In [1]:

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

In [68]:

```
directory = "CASE-IN/"
telemetry = pd.read_csv(directory+"PdM_telemetry.csv")
errors = pd.read_csv(directory+"PdM_errors.csv")
maint = pd.read_csv(directory+"PdM_maint.csv")
machines = pd.read_csv(directory+"PdM_machines.csv")
failures = pd.read_csv(directory+"PdM_failures.csv")
```

In [69]:

```
telemetry['datetime'] = pd.to_datetime(telemetry['datetime'])
```

In [70]:

```
minDatetimeByMachineId = telemetry[['machineID','datetime']].groupby('machineID').min()
minDatetimeByMachineId.rename(columns={'datetime': 'min_datetime'}, inplace=True)
minDatetimeByMachineId = minDatetimeByMachineId.reset_index()
```

In [71]:

```
telemetry = pd.merge(telemetry,minDatetimeByMachineId,on='machineID')
telemetry['time'] = telemetry['datetime'] - telemetry['min_datetime']
```

In [72]:

```
def timedelta_to_hours(x):
    return x.days*24+x.seconds/3600
telemetry['hours'] = telemetry['time'].apply(timedelta_to_hours)
```

In [73]:

```
errors['datetime'] = pd.to_datetime(errors['datetime'])
```

In [74]:

```
merge = pd.merge(telemetry, errors,  how='left', left_on=['machineID','datetime'], right
_on =['machineID','datetime'])
merge = merge.fillna('no error')
merge = pd.merge(merge, machines, on='machineID')
```

In [75]:

```
failures['datetime'] = pd.to_datetime(failures['datetime'])
merge = pd.merge(merge, failures,  how='left', left_on=['machineID','datetime'], right_o
n =['machineID','datetime'])
```

In [76]:

```
merge = merge.fillna('no failure')
```

In [77]:

```
merge = merge.drop(axis=1,labels=['min_datetime','time'])
```

In [78]:

```
data = merge
```

In [79]:

```
maint['datetime'] = pd.to_datetime(maint['datetime'])
```

In [104]:

```
data.head()
```

Out[104]:

	datetime	machineID	volt	rotate	pressure	vibration	hours	errorID	model	age	failure
0	2015-01-01 06:00:00	1	176.217853	418.504078	113.077935	45.087686	0.0	no error	model3	18	no failure
1	2015-01-01 07:00:00	1	162.879223	402.747490	95.460525	43.413973	1.0	no error	model3	18	no failure
2	2015-01-01 08:00:00	1	170.989902	527.349825	75.237905	34.178847	2.0	no error	model3	18	no failure
3	2015-01-01 09:00:00	1	162.462833	346.149335	109.248561	41.122144	3.0	no error	model3	18	no failure
4	2015-01-01 10:00:00	1	157.610021	435.376873	111.886648	25.990511	4.0	no error	model3	18	no failure

In [ ]:

```

```

In [81]:

```
maint_df = pd.DataFrame()
maint_df['datetime'] = maint['datetime']
maint_df['machineID'] = maint['machineID']
unique_comps = maint['comp'].unique()
for comp in unique_comps:
    maint_df[comp] = 0
    maint_df[comp+"_prev"] = 0
for index, row in maint.iterrows():
    maint_df.loc[index,row['comp']] = 1
```

In [82]:

```
min_data_df = data.groupby('machineID')['datetime'].min().reset_index()
maint_df = pd.merge(maint_df,min_data_df,on=['machineID','datetime'],how='outer',sort=Tr
ue)
maint_df = maint_df.fillna(0)
```

In [83]:

```
maint_df['diff_datetime'] = maint_df.groupby('machineID')['datetime'].diff()
maint_df = maint_df.fillna(pd.Timedelta(0))
```

In [84]:

```
min_data_df.rename(columns={'datetime': 'min_datetime'}, inplace=True)
maint_df = pd.merge(maint_df,min_data_df,on=['machineID'])

maint_df['diff_min_datetime'] = maint_df['datetime'] - maint_df['min_datetime']
maint_df['diff_date_hours'] = maint_df['diff_min_datetime'].apply(timedelta_to_hours)
maint_df['diff_datetime_hours'] = maint_df['diff_datetime'].apply(timedelta_to_hours)
```

In [85]:

```
maint_df = maint_df.drop(axis=1,labels=['min_datetime','diff_datetime','diff_min_datetime
'])
```

In [86]:

```
drop_indices = []
prevID = 0
prevTime = 0
for index, row in maint_df.iterrows():
    machineID = row['machineID']
```

```
    time = row['datetime']
    time_to_observ = row['diff_date_hours']
    if index != 0:
        if prevID == machineID:
            if time == prevTime or time_to_observ <= 0.0:
                for comp in unique_comps:
                    maint_df.loc[index,comp] = maint_df.loc[index-1,comp] + maint_df.loc
[index,comp]
                    if row[comp] != 1:
                        maint_df.loc[index,comp+"_prev"]=maint_df.loc[index-1,comp+"_pre
v"] + maint_df.loc[index,'diff_datetime_hours']
                    else:
                        maint_df.loc[index,comp+"_prev"] = 0
                if time_to_observ < 0.0:
                    drop_indices.append(index)
                else:
                    maint_df.loc[index,'diff_datetime_hours'] = maint_df.loc[index -1,'d
iff_datetime_hours']
                    drop_indices.append(index-1)
        else:
            if time_to_observ < 0.0:
                 drop_indices.append(index)
    else:
        if time_to_observ < 0.0:
                 drop_indices.append(index)
    prevID = machineID
    prevTime = time
```

In [87]:

```
maint_df = maint_df.drop(index=drop_indices)
```

In [88]:

```
maint_df = maint_df.drop(axis=1,labels=['diff_date_hours','diff_datetime_hours'])
```

In [89]:

```
maint_df.head()
```

Out[89]:

	datetime	machineID	comp2	comp2_prev	comp4	comp4_prev	comp3	comp3_prev	comp1	comp1_prev
4	**2015-01-01 06:00:00**	**1**	**1.0**	**5136.0**	**1.0**	**4056.0**	**1.0**	**3696.0**	**1.0**	**456.0**
6	**2015-01-05 06:00:00**	**1**	**0.0**	**0.0**	**1.0**	**0.0**	**0.0**	**0.0**	**1.0**	**0.0**
8	**2015-01-20 06:00:00**	**1**	**0.0**	**0.0**	**0.0**	**0.0**	**1.0**	**0.0**	**1.0**	**0.0**
10	**2015-02-04 06:00:00**	**1**	**0.0**	**0.0**	**1.0**	**0.0**	**1.0**	**0.0**	**0.0**	**0.0**
11	**2015-02-19 06:00:00**	**1**	**0.0**	**0.0**	**0.0**	**0.0**	**1.0**	**0.0**	**0.0**	**0.0**

In [95]:

```
data2 = pd.merge(data, maint_df, how='left', on=['machineID','datetime'])
```

In [96]:

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 876445 entries, 0 to 876444
Data columns (total 19 columns):
datetime       876445 non-null datetime64[ns]
machineID      876445 non-null int64
volt           876445 non-null float64
rotate         876445 non-null float64
pressure       876445 non-null float64
vibration      876445 non-null float64
hours          876445 non-null float64
errorID        876445 non-null object
```

```
model          876445 non-null object
age            876445 non-null int64
failure        876445 non-null object
comp2          2297 non-null float64
comp2_prev     2297 non-null float64
comp4          2297 non-null float64
comp4_prev     2297 non-null float64
comp3          2297 non-null float64
comp3_prev     2297 non-null float64
comp1          2297 non-null float64
comp1_prev     2297 non-null float64
dtypes: datetime64[ns](1), float64(13), int64(2), object(3)
memory usage: 133.7+ MB
```

In [97]:

