Modeling Notebook One

This notebook contains the logic for performing Linear Regression, Ridge Regression, Random Forest Regression, and Random Forest Classification.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
!ls
```

```
cleaned = pd.read_csv("/content/drive/Shareddrives/Data Science 303 Group Project/csv/cleaned_fires_data_with_four_closest_stations_nov21.csv")
cleaned = cleaned.replace([np.inf, -np.inf], np.nan)
cleaned = cleaned.dropna()
```

```
display(cleaned.head())
display(cleaned.info(verbose=True))

STATION_LIST = ["BODIE", "BROOKS", "COHASSET", "EEL_RIVER", "HELL_HOLE", "HERNANDEZ"]
STATION_LIST += ["HUNTER_MOUNTAIN", "JUANITA_LAKE", "LADDER_BUTTE", "LAS_TABLAS", "LA_HONDA", "OAK_CREEK"]
STATION_LIST += ["PANAMINT", "PILOT_HILL", "SCORPION", "SOLDIER_MOUNTAIN", "SQUAW_LAKE", "STAMPEDE", "VAN_BREMMER", "WOLVERTON"]
STATION_LIST = set(STATION_LIST)
```

Final Part of Normalization: MinMax scaling latitude and longitude

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

lat = scaler.fit_transform(np.array(cleaned["LATITUDE"]).reshape(-1, 1))
long = scaler.fit_transform(np.array(cleaned["LONGITUDE"]).reshape(-1, 1))

# DROP ALL LATITUDE LONGITUDE COLUMNS, THEN READD THE SCALED LATITUDE AND LONGITUDE
drops = []
for col in cleaned.columns:
  if "LATITUDE" in col or "LONGITUDE" in col:
    drops.append(col)

for col in drops:
  del cleaned[col]

cleaned["S_LATITUDE"] = lat
cleaned["S_LONGITUDE"] = long

```

```
# Elevation for the primary station is redundant. Drop all elevation columns from the primary station, except for one
elevation = cleaned["PRIMARY_STATION_1_MONTHS_PRIOR_ELEVATION"]

drops = []
for col in cleaned.columns:
  if "PRIMARY_STATION" in col and "ELEVATION" in col:
    drops.append(col)

for col in drops:
  del cleaned[col]

cleaned["PRIMARY_STATION_ELEVATION"] = elevation

```

```
# We should have a large number of fires. If not, fail
print(len(cleaned))
assert(len(cleaned) > 70000)
```

```
print( (1 - 360) % 365)
```

```
cleaned['DURATION'] = (cleaned['CONT_DOY']-cleaned['DISCOVERY_DOY'])%365
```

```
cleaned['DURATION'].max()
```

```
from sklearn.linear_model import LinearRegression, Ridge
```

```
# Working off of closest_station_1 for now, we can replicate later with other stations if we want
# 1. Get number of days that the fire was going (containment date - discovery date)
# Model 1a: fire duration
```

```
numerical = cleaned.select_dtypes(include="number")
numerical.head()
numerical.info(verbose=True)
```

```
cols = numerical.columns
closest = []

for col in cols:
  if (col[:16] == 'CLOSEST_STATION_' or col in STATION_LIST or col == "S_LATITUDE" or col == "S_LONGITUDE"):
    closest.append(col)
```

```
primary_station_data_cols = []
for col in cols:
```

```
  if ("PRIMARY_STATION" in col or col in STATION_LIST or col == "S_LATITUDE" or col == "S_LONGITUDE"):
    primary_station_data_cols.append(col)

primary_station_df = pd.DataFrame()
for col in primary_station_data_cols:
  primary_station_df[col] = numerical[col]

primary_station_df["FIRE_SIZE"] = numerical["FIRE_SIZE"]
primary_station_df["DURATION"] = numerical["DURATION"]

```

```
NUM_FEATURES_TO_KEEP = 300 + 2
correlation = primary_station_df.corr(method='pearson')
highest_correlation = (correlation.nlargest(NUM_FEATURES_TO_KEEP, 'FIRE_SIZE').index)
print(f"TOP {NUM_FEATURES_TO_KEEP} FEATURES WITH HIGHEST CORRELATION TO FIRE SIZE")
print(list(highest_correlation))

print(f"TOP 10 FEATURES WITH HIGHEST CORRELATION TO FIRE SIZE")
display(highest_correlation[2:12])
display(correlation.nlargest(NUM_FEATURES_TO_KEEP, 'FIRE_SIZE')[2:12]["FIRE_SIZE"])

fire_size_prediction_df = primary_station_df[highest_correlation]
del fire_size_prediction_df["FIRE_SIZE"]
del fire_size_prediction_df["DURATION"]
```

```
display(fire_size_prediction_df)
```

```
NUM_FEATURES_TO_KEEP = 300 + 2
correlation = primary_station_df.corr(method='pearson')
highest_correlation = (correlation.nlargest(NUM_FEATURES_TO_KEEP, 'DURATION').index)
print(f"TOP {NUM_FEATURES_TO_KEEP} FEATURES WITH HIGHEST CORRELATION TO FIRE DURATION")
display(highest_correlation)

print(f"TOP 10 FEATURES WITH HIGHEST CORRELATION TO FIRE DURATION")
display(highest_correlation[2:12])
display(correlation.nlargest(NUM_FEATURES_TO_KEEP, 'DURATION')[2:12]["DURATION"])

fire_duration_prediction_df = primary_station_df[highest_correlation]
del fire_duration_prediction_df["FIRE_SIZE"]
del fire_duration_prediction_df["DURATION"]
```

```
display(fire_duration_prediction_df)
```

```
closest_df = pd.DataFrame()

for col in closest:
  closest_df[col] = numerical[col]
```

```
closest_df.head()
```

Model 1: Linear Regression

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, median_absolute_error, r2_score
from sklearn.metrics import explained_variance_score
from sklearn import svm
from sklearn.preprocessing import PowerTransformer
from sklearn.ensemble import RandomForestRegressor

# def log_transform(column):
#   """
#   Transforms a feature so that it is scaled logarithmically. Useful for correcting floating point errors
#   """


# def min_max_transform(column):
#   """
#   Transforms a feature using min-max scaling
#   """
#   pass

def run_full_linear_regression_with_accuracy(features, response, n_highest = 0):
  print(f"Response Minimum: {response.min()}")
  print(f"Response Maximum: {response.max()}")

  model = LinearRegression()
  # Split into test and training set
  X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
  model.fit(X_train, y_train)

  training_predictions = model.predict(X_train)
  test_predictions = model.predict(X_test)
  print(f"Linear Regression Coefficients: { model.coef_ }\nLinear Regression Intercept: { model.intercept_}\n")

  print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
  print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))

  print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
  print("Test average error rate: ", median_absolute_error(y_test, test_predictions))

  print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
  print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))

  print("R2 Score training set: ", r2_score(y_train, training_predictions))
  print("R2 Score testing set: ", r2_score(y_test, test_predictions))

  fig = plt.figure()
```

```
  ax1 = fig.add_subplot()
  ax1.set_xlabel("True Value from Test Set")
  ax1.set_ylabel('Prediction from Test Set')
  ax1.set_title('True Value vs Predicted Value for Test Set: Linear Regression')
  ax1.scatter(y_test, test_predictions)
  plt.show()

  # Residual plot against predictor
  fig = plt.figure()
  ax1 = fig.add_subplot()
  ax1.set_xlabel("Test Set Prediction")
  ax1.set_ylabel('Residuals from Test Set')
  ax1.set_title('Residual Graph: Linear Regression')
  ax1.scatter(test_predictions, y_test - test_predictions)
  plt.show()



def run_full_ridge_regression_with_accuracy(features, response, n_highest = 0):
  print(f"Response Minimum: {response.min()}")
  print(f"Response Maximum: {response.max()}")


  model = Ridge()
  # Split into test and training set
  X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
  model.fit(X_train, y_train)

  training_predictions = model.predict(X_train)
  test_predictions = model.predict(X_test)
  print(f"Ridge Regression Coefficients: { model.coef_ }\Ridge Regression Intercept: { model.intercept_}\n")

  print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
  print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))

  print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
  print("Test average error rate: ", median_absolute_error(y_test, test_predictions))

  print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
  print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))

  print("R2 Score training set: ", r2_score(y_train, training_predictions))
  print("R2 Score testing set: ", r2_score(y_test, test_predictions))

  fig = plt.figure()
  ax1 = fig.add_subplot()
  ax1.set_xlabel("True Value from Test Set")
  ax1.set_ylabel('Prediction from Test Set')
  ax1.set_title('True Value vs Predicted Value for Test Set: Ridge Regression')
  ax1.scatter(y_test, test_predictions)
  plt.show()

  # Residual plot against predictor
  fig = plt.figure()
  ax1 = fig.add_subplot()
  ax1.set_xlabel("Test Set Prediction")
  ax1.set_ylabel('Residuals from Test Set')
  ax1.set_title('Residual Graph: Ridge Regression')
  ax1.scatter(test_predictions, y_test - test_predictions)
  plt.show()

def run_full_SVM_regression_with_accuracy(features, response, n_highest = 0):
  print(f"Response Minimum: {response.min()}")
  print(f"Response Maximum: {response.max()}")


  model = svm.SVR()
  # Split into test and training set
  X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
  model.fit(X_train, y_train)

  training_predictions = model.predict(X_train)
  test_predictions = model.predict(X_test)

  print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
  print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))

  print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
  print("Test average error rate: ", median_absolute_error(y_test, test_predictions))

  print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
  print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))

  fig = plt.figure()
  ax1 = fig.add_subplot()
  ax1.set_xlabel("True Value from Test Set")
  ax1.set_ylabel('Prediction from Test Set')
  ax1.set_title('True Value vs Predicted Value for Test Set: SVM Regression')
  ax1.scatter(y_test, test_predictions)
  plt.show()

  # Residual plot against predictor
  fig = plt.figure()
  ax1 = fig.add_subplot()
  ax1.set_xlabel("Test Set Prediction")
  ax1.set_ylabel('Residuals from Test Set')
  ax1.set_title('Residual Graph: SVM Regression')
  ax1.scatter(test_predictions, y_test - test_predictions)
  plt.show()

def run_full_random_forest_regression_with_accuracy(features, response, n_highest = 0):
  print(f"Response Minimum: {response.min()}")
  print(f"Response Maximum: {response.max()}")

```

```
 model = RandomForestRegressor(n_estimators=100)
 # Split into test and training set
 X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
 model.fit(X_train, y_train)

 training_predictions = model.predict(X_train)
 test_predictions = model.predict(X_test)

 print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
 print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))

 print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
 print("Test average error rate: ", median_absolute_error(y_test, test_predictions))

 print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
 print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))

 fig = plt.figure()
 ax1 = fig.add_subplot()
 ax1.set_xlabel("True Value from Test Set")
 ax1.set_ylabel('Prediction from Test Set')
 ax1.set_title('True Value vs Predicted Value for Test Set: Random Forest Regression')
 ax1.scatter(y_test, test_predictions)
 plt.show()

 fig = plt.figure()

 ax1 = fig.add_subplot()
 ax1.set_xlabel("True Value from Training Set")
 ax1.set_ylabel('Prediction from Training Set')
 ax1.set_title('True Value vs Predicted Value for Training Set: Random Forest Regression')
 ax1.scatter(y_train, training_predictions)
 plt.show()

 # Residual plot against