Binary Classification of Machine Failures

In [1]:
```
!wget http://bit.ly/3ZLyF82 -O CSS.css -q

from IPython.core.display import HTML
with open('./CSS.css', 'r') as file:
    custom_css = file.read()

HTML(custom_css)
```

Out[1]:

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import re as re
from collections import Counter

from tqdm.auto import tqdm
import math
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, G
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix, Co
import warnings
warnings.filterwarnings('ignore')

import time
from xgboost import XGBClassifier
%matplotlib inline
tqdm.pandas()

rc = {
    "axes.facecolor": "#FFF9ED",
    "figure.facecolor": "#FFF9ED",
    "axes.edgecolor": "#000000",
    "grid.color": "#EBEBE7",
    "font.family": "serif",
    "axes.labelcolor": "#000000",
    "xtick.color": "#000000",
    "ytick.color": "#000000",
    "grid.alpha": 0.4
}

sns.set(rc=rc)

from colorama import Style, Fore
red = Style.BRIGHT + Fore.RED
blu = Style.BRIGHT + Fore.BLUE
mgt = Style.BRIGHT + Fore.MAGENTA
gld = Style.BRIGHT + Fore.YELLOW
res = Style.RESET_ALL
```

In [3]:

```
train = pd.read_csv('/input/playground-series-s3e17/train.csv')
test = pd.read_csv('/input/playground-series-s3e17/test.csv')
```

EDA

In [4]:
```python
# summary table function
pd.options.display.float_format = '{:,.2f}'.format
def summary(df):
    print(f'data shape: {df.shape}')
    summ = pd.DataFrame(df.dtypes, columns=['data type'])
    summ['#missing'] = df.isnull().sum().values
    summ['%missing'] = df.isnull().sum().values / len(df) * 100
    summ['#unique'] = df.nunique().values
    desc = pd.DataFrame(df.describe(include='all').transpose())
    summ['min'] = desc['min'].values
    summ['max'] = desc['max'].values
    summ['average'] = desc['mean'].values
    summ['standard_deviation'] = desc['std'].values
    summ['first value'] = df.loc[0].values
    summ['second value'] = df.loc[1].values
    summ['third value'] = df.loc[2].values

    return summ
```

In [5]:
```python
summary(train).style.background_gradient(cmap='YlOrBr')
```

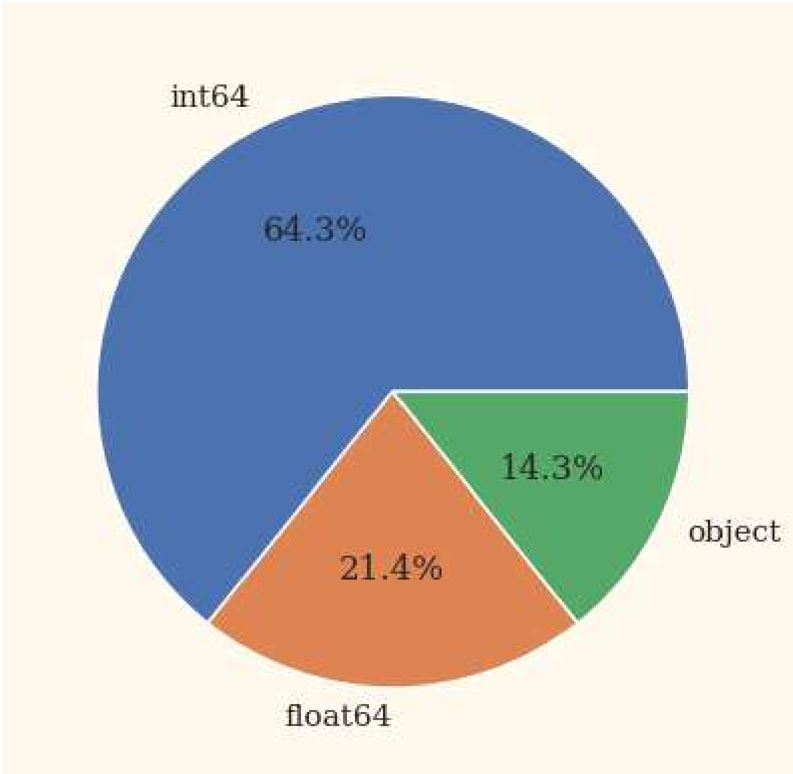
```
data shape: (136429, 14)
```

Out[5]:

	data type	#missing	%missing	#unique	min	max	average	sta
id	int64	0	0.000000	136429	0.000000	136428.000000	68214.000000	
Product ID	object	0	0.000000	9976	nan	nan	nan	
Type	object	0	0.000000	3	nan	nan	nan	
Air temperature [K]	float64	0	0.000000	95	295.300000	304.400000	299.862776	
Process temperature [K]	float64	0	0.000000	81	305.800000	313.800000	309.941070	
Rotational speed [rpm]	int64	0	0.000000	952	1181.000000	2886.000000	1520.331110	
Torque [Nm]	float64	0	0.000000	611	3.800000	76.600000	40.348643	
Tool wear [min]	int64	0	0.000000	246	0.000000	253.000000	104.408901	
Machine failure	int64	0	0.000000	2	0.000000	1.000000	0.015744	
TWF	int64	0	0.000000	2	0.000000	1.000000	0.001554	
HDF	int64	0	0.000000	2	0.000000	1.000000	0.005160	
PWF	int64	0	0.000000	2	0.000000	1.000000	0.002397	
OSF	int64	0	0.000000	2	0.000000	1.000000	0.003958	
RNF	int64	0	0.000000	2	0.000000	1.000000	0.002258	

In [6]:
```
train.dtypes.value_counts().plot(kind='pie',autopct='%.1f%%')
```
Out[6]:

```
<Axes: >
```

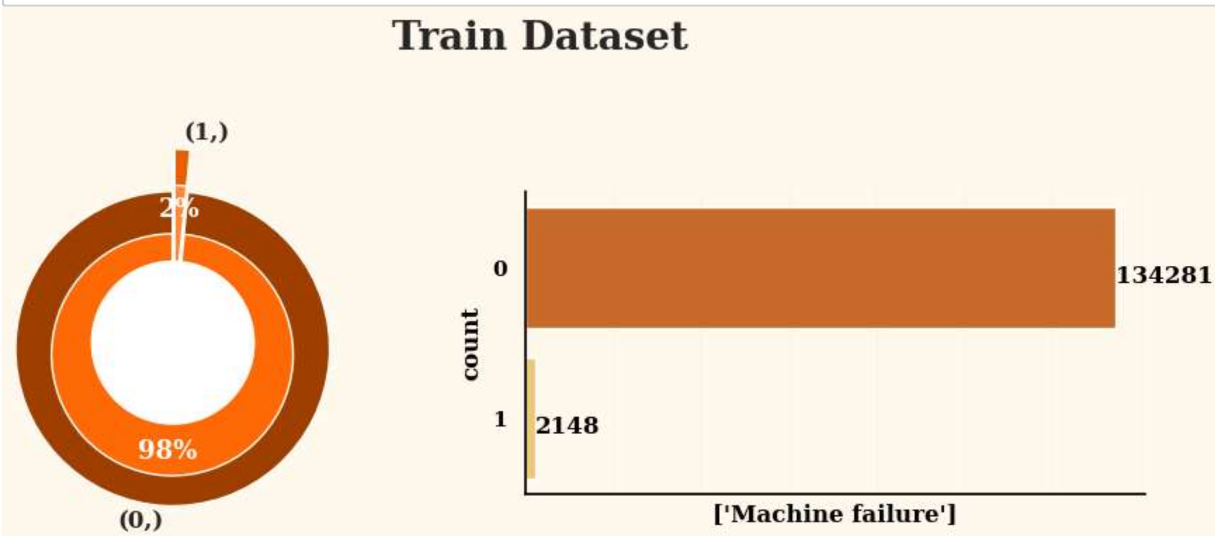


In [7]:
```
# select numerical and categorical variables respectively.
num_cols = test.select_dtypes(include=['float64','int64']).columns.tolist()
cat_cols = test.select_dtypes(include=['object']).columns.tolist()
num_cols.remove('id')
all_features = num_cols + cat_cols
```

In [8]:
```python
def plot_count(df: pd.core.frame.DataFrame, col_list: list, title_name: str='Tr
    """Draws the pie and count plots for categorical variables.

    Args:
        df: train or test dataframes
        col_list: a list of the selected categorical variables.
        title_name: 'Train' or 'Test' (default 'Train')

    Returns:
        subplots of size (len(col_list), 2)
    """
    f, ax = plt.subplots(len(col_list), 2, figsize=(10, 4))
    plt.subplots_adjust(wspace=0)

    s1 = df[col_list].value_counts()
    N = len(s1)

    outer_sizes = s1
    inner_sizes = s1/N

    outer_colors = ['#9E3F00', '#eb5e00', '#ff781f', '#ff9752', '#ff9752']
    inner_colors = ['#ff6905', '#ff8838', '#ffa66b']

    ax[0].pie(
        outer_sizes,colors=outer_colors,
        labels=s1.index.tolist(),
        startangle=90,frame=True, radius=1.3,
        explode=([0.05]*(N-1) + [.3]),
        wedgeprops={ 'linewidth' : 1, 'edgecolor' : 'white'},
        textprops={'fontsize': 12, 'weight': 'bold'}
    )

    textprops = {
        'size':13,
        'weight': 'bold',
        'color':'white'
    }

    ax[0].pie(
        inner_sizes, colors=inner_colors,
        radius=1, startangle=90,
        autopct='%1.f%%',explode=([.1]*(N-1) + [.3]),
        pctdistance=0.8, textprops=textprops
    )

    center_circle = plt.Circle((0,0), .68, color='black',
                               fc='white', linewidth=0)
    ax[0].add_artist(center_circle)

    x = s1
    y = [0, 1]
    sns.barplot(
        x=x, y=y, ax=ax[1],
        palette='YlOrBr_r', orient='horizontal'
    )

    ax[1].spines['top'].set_visible(False)
```

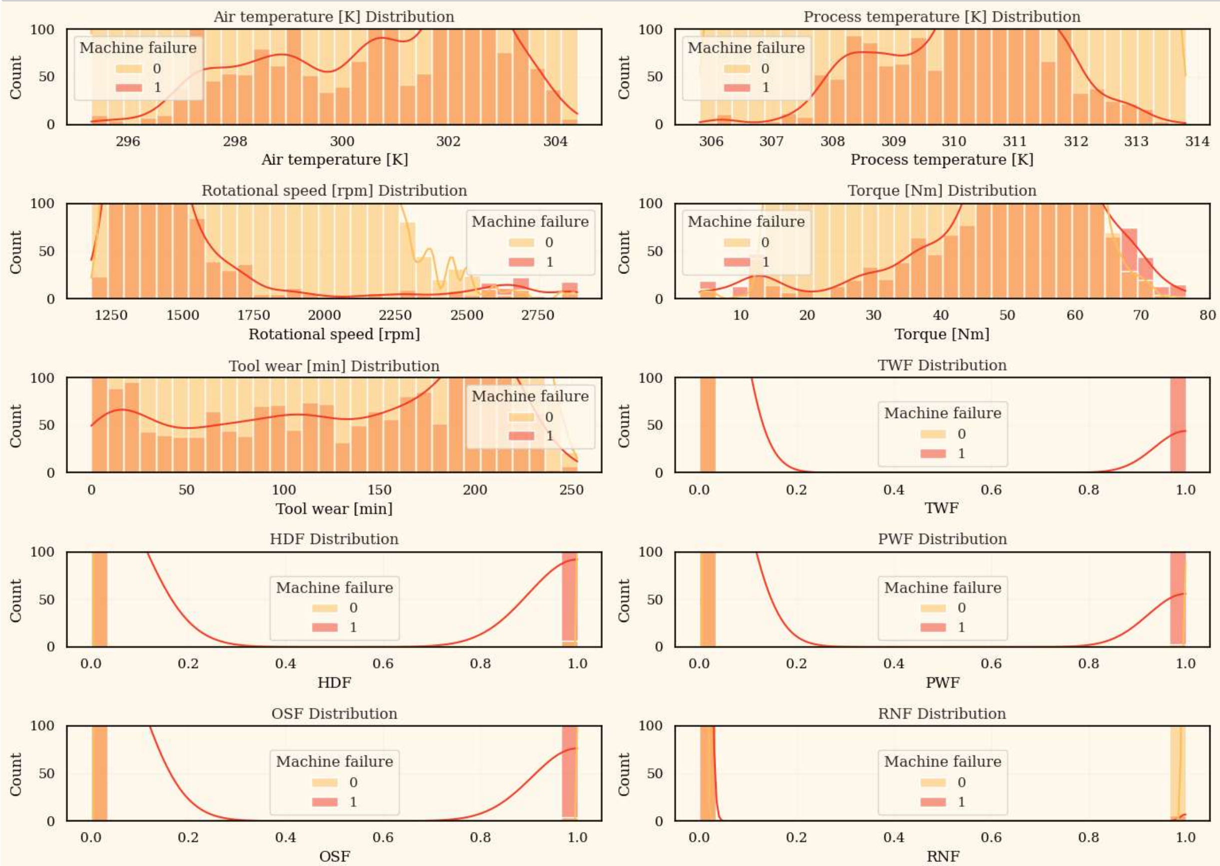
```
    ax[1].spines['right'].set_visible(False)
    ax[1].tick_params(
        axis='x',
        which='both',
        bottom=False,
        labelbottom=False
    )

    for i, v in enumerate(s1):
        ax[1].text(v, i+0.1, str(v), color='black',
                     fontweight='bold', fontsize=12)

#     plt.title(col_list)
    plt.setp(ax[1].get_yticklabels(), fontweight="bold")
    plt.setp(ax[1].get_xticklabels(), fontweight="bold")
    ax[1].set_xlabel(col_list, fontweight="bold", color='black')
    ax[1].set_ylabel('count', fontweight="bold", color='black')

    f.suptitle(f'{title_name} Dataset', fontsize=20, fontweight='bold')
    plt.tight_layout()
    plt.show()
```

In [9]:
```
plot_count(train, ['Machine failure'], 'Train')
```



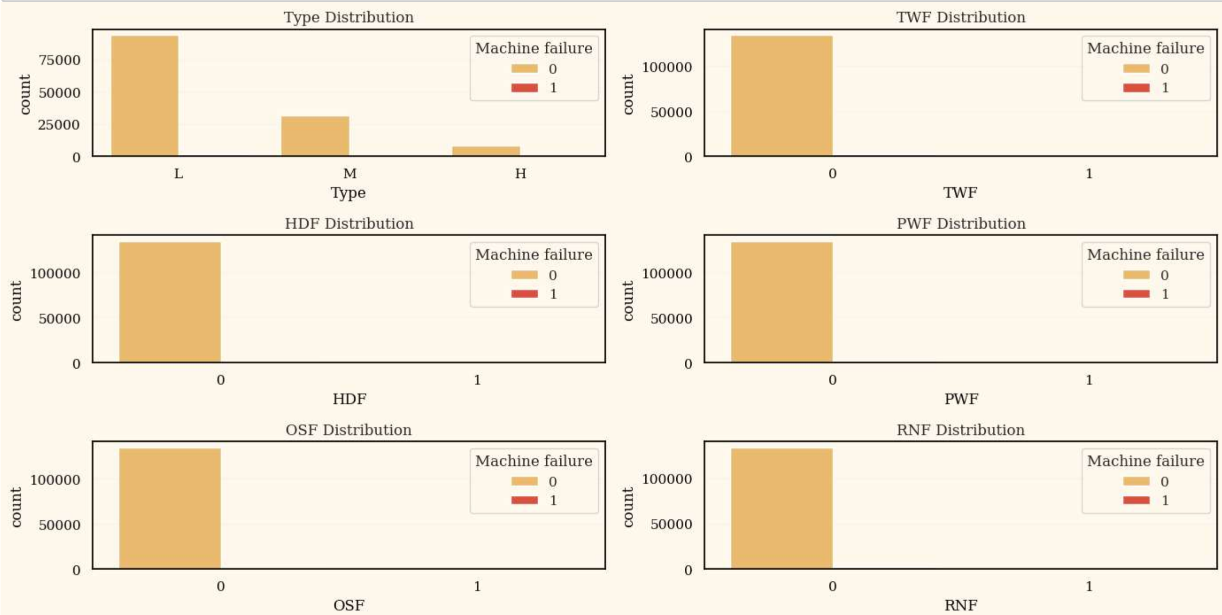
- This dataset is a highly imbalanced dataset

In [10]:
```
plt.figure(figsize=(14,10))
for idx,column in enumerate(num_cols):
    plt.subplot(5,2,idx+1)
    sns.histplot(x=column, hue="Machine failure", data=train,bins=30,kde=True,p
    plt.title(f"{column} Distribution")
    plt.ylim(0,100)
    plt.tight_layout()
```



You can see that, in fact, TWF, HDF, PWF, OSF, RNF variables are binary variables! So, I will add these variables to the cat_cols list.

In [11]:
```
cat_cols = ['Type','TWF', 'HDF', 'PWF', 'OSF', 'RNF']

plt.figure(figsize=(14,9))
for idx,column in enumerate(cat_cols):
    plt.subplot(4,2,idx+1)
    sns.countplot(x=column, hue="Machine failure", data=train, palette="YlOrRd"
    plt.title(f"{column} Distribution")
    plt.tight_layout()
```

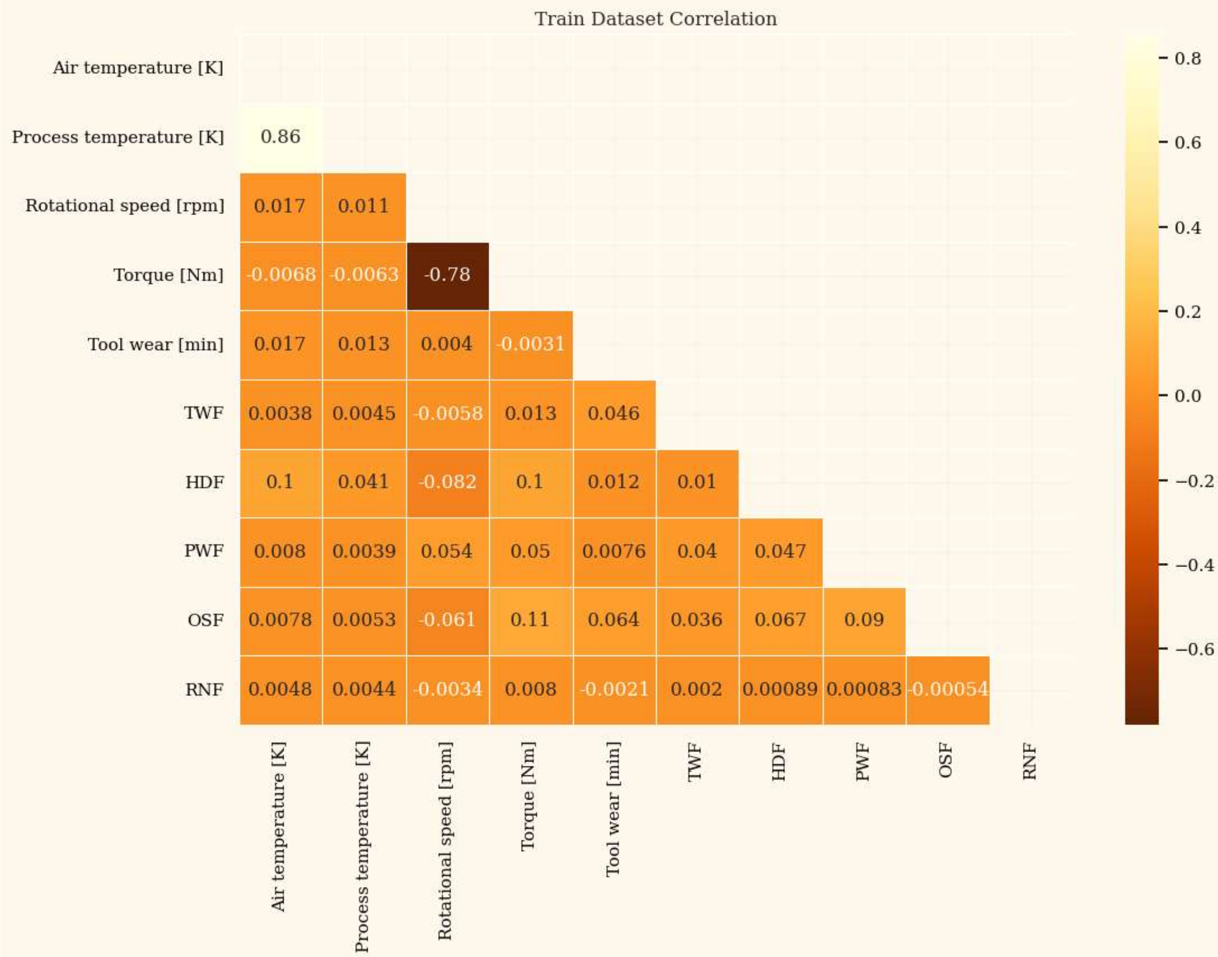


In [12]:
```
# check the distribution of PWF when 'machine failture' equals 1
train[train['Machine failure']==1]['PWF'].value_counts()
```

Out[12]:
```
0    1824
1     324
Name: PWF, dtype: int64
```

Since this is a highly imbalanced dataset, it's difficult to discern the 'failure' on the above bar graph. Therefore, we can verify it using the 'value_counts()' method instead.

```
In [13]: def plot_correlation_heatmap(df: pd.core.frame.DataFrame, title_name: str='Trai

    corr = df.corr()
    fig, axes = plt.subplots(figsize=(12, 8))
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True
    sns.heatmap(corr, mask=mask, linewidths=.5, cmap='YlOrBr_r', annot=True)
    plt.title(title_name)
    plt.show()


plot_correlation_heatmap(train[num_cols], 'Train Dataset Correlation')
```



- Torque and Rotation speed are highly correlated.

Feature engineering and baseline modeling

In [14]:
```python
LABEL_CATEGORICAL = ['Type']
def encode_categ_features(df, categ_colums = LABEL_CATEGORICAL):
    """
    Use the label encoder to encode categorical features...
    Args
        df
        categ_colums
    Returns
        df
    """
    le = LabelEncoder()
    for col in categ_colums:
        df['enc_'+col] = le.fit_transform(df[col])
    df.drop(categ_colums, axis=1, inplace=True)
    return df

train = encode_categ_features(train)
test = encode_categ_features(test)
```

In [15]:
```python
train.columns
```

Out[15]:
```
Index(['id', 'Product ID', 'Air temperature [K]', 'Process temperature
[K]',
       'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]',
       'Machine failure', 'TWF', 'HDF', 'PWF', 'OSF', 'RNF', 'enc_Type'],
      dtype='object')
```

as you cannot contains '[]' in feature name, we need to change it.

In [16]:
```python
train.rename(columns = {'Air temperature [K]': 'air_temp',
                     'Process temperature [K]': 'process_temp',
                     'Rotational speed [rpm]': 'rotational_speed',
                     'Torque [Nm]': 'torque',
                     'Tool wear [min]': 'tool_wear'}, inplace=True)
test.rename(columns = {'Air temperature [K]': 'air_temp',
                     'Process temperature [K]': 'process_temp',
                     'Rotational speed [rpm]': 'rotational_speed',
                     'Torque [Nm]': 'torque',
                     'Tool wear [min]': 'tool_wear'}, inplace=True)
```

In [17]:
```
train.columns
```

Out[17]:
```
Index(['id', 'Product ID', 'air_temp', 'process_temp', 'rotational_spee
d',
       'torque', 'tool_wear', 'Machine failure', 'TWF', 'HDF', 'PWF', 'OS
F',
       'RNF', 'enc_Type'],
      dtype='object')
```

In [18]:
```
all_features_final = ['air_temp', 'process_temp', 'rotational_speed',
       'torque', 'tool_wear', 'TWF', 'HDF', 'PWF', 'OSF',
       'RNF', 'enc_Type']
```

In [19]:
```python
# create fuctions for evaluation
def f_importance_plot(f_imp):
    fig = plt.figure(figsize=(12, 0.20*len(f_imp)))
    plt.title('Feature importances', size=16, y=1.05,
              fontweight='bold', color='#444444')
    a = sns.barplot(data=f_imp, x='avg_imp', y='feature',
                    palette='YlOrBr_r', linestyle="-",
                    linewidth=0.5, edgecolor="black")
    plt.xlabel('')
    plt.xticks([])
    plt.ylabel('')
    plt.yticks(size=11, color='#444444')

    for j in ['right', 'top', 'bottom']:
        a.spines[j].set_visible(False)
    for j in ['left']:
        a.spines[j].set_linewidth(0.5)
    plt.tight_layout()
    plt.show()

def show_confusion_roc(oof: list) -> None:
    """Draws a confusion matrix and roc_curve with AUC score.

        Args:
            oof: predictions for each fold stacked. (list of tuples)

        Returns:
            None
    """

    f, ax = plt.subplots(1, 2, figsize=(13.3, 4))
    df = pd.DataFrame(np.concatenate(oof), columns=['id', 'preds', 'target']).s
    df.index = df.index.astype(int)
    cm = confusion_matrix(df.target, df.preds.ge(0.5).astype(int))
    cm_display = ConfusionMatrixDisplay(cm).plot(cmap='YlOrBr_r', ax=ax[0])
    ax[0].grid(False)
    RocCurveDisplay.from_predictions(df.target, df.preds, color='#20BEFF', ax=a
    plt.tight_layout();

def get_mean_auc(oof: np.array):
    """oof: ['val_idx', 'preds', 'target']"""
    oof = pd.DataFrame(np.concatenate(oof), columns=['id', 'preds', 'target']).
    oof.index = oof.index.astype(int)
    mean_val_auc = roc_auc_score(oof.target, oof.preds)
    return mean_val_auc
```

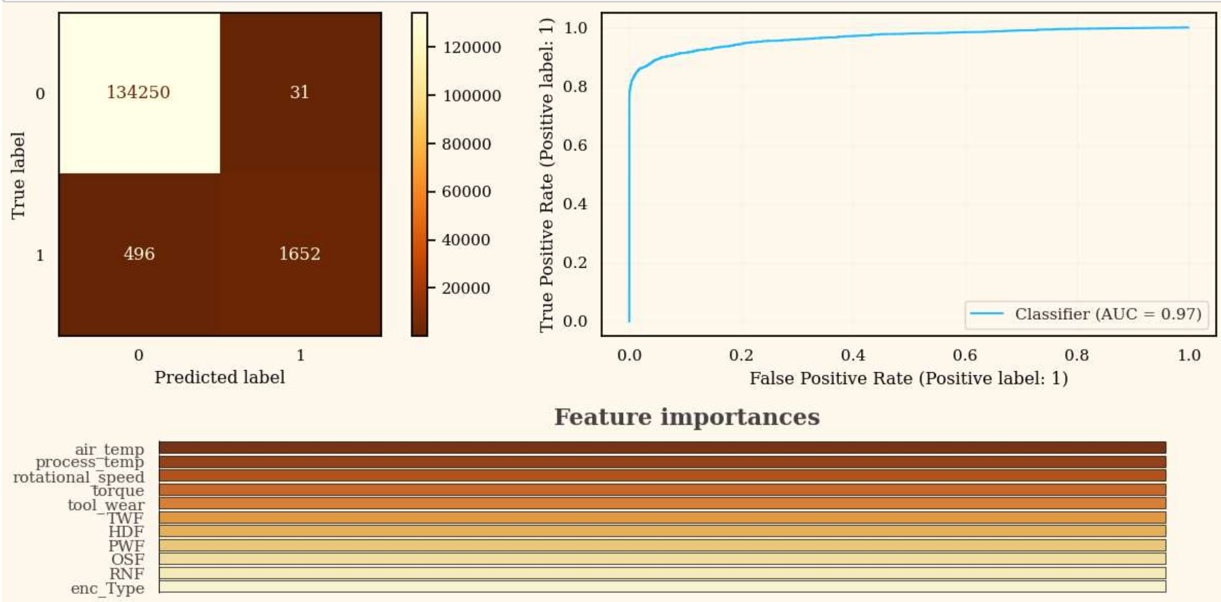
In [20]:

```
FOLDS = 10
SEED = 1004
xgb_models = []
xgb_oof = []
test = test[all_features_final]
predictions = np.zeros(len(test))
f_imp = []

counter = 1
X = train[all_features_final]
y = train['Machine failure']
skf = StratifiedKFold(n_splits=FOLDS, shuffle=True, random_state=SEED)
for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
    if (fold + 1)%5 == 0 or (fold + 1) == 1:
        print(f'{"#"*24} Training FOLD {fold+1} {"#"*24}')

    X_train, y_train = X.iloc[train_idx], y.iloc[train_idx]
    X_valid, y_valid = X.iloc[val_idx], y.iloc[val_idx]
    watchlist = [(X_train, y_train), (X_valid, y_valid)]

    # XGboost model and fit
    model = XGBClassifier(n_estimators=1000, n_jobs=-1, max_depth=4, eta=0.2, c
    model.fit(X_train, y_train, eval_set=watchlist, early_stopping_rounds=300,

    val_preds = model.predict_proba(X_valid)[:, 1]
    val_score = roc_auc_score(y_valid, val_preds)
    best_iter = model.best_iteration

    idx_pred_target = np.vstack([val_idx,  val_preds, y_valid]).T # shape(len(v
    f_imp.append({i: j for i in model.feature_names_in_ for j in model.feature_
    print(f'{" "*20} auc:{blu}{val_score:.5f}{res} {" "*6} best iteration  :{bl

    xgb_oof.append(idx_pred_target)
    xgb_models.append(model)
#     test_preds = model.predict_proba(test)[:,1] / FOLDS
#     predictions += test_preds
    if val_score > 0.80:
        test_preds = model.predict_proba(test)[:,1]
        predictions += test_preds
        counter += 1

predictions /= counter
mean_val_auc = get_mean_auc(xgb_oof)
print('*'*45)
print(f'{red}Mean{res} AUC: {red}{mean_val_auc:.5f}{res}')
```

```
######################## Training FOLD 1 ########################
                     auc:0.96565       best iteration  :286
                     auc:0.97312       best iteration  :147
                     auc:0.97190       best iteration  :125
                     auc:0.95767       best iteration  :88
######################## Training FOLD 5 ########################
                     auc:0.97075       best iteration  :60
                     auc:0.96047       best iteration  :61
                     auc:0.96885       best iteration  :237
                     auc:0.96164       best iteration  :201
                     auc:0.96107       best iteration  :128
######################## Training FOLD 10 ########################
                     auc:0.97498       best iteration  :231
*********************************************
Mean AUC: 0.96616
```

Evaluation

In [21]:
```
show_confusion_roc(xgb_oof)
f_imp_df = pd.DataFrame(f_imp).mean().reset_index()
f_imp_df.columns = ['feature', 'avg_imp']
f_importance_plot(f_imp_df)
```



- It seems that air_temp, process-temp, rotational speed, torque, tool wear have strong predictive power.

I skipped detailed feature engineering and hyperparameter tuning, which could enhance the model's performance.

You can apply this template to almost any classification problem.