

SimulatedData-RandomForest

March 16, 2020

```
In [1]: # Notebook for creating & running random forest models on the simulated data

In [2]: from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import RandomForestRegressor
        import seaborn as sn
        import numpy as np
        import pandas as pd

In [3]: #np.random.seed(31415)

In [4]: # GROUP: OCELOT
        # CSM 185

In [5]: sim_staticV1 = pd.read_csv("../Data/Static_Data_ICU_var1_peak.csv")
        sim_staticV2 = pd.read_csv("../Data/Static_Data_ICU_var2_peak.csv")
        sim_staticV3 = pd.read_csv("../Data/Static_Data_ICU_var3.csv")

        # var2 dataset has higher variance in simulated values, so we would expect worse model
        # var3 dataset should have even higher variance

        sim_staticV1.head()
```

Out [5]:

	Unnamed: 0	ICU	Age	Sex	Survival	Probability	Score	Kidney	\
0	1	1	49	F		0.819668	0.932323	0.851415	
1	2	1	85	M		0.493352	0.545629	0.400727	
2	3	1	65	M		0.658073	0.648700	0.612584	
3	4	1	40	F		0.907399	0.859131	0.924469	
4	5	1	40	F		0.850763	0.930617	0.895776	

	Blood	Cholesterol	Immune	...	O2	HR	BP	\
0	0.789044	0.743088	0.848733	...	96.552552	70.815957	68.640993	
1	0.498132	0.448064	0.464661	...	95.167034	99.189290	63.011305	
2	0.635265	0.596647	0.690954	...	96.031453	63.289582	66.286136	
3	0.954992	0.946001	1.000000	...	97.090119	71.956946	92.109964	
4	0.890731	0.876580	0.767765	...	97.769863	90.973826	67.955142	

	RBC	LDL	HDL	Platelet	WBC	INR	\
--	-----	-----	-----	----------	-----	-----	---

0	4.445352	141.438747	80.428266	377446.453569	8689.129619	5.282563
1	4.124222	157.806799	33.992189	0.000000	3484.471712	6.783162
2	4.599221	163.033611	88.730209	384086.644694	5034.262028	7.582851
3	5.577905	117.153138	68.678224	299955.144276	7851.228124	1.729525
4	4.623505	126.284526	80.002718	358008.401336	4736.958890	6.370734

Death	
0	0
1	1
2	0
3	0
4	0

[5 rows x 25 columns]

In [6]: *# Clean data in preparation:*

```
sim_staticV1 = sim_staticV1.drop(columns=["Unnamed: 0", "CHM"])
sim_staticV1 = sim_staticV1.replace(["M", "F"], [0,1])

sim_staticV2 = sim_staticV2.drop(columns=["Unnamed: 0", "CHM"])
sim_staticV2 = sim_staticV2.replace(["M", "F"], [0,1])

sim_staticV3 = sim_staticV3.drop(columns=["Unnamed: 0", "CHM"])
sim_staticV3 = sim_staticV3.replace(["M", "F"], [0,1])

sim_staticV3.tail()
```

Out [6]:

	ICU	Age	Sex	Survival_Probability	Score	Kidney	Blood	\
995	1	48	0	0.784171	0.784171	0.784171	0.784171	
996	1	37	0	0.856422	0.856422	0.856422	0.856422	
997	1	69	0	0.695986	0.695986	0.695986	0.695986	
998	1	84	1	0.565340	0.565340	0.565340	0.565340	
999	1	74	0	0.663907	0.663907	0.663907	0.663907	

	Cholesterol	Immune	ApacheII	...	O2	HR	BP	\
995	0.784171	0.784171	19.969460	...	96.671757	93.312974	68.358783	
996	0.856422	0.856422	17.078377	...	97.153604	91.385585	70.768019	
997	0.695986	0.695986	23.101824	...	96.149696	95.401216	96.550912	
998	0.565340	0.565340	27.612627	...	95.397896	61.591582	61.989478	
999	0.663907	0.663907	24.202210	...	95.966298	63.865193	97.101105	

	RBC	LDL	HDL	Platelet	WBC	INR	\
995	4.517634	139.938921	83.297704	384132.790327	5170.229613	5.692976	
996	5.690815	134.156754	45.768019	365148.008652	5507.522689	5.028027	
997	5.881558	146.203648	40.748480	404701.978904	9810.182420	6.413420	
998	6.024400	155.225254	92.214731	434322.916193	10261.262683	7.450904	
999	4.383597	148.404420	88.235911	65029.504086	4676.408860	6.666508	

	Death
995	0
996	0
997	0
998	1
999	0

[5 rows x 23 columns]

```
In [7]: sim_staticV1.isna().sum() # no null values (because data is simulated)
```

```
Out[7]: ICU          0
Age                0
Sex                0
Survival Probability 0
Score              0
Kidney             0
Blood              0
Cholesterol        0
Immune             0
ApacheII           0
Urine Volume       0
Creatinine         0
Urea               0
O2                 0
HR                 0
BP                 0
RBC                0
LDL                0
HDL                0
Platelet           0
WBC                0
INR                0
Death              0
dtype: int64
```

```
In [8]: # Goal: predict death tag based on data
```

```
x_train1, x_test1, y_train1, y_test1 = train_test_split(sim_staticV1.drop(["ICU", "Death"], axis=1), sim_staticV1["Death"],
```

```
x_train2, x_test2, y_train2, y_test2 = train_test_split(sim_staticV2.drop(["ICU", "Death"], axis=1), sim_staticV2["Death"],
```

```
x_train3, x_test3, y_train3, y_test3 = train_test_split(sim_staticV3.drop(["ICU", "Death"], axis=1), sim_staticV3["Death"],
```

```
x_trainR1, x_testR1, y_trainR1, y_testR1 = train_test_split(sim_staticV1.drop(["ICU", "Death"], axis=1), sim_staticV1["Death"],
```

```
x_trainR2, x_testR2, y_trainR2, y_testR2 = train_test_split(sim_staticV2.drop(["ICU", "Death"], axis=1), sim_staticV2["Death"],
```

```
x_trainR3, x_testR3, y_trainR3, y_testR3 = train_test_split(sim_staticV3.drop(["ICU", "Death"], axis=1), sim_staticV3["Death"],
```

```
# creating train/test cohorts for model creation
```

```
In [9]: len(x_train1) # 750 elements  
model_staticV1 = RandomForestClassifier(n_estimators = 100)  
model_staticV1.fit(x_train1, y_train1)  
  
model_staticV2 = RandomForestClassifier(n_estimators = 100)  
model_staticV2.fit(x_train2, y_train2)  
  
model_staticV3 = RandomForestClassifier(n_estimators=100)  
model_staticV3.fit(x_train3, y_train3)  
  
model_staticRegV1 = RandomForestRegressor(n_estimators=100)  
model_staticRegV1.fit(x_trainR1, y_trainR1)  
  
model_staticRegV2 = RandomForestRegressor(n_estimators=100)  
model_staticRegV2.fit(x_trainR2, y_trainR2)  
  
model_staticRegV3 = RandomForestRegressor(n_estimators=100)  
model_staticRegV3.fit(x_trainR3, y_trainR3)
```

```
# each random forest classifier will consist of 100 individual decision trees
```

```
Out [9]: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',  
                                max_depth=None, max_features='auto', max_leaf_nodes=None,  
                                max_samples=None, min_impurity_decrease=0.0,  
                                min_impurity_split=None, min_samples_leaf=1,  
                                min_samples_split=2, min_weight_fraction_leaf=0.0,  
                                n_estimators=100, n_jobs=None, oob_score=False,  
                                random_state=None, verbose=0, warm_start=False)
```

```
In [10]: model_staticV1.score(x_test1, y_test1) # model 1 score
```

```
Out [10]: 0.6970588235294117
```

```
In [11]: model_staticV2.score(x_test2, y_test2) # model 2 score
```

```
Out [11]: 0.6911764705882353
```

```
In [12]: model_staticV3.score(x_test3, y_test3)
```

```
Out [12]: 0.6294117647058823
```

```
In [13]: y_predicted1 = model_staticV1.predict(x_test1)
```

```
In [14]: y_predicted2 = model_staticV2.predict(x_test2)
```

```

In [15]: y_predicted3 = model_staticV3.predict(x_test3)

In [16]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         from sklearn.metrics import auc

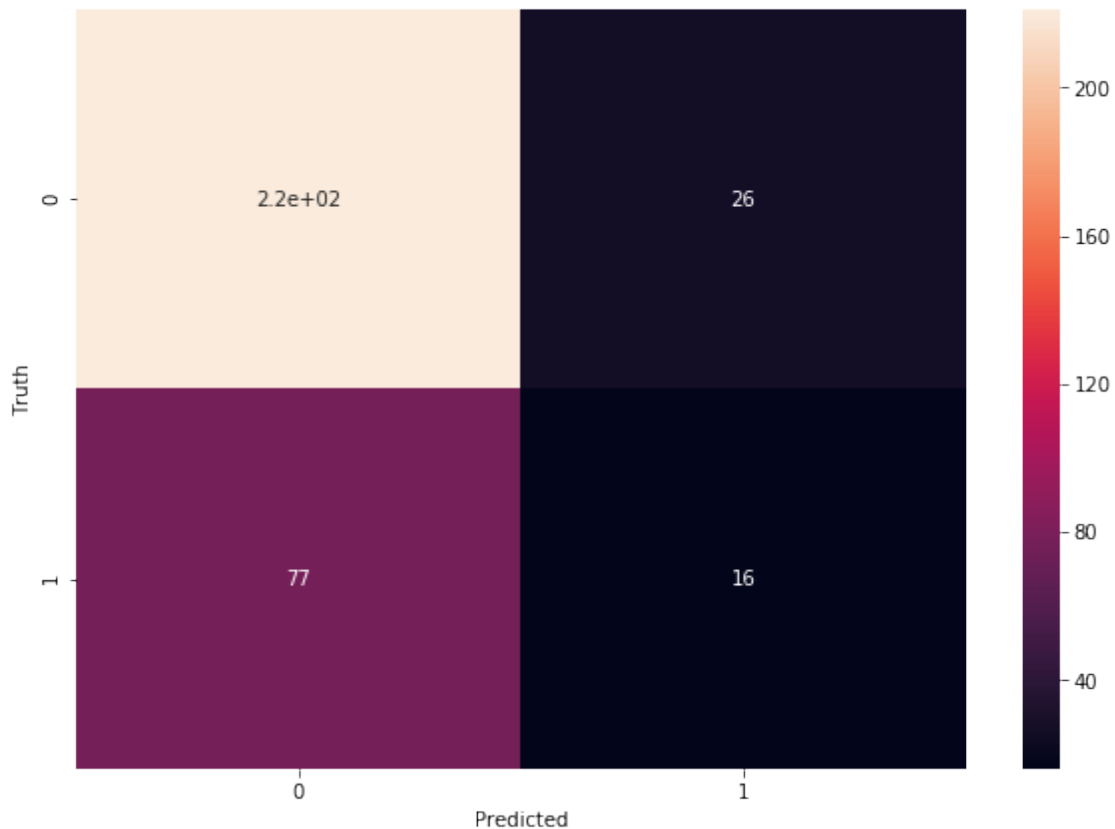
         # for visualizations

In [17]: cm_staticV1 = confusion_matrix(y_test1, y_predicted1)
         cm_staticV2 = confusion_matrix(y_test2, y_predicted2)
         cm_staticV3 = confusion_matrix(y_test3, y_predicted3)

In [18]: %matplotlib inline
         import matplotlib.pyplot as plt
         plt.figure(figsize = (10,7))
         sn.heatmap(cm_staticV1, annot=True)
         plt.xlabel("Predicted")
         plt.ylabel("Truth")
         plt.savefig("V1-Classification-Confusion.png")

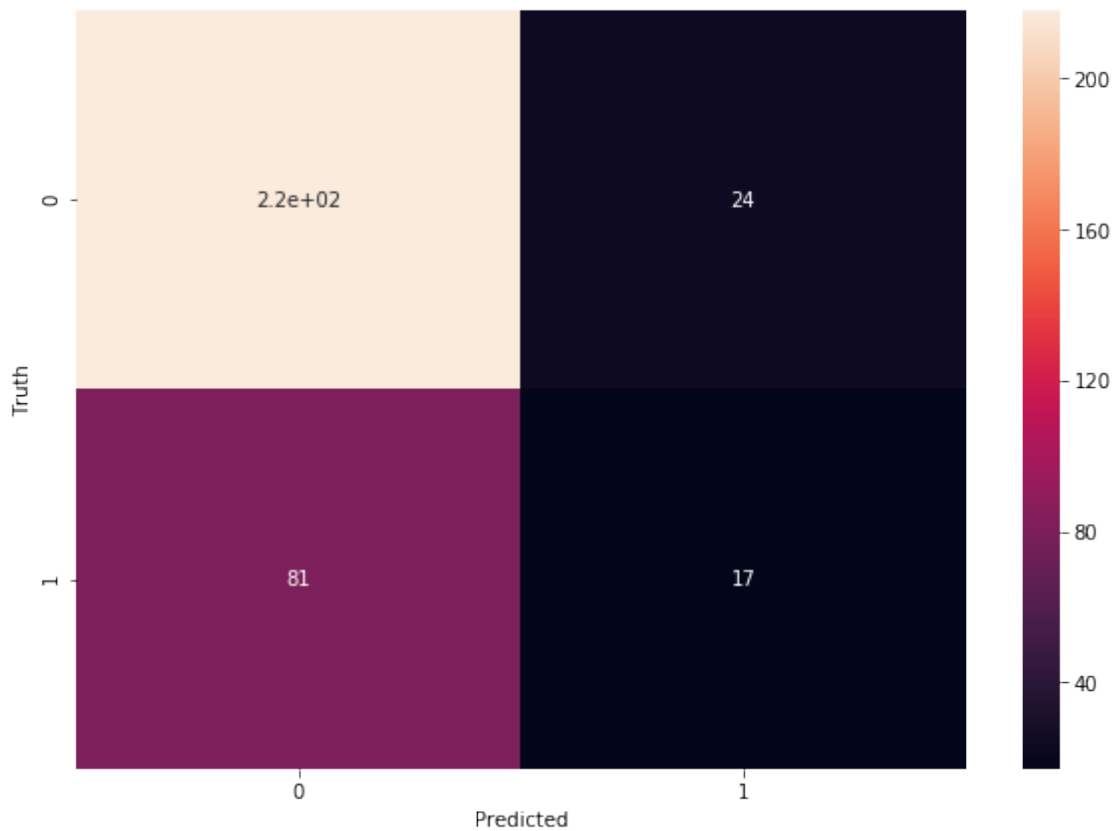
         # confusion matrix for v1

```



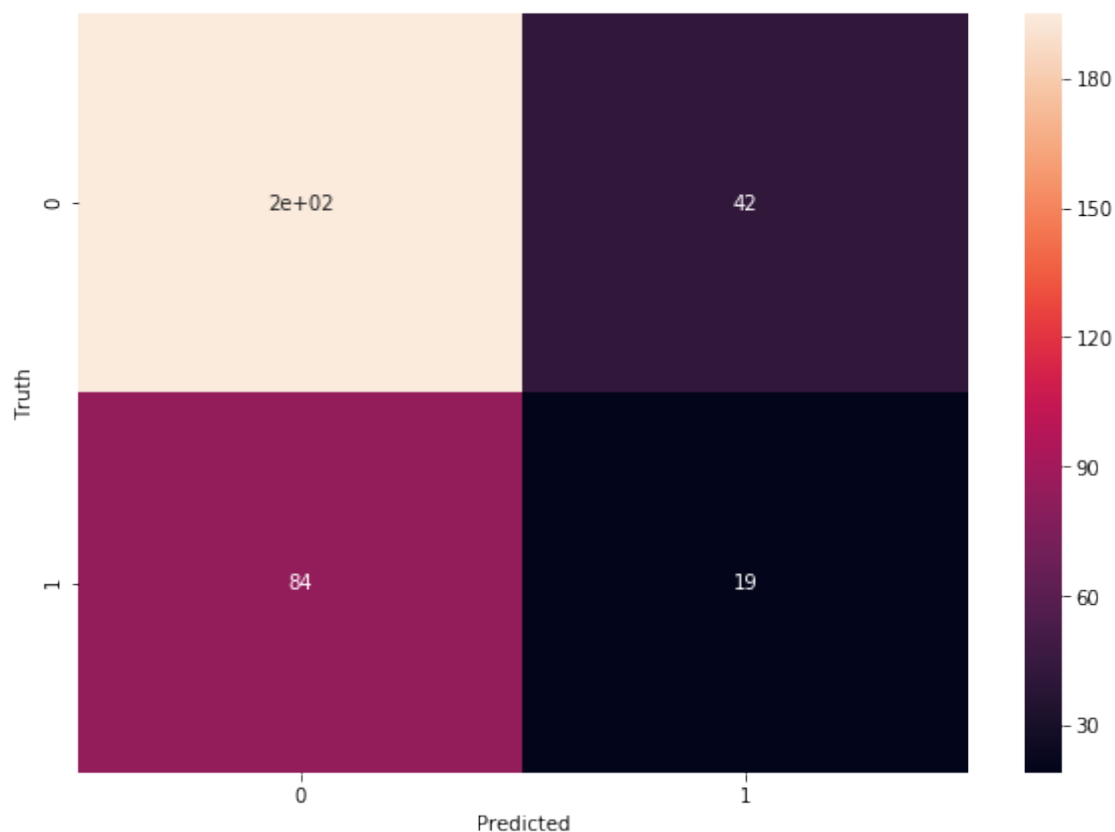
```
In [19]: plt.figure(figsize = (10,7))
         sn.heatmap(cm_staticV2, annot=True)
         plt.xlabel("Predicted")
         plt.ylabel("Truth")
         plt.savefig("V2-Classification-Confusion.png")

         # confusion matrix for v2
```



```
In [20]: plt.figure(figsize = (10,7))
         sn.heatmap(cm_staticV3, annot=True)
         plt.xlabel("Predicted")
         plt.ylabel("Truth")
         plt.savefig("V3-Classification-Confusion.png")

         # confusion matrix for v3
```



```
In [21]: roc_auc_score(np.array(y_test1), np.array(y_predicted1))
```

```
Out[21]: 0.5333899264289756
```

```
In [22]: roc_auc_score(np.array(y_test2), np.array(y_predicted2))
```

```
Out[22]: 0.5371479170180469
```

```
In [23]: roc_auc_score(np.array(y_test3), np.array(y_predicted3))
```

```
Out[23]: 0.5036254147720289
```

```
In [24]: roc_curve(y_test1, y_predicted1)
```

```
Out[24]: (array([0.          , 0.10526316, 1.          ]),
          array([0.          , 0.17204301, 1.          ]),
          array([2, 1, 0]))
```

```
In [25]: roc_curve(y_test2, y_predicted2)
```

```
Out[25]: (array([0.          , 0.09917355, 1.          ]),
          array([0.          , 0.17346939, 1.          ]),
          array([2, 1, 0]))
```

```

In [26]: roc_curve(y_test3, y_predicted3)

Out[26]: (array([0.          , 0.17721519, 1.          ]),
          array([0.          , 0.18446602, 1.          ]),
          array([2, 1, 0]))

In [27]: fpr1, tpr1, threshold1 = roc_curve(y_test1, y_predicted1)
         roc_auc1 = auc(fpr1, tpr1)

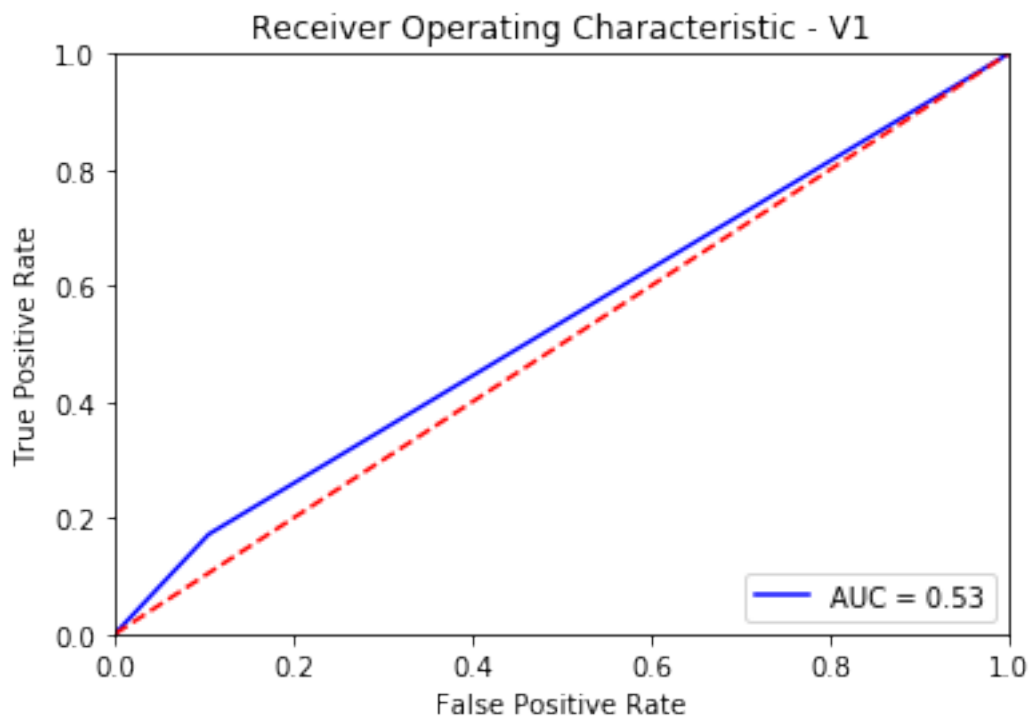
         fpr2, tpr2, threshold2 = roc_curve(y_test2, y_predicted2)
         roc_auc2 = auc(fpr2, tpr2)

         fpr3, tpr3, threshold3 = roc_curve(y_test3, y_predicted3)
         roc_auc3 = auc(fpr3, tpr3)

In [28]: plt.title('Receiver Operating Characteristic - V1')
         plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
         plt.legend(loc = 'lower right')
         plt.plot([0, 1], [0, 1], 'r--')
         plt.xlim([0, 1])
         plt.ylim([0, 1])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.savefig("V1-Classification-ROC.png")
         plt.show()

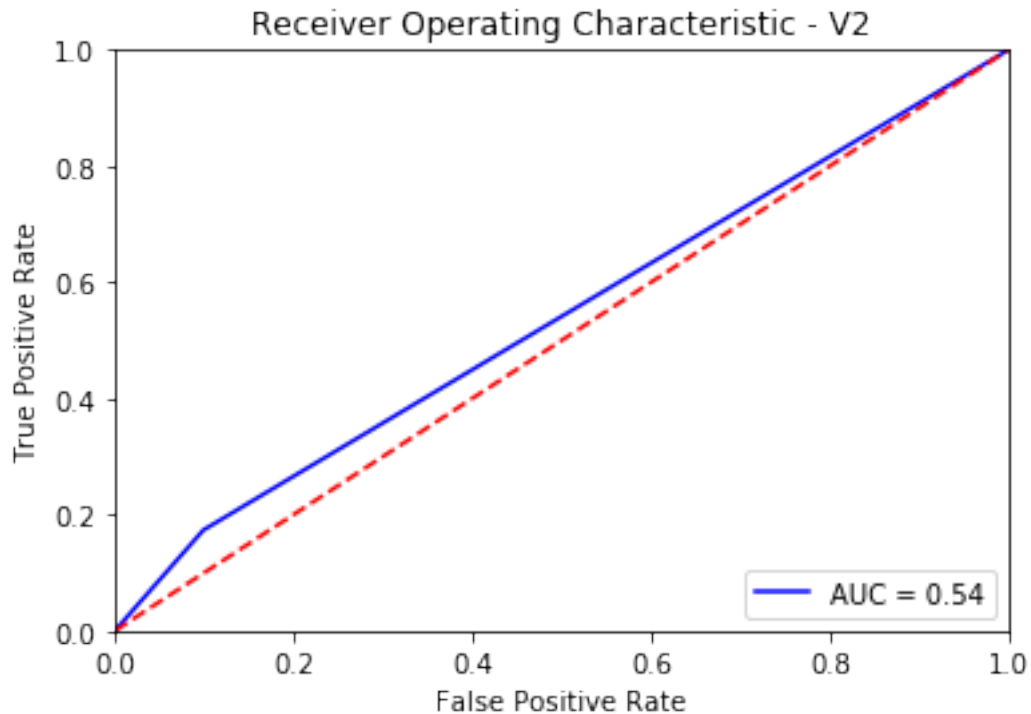
```

ROC Curve for v1




```
In [29]: plt.title('Receiver Operating Characteristic - V2')
plt.plot(fpr2, tpr2, 'b', label = 'AUC = %0.2f' % roc_auc2)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.savefig("V2-Classification-ROC.png")
plt.show()
```

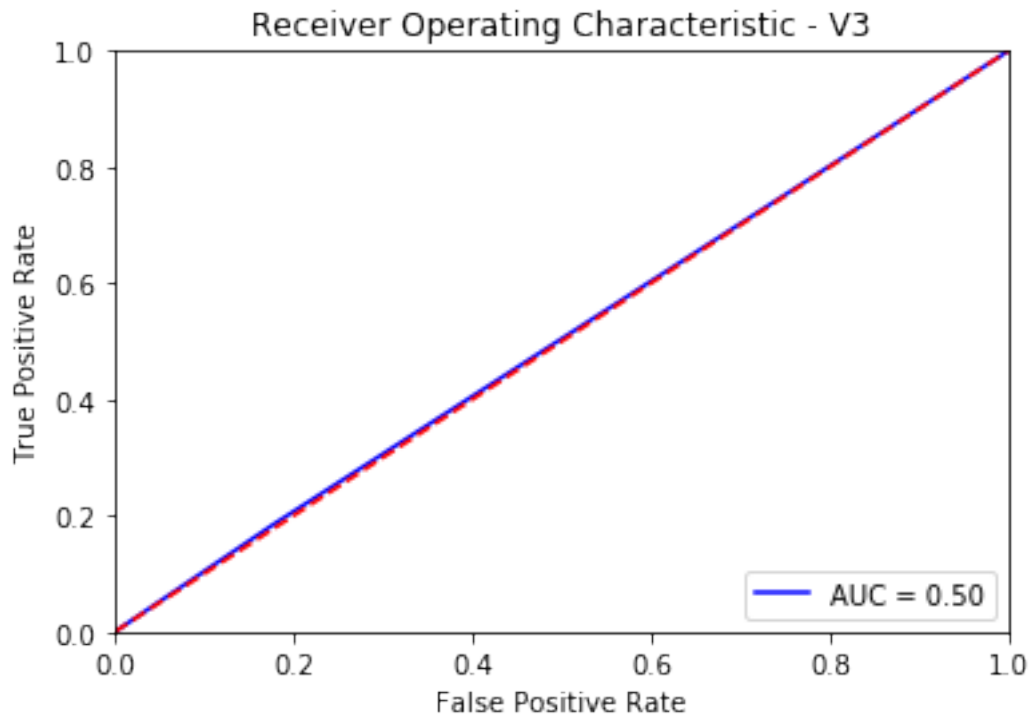
ROC Curve for v2



```
In [30]: plt.title('Receiver Operating Characteristic - V3')
plt.plot(fpr3, tpr3, 'b', label = 'AUC = %0.2f' % roc_auc3)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.savefig("V3-Classification-ROC.png")
plt.show()
```

ROC Curve for v3



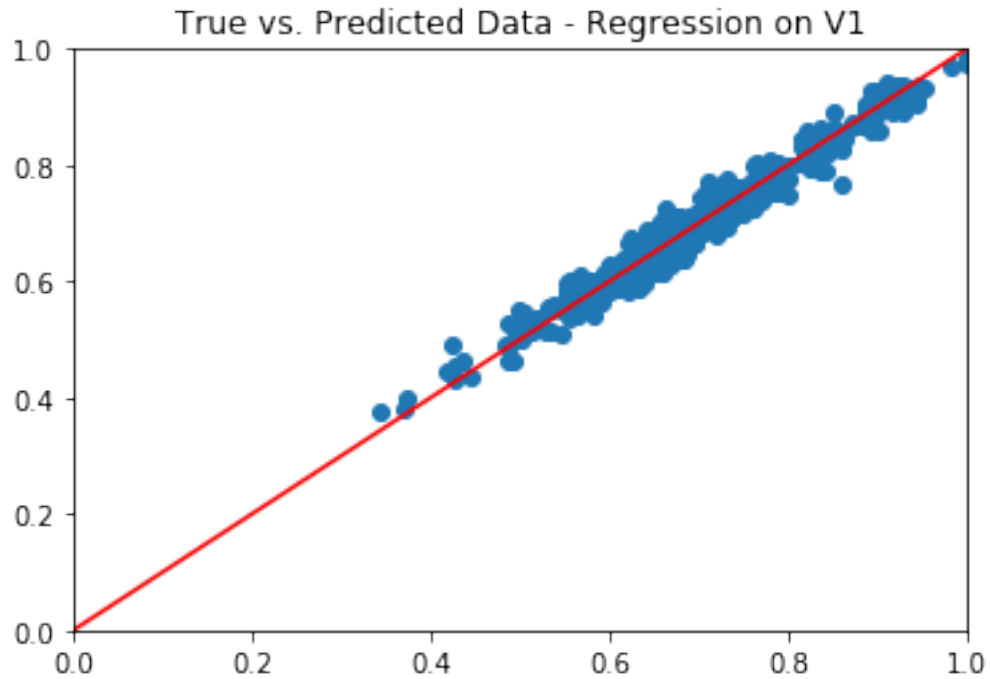
```
In [31]: model_staticRegV1.score(x_testR1, y_testR1)
```

```
Out[31]: 0.9632904292279241
```

```
In [32]: y_predictedR1 = model_staticRegV1.predict(x_testR1)
```

```
In [33]: import matplotlib.lines as mlines
plt.title('True vs. Predicted Data - Regression on V1')
plt.scatter(y_testR1, y_predictedR1)
line = mlines.Line2D([0, 1], [0, 1], color='red')
plt.plot([0, 1], [0, 1], 'r-')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.savefig("V1-Regression-Comparison.png")
plt.show()
```

plot true vs predicted data (want to see a y=x correlation)



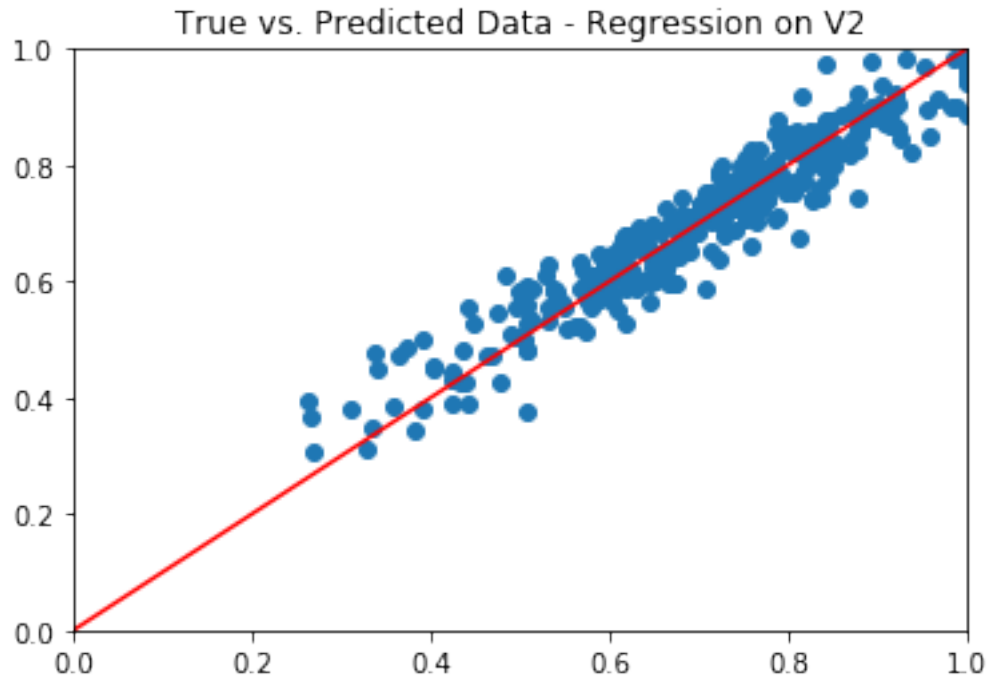
```
In [34]: model_staticRegV2.score(x_testR2, y_testR2)
```

```
Out[34]: 0.9055918556131938
```

```
In [35]: y_predictedR2 = model_staticRegV2.predict(x_testR2)
```

```
In [36]: import matplotlib.lines as mlines
plt.title('True vs. Predicted Data - Regression on V2')
plt.scatter(y_testR2, y_predictedR2)
line = mlines.Line2D([0, 1], [0, 1], color='red')
plt.plot([0, 1], [0, 1], 'r-')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.savefig("V2-Regression-Comparison.png")
plt.show()

# plot true vs predicted data (want to see a y=x correlation)
```



```
In [37]: model_staticRegV3.score(x_testR3, y_testR3)
```

```
Out[37]: 0.9999991356844019
```

```
In [38]: y_predictedR3 = model_staticRegV3.predict(x_testR3)
```

```
In [39]: import matplotlib.lines as mlines
plt.title('True vs. Predicted Data - Regression on V3')
plt.scatter(y_testR3, y_predictedR3)
line = mlines.Line2D([0, 1], [0, 1], color='red')
plt.plot([0, 1], [0, 1], 'r-')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.savefig("V3-Regression-Comparison.png")
plt.show()

# plot true vs predicted data (want to see a y=x correlation)
```



In [40]: *### START HERE!*

```
import matplotlib.pyplot as plt
```

```
sim_staticV4 = pd.read_csv("../Data/Static_Data_ICU_var4_peak_v2.csv")
```

```
sim_staticV4 = sim_staticV4.drop(columns=["Unnamed: 0", "CHM"])
```

```
sim_staticV4 = sim_staticV4.replace(["M", "F"], [0,1])
```

```
sim_staticV4.columns
```

```
# same process as above just for high variance dataset -- not commenting below but th
```

```
Out[40]: Index(['ICU', 'Age', 'Sex', 'Survival Probability', 'Score', 'Kidney', 'Blood',
               'Cholesterol', 'Immune', 'ApacheII', 'Urine Volume', 'Creatinine',
               'Urea', 'O2', 'HR', 'BP', 'RBC', 'LDL', 'HDL', 'Platelet', 'WBC', 'INR',
               'Death'],
              dtype='object')
```

```
In [41]: x_trainR4, x_testR4, y_trainR4, y_testR4 = train_test_split(sim_staticV4.drop(["ICU",
```

```
In [42]: model_staticRegV4 = RandomForestRegressor(n_estimators=100)
```

```
model_staticRegV4.fit(x_trainR4, y_trainR4)
```

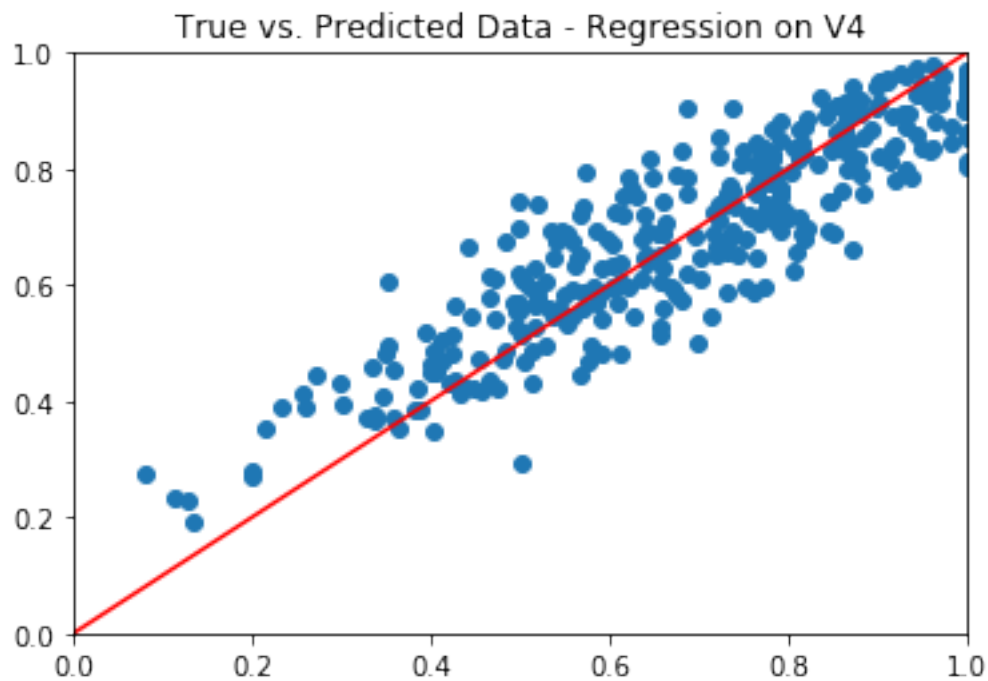
```
Out[42]: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                               max_depth=None, max_features='auto', max_leaf_nodes=None,
                               max_samples=None, min_impurity_decrease=0.0,
                               min_impurity_split=None, min_samples_leaf=1,
                               min_samples_split=2, min_weight_fraction_leaf=0.0,
                               n_estimators=100, n_jobs=None, oob_score=False,
                               random_state=None, verbose=0, warm_start=False)
```

```
In [43]: model_staticRegV4.score(x_testR4, y_testR4)
```

```
Out[43]: 0.8114976152528983
```

```
In [44]: y_predictedR4 = model_staticRegV4.predict(x_testR4)
```

```
In [45]: import matplotlib.lines as mlines
plt.title('True vs. Predicted Data - Regression on V4')
plt.scatter(y_testR4, y_predictedR4)
line = mlines.Line2D([0, 1], [0, 1], color='red')
plt.plot([0, 1], [0, 1], 'r-')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.savefig("V4-Regression-Comparison.png")
plt.show()
```



```
In [1]: RV1_feature_importances = pd.DataFrame(model_staticRegV1.feature_importances_,
                                                index = x_trainR1.columns,
                                                columns=['importance']).sort_values('importance',
RV1_feature_importances

# rank feature importance on Gini index
```

NameError

Traceback (most recent call last)

```
<ipython-input-1-fe08ca6e93f0> in <module>()
----> 1 RV1_feature_importances = pd.DataFrame(model_staticRegV1.feature_importances_,
      2                                     index = x_trainR1.columns,
      3                                     columns=['importance']).sort_values('importance',
      4 RV1_feature_importances
```

NameError: name 'pd' is not defined

```
In [55]: RV2_feature_importances = pd.DataFrame(model_staticRegV2.feature_importances_,
      index = x_trainR2.columns,
      columns=['importance']).sort_values('importance',

RV2_feature_importances
```

```
Out [55]:
```

	importance
Blood	0.310696
Score	0.192722
Immune	0.160499
Kidney	0.136271
Cholesterol	0.124656
Age	0.013184
INR	0.007949
LDL	0.007047
Platelet	0.005341
ApacheII	0.004963
WBC	0.004746
HDL	0.004684
BP	0.004180
Creatinine	0.004135
Urea	0.003893
O2	0.003824
Urine Volume	0.003711
HR	0.003676
RBC	0.003169
Sex	0.000654

```
In [56]: RV3_feature_importances = pd.DataFrame(model_staticRegV3.feature_importances_,
      index = x_trainR3.columns,
      columns=['importance']).sort_values('importance',

RV3_feature_importances
```

```
Out [56]:
```

	importance
Age	0.220330
ApacheII	0.177331
INR	0.158671

LDL	0.157207
Blood	0.053940
Kidney	0.053715
Immune	0.052674
O2	0.048525
Cholesterol	0.047285
Score	0.030198
Platelet	0.000068
HR	0.000056
Urine_Volume	0.000000
Creatinine	0.000000
Sex	0.000000
BP	0.000000
RBC	0.000000
HDL	0.000000
WBC	0.000000
Urea	0.000000

```
In [57]: RV4_feature_importances = pd.DataFrame(model_staticRegV4.feature_importances_,
                                                index = x_trainR4.columns,
                                                columns=['importance']).sort_values('importance',
RV4_feature_importances
```

```
Out [57]:
```

	importance
Cholesterol	0.342299
Immune	0.153941
Blood	0.152139
Kidney	0.113123
Score	0.102564
RBC	0.015791
O2	0.013796
HR	0.013323
Age	0.011820
INR	0.010699
LDL	0.009893
ApacheII	0.009221
Urine Volume	0.007415
Urea	0.007408
WBC	0.007355
HDL	0.007209
Creatinine	0.007056
BP	0.006958
Platelet	0.006893
Sex	0.001098

```
In [ ]:
```