In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set_theme(style="darkgrid")
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GroupKFold
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
import lightgbm as lgb
Log=False
```

In [2]:
```
data=pd.read_csv('train.csv')
```

In [3]:
```
data.head()
```

Out[3]:

| | id | breath_id | R | C | time_step | u_in | u_out | pressure |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 20 | 50 | 0.000000 | 0.083334 | 0 | 5.837492 |
| **1** | 2 | 1 | 20 | 50 | 0.033652 | 18.383041 | 0 | 5.907794 |
| **2** | 3 | 1 | 20 | 50 | 0.067514 | 22.509278 | 0 | 7.876254 |
| **3** | 4 | 1 | 20 | 50 | 0.101542 | 22.808822 | 0 | 11.742872 |
| **4** | 5 | 1 | 20 | 50 | 0.135756 | 25.355850 | 0 | 12.234987 |

# EDA

## NaN check

In [4]:
```
for x in data.columns:
    print(data[x].isna().sum(),'NaN values for column: {0} : '.format(x))
```

```
0 NaN values for column: id :
0 NaN values for column: breath_id :
0 NaN values for column: R :
0 NaN values for column: C :
0 NaN values for column: time_step :
0 NaN values for column: u_in :
0 NaN values for column: u_out :
0 NaN values for column: pressure :
```

## Data Statistics by Column

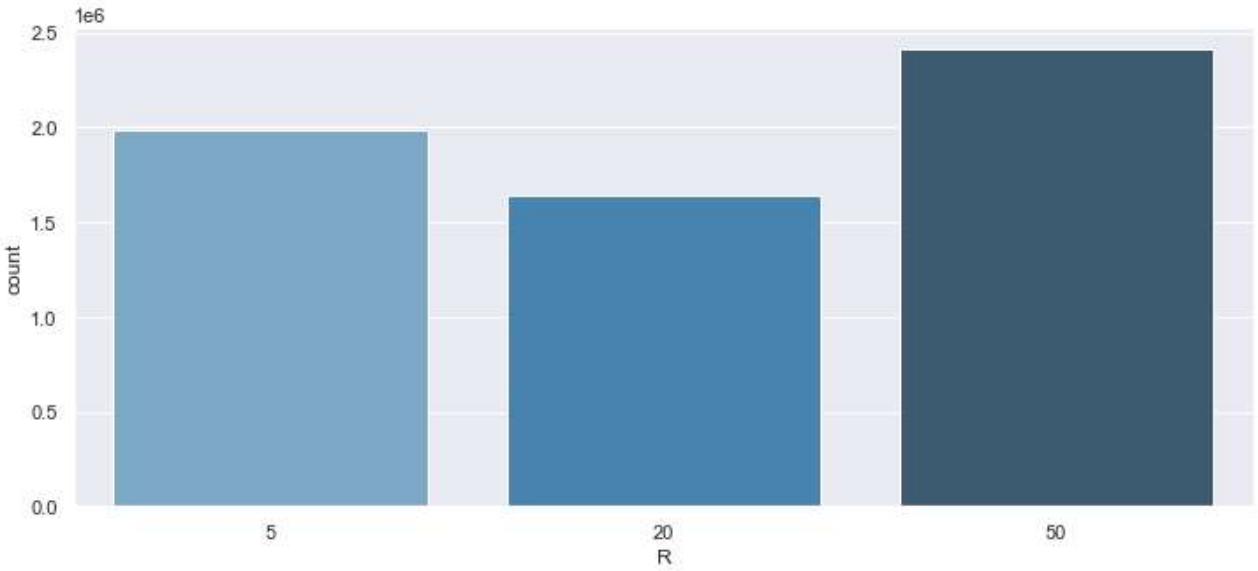
### Breath_id : Globally unique time step identifier across an entire file

In [5]:
```
data['breath_id'].unique().shape
```

Out[5]: (75450,)

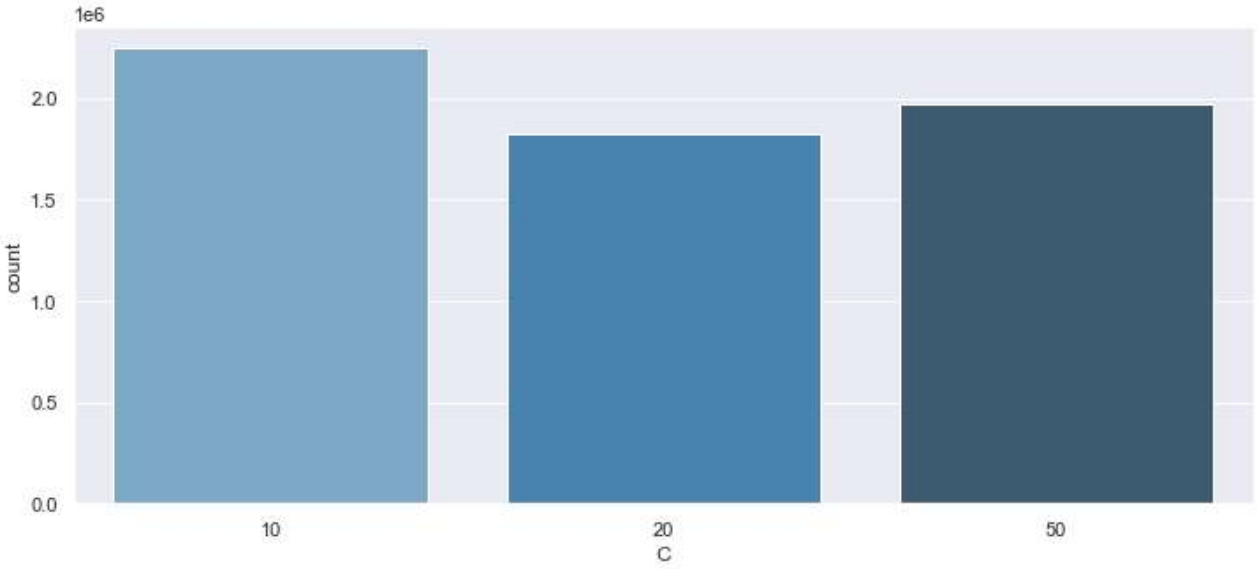
## R : lung attribute indicating how restricted the airway is (in cmH2O/L/S). Physically, this is the change in pressure per change in flow (air volume per time). Intuitively, one can imagine blowing up a balloon through a straw. We can change R by changing the diameter of the straw, with higher R being harder to blow.

In [6]:
```
plt.figure(figsize = (12,5))
ax = sns.countplot(x="R", data=data, palette="Blues_d")
```




## C : lung attribute indicating how compliant the lung is (in mL/cmH2O). Physically, this is the change in volume per change in pressure. Intuitively, one can imagine the same balloon example. We can change C by changing the thickness of the balloon's latex, with higher C having thinner latex and easier to blow.

In [7]:
```
plt.figure(figsize = (12,5))
ax = sns.countplot(x="C", data=data,palette="Blues_d")
```



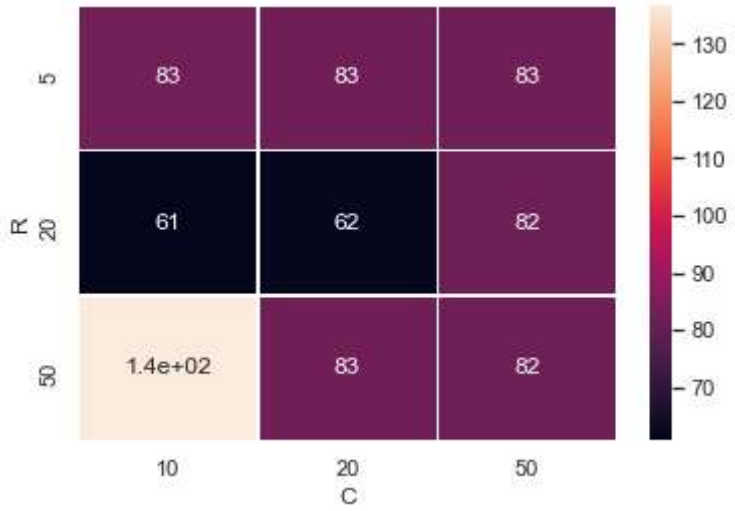
## R & C combination :

In [8]:
```
pd.crosstab(data["R"],data["C"]) /80
```

Out[8]:

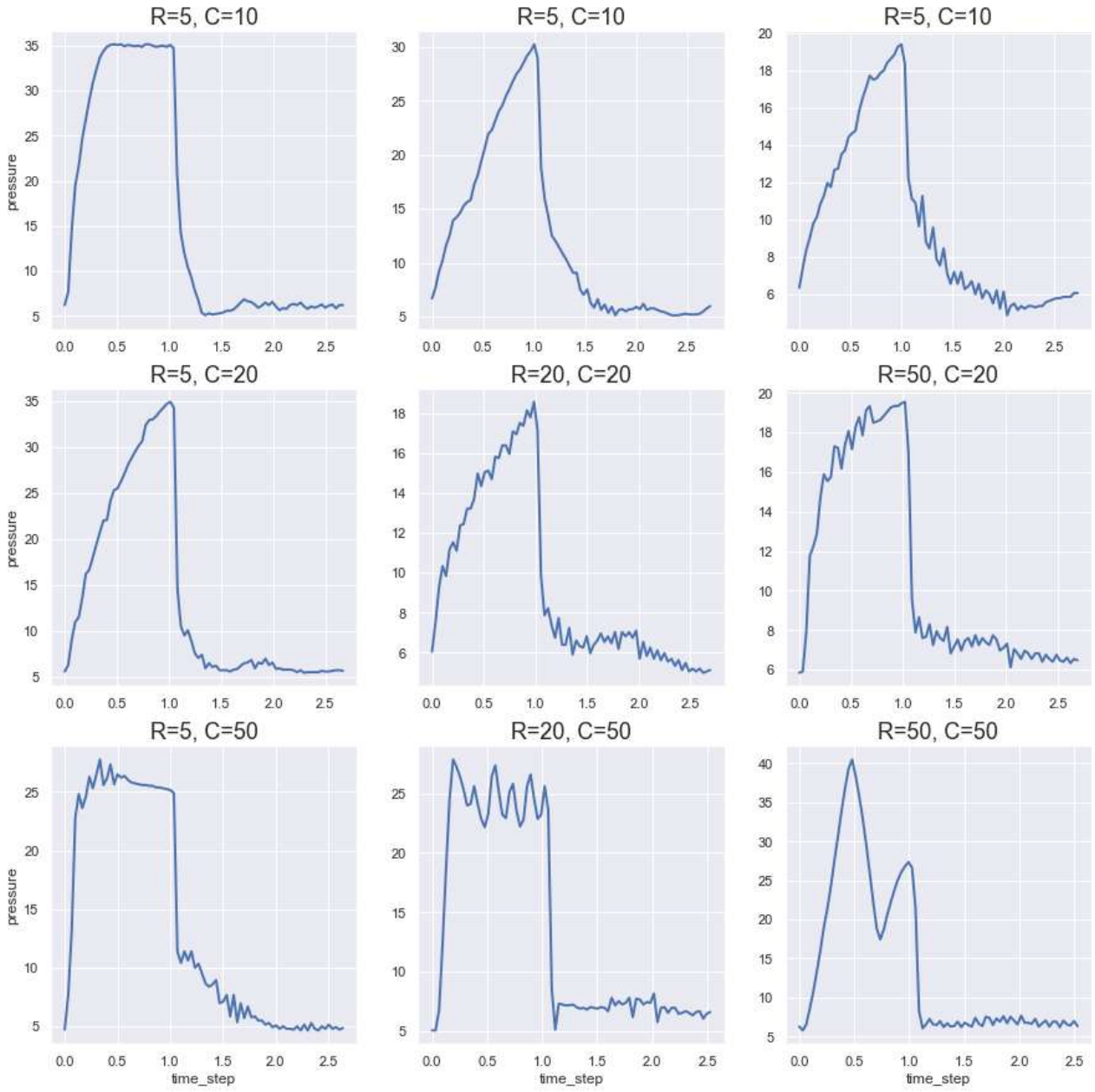
| C | 10 | 20 | 50 |
|---|---|---|---|
| R | | | |
| 5 | 8312.0 | 8277.0 | 8271.0 |
| 20 | 6070.0 | 6208.0 | 8186.0 |
| 50 | 13677.0 | 8260.0 | 8189.0 |

In [9]:
```
temp=(pd.crosstab(data["R"],data["C"]) /80)/100 # Div by 80 as per breather 80 cycles g
ax = sns.heatmap(temp,annot=True,linewidths=.5)
```




In [10]:
```
#data.query('R == 50 & C == 50').head(2) to find id of that R_C combination

P_R_C_5_10 = data.query('breath_id == 39').reset_index(drop = True)
P_R_C_5_20 = data.query('breath_id == 17').reset_index(drop = True)
P_R_C_5_50 = data.query('breath_id == 5').reset_index(drop = True)
P_R_C_20_10 = data.query('breath_id == 21').reset_index(drop = True)
P_R_C_20_20 = data.query('breath_id == 2').reset_index(drop = True)
P_R_C_20_50 = data.query('breath_id == 1').reset_index(drop = True)
P_R_C_50_10 = data.query('breath_id == 18').reset_index(drop = True)
P_R_C_50_20 = data.query('breath_id == 3').reset_index(drop = True)
P_R_C_50_50 = data.query('breath_id == 4').reset_index(drop = True)

fig, axes = plt.subplots(3,3,figsize=(15,15))
sns.lineplot(data=P_R_C_5_10, x="time_step", y="pressure", lw=2, ax=axes[0,0])
axes[0,0].set_title ("R=5, C=10", fontsize=18)
axes[0,0].set(xlabel='')

sns.lineplot(data=P_R_C_5_20, x="time_step", y="pressure",  lw=2, ax=axes[0,1])
axes[0,1].set_title ("R=5, C=10", fontsize=18)
axes[0,1].set(xlabel='')
axes[0,1].set(ylabel='')

sns.lineplot(data=P_R_C_5_50, x="time_step", y="pressure",  lw=2,ax=axes[0,2])
```

```
axes[0,2].set_title ("R=5, C=10", fontsize=18)
axes[0,2].set(xlabel='')
axes[0,2].set(ylabel='')

sns.lineplot(data=P_R_C_20_10, x="time_step", y="pressure",  lw=2,ax=axes[1,0])
axes[1,0].set_title ("R=5, C=20", fontsize=18)
axes[1,0].set(xlabel='')

sns.lineplot(data=P_R_C_20_20, x="time_step", y="pressure",  lw=2,ax=axes[1,1])
axes[1,1].set_title ("R=20, C=20", fontsize=18)
axes[1,1].set(xlabel='')
axes[1,1].set(ylabel='')

sns.lineplot(data=P_R_C_20_50, x="time_step", y="pressure",  lw=2,ax=axes[1,2])
axes[1,2].set_title ("R=50, C=20", fontsize=18)
axes[1,2].set(xlabel='')
axes[1,2].set(ylabel='')

sns.lineplot(data=P_R_C_50_10, x="time_step", y="pressure",  lw=2,ax=axes[2,0])
axes[2,0].set_title ("R=5, C=50", fontsize=18)

sns.lineplot(data=P_R_C_50_20, x="time_step", y="pressure",  lw=2,ax=axes[2,1])
axes[2,1].set_title ("R=20, C=50", fontsize=18)
axes[2,1].set(ylabel='')

sns.lineplot(data=P_R_C_50_50, x="time_step", y="pressure",  lw=2,ax=axes[2,2])
axes[2,2].set_title ("R=50, C=50", fontsize=18)
axes[2,2].set(ylabel='')


plt.show();
```




## Time_step : the actual time stamp in decimal

In [11]:
```
data['time_step'].describe().to_frame().transpose()
```

Out[11]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **time_step** | 6036000.0 | 1.307225 | 0.765978 | 0.0 | 0.6429 | 1.308123 | 1.965502 | 2.937238 |

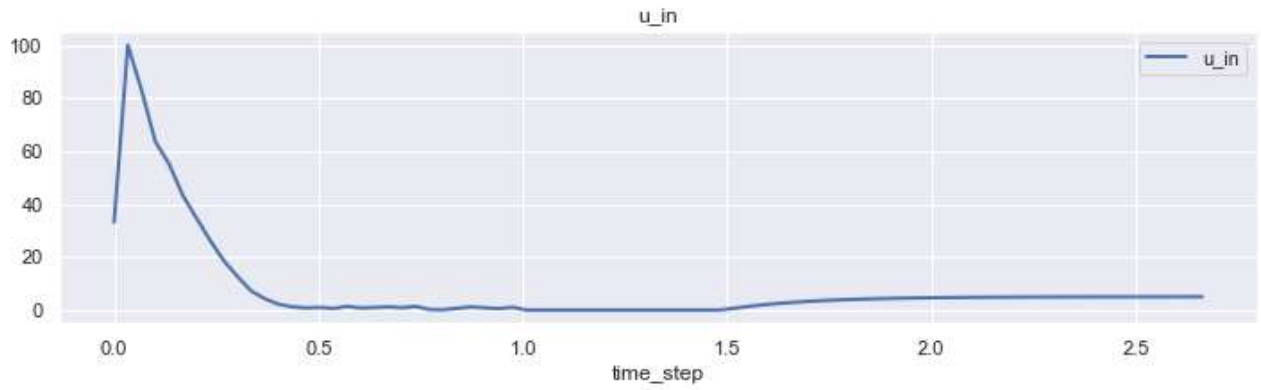
## U_IN - the control input for the inspiratory solenoid valve. Ranges from 0 to 100.

In [12]:
```
data['u_in'].describe().to_frame().transpose()
```

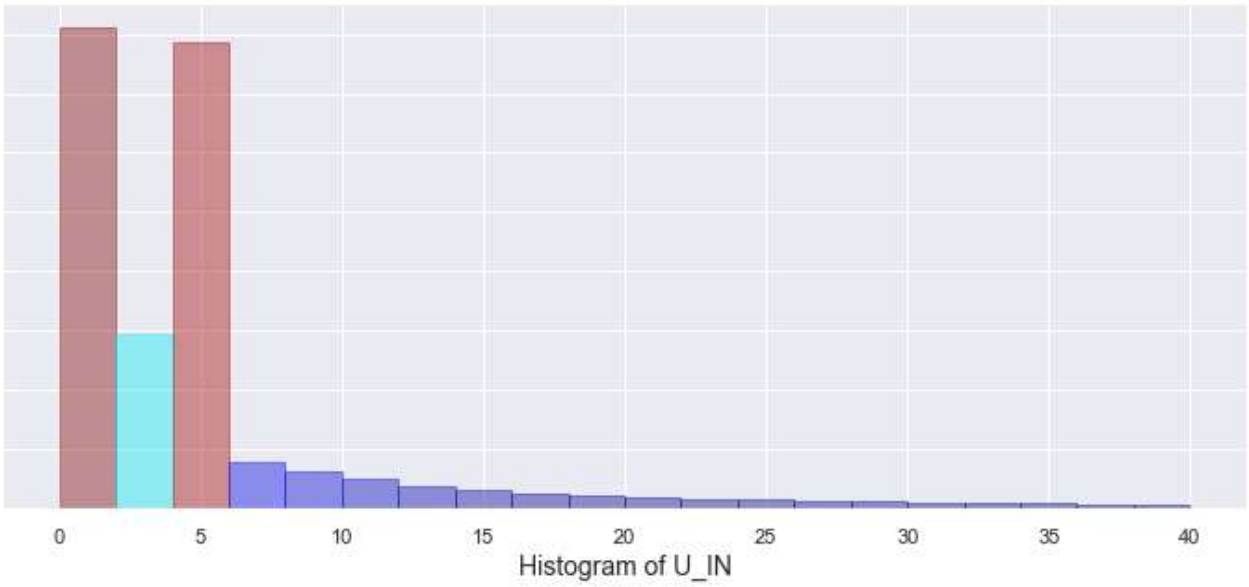
Out[12]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **u_in** | 6036000.0 | 7.321615 | 13.434701 | 0.0 | 0.393662 | 4.386146 | 4.983895 | 100.0 |

In [13]:
```
# example
P_R_C_5_10.plot(x="time_step", y="u_in", kind='line',figsize=(12,3), lw=2, title="u_in"
```




In [14]:
```
plt.figure(figsize = (12,5))
ax = sns.distplot(data['u_in'],
                  bins=20,
                  kde_kws={"clip":(0,40)},
                  hist_kws={"range":(0,40)},
                  color='darkcyan',
                  kde=False);
values = np.array([rec.get_height() for rec in ax.patches])
norm = plt.Normalize(values.min(), values.max())
colors = plt.cm.jet(norm(values))
for rec, col in zip(ax.patches, colors):
    rec.set_color(col)
plt.xlabel("Histogram of U_IN", size=14)
ax.set(yticklabels=[])
plt.show();
```




## U_OUT - the control input for the exploratory solenoid valve. Either 0 or 1.

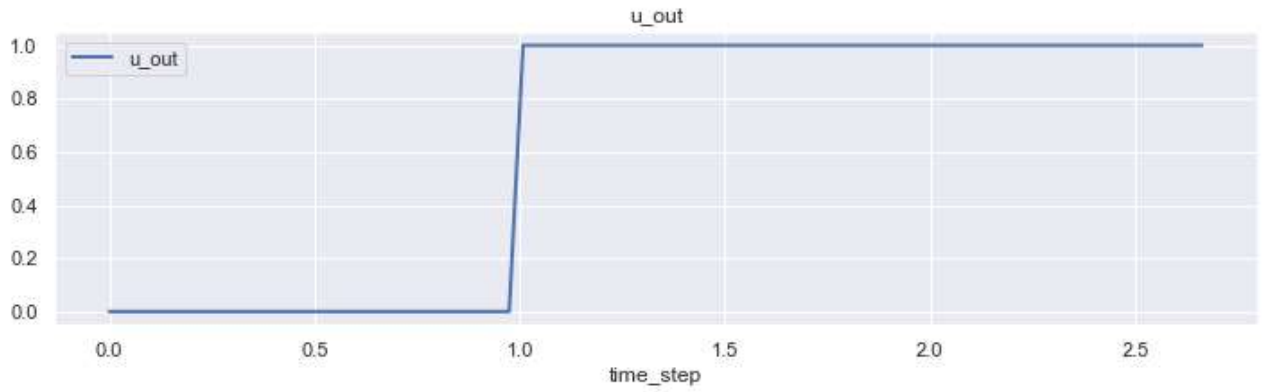
In [15]:
```
data['u_out'].describe().to_frame().transpose()
```

Out[15]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **u_out** | 6036000.0 | 0.620449 | 0.485275 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 |

In [16]:
```
# example
P_R_C_5_10.plot(x="time_step", y="u_out", kind='line',figsize=(12,3), lw=2, title="u_ou
```



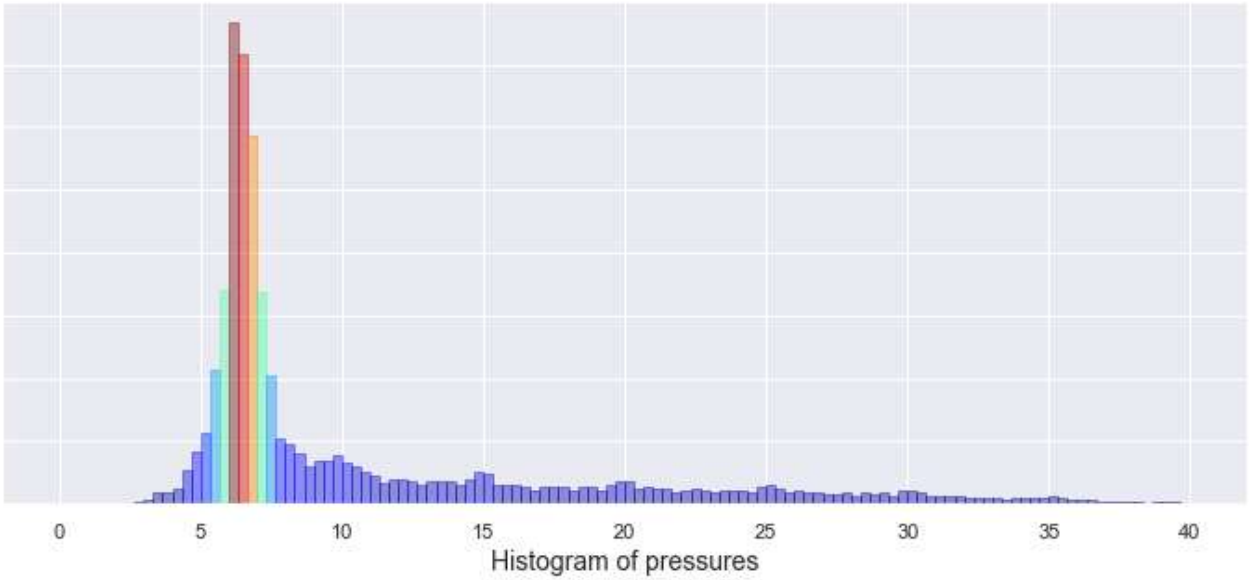

## Pressure - the airway pressure measured in the respiratory circuit, measured in cmH2O.

In [17]:
```
data['pressure'].describe().to_frame().transpose()
```

Out[17]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **pressure** | 6036000.0 | 11.220408 | 8.109703 | -1.895744 | 6.329607 | 7.032628 | 13.64103 | 64.820992 |

In [18]:
```
plt.figure(figsize = (12,5))
ax = sns.distplot(data['pressure'],
                  bins=120,
                  kde_kws={"clip":(0,40)},
                  hist_kws={"range":(0,40)},
                  color='darkcyan',
                  kde=False);
values = np.array([rec.get_height() for rec in ax.patches])
norm = plt.Normalize(values.min(), values.max())
colors = plt.cm.jet(norm(values))
for rec, col in zip(ax.patches, colors):
    rec.set_color(col)
plt.xlabel("Histogram of pressures", size=14)
ax.set(yticklabels=[])
plt.show();
```




# Feature Engineering

In [19]:
```
def FF(data,Log=False):

    # Adding Last Value of U_in column:
    data['u_in_last_value'] = data.groupby('breath_id')['u_in'].transform('last')

    # Adding Lag as time series # https://www.analyticsvidhya.com/blog/2019/12/6-powerf
    data['u_in_lag1'] = data.groupby('breath_id')['u_in'].shift(1)
    data['u_out_lag1'] = data.groupby('breath_id')['u_out'].shift(1)

    #One Hot Encoding
    encoded_columns = pd.get_dummies(data['R'].astype(str)+'_r')
    data = data.join(encoded_columns).drop('R', axis=1)

    encoded_columns = pd.get_dummies(data['C'].astype(str)+'_c')
    data = data.join(encoded_columns).drop('C', axis=1)

    #Log Transform

    if (Log==True):
        data['log_time_step'] = (data['time_step']+1).transform(np.log)
        data['log_u_in'] = (data['u_in']+1).transform(np.log)

    #Fill NAN generated by Lag
    data = data.fillna(0)
    #NAN check
    for x in data.columns:
        print(data[x].isna().sum(),'NaN values for column: {0} : '.format(x))

    return data
```

In [20]:
```
FF_data=FF(data)
FF_data.head()
```

```
0 NaN values for column: id :
0 NaN values for column: breath_id :
```

```
0 NaN values for column: time_step :
0 NaN values for column: u_in :
0 NaN values for column: u_out :
0 NaN values for column: pressure :
0 NaN values for column: u_in_last_value :
0 NaN values for column: u_in_lag1 :
0 NaN values for column: u_out_lag1 :
0 NaN values for column: 20_r :
0 NaN values for column: 50_r :
0 NaN values for column: 5_r :
0 NaN values for column: 10_c :
0 NaN values for column: 20_c :
0 NaN values for column: 50_c :
```

Out[20]:

| | id | breath_id | time_step | u_in | u_out | pressure | u_in_last_value | u_in_lag1 | u_out_lag1 | 20_r | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 0.000000 | 0.083334 | 0 | 5.837492 | 4.987079 | 0.000000 | 0.0 | 1 | |
| **1** | 2 | 1 | 0.033652 | 18.383041 | 0 | 5.907794 | 4.987079 | 0.083334 | 0.0 | 1 | |
| **2** | 3 | 1 | 0.067514 | 22.509278 | 0 | 7.876254 | 4.987079 | 18.383041 | 0.0 | 1 | |
| **3** | 4 | 1 | 0.101542 | 22.808822 | 0 | 11.742872 | 4.987079 | 22.509278 | 0.0 | 1 | |
| **4** | 5 | 1 | 0.135756 | 25.355850 | 0 | 12.234987 | 4.987079 | 22.808822 | 0.0 | 1 | |

### Final Test Train Slipt

In [21]:

```
columns = [col for col in FF_data.columns if col not in ['id', 'breath_id', 'pressure']
X = FF_data[columns]
y = FF_data['pressure']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
del FF_data, X, y, data
```

```
(4044120, 12) (1991880, 12) (4044120,) (1991880,)
```

In [ ]:

```
```

# Training Model (LGBM)

In [22]:

```
params = {'objective': 'regression',
          'learning_rate': 0.03,
          "boosting_type": "gbdt",
          "metric": 'mae',
          'n_jobs': -1,
          'min_data_in_leaf':32,
          'num_leaves':512,
         }
```

In [23]:

```
model = lgb.LGBMRegressor(**params, n_estimators=10000)

model.fit(X_train,
          y_train,
```

```
          eval_set=[(X_train, y_train)],
          verbose=100,
          early_stopping_rounds=15)
```

```
[LightGBM] [Warning] min_data_in_leaf is set=32, min_child_samples=20 will be ignored. C
urrent value: min_data_in_leaf=32
[100]   training's l1: 1.57241
[200]   training's l1: 1.43624
[300]   training's l1: 1.3998
[400]   training's l1: 1.3783
[500]   training's l1: 1.36458
[600]   training's l1: 1.35191
[700]   training's l1: 1.34198
[800]   training's l1: 1.3327
[900]   training's l1: 1.32439
[1000]  training's l1: 1.31696
[1100]  training's l1: 1.30973
[1200]  training's l1: 1.30341
[1300]  training's l1: 1.29728
[1400]  training's l1: 1.29157
[1500]  training's l1: 1.28666
[1600]  training's l1: 1.28245
[1700]  training's l1: 1.27764
[1800]  training's l1: 1.27369
[1900]  training's l1: 1.26957
[2000]  training's l1: 1.26604
[2100]  training's l1: 1.26217
[2200]  training's l1: 1.2582
[2300]  training's l1: 1.25491
[2400]  training's l1: 1.2514
[2500]  training's l1: 1.24797
[2600]  training's l1: 1.24485
[2700]  training's l1: 1.24193
[2800]  training's l1: 1.23918
[2900]  training's l1: 1.23649
[3000]  training's l1: 1.23381
[3100]  training's l1: 1.23105
[3200]  training's l1: 1.22841
[3300]  training's l1: 1.22563
[3400]  training's l1: 1.22326
[3500]  training's l1: 1.22062
[3600]  training's l1: 1.21816
[3700]  training's l1: 1.21561
[3800]  training's l1: 1.21318
[3900]  training's l1: 1.21095
[4000]  training's l1: 1.20874
[4100]  training's l1: 1.20678
[4200]  training's l1: 1.20464
[4300]  training's l1: 1.20272
[4400]  training's l1: 1.20068
[4500]  training's l1: 1.19854
[4600]  training's l1: 1.19647
[4700]  training's l1: 1.19437
[4800]  training's l1: 1.19209
[4900]  training's l1: 1.19033
[5000]  training's l1: 1.1884
[5100]  training's l1: 1.18679
[5200]  training's l1: 1.18524
[5300]  training's l1: 1.18375
[5400]  training's l1: 1.18214
```

```
[5500]  training's l1: 1.18018
[5600]  training's l1: 1.17861
[5700]  training's l1: 1.1772
[5800]  training's l1: 1.1758
[5900]  training's l1: 1.17436
[6000]  training's l1: 1.17298
[6100]  training's l1: 1.17177
[6200]  training's l1: 1.17052
[6300]  training's l1: 1.16894
[6400]  training's l1: 1.16775
[6500]  training's l1: 1.16647
[6600]  training's l1: 1.16524
[6700]  training's l1: 1.16384
[6800]  training's l1: 1.16249
[6900]  training's l1: 1.16121
[7000]  training's l1: 1.15991
[7100]  training's l1: 1.15858
[7200]  training's l1: 1.15722
[7300]  training's l1: 1.15581
[7400]  training's l1: 1.15441
[7500]  training's l1: 1.15294
[7600]  training's l1: 1.15145
[7700]  training's l1: 1.15013
[7800]  training's l1: 1.14875
[7900]  training's l1: 1.1475
[8000]  training's l1: 1.14603
[8100]  training's l1: 1.14465
[8200]  training's l1: 1.14331
[8300]  training's l1: 1.14212
[8400]  training's l1: 1.14072
[8500]  training's l1: 1.13941
[8600]  training's l1: 1.13836
[8700]  training's l1: 1.13729
[8800]  training's l1: 1.13619
[8900]  training's l1: 1.13507
[9000]  training's l1: 1.13388
[9100]  training's l1: 1.13278
[9200]  training's l1: 1.13153
[9300]  training's l1: 1.13031
[9400]  training's l1: 1.12936
[9500]  training's l1: 1.12836
[9600]  training's l1: 1.12736
[9700]  training's l1: 1.12634
[9800]  training's l1: 1.12533
[9900]  training's l1: 1.12434
[10000] training's l1: 1.12329
```

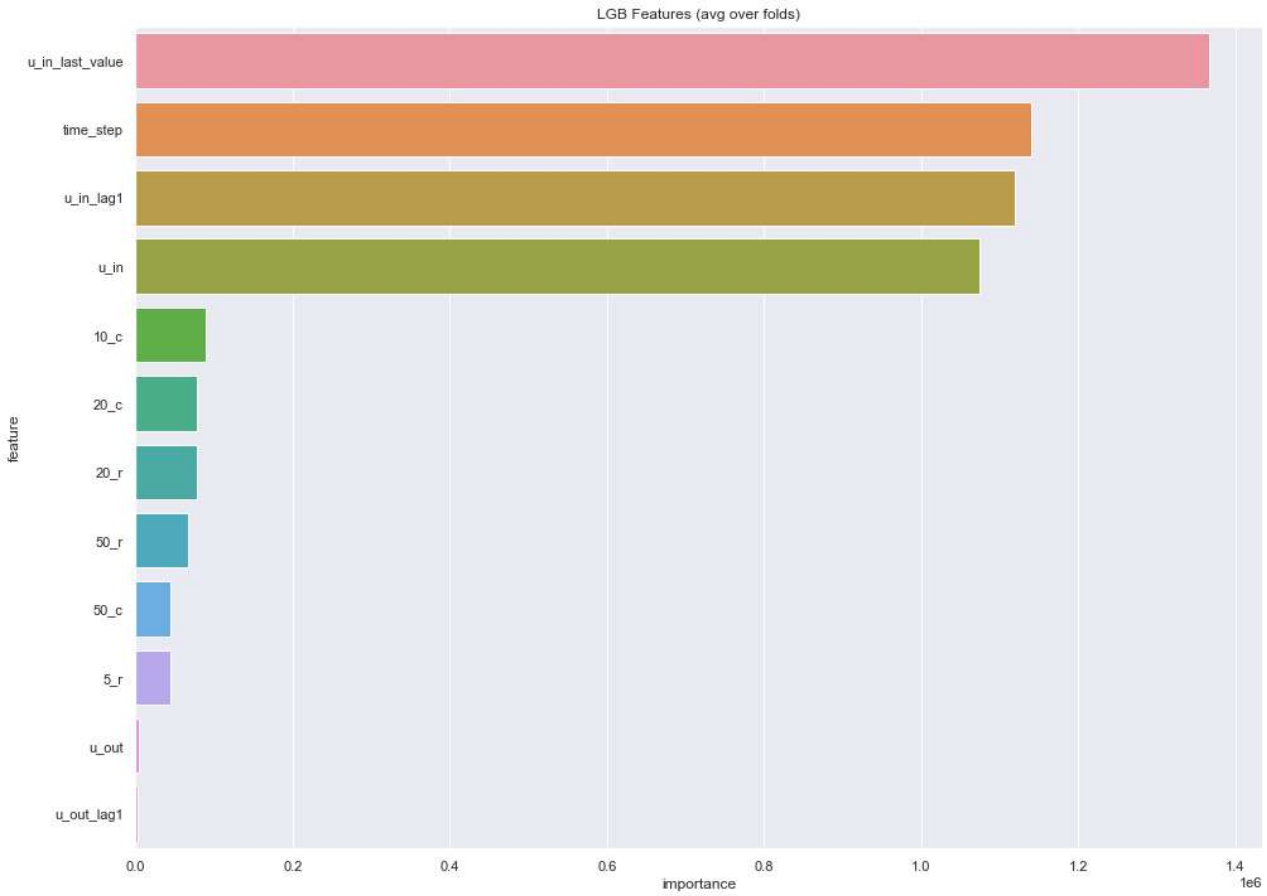
Out[23]:
```
LGBMRegressor(learning_rate=0.03, metric='mae', min_data_in_leaf=32,
              n_estimators=10000, num_leaves=512, objective='regression')
```

In [25]:
```
print(metrics.mean_absolute_error(y_test, model.predict(X_test)))
```

```
1.3062220344816675
```

In [26]:
```
feature_importance = pd.DataFrame()
feature_importance["feature"] = columns
feature_importance["importance"] = model.feature_importances_
cols = feature_importance[["feature", "importance"]].groupby("feature").mean().sort_val
best_features = feature_importance.loc[feature_importance.feature.isin(cols)]
```

```
plt.figure(figsize=(16, 12));
sns.barplot(x="importance", y="feature", data=best_features.sort_values(by="importance"
plt.title('LGB Features (avg over folds)');
```




# Work In Progess

1. Hyparameter Tuning.
2. Explore LSTM and XGBM
3. Fold and Cross validation

In [ ]: