In [44]:

```
import numpy as np
import pandas as pd
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
LABELS=["0.4","0.2","0.6"]
```

In [45]:

```
data = pd.read_csv('cnc')
data.head()
```

Out[45]:

	Feed Rate	Rotational Rate	Depth of cut	Surface Roughness
0	0.15	1250	0.6	1.69
1	0.30	1000	0.4	1.47
2	0.15	750	0.2	1.91
3	0.30	1000	0.4	1.47
4	0.15	1000	0.4	1.77

In [46]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Feed Rate          20 non-null     float64
 1   Rotational Rate    20 non-null     int64  
 2   Depth of cut       20 non-null     float64
 3   Surface Roughness  20 non-null     float64
dtypes: float64(3), int64(1)
memory usage: 768.0 bytes
```

In [47]:

```
#Create independent and Dependent Features
columns = data.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Depth of cut"]]
# Store the variable we are predicting
target = "Surface Roughness"
# Define a random state
state = np.random.RandomState(42)
X = data[columns]
Y = data[target]
# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(20, 3)
(20,)
```

In [48]:

```
count_classes = pd.value_counts(data['Depth of cut'], sort = True)

count_classes.plot(kind = 'bar', rot=0)

plt.title("Transaction Class Distribution")

plt.xticks(range(3), LABELS)

plt.xlabel("Depth of cut")

plt.ylabel("Frequency")
```

Out[48]:

```
Text(0, 0.5, 'Frequency')
```

In [49]:

```
DOC1 = data[data['Depth of cut']==0]
DOC2 = data[data['Depth of cut']==1]
DOC3 = data[data['Depth of cut']==2]
```

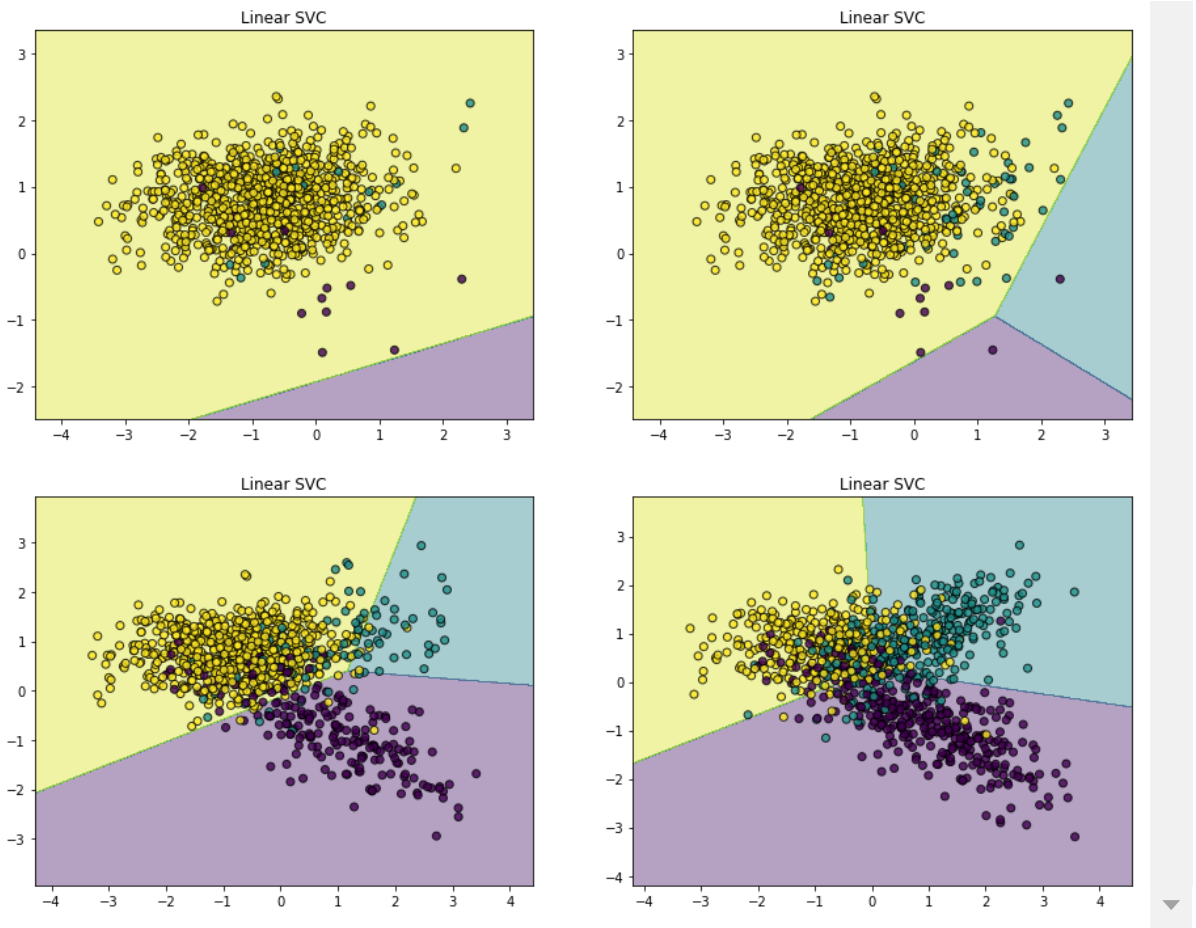
In [50]:

```
print(DOC1.shape,DOC2.shape,DOC3.shape)
```

```
(0, 4) (0, 4) (0, 4)
```

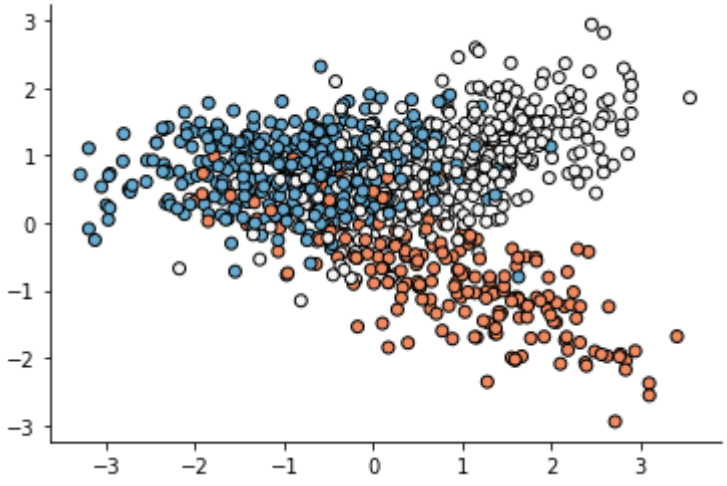
In [51]:

```
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
import numpy as np
from collections import Counter
from sklearn.datasets import make_classification

def create_dataset(n_samples=1000, weights=(0.4, 0.2, 0.6), n_classes=3,
                   class_sep=0.8, n_clusters=1):
    return make_classification(n_samples=n_samples, n_features=2,
                               n_informative=2, n_redundant=0, n_repeated=0,
                               n_classes=n_classes,
                               n_clusters_per_class=n_clusters,
                               weights=list(weights),
                               class_sep=class_sep, random_state=0)

def plot_decision_function(X, y, clf, ax):
    plot_step = 0.01
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=0.4)
    ax.scatter(X[:, 0], X[:, 1], alpha=0.8, c=y, edgecolor='k')

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12))

ax_arr = (ax1, ax2, ax3, ax4)
weights_arr = ((0.01, 0.01, 0.98), (0.01, 0.05, 0.94),
               (0.2, 0.1, 0.7), (0.33, 0.33, 0.33))
for ax, weights in zip(ax_arr, weights_arr):
    X, y = create_dataset(n_samples=1000, weights=weights)
    clf = LinearSVC().fit(X, y)
    plot_decision_function(X, y, clf, ax)
    ax.set_title('Linear SVC')
```



In [52]:

```
import seaborn as sns
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_classes=3,
                           n_clusters_per_class=1,
                           weights=[0.2, 0.4, 0.8],
                           class_sep=0.8, random_state=0)

import matplotlib.pyplot as plt
colors = ['#ef8a62' if v == 0 else '#f7f7f7' if v == 1 else '#67a9cf' for v in y]
kwarg_params = {'linewidth': 1, 'edgecolor': 'black'}
fig = plt.Figure(figsize=(12,6))
plt.scatter(X[:, 0], X[:, 1], c=colors, **kwarg_params)
sns.despine()
```

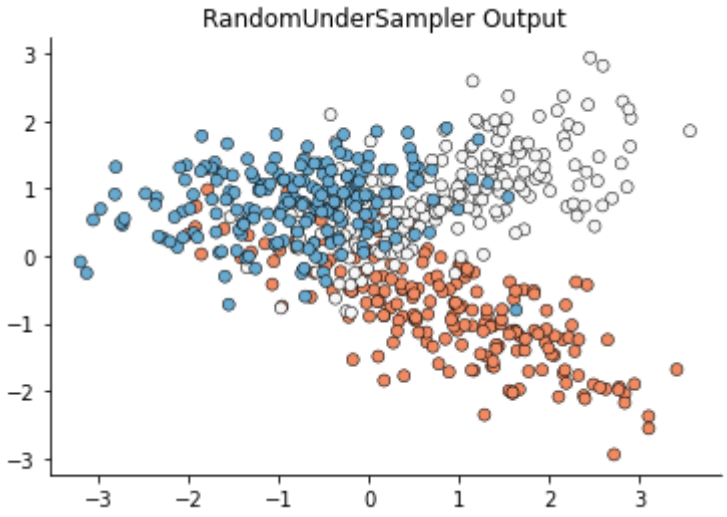


In [53]:

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
```

In [54]:

```
ros = RandomUnderSampler(random_state=0)
ros.fit(X, y)
X_resampled, y_resampled = ros.fit_resample(X, y)
colors = ['#ef8a62' if v == 0 else '#f7f7f7' if v == 1 else '#67a9cf' for v in y_resampled]
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=colors, linewidth=0.5, edgecolor='black
sns.despine()
plt.title("RandomUnderSampler Output")
pass
```



In [3]:

```
from sklearn import datasets, linear_model, metrics

# load the digit dataset
digits = datasets.load_digits()

# defining feature matrix(X) and response vector(y)
X = digits.data
y = digits.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=1)

# create logistic regression object
reg = linear_model.LogisticRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# making predictions on the testing set
y_pred = reg.predict(X_test)

# comparing actual response values (y_test) with predicted response values (y_pred)
print("Logistic Regression model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
```

```
Logistic Regression model accuracy(in %): 96.88888888888889

c:\users\1rn19\appdata\local\programs\python\python38\lib\site-packages\skle
arn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to conve
rge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

In [56]:

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[56]:

```
array([[52,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0, 41,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0,  0, 41,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 49,  0,  0,  0,  1,  2,  0],
       [ 0,  0,  0,  0, 47,  0,  0,  0,  0,  0],
       [ 0,  1,  0,  1,  0, 36,  0,  0,  0,  1],
       [ 0,  0,  0,  0,  0,  0, 43,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 46,  0,  1],
       [ 0,  0,  0,  0,  0,  2,  0,  0, 35,  0],
       [ 0,  0,  0,  0,  0,  1,  0,  0,  1, 46]], dtype=int64)
```

In [52]:

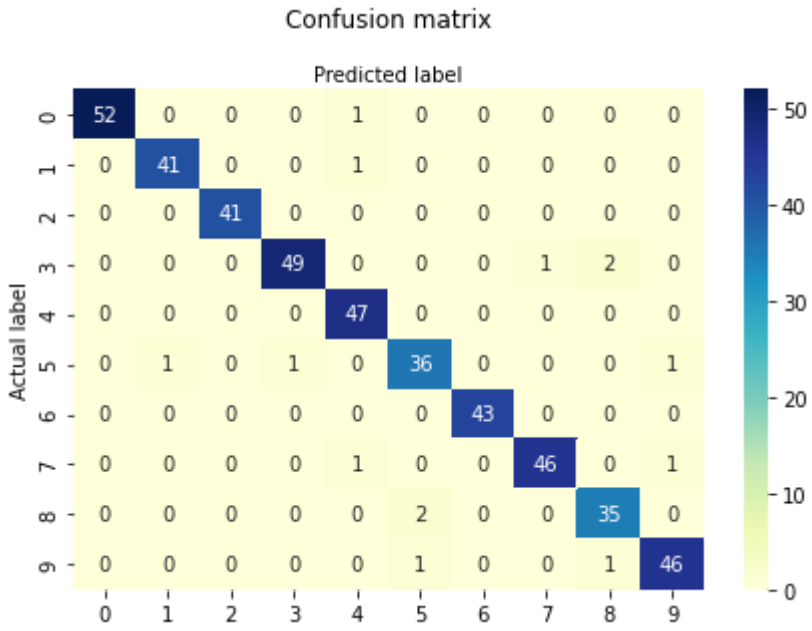
```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [59]:

```
 class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[59]:

```
Text(0.5, 257.44, 'Predicted label')
```



In [55]:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9688888888888889
```

In [66]:

```
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

```
0.3888888888888889
```

In [20]:

```
data.shape
```

Out[20]:

```
(20, 4)
```

In [21]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Feed Rate          20 non-null     float64
 1   Rotational Rate    20 non-null     int64  
 2   Depth of cut       20 non-null     float64
 3   Surface Roughness  20 non-null     float64
dtypes: float64(3), int64(1)
memory usage: 768.0 bytes
```

In [22]:

```
data.describe()
```

Out[22]:

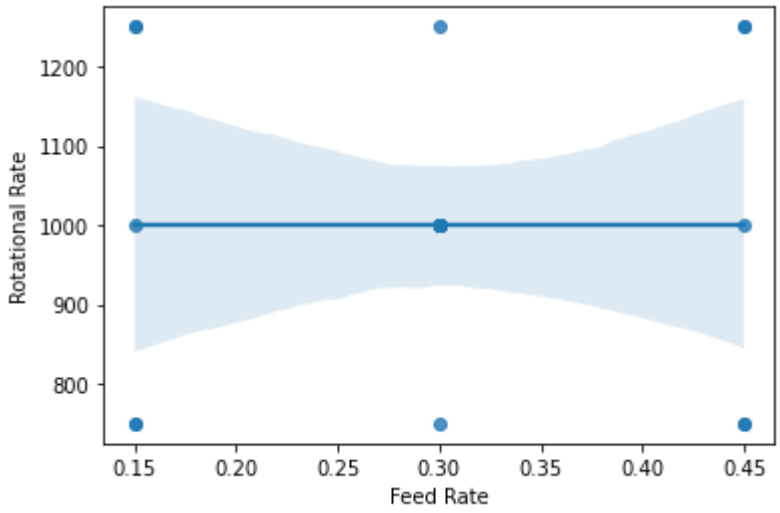
	Feed Rate	Rotational Rate	Depth of cut	Surface Roughness
count	20.000000	20.000000	20.000000	20.000000
mean	0.300000	1000.000000	0.400000	1.572500
std	0.108821	181.369063	0.145095	0.160292
min	0.150000	750.000000	0.200000	1.320000
25%	0.262500	937.500000	0.350000	1.470000
50%	0.300000	1000.000000	0.400000	1.540000
75%	0.337500	1062.500000	0.450000	1.667500
max	0.450000	1250.000000	0.600000	1.910000

In [26]:

```
sns.regplot(x='Feed Rate',y='Rotational Rate',data=data)
```

Out[26]:

```
<AxesSubplot:xlabel='Feed Rate', ylabel='Rotational Rate'>
```

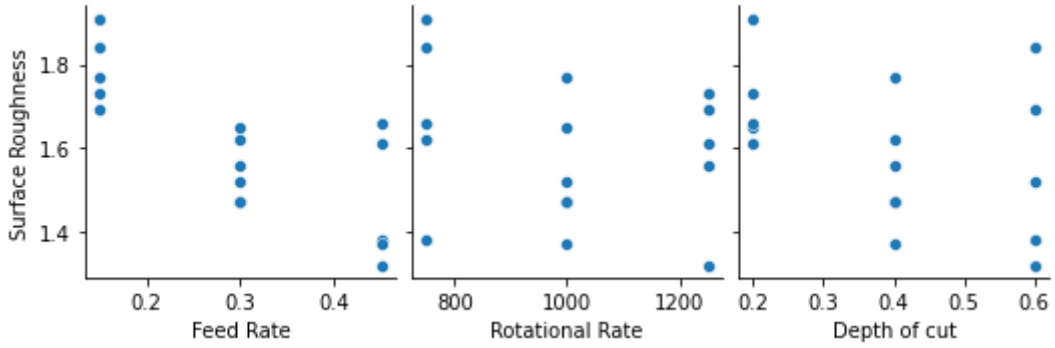


In [28]:

```
sns.pairplot(data=data,
             x_vars=['Feed Rate','Rotational Rate','Depth of cut'],
             y_vars='Surface Roughness')
```

Out[28]:

```
<seaborn.axisgrid.PairGrid at 0x243f58df4f0>
```



In [29]:

```
data.corr()
```

Out[29]:

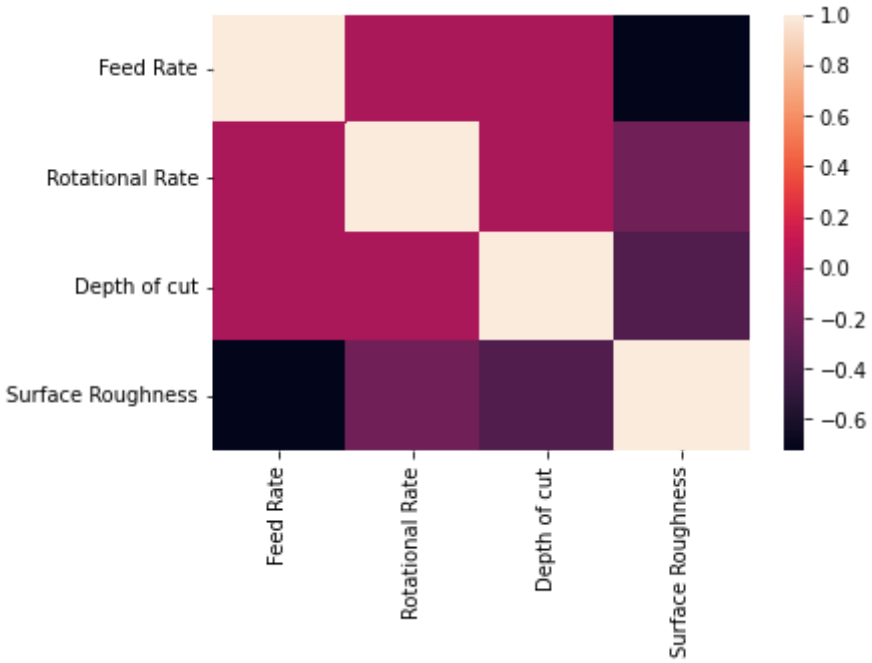
	Feed Rate	Rotational Rate	Depth of cut	Surface Roughness
Feed Rate	1.000000e+00	1.894781e-17	1.232770e-16	-0.724156
Rotational Rate	1.894781e-17	1.000000e+00	-4.263256e-17	-0.226299
Depth of cut	1.232770e-16	-4.263256e-17	1.000000e+00	-0.366604
Surface Roughness	-7.241561e-01	-2.262988e-01	-3.666040e-01	1.000000

In [30]:

```
sns.heatmap(data.corr())
```

Out[30]:

<AxesSubplot:>

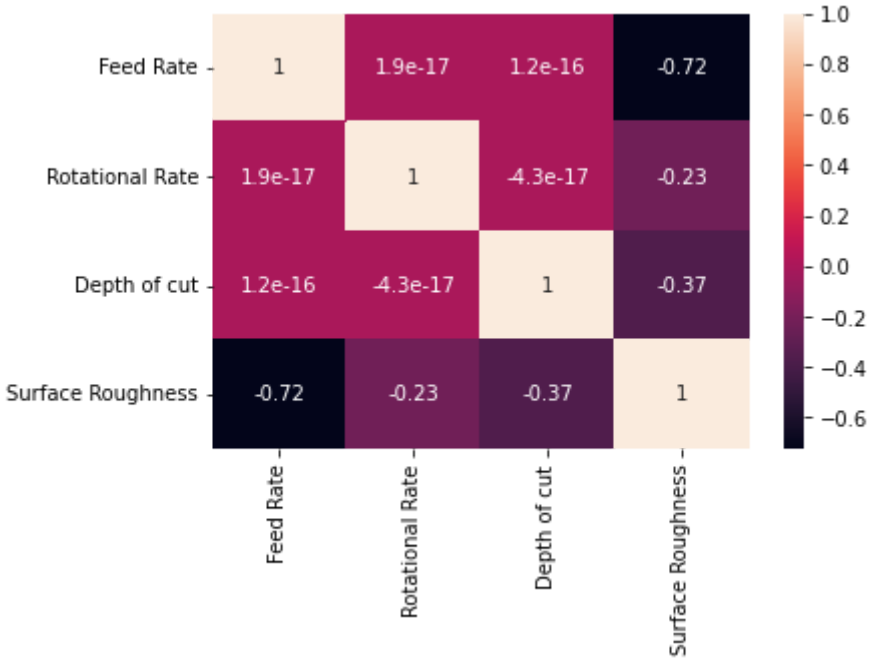


In [31]:

```
sns.heatmap(data.corr(),annot=True)
```

Out[31]:

<AxesSubplot:>



In [34]:

```
X = data['Feed Rate']
y = data['Rotational Rate']
```

In [37]:

```
# train-test split
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.70,test_size=0.30,rand
#X_train, X_test, y_train, y_test
X_train.shape
X_test.shape
```

Out[37]:

```
(6,)
```

In [38]:

```
X_train
```

Out[38]:

```
16    0.45
1     0.30
9     0.30
14    0.45
12    0.30
5     0.15
2     0.15
4     0.15
10    0.30
0     0.15
15    0.30
7     0.45
3     0.30
8     0.15
Name: Feed Rate, dtype: float64
```

In [40]:

```
# training the model
import statsmodels.api as sm
X_train_sm = sm.add_constant(X_train)
X_train_sm.head()
```

Out[40]:

	const	Feed Rate
16	1.0	0.45
1	1.0	0.30
9	1.0	0.30
14	1.0	0.45
12	1.0	0.30

In [41]:

```
# fitting the model
# ols-ordinary least squares
lr = sm.OLS(y_train, X_train_sm)#creates a linear regression object
lr_model = lr.fit()
lr_model.params
```

Out[41]:

```
const        1050.925926
Feed Rate    -246.913580
dtype: float64
```

In [42]:

```
lr_model.summary()
```

```
c:\users\1rn19\appdata\local\programs\python\python38\lib\site-packages\scip
y\stats\stats.py:1541: UserWarning: kurtosistest only valid for n>=20 ... co
ntinuing anyway, n=14
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[42]:

OLS Regression Results

Dep. Variable:	Rotational Rate	**R-squared:**	0.019
Model:	OLS	**Adj. R-squared:**	-0.063
Method:	Least Squares	**F-statistic:**	0.2320
Date:	Thu, 06 Jan 2022	**Prob (F-statistic):**	0.639
Time:	18:54:27	**Log-Likelihood:**	-93.883
No. Observations:	14	**AIC:**	191.8
Df Residuals:	12	**BIC:**	193.0
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	1050.9259	153.803	6.833	0.000	715.818	1386.034
Feed Rate	-246.9136	512.677	-0.482	0.639	-1363.941	870.114

Omnibus:	2.571	**Durbin-Watson:**	1.192
Prob(Omnibus):	0.277	**Jarque-Bera (JB):**	1.057
Skew:	0.134	**Prob(JB):**	0.590
Kurtosis:	1.681	**Cond. No.**	9.69

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [ ]:

```
```