```
data2 = data2.fillna(0.0)
```

In [98]:

```
prevID = 0
prevTime = 0
for index, row in data2.iterrows():
    machineID = row['machineID']
    time = row['datetime']
    if index != 0:
        if prevID == machineID:
            for comp in unique_comps:
                data2.loc[index,comp] = data2.loc[index-1,comp] + data2.loc[index,comp]
                if row[comp] != 1:
                    data2.loc[index,comp+"_prev"]=data2.loc[index-1,comp+"_prev"] + 1
                else:
                    data2.loc[index,comp+"_prev"] = 0
    prevID = machineID
    prevTime = time
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-98-f1d43578e9a4> in <module>
      9                 data2.loc[index,comp] = data2.loc[index-1,comp] + data2.loc[inde
x,comp]
     10                 if row[comp] != 1:
---> 11                     data2.loc[index,comp+"_prev"]=data2.loc[index-1,comp+"_prev"
] + 1
     12                 else:
     13                     data2.loc[index,comp+"_prev"] = 0

~\Anaconda\lib\site-packages\pandas\core\indexing.py in __setitem__(self, key, value)
    188             key = com.apply_if_callable(key, self.obj)
    189         indexer = self._get_setitem_indexer(key)
--> 190         self._setitem_with_indexer(indexer, value)
    191
    192     def _validate_key(self, key, axis):

~\Anaconda\lib\site-packages\pandas\core\indexing.py in _setitem_with_indexer(self, index
er, value)
    618                 # scalar
    619                 for item in labels:
--> 620                     setter(item, value)
    621
    622         else:

~\Anaconda\lib\site-packages\pandas\core\indexing.py in setter(item, v)
    536                     # set the item, possibly having a dtype change
    537                     s._consolidate_inplace()
--> 538                     s = s.copy()
    539                     s._data = s._data.setitem(indexer=pi, value=v)
    540                     s._maybe_update_cacher(clear=True)

~\Anaconda\lib\site-packages\pandas\core\generic.py in copy(self, deep)
   5802         dtype: object
```

```
   5803         """
-> 5804         data = self._data.copy(deep=deep)
   5805         return self._constructor(data).__finalize__(self)
   5806

~\Anaconda\lib\site-packages\pandas\core\internals\managers.py in copy(self, deep)
    732             new_axes = list(self.axes)
    733         return self.apply('copy', axes=new_axes, deep=deep,
--> 734                           do_integrity_check=False)
    735
    736     def as_array(self, transpose=False, items=None):

~\Anaconda\lib\site-packages\pandas\core\internals\managers.py in apply(self, f, axes, fi
lter, do_integrity_check, consolidate, **kwargs)
    393                                                 copy=align_copy)
    394
--> 395             applied = getattr(b, f)(**kwargs)
    396             result_blocks = _extend_blocks(applied, result_blocks)
    397

~\Anaconda\lib\site-packages\pandas\core\internals\blocks.py in copy(self, deep)
    751         values = self.values
    752         if deep:
--> 753             values = values.copy()
    754         return self.make_block_same_class(values)
    755

KeyboardInterrupt:
```

Data analysis

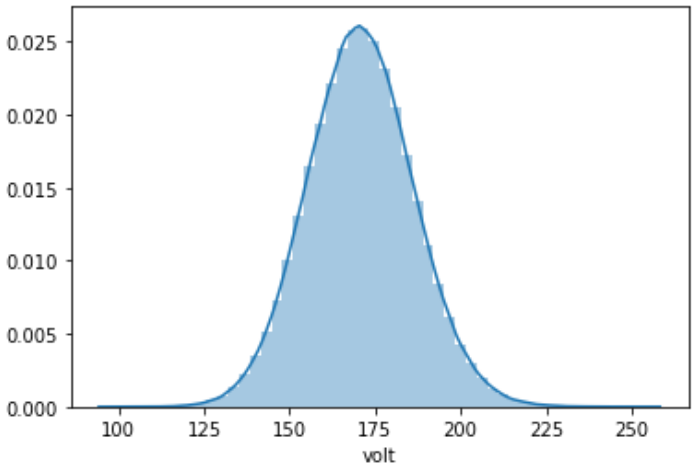
Распределения телеметрии

In [28]:

```
sns.distplot(telemetry['volt'])
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf02a13cf8>
```

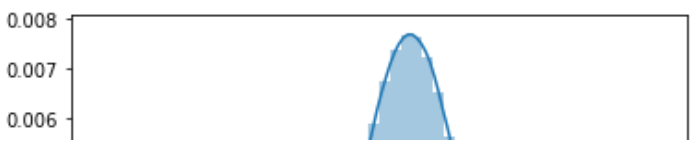


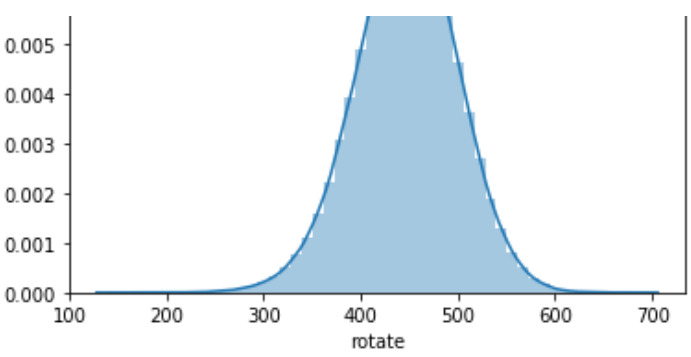
In [29]:

```
sns.distplot(telemetry['rotate'])
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf03276f28>
```



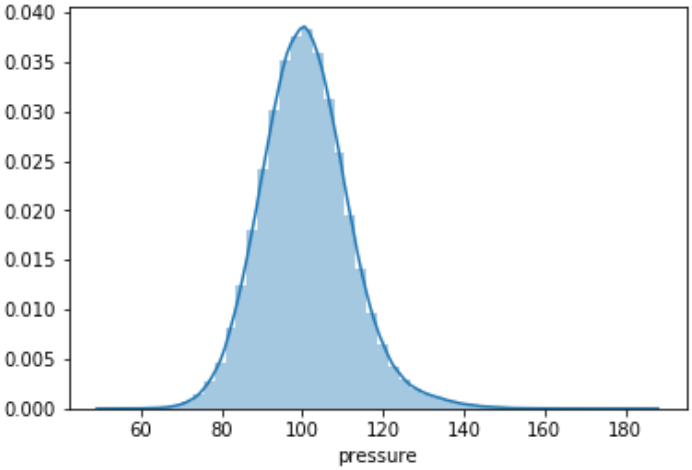


In [30]:

```
sns.distplot(telemetry['pressure'])
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf02dbf940>
```

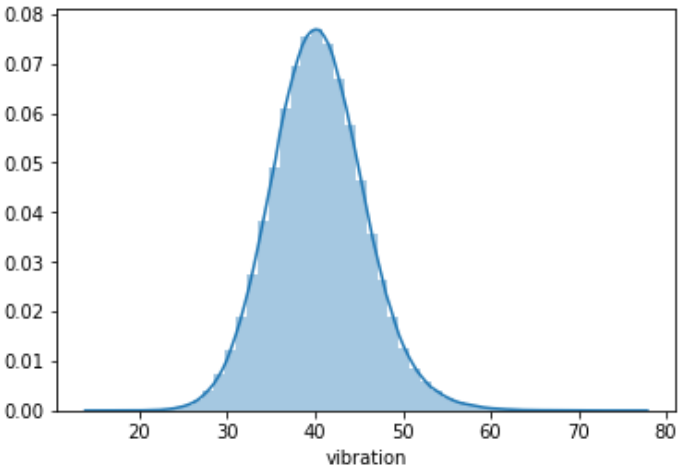


In [31]:

```
sns.distplot(telemetry['vibration'])
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf02ea9f28>
```



Корреляция телеметрии

In [123]:

```
data[['volt','rotate','pressure','vibration']].corr()
```

Out[123]:

	volt	rotate	pressure	vibration

	volt	rotate	pressure	vibration
volt	1.000000	-0.001541	0.001671	0.002425
rotate	-0.001541	1.000000	-0.000694	-0.003124
pressure	0.001671	-0.000694	1.000000	0.001471
vibration	0.002425	-0.003124	0.001471	1.000000

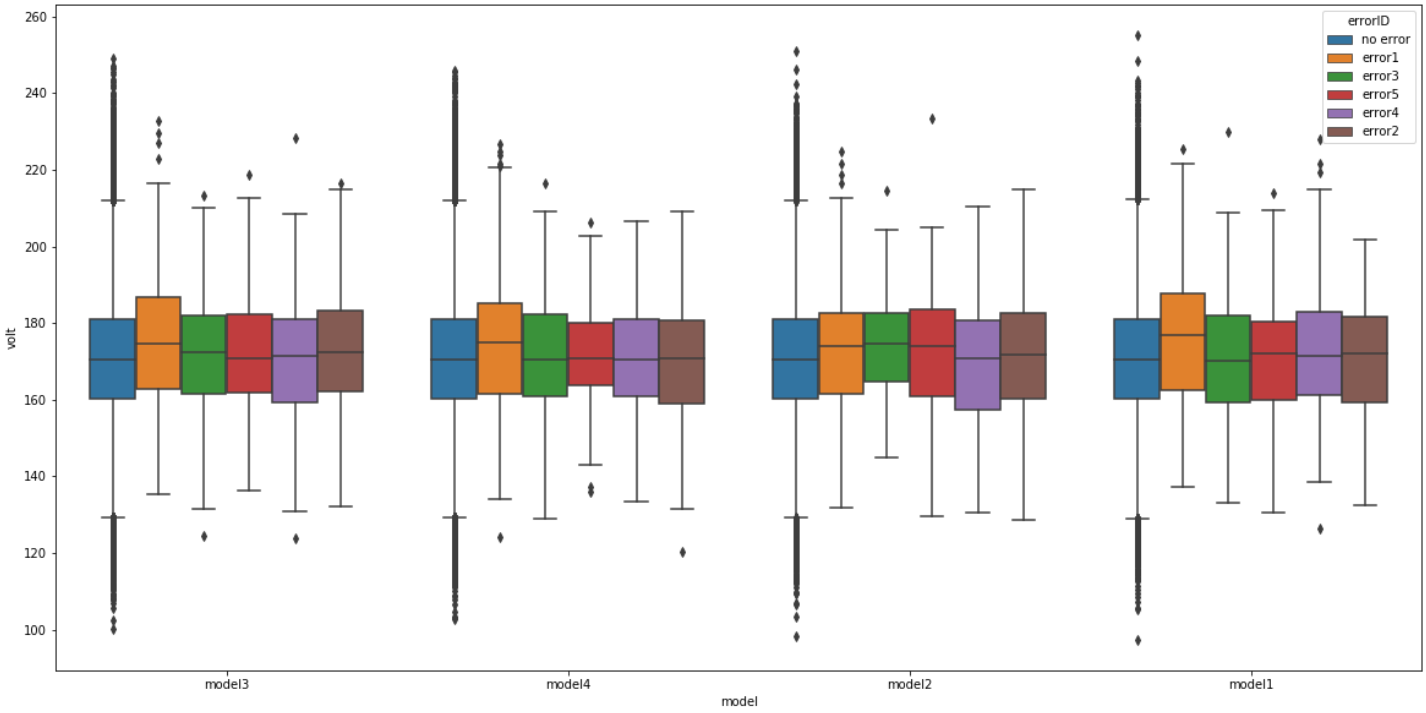
Распределение телеметрии по ошибкам у различных моделей станков

In [109]:

```
plt.figure(figsize=(20,10))
sns.boxplot(x="model", y="volt", hue='errorID',data=data)
```

Out[109]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf1e59a518>
```

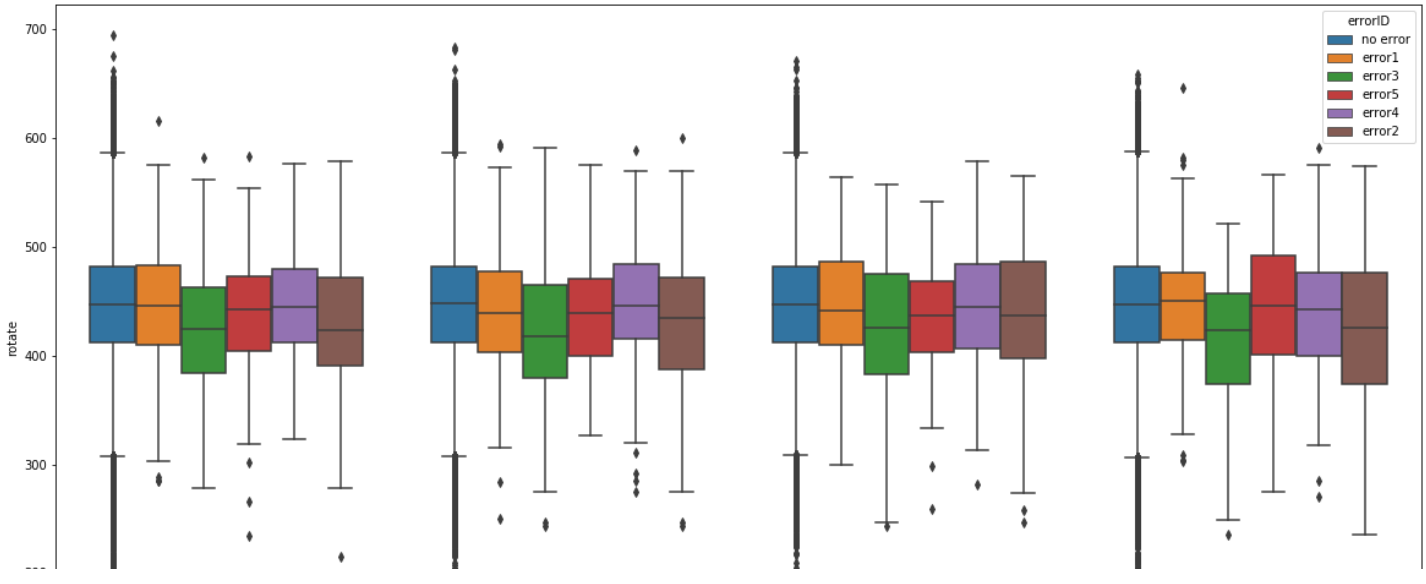


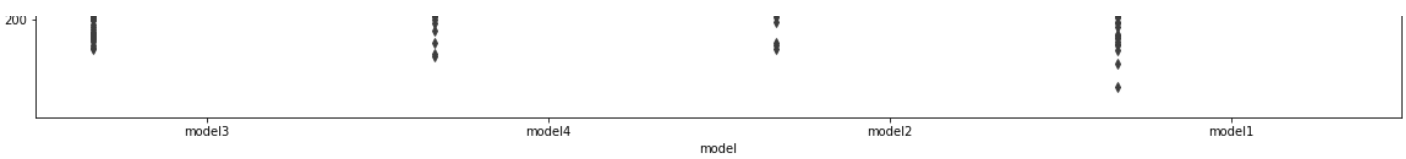
In [110]:

```
plt.figure(figsize=(20,10))
sns.boxplot(x="model", y="rotate", hue='errorID',data=data)
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf1e80e908>
```



