Objectives

Visualize the database

- Tg distribution
- char length distribution

In [1]:

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
polymer_df = pd.read_csv("./data/polymer_tg.txt", delim_whitespace=True)
```

In [3]:

```
polymer_df
```

Out[3]:

	SMILES	Tg-celsius
0	[*]C[*]	-63.48
1	[*]CC([*])C	-2.73
2	[*]CC([*])CC	-22.54
3	[*]CC([*])CCC	-32.29
4	[*]CC([*])C(C)C	10.97
...	...	...
7227	[*]CC(F)(F)C1(F)CC(C(O)(C(F)(F)F)C(F)(F)F)CC1[*]	118.00
7228	[*]c1ccc2c(c1)C(CCCCCC)(CCCCCC)c1cc(-c3ccc4c(c...	161.00
7229	[*]c1ccc2c(c1)C(CCCCCC)(CCCCCC)c1cc(-c3ccc4c(c...	142.00
7230	[*]CC([*])(F)C(=O)OCC(Cl)(Cl)Cl	127.00
7231	[*]C(OCOC([*])C([*])(F)F)C([*])(F)F	122.00

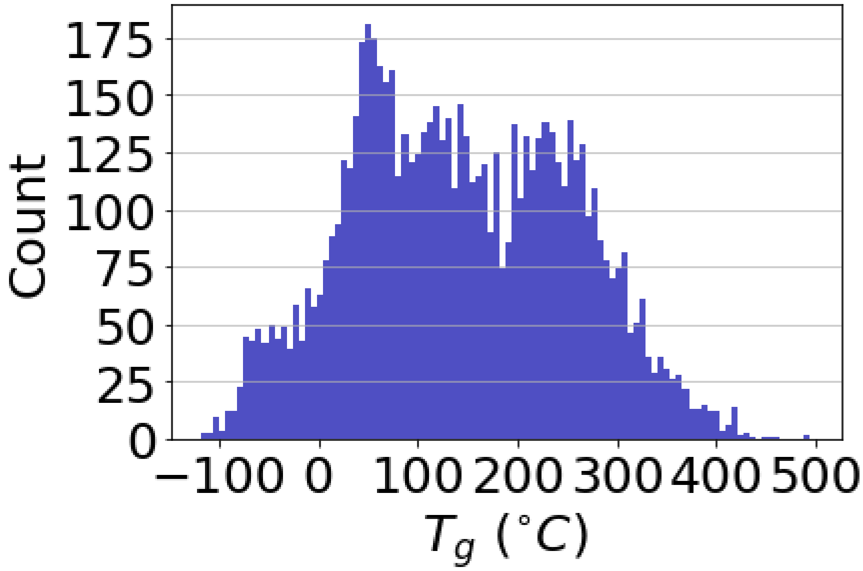
7232 rows × 2 columns

In [4]:

```
labelsize = 25
ticksize = 25
titlesize= 25

plt.figure(figsize=[12,8])
polymer_df.hist(column="Tg-celsius", grid=False, bins=100, color='#0504aa',alpha=0.7)

plt.grid(axis='y', alpha=0.75)
plt.xlabel('$T_g$ $(^{\circ}C)$',fontsize=labelsize)
plt.ylabel('Count',fontsize=labelsize)
plt.xticks(fontsize=ticksize)
plt.yticks(fontsize=ticksize)
plt.title(label="")
# plt.savefig('Tg.png',dpi=1000,bbox_inches='tight')
```

Out[4]:

```
Text(0.5, 1.0, '')

<Figure size 864x576 with 0 Axes>
```



In [5]:

```python
# add char list and corresponding length
char_list = []
len_list = []

for ismi in polymer_df.SMILES:
    char_list.append(list(ismi))
    len_list.append(len(list(ismi)))
```

In [6]:

```python
len(char_list)
```

Out[6]:

```
7232
```

In [7]:

```python
len(len_list)
```

Out[7]:

```
7232
```

In [8]:

```python
polymer_df['CharList'] = char_list
```

In [9]:

```python
polymer_df['CharLen'] = np.array(len_list)
```

In [10]:

```python
polymer_df.head()
```

Out[10]:

	SMILES	Tg-celsius	CharList	CharLen
0	[*]C[*]	-63.48	[[, *,], C, [, *,]]	7
1	[*]CC([*])C	-2.73	[[, *,], C, C, (, [, *,],), C]	11
2	[*]CC([*])CC	-22.54	[[, *,], C, C, (, [, *,],), C, C]	12
3	[*]CC([*])CCC	-32.29	[[, *,], C, C, (, [, *,],), C, C, C]	13
4	[*]CC([*])C(C)C	10.97	[[, *,], C, C, (, [, *,],), C, (, C,), C]	15

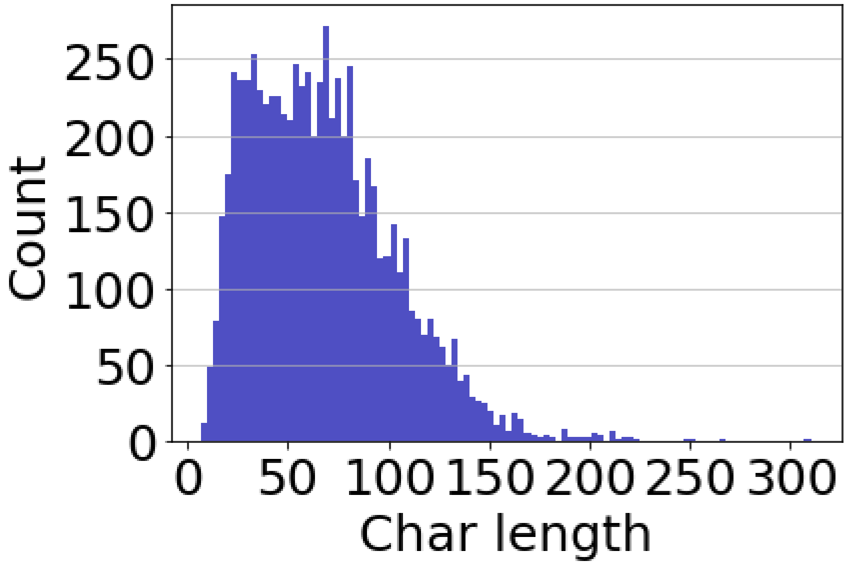
In [11]:

```
labelsize = 25
ticksize = 25
titlesize= 25

plt.figure(figsize=[12,8])
polymer_df.hist(column="CharLen", grid=False, bins=100, color='#0504aa',alpha=0.7)

plt.grid(axis='y', alpha=0.75)
plt.xlabel('Char length',fontsize=labelsize)
plt.ylabel('Count',fontsize=labelsize)
plt.xticks(fontsize=ticksize)
plt.yticks(fontsize=ticksize)
plt.title(label="")
# plt.savefig('CharLen.png',dpi=1000,bbox_inches='tight')
```

Out[11]:

```
Text(0.5, 1.0, '')
```

```
<Figure size 864x576 with 0 Axes>
```



In [12]:

```
def GetRatio(critical):


    new_df = polymer_df[polymer_df["CharLen"] < critical]
    ratio = new_df.shape[0]/polymer_df.shape[0]

    return ratio
```

In [13]:

```
print(GetRatio(150))
```

```
0.9795353982300885
```

In [14]:

```
print(GetRatio(100))
```

```
0.8210730088495575
```

In [15]:

```
print(GetRatio(120))
```

```
0.912195796460177
```

Use the value of 120 as the crytical length.

Next steps:

- tokenize the SMILES forms for all polymers
- truncate those that have longer length than 120

In [16]:

```
# get the unique tokens

tokens = list(set(''.join(polymer_df.SMILES)))
```

```
In  [17]: tokens = list(np.sort(tokens))
          tokens
```

```
Out[17]: ['#',
          '%',
          '(',
          ')',
          '*',
          '+',
          '-',
          '0',
          '1',
          '2',
          '3',
          '4',
          '5',
          '6',
          '7',
          '8',
          '9',
          '=',
          'B',
          'C',
          'F',
          'G',
          'H',
          'I',
          'K',
          'L',
          'N',
          'O',
          'P',
          'S',
          'T',
          'Z',
          '[',
          ']',
          'a',
          'b',
          'c',
          'd',
          'e',
          'i',
          'l',
          'n',
          'o',
          'r',
          's']
```

```
In  [18]: token2idx = dict((token, i) for i, token in enumerate(tokens))
```

In [19]:
```
token2idx
```

Out[19]:
```
{'#': 0,
 '%': 1,
 '(': 2,
 ')': 3,
 '*': 4,
 '+': 5,
 '-': 6,
 '0': 7,
 '1': 8,
 '2': 9,
 '3': 10,
 '4': 11,
 '5': 12,
 '6': 13,
 '7': 14,
 '8': 15,
 '9': 16,
 '=': 17,
 'B': 18,
 'C': 19,
 'F': 20,
 'G': 21,
 'H': 22,
 'I': 23,
 'K': 24,
 'L': 25,
 'N': 26,
 'O': 27,
 'P': 28,
 'S': 29,
 'T': 30,
 'Z': 31,
 '[': 32,
 ']': 33,
 'a': 34,
 'b': 35,
 'c': 36,
 'd': 37,
 'e': 38,
 'i': 39,
 'l': 40,
 'n': 41,
 'o': 42,
 'r': 43,
 's': 44}
```

In [20]:
```
len(token2idx)
```

Out[20]:
```
45
```

In [21]:

```
def TokenizeSMILESasInt(smiles, token2idx):

    char_list = list(smiles)
    Nchars = len(char_list)
    Ntokens = len(token2idx)
    tokenized_arr = np.zeros((Nchars))

    for i in range(Nchars):
        try:
            tokenized_arr[i] = token2idx[char_list[i]]
        except:
            print(char_list[i])

    return tokenized_arr
```

In [22]:

```
def TokenPolymerInt(SMI_list, token2idx, maxLen):
    Nsamples = len(SMI_list)
    Tokened_polymer = np.zeros((Nsamples, maxLen))

    for i in range(Nsamples):
        ismi = SMI_list[i]
        iarr = TokenizeSMILESasInt(ismi, token2idx)

        if iarr.shape[0] < maxLen:
            Tokened_polymer[i, 0:iarr.shape[0]] = iarr

        else:
            Tokened_polymer[i, :] = iarr[:maxLen]

    return Tokened_polymer
```

In [23]:

```
%%time
Tokened_polymer=TokenPolymerInt(polymer_df.SMILES,token2idx,120)
Tokened_polymer.shape
```

```
CPU times: user 234 ms, sys: 31.2 ms, total: 266 ms
Wall time: 254 ms
```

Out[23]:

```
(7232, 120)
```

In [ ]: