UPF Regressor and Classifier Designing using Random Forest

Importing Libraries

In [387…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
import pydot
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
```

Data Import and Summary

In [388…

```
df=pd.read_csv('D:/UPF Internship/UPF_train.csv')
df.head()
```

Out[388…

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
0	2	20	20	58	57	75	26
1	3	20	30	60	58	50	17
2	4	20	20	81	79	50	176
3	5	20	20	84	77	74	202
4	6	20	20	80	70	50	152

In [389…

```
df.describe()
```

Out[389…

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
count	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000
mean	21.805556	29.444444	31.388889	71.444444	68.861111	49.944444	37.277778
std	12.160403	7.908203	7.983117	9.705505	8.734669	35.736292	51.766294
min	2.000000	20.000000	20.000000	56.000000	57.000000	0.000000	4.000000
25%	10.750000	20.000000	27.500000	60.750000	60.000000	24.750000	8.750000
50%	22.500000	30.000000	30.000000	71.000000	69.000000	50.000000	16.000000
75%	32.250000	40.000000	40.000000	80.250000	77.250000	75.000000	29.250000
max	42.000000	40.000000	40.000000	88.000000	84.000000	100.000000	202.000000

In [390…

```
df.shape
```

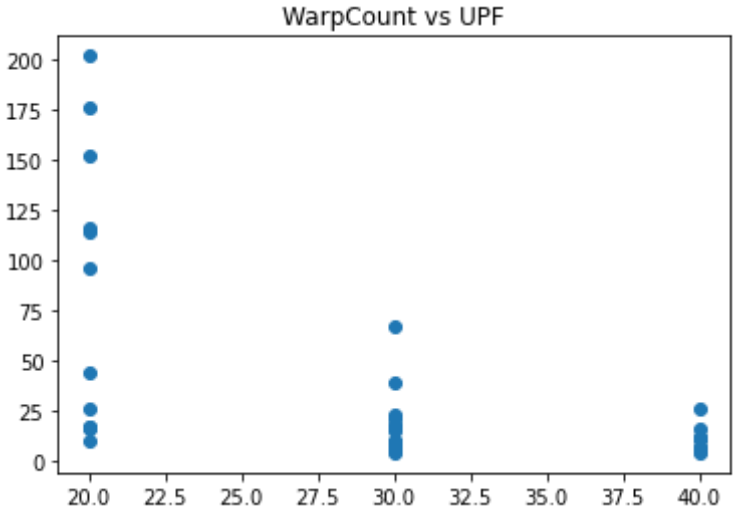
Out[390… (36, 7)

Data Visualisation

In [391…

```
x=df['Warpcount(Ne)']
y=df['UPF']
plt.scatter(x,y)
plt.title('WarpCount vs UPF')
```

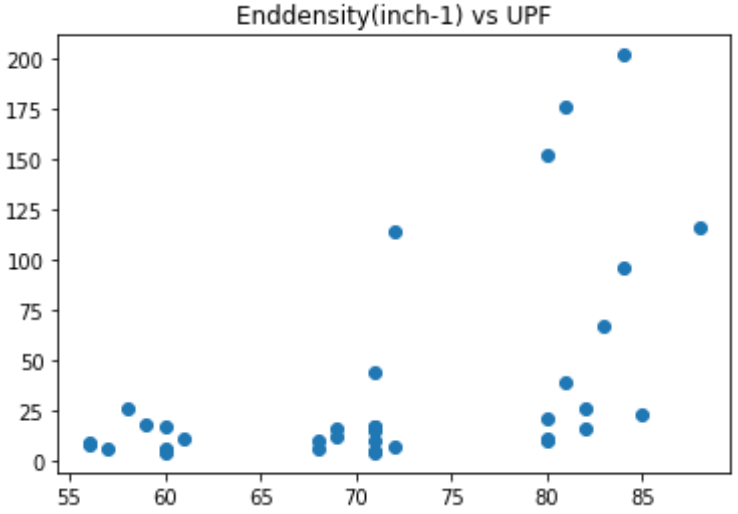
Out[391… Text(0.5, 1.0, 'WarpCount vs UPF')



In [393…

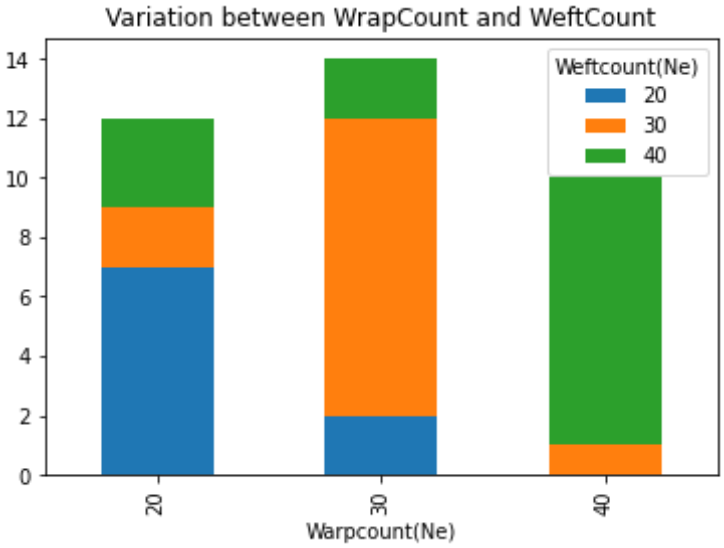
```
x=df['Enddensity(inch-1)']
y=df['UPF']
plt.scatter(x,y)
plt.title('Enddensity(inch-1) vs UPF')
```

Out[393… Text(0.5, 1.0, 'Enddensity(inch-1) vs UPF')



In [395…

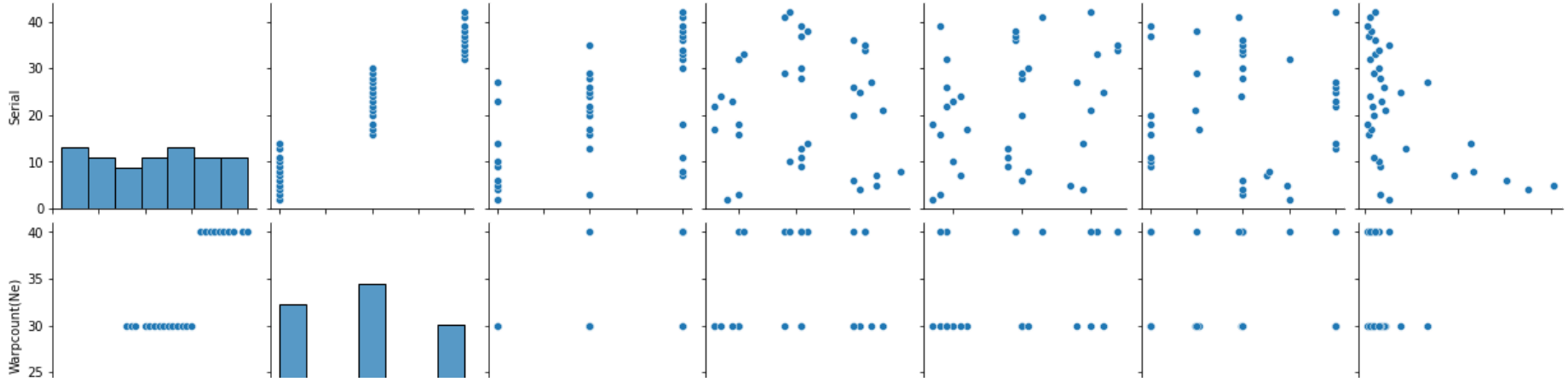
```
df.groupby(['Warpcount(Ne)','Weftcount(Ne)']).size().unstack().plot(kind='bar',stacked=True)
plt.title('Variation between WrapCount and WeftCount')
plt.show()
```

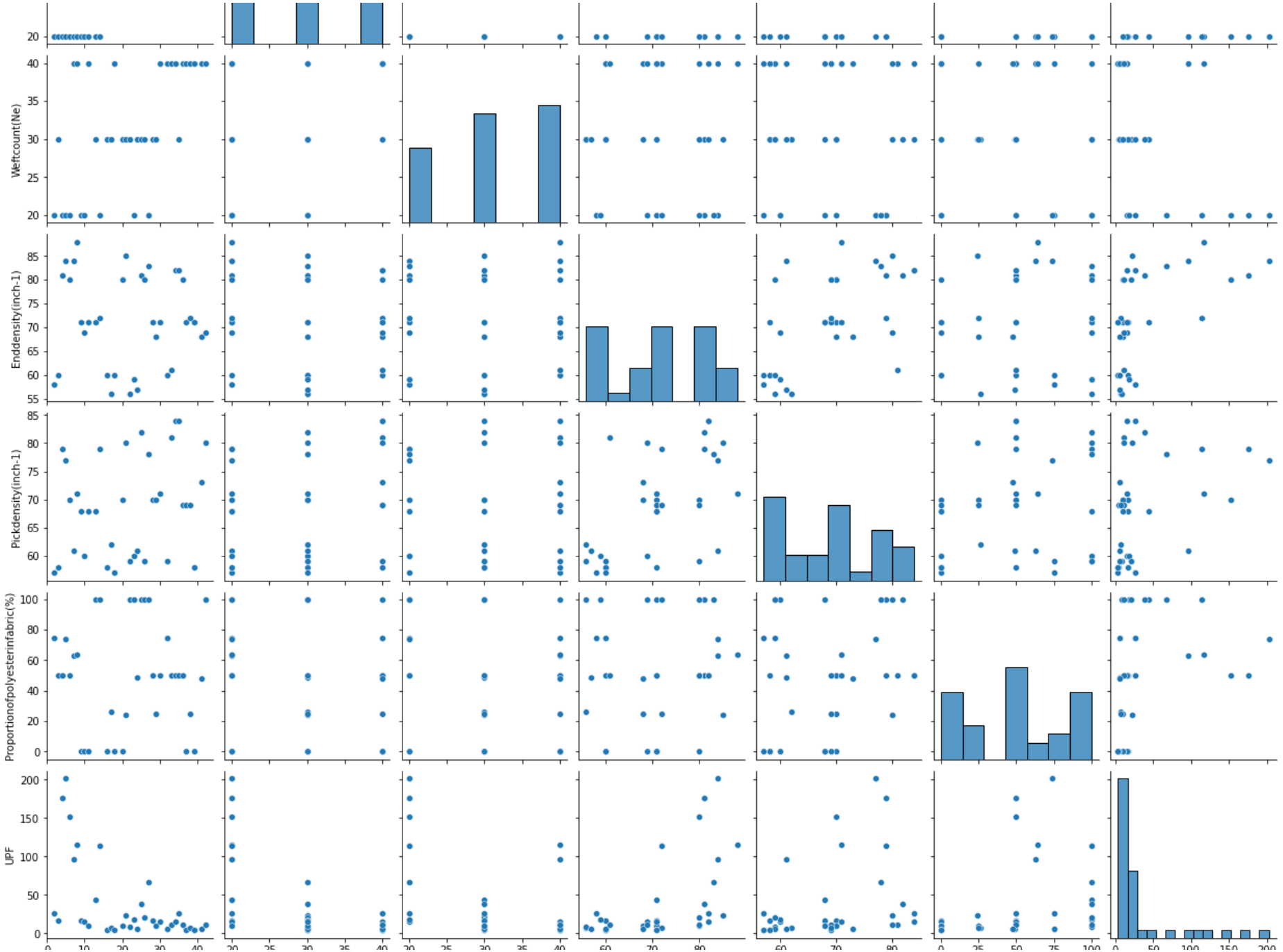


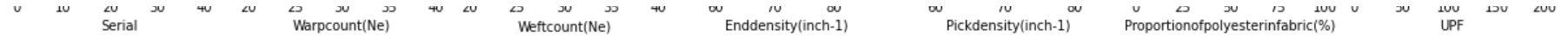
In [206…

```
sns.pairplot(df)
```

Out[206… <seaborn.axisgrid.PairGrid at 0x1f3bf323748>







In [217…

```
l=df['Warpcount(Ne)'].value_counts()
l
```

Out[217…

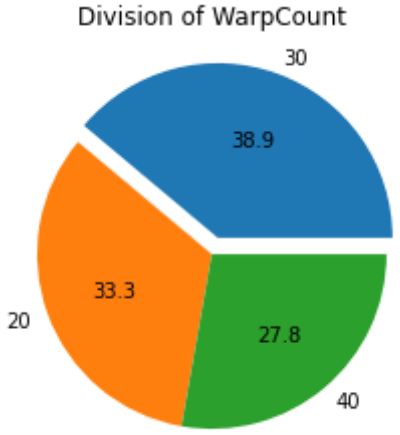
```
30    14
20    12
40    10
Name: Warpcount(Ne), dtype: int64
```

In [396…

```
explode = np.zeros(len(l))
explode[l.argmax()] = 0.1
plt.pie(l,autopct='%0.1f',explode=explode,labels=['30','20','40'])
plt.title('Division of WarpCount')
```

Out[396…

```
Text(0.5, 1.0, 'Division of WarpCount')
```



In [221…

```
l2=df['Weftcount(Ne)'].value_counts()
l2
```

Out[221…

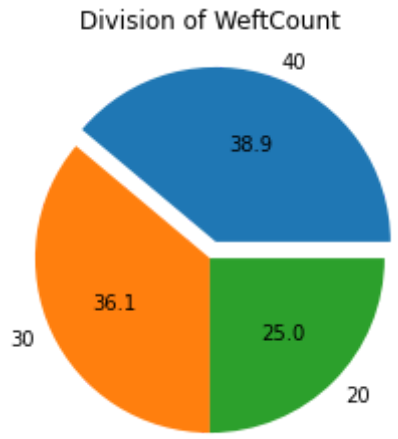
```
40    14
30    13
```

```
20     9
Name: Weftcount(Ne), dtype: int64
```

In [397…

```
explode = np.zeros(len(l2))
explode[l2.argmax()] = 0.1
plt.pie(l2,autopct='%0.1f',explode=explode,labels=['40','30','20'])
plt.title('Division of WeftCount')
```

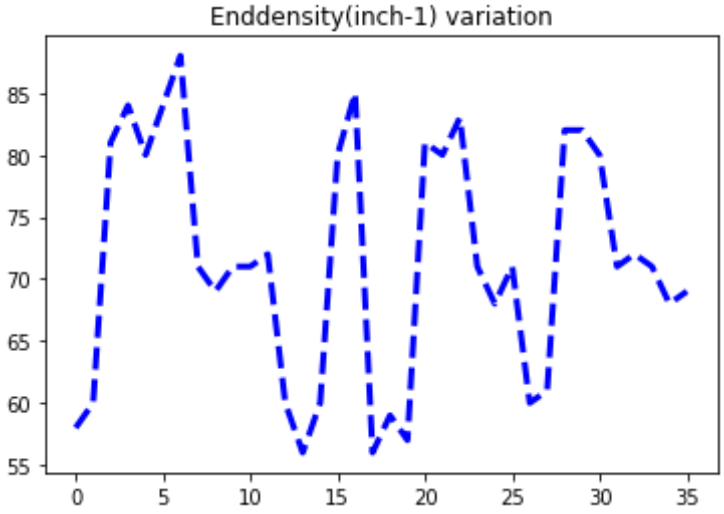
Out[397… Text(0.5, 1.0, 'Division of WeftCount')



In [398…

```
plt.plot(df['Enddensity(inch-1)'], color = 'blue', linewidth=3, linestyle='dashed')
plt.title('Enddensity(inch-1) variation')
```

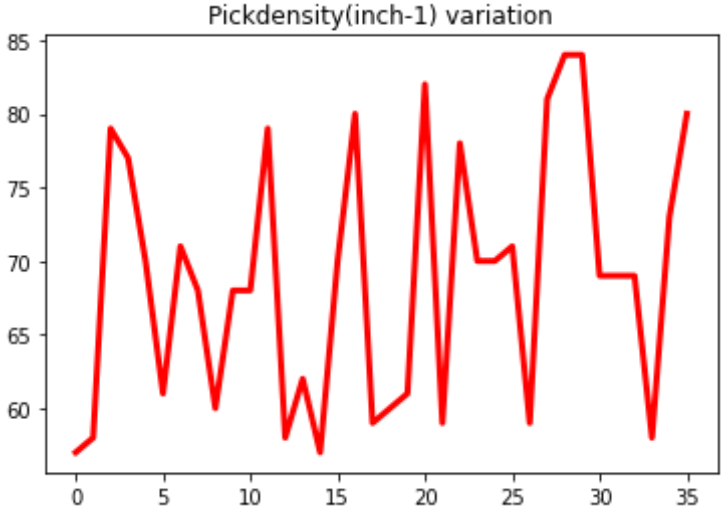
Out[398… Text(0.5, 1.0, 'Enddensity(inch-1) variation')



In [399…

```
plt.plot(df['Pickdensity(inch-1)'], color = 'red', linewidth=3)
plt.title('Pickdensity(inch-1) variation')
```

Out[399… `Text(0.5, 1.0, 'Pickdensity(inch-1) variation')`



In [400…

```
plt.plot(df['Proportionofpolyesterinfabric(%)'], color = 'green', linewidth=3,linestyle='dotted')
```

```
plt.title('Proportionofpolyesterinfabric(%) variation')
```

Out[400… Text(0.5, 1.0, 'Proportionofpolyesterinfabric(%) variation')

Test Set

In [383…

```
df1=pd.read_csv('D:/UPF Internship/UPF_test.csv')
df1.head()
```

Out[383…

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
0	1	20	20	58	59	50	23
1	12	20	20	70	67	100	75
2	15	20	40	72	72	100	33
3	19	30	30	82	82	0	11
4	31	40	40	58	59	50	5

In [384…

```
#Train-Test Dataset
```

```
X=df.iloc[:,1:5]
y=df['UPF']
```

In [385…

```
X_test=df1.iloc[:,1:5]
y_test=df1['UPF']
```

Random Forest Regressor

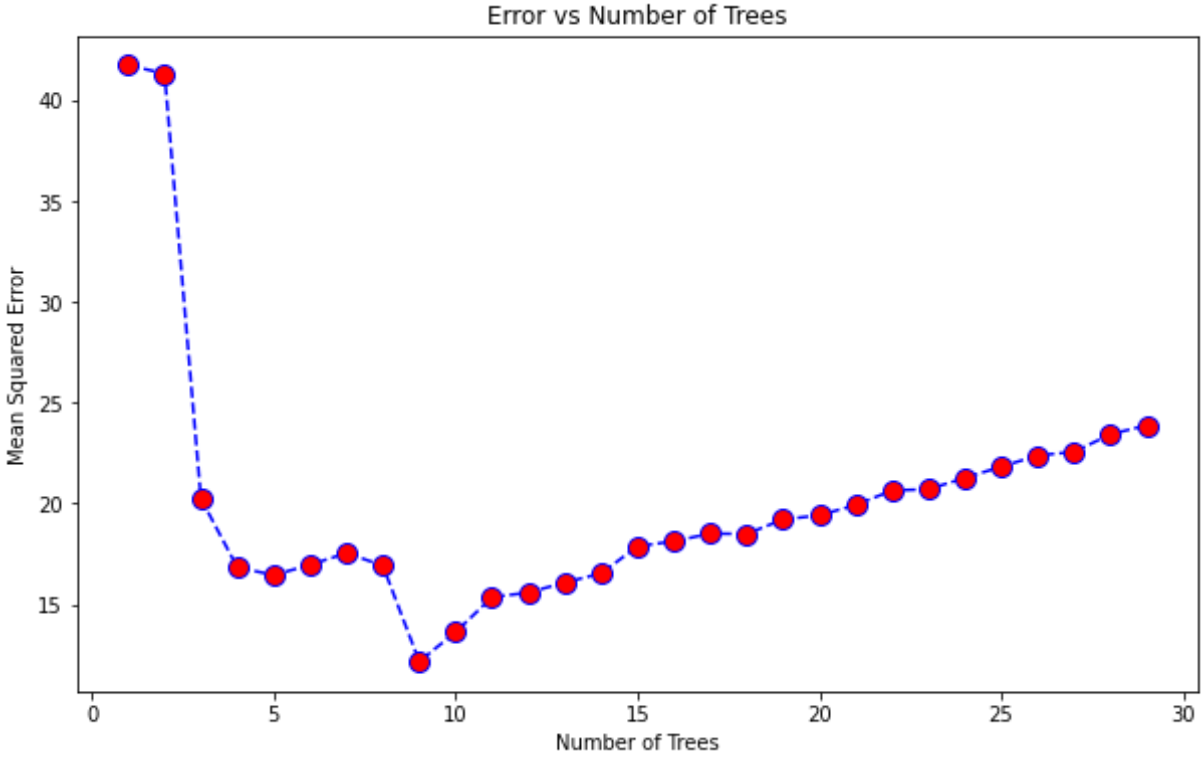
In [238…

```
#Determination of number of trees for which error will be least
error=[]
for i in range(1,30,1):
    RFReg= RandomForestRegressor(n_estimators=i,random_state=0)
    RFReg.fit(X,y)
    y_predict=RFReg.predict(X_test)
    error_exact=np.sqrt(metrics.mean_squared_error(y_test, y_predict))
    print('Error at ',i,'is',error_exact)
    error.append(error_exact)

plt.figure(figsize=(10,6))
plt.plot(range(1,30),error,color='blue', linestyle='dashed',marker='o',markerfacecolor='red', markersize=10)
plt.title('Error vs Number of Trees')
plt.xlabel('Number of Trees')
plt.ylabel('Mean Squared Error')
print("Minimum error:",min(error),"at Number of Trees=",error.index(min(error))+1)
```

```
Error at  1 is 41.73328008516305
Error at  2 is 41.32694843158235
Error at  3 is 20.235648778368375
Error at  4 is 16.82600913268899
Error at  5 is 16.444654653311108
Error at  6 is 16.94708540439045
Error at  7 is 17.538296289397493
Error at  8 is 16.910533576245705
Error at  9 is 12.160703764559956
Error at  10 is 13.613657358207114
Error at  11 is 15.343585625097896
Error at  12 is 15.575941502962309
Error at  13 is 16.070388808292854
Error at  14 is 16.529194950580195
```

```
Error at  15 is 17.867101984746526
Error at  16 is 18.124066067892528
Error at  17 is 18.501815740564723
Error at  18 is 18.484227510682363
Error at  19 is 19.20605571629129
Error at  20 is 19.40525057812962
Error at  21 is 19.92614860772429
Error at  22 is 20.641118795355233
Error at  23 is 20.700857949581973
Error at  24 is 21.266136302599513
Error at  25 is 21.822606016086468
Error at  26 is 22.376248546479072
Error at  27 is 22.55786138348705
Error at  28 is 23.42793172036058
Error at  29 is 23.87584314427824
Minimum error: 12.160703764559956 at Number of Trees= 9
```



In [239…

```
RFReg= RandomForestRegressor(n_estimators=9,random_state=0)
```

```
RFReg
#criterian=squared error
#max_depth=None
#min_samples_split=2
#min_samples_leaf=1
#min_weight_fraction_leaf=0.0
#max_features='auto'
#max_leaf_nodes=None
#random_state=None
#max_samples=None
```

Out[239… RandomForestRegressor(n_estimators=9, random_state=0)

In [240…
```
RFReg.fit(X,y)
```

Out[240… RandomForestRegressor(n_estimators=9, random_state=0)

In [241…
```
y_predict=RFReg.predict(X_test)
```

In [242…
```
r2_score=RFReg.score(X_test,y_test)
r2_score
```

Out[242… 0.7401006747814012

In [243…
```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_predict)))
```

```
Mean Absolute Error: 8.203703703703704
Mean Squared Error: 147.8827160493827
Root Mean Squared Error: 12.160703764559956
```

Visualisation of the Decision Tree

In [244…
```
feature_list=list(X.columns)
tree = RFReg.estimators_[5]
export_graphviz(tree, out_file ='tree.dot', feature_names = feature_list, rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('tree.dot')
graph.write_png('tree.png')
```

Random Forest Classifier

Data Conversion

In [349…
```
df=pd.read_csv('D:/UPF Internship/UPF_train.csv')
```

In [350…
```
def convertUPF(X):
    for i in range(0,36,1):
        if X['UPF'][i]>0 and X['UPF'][i]<=50:
            X['UPF'][i]=1
        elif X['UPF'][i]>50 and X['UPF'][i]<=100:
            X['UPF'][i]=2
        elif X['UPF'][i]>100 and X['UPF'][i]<=200:
            X['UPF'][i]=3
        else:
            X['UPF'][i]=4
    return X
```

In [351…
```
df_classi=convertUPF(df)
df_classi
```

Out[351…

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
0	2	20	20	58	57	75	1
1	3	20	30	60	58	50	1
2	4	20	20	81	79	50	3
3	5	20	20	84	77	74	4

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
4	6	20	20	80	70	50	3
5	7	20	40	84	61	63	2
6	8	20	40	88	71	64	3
7	9	20	20	71	68	0	1
8	10	20	20	69	60	0	1
9	11	20	40	71	68	0	1
10	13	20	30	71	68	100	1
11	14	20	20	72	79	100	3
12	16	30	30	60	58	0	1
13	17	30	30	56	62	26	1
14	18	30	40	60	57	0	1
15	20	30	30	80	70	0	1
16	21	30	30	85	80	24	1
17	22	30	30	56	59	100	1
18	23	30	20	59	60	100	1
19	24	30	30	57	61	49	1
20	25	30	30	81	82	100	1
21	26	30	30	80	59	100	1
22	27	30	20	83	78	100	2
23	28	30	30	71	70	50	1
24	29	30	30	68	70	25	1
25	30	30	40	71	71	50	1
26	32	40	40	60	59	75	1
27	33	40	40	61	81	50	1
28	34	40	40	82	84	50	1

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
29	35	40	30	82	84	50	1
30	36	40	40	80	69	50	1
31	37	40	40	71	69	0	1
32	38	40	40	72	69	25	1
33	39	40	40	71	58	0	1
34	41	40	40	68	73	48	1
35	42	40	40	69	80	100	1

In [352…

```
df1=pd.read_csv('D:/UPF Internship/UPF_test.csv')
df1.head()
```

Out[352…

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
0	1	20	20	58	59	50	23
1	12	20	20	70	67	100	75
2	15	20	40	72	72	100	33
3	19	30	30	82	82	0	11
4	31	40	40	58	59	50	5

In [353…

```
def convertUPF(X):
    for i in range(0,6,1):
        if X['UPF'][i]>0 and X['UPF'][i]<=50:
            X['UPF'][i]=1
        elif X['UPF'][i]>50 and X['UPF'][i]<=100:
            X['UPF'][i]=2
        elif X['UPF'][i]>100 and X['UPF'][i]<=200:
            X['UPF'][i]=3
        else:
            X['UPF'][i]=4
    return X
```

In [354…

```
df_classi_test=convertUPF(df1)
df_classi_test
```

Out[354…

	Serial	Warpcount(Ne)	Weftcount(Ne)	Enddensity(inch-1)	Pickdensity(inch-1)	Proportionofpolyesterinfabric(%)	UPF
0	1	20	20	58	59	50	1
1	12	20	20	70	67	100	2
2	15	20	40	72	72	100	1
3	19	30	30	82	82	0	1
4	31	40	40	58	59	50	1
5	40	40	40	69	68	100	1

In [380…

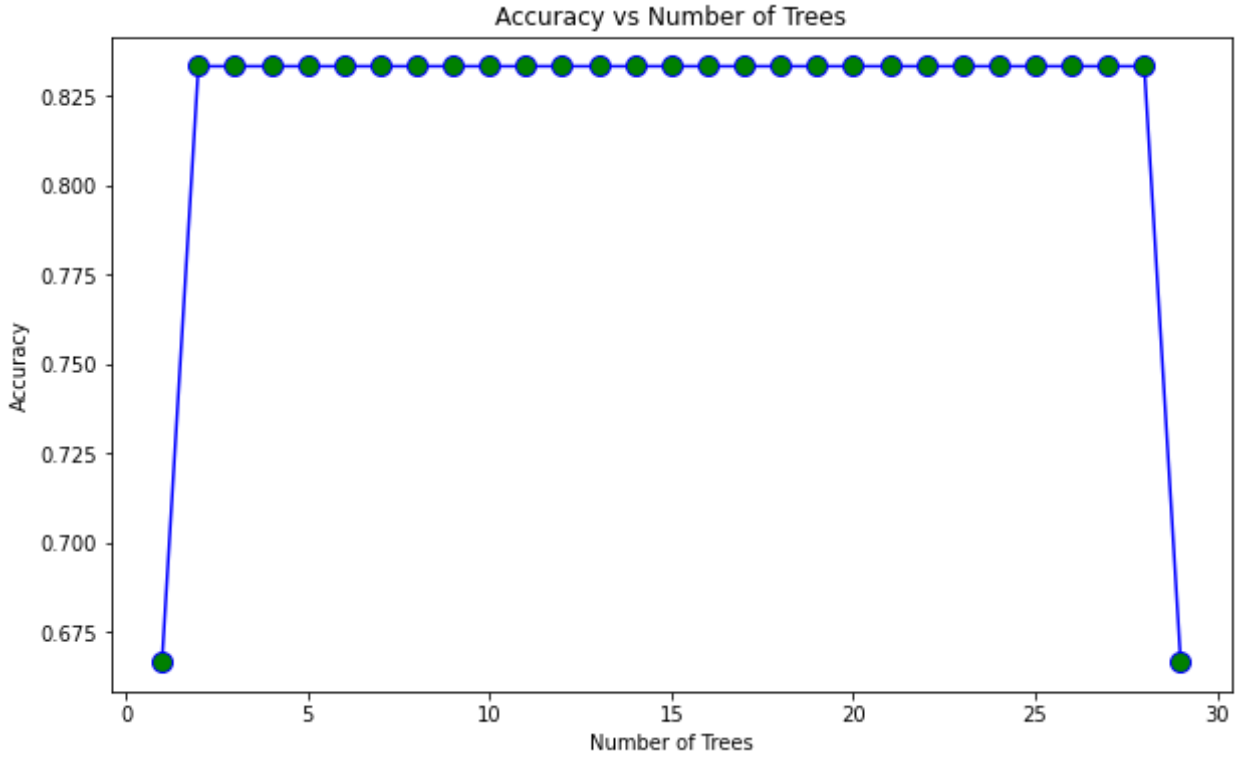
```
#Train-Test Dataset
X=df_classi.iloc[:,1:5]
y=df_classi['UPF']
X_test=df_classi_test.iloc[:,1:5]
y_test=df_classi_test['UPF']
```

In [368…

```
#Determination of Number of Trees for maximum accuracy
acc1=[]
for i in range(1,30,1):
    RFClas= RandomForestClassifier(n_estimators=i,criterion = 'entropy',random_state=0)
    RFClas.fit(X,y)
    y_predict=RFClas.predict(X_test)
    acc=RFClas.score(X_test,y_test)
    print('Accuracy at ',i,'is',acc)
    acc1.append(acc)

plt.figure(figsize=(10,6))
plt.plot(range(1,30),acc1,color='blue', linestyle='solid',marker='o',markerfacecolor='green', markersize=10)
plt.title('Accuracy vs Number of Trees')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
print("Maximum accuracy",min(acc1),"at Number of trees",acc1.index(max(acc1))+1)
```

```
Accuracy at  1 is 0.6666666666666666
Accuracy at  2 is 0.8333333333333334
Accuracy at  3 is 0.8333333333333334
Accuracy at  4 is 0.8333333333333334
Accuracy at  5 is 0.8333333333333334
Accuracy at  6 is 0.8333333333333334
Accuracy at  7 is 0.8333333333333334
Accuracy at  8 is 0.8333333333333334
Accuracy at  9 is 0.8333333333333334
Accuracy at  10 is 0.8333333333333334
Accuracy at  11 is 0.8333333333333334
Accuracy at  12 is 0.8333333333333334
Accuracy at  13 is 0.8333333333333334
Accuracy at  14 is 0.8333333333333334
Accuracy at  15 is 0.8333333333333334
Accuracy at  16 is 0.8333333333333334
Accuracy at  17 is 0.8333333333333334
Accuracy at  18 is 0.8333333333333334
Accuracy at  19 is 0.8333333333333334
Accuracy at  20 is 0.8333333333333334
Accuracy at  21 is 0.8333333333333334
Accuracy at  22 is 0.8333333333333334
Accuracy at  23 is 0.8333333333333334
Accuracy at  24 is 0.8333333333333334
Accuracy at  25 is 0.8333333333333334
Accuracy at  26 is 0.8333333333333334
Accuracy at  27 is 0.8333333333333334
Accuracy at  28 is 0.8333333333333334
Accuracy at  29 is 0.6666666666666666
Maximum accuracy 0.6666666666666666 at Number of trees 2
```



In [373…
```
RFClas= RandomForestClassifier(n_estimators=2,criterion = 'entropy',random_state=0)
RFClas.fit(X,y)
y_predict=RFClas.predict(X_test)
```

In [377…
```
RFClas.score(X_test,y_test)
```

Out[377… 0.8333333333333334

Visualisation of one decision tree from this Random Forest

In [382…
```
feature_list=list(X.columns)
tree = RFClas.estimators_[1]
export_graphviz(tree, out_file ='tree_class.dot', feature_names = feature_list, rounded = True, precision = 1)
```

```
(graph, ) = pydot.graph_from_dot_file('tree_class.dot')
graph.write_png('tree_class.png')
```

Evaluation

In [374…

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           1       0.83      1.00      0.91         5
           2       0.00      0.00      0.00         1

    accuracy                           0.83         6
   macro avg       0.42      0.50      0.45         6
weighted avg       0.69      0.83      0.76         6
```

```
C:\Users\Kathakoli\anaconda3\envs\env_dlib\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetric
Warning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_di
vision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Kathakoli\anaconda3\envs\env_dlib\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetric
Warning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_di
vision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Kathakoli\anaconda3\envs\env_dlib\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetric
Warning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_di
vision` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [375…

```
cf_matrix=confusion_matrix(y_test,y_predict)
cf_matrix
```

Out[375…

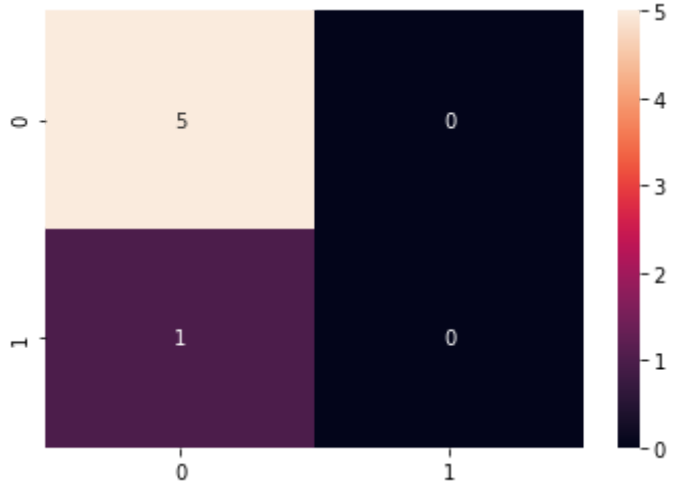
```
array([[5, 0],
       [1, 0]], dtype=int64)
```

In [376…

```
import seaborn as sns
sns.heatmap(cf_matrix, annot=True)
```

Out[376…

```
<AxesSubplot:>
```



Conclusion

Regression Model :

The typical random forest regression model designed here has 9 decision tree since it is found to give the best accuracy. It shows a mean-squared error of only 12.16%. Hence the regressor defined can be considered as a good regressor model giving optimum result.

Classifier Model :

For designing the classifier model, the entire dataset is divided into 4 classes trying to put equal number of data in all. The classifier here gives an accuracy score of 83.33% but it is not a good classifier since there is no false positive and false negative values. This is majorly due to the less number of data that the testing dataset doesnt have any instance of few classes. Also, if more classes are formed

If more classes are formed in the dataset, it will result in unnecessary underfitting of the data and less will result in overfitting with an accuracy of 1 that is not desired. Hence this 4 class classification is chosen.