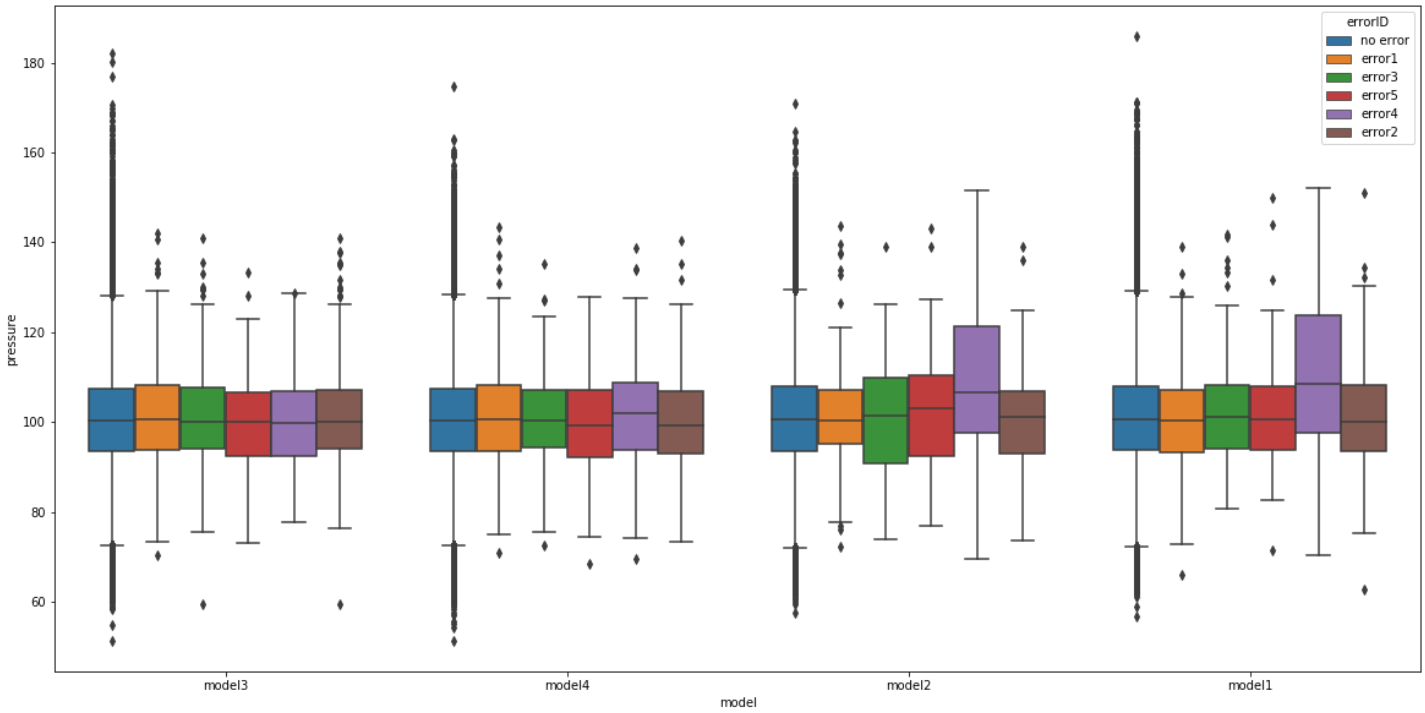


In [112]:

```
plt.figure(figsize=(20,10))
sns.boxplot(x="model", y="pressure", hue='errorID',data=data)
```

Out[112]:

<matplotlib.axes._subplots.AxesSubplot at 0x1bf20cbc3c8>

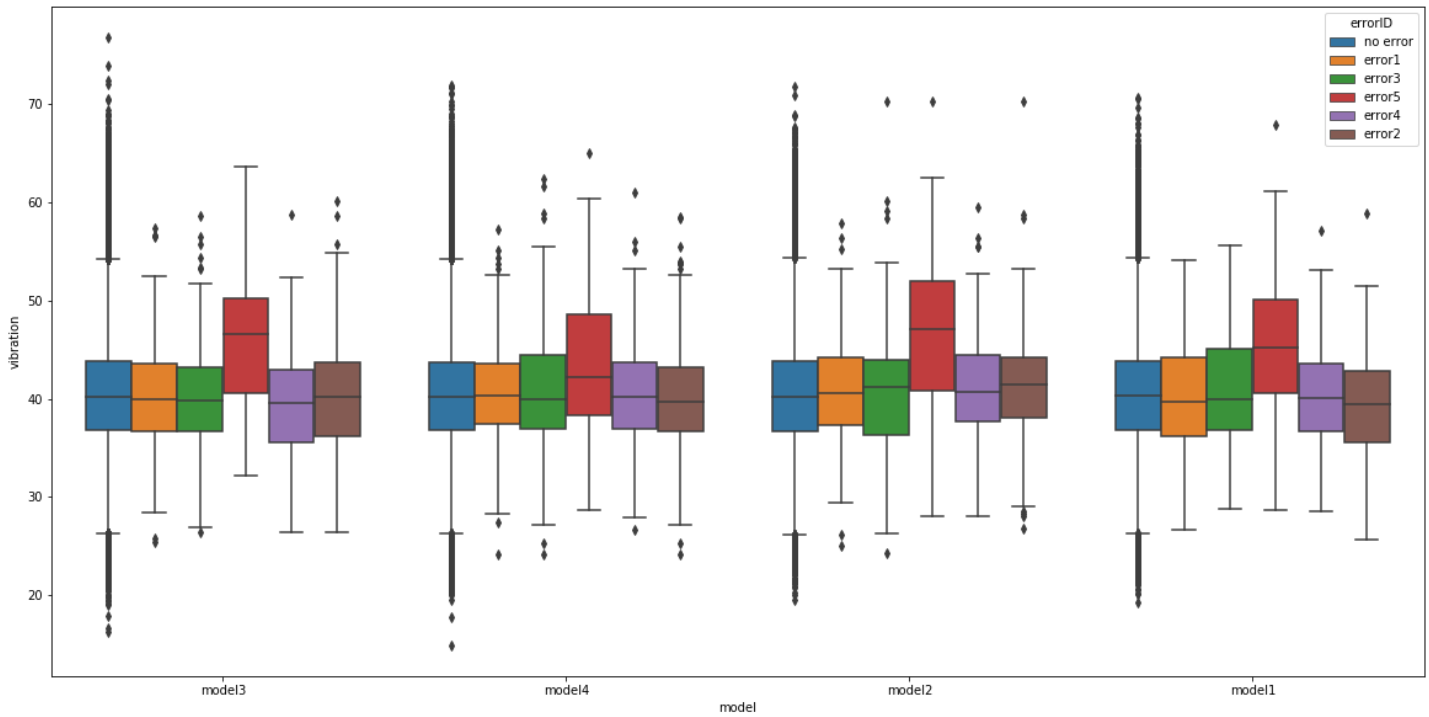


In [113]:

```
plt.figure(figsize=(20,10))
sns.boxplot(x="model", y="vibration", hue='errorID',data=data)
```

Out[113]:

<matplotlib.axes._subplots.AxesSubplot at 0x1bf20aa1048>



In [117]:

```
data.head()
```

Out[117]:

	datetime	machineID	volt	rotate	pressure	vibration	hours	errorID	model	age	failure
0	2015-01-01 06:00:00	1	176.217853	418.504078	113.077935	45.087686	0.0	no error	model3	18	no failure
1	2015-01-01 07:00:00	1	162.879223	402.747490	95.460525	43.413973	1.0	no error	model3	18	no failure
2	2015-01-01 08:00:00	1	170.989902	527.349825	75.237905	34.178847	2.0	no error	model3	18	no failure
3	2015-01-01 09:00:00	1	162.462833	346.149335	109.248561	41.122144	3.0	no error	model3	18	no failure
4	2015-01-01 10:00:00	1	157.610021	435.376873	111.886648	25.990511	4.0	no error	model3	18	no failure

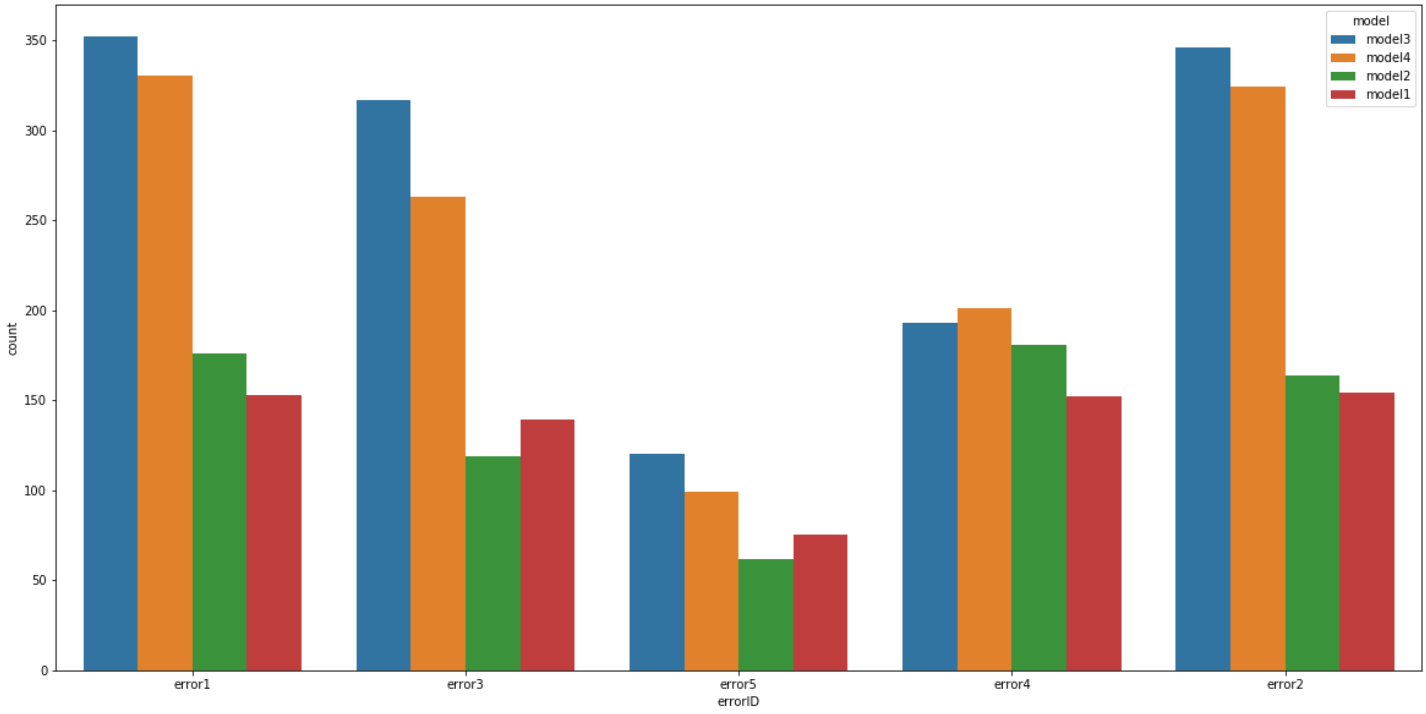
Распределение ошибок по моделям

In [119]:

```
plt.figure(figsize=(20,10))
sns.countplot(x="errorID",hue='model', data=data.loc[data['errorID'] != 'no error'])
```

Out[119]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf26305828>
```



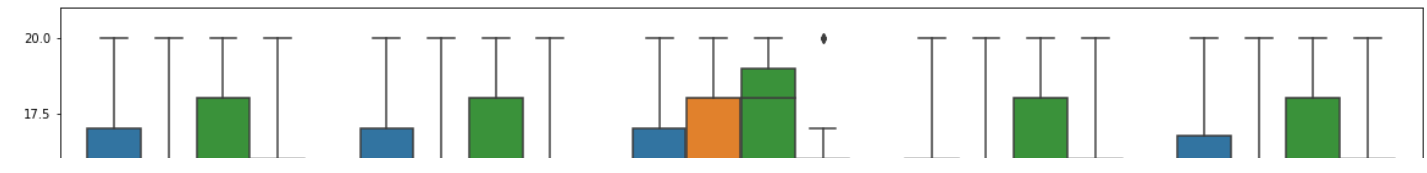
Распределение зависимости ошибки от возраста у моделей станков

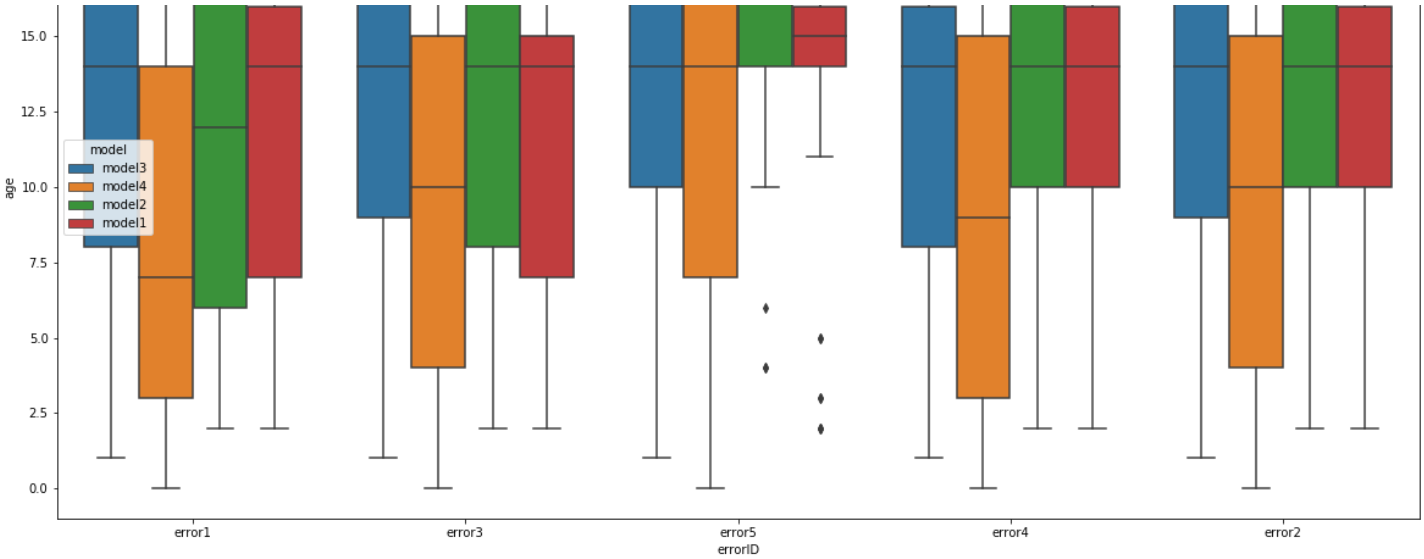
In [120]:

```
plt.figure(figsize=(20,10))
sns.boxplot(x="errorID",y='age',hue='model', data=data.loc[data['errorID'] != 'no error'
])
```

Out[120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf26351be0>
```





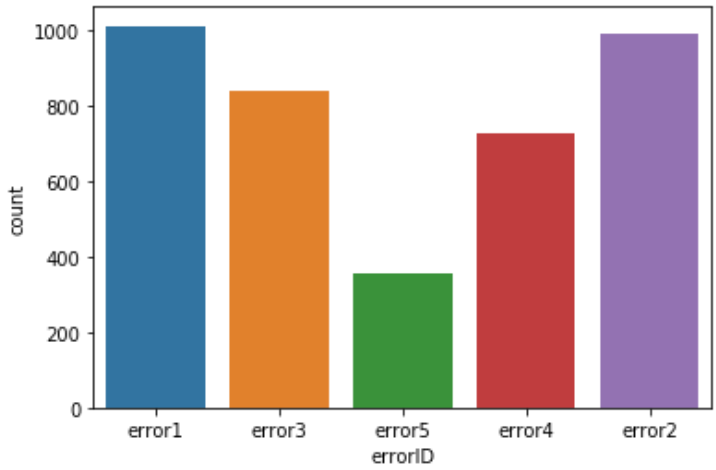
Распределение количества ошибок

In [21]:

```
sns.countplot(x="errorID", data=errors)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf02971ef0>
```



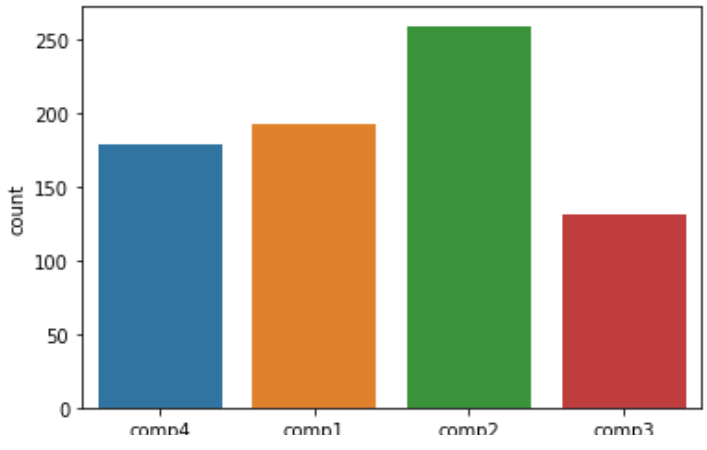
Распределение замен узлов из-за отказов

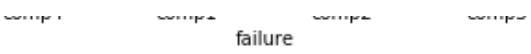
In [19]:

```
sns.countplot(x="failure", data=failures)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf029117f0>
```





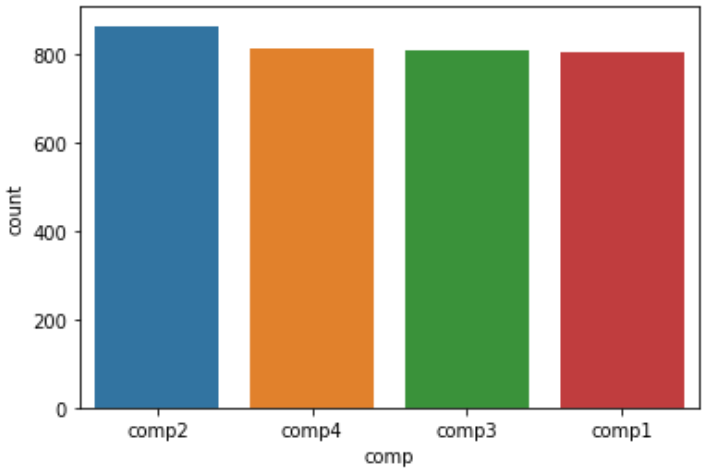
Распределение ремонта компонентов

In [16]:

```
sns.countplot(x="comp", data=maint)
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf028c12e8>
```



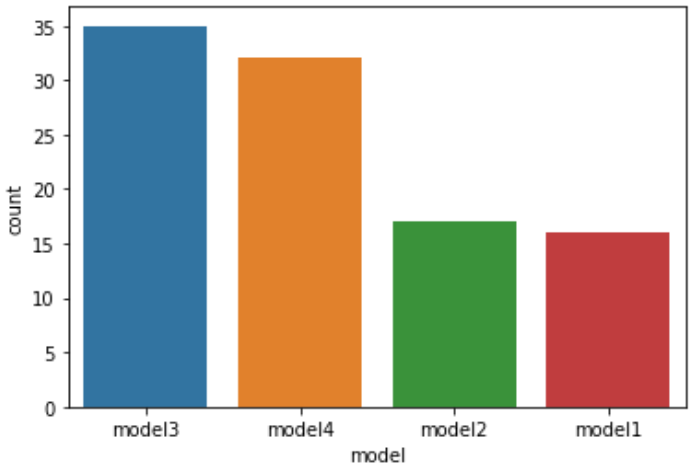
Распределение моделей станков

In [13]:

```
sns.countplot(x="model", data=machines)
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf025e9b00>
```



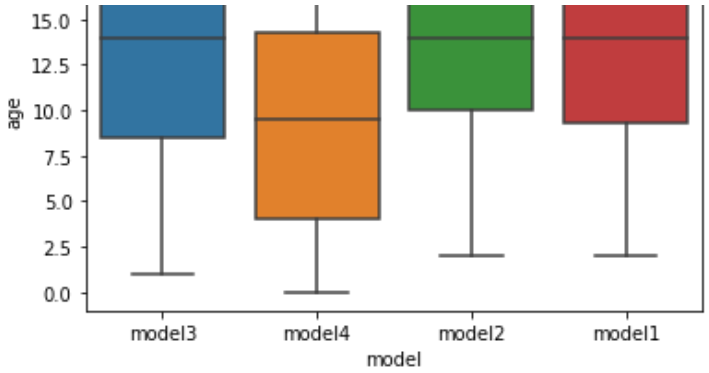
Распределение возраста моделей станков

In [14]:

```
sns.boxplot(x="model", y="age", data=machines)
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf0271ddd8>
```



Обучение модели

In [106]:

```
from sklearn import preprocessing
errorEncoder = preprocessing.LabelEncoder()
errorEncoder.fit(data['errorID'])
data['errorID'] = errorEncoder.transform(data['errorID'])

modelEncoder = preprocessing.LabelEncoder()
modelEncoder.fit(data['model'])
data['model'] = modelEncoder.transform(data['model'])

failureEncoder = preprocessing.LabelEncoder()
failureEncoder.fit(data['failure'])
data['failure'] = failureEncoder.transform(data['failure'])
```

In [116]:

```
X = data.drop(axis=1,labels=['datetime','failure'])
y = data['failure']
```

In [148]:

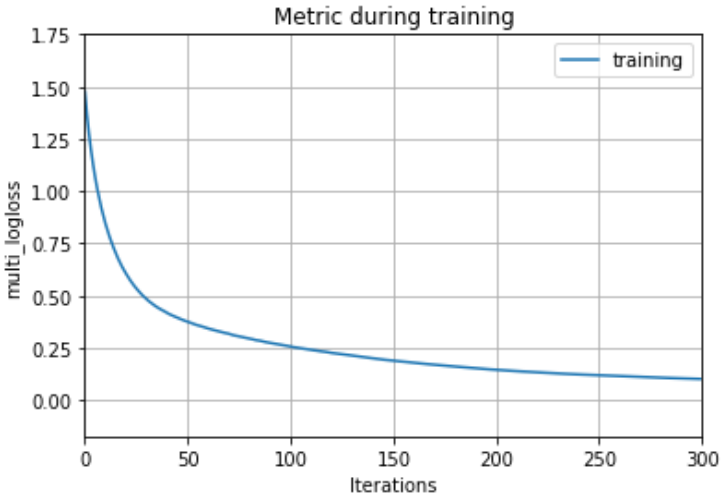
```
from lightgbm import LGBMClassifier
import lightgbm as lgb
lgb_params = {
        'boosting_type': 'gbdt',
        'objective': 'multiclass',
        "num_class" : 5,
        'metric': 'multi_logloss',
        'n_estimators': 300,
        'max_depth': 7,
        'verbose':-1,
        'class_weight':'balanced'
    }

trn_x = X
trn_y = y
val_x = X
val_y = y
clf = LGBMClassifier(**lgb_params)
clf.fit(
    trn_x, trn_y,
    eval_set = [(val_x,val_y)],
    eval_metric='multi_logloss',
    verbose=10,
    early_stopping_rounds = 50
)
lgb.plot_metric(clf.evals_result_)
```

```
Training until validation scores don't improve for 50 rounds
[10] training's multi_logloss: 0.880414
[20] training's multi_logloss: 0.621886
[30] training's multi_logloss: 0.492427
[40] training's multi_logloss: 0.42325
[50] training's multi_logloss: 0.378645
```

```
[60] training's multi_logloss: 0.346119
[70] training's multi_logloss: 0.320441
[80] training's multi_logloss: 0.296368
[90] training's multi_logloss: 0.276179
[100] training's multi_logloss: 0.258731
[110] training's multi_logloss: 0.242114
[120] training's multi_logloss: 0.227525
[130] training's multi_logloss: 0.214872
[140] training's multi_logloss: 0.201749
[150] training's multi_logloss: 0.190353
[160] training's multi_logloss: 0.179948
[170] training's multi_logloss: 0.170458
[180] training's multi_logloss: 0.161853
[190] training's multi_logloss: 0.15359
[200] training's multi_logloss: 0.146044
[210] training's multi_logloss: 0.139567
[220] training's multi_logloss: 0.133718
[230] training's multi_logloss: 0.128227
[240] training's multi_logloss: 0.123717
[250] training's multi_logloss: 0.119318
[260] training's multi_logloss: 0.115132
[270] training's multi_logloss: 0.111619
[280] training's multi_logloss: 0.107967
[290] training's multi_logloss: 0.104578
[300] training's multi_logloss: 0.101692
Did not meet early stopping. Best iteration is:
[300] training's multi_logloss: 0.101692
```

Out[148]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ecd1044e48>
```



In [149]:

```
cnt = 0
for y_pred, y_true in zip(clf.predict(X),y):
    if y_pred != y_true:
        cnt= cnt+1
print(cnt)
```

```
29514
```

In [147]:

```
len(y)
```

Out[147]:

```
876445
```

In [144]:

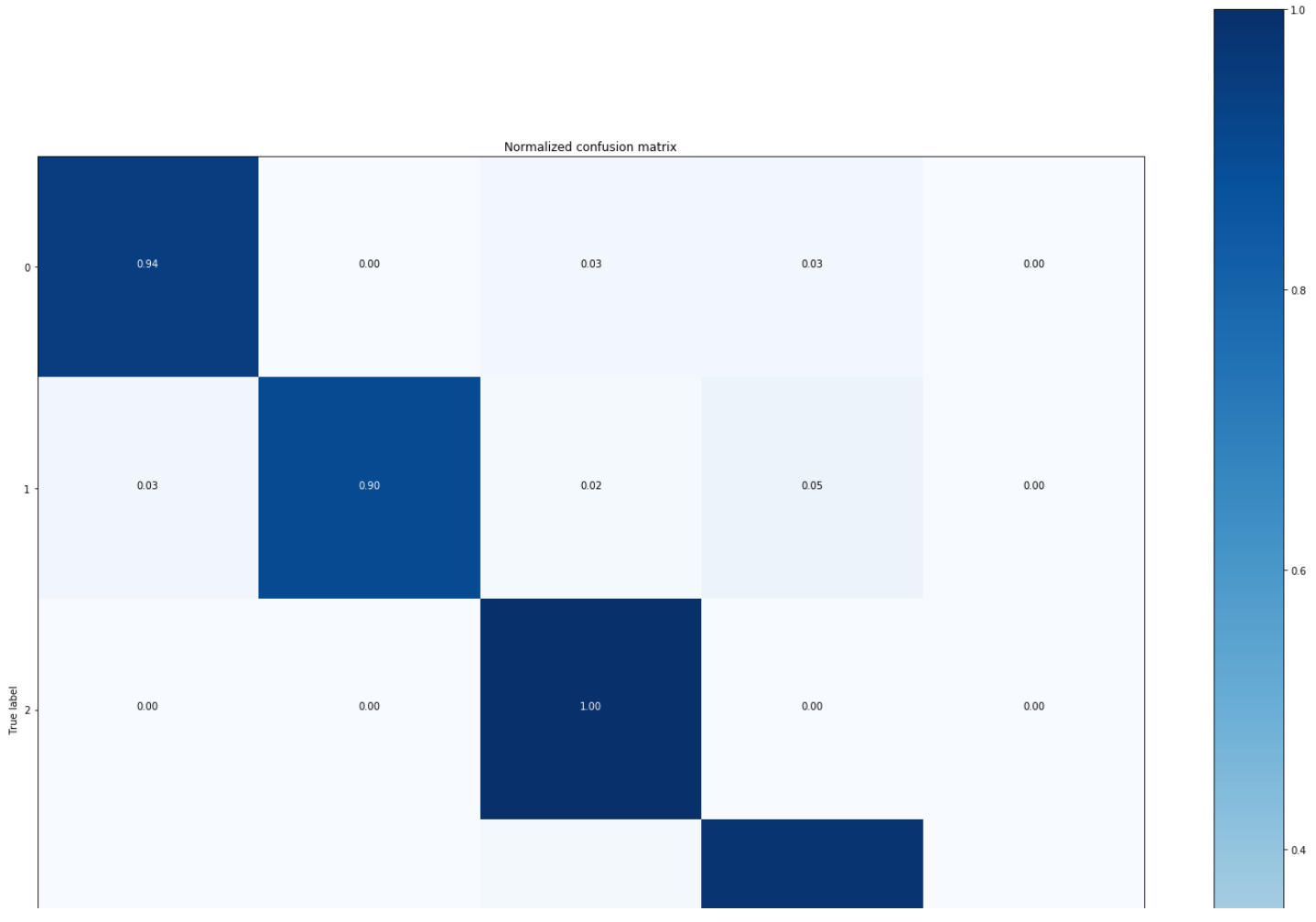
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
```

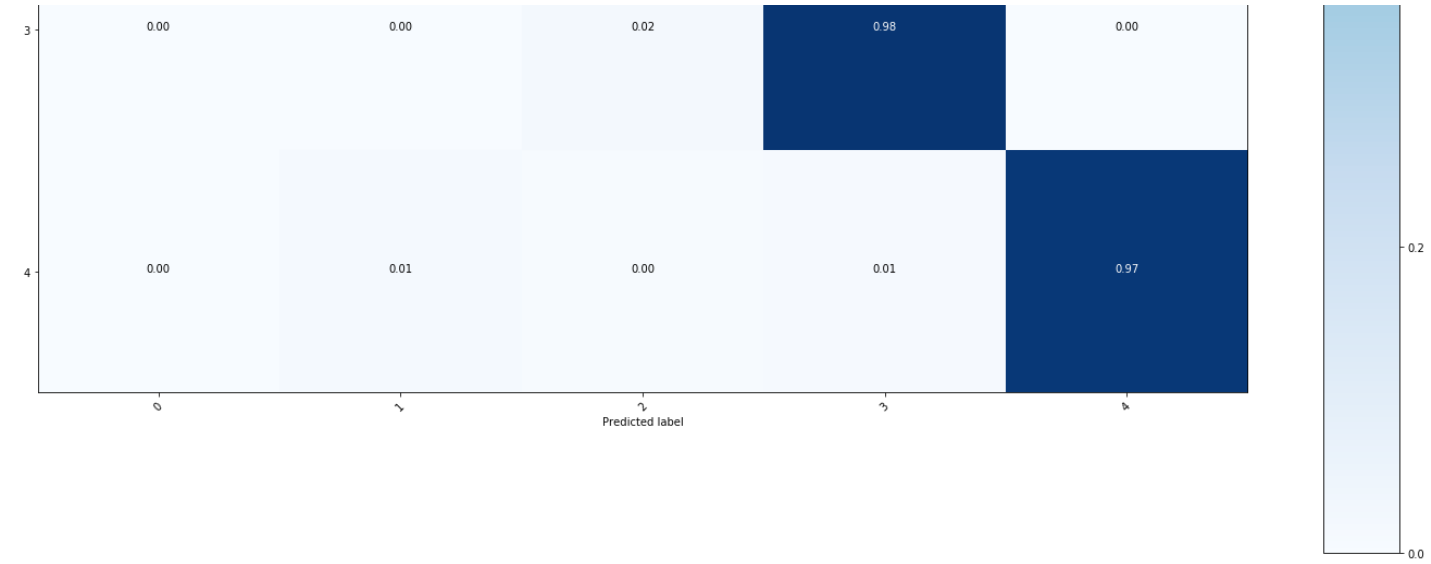
```
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [150]:

```
cnf_matrix = confusion_matrix(y, clf.predict(X))
plt.figure(figsize = (20,20))
plot_confusion_matrix(cnf_matrix, classes=np.unique(y), normalize=True,
                      title='Normalized confusion matrix')
```

```
Normalized confusion matrix
```

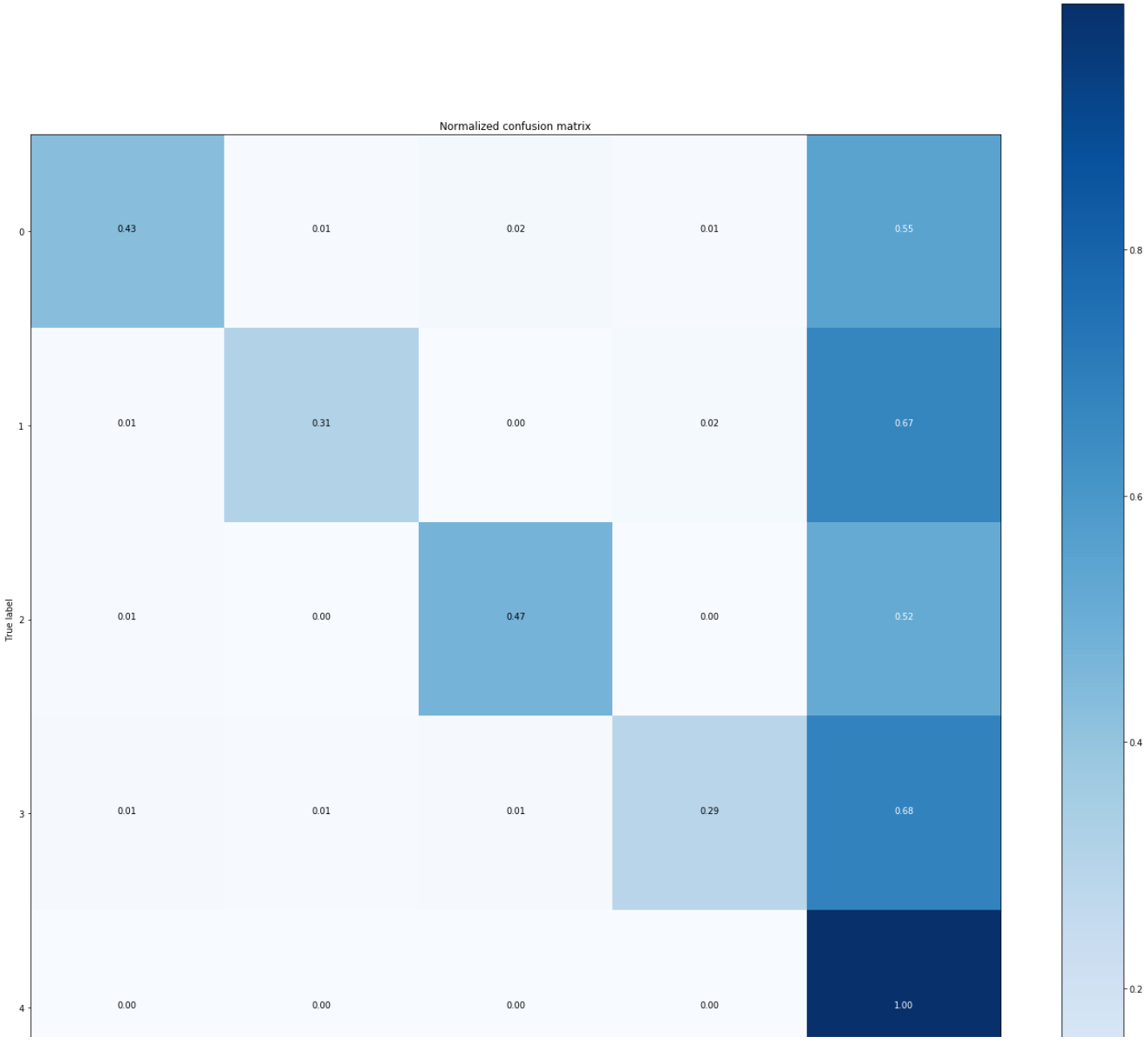


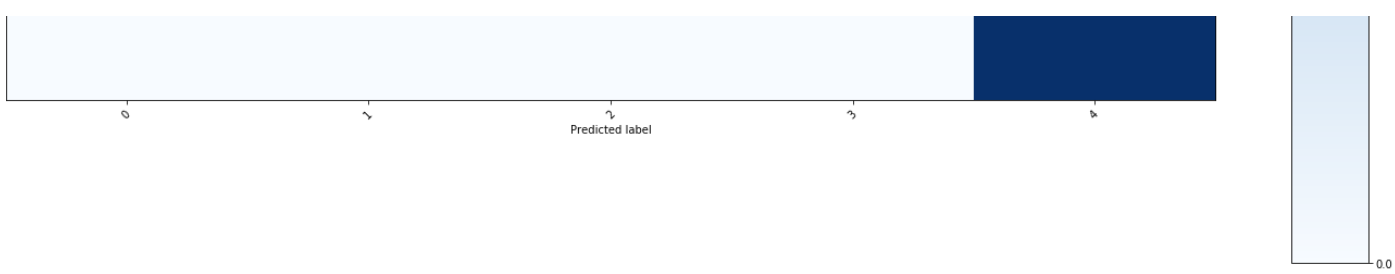


In [140]:

```
import itertools
from sklearn.metrics import confusion_matrix
cnf_matrix = confusion_matrix(y, clf.predict(X))
plt.figure(figsize = (20,20))
plot_confusion_matrix(cnf_matrix, classes=np.unique(y), normalize=True,
                      title='Normalized confusion matrix')
```

Normalized confusion matrix





In [ ]: