In [1]:
```
# Importing the required libraries
import numpy as np
import pandas as pd
import pyodbc
import time
from datetime import timedelta
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sqlalchemy import create_engine

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix , classification_report, f1_score,\
                            accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import GridSearchCV

SEED = 0
```

In [2]:
```
# establish connection
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=LAPTOP-UQH1JSU8;'
                      'Database=Summative Capstone;'
                      'Trusted_Connection=yes;')
cursor = conn.cursor()
```

In [3]:
```
newblade1 = pd.read_sql_query('SELECT * FROM dbo.NewBlade001',conn)
newblade2 = pd.read_sql_query('SELECT * FROM dbo.NewBlade002',conn)
newblade3 = pd.read_sql_query('SELECT * FROM dbo.NewBlade003',conn)
wornblade1 = pd.read_sql_query('SELECT * FROM dbo.WornBlade001',conn)
wornblade2 = pd.read_sql_query('SELECT * FROM dbo.WornBlade002',conn)
wornblade3 = pd.read_sql_query('SELECT * FROM dbo.WornBlade003',conn)
```

In [4]:
```
newblade1["Condition"] = 1
newblade2["Condition"] = 1
newblade3["Condition"] = 1
wornblade1["Condition"] = 0
wornblade2["Condition"] = 0
wornblade3["Condition"] = 0
```

In [5]:
```
df = pd.concat([newblade1,
                newblade2,
                newblade3,
                wornblade1,
                wornblade2,
                wornblade3])
```

In [6]:
```
df.head()
```

Out[6]:

	Timestamp	pCut Motor: Torque	pCut CTRL Position controller: Lag error	pCut CTRL Position controller: Actual position	pCut CTRL Position controller: Actual speed	pSvolFilm CTRL Position controller: Actual position	pSvolFilm CTRL Position controller: Actual speed	pSvolFilm CTRL Position controller: Lag error	Condition
0	-0.188	-0.112131	-0.002490	-884606.0	0.000000	11128.0	2.504289	0.261085	1
1	-0.184	-0.088931	-0.003863	-884606.0	17.166138	11128.0	-2.504289	0.260083	1
2	-0.180	-0.115141	0.001630	-884606.0	-6.866455	11128.0	7.513016	0.259081	1
3	-0.176	-0.111815	0.003003	-884606.0	-13.732910	11128.0	-2.504289	0.260083	1
4	-0.172	-0.130970	0.004376	-884606.0	-6.866455	11128.0	0.000000	0.261085	1

In [7]:
```
df.info()
```

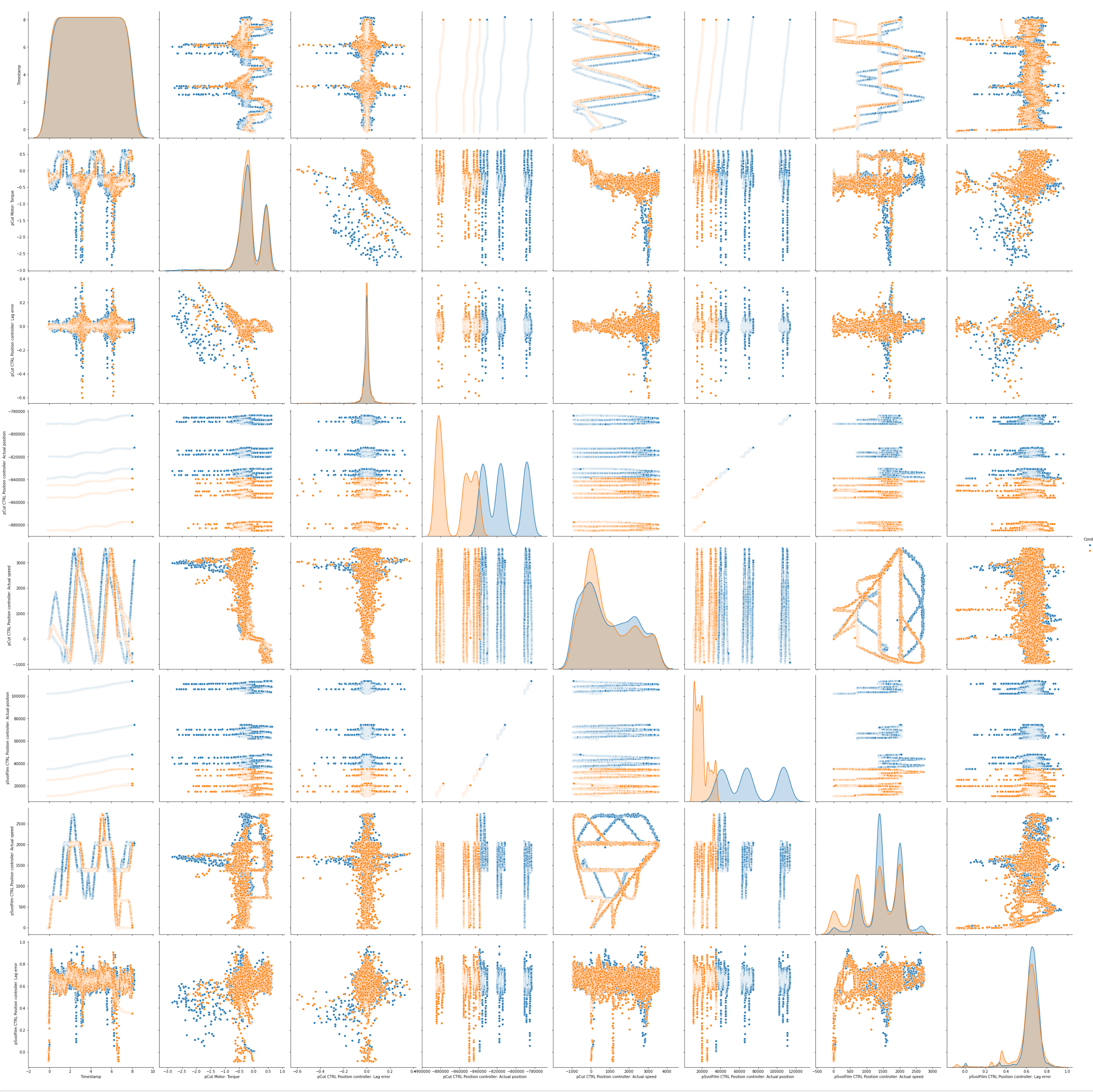
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14336 entries, 0 to 2047
Data columns (total 9 columns):
 #   Column                                              Non-Null Count  Dtype  
---  ------                                              --------------  -----  
 0   Timestamp                                           14336 non-null  float64
 1   pCut Motor: Torque                                  14336 non-null  float64
 2    pCut CTRL Position controller: Lag error           14336 non-null  float64
 3    pCut CTRL Position controller: Actual position     14336 non-null  float64
 4    pCut CTRL Position controller: Actual speed        14336 non-null  float64
 5    pSvolFilm CTRL Position controller: Actual position 14336 non-null  float64
 6    pSvolFilm CTRL Position controller: Actual speed   14336 non-null  float64
 7    pSvolFilm CTRL Position controller: Lag error      14336 non-null  float64
 8   Condition                                           14336 non-null  int64  
dtypes: float64(8), int64(1)
memory usage: 1.1 MB
```

In [8]:
```
df['Condition'].value_counts()
```

Out[8]:
```
1    8192
0    6144
Name: Condition, dtype: int64
```

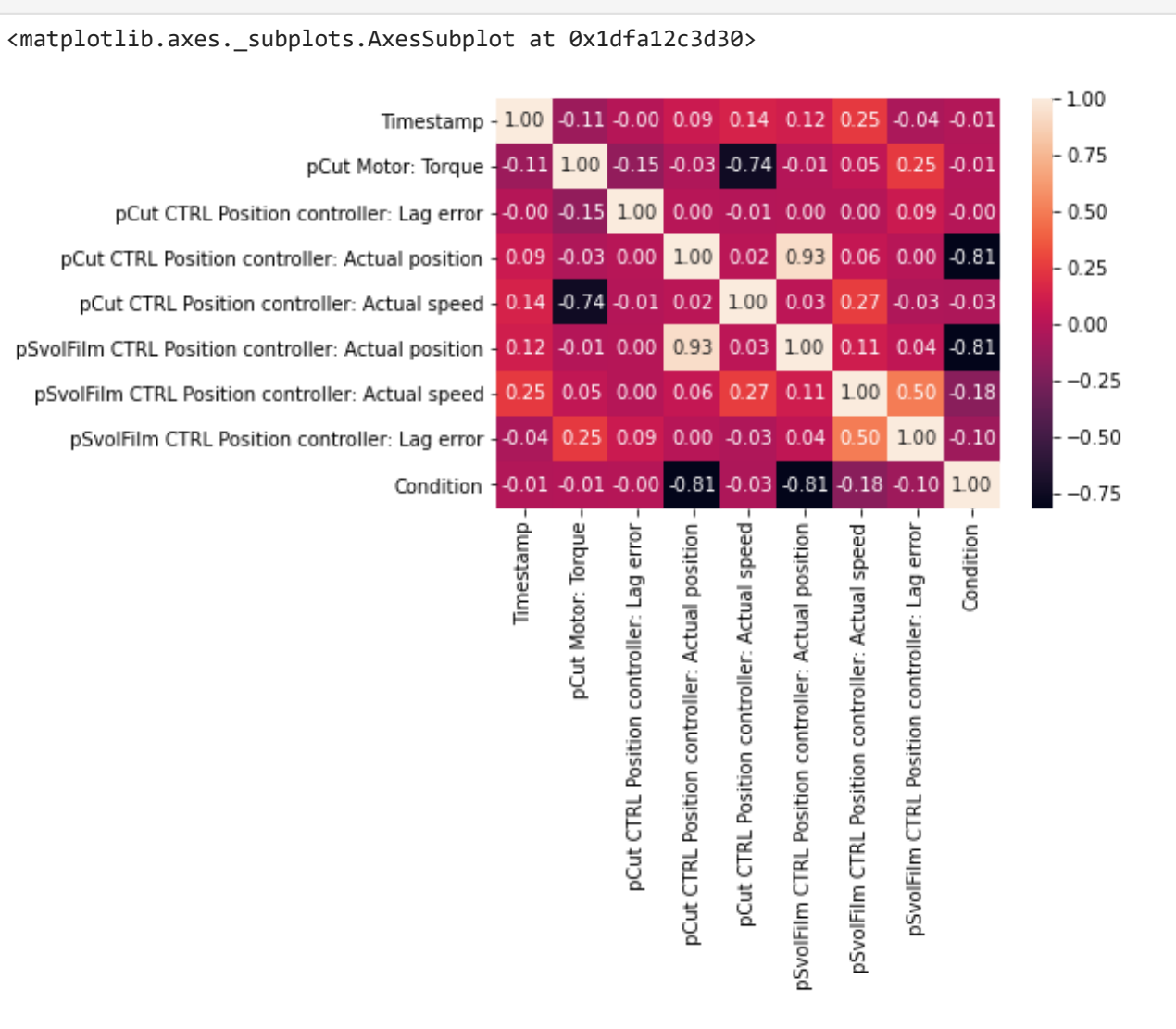
In [9]:
```
sns.pairplot(data=df, hue='Condition', height =5)
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1df998dce80>
```



In [10]:
```
sns.heatmap(df.corr(), annot=True, fmt='.2f')
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1dfa12c3d30>
```



	Timestamp	pCut Motor: Torque	pCut CTRL Position controller: Lag error	pCut CTRL Position controller: Actual position	pCut CTRL Position controller: Actual speed	pSvolFilm CTRL Position controller: Actual position	pSvolFilm CTRL Position controller: Actual speed	pSvolFilm CTRL Position controller: Lag error	Condition
Timestamp	1.00	-0.11	-0.00	0.09	0.14	0.12	0.25	-0.04	-0.01
pCut Motor: Torque	-0.11	1.00	-0.15	-0.03	-0.74	-0.01	0.05	0.25	-0.01
pCut CTRL Position controller: Lag error	-0.00	-0.15	1.00	0.00	-0.01	0.00	0.00	0.09	-0.00
pCut CTRL Position controller: Actual position	0.09	-0.03	0.00	1.00	0.02	0.93	0.06	0.00	-0.81
pCut CTRL Position controller: Actual speed	0.14	-0.74	-0.01	0.02	1.00	0.03	0.27	-0.03	-0.03
pSvolFilm CTRL Position controller: Actual position	0.12	-0.01	0.00	0.93	0.03	1.00	0.11	0.04	-0.81
pSvolFilm CTRL Position controller: Actual speed	0.25	0.05	0.00	0.06	0.27	0.11	1.00	0.50	-0.18
pSvolFilm CTRL Position controller: Lag error	-0.04	0.25	0.09	0.00	-0.03	0.04	0.50	1.00	-0.10
Condition	-0.01	-0.01	-0.00	-0.81	-0.03	-0.81	-0.18	-0.10	1.00

In [11]:
```
df.isnull().values.any()
```

Out[11]:
```
False
```

In [12]:
```
X = df.drop(['Condition'], axis=1) # input
y = df['Condition'] # output (dependent variable)
```

Split and train the models

In [13]:
```
# Split the Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = SEED)
```

Feature Scaling

In [14]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit only to the training data
scaler.fit(X_train)

StandardScaler(copy=True, with_mean=True, with_std=True)

# Now apply the transformations to the data:
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Baseline model using Logistic Regression

In [15]:
```
logreg = LogisticRegression()
```

In [16]:
```
# K-fold cross validation using F1-score as scorer
scores = cross_val_score(logreg,
                         X_train_scaled,
                         y_train,
                         cv=5,
                         scoring='f1_macro')
print('F1 scores:', scores)
print('Mean & standard deviation: {:.2} {:.2f}'.format(scores.mean(), np.std(scores)))
```

```
F1 scores: [1. 1. 1. 1. 1.]
Mean & standard deviation: 1.0 0.00
```

Model Selection - Train and evaluate multiple models

In [17]:
```
# Lists to keep track of model metrics
models = []
precision_list = []
recall_list = []
f1_score_list = []
training_acc_list = []
testing_acc_list = []
time_taken_list = []
params_list = []
```

In [18]:
```
# Logistic Regression
logreg = LogisticRegression(n_jobs=-1)
```

In [19]:
```
%%time
# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
start_time = time.monotonic()

param_grid = {'penalty': ['l1', 'l2', 'elasticnet', 'none'],
              'C': np.logspace(-2, 2, 5)}

gs_logreg = GridSearchCV(logreg,
                      param_grid,
                      cv=5,
                      scoring='f1_macro',
                      n_jobs=-1)
gs_logreg.fit(X_train_scaled, y_train)

end_time = time.monotonic()
time_taken = str(timedelta(seconds=end_time - start_time))
```

```
Wall time: 3.19 s
C:\Users\kerby\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penalty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
```

In [20]:
```
# Best model hyperparameters and score
print(gs_logreg.best_estimator_)
print(gs_logreg.best_params_)
print(gs_logreg.best_score_)
```

```
LogisticRegression(C=0.01, n_jobs=-1, penalty='none')
{'C': 0.01, 'penalty': 'none'}
1.0
```

In [21]:
```
# Using best estimator found by GridSearchCV
logreg = gs_logreg.best_estimator_
logreg.fit(X_train_scaled, y_train)
```

