.50
0.75
1.00
B
Min vertical offset
n
No fault
Fault
Min offset [m]
Most inward part of connector
Connector tip
Offset location along connector
n