# Importing Libraries

In [28]:
```
# import necessary libraries
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import pandas as pd
import tensorflow as tf
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, cross_val_predict, valida
from sklearn.metrics import classification_report
from sklearn.metrics import recall_score
import matplotlib.ticker as mtick
warnings.filterwarnings("ignore")
```

# Reading the Dataset

In [3]:
```
data = pd.read_csv('HeartAttack.csv')
```

In [4]:
```
data.head(5)
```

Out[4]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 70 | 1 | 4 | 130 | 322 | 0 | 2 | 109 | 0 | 2.4 | 2 | 3 | 3 | 2 |
| **1** | 67 | 0 | 3 | 115 | 564 | 0 | 2 | 160 | 0 | 1.6 | 2 | 0 | 7 | 1 |
| **2** | 57 | 1 | 2 | 124 | 261 | 0 | 0 | 141 | 0 | 0.3 | 1 | 0 | 7 | 2 |
| **3** | 64 | 1 | 4 | 128 | 263 | 0 | 0 | 105 | 1 | 0.2 | 2 | 1 | 7 | 1 |
| **4** | 74 | 0 | 2 | 120 | 269 | 0 | 2 | 121 | 1 | 0.2 | 1 | 1 | 3 | 1 |

There are five categorical values that will need to be converted into binary variables in order to input them into the sklearn models

# Data Manipulation

In [5]:
```
#Seperating out input and output variables
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

In [6]:
```
#Turning categorical values into binary for sklearn
X = pd.get_dummies(X, prefix = ['cp','restecg','slope','thal','ca'], columns =
```

In [7]:
```
#Removing the categorical data which has binary values now and preparing the da
X_0 = X.iloc[:,[0,2,3,5,7]].values
X_1 = X.iloc[:,[1,4,6,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]].values
```

In [8]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
X_0 = sc.fit_transform(X_0)

X = np.hstack((X_0, X_1))
```

We are using MixMaxScaler instead of Standard Scaler given some of the model will be using will be non-linear

# EDA (Explanatory Data Analysis)

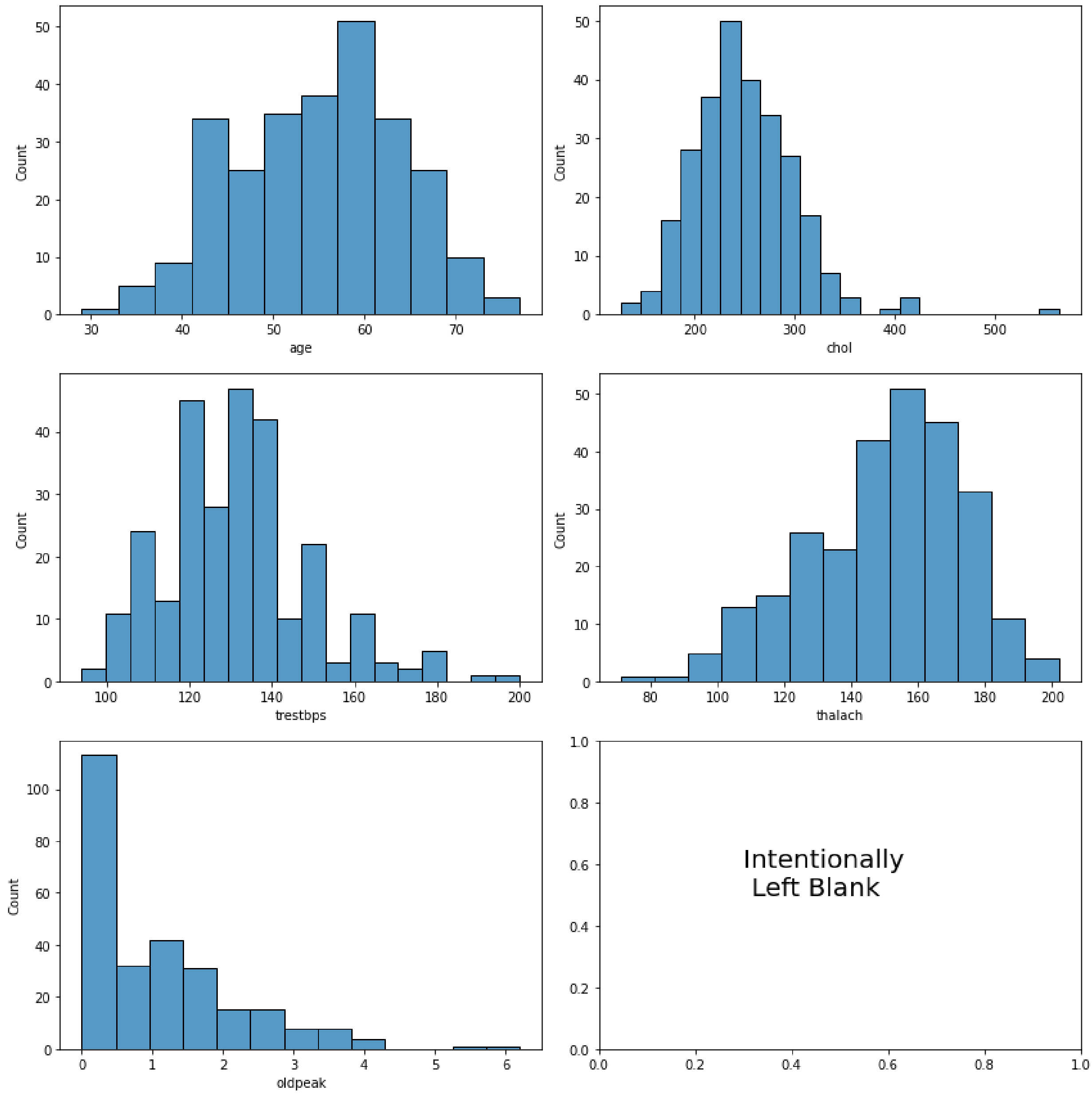
In [9]:
```
pd.DataFrame(data.describe()).style.format('{:.1f}')
```

Out[9]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 270.0 | 270.0 | 270.0 | 270.0 | 270.0 | 270.0 | 270.0 | 270.0 | 270.0 | 270.0 | 270.0 | 27( |
| **mean** | 54.4 | 0.7 | 3.2 | 131.3 | 249.7 | 0.1 | 1.0 | 149.7 | 0.3 | 1.0 | 1.6 | ( |
| **std** | 9.1 | 0.5 | 1.0 | 17.9 | 51.7 | 0.4 | 1.0 | 23.2 | 0.5 | 1.1 | 0.6 | ( |
| **min** | 29.0 | 0.0 | 1.0 | 94.0 | 126.0 | 0.0 | 0.0 | 71.0 | 0.0 | 0.0 | 1.0 | ( |
| **25%** | 48.0 | 0.0 | 3.0 | 120.0 | 213.0 | 0.0 | 0.0 | 133.0 | 0.0 | 0.0 | 1.0 | ( |
| **50%** | 55.0 | 1.0 | 3.0 | 130.0 | 245.0 | 0.0 | 2.0 | 153.5 | 0.0 | 0.8 | 2.0 | ( |
| **75%** | 61.0 | 1.0 | 4.0 | 140.0 | 280.0 | 0.0 | 2.0 | 166.0 | 1.0 | 1.6 | 2.0 | ' |
| **max** | 77.0 | 1.0 | 4.0 | 200.0 | 564.0 | 1.0 | 2.0 | 202.0 | 1.0 | 6.2 | 3.0 | ; |

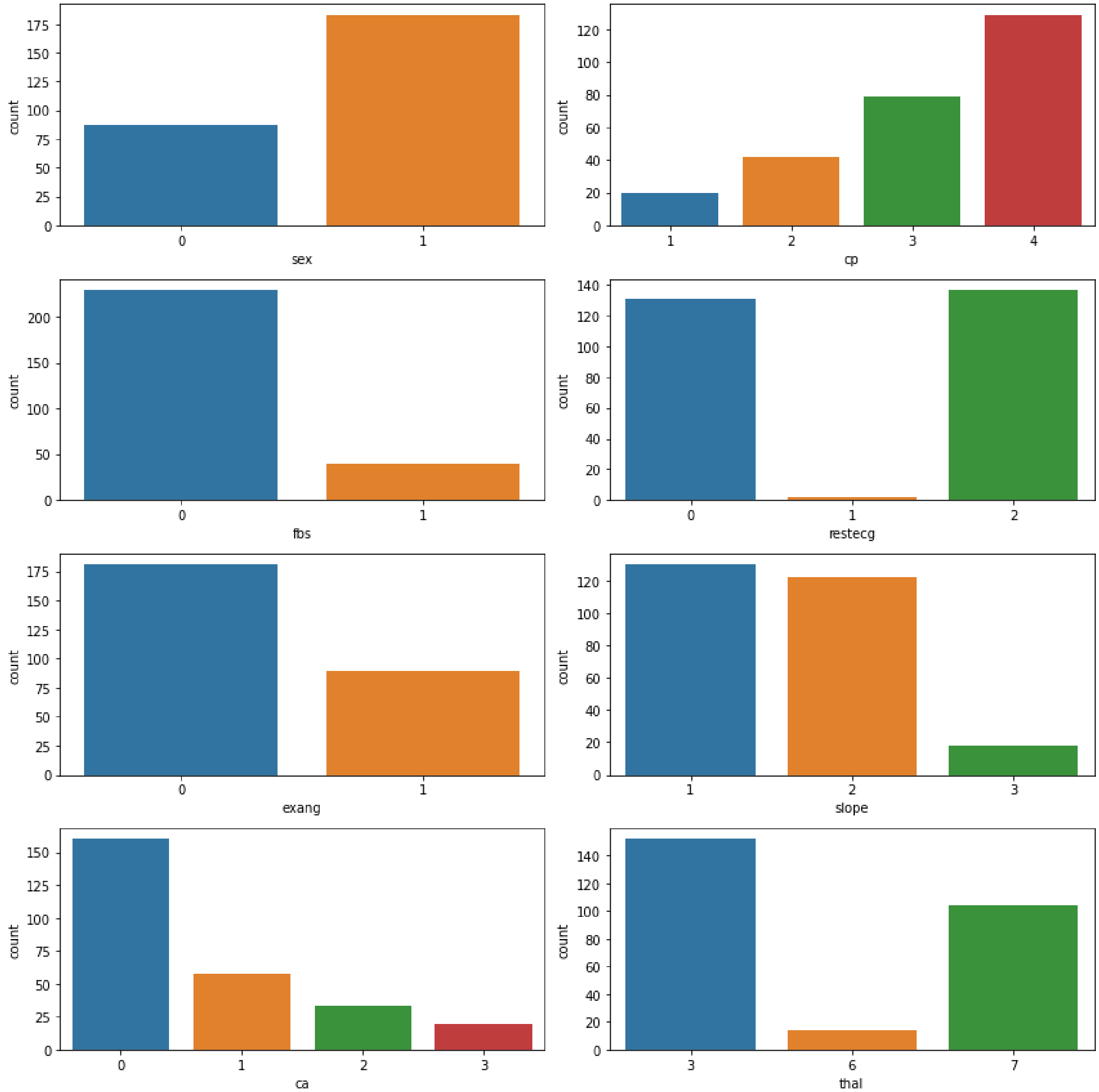
The description given us detailed information on continuous variables and specifies which variables are binary.

In [11]:
```
fig, ax = plt.subplots(3,2, figsize=(12,12), tight_layout = True)
sns.histplot(data = data, x='age', ax = ax[0,0])
sns.histplot(data = data, x='chol', ax = ax[0,1])
sns.histplot(data = data, x='trestbps', ax = ax[1,0])
sns.histplot(data = data, x='thalach', ax = ax[1,1])
sns.histplot(data = data, x='oldpeak', ax = ax[2,0])
ax[2,1].annotate("Intentionally \n Left Blank",(0.3,0.5), size = 20)
plt.show()
```




For "age", we are seeing a normal distribution. Our study has more data on people around ages 55 to 65 which is perfect given that it is more common to see heart attack patients in that age population. All the other variables expect "oldpeak" have a normal distribution. The "oldpeak" is a left skewed distribution. Besides the variable "age" , it is difficult to spot check the data given lack of medical knowledge.

In [12]:
```
fig, ax = plt.subplots(4,2, figsize=(12,12), tight_layout = True)
sns.countplot(x= 'sex', data = data, ax = ax[0,0])
sns.countplot(x= 'cp', data = data, ax = ax[0,1])
sns.countplot(x= 'fbs', data = data, ax = ax[1,0])
sns.countplot(x= 'restecg', data = data, ax = ax[1,1])
sns.countplot(x= 'exang', data = data, ax = ax[2,0])
sns.countplot(x= 'slope', data = data, ax = ax[2,1])
sns.countplot(x= 'ca', data = data, ax = ax[3,0])
sns.countplot(x= 'thal', data = data, ax = ax[3,1])
plt.show()
```

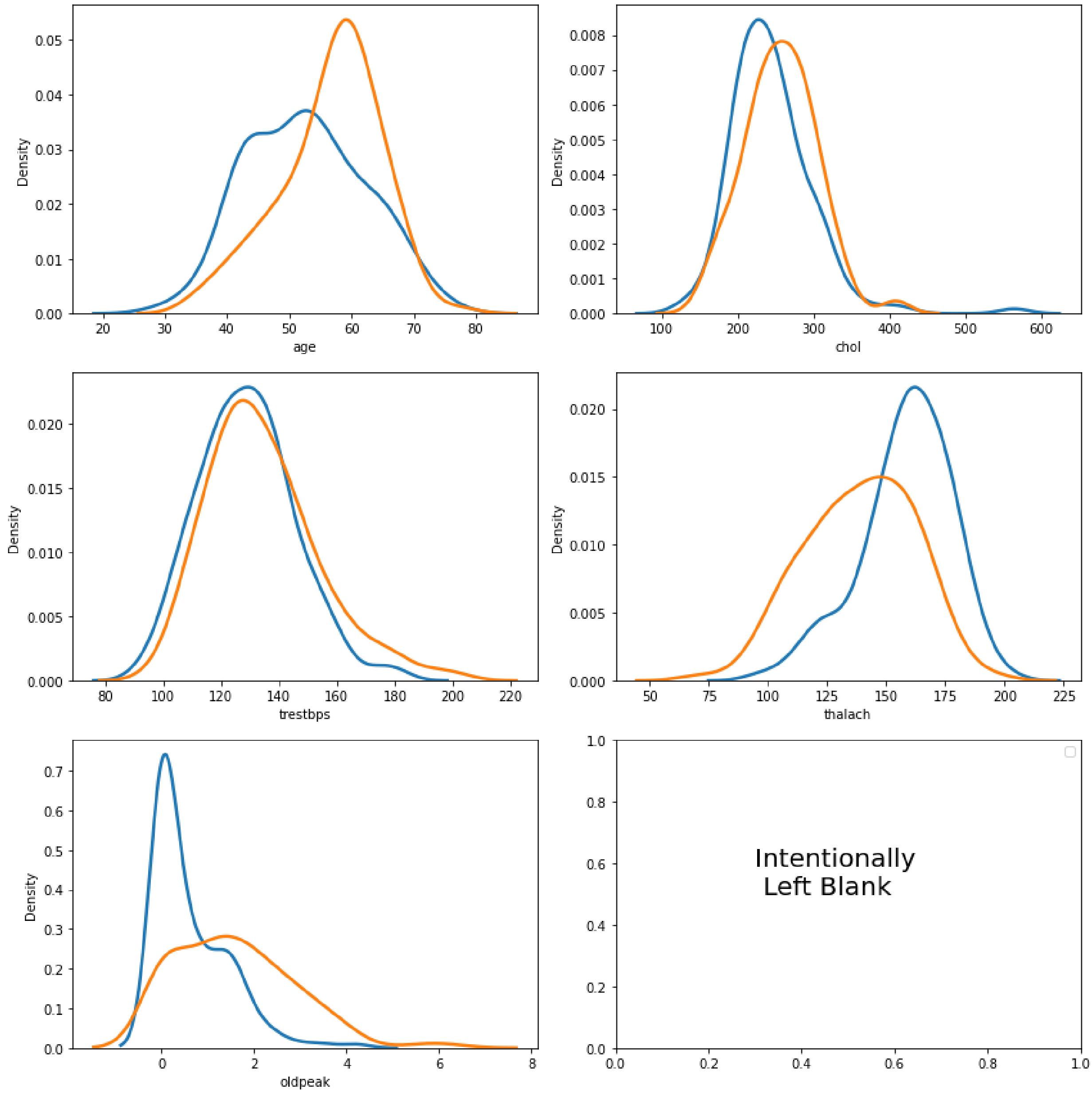



The graphs above show all the binary variables. There is a class imbalance with majority of the variables. If our selected models are not performing as predicted due to overfitting our underfitting, we will need to dig more into the binary variables and conduct under or over sampling as need.

In [13]:
```
fig, ax = plt.subplots(3,2, figsize=(12,12), tight_layout = True)
for i in sorted(data.target.unique()):
    age = data.query("target=={}".format(i))
    sns.distplot(age['age'], label = i, hist = False, kde_kws=dict(linewidth=2
for i in sorted(data.target.unique()):
    chol = data.query("target=={}".format(i))
    sns.distplot(chol['chol'], label = i, hist = False, kde_kws=dict(linewidth=
for i in sorted(data.target.unique()):
    trestbps = data.query("target=={}".format(i))
    sns.distplot(trestbps['trestbps'], label = i, hist = False, kde_kws=dict(l
for i in sorted(data.target.unique()):
    thalach = data.query("target=={}".format(i))
    sns.distplot(thalach['thalach'], label = i, hist = False, kde_kws=dict(line
for i in sorted(data.target.unique()):
    oldpeak = data.query("target=={}".format(i))
    sns.distplot(oldpeak['oldpeak'], label = i, hist = False, kde_kws=dict(line
```

```
ax[2,1].annotate("Intentionally \n Left Blank",(0.3,0.5), size = 20)
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```




These graphs are showing how the target (ouput) variable is distributed thoughout our input varibales. For instance, we are seeing that people between ages 55 to 65 are more likely to get a heart attack compared to someone who is younger.

# Model Deployment and Parameter Choice

Models we will be testing are listed below:

1) Support Vector Machine

2) Random Forest

3) K Nearest Neighbors

4) Logistic Regression

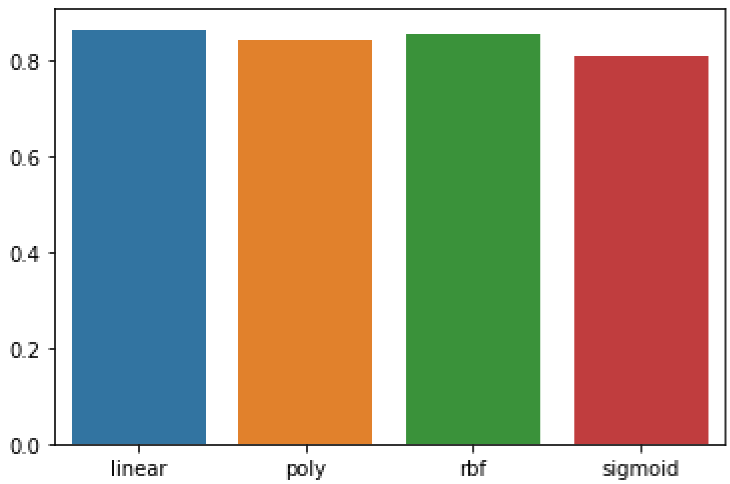
Our goal is find the most accurate and robust model. We will do so by testing parameters of each given model using 3 fold cross validations.

In [33]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rand
```

Support Vector Machine

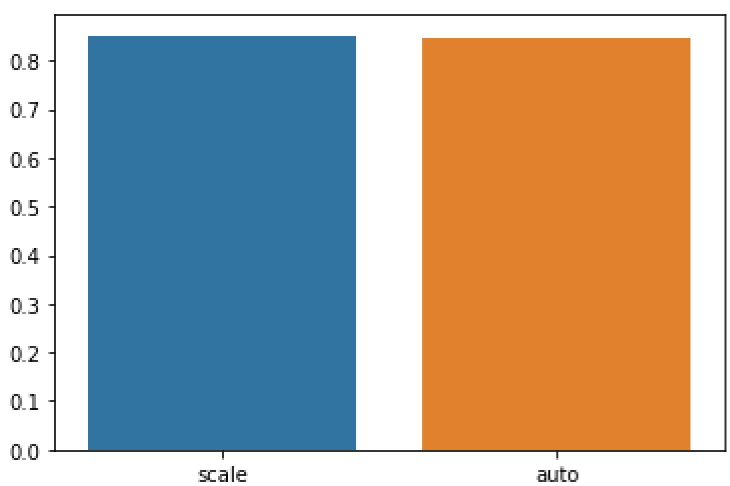
In [34]:
```
train_scores, valid_scores = validation_curve(SVC(), X, y, "kernel",['linear',
sns.barplot(x = ['linear', 'poly', 'rbf', 'sigmoid'], y = [i.mean() for i in va
```

Out[34]: <AxesSubplot:>




In [35]:
```
train_scores, valid_scores = validation_curve(SVC(), X, y, "gamma",['scale','au
sns.barplot(x = ['scale','auto'], y = [i.mean() for i in valid_scores])
```

Out[35]: <AxesSubplot:>




In [36]:
```
classifier_svc = SVC(kernel = 'linear', verbose = 0, random_state = 0)
classifier_svc.fit(X_train, y_train)
y_pred_svc = classifier_svc.predict(X_test)
y_pred_svc_train = classifier_svc.predict(X_train)
print('\nSupport Vector Machine Accuracy Score on Test Set:', end = ' ')
print(accuracy_score(y_test, y_pred_svc)*100)
```

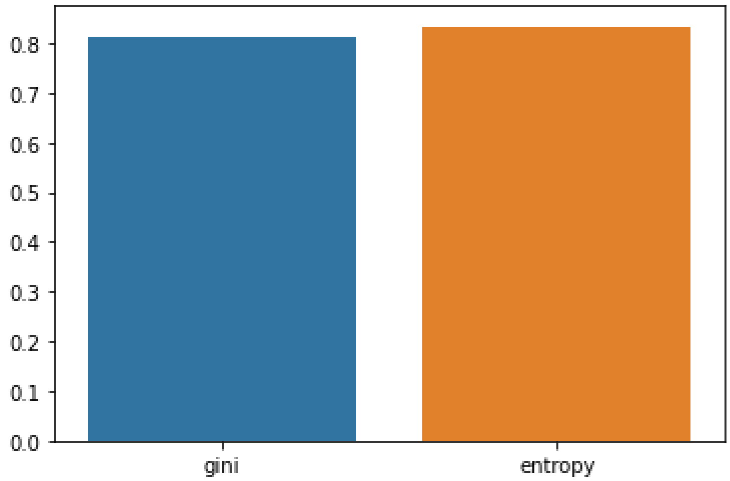
```
print('Support Vector Machine Accuracy Score on Training Set:', end = ' ')
print(accuracy_score(y_train, y_pred_svc_train)*100)
```

```
Support Vector Machine Accuracy Score on Test Set: 77.77777777777779
Support Vector Machine Accuracy Score on Training Set: 90.74074074074075
```

Random Forest

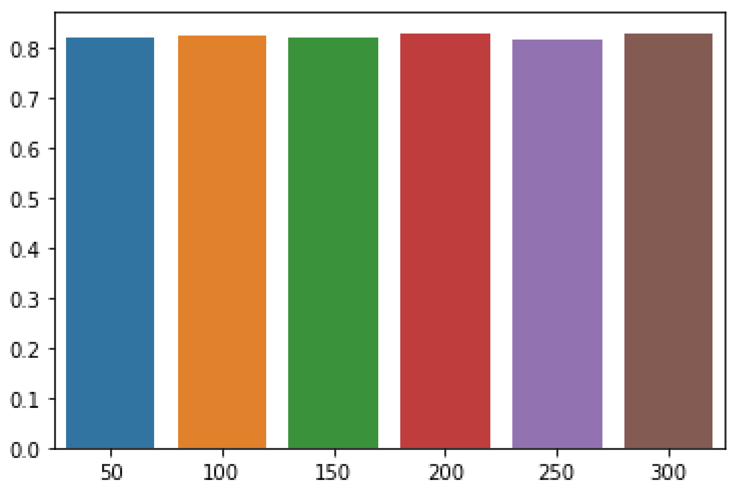
In [37]:
```
train_scores, valid_scores = validation_curve(RandomForestClassifier(), X, y, '
sns.barplot(x = ['gini','entropy'], y = [i.mean() for i in valid_scores])
```

Out[37]: <AxesSubplot:>



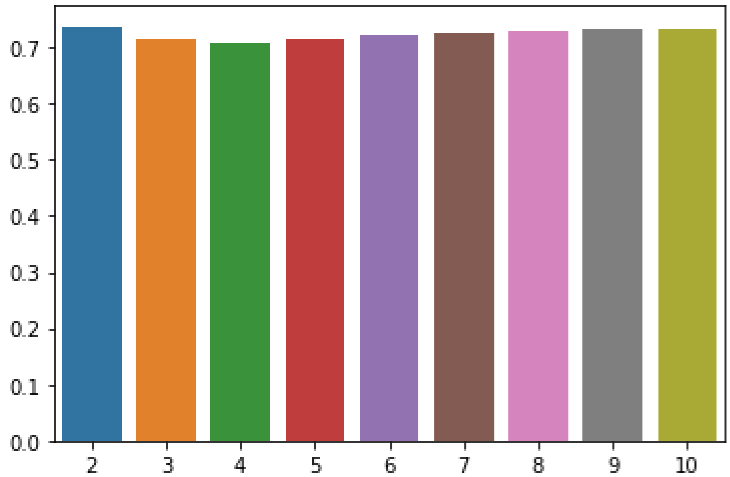

In [38]:
```
train_scores, valid_scores = validation_curve(RandomForestClassifier(), X, y, '
sns.barplot(x = [50, 100,150,200,250,300], y = [i.mean() for i in valid_scores
```

Out[38]: <AxesSubplot:>




In [39]:
```
train_scores, valid_scores = validation_curve(DecisionTreeClassifier(), X, y, '
sns.barplot(x = [2,3,4,5,6,7,8,9,10], y = [i.mean() for i in valid_scores])
```

Out[39]: <AxesSubplot:>




In [40]:
```
classifier_rf = RandomForestClassifier(n_estimators = 200, criterion = 'entropy
classifier_rf.fit(X_train, y_train)
y_pred_rf = classifier_rf.predict(X_test)
y_pred_rf_train = classifier_rf.predict(X_train)
print('Random Forest on Accuracy Score Test Set:', end = ' ')
print(accuracy_score(y_test, y_pred_rf)*100)
print('Random Forest on Accuracy Score Training Set:', end = ' ')
print(accuracy_score(y_train, y_pred_svc_train)*100)
```

```
Random Forest on Accuracy Score Test Set: 79.62962962962963
Random Forest on Accuracy Score Training Set: 90.74074074074075
```

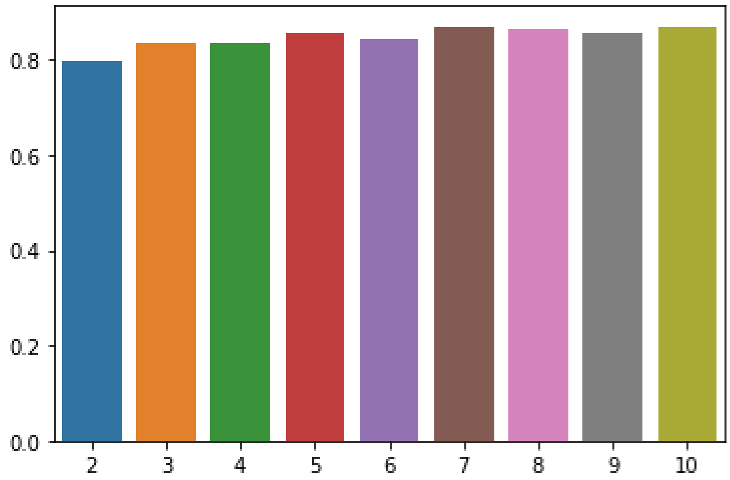
In [41]:
```
classifier_rf.feature_importances_
```

Out[41]:
```
array([7.10225725e-02, 4.97848404e-02, 6.54298640e-02, 8.56730872e-02,
       1.23401595e-01, 2.67546522e-02, 5.07173984e-03, 5.05155321e-02,
       7.64424464e-03, 1.10856443e-02, 1.40668029e-02, 1.26543404e-01,
       9.02210180e-03, 8.39698617e-05, 9.31222733e-03, 3.92395508e-02,
       2.44214908e-02, 2.23234037e-03, 6.84388914e-02, 4.68878025e-03,
       5.84367663e-02, 1.10837432e-01, 1.25712134e-02, 1.38455852e-02,
       9.87567164e-03])
```

K Nearest Neighbor

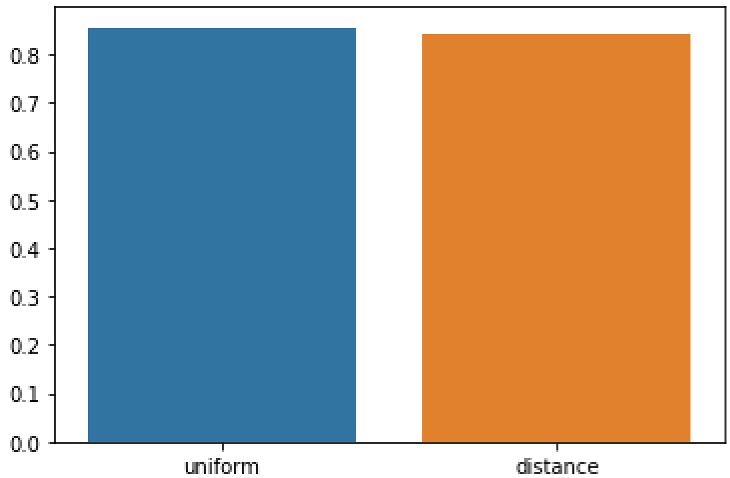
In [42]:
```
train_scores, valid_scores = validation_curve(KNeighborsClassifier(), X, y, "n_
sns.barplot(x = [2,3,4,5,6,7,8,9,10], y = [i.mean() for i in valid_scores])
```

Out[42]:
```
<AxesSubplot:>
```



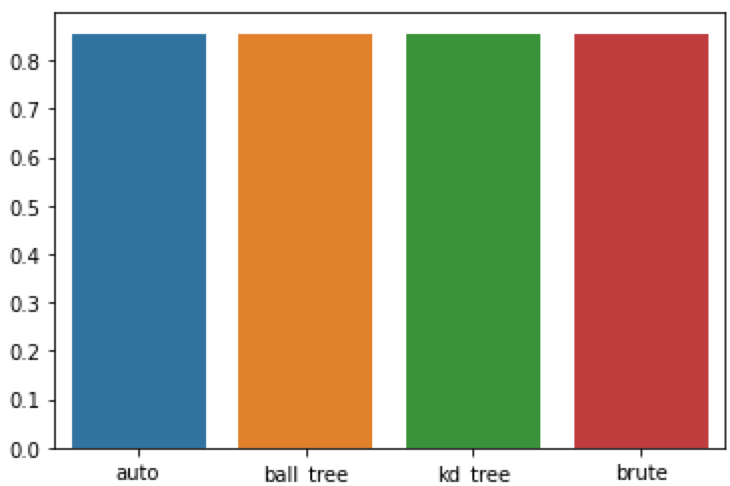

In [43]:
```
train_scores, valid_scores = validation_curve(KNeighborsClassifier(), X, y, "we
sns.barplot(x = ['uniform', 'distance'], y = [i.mean() for i in valid_scores])
```

Out[43]: <AxesSubplot:>




In [44]:
```
train_scores, valid_scores = validation_curve(KNeighborsClassifier(), X, y, "a
sns.barplot(x = ['auto', 'ball_tree', 'kd_tree', 'brute'], y = [i.mean() for i
```

Out[44]: <AxesSubplot:>



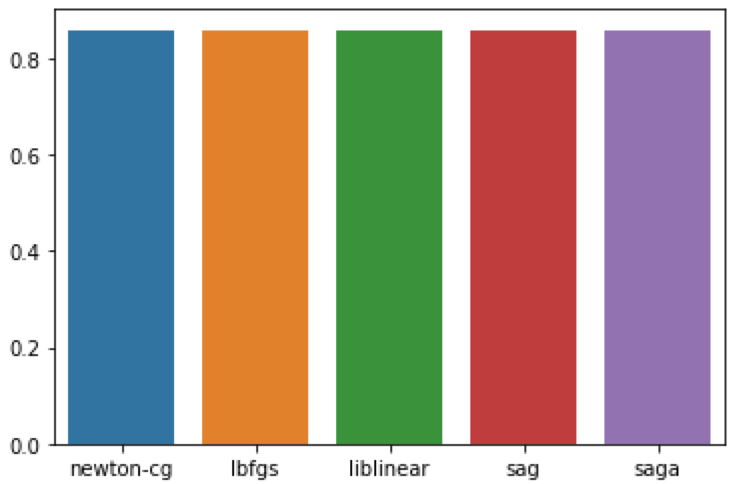
In [45]:
```
#K Nearest Neighbor
classifier_knn = KNeighborsClassifier(n_neighbors=9, weights = 'uniform')
classifier_knn.fit(X_train, y_train)
y_pred_knn = classifier_knn.predict(X_test)
y_pred_knn_train = classifier_knn.predict(X_train)
print('K Nearest Neighbor Accuracy Score on Test Set:', end = ' ')
print(accuracy_score(y_test, y_pred_knn)*100)
print('K Nearest Neighbor Accuracy Score on Training Set:', end = ' ')
print(accuracy_score(y_train, y_pred_knn_train)*100)
```

```
K Nearest Neighbor Accuracy Score on Test Set: 79.62962962962963
K Nearest Neighbor Accuracy Score on Training Set: 87.96296296296296
```

Logistic Regression

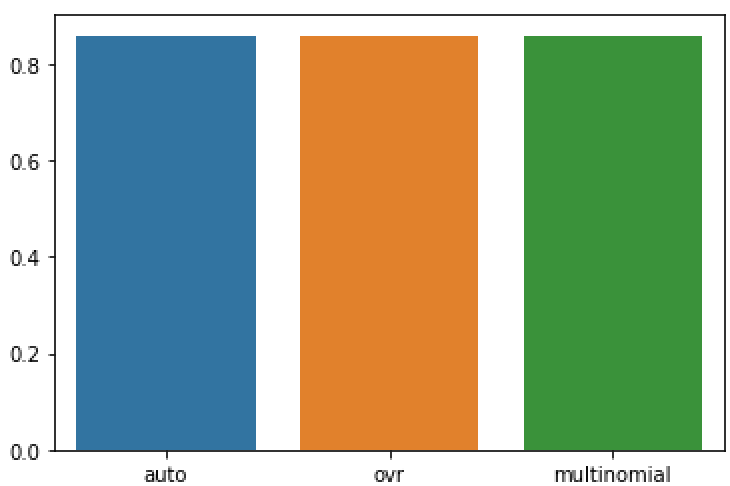
In [46]:
```
train_scores, valid_scores = validation_curve(LogisticRegression(), X, y, "solv
sns.barplot(x = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], y = [i.mear
```

Out[46]: <AxesSubplot:>



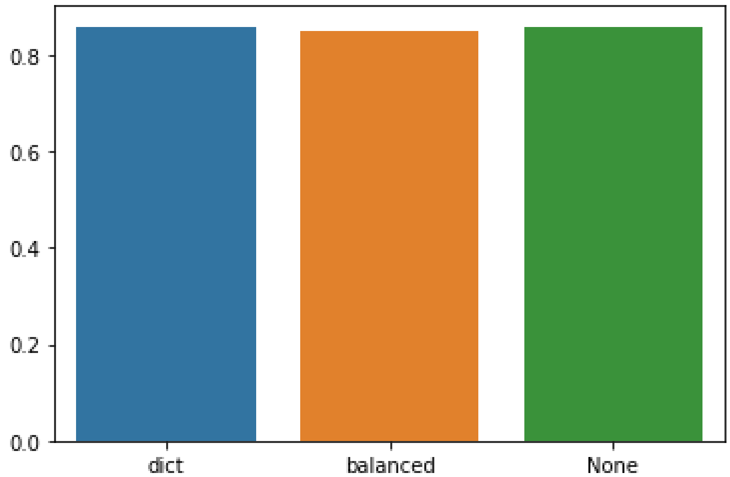

In [47]:
```
train_scores, valid_scores = validation_curve(LogisticRegression(), X, y, "mult
sns.barplot(x = ['auto', 'ovr', 'multinomial'], y = [i.mean() for i in valid_s
```

Out[47]: <AxesSubplot:>




In [48]:
```
train_scores, valid_scores = validation_curve(LogisticRegression(), X, y, "clas
sns.barplot(x = ['dict', 'balanced', 'None'], y = [i.mean() for i in valid_sco
```

Out[48]: <AxesSubplot:>




In [49]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression(random_state = 0)
classifier_lr.fit(X_train, y_train)
y_pred_lr = classifier_lr.predict(X_test)
y_pred_lr_train = classifier_lr.predict(X_train)
print('K Nearest Neighbor Accuracy Score on Test Set:', end = ' ')
print(accuracy_score(y_test, y_pred_lr)*100)
print('K Nearest Neighbor Accuracy Score on Training Set:', end = ' ')
print(accuracy_score(y_train, y_pred_lr_train)*100)
```

```
K Nearest Neighbor Accuracy Score on Test Set: 81.48148148148148
K Nearest Neighbor Accuracy Score on Training Set: 90.27777777777779
```

For all the models above, we are checking both Test data accuracy and Training data accuracy. If the difference between testing data and training data accuracy is drastic, we are overfitting our model. As you can see from the numbers above, the differnce training and testing data is less than 15% which means none of the model selected are overfitting the data.

# Model Robustness Testing/Sensitivity Check

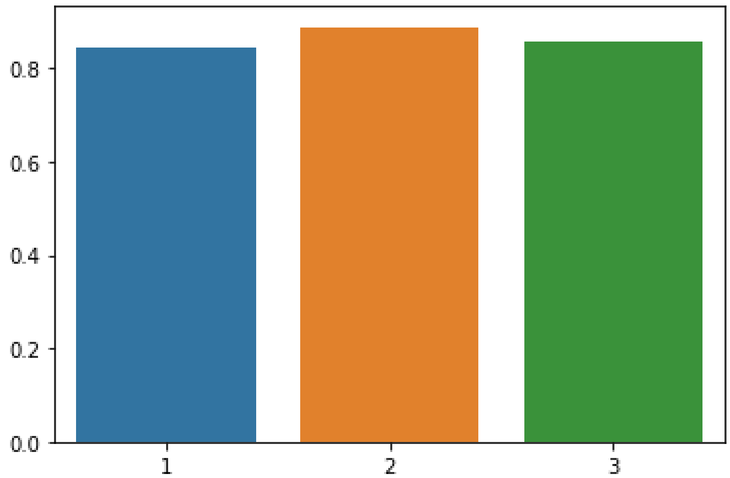
In [50]:
```
num_of_cross_val = 3
```

### Support Vector Machine

In [51]:
```
scores_svc = cross_val_score(classifier_svc, X, y, cv=num_of_cross_val, verbose
y_pred_svc = cross_val_predict(classifier_svc, X, y, cv=num_of_cross_val)
cm_svc = confusion_matrix(y, y_pred_svc)
print("\nCross Validation Scores:")
print("{:.1%} accuracy with a standard deviation of {:.1%}".format(scores_svc.r
print(" ")
print("Accuracy Percentage:", end = ' ')
print("{:.1%}".format(accuracy_score(y, y_pred_svc)))
sns.barplot(x = list(range(1,num_of_cross_val + 1)),y=scores_svc)
plt.show()
print("Confusion Matrix:")
print(cm_svc)
print(" ")
```

```
target_names = ['class 1', 'class 2']
print("Classification Report:")
print(classification_report(y, y_pred_svc, target_names=target_names))
```

```
Cross Validation Scores:
86.3% accuracy with a standard deviation of 1.9%

Accuracy Percentage: 86.3%
```

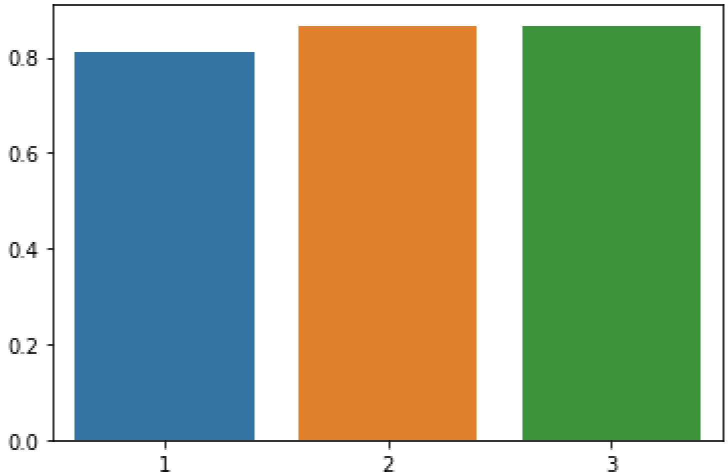



```
Confusion Matrix:
[[136  14]
 [ 23  97]]

Classification Report:
              precision    recall  f1-score   support

     class 1       0.86      0.91      0.88       150
     class 2       0.87      0.81      0.84       120

    accuracy                           0.86       270
   macro avg       0.86      0.86      0.86       270
weighted avg       0.86      0.86      0.86       270
```

### Random Forest

In [52]:
```
scores_rf = cross_val_score(classifier_rf, X, y, cv=num_of_cross_val)
y_pred_rf = cross_val_predict(classifier_rf, X, y, cv=num_of_cross_val)
cm_rf = confusion_matrix(y, y_pred_rf)
print("\nCross Validation Scores:")
print("{:.1%} accuracy with a standard deviation of {:.1%}".format(scores_rf.me
print(" ")
print("Accuracy Percentage:", end = ' ')
print("{:.1%}".format(accuracy_score(y, y_pred_rf)))
sns.barplot(x = list(range(1,num_of_cross_val + 1)),y=scores_rf)
plt.show()
print("Confusion Matrix:")
print(cm_rf)
print(" ")
target_names = ['class 1', 'class 2']
print("Classification Report:")
print(classification_report(y, y_pred_rf, target_names=target_names))
```

```
Cross Validation Scores:
84.8% accuracy with a standard deviation of 2.6%

Accuracy Percentage: 84.8%
```



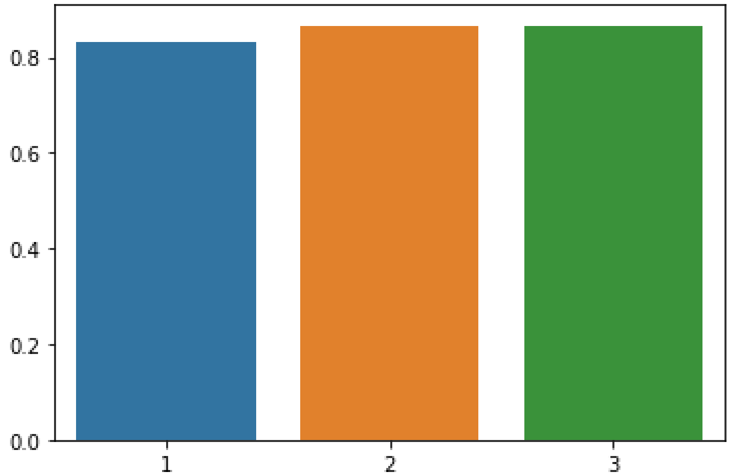

```
Confusion Matrix:
[[134  16]
 [ 25  95]]

Classification Report:
              precision    recall  f1-score   support

     class 1       0.84      0.89      0.87       150
     class 2       0.86      0.79      0.82       120

    accuracy                           0.85       270
   macro avg       0.85      0.84      0.84       270
weighted avg       0.85      0.85      0.85       270
```

### K Nearest Neighbor

In [53]:

```
scores_knn = cross_val_score(classifier_knn, X, y, cv=num_of_cross_val)
y_pred_knn = cross_val_predict(classifier_knn, X, y, cv=num_of_cross_val)
cm_knn = confusion_matrix(y, y_pred_knn)
print("\nCross Validation Scores:")
print("{:.1%} accuracy with a standard deviation of {:.1%}".format(scores_knn.r
print(" ")
print("Accuracy Percentage:", end = ' ')
print("{:.1%}".format(accuracy_score(y, y_pred_knn)))
sns.barplot(x = list(range(1,num_of_cross_val + 1)),y=scores_knn)
plt.show()
print("Confusion Matrix:")
print(cm_knn)
print(" ")
target_names = ['class 1', 'class 2']
print("Classification Report:")
print(classification_report(y, y_pred_knn, target_names=target_names))
```

```
Cross Validation Scores:
85.6% accuracy with a standard deviation of 1.6%

Accuracy Percentage: 85.6%
```

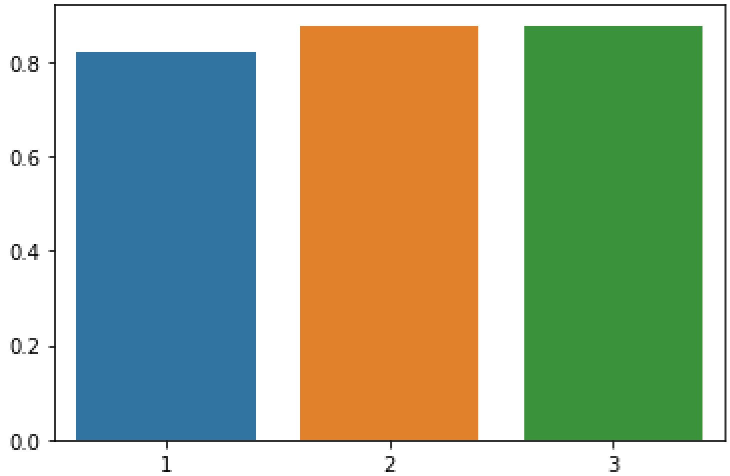



```
Confusion Matrix:
[[134  16]
 [ 23  97]]

Classification Report:
              precision    recall  f1-score   support

     class 1       0.85      0.89      0.87       150
     class 2       0.86      0.81      0.83       120

    accuracy                           0.86       270
   macro avg       0.86      0.85      0.85       270
weighted avg       0.86      0.86      0.86       270
```

### Logistic Regression

In [54]:
```
scores_lr = cross_val_score(classifier_lr, X, y, cv=num_of_cross_val)
y_pred_lr = cross_val_predict(classifier_lr, X, y, cv=num_of_cross_val)
cm_lr = confusion_matrix(y, y_pred_lr)
print("\nCross Validation Scores:")
print("{:.1%} accuracy with a standard deviation of {:.1%}".format(scores_lr.me
print(" ")
print("Accuracy Percentage:", end = ' ')
print("{:.1%}".format(accuracy_score(y, y_pred_lr)))
sns.barplot(x = list(range(1,num_of_cross_val + 1)),y=scores_lr)
plt.show()
print("Confusion Matrix:")
print(cm_lr)
print(" ")
target_names = ['class 1', 'class 2']
print("Classification Report:")
print(classification_report(y, y_pred_lr, target_names=target_names))
```

```
Cross Validation Scores:
85.9% accuracy with a standard deviation of 2.6%

Accuracy Percentage: 85.9%
```




```
Confusion Matrix:
[[135  15]
 [ 23  97]]

Classification Report:
              precision    recall  f1-score   support

     class 1       0.85      0.90      0.88       150
     class 2       0.87      0.81      0.84       120

    accuracy                           0.86       270
   macro avg       0.86      0.85      0.86       270
weighted avg       0.86      0.86      0.86       270
```

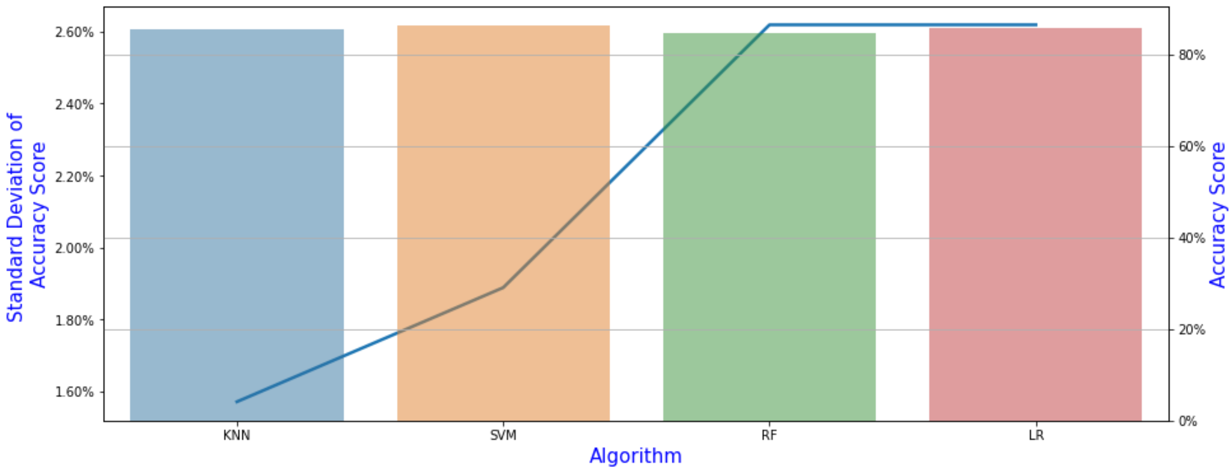
# Model Evaluation

In [55]:

```
models = ["KNN", "SVM", "RF", "LR"]
accuracy =[scores_knn.mean(), scores_svc.mean(), scores_rf.mean(), scores_lr.me
std = [scores_knn.std(),scores_svc.std(), scores_rf.std(),scores_lr.std()]
```

In [56]:

```
fig, ax = plt.subplots(1, 1, figsize=(15, 6))

sns.lineplot(x=models, y=std, ax=ax, linewidth = 2.5)
ax.yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1))
ax2 = ax.twinx()
sns.barplot(x=models, y=accuracy,  ax=ax2, alpha=0.5,order = ['KNN', 'SVM', 'RF
ax.yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1))
ax2.yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1))

ax.set_xlabel('Algorithm', color = 'blue', size = 15)
ax.set_ylabel("Standard Deviation of \nAccuracy Score",color = 'blue', size = 1
ax2.set_ylabel("Accuracy Score",color = 'blue', size = 15)
plt.grid()
plt.show()
```




Support Vector Machine model has the high accuracy and the second smallest standard deviation. K Nearest neighbor is the most robust model and would've been the best option if the accuracy was a couple percent higher.

# Model Selection and Finalization (Sensitivity and Accuracy Matrix)

In [57]:
```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

In [58]:
```
X = pd.get_dummies(X, prefix = ['cp','restecg','slope','thal','ca'], columns =
```

In [107…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.333, ra
```

In [59]:
```
#Removing the categorical data which has binary values now and preparing the da
X_0 = X.iloc[:,[0,2,3,5,7]].values
X_1 = X.iloc[:,[1,4,6,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]].values
```

In [60]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
X_0 = sc.fit_transform(X_0)

X = np.hstack((X_0, X_1))
```

In [61]:
```
classifier = SVC(kernel = 'linear', verbose = 0, random_state = 0)
classifier.fit(X_train, y_train)
```

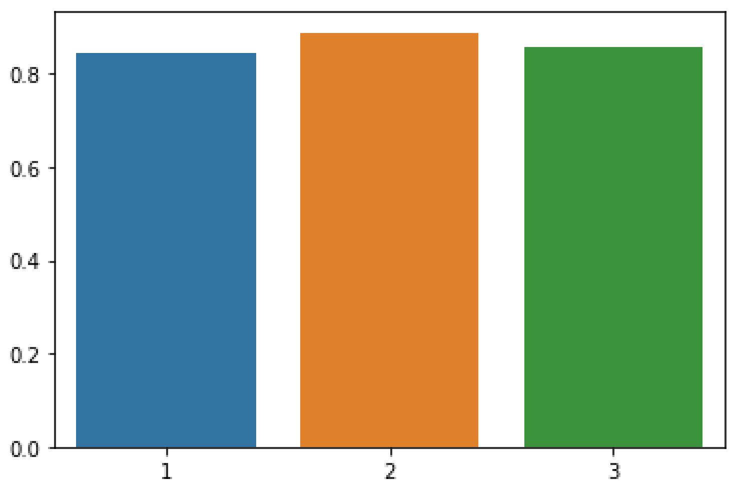
Out[61]:
```
SVC(kernel='linear', random_state=0, verbose=0)
```

In [62]:
```
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print('Accuary Score for Support Vector Machine based on Train-Test Split:', en
print (accuracy_score(y_test, y_pred)*100)
```

```
Accuary Score for Support Vector Machine based on Train-Test Split: 77.7777777
7777779
```

In [63]:
```
scores = cross_val_score(classifier, X, y, cv=num_of_cross_val)
y_pred = cross_val_predict(classifier, X, y, cv=num_of_cross_val)

cm = confusion_matrix(y, y_pred)
recall = recall_score(y, y_pred, average = 'micro')
print("Cross Validation Scores Based on 3 Fold Cross Validation (67%-33% Split
print("{:.1%} accuracy with a standard deviation of {:.1%}".format(scores.mean
print(" ")
print("Recall Score:", end = ' ')
print("{:.1%}".format(recall))
print(" ")
print("Accuracy Percentage:", end = ' ')
print("{:.1%}".format(accuracy_score(y, y_pred)))
sns.barplot(x = list(range(1,num_of_cross_val + 1)),y=scores)
plt.show()
print(" ")
print("Confusion Matrix:")
print(cm)
```

```
Cross Validation Scores Based on 3 Fold Cross Validation (67%-33% Split):
86.3% accuracy with a standard deviation of 1.9%

Recall Score: 86.3%

Accuracy Percentage: 86.3%
```




```
Confusion Matrix:
[[136  14]
 [ 23  97]]
```

Support Vector Machine model is accurate and robust. It only misclassified 37 out 270 instances.