```
C:\Users\kerby\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penalty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
```

Out[21]:
```
LogisticRegression(C=0.01, n_jobs=-1, penalty='none')
```

In [22]:
```
y_pred = logreg.predict(X_test_scaled)
```

In [23]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1569
           1       1.00      1.00      1.00      2015

    accuracy                           1.00      3584
   macro avg       1.00      1.00      1.00      3584
weighted avg       1.00      1.00      1.00      3584
```

In [24]:
```
accuracy_score(y_test, y_pred)
```

Out[24]:
```
1.0
```

In [25]:
```
models.append('Logistic Regression')
precision_list.append(precision_score(y_test, y_pred, average='macro'))
recall_list.append(recall_score(y_test, y_pred, average='macro'))
f1_score_list.append(f1_score(y_test, y_pred, average='macro'))
training_acc_list.append(accuracy_score(y_train, logreg.predict(X_train_scaled)))
testing_acc_list.append(accuracy_score(y_test, y_pred))
time_taken_list.append(time_taken)
params_list.append(gs_logreg.best_params_)
```

Random Forest

In [26]:
```
# Random Forest
rf = RandomForestClassifier(n_jobs=-1)
```

In [27]:
```
%%time
# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
start_time = time.monotonic()

param_grid = {'n_estimators': [100, 200, 300],
              'criterion': ['gini', 'entropy'],
              'max_depth': [5, 10, 25],
              'min_samples_split': [5, 10, 15]}

gs_rf = GridSearchCV(rf,
                    param_grid,
                    cv=5,
                    scoring='f1_macro',
                    n_jobs=-1)
gs_rf.fit(X_train_scaled, y_train)

end_time = time.monotonic()

time_taken = str(timedelta(seconds=end_time - start_time))
```

```
Wall time: 1min 10s
```

In [28]:
```
# Best model hyperparameters and score
print(gs_rf.best_estimator_)
print(gs_rf.best_params_)
print(gs_rf.best_score_)
```

```
RandomForestClassifier(max_depth=5, min_samples_split=5, n_jobs=-1)
{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 100}
0.9997145732586015
```

In [29]:
```
# Using best estimator found by GridSearchCV
rf = gs_rf.best_estimator_
rf.fit(X_train_scaled, y_train)
```

Out[29]:
```
RandomForestClassifier(max_depth=5, min_samples_split=5, n_jobs=-1)
```

In [30]:
```
y_pred = rf.predict(X_test_scaled)
```

In [31]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1569
           1       1.00      1.00      1.00      2015

    accuracy                           1.00      3584
   macro avg       1.00      1.00      1.00      3584
weighted avg       1.00      1.00      1.00      3584
```

In [32]:
```
models.append('Random Forest')
precision_list.append(precision_score(y_test, y_pred, average='macro'))
recall_list.append(recall_score(y_test, y_pred, average='macro'))
f1_score_list.append(f1_score(y_test, y_pred, average='macro'))
training_acc_list.append(accuracy_score(y_train, rf.predict(X_train_scaled)))
testing_acc_list.append(accuracy_score(y_test, y_pred))
time_taken_list.append(time_taken)
params_list.append(gs_rf.best_params_)
```

Multi-level Perceptron

In [33]:
```
# Multi-Level Perceptron
mlp = MLPClassifier(solver='sgd')
```

In [34]:
```
%%time
# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
start_time = time.monotonic()

param_grid = {'hidden_layer_sizes': [(3,3),
                                     (2),
                                     (3)],
              'alpha': np.logspace(-4, -1, 4),
              'max_iter': [400, 450, 500, 550]}

gs_mlp = GridSearchCV(mlp,
                      param_grid,
                      cv=5,
                      scoring='f1_macro',
                      n_jobs=-1)
gs_mlp.fit(X_train_scaled, y_train)

end_time = time.monotonic()

time_taken = str(timedelta(seconds=end_time - start_time))
```

```
Wall time: 3min 45s
```

In [35]:
```
# Best model hyperparameters and score
print(gs_mlp.best_estimator_)
print(gs_mlp.best_params_)
print(gs_mlp.best_score_)
```

```
MLPClassifier(hidden_layer_sizes=(3, 3), max_iter=400, solver='sgd')
{'alpha': 0.0001, 'hidden_layer_sizes': (3, 3), 'max_iter': 400}
1.0
```

In [36]:
```
# Using best estimator found by GridSearchCV
mlp = gs_mlp.best_estimator_
mlp.fit(X_train_scaled, y_train)
```

Out[36]:
```
MLPClassifier(hidden_layer_sizes=(3, 3), max_iter=400, solver='sgd')
```

In [37]:
```
y_pred = mlp.predict(X_test_scaled)
```

In [38]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1569
           1       1.00      1.00      1.00      2015

    accuracy                           1.00      3584
   macro avg       1.00      1.00      1.00      3584
weighted avg       1.00      1.00      1.00      3584
```

In [39]:
```
models.append('Multilevel Perceptron')
precision_list.append(precision_score(y_test, y_pred, average='macro'))
recall_list.append(recall_score(y_test, y_pred, average='macro'))
f1_score_list.append(f1_score(y_test, y_pred, average='macro'))
training_acc_list.append(accuracy_score(y_train, mlp.predict(X_train_scaled)))
testing_acc_list.append(accuracy_score(y_test, y_pred))
time_taken_list.append(time_taken)
params_list.append(gs_mlp.best_params_)
```

In [40]:
```
model_metrics = pd.DataFrame(zip(models, params_list, training_acc_list, testing_acc_list,
                                 precision_list, recall_list, f1_score_list, time_taken_list),
                             columns=['Model', 'Parameters', 'Training Acc', 'Testing Acc', 'Precision', 'Recall', 'F1 Score', 'Time Taken'])
```

In [41]:
```
model_metrics.sort_values(by='F1 Score', ascending=False)
```

Out[41]:

	Model	Parameters	Training Acc	Testing Acc	Precision	Recall	F1 Score	Time Taken
0	Logistic Regression	{'C': 0.01, 'penalty': 'none'}	1.000000	1.000000	1.000000	1.000000	1.000000	0:00:03.188000
2	Multilevel Perceptron	{'alpha': 0.0001, 'hidden_layer_sizes': (3, 3)...	1.000000	1.000000	1.000000	1.000000	1.000000	0:03:45.360000
1	Random Forest	{'criterion': 'gini', 'max_depth': 5, 'min_sam...	0.999907	0.999721	0.999752	0.999681	0.999717	0:01:10.500000

In [49]:
```
model_metrics.iloc[0,1]
```

Out[49]:
```
{'C': 0.01, 'penalty': 'none'}
```

In [43]:
```
cursor.execute('''
               INSERT INTO dbo.Audit_log (model, parameters, train_accuracy, test_accuracy, precision, recall, f1_score, time_taken)
               VALUES
               ('Logistic Regression', 'C: 0.01, penalty: none',1.000000,1.000000,1.000000,1.000000,1.000000, '0:00:03.188000'),
               ('Random Forest', 'criterion: gini, max_depth: 5, min_samples_split: 5, n_estimators: 100',0.999907,0.999721,0.999752,0.999681,0.999717, '0:01:10.500000'),
               ('Multilevel Perceptron', 'alpha: 0.0001, hidden_layer_sizes: (3, 3), max_iter: 400',1.000000,1.000000,1.000000,1.000000,1.000000, '0:03:45.360000')
               ''')
conn.commit()
```

Step 4: Predict and Evaluate with the best model

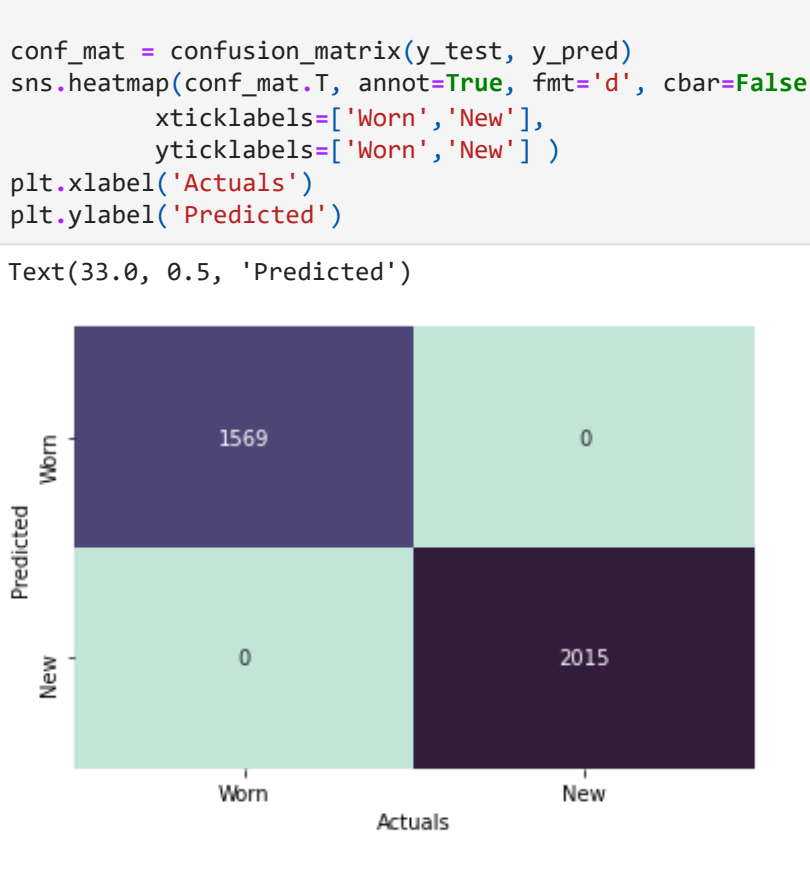
In [44]:
```
# Prediction by selected model
y_pred = logreg.predict(X_test_scaled)
```

In [45]:
```
# Classification Report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1569
           1       1.00      1.00      1.00      2015

    accuracy                           1.00      3584
   macro avg       1.00      1.00      1.00      3584
weighted avg       1.00      1.00      1.00      3584
```

In [46]:
```
## Confusion Matrix

conf_mat = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_mat.T, annot=True, fmt='d', cbar=False, cmap=sns.cubehelix_palette(start=.5, rot=-.5, as_cmap=True),
         xticklabels=['Worn','New'],
         yticklabels=['Worn','New'] )
plt.xlabel('Actuals')
plt.ylabel('Predicted')
```

Out[46]:
```
Text(33.0, 0.5, 'Predicted')
```

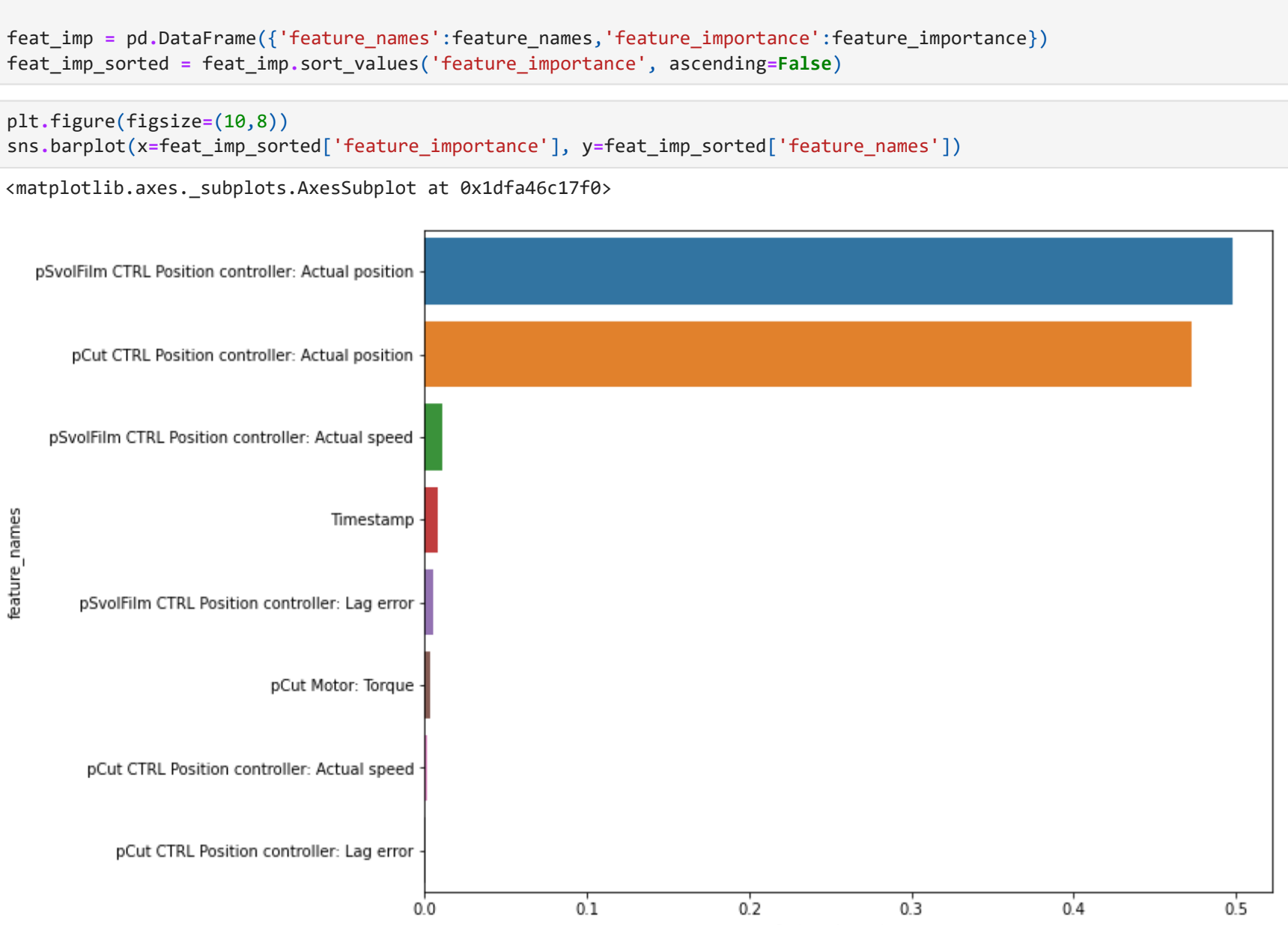


Step 5: Get and plot features importances

In [47]:
```
# Creating the feature importances dataframe
feature_importance = np.array(rf.feature_importances_)
feature_names = np.array(X.columns)

feat_imp = pd.DataFrame({'feature_names':feature_names,'feature_importance':feature_importance})
feat_imp_sorted = feat_imp.sort_values('feature_importance', ascending=False)
```

In [48]:
```
plt.figure(figsize=(10,8))
sns.barplot(x=feat_imp_sorted['feature_importance'], y=feat_imp_sorted['feature_names'])
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1dfa46c17f0>
```



In [ ]:

In [ ]:
```
# # ----- saving best params -----
# new_file = open('logRe_gs_params','wb')
# pickle.dump(logRe_gs.best_params_, new_file)
# new_file.close()

# ----- reading best params from saved file -----
read_file = open('logRe_gs_params', 'rb')
logRe_gs_params = pickle.load(read_file)
read_file.close()
```