In [418]:
```
#Import the packages
import sys
#!{sys.executable} -m pip install torch
#!{sys.executable} -m pip install plotly
#!{sys.executable} -m pip install scikit-learn
import warnings
warnings.filterwarnings('ignore')

import torch
import torch.nn as nn
from torch.optim import SGD
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
```

In [302]:
```
dataset=pd.read_csv("/Users/xueying/Downloads/SPH6004/Assignment 1 questions (individua/train.csv")

column_names=list(dataset.columns)
#print(column_names)
print(dataset.shape)
print(dataset.isna().sum())
#notice that Capillary refill rate ,  Fraction inspired oxygen, Height have a large
#amount of missing values, hence drop them in the following analysis
dataset_clean=dataset[ ['icustay id','Diastolic blood pressure',  'Glascow coma scale total', 'Glascow coma s
cale eye opening',
                        'Glascow coma scale motor response','Glascow coma scale verbal response',
                       'Glucose', 'Heart Rate', 'Mean blood pressure', 'Oxygen saturation', 'Respiratory rat
e',
                       'Systolic blood pressure', 'Temperature','Weight', 'pH', 'label']]
#drop those entries with any NA
data_cleaned=dataset_clean.dropna()
print(data_cleaned.shape)
#data_cleaned.head(20)
```

```
(17903, 19)
icustay id                              0
Capillary refill rate               17580
Diastolic blood pressure              214
Fraction inspired oxygen            12594
Glascow coma scale eye opening        164
Glascow coma scale motor response     166
Glascow coma scale total             7516
Glascow coma scale verbal response    167
Glucose                                17
Heart Rate                            214
Height                              14518
Mean blood pressure                   216
Oxygen saturation                     120
Respiratory rate                      220
Systolic blood pressure               214
Temperature                           349
Weight                               4851
pH                                   3127
label                                   0
dtype: int64
(5145, 16)
```

Out[302]:

	icustay id	Diastolic blood pressure	Glascow coma scale total	Glascow coma scale eye opening	Glascow coma scale motor response	Glascow coma scale verbal response	Glucose	Heart Rate	Mean blood pressure	Oxygen saturation	Respiratory rate	S pı
6	7039_episode1_timeseries.csv	59.0	8.0	1 No Response	6 Obeys Commands	1.0 ET/Trach	120.0	81.0	91.000000	97.0	18.0	
10	2344_episode1_timeseries.csv	70.0	10.0	3 To speech	6 Obeys Commands	1.0 ET/Trach	166.0	98.0	97.000000	100.0	2.0	
13	29252_episode1_timeseries.csv	58.0	3.0	1 No Response	1 No Response	1.0 ET/Trach	152.0	94.0	75.000000	100.0	10.0	
16	27333_episode1_timeseries.csv	80.0	8.0	1 No Response	6 Obeys Commands	1.0 ET/Trach	223.0	69.0	100.000000	100.0	16.0	
19	24088_episode1_timeseries.csv	39.0	15.0	4 Spontaneously	6 Obeys Commands	5 Oriented	300.0	96.0	65.000000	87.0	24.0	
29	19996_episode1_timeseries.csv	70.0	3.0	1 No Response	1 No Response	1.0 ET/Trach	156.0	93.0	81.000000	100.0	10.0	
31	4793_episode1_timeseries.csv	68.0	8.0	2 To pain	5 Localizes Pain	1.0 ET/Trach	495.0	107.0	89.000000	96.0	32.0	
32	16362_episode1_timeseries.csv	75.0	9.0	3 To speech	5 Localizes Pain	1.0 ET/Trach	175.0	109.0	91.000000	98.0	41.0	
36	14083_episode1_timeseries.csv	63.0	3.0	1 No Response	1 No Response	1.0 ET/Trach	159.0	88.0	82.000000	100.0	14.0	
37	3184_episode1_timeseries.csv	74.0	15.0	4 Spontaneously	6 Obeys Commands	5 Oriented	103.0	120.0	89.666702	100.0	18.0	
44	222_episode3_timeseries.csv	61.0	15.0	4 Spontaneously	6 Obeys Commands	5 Oriented	172.0	74.0	74.000000	92.0	24.0	
48	12_episode1_timeseries.csv	68.0	5.0	3 To speech	1 No Response	1.0 ET/Trach	170.0	86.0	92.000000	100.0	12.0	
51	30360_episode1_timeseries.csv	77.0	3.0	1 No Response	1 No Response	1.0 ET/Trach	136.0	83.0	95.000000	100.0	13.0	
52	32399_episode1_timeseries.csv	82.0	10.0	3 To speech	6 Obeys Commands	1 No Response	134.0	95.0	90.000000	97.0	8.0	
54	28326_episode1_timeseries.csv	62.0	10.0	3 To speech	6 Obeys Commands	1.0 ET/Trach	261.0	95.0	69.000000	97.0	22.0	

	icustay id	Diastolic blood pressure	Glascow coma scale total	Glascow coma scale eye opening	Glascow coma scale motor response	Glascow coma scale verbal response	Glucose	Heart Rate	Mean blood pressure	Oxygen saturation	Respiratory rate
55	22706_episode1_timeseries.csv	54.0	15.0	4 Spontaneously	6 Obeys Commands	5 Oriented	205.0	98.0	64.000000	93.0	26.0
62	4743_episode1_timeseries.csv	53.0	7.0	1 No Response	4 Flex-withdraws	1.0 ET/Trach	139.0	86.0	66.666702	100.0	29.0
65	27295_episode1_timeseries.csv	65.0	11.0	4 Spontaneously	6 Obeys Commands	1.0 ET/Trach	131.0	66.0	90.000000	100.0	15.0
67	11550_episode1_timeseries.csv	71.0	15.0	4 Spontaneously	6 Obeys Commands	5 Oriented	97.0	97.0	84.000000	97.0	19.0
71	18960_episode1_timeseries.csv	73.0	3.0	1 No Response	1 No Response	1.0 ET/Trach	115.0	131.0	81.000000	100.0	12.0

In [303]:
```
continuous_x=data_cleaned[ ['Diastolic blood pressure',  'Glascow coma scale total',
                       'Glucose', 'Heart Rate', 'Mean blood pressure', 'Oxygen saturation', 'Respiratory rat
e',
                       'Systolic blood pressure', 'Temperature','Weight', 'pH']].apply(lambda x: (x-x.mean())
/ x.std(), axis=0)

#generate dummy variables for selected column
with_dummies=pd.get_dummies( data=data_cleaned, columns=['Glascow coma scale eye opening','Glascow coma scale
motor response','Glascow coma scale verbal response'], drop_first=True)

#select those newly generated dummy variables
dummies_list=list()
for i in list(with_dummies.columns):
    if i not in column_names:
        dummies_list.append(i)
dummies_x=with_dummies[dummies_list]

#concate continuous standardized x and dummy categorical x together
X = pd.concat([continuous_x, dummies_x] , axis = 1)
concated_fulldf=X
column_list=list(X.columns)
#X["bias"]=1
#print(list(X.columns))
Y =data_cleaned["label"]

#transform to tensot
X=torch.tensor(X.to_numpy()).type(torch.float32)
Y=torch.tensor(Y.to_numpy()).type(torch.float32)
Y = Y.reshape(-1,1)

print(X.shape)
print(Y.shape)

#list(with_dummies.columns)
print(column_list)
```

```
torch.Size([5145, 24])
torch.Size([5145, 1])
['Diastolic blood pressure', 'Glascow coma scale total', 'Glucose', 'Heart Rate', 'Mean blood pressure', 'Ox
ygen saturation', 'Respiratory rate', 'Systolic blood pressure', 'Temperature', 'Weight', 'pH', 'Glascow com
a scale eye opening_2 To pain', 'Glascow coma scale eye opening_3 To speech', 'Glascow coma scale eye openin
g_4 Spontaneously', 'Glascow coma scale motor response_2 Abnorm extensn', 'Glascow coma scale motor response
_3 Abnorm flexion', 'Glascow coma scale motor response_4 Flex-withdraws', 'Glascow coma scale motor response
_5 Localizes Pain', 'Glascow coma scale motor response_6 Obeys Commands', 'Glascow coma scale verbal respons
e_1.0 ET/Trach', 'Glascow coma scale verbal response_2 Incomp sounds', 'Glascow coma scale verbal response_3
Inapprop words', 'Glascow coma scale verbal response_4 Confused', 'Glascow coma scale verbal response_5 Orie
nted']
```

In [362]:

```
#splitting strategy: 80% train, 20% test
observ_num = X.shape[0]
observ_shuffled = torch.randperm(observ_num)

train_proportion = 0.8
train_list = observ_shuffled[:int(train_proportion*observ_num)]
test_list = observ_shuffled[int(train_proportion*observ_num):]

x_train, x_test = X[train_list], X[test_list]
y_train, y_test = Y[train_list], Y[test_list]

#x_train
```

In [363]:

```
#Implement Elastic Net Rregression to perform feature selection
nIter = 1000 # We perform 1000 iterations of GD steps
loss_record = []

inputSize, outputSize = 24, 1


model_elastic = nn.Sequential(nn.Linear(inputSize, outputSize), # innner product
                      nn.Sigmoid())               # sigmoid

# binary cross entrophy loss
J = nn.BCELoss()


#lambda parameter for L1 norm regularization
lbd=0.025

# SGD optimizer in PyTorch
# 'weight_decay' parameter indicates L2 norm (Ridge regression) is used
optimizer = SGD(model_elastic.parameters(),
                lr = 0.05,
                weight_decay=0.025,
                momentum = 0.5)

#L1 norm regurlarization, sum up the absolute value of every parameters
def L1Norm(model):
     result = torch.tensor(0)
     for param in model.parameters():
        result = result + param.abs().sum()

     return result


for i in range(nIter):
    #print(i)
    optimizer.zero_grad()
    hat_y = model_elastic(x_train)
    #print(hat_y)
    loss = J(hat_y,y_train)
    (loss + lbd*L1Norm(model_elastic)).backward() #?
    optimizer.step()
```

```
    loss_record.append(loss.item())
    if i%100 == 99 or i==0:
        print('At iteration {} loss is {:.4f}'.format(i+1,loss.item()))

#check the thetas after optimization
params=torch.tensor(0)
for param in model_elastic.parameters():
    params=param.abs()
    break
    print(param.abs())
print(params)
```

```
At iteration 1 loss is 0.7888
At iteration 100 loss is 0.4350
At iteration 200 loss is 0.4324
At iteration 300 loss is 0.4324
At iteration 400 loss is 0.4324
At iteration 500 loss is 0.4324
At iteration 600 loss is 0.4324
At iteration 700 loss is 0.4324
At iteration 800 loss is 0.4323
At iteration 900 loss is 0.4323
At iteration 1000 loss is 0.4322
tensor([[1.0571e-03, 1.2866e-03, 4.6380e-04, 2.2794e-03, 1.5325e-04, 8.4678e-04,
         1.0761e-01, 1.1370e-03, 6.8552e-04, 2.8688e-03, 2.0877e-03, 8.6852e-04,
         3.1887e-04, 1.6228e-03, 1.5235e-03, 6.1009e-04, 4.8038e-04, 1.3648e-03,
         1.7437e-01, 2.1459e-01, 1.2495e-04, 7.8747e-04, 7.0564e-04, 3.6913e-03]],
       grad_fn=<AbsBackward0>)
```

In [364]:
```
#print(column_list)
params_list=params.tolist()[0]
reduced_feature=list()
dropped_feature=list()
#Based on the output of parameters after elastic regularization, drop those feature with coefficient<0.001
for i in range(0, len(params_list)):
    if params_list[i]< 0.001:
        #print(params_list[i])
        dropped_feature.append(column_list[i])
    else:
        reduced_feature.append(column_list[i])

print(len(dropped_feature))
print(len(reduced_feature))
#left with 15 features
print(reduced_feature)
```

```
11
13
['Diastolic blood pressure', 'Glascow coma scale total', 'Heart Rate', 'Respiratory rate', 'Systolic blood p
ressure', 'Weight', 'pH', 'Glascow coma scale eye opening_4 Spontaneously', 'Glascow coma scale motor respon
se_2 Abnorm extensn', 'Glascow coma scale motor response_5 Localizes Pain', 'Glascow coma scale motor respon
se_6 Obeys Commands', 'Glascow coma scale verbal response_1.0 ET/Trach', 'Glascow coma scale verbal response
_5 Oriented']
```

In [386]:
```
X_reduced=concated_fulldf[reduced_feature]
X_reduced=torch.tensor(X_reduced.to_numpy()).type(torch.float32)
print(X_reduced.size())
print(Y.size())

#regrenerate x_train, x_test based on dataframe with selected features
x_train, x_test = X_reduced[train_list], X_reduced[test_list]
y_train, y_test = Y[train_list], Y[test_list]
X_reduced.size()[1]
x_test.shape
```

```
torch.Size([5145, 13])
torch.Size([5145, 1])
```

Out[386]:
```
torch.Size([1029, 13])
```

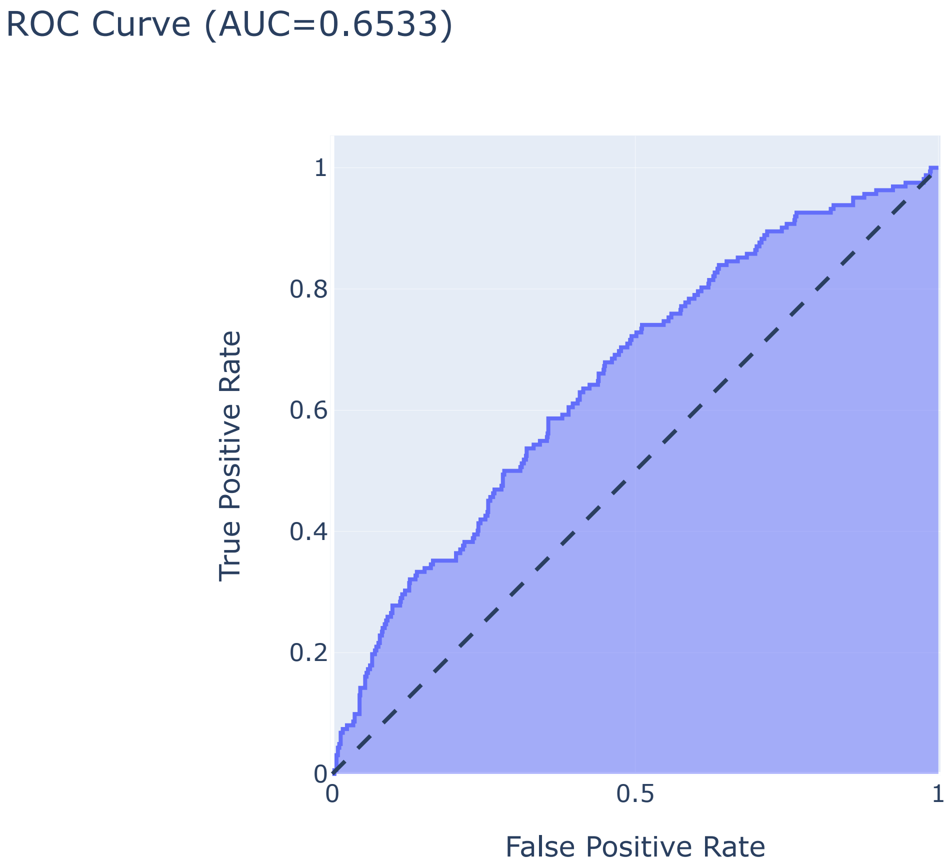
In [387]:
```
#Model 1: Logistic Regression with Gradient Descent
nIter = 1000 # We perform 1000 iterations of GD steps
loss_record = []

inSize, outSize = X_reduced.size()[1], 1

model_logistic = nn.Sequential(nn.Linear(inSize, outSize), # innner product
                      nn.Sigmoid())                # sigmoid

# binary cross entropy loss
J = nn.BCELoss()

# SGD optimizer in PyTorch
optimizer = SGD(model_logistic.parameters(),
                lr = 0.05,
                momentum = 0.5)

for i in range(nIter):
    optimizer.zero_grad()
    hat_y = model_logistic(x_train)
    loss = J(hat_y,y_train)
    loss.backward()
    optimizer.step()
    loss_record.append(loss.item())
    if i%100 == 99 or i==0:
        print('At iteration {} loss is {:.4f}'.format(i+1,loss.item()))

hat_y_test = model_logistic(x_test).detach()
```

```
At iteration 1 loss is 0.6413
At iteration 100 loss is 0.4046
At iteration 200 loss is 0.4030
At iteration 300 loss is 0.4025
At iteration 400 loss is 0.4021
At iteration 500 loss is 0.4018
At iteration 600 loss is 0.4015
At iteration 700 loss is 0.4013
At iteration 800 loss is 0.4011
At iteration 900 loss is 0.4009
At iteration 1000 loss is 0.4008
```

In [388]:
```
#Calculate AUROC to acess model performance
hat_y_test = model_logistic(x_test).detach()
metrics.roc_auc_score(y_test, hat_y_test)

false_pos_rate, true_pos_rate, thresholds =roc_curve(y_test, hat_y_test)
roc_auc = auc(false_pos_rate, true_pos_rate)

index_i = np.arange(len(true_pos_rate))
roc_curve_df = pd.DataFrame({'equation_true_false_rate' : pd.Series(true_pos_rate-(1-false_pos_rate), index=i
ndex_i), 'threshold' : pd.Series(thresholds, index=index_i)})
# The optimal cut off occurs when high in true postive rate and low in false postive rate
# true_pos_rate-(1-false_pos_rate) approaches 0 when reaching the optimal cut off point
roc_optimal_threshold = roc_curve_df.iloc[(roc_curve_df.equation_true_false_rate-0).abs().argsort()[:1]]

optimal_threshold=list(roc_optimal_threshold['threshold'])

#hat_y_test_pred=hat_y_test.tolist().map(lambda p: 1 if p > threshold else 0)
#print(hat_y_test_pred)


hat_y_test_pred=list()
for i in range(0, len(hat_y_test.tolist())):
    if hat_y_test.tolist()[i]>optimal_threshold:
        hat_y_test_pred.append(1)
    else:
        hat_y_test_pred.append(0)
hat_y_test_pred_reshape=torch.reshape(torch.FloatTensor(hat_y_test_pred),(1029,1))
hat_y_test_pred_reshape

accuracy_at_optithresh=(y_test==hat_y_test_pred_reshape).sum()/len(y_test)

print(accuracy_at_optithresh.item())


#Accuracy_AUROC(x_test, y_test)
```

```
0.6044703722000122
```

In [441]:
```
fig = px.area(
    x=false_pos_rate, y=true_pos_rate,
    title=f'ROC Curve (AUC={auc(false_pos_rate, true_pos_rate):.4f})',
    labels=dict(x='False Positive Rate', y='True Positive Rate'),
    width=700, height=500
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)

fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')
fig.show()
```


ROC Curve (AUC=0.6533)
True Positive Rate
False Positive Rate
0
0.2
0.4
0.6
0.8
1
0
0.5
1

In [405]:
```
#Model 2 Simple Decision tree
from sklearn.tree import DecisionTreeClassifier

TreeClassifier = DecisionTreeClassifier(criterion="entropy", max_depth=10, random_state=15)
clf = TreeClassifier.fit(x_train,y_train)

y_test_pred_mod2 = clf.predict(x_test)
y_test_array=y_test.detach().numpy().reshape(y_test_array.shape[0])
#print(y_test_array)
#print(y_test_pred_mod2)
print((y_test_array==y_test_pred_mod2).sum()/len(y_test_array))
```

```
0.7949465500485908
```

In [191]:
```
#Model 3 Adaboost
from sklearn.ensemble import AdaBoostClassifier

BoostClassifier = AdaBoostClassifier(n_estimators=20)
abc = BoostClassifier.fit(x_train,y_train.ravel())

y_test_pred_mod3 = abc.predict(x_test)

print((y_test_array==y_test_pred_mod3).sum()/len(y_test_array))
```

```
0.8668610301263362
```

In [194]:
```
#Model 4 Random Forest
from sklearn.ensemble import RandomForestClassifier

ForestClassifier =RandomForestClassifier(
    n_estimators=50,
    n_jobs=-1,
    criterion='entropy'
)
rfc = ForestClassifier.fit(x_train,y_train.ravel())

y_test_pred_mod4 = rfc.predict(x_test)


print((y_test_array==y_test_pred_mod4).sum()/len(y_test_array))
```

```
0.8678328474246841
```

In [202]:
```
#Model 5 Linear SVM
from sklearn import svm

linear_SVM = svm.SVC(kernel='linear')
linear_SVM.fit(x_train,y_train.ravel())

y_test_pred_mod5 = linear_SVM.predict(x_test)
accuracy_mod5 = (y_test_array==y_test_pred_mod5).sum()/len(y_test_array)
print(accuracy_mod5)
```

```
0.8658892128279884
```

In [204]:
```
#Model 6 Non-linear Kernel SVM
rbf_SVM = svm.SVC(kernel='rbf')
rbf_SVM.fit(x_train,y_train.ravel())

y_test_pred_mod6 = rbf_SVM.predict(x_test)

accuracy_mod6 = (y_test_array==y_test_pred_mod6).sum()/len(y_test_array)
print(accuracy_mod6)
```

```
/Users/xueying/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193: FutureWarning:

The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.


0.8658892128279884
```

In [402]:
```
def Calcuate_accuracy_AUROC(x_test, y_test):
    hat_y_test = model_logistic(x_test).detach()
    metrics.roc_auc_score(y_test, hat_y_test)

    false_pos_rate, true_pos_rate, thresholds =roc_curve(y_test, hat_y_test)
    roc_auc = auc(false_pos_rate, true_pos_rate)

    index_i = np.arange(len(true_pos_rate))
    roc_curve_df = pd.DataFrame({'equation_true_false_rate' : pd.Series(true_pos_rate-(1-false_pos_rate), ind
ex=index_i), 'threshold' : pd.Series(thresholds, index=index_i)})
    # The optimal cut off occurs when high in true postive rate and low in false postive rate
    # true_pos_rate-(1-false_pos_rate) approaches 0 when reaching the optimal cut off point
    roc_optimal_threshold = roc_curve_df.iloc[(roc_curve_df.equation_true_false_rate-0).abs().argsort()[:1]]

    optimal_threshold=list(roc_optimal_threshold['threshold'])

    #hat_y_test_pred=hat_y_test.tolist().map(lambda p: 1 if p > threshold else 0)
    #print(hat_y_test_pred)


    hat_y_test_pred=list()
    for i in range(0, len(hat_y_test.tolist())):
        if hat_y_test.tolist()[i]>optimal_threshold:
            hat_y_test_pred.append(1)
        else:
            hat_y_test_pred.append(0)
    hat_y_test_pred_reshape=torch.reshape(torch.FloatTensor(hat_y_test_pred),(y_test_fold.shape[0],1))
    hat_y_test_pred_reshape

    accuracy_at_optithresh=(y_test==hat_y_test_pred_reshape).sum()/len(y_test)

    return(accuracy_at_optithresh.item())
```

In [438]:
```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10)
kf.get_n_splits(X_reduced)
#iterate through train and test folds
logistic_acc=list()
simple_decisiontree_acc=list()
adaboost_acc=list()
random_forest_acc=list()
linear_svm_acc=list()
nonlinear_svm_acc=list()

for train_index, test_index in kf.split(X_reduced):
     #print('TRAIN:', train_index, 'TEST:', test_index)
    x_train_fold, x_test_fold = X_reduced[train_index], X_reduced[test_index]
    y_train_fold, y_test_fold = Y[train_index], Y[test_index]

    y_test_array=y_test_fold.detach().numpy().reshape(y_test_fold.shape[0])

    #Logistic Regression:
    for i in range(nIter):
        optimizer.zero_grad()
        hat_y_fold = model_logistic(x_train_fold)
        loss = J(hat_y_fold,y_train_fold)
        loss.backward()
        optimizer.step()
        loss_record.append(loss.item())
    logistic_acc.append(Calcuate_accuracy_AUROC(x_test_fold, y_test_fold))

    #Simple decision tree
    clf = TreeClassifier.fit(x_train_fold,y_train_fold)
    y_test_pred_mod2 = clf.predict(x_test_fold)
    simple_decisiontree_acc.append((y_test_array==y_test_pred_mod2).sum()/len(y_test_array))

    #Adaboosting model
    abc = BoostClassifier.fit(x_train_fold,y_train_fold.ravel())
    y_test_pred_mod3 = abc.predict(x_test_fold)
    adaboost_acc.append((y_test_array==y_test_pred_mod3).sum()/len(y_test_array))

    #Random Forest
    rfc = ForestClassifier.fit(x_train_fold,y_train_fold.ravel())
    y_test_pred_mod4 = rfc.predict(x_test_fold)
```

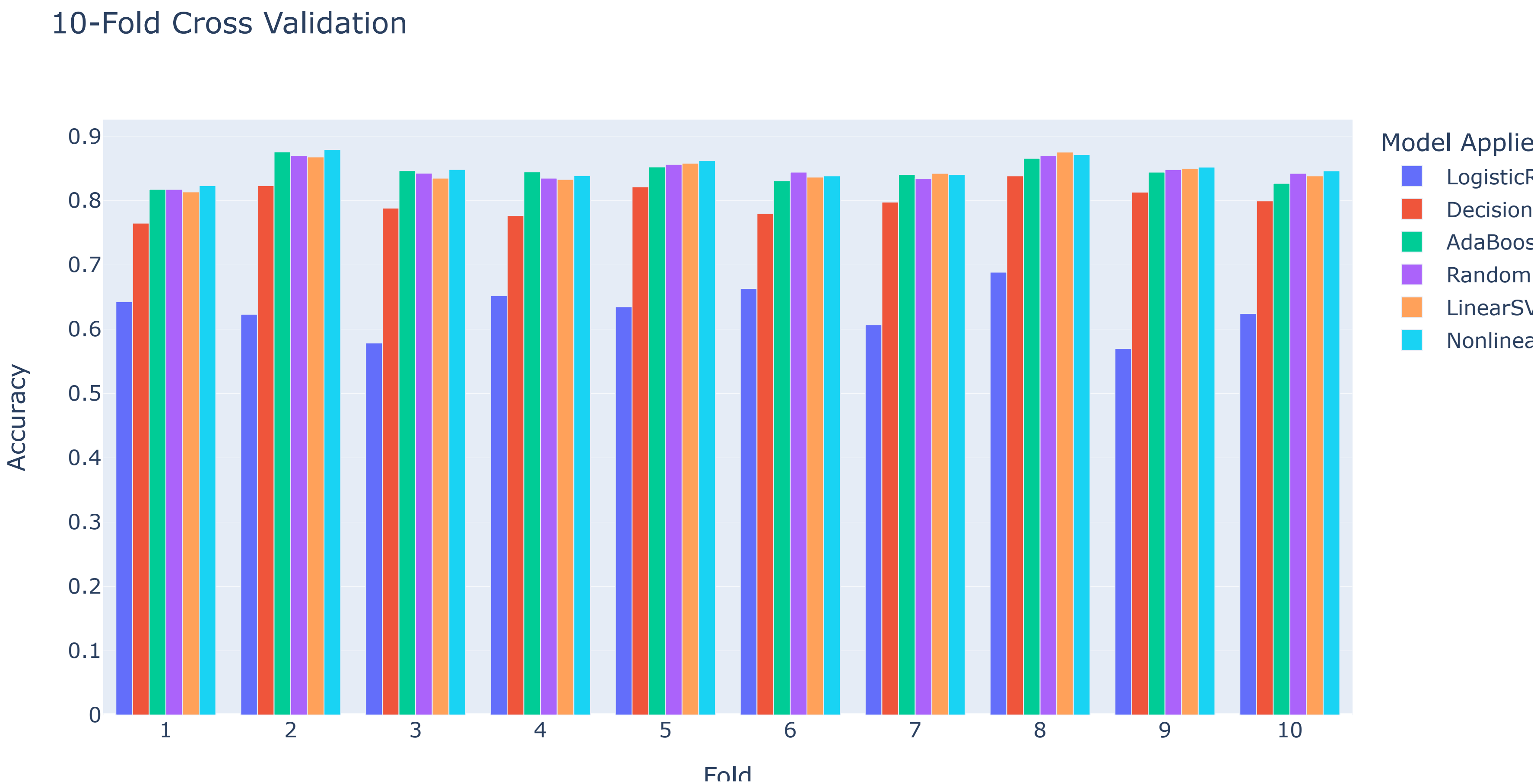
```
    random_forest_acc.append((y_test_array==y_test_pred_mod4).sum()/len(y_test_array))


    #Linear SVM model
    rfc = ForestClassifier.fit(x_train_fold,y_train_fold.ravel())
    y_test_pred_mod4 = rfc.predict(x_test_fold)
    linear_svm_acc.append((y_test_array==y_test_pred_mod4).sum()/len(y_test_array))

    #Nonlinear SVM
    rbf_SVM.fit(x_train_fold,y_train_fold.ravel())
    y_test_pred_mod6 = rbf_SVM.predict(x_test_fold)
    nonlinear_svm_acc.append((y_test_array==y_test_pred_mod6).sum()/len(y_test_array))
```

Out[438]:

```
[0.8174757281553398,
 0.8699029126213592,
 0.8427184466019417,
 0.8349514563106796,
 0.8563106796116505,
 0.8443579766536965,
 0.8346303501945526,
 0.8696498054474708,
 0.8482490272373541,
 0.8424124513618677]
```

In [440]:

```
#Create a bar plot to visualize the performance of models
zipped = list(zip(logistic_acc, simple_decisiontree_acc, adaboost_acc, random_forest_acc, linear_svm_acc,nonl
inear_svm_acc))
performance_df=pd.DataFrame(zipped, columns=['Logistic', 'DecisionTree','AdaBoost','RandomForest','LinearSVM'
,'NonlinearSVM'])
performance_df

import plotly.offline as pyo
import plotly.graph_objs as go
# Set notebook mode to work in offline
pyo.init_notebook_mode()

import plotly.graph_objects as go
folds=['1', '2', '3','4','5','6','7','8','9','10']

fig = go.Figure(data=[
    go.Bar(name='LogisticRegression', x=folds, y=logistic_acc),
    go.Bar(name='DecisionTree', x=folds, y=simple_decisiontree_acc),
    go.Bar(name='AdaBoost', x=folds, y=adaboost_acc),
    go.Bar(name='RandomForest', x=folds, y=random_forest_acc),
    go.Bar(name='LinearSVM', x=folds, y=linear_svm_acc),
    go.Bar(name='NonlinearSVM', x=folds, y=nonlinear_svm_acc),

])
# Change the bar mode
fig.update_layout(barmode='group',
    title="10-Fold Cross Validation",
    xaxis_title="Fold",
    yaxis_title="Accuracy",
    legend_title="Model Applied",)
fig.show()
```


10-Fold Cross Validation
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
10
Fold
Model Applie
LogisticR
Decision
AdaBoos
Random
LinearSV
Nonlinea

In [ ]: