

Gridsearching a Decision Tree & Random Forest, to find best hyperparameter values

The purpose of this notebook is to discover the best type of tree-based model and the ideal hyperparameters a long with it. The general strategy is to iterate through a series of gridsearches that become increasingly narrow and continue iterating until F1-score begins to decrease instead of increase.

As mentioned in the [previous notebook \(./resampling.ipynb\)](#), I am using the F1-score here as my primary performance metric because it is more important to correctly classify the target class as much as possible than it is to maximize true predictions across both classes of the target feature (i.e. accuracy), and it is not necessarily a liability to the business to have a occasional missclassification of the non-target class. Furthermore, there is not an argument obvious to me to prioritize either precision or recall over the other. Therefore, I rely on F1-score to evaluate and rank the models in this notebook.

Import dependencies, load data

```
In [5]: 1 from preprocessor import data_cleaner, data_sampler
2 from sklearn.model_selection import cross_val_score, GridSearchCV
3 from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
4 from sklearn.metrics import plot_roc_curve, plot_confusion_matrix, roc_curve
5
6 from imblearn.over_sampling import SMOTE
7
8 import pandas as pd
9 import numpy as np
10
11 import seaborn as sns
12 import matplotlib.pyplot as plt
13 %matplotlib inline
14
15 sns.set_style('darkgrid', {'axes.facecolor': '0.9'})
16
```

```
In [6]: 1 # data cleaning and OHE
2 X_train, y_train = data_cleaner("../data/train.csv.zip")
3 X_test, y_test = data_cleaner("../data/test.csv.zip")
4
```

```
In [7]: 1 # resample training data
2 smote = SMOTE()
3 X_train, y_train = smote.fit_resample(X_train, y_train)
```

Decision Tree

```
In [8]: 1 from sklearn.tree import DecisionTreeClassifier, plot_tree
```

First gridsearch

My best guess at a range of values to test

```
In [9]: 1 dtc_grid = DecisionTreeClassifier()
2
3 # define parameter grid to search
4 grid = [
5     {'criterion': ['gini', 'entropy', 'log_loss'],
6      'splitter': ['best', 'random'],
7      'max_depth': [2, 5, 10],
8      'min_samples_split': [2, 5, 10],
9      'min_samples_leaf': [1, 5, 10],
10     'max_features': [None, 'auto'],
11     'max_leaf_nodes': [10, 50, 100]}
12 ]
13
14 gridsearch = GridSearchCV(estimator=dtc_grid, param_grid=grid, scoring='f1')
15 gridsearch.fit(X_train, y_train)
16
17 grid_pred = gridsearch.predict(X_test)
18 grid_report = classification_report(y_test, grid_pred, output_dict=True)
19 grid_report = pd.DataFrame(grid_report).iloc[:, 0:3]
20 grid_report
```

```
Out[9]:
```

	0.0	1.0	accuracy
precision	0.984225	0.778589	0.938674
recall	0.939840	0.933528	0.938674
f1-score	0.961521	0.849048	0.938674
support	21177.000000	4799.000000	0.938674

```
In [10]: 1 gridsearch.best_params_
```

```
Out[10]: {'criterion': 'gini',
'max_depth': 10,
'max_features': None,
'max_leaf_nodes': 100,
'min_samples_leaf': 1,
'min_samples_split': 2,
'splitter': 'best'}
```

second gridsearch

The values with the greatest magnitude were chosen for Max_depth, max_features, max_leaf_nodes, min_samples_leaf. So the next gridsearch will include these chosen values as the low end of the range that is being searched.

```
In [11]: 1 grid_two = [
2         {'criterion': ['gini'],
3          'max_depth': [10,25,50],
4          'max_features': ['log2',10,20],
5          'max_leaf_nodes': [100,150,200],
6          'min_samples_leaf': [10,25,50],
7          'min_samples_split': [2],
8          'splitter': ['best']}
9     ]
10
11 gridsearch_two = GridSearchCV(estimator=dtc_grid,param_grid=grid_two,sco
12 gridsearch_two.fit(X_train,y_train)
13
```

```
Out[11]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=3,
                  param_grid=[{'criterion': ['gini'], 'max_depth': [10, 25, 50],
                                'max_features': ['log2', 10, 20],
                                'max_leaf_nodes': [100, 150, 200],
                                'min_samples_leaf': [10, 25, 50],
                                'min_samples_split': [2], 'splitter': ['best']}],
                  scoring='f1')
```

```
In [12]: 1
2 grid_two_pred = gridsearch_two.predict(X_test)
3 grid_two_report = classification_report(y_test,grid_two_pred,output_dict=
4 grid_two_report = pd.DataFrame(grid_two_report).iloc[:,0:3]
5 grid_two_report
```

```
Out[12]:
```

	0.0	1.0	accuracy
precision	0.981606	0.865686	0.958847
recall	0.967654	0.919983	0.958847
f1-score	0.974580	0.892009	0.958847
support	21177.000000	4799.000000	0.958847

```
In [13]: 1 gridsearch_two.best_params_
```

```
Out[13]: {'criterion': 'gini',
          'max_depth': 25,
          'max_features': 20,
          'max_leaf_nodes': 200,
          'min_samples_leaf': 10,
          'min_samples_split': 2,
          'splitter': 'best'}
```

third gridsearch

Max_depth, max_leaf_nodes, continue selecting the greatest value. Max_features is still unclear.

```
In [14]: 1 grid_three = [
2         {'criterion': ['gini'],
3          'max_depth': [50,100,200],
4          'max_features': [15,20,23],
5          'max_leaf_nodes': [200,300,500],
6          'min_samples_leaf': [10],
7          'min_samples_split': [2],
8          'splitter': ['best']}
9         ]
10
11 gridsearch_three = GridSearchCV(estimator=dtc_grid,param_grid=grid_three,
12 gridsearch_three.fit(X_train,y_train)
13
14 grid_three_pred = gridsearch_three.predict(X_test)
15 grid_three_report = classification_report(y_test,grid_three_pred,output_c
16 grid_three_report = pd.DataFrame(grid_three_report).iloc[:,0:3]
17 grid_three_report
```

```
Out[14]:
```

	0.0	1.0	accuracy
precision	0.984336	0.910554	0.970396
recall	0.979270	0.931236	0.970396
f1-score	0.981797	0.920779	0.970396
support	21177.000000	4799.000000	0.970396

```
In [15]: 1 gridsearch_three.best_params_
```

```
Out[15]: {'criterion': 'gini',
'max_depth': 50,
'max_features': 20,
'max_leaf_nodes': 500,
'min_samples_leaf': 10,
'min_samples_split': 2,
'splitter': 'best'}
```

fourth gridsearch

Everything seems to be getting zeroed in on except for max_leaf_nodes.

```

In [16]: 1 grid_four = [
2         {'criterion': ['gini'],
3          'max_depth': [75,100,125],
4          'max_features': [18,19,20],
5          'max_leaf_nodes': [500,750,1000],
6          'min_samples_leaf': [5,10,15],
7          'min_samples_split': [2],
8          'splitter': ['best']}
9     ]
10
11 gridsearch_four = GridSearchCV(estimator=dtc_grid,param_grid=grid_four,
12 gridsearch_four.fit(X_train,y_train)
13
14 grid_four_pred = gridsearch_four.predict(X_test)
15 grid_four_report = classification_report(y_test,grid_four_pred,output_dic
16 grid_four_report = pd.DataFrame(grid_four_report).iloc[:,0:3]
17 grid_four_report

```

```

Out[16]:

```

	0.0	1.0	accuracy
precision	0.985142	0.913646	0.971628
recall	0.979978	0.934778	0.971628
f1-score	0.982553	0.924091	0.971628
support	21177.000000	4799.000000	0.971628

```

In [17]: 1 gridsearch_four.best_params_

```

```

Out[17]: {'criterion': 'gini',
          'max_depth': 75,
          'max_features': 19,
          'max_leaf_nodes': 1000,
          'min_samples_leaf': 5,
          'min_samples_split': 2,
          'splitter': 'best'}

```

fifth gridsearch

```

In [18]: 1 grid_five = [
2         {'criterion': ['gini'],
3          'max_depth': [115,125,150,175],
4          'max_features': [19],
5          'max_leaf_nodes': [600,750,900],
6          'min_samples_leaf': [5],
7          'min_samples_split': [2],
8          'splitter': ['best']}
9         ]
10
11 gridsearch_five = GridSearchCV(estimator=dtc_grid,param_grid=grid_five,s
12 gridsearch_five.fit(X_train,y_train)
13
14 grid_five_pred = gridsearch_five.predict(X_test)
15 grid_five_report = classification_report(y_test,grid_five_pred,output_di
16 grid_five_report = pd.DataFrame(grid_five_report).iloc[:,0:3]
17 grid_five_report

```

Out[18]:

	0.0	1.0	accuracy
precision	0.985383	0.915596	0.972205
recall	0.980450	0.935820	0.972205
f1-score	0.982910	0.925598	0.972205
support	21177.000000	4799.000000	0.972205

```

In [19]: 1 gridsearch_five.best_params_

```

Out[19]: {'criterion': 'gini',
'max_depth': 115,
'max_features': 19,
'max_leaf_nodes': 750,
'min_samples_leaf': 5,
'min_samples_split': 2,
'splitter': 'best'}

sixth gridsearch

```

In [20]: 1 grid_six = [
2         {'criterion': ['gini'],
3          'max_depth': [125,135,150],
4          'max_features': [19],
5          'max_leaf_nodes': [550,600,650],
6          'min_samples_leaf': [5],
7          'min_samples_split': [2],
8          'splitter': ['best']}
9     ]
10
11 gridsearch_six = GridSearchCV(estimator=dtc_grid,param_grid=grid_six,score_func=accuracy)
12 gridsearch_six.fit(X_train,y_train)
13
14 grid_six_pred = gridsearch_six.predict(X_test)
15 grid_six_report = classification_report(y_test,grid_six_pred,output_dict=True)
16 grid_six_report = pd.DataFrame(grid_six_report).iloc[:,0:3]
17 grid_six_report

```

```

Out[20]:

```

	0.0	1.0	accuracy
precision	0.985733	0.921910	0.973745
recall	0.982009	0.937279	0.973745
f1-score	0.983867	0.929531	0.973745
support	21177.000000	4799.000000	0.973745

Observations:

F1, precision, and recall performance continue to decrease after the fourth gridsearch. So, the the best_params_ of gridsearch_four will be used as a start point to gridsearch a random forest and bagging tree.

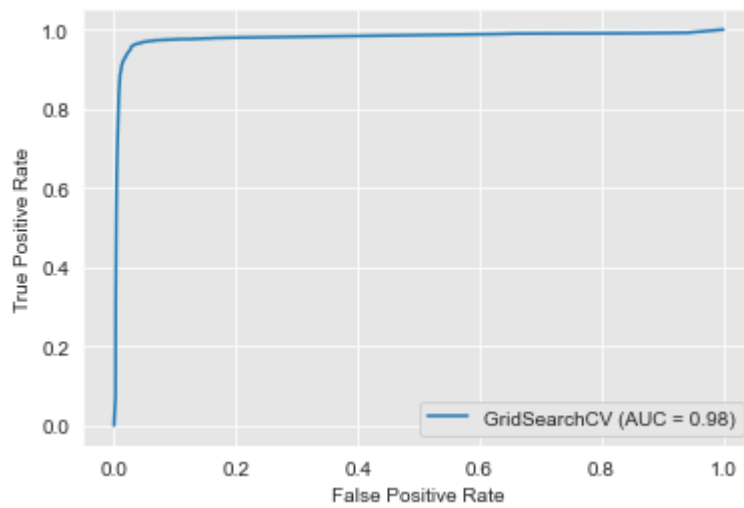
Gridsearch_four.best_params_ returns the following dict:

```

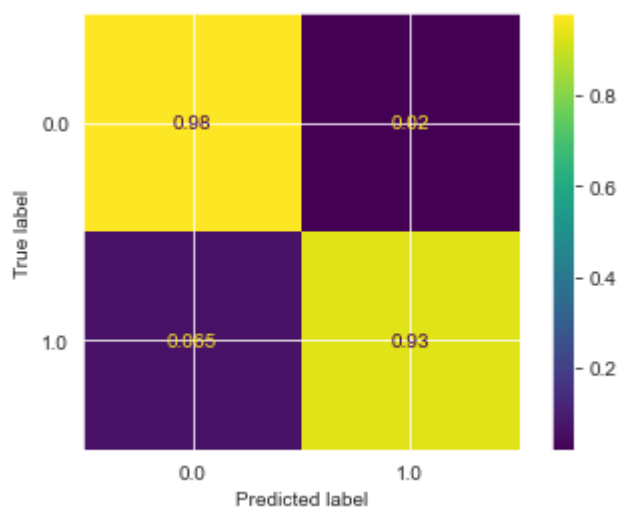
{'criterion': 'gini',
 'max_depth': 125,
 'max_features': 19,
 'max_leaf_nodes': 750,
 'min_samples_leaf': 5,
 'min_samples_split': 2,
 'splitter': 'best'}

```

```
In [21]: 1 plot_roc_curve(gridsearch_four,X_test,y_test)
2         plt.show()
```



```
In [22]: 1 plot_confusion_matrix(gridsearch_four,X_test,y_test,normalize='true')
2         plt.show()
```



BaggingClassifier

baseline model (default bagging parameters)

```
In [23]: 1 bagging_tree = DecisionTreeClassifier(criterion='gini',
2                                             max_depth=125,
3                                             max_features=19,
4                                             max_leaf_nodes=750,
5                                             min_samples_leaf=5,
6                                             min_samples_split=2,
7                                             splitter='best')
8
9 tree_bagger = BaggingClassifier(base_estimator=bagging_tree)
10 tree_bagger.fit(X_train,y_train)
11 tree_bagger_pred = tree_bagger.predict(X_test)
12 tree_bagger_report = classification_report(y_test,tree_bagger_pred,output
13 tree_bagger_report = pd.DataFrame(tree_bagger_report).iloc[:,0:3]
14 tree_bagger_report
15
```

Out[23]:

	0.0	1.0	accuracy
precision	0.988384	0.932242	0.977826
recall	0.984370	0.948948	0.977826
f1-score	0.986373	0.940520	0.977826
support	21177.000000	4799.000000	0.977826

Gridsearch bagging classifier

First gridsearch

```

In [24]: 1 from sklearn.ensemble import BaggingClassifier
2
3 bagging_tree = DecisionTreeClassifier(criterion='gini',
4                                     max_depth=125,
5                                     max_features=19,
6                                     max_leaf_nodes=750,
7                                     min_samples_leaf=5,
8                                     min_samples_split=2,
9                                     splitter='best')
10
11 tree_bagger = BaggingClassifier()
12
13 bag_grid = [{
14     'base_estimator':[bagging_tree],
15     'n_estimators':[5,10,15,20],
16     'max_samples':[1.0,3.0,5.0],
17     'max_features':[1.0,5.0,10.0],
18     'bootstrap':[True,False],
19     'bootstrap_features':[True,False],
20     'n_jobs':[3]
21 }]
22
23 bagged_grid = GridSearchCV(estimator=tree_bagger,param_grid=bag_grid,score_func=roc_auc_score)
24 bagged_grid.fit(X_train,y_train)
25 bagged_grid_pred = bagged_grid.predict(X_test)
26 bagged_grid_report = classification_report(y_test,bagged_grid_pred,output_dict=True)
27 bagged_grid_report = pd.DataFrame(bagged_grid_report).iloc[:,0:3]
28 bagged_grid_report
29

```

Out[24]:

	0.0	1.0	accuracy
precision	0.988602	0.943295	0.980174
recall	0.987061	0.949781	0.980174
f1-score	0.987831	0.946527	0.980174
support	21177.000000	4799.000000	0.980174

In [25]: 1 bagged_grid_report

Out[25]:

	0.0	1.0	accuracy
precision	0.988602	0.943295	0.980174
recall	0.987061	0.949781	0.980174
f1-score	0.987831	0.946527	0.980174
support	21177.000000	4799.000000	0.980174

In [26]: 1 bagged_grid.best_params_

Out[26]: {'base_estimator': DecisionTreeClassifier(max_depth=125, max_features=19, max_leaf_nodes=750, min_samples_leaf=5),
'bootstrap': False,
'bootstrap_features': True,
'max_features': 1.0,
'max_samples': 1.0,
'n_estimators': 15,
'n_jobs': 3}

Second gridsearch

-searching only n_estimators

In [27]: 1 second_bag_grid = [{
2 'base_estimator':[bagging_tree],
3 'n_estimators':[20,23,25,30],
4 'max_samples':[1.0],
5 'max_features':[1.0],
6 'bootstrap':[False],
7 'bootstrap_features':[True],
8 'n_jobs':[3]
9 }]
10
11 second_bagged_grid = GridSearchCV(estimator=tree_bagger,param_grid=second_bag_grid)
12 second_bagged_grid.fit(X_train,y_train)
13 second_bagged_grid_pred = second_bagged_grid.predict(X_test)
14 second_bagged_grid_report = classification_report(y_test,second_bagged_grid_pred)
15 second_bagged_grid_report = pd.DataFrame(second_bagged_grid_report).iloc[0]
16 second_bagged_grid_report

Out[27]:

	0.0	1.0	accuracy
precision	0.989982	0.933849	0.979366
recall	0.984653	0.956033	0.979366
f1-score	0.987311	0.944811	0.979366
support	21177.000000	4799.000000	0.979366

In [28]: 1 second_bagged_grid.best_params_

Out[28]: {'base_estimator': DecisionTreeClassifier(max_depth=125, max_features=19, max_leaf_nodes=750, min_samples_leaf=5),
'bootstrap': False,
'bootstrap_features': True,
'max_features': 1.0,
'max_samples': 1.0,
'n_estimators': 25,
'n_jobs': 3}

finding best n_estimators

Here I re-run the same cell(s) adjusting only n_estimators to find the value that improves F1 the most.

```
In [29]: 1 n_trees_bagger = BaggingClassifier(base_estimator=bagging_tree,
2                                           bootstrap=False,
3                                           bootstrap_features=True,
4                                           n_estimators=31,
5                                           n_jobs=3)
6
7 n_trees_bagger.fit(X_train,y_train)
8
9 n_trees_pred = n_trees_bagger.predict(X_test)
10 n_trees_report = classification_report(y_test,n_trees_pred,output_dict=True)
11 n_trees_report = pd.DataFrame(n_trees_report).iloc[:,0:3]
12 n_trees_report
```

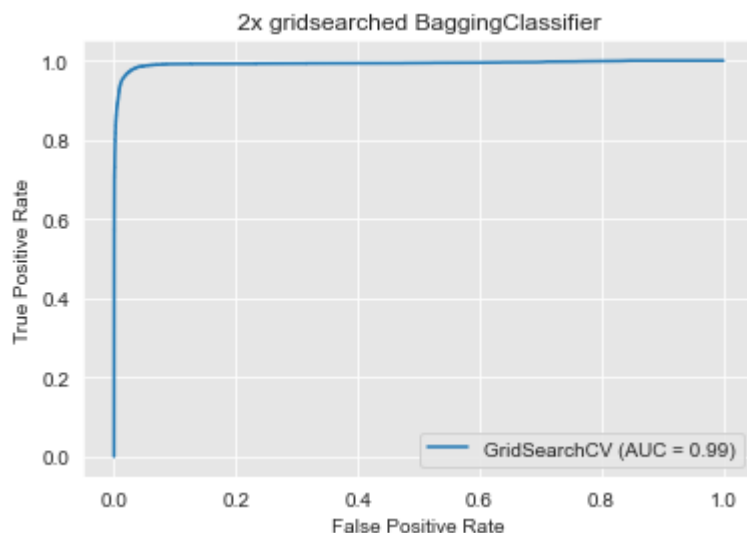
Out[29]:

	0.0	1.0	accuracy
precision	0.989921	0.947129	0.981945
recall	0.987911	0.955616	0.981945
f1-score	0.988915	0.951354	0.981945
support	21177.000000	4799.000000	0.981945

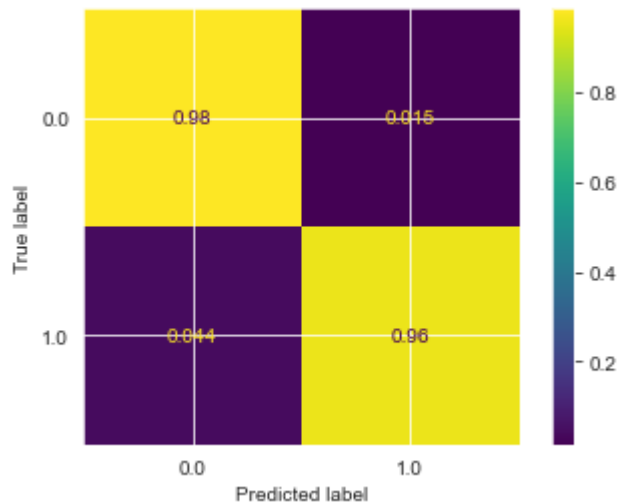
Observations:

The best performing Bagging classifier is the second grid searches best parameters

```
In [30]: 1 plot_roc_curve(second_bagged_grid,X_test,y_test)
2 plt.title('2x gridsearched BaggingClassifier')
3 plt.show()
```



```
In [31]: 1 plot_confusion_matrix(second_bagged_grid,X_test,y_test,normalize='true')
2 plt.show()
```



Random Forest

Baseline RF

```
In [32]: 1 RFC = RandomForestClassifier(n_jobs=3)
          2 RFC.fit(X_train,y_train)
          3
          4 rfc_pred = RFC.predict(X_test)
          5 rfc_report = classification_report(y_test,rfc_pred,output_dict='true')
          6 rfc_report = pd.DataFrame(rfc_report).iloc[:,0:3]
          7 rfc_report
```

```
Out[32]:
```

	0.0	1.0	accuracy
precision	0.989985	0.954035	0.983331
recall	0.989564	0.955824	0.983331
f1-score	0.989774	0.954929	0.983331
support	21177.000000	4799.000000	0.983331

Tuning RF hyperparameters

using the best_params_ from the decision tree model gridsearch_four

```
In [33]: 1 RFC_tuned = RandomForestClassifier(criterion='entropy',
          2                                     max_depth=125,
          3                                     max_features=20,
          4                                     max_leaf_nodes=750,
          5                                     min_samples_leaf=5,
          6                                     min_samples_split=2)
          7 RFC_tuned.fit(X_train,y_train)
          8
          9 rfc_tuned_pred = RFC_tuned.predict(X_test)
          10
          11 rfc_tuned_report = classification_report(y_test,rfc_tuned_pred,output_dict='true')
          12 rfc_tuned_report = pd.DataFrame(rfc_tuned_report).iloc[:,0:3]
          13 rfc_tuned_report
```

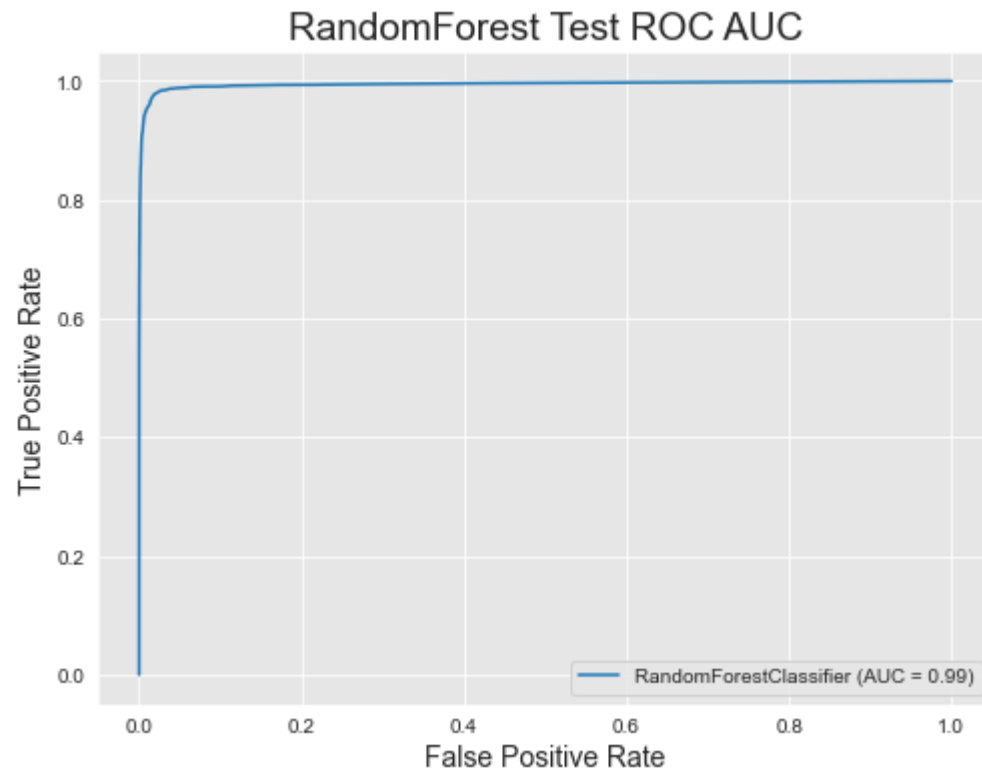
```
Out[33]:
```

	0.0	1.0	accuracy
precision	0.989164	0.943630	0.980674
recall	0.987109	0.952282	0.980674
f1-score	0.988135	0.947936	0.980674
support	21177.000000	4799.000000	0.980674

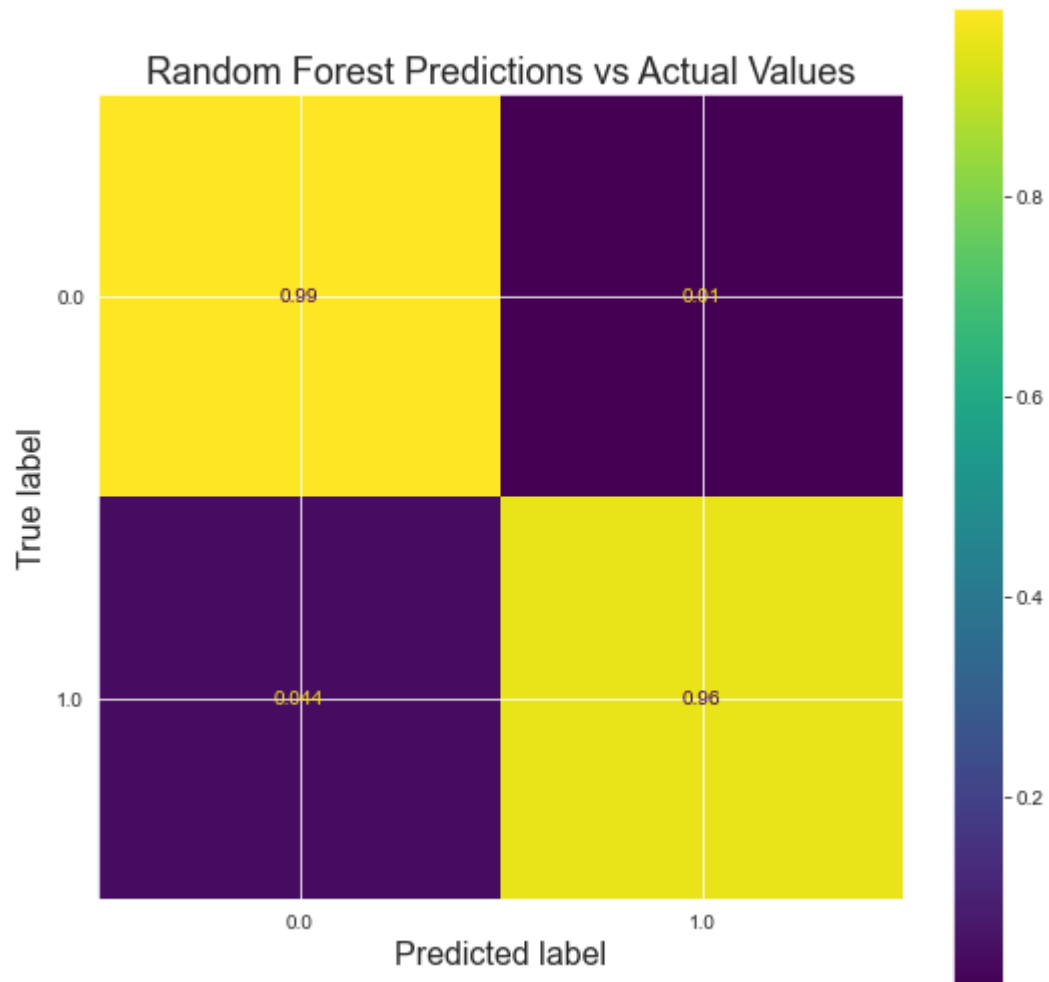
Observations:

Random forest default settings consistently outperforms parameters used from best decision tree.
Computational demands are too great to perform a gridsearch

```
In [34]: 1 # baseline Random Forest ROC AUC
2 fig, ax = plt.subplots(figsize=(8, 6))
3 plot_roc_curve(RFC,X_test,y_test,ax=ax)
4 ax.set_title("RandomForest Test ROC AUC",fontsize=19)
5 ax.set_ylabel('True Positive Rate',fontsize=14)
6 ax.set_xlabel('False Positive Rate',fontsize=14)
7 plt.show()
```



```
In [35]: 1 # baseline Random Forest confusion matrix
2 fig, ax = plt.subplots(figsize=(9, 9))
3 plot_confusion_matrix(RFC,X_test,y_test,normalize='true',ax=ax)
4 ax.set_title("Random Forest Predictions vs Actual Values",fontsize=18)
5 ax.set_ylabel('True label',fontsize=16)
6 ax.set_xlabel('Predicted label',fontsize=16)
7 plt.show()
```



Iterative Performance Visualization

F1 Score

```
In [43]: 1 f1_dict = {  
2         'baseline\nDecisionTree':grid_report.iloc[2,1],  
3         '4x_gridsearched\nDecisionTree':grid_four_report.iloc[2,1],  
4         'baseline Bagging':tree_bagger_report.iloc[2,1],  
5         '2x_gridsearched\nBagging':second_bagged_grid_report.iloc[2,1],  
6         'baseline\nRandomForest':rfc_report.iloc[2,1],  
7         'tuned\nRandomForest':rfc_tuned_report.iloc[2,1]  
8     }  
9 f1_dict = dict(sorted(f1_dict.items(), key=lambda item: item[1],reverse=
```

```
In [53]: 1 plt.figure(figsize=(11,9))
2
3 plt.bar(x=f1_dict.keys(),height=f1_dict.values(),color='lightblue')
4
5 plt.title('Tree Based Model(s) Performance',fontsize=24)
6 plt.ylabel('F1 Score',fontsize=16)
7 plt.xlabel('Model version',fontsize=16)
8 plt.show()
```



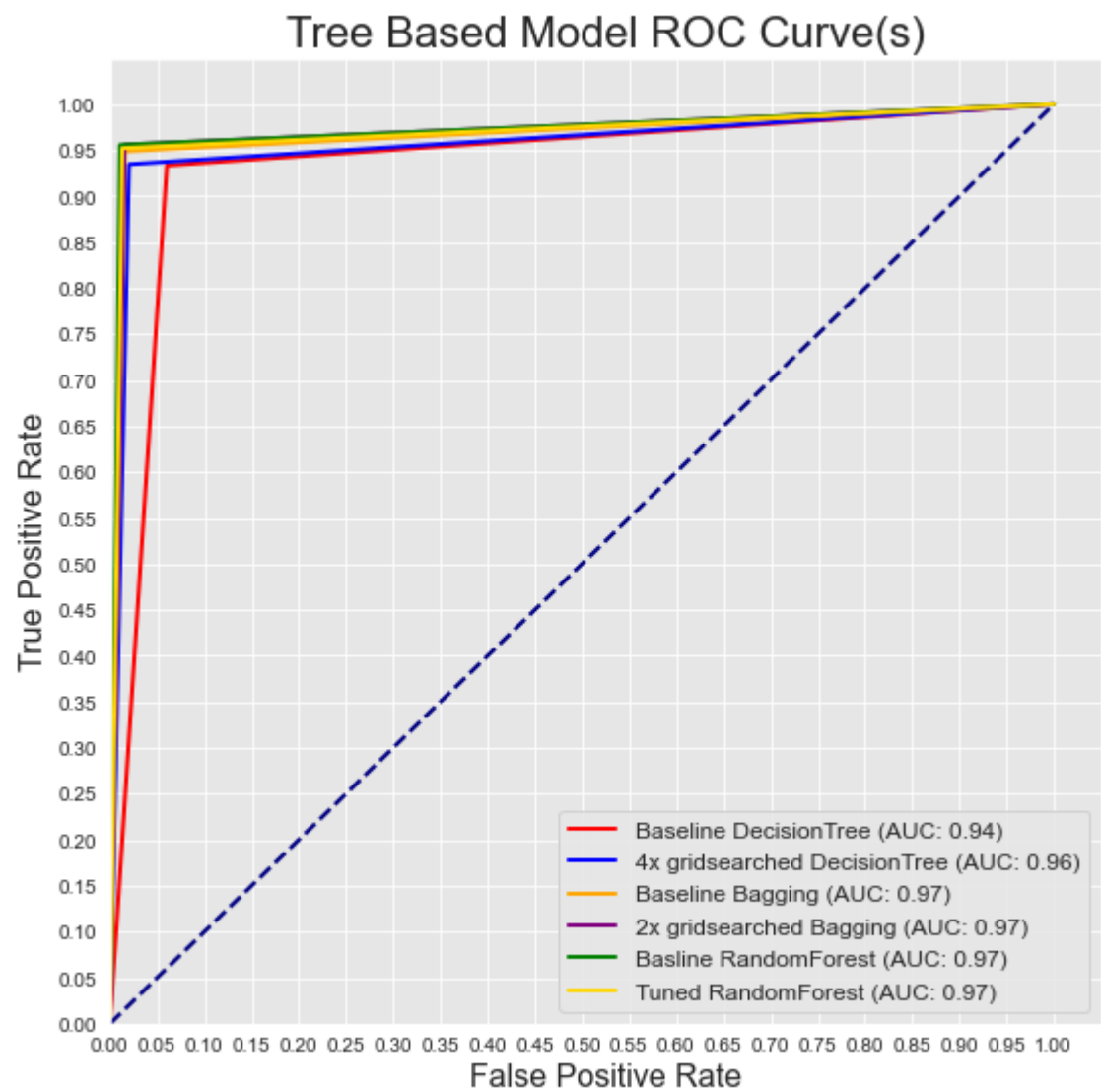
The difference is not massive, but the random forest with the default hyperparameters is the clear best performer, with an F1-score of 0.95. Precision and Recall (the basal components of F1) are also pretty good with both scoring 0.95+.

ROC AUC Curve

In [47]:

```
1 # calculate auc's
2 # first decision tree gridsearch
3 base_gridsearch_fpr , base_gridsearch_tpr, base_gridsearch_thresholds = roc_curve(y_test, base_model.predict_proba(X_test))
4 base_gridsearch_auc = auc(base_gridsearch_fpr, base_gridsearch_tpr)
5
6 # best (fourth) decision tree gridsearch
7 gridsearch_four_fpr , gridsearch_four_tpr, gridsearch_four_thresholds = roc_curve(y_test, gridsearch_four_model.predict_proba(X_test))
8 gridsearch_four_auc = auc(gridsearch_four_fpr, gridsearch_four_tpr)
9
10 # baseline bagging
11 base_bagging_fpr , base_bagging_tpr, base_bagging_thresholds = roc_curve(y_test, base_model.predict_proba(X_test))
12 base_bagging_auc = auc(base_bagging_fpr, base_bagging_tpr)
13
14 # second gridsearched bagging
15 second_grid_bagging_fpr , second_grid_bagging_tpr, second_grid_bagging_thresholds = roc_curve(y_test, second_grid_bagging_model.predict_proba(X_test))
16 second_grid_bagging_auc = auc(second_grid_bagging_fpr, second_grid_bagging_tpr)
17
18 # baseline random forest
19 base_rfc_fpr , base_rfc_tpr, base_rfc_thresholds = roc_curve(y_test, base_model.predict_proba(X_test))
20 base_rfc_auc = auc(base_rfc_fpr, base_rfc_tpr)
21
22 # tuned random forest
23 tuned_rfc_fpr , tuned_rfc_tpr, tuned_rfc_thresholds = roc_curve(y_test, tuned_rfc_model.predict_proba(X_test))
24 tuned_rfc_auc = auc(tuned_rfc_fpr, tuned_rfc_tpr)
25
26 auc_list = [base_gridsearch_auc, gridsearch_four_auc, base_bagging_auc, second_grid_bagging_auc, base_rfc_auc, tuned_rfc_auc]
```

```
In [48]: ▶ 1 plt.figure(figsize=(9,9))
2 lw = 2
3
4 plt.plot(base_gridsearch_fpr, base_gridsearch_tpr, color='red',
5          lw=lw, label=f'Baseline DecisionTree (AUC: {round(base_gridsearch_auc,2)})')
6 plt.plot(gridsearch_four_fpr, gridsearch_four_tpr, color='blue',
7          lw=lw, label=f'4x gridsearched DecisionTree (AUC: {round(gridsearch_four_auc,2)})')
8 plt.plot(base_bagging_fpr, base_bagging_tpr, color='orange',
9          lw=lw, label=f'Baseline Bagging (AUC: {round(base_bagging_auc,2)})')
10 plt.plot(second_grid_bagging_fpr, second_grid_bagging_tpr, color='purple',
11          lw=lw, label=f'2x gridsearched Bagging (AUC: {round(second_grid_bagging_auc,2)})')
12 plt.plot(base_rfc_fpr, base_rfc_tpr, color='green',
13          lw=lw, label=f'Baseline RandomForest (AUC: {round(base_rfc_auc,2)})')
14 plt.plot(tuned_rfc_fpr, tuned_rfc_tpr, color='gold',
15          lw=lw, label=f'Tuned RandomForest (AUC: {round(tuned_rfc_auc,2)})')
16
17 # Formatting
18 plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
19 plt.xlim([0.0, 1.05])
20 plt.ylim([0.0, 1.05])
21 plt.yticks([i/20.0 for i in range(21)])
22 plt.xticks([i/20.0 for i in range(21)])
23 plt.xlabel('False Positive Rate', fontsize=16)
24 plt.ylabel('True Positive Rate', fontsize=16)
25 plt.title('Tree Based Model ROC Curve(s)', fontsize=22)
26 plt.legend(loc="lower right", prop={'size':12})
27
28 plt.show()
```

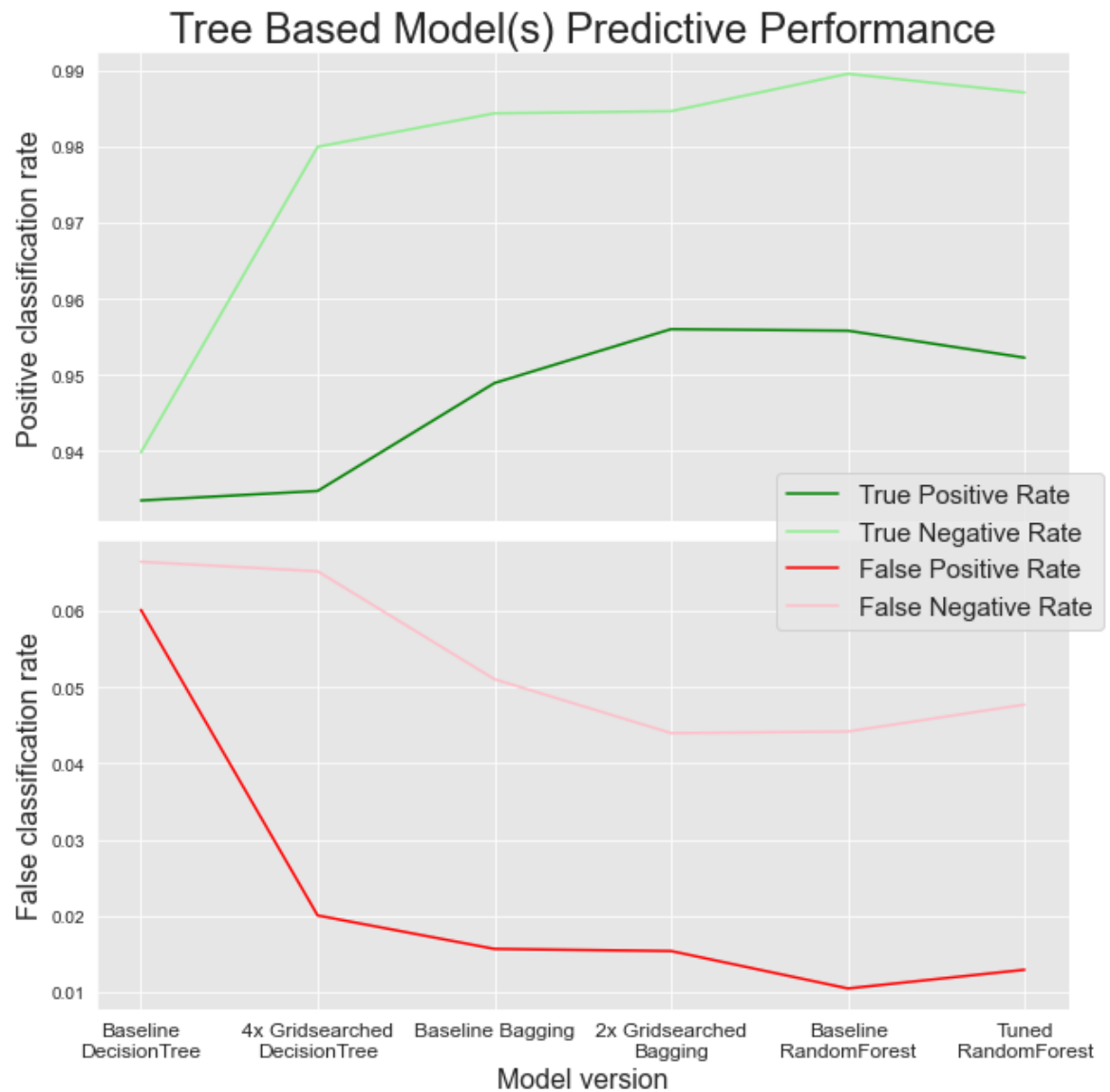


True Positive rate

```
In [68]: ▶ 1 prediction_labels = ['Baseline\nDecisionTree',
2                             '4x Gridsearched\nDecisionTree',
3                             'Baseline Bagging',
4                             '2x Gridsearched\nBagging',
5                             'Baseline\nRandomForest',
6                             'Tuned\nRandomForest']
7 predictions = [grid_pred,grid_four_pred,tree_bagger_pred,second_bagged_g
8
9
10
11 tprs = []
12 fprs = []
13
14 tnrs = []
15 fnrs = []
16
17 for pred in predictions:
18     matrix = confusion_matrix(y_test,pred,normalize='true')
19     tprs.append(matrix[1][1])
20     fprs.append(matrix[0][1])
21     tnrs.append(matrix[0][0])
22     fnrs.append(matrix[1][0])
23
```

```
In [69]: 1 fig,(ax1,ax2) = plt.subplots(nrows=2,sharex=True)
2
3 # positive predictions
4 ax1.set_title("Tree Based Model(s) Predictive Performance",fontsize=24)
5
6 ax1.plot(range(0,6),tprs,color='green',label='True Positive Rate')
7 ax1.plot(range(0,6),tnrs,color='lightgreen',label='True Negative Rate')
8
9 ax1.set_ylabel("Positive classification rate",fontsize=16)
10
11 # negative predictions
12 ax2.plot(range(0,6),fprs,color='red',label='False Positive Rate')
13 ax2.plot(prediction_labels,fnrs,color='pink',label='False Negative Rate')
14
15 ax2.set_ylabel("False classification rate",fontsize=16)
16 ax2.set_xlabel("Model version",fontsize=16)
17
18 ax2.set_xticklabels(prediction_labels,fontsize=12)
19
20 fig.set_size_inches(9, 9)
21 fig.tight_layout()
22 fig.legend(loc='center right',prop={'size':15})
23 plt.show()
```

```
<ipython-input-69-3fc35f12512d>:18: UserWarning: FixedFormatter should only
be used together with FixedLocator
  ax2.set_xticklabels(prediction_labels,fontsize=12)
```



Final Model Interpretation and Evaluation

Plot Random Forest Feature Importance

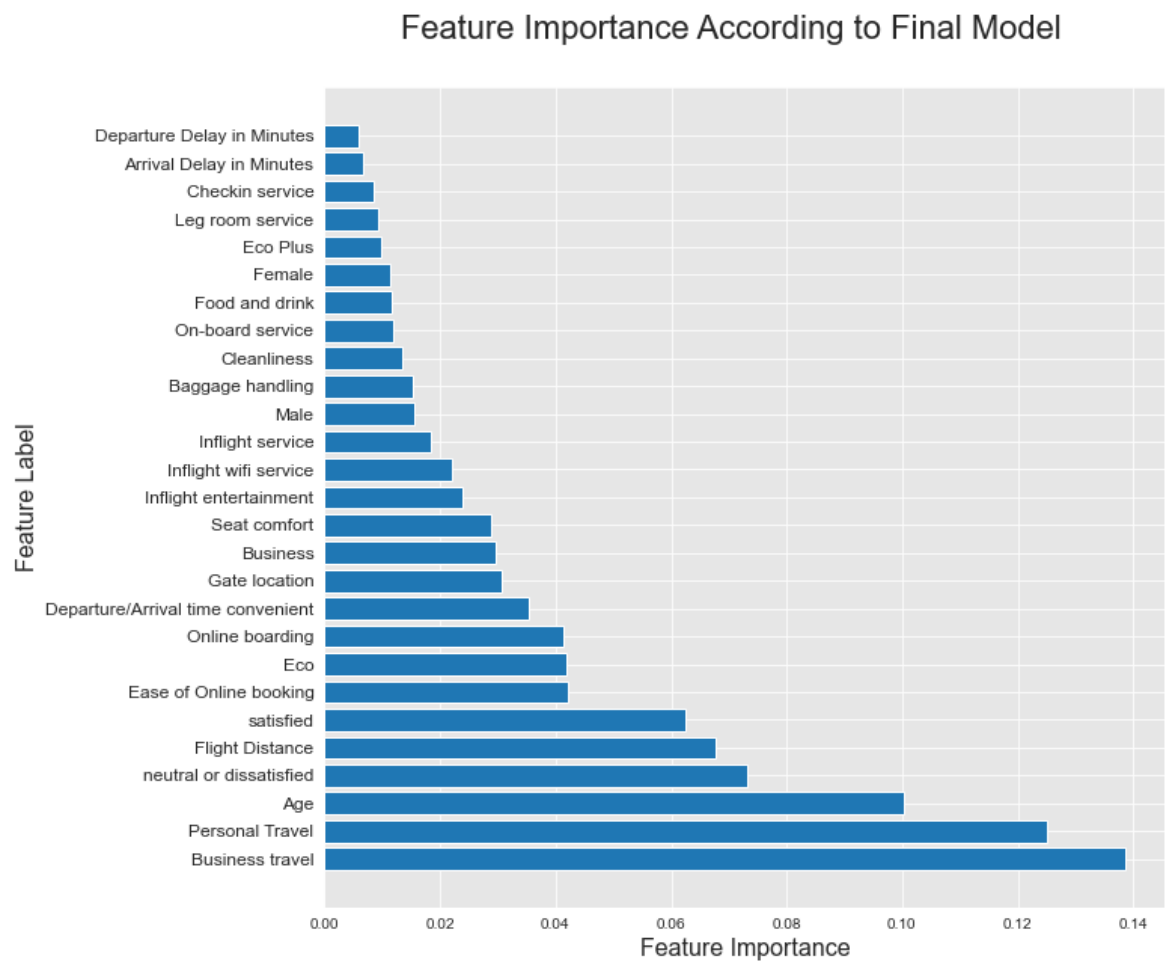
```
In [80]: ▶ 1 # zip feature labels and importances into a dict
          2 feature_importance_dict = {t[0]:t[1] for t in list(zip(X_train.columns.values, feature_importances_))}
          3 # sort the dict by value
          4 feature_importance_dict = dict(sorted(feature_importance_dict.items(), key=lambda item: item[1], reverse=True))
```



```

In [85]: 1 def plot_feature_importances(model):
2         n_features = X_train.shape[1]
3         plt.figure(figsize=(10,10))
4         plt.barh(range(len(feature_importance_dict.keys())), list(feature_im
5         plt.yticks(np.arange(n_features),list(feature_importance_dict.keys()
6         plt.xlabel('Feature importance')
7         plt.ylabel('Feature')
8
9         plot_feature_importances(RFC)
10        plt.suptitle("\nFeature Importance According to Final Model",fontsize=22
11        plt.xlabel('Feature Importance',fontsize=16)
12        plt.ylabel('Feature Label',fontsize=16)
13        plt.show()

```



Concatenate Train and Test Samples for Analysis

In [97]:

```
1 y_vert = pd.Series(y_test)
2 y_vert
```

Out[97]:

0	0.0
1	0.0
2	1.0
3	0.0
4	0.0
	...
25971	1.0
25972	0.0
25973	0.0
25974	0.0
25975	0.0

Length: 25976, dtype: float64

In [91]:

```
1 y_test
```

Out[91]: array([0., 0., 1., ..., 0., 0., 0.])

In [102]:

```
1 # concat X and y, train and test samples into a single df for descriptive
2 X_evaluate = pd.concat([X_train,X_test])
3
4 y_train = pd.Series(y_train)
5 y_test = pd.Series(y_test)
6 y_evaluate = pd.concat([y_train,y_test])
7
8 eval_df = pd.concat([y_evaluate,X_evaluate],axis=1)
9 eval_df.rename(columns = {0:'disloyal Customer'}, inplace=True)
10
11 eval_df.head()
```

Out[102]:

	disloyal Customer	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	co
0	0.0	13	460	3	4	3	1	5	3	
1	1.0	25	235	3	2	3	3	1	3	
2	0.0	26	1142	2	2	2	2	5	5	
3	0.0	25	562	2	5	5	5	2	2	
4	0.0	61	214	3	3	3	3	4	5	

5 rows × 28 columns

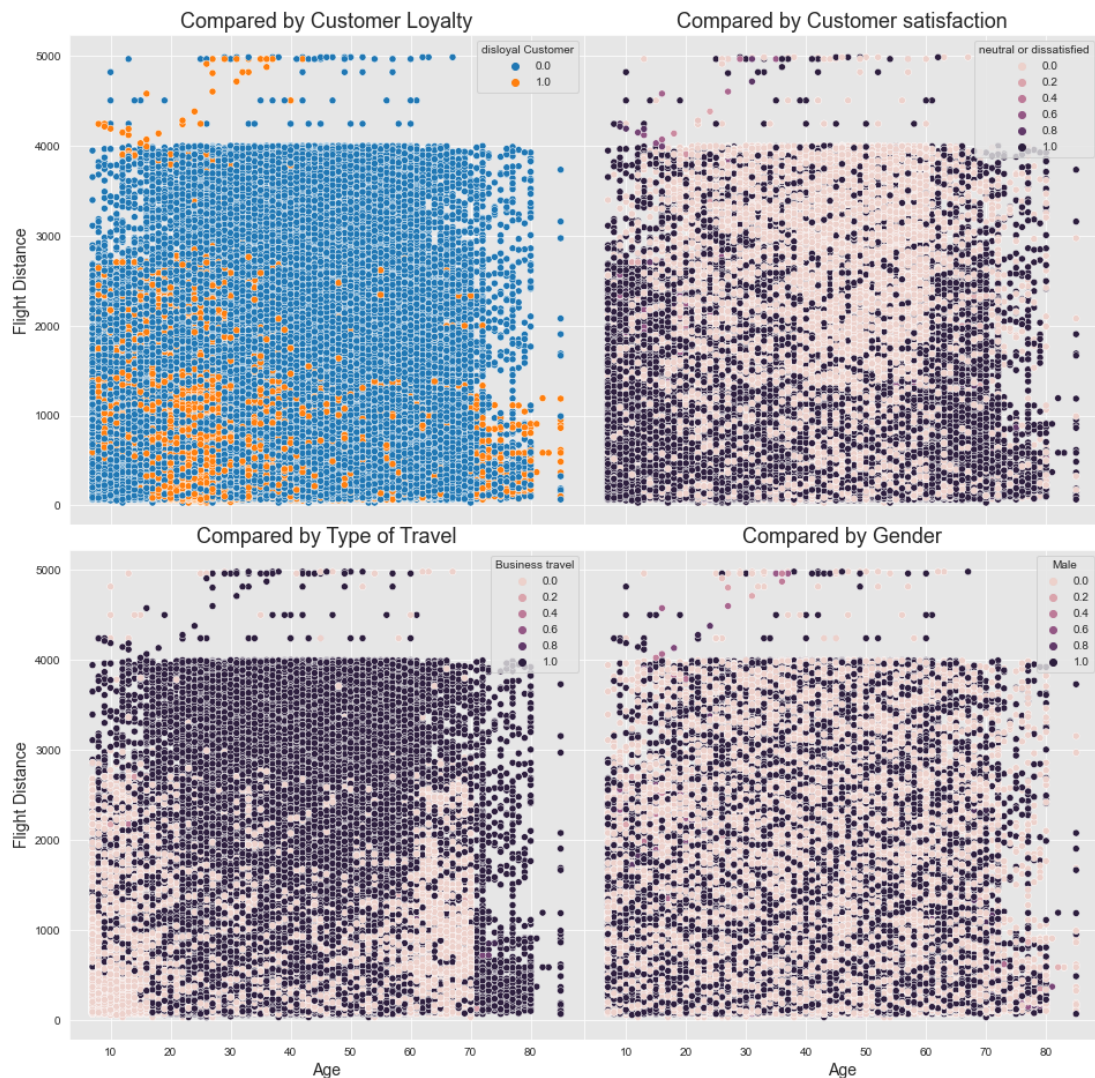
Age vs. Flight Distance by Customer Loyalty

```
In [105]: 1 fig,axs = plt.subplots(2,2,figsize=(16,16),sharex=True,sharey=True)
2
3 loy = sns.scatterplot(x=eval_df.Age,y=eval_df['Flight Distance'],hue=eval
4 axs[0,0].set_title('Compared by Customer Loyalty',fontsize=18)
5 axs[0,0].set_ylabel('Flight Distance',fontsize=14)
6 axs[0,0].set_xlabel('Age',fontsize=14)
7
8 sat = sns.scatterplot(x=eval_df.Age,y=eval_df['Flight Distance'],hue=eval
9 axs[0,1].set_title('Compared by Customer satisfaction',fontsize=18)
10 axs[0,1].set_ylabel('Flight Distance',fontsize=14)
11 axs[0,1].set_xlabel('Age',fontsize=14)
12
13 bix = sns.scatterplot(x=eval_df.Age,y=eval_df['Flight Distance'],hue=eval
14 axs[1,0].set_title('Compared by Type of Travel',fontsize=18)
15 axs[1,0].set_ylabel('Flight Distance',fontsize=14)
16 axs[1,0].set_xlabel('Age',fontsize=14)
17
18 gen = sns.scatterplot(x=eval_df.Age,y=eval_df['Flight Distance'],hue=eval
19 axs[1,1].set_title('Compared by Gender',fontsize=18)
20 axs[1,1].set_ylabel('Flight Distance',fontsize=14)
21 axs[1,1].set_xlabel('Age',fontsize=14)
22
23
24 fig.suptitle('Age vs Flight Distance',fontsize=32)
25 plt.subplots_adjust(wspace=.001,hspace=.05)
26 plt.show()
```

c:\Users\zethu\anaconda3\envs\learn-env\lib\site-packages\IPython\core\pyla
btools.py:132: UserWarning: Creating legend with loc="best" can be slow wit
h large amounts of data.

fig.canvas.print_figure(bytes_io, **kw)

Age vs Flight Distance



The above plot isn't incredibly telling in and of itself, however there are some notable clusters. It's obvious that loyal customers tend to be more satisfied, but the focus of this analysis is to discover what is stopping disloyal customers from becoming loyal. With that in mind, there is a cluster of 16-39 year old, flying less than 300 miles, that are mostly disloyal but relatively even between satisfied and dissatisfied. It also appears that 70+ year olds are more likely to be satisfied if they fly more than 1500 miles, while most of the disloyal customers in this age group fly less than 1500 miles. It also appears that business travel accounts for a significant portion of the aforementioned age groups.

Survey Responses Based on Age, Flight Distance, Purpose for Travel

```
In [106]: 1 # inspect disloyal customers between 15 and 40, for flights less than 3000
2 disloyal_youth_df = eval_df.loc[(eval_df['Age']<40)&(eval_df['Age']>15)&
3
4 disloyal_youth_survey_dict = {}
5 for col in disloyal_youth_df.iloc[:,3:17].columns:
6     disloyal_youth_survey_dict[col]=disloyal_youth_df[col].sum()
7
8 youth_colors = []
9 for val in disloyal_youth_survey_dict.values():
10     if val <195000:
11         youth_colors.append('orangered')
12     elif val > 210000:
13         youth_colors.append('mediumseagreen')
14     else:
15         youth_colors.append('gold')
16
```

```
In [107]: 1 # Inspect Disloyal Senior (70+) Customers, for flights less than 1500 miles
2 disloyal_seniors_df = eval_df.loc[(eval_df['Age']>70)&(eval_df['disloyal
3
4 disloyal_seniors_survey_dict = {}
5 for col in disloyal_seniors_df.iloc[:,3:17].columns:
6     disloyal_seniors_survey_dict[col]=disloyal_seniors_df[col].sum()
7
8 seniors_colors = []
9 for val in disloyal_seniors_survey_dict.values():
10     if val <840:
11         seniors_colors.append('orangered')
12     elif val > 930:
13         seniors_colors.append('mediumseagreen')
14     else:
15         seniors_colors.append('gold')
```

```
In [115]: 1 # Inspect Disloyal business travel, all distances
2 disloyal_business_df = eval_df.loc[(eval_df['Business travel']==0)&(eval
3
4 disloyal_business_survey_dict = {}
5 for col in disloyal_business_df.iloc[:,3:17].columns:
6     disloyal_business_survey_dict[col] = disloyal_business_df[col].sum()
7
8 disloyal_business_colors = []
9 for val in disloyal_business_survey_dict.values():
10     if val <= np.quantile(list(disloyal_business_survey_dict.values()),0
11         disloyal_business_colors.append('orangered')
12     elif val >= np.quantile(list(disloyal_business_survey_dict.values()),
13         disloyal_business_colors.append('mediumseagreen')
14     else:
15         disloyal_business_colors.append('gold')
```

```

In [116]: 1 import matplotlib.patches as mpatches
2 from matplotlib.lines import Line2D
3
4 red_patch = mpatches.Patch(color='orangered', label='Needs Improvement')
5 yellow_patch = mpatches.Patch(color='gold', label='Just Okay')
6 green_patch = mpatches.Patch(color='mediumseagreen',label='Doing Great')
7
8 x_labels = ['Inflight\nwifi service',
9 'Departure/Arrival\ntime convenience',
10 'Ease of Online\nbooking',
11 'Gate location',
12 'Food and drink',
13 'Online boarding',
14 'Seat comfort',
15 'Inflight\nentertainment',
16 'On-board\nservice',
17 'Leg room',
18 'Baggage\nhandling',
19 'Checkin\nservice',
20 'Inflight\nservice',
21 'Cleanliness']
22
23 fig,axs = plt.subplots(3,1,figsize=(40,25),sharex=True)
24
25 axs[0].bar(x=list(disloyal_youth_survey_dict.keys()),height=list(disloyal_youth_survey_dict.values()))
26 axs[0].set_yticklabels([0,50000,100000,150000,200000,250000],fontsize=16)
27 axs[0].set_title('Ages 16-39, flights shorter than 3000 miles',fontsize=16)
28
29
30 axs[1].bar(x=list(disloyal_seniors_survey_dict.keys()),height=list(disloyal_seniors_survey_dict.values()))
31 axs[1].set_yticklabels([0,200,400,600,800,1000],fontsize=18)
32 axs[1].set_xticklabels(x_labels,fontsize=20)
33 axs[1].set_title("Ages 70+, flights shorter than 1500 miles",fontsize=32)
34
35 axs[2].bar(x=list(disloyal_business_survey_dict.keys()),height=list(disloyal_business_survey_dict.values()))
36 axs[2].set_yticklabels([0,50000,100000,150000,200000,250000,300000],fontsize=16)
37 axs[2].set_xticklabels(x_labels,fontsize=20)
38 axs[2].set_title("All disloyal business travel",fontsize=36)
39
40
41 fig.legend(handles=[green_patch,yellow_patch,red_patch],prop={'size':25})
42 fig.suptitle("Disloyal Customers Survey Responses, By age and length of flight",fontsize=24)
43 plt.show()

```

<ipython-input-116-573ec48e6e41>:26: UserWarning: FixedFormatter should only be used together with FixedLocator

```
axs[0].set_yticklabels([0,50000,100000,150000,200000,250000],fontsize=16)
```

<ipython-input-116-573ec48e6e41>:31: UserWarning: FixedFormatter should only be used together with FixedLocator

```
axs[1].set_yticklabels([0,200,400,600,800,1000],fontsize=18)
```

<ipython-input-116-573ec48e6e41>:32: UserWarning: FixedFormatter should only be used together with FixedLocator

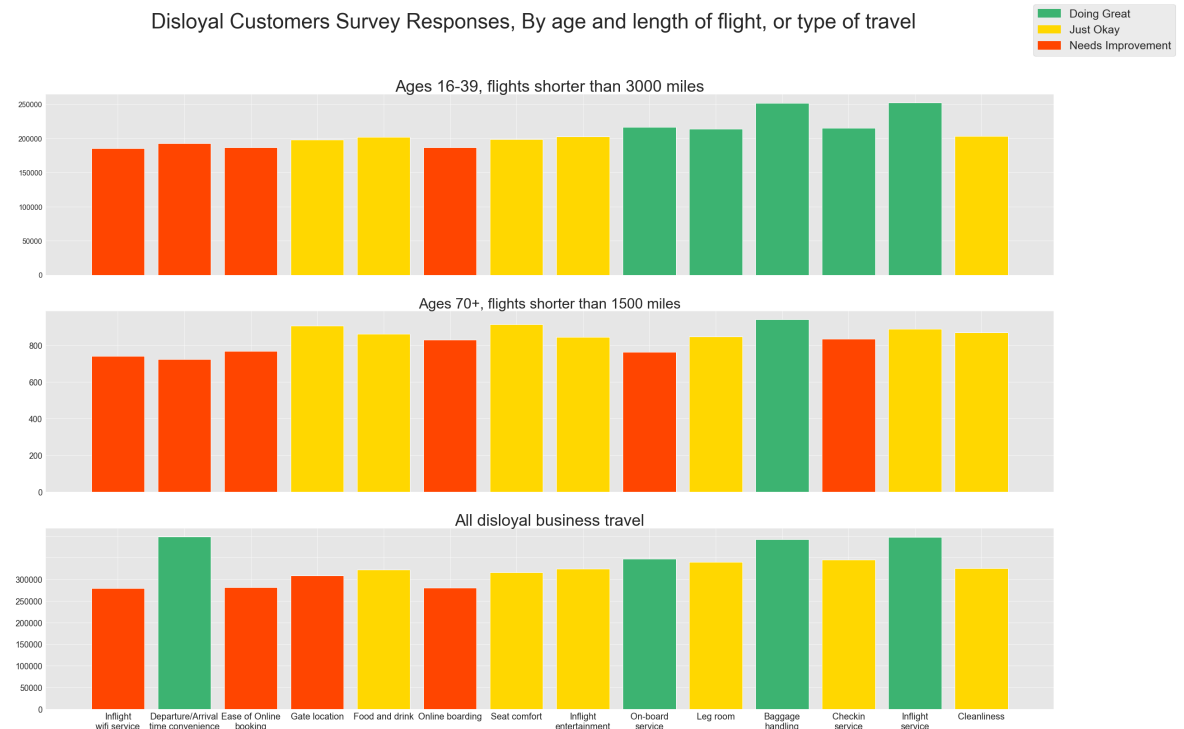
```
axs[1].set_xticklabels(x_labels,fontsize=20)
```

<ipython-input-116-573ec48e6e41>:36: UserWarning: FixedFormatter should only be used together with FixedLocator


```

y be used together with FixedLocator
    axs[2].set_yticklabels([0,50000,100000,150000,200000,250000,300000],fontsize=18)
<ipython-input-116-573ec48e6e41>:37: UserWarning: FixedFormatter should only
y be used together with FixedLocator
    axs[2].set_xticklabels(x_labels,fontsize=20)

```



Across both age groups as well as for all business travel, Inflight wifi, departure/arrival time convenience and ease of online booking all scored in the bottom quartile of overall customer satisfaction. Food and drink, and cleanliness are just average for accross the board. Baggage handling and Inflight wifi score in the fourth quartile accross the board.

Online boarding, one of the top 5 most important features to the model, is in the bottom quartile for young adults as well as businss travel, and only about average for seniors. Businss travel and young adults are well pleased with the check-in service but it is highly disliked by seniors. Gate Location is just okay for businss travelers and young adults, but seniors are highly satisfied here.

Based on this graph I would recommend focusing on improving the online experience (boarding and booking) for young adults and businss travelers, and improve inflight wifi, food and drink, seat comfort, and overall cleanliness. Departure/Arrival times and gate locations would probably have significant impact if improved but it is likely impossible to improve these things strictly internally.

Market Share by Age and Purpose for Travel

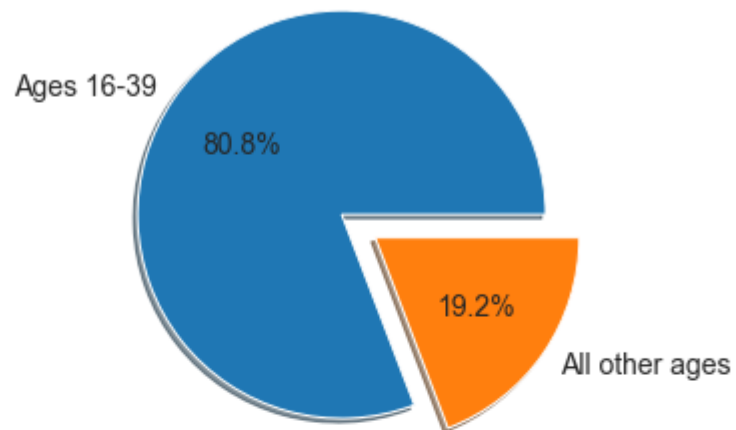
```
In [125]: 1 eval_df['Business travel'].loc[eval_df['Business travel']==1].sum()
```

Out[125]: 154634.0

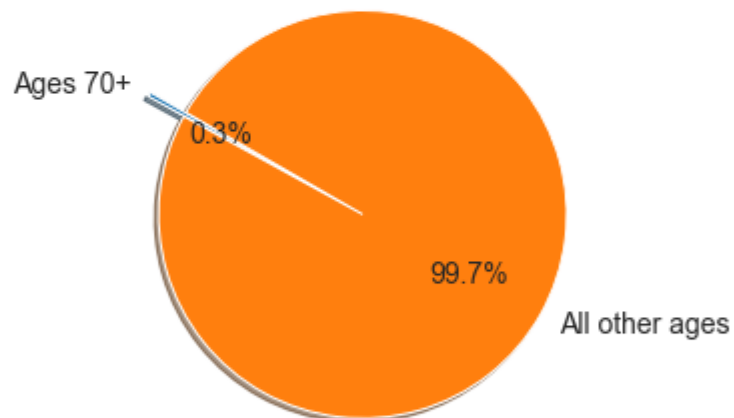
```
In [126]: 1 fig,axs = plt.subplots(3,1,figsize=(10,15))
2         axs[0].pie([len(disloyal_youth_df),len(eval_df.loc[eval_df['disloyal Cus
3             labels=['Ages 16-39','All other ages'],
4             startangle=0,
5             shadow=True,
6             explode=[0,0.2],
7             autopct='%1.1f%%',
8             textprops={'fontsize':14})
9         axs[0].set_title("Young adults flying less than 3000 miles",fontsize=16)
10
11
12         axs[1].pie([len(disloyal_seniors_df),len(eval_df.loc[eval_df['disloyal C
13             labels=['Ages 70+', 'All other ages'],
14             startangle=150,
15             shadow=True,
16             explode=[.2,0],
17             autopct='%1.1f%%',
18             textprops={'fontsize':14})
19         axs[1].set_title("Seniors flying less than 1500 miles",fontsize=16)
20
21         axs[2].pie([eval_df['Business travel'].loc[eval_df['Business travel']==1
22             labels=['Business travel','Personal Travel'],
23             startangle=-30,
24             shadow=True,
25             explode=[0,0.2],
26             autopct='%1.1f%%',
27             textprops={'fontsize':14})
28         axs[2].set_title("Business travel, all ages.\nall distances",fontsize=16)
29
30         fig.suptitle("\n\nDisloyal Market Share by Demographic Groups",fontsize=
31         plt.subplots_adjust(hspace=0.1)
32         plt.show()
```


Disloyal Market Share by Demographic Groups

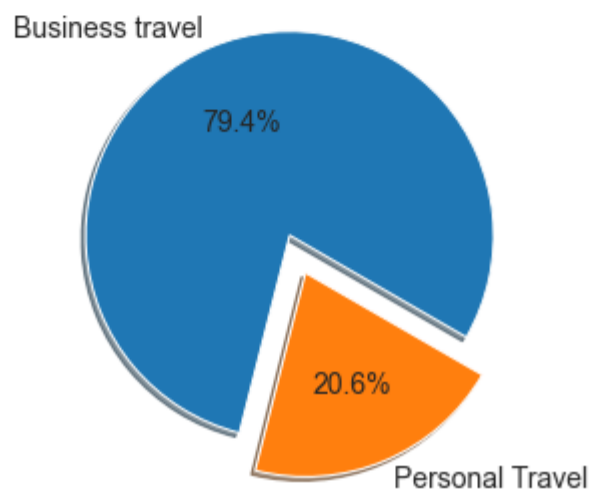
Young adults flying less than 3000 miles



Seniors flying less than 1500 miles



Business travel, all ages.
all distances



Examination of the market share of each of group shows that improving satisfaction for senior

customers would probably have a significant impact on overall business performance. Based on the above figure it is clearly best to focus on improving the customer experience for business traveler's and young adults, obviously there is likely much overlap between these two groups.

Survey Responses by Customer Loyalty and Satisfaction

In [131]:

```

1  # sum of survey responses for disloyal satisfied customers
2  disloyal_survey_satisfied = {}
3  for col in eval_df.iloc[:,3:17].columns:
4      disloyal_survey_satisfied[col] = eval_df.loc[(eval_df['disloyal Customer
5
6  disloyal_satisfied_colors = []
7  for val in disloyal_survey_satisfied.values():
8      if val < np.quantile(list(disloyal_survey_satisfied.values()),0.25):
9          disloyal_satisfied_colors.append('orangered')
10     elif val > np.quantile(list(disloyal_survey_satisfied.values()),0.75):
11         disloyal_satisfied_colors.append('mediumseagreen')
12     else:
13         disloyal_satisfied_colors.append('gold')
14
15  # sum of survey responses for disloyal dissatisfied customers
16  disloyal_survey_dissatisfied = {}
17  for col in eval_df.iloc[:,3:17].columns:
18      disloyal_survey_dissatisfied[col] = eval_df.loc[(eval_df['disloyal Customer
19
20  disloyal_dissatisfied_colors = []
21  for val in disloyal_survey_dissatisfied.values():
22      if val < np.quantile(list(disloyal_survey_dissatisfied.values()),0.25):
23          disloyal_dissatisfied_colors.append('orangered')
24      elif val > np.quantile(list(disloyal_survey_dissatisfied.values()),0.75):
25          disloyal_dissatisfied_colors.append('mediumseagreen')
26      else:
27          disloyal_dissatisfied_colors.append('gold')
28
29  #####
30
31  # sum of survey responses for loyal satisfied customers
32  loyal_survey_satisfied = {}
33  for col in eval_df.iloc[:,3:17].columns:
34      loyal_survey_satisfied[col] = eval_df.loc[(eval_df['disloyal Customer
35
36  loyal_satisfied_colors = []
37  for val in loyal_survey_satisfied.values():
38      if val < np.quantile(list(loyal_survey_satisfied.values()),0.25):
39          loyal_satisfied_colors.append('orangered')
40      elif val > np.quantile(list(loyal_survey_satisfied.values()),0.75):
41          loyal_satisfied_colors.append('mediumseagreen')
42      else:
43          loyal_satisfied_colors.append('gold')
44
45  # sum of survey responses for loyal dissatisfied customers
46  loyal_survey_dissatisfied = {}
47  for col in eval_df.iloc[:,3:17].columns:
48      loyal_survey_dissatisfied[col] = eval_df.loc[(eval_df['disloyal Customer
49
50  loyal_dissatisfied_colors = []
51  for val in loyal_survey_dissatisfied.values():
52      if val < np.quantile(list(loyal_survey_dissatisfied.values()),0.25):
53          loyal_dissatisfied_colors.append('orangered')
54      elif val > np.quantile(list(loyal_survey_dissatisfied.values()),0.75):
55          loyal_dissatisfied_colors.append('mediumseagreen')
56      else:

```

57

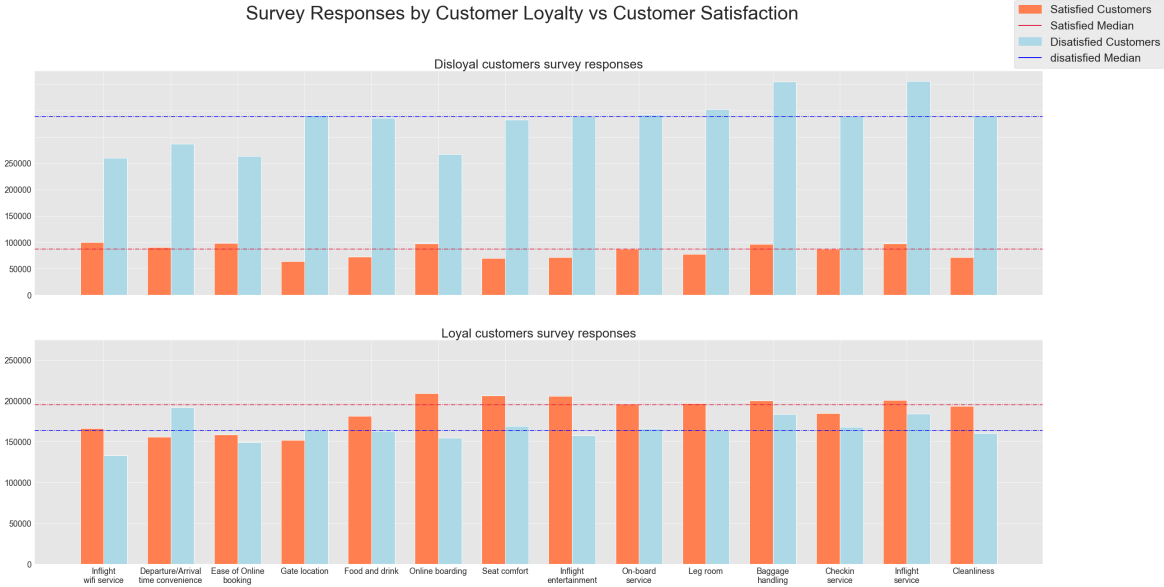
```
loyal_disatisfied_colors.append('gold')
```

```

In [133]: ▶ 1 satisfied_patch = mpatches.Patch(color='coral', label='Satisfied Customer
2 dissatisfied_patch = mpatches.Patch(color='lightblue', label='Disatisfied
3 satisfied_median = Line2D([0],[0],color='crimson',lw=2,label='Satisfied M
4 dissatisfied_median = Line2D([0],[0],color='blue',lw=2,label='disatisfied
5
6 width = .35
7
8 x_labels = ['Inflight\nwifi service',
9 'Departure/Arrival\ntime convenience',
10 'Ease of Online\nbooking',
11 'Gate location',
12 'Food and drink',
13 'Online boarding',
14 'Seat comfort',
15 'Inflight\nentertainment',
16 'On-board\nservice',
17 'Leg room',
18 'Baggage\nhandling',
19 'Checkin\nservice',
20 'Inflight\nservice',
21 'Cleanliness']
22
23 x = np.arange(len(x_labels))
24
25 fig,axs = plt.subplots(2,1,figsize=(40,20),sharex=True)
26
27 axs[0].bar(x= x + width/2,height=list(disloyal_survey_dissatisfied.values
28 axs[0].axhline(np.quantile(list(disloyal_survey_dissatisfied.values()),0.5)
29
30 axs[0].bar(x= x - width/2,height=list(disloyal_survey_satisfied.values())
31 axs[0].axhline(np.quantile(list(disloyal_survey_satisfied.values()),0.5)
32
33 axs[0].set_title('Disloyal customers survey responses',fontsize=28)
34 axs[0].set_yticklabels([0,50000,100000,150000,200000,250000],fontsize=18)
35
36
37
38 axs[1].bar(x= x - width/2,height=list(loyal_survey_satisfied.values()),w
39 axs[1].axhline(np.quantile(list(loyal_survey_satisfied.values()),0.5),co
40
41 axs[1].bar(x= x + width/2,height=list(loyal_survey_dissatisfied.values())
42 axs[1].axhline(np.quantile(list(loyal_survey_dissatisfied.values()),0.5),
43
44 axs[1].set_title('Loyal customers survey responses',fontsize=28)
45 axs[1].set_ylim(0,275000)
46 plt.xticks(x,x_labels,fontsize=18)
47 plt.yticks(fontsize=18)
48
49
50 fig.legend(handles=[satisfied_patch,satisfied_median,dissatisfied_patch,d
51 fig.suptitle("Survey Responses by Customer Loyalty vs Customer Satisfact
52 plt.show()

```

<ipython-input-133-8d324d897e82>:34: UserWarning: FixedFormatter should only be used together with FixedLocator



As one would expect disloyal customers tend to be more dissatisfied. Unlike they're loyal counterparts, disloyal customers are displeased the most with checkin service and inflight service. They are also more displeased than usual with on-board service, leg room, check-in service, and cleanliness.

Based on this figure with consideration to feature importance of the predictive model, I recommend focusing on improving services involved with the check-in and boarding process as well as the in-flight experience; especially in-flight entertainment, online boarding, and seat comfort.

Age and Flight Distance by Customer Loyalty and Satisfaction

In [134]:

```
1 # bar heights for loyal satisfied customers
2 loyal_satisfied_flight_distance = eval_df['Flight Distance'].loc[(eval_d
3 loyal_satisfied_flight_ranges = pd.cut(loyal_satisfied_flight_distance,b
4
5 loyal_satisfied_flight_heights = []
6 for interval in loyal_satisfied_flight_ranges:
7     chunk = sum((loyal_satisfied_flight_distance>interval.left)&(loyal_s
8     loyal_satisfied_flight_heights.append(chunk)
9
10 # bar heights for loyal but dissatisfied customers
11 loyal_dissatisfied_flight_distance = eval_df['Flight Distance'].loc[(eval
12 loyal_dissatisfied_flight_ranges = pd.cut(loyal_dissatisfied_flight_distanc
13
14 loyal_dissatisfied_flight_heights = []
15 for interval in loyal_dissatisfied_flight_ranges:
16     chunk = sum((loyal_dissatisfied_flight_distance>interval.left)&(loyal
17     loyal_dissatisfied_flight_heights.append(chunk)
18
19 # bar heights for disloyal but satisfied customers
20 disloyal_satisfied_flight_distance = eval_df['Flight Distance'].loc[(eval
21 disloyal_satisfied_flight_ranges = pd.cut(disloyal_satisfied_flight_distanc
22
23 disloyal_satisfied_flight_heights = []
24 for interval in disloyal_satisfied_flight_ranges:
25     chunk = sum((disloyal_satisfied_flight_distance>interval.left)&(dislo
26     disloyal_satisfied_flight_heights.append(chunk)
27 disloyal_satisfied_flight_heights = [-x for x in disloyal_satisfied_flight
28
29 # bar heights for disloyal and dissatisfied customers
30 disloyal_dissatisfied_flight_distance = eval_df['Flight Distance'].loc[(ev
31 disloyal_dissatisfied_flight_ranges = pd.cut(disloyal_dissatisfied_flight_c
32
33 disloyal_dissatisfied_flight_heights = []
34 for interval in disloyal_dissatisfied_flight_ranges:
35     chunk = sum((disloyal_dissatisfied_flight_distance>interval.left)&(dis
36     disloyal_dissatisfied_flight_heights.append(chunk)
37 disloyal_dissatisfied_flight_heights = [-x for x in disloyal_dissatisfied_
```

In [135]:

```

1  # bar heights for loyal satisfied customers
2  loyal_satisfied_age = eval_df['Age'].loc[(eval_df['satisfied']==1)&(eval_
3  loyal_satisfied_age_ranges = pd.cut(loyal_satisfied_age,bins=50).value_co
4
5  loyal_satisfied_age_heights = []
6  for interval in loyal_satisfied_age_ranges:
7      chunk = sum((loyal_satisfied_age>interval.left)&(loyal_satisfied_age
8      loyal_satisfied_age_heights.append(chunk)
9
10 # bar heights for loyal but dissatisfied customers
11 loyal_dissatisfied_age = eval_df['Age'].loc[(eval_df['satisfied']==0)&(ev
12 loyal_dissatisfied_age_ranges = pd.cut(loyal_dissatisfied_age,bins=50).val
13
14 loyal_dissatisfied_age_heights = []
15 for interval in loyal_dissatisfied_age_ranges:
16     chunk = sum((loyal_dissatisfied_age>interval.left)&(loyal_dissatisfied
17     loyal_dissatisfied_age_heights.append(chunk)
18
19 # bar heights for disloyal but satisfied customers
20 disloyal_satisfied_age = eval_df['Age'].loc[(eval_df['satisfied']==1)&(e
21 disloyal_satisfied_age_ranges = pd.cut(disloyal_satisfied_age,bins=50).va
22
23 disloyal_satisfied_age_heights = []
24 for interval in disloyal_satisfied_age_ranges:
25     chunk = sum((disloyal_satisfied_age>interval.left)&(disloyal_satisfi
26     disloyal_satisfied_age_heights.append(chunk)
27 disloyal_satisfied_age_heights = [-x for x in disloyal_satisfied_age_heig
28
29 # bar heights for disloyal and dissatisfied customers
30 disloyal_dissatisfied_age = eval_df['Age'].loc[(eval_df['satisfied']==0)&
31 disloyal_dissatisfied_age_ranges = pd.cut(disloyal_dissatisfied_age,bins=50
32
33 disloyal_dissatisfied_age_heights = []
34 for interval in disloyal_dissatisfied_age_ranges:
35     chunk = sum((disloyal_dissatisfied_age>interval.left)&(disloyal_disat
36     disloyal_dissatisfied_age_heights.append(chunk)
37 disloyal_dissatisfied_age_heights = [-x for x in disloyal_dissatisfied_age

```

In [136]:

```

1  flight_x_ticks = np.linspace(min(eval_df['Flight Distance']),max(eval_df
2  age_x_ticks = np.linspace(min(eval_df['Age']),max(eval_df['Age']),num=50

```

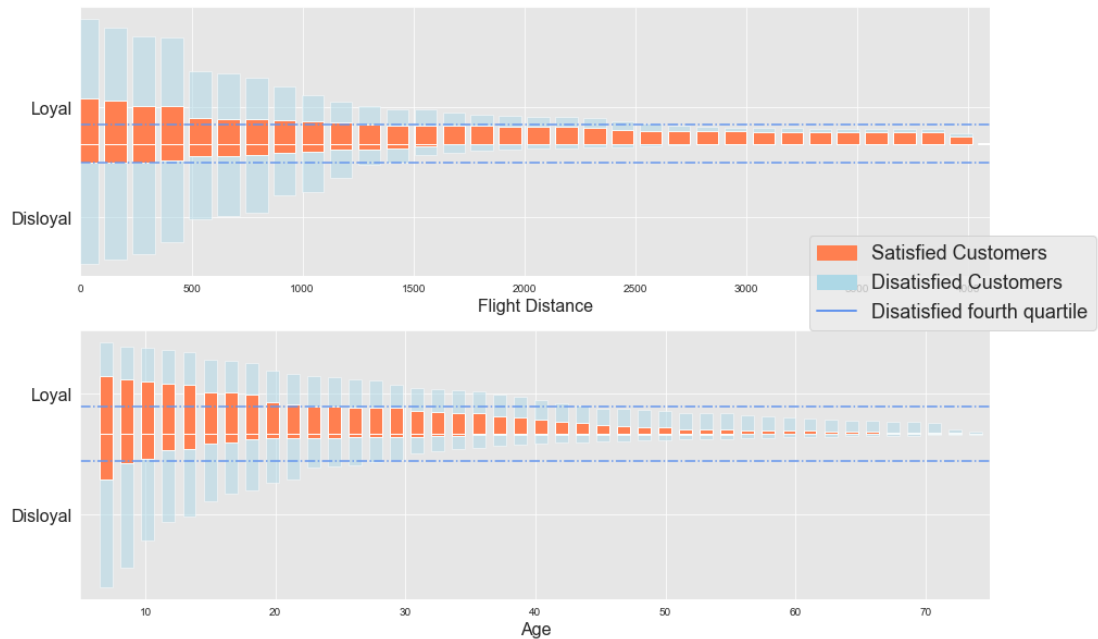

In [137]:

```

1  satisfied_patch = mpatches.Patch(color='coral', label='Satisfied Customer')
2  disatisfied_patch = mpatches.Patch(color='lightblue', label='Disatisfied Customer')
3  satisfied_quartile = Line2D([0],[0],color='cornflowerblue',lw=2,label='Disatisfied Customer')
4
5  fig,axs = plt.subplots(2,1,figsize=(15,10))
6
7  axs[0].bar(x=fly_x_ticks,height=loyal_satisfied_fly_heights,color='coral')
8  axs[0].bar(x=fly_x_ticks,height=loyal_disatisfied_fly_heights,bottom=loyal_satisfied_fly_heights,color='lightblue',width=100,alpha=0.5)
9
10 axs[0].axhline(np.quantile(loyal_disatisfied_fly_heights,0.75),color='cornflowerblue',lw=2)
11
12 axs[0].bar(x=fly_x_ticks,height=disloyal_satisfied_fly_heights,color='coral')
13 axs[0].bar(x=fly_x_ticks,height=disloyal_disatisfied_fly_heights,bottom=disloyal_satisfied_fly_heights,color='lightblue',width=100,alpha=0.5)
14
15 axs[0].axhline(np.quantile(disloyal_disatisfied_fly_heights,0.25),color='cornflowerblue',lw=2)
16 axs[0].set_xlim(0,4100)
17 axs[0].set_yticks(ticks=[3000,-6000])
18 axs[0].set_yticklabels(['Loyal','Disloyal'],fontsize=16)
19 axs[0].set_xlabel("Flight Distance",fontsize=16)
20
21
22
23 axs[1].bar(x=age_x_ticks,height=loyal_satisfied_age_heights,color='coral')
24 axs[1].bar(x=age_x_ticks,height=loyal_disatisfied_age_heights,bottom=loyal_satisfied_age_heights,color='lightblue',width=1,alpha=0.5)
25
26 axs[1].axhline(np.quantile(loyal_disatisfied_age_heights,0.75),color='cornflowerblue',lw=2)
27
28 axs[1].bar(x=age_x_ticks,height=disloyal_satisfied_age_heights,color='coral')
29 axs[1].bar(x=age_x_ticks,height=disloyal_disatisfied_age_heights,bottom=disloyal_satisfied_age_heights,color='lightblue',width=1,alpha=0.5)
30
31 axs[1].axhline(np.quantile(disloyal_disatisfied_age_heights,0.25),color='cornflowerblue',lw=2)
32 axs[1].set_xlim(5,75)
33 axs[1].set_yticks(ticks=[2500,-5000])
34 axs[1].set_yticklabels(['Loyal','Disloyal'],fontsize=16)
35 axs[1].set_xlabel("Age",fontsize=16)
36
37 fig.legend(handles=[satisfied_patch,disatisfied_patch,satisfied_quartile],loc='best')
38 fig.suptitle("Customer Satisfaction By Age, Flight Distance and Loyalty",fontsize=16)
39 plt.show()

```

Customer Satisfaction By Age, Flight Distance and Loyalty



Based on the figure above, the most dissatisfaction to address is for flights shorter than 1500 miles, and customers under the age of 40. Its also interesting to not that there is just not much data on disloyal customers traveling farther than 2500 miles.

Based on this figure I recommend focusing on improve checkin-in, boarding, and in-flight services for flights under 2000 miles and customers under age 40

Conclusion

The data used in this analysis does not contain any information on the revenue or profits made from each ticket represented; there is some details on the flight itself and most of the data represents the subject satisfaction report of the customer represented. So it is impossible to make any recomendations at this point about how to drive profits, but it is posible to make recomendations on how to increase customer satisfaction and which populations within the market are the most dissatisfied.

The Random Forest feature importance showed that the age of the customer, wether the travel is for business or personal, the distance of the flight,and wether or not the customer was satisfeid were the most important features (in terms of the model making predictions) by a significant margin, so my broad strategy was to analyze the survey responses for based on various subsets of those features.

The broad patterns I see are that disloyal customers are the most displeased with the process of actually getting on the plane, and some in-flight services need improvement. The most important groups to focus on improving services for is young adults (16-39) and business travel (all ages) based on rate of dissatisfaction and market share.

Specific pain points to investigate for improvement is the UX for online booking as well as online boarding. Customers are also very displeased with internet related services in-flight, specifically wifi and entertainment. Improving customer wifi service should improve satisfaction with entertainment as well. Customer accross the board are apathetic about cleanliness as well as food and drink. Seat comfort, of course, could use some improvement. Customer's are extremely dissatisfied with departure/arival times, and gate locations, but I don't think there is much that can be done about these issues without serious cooperation and concession with other airlines. I believe more impactful change could be made quicker and more economically with the aformentioned pain points.

Type *Markdown* and LaTeX: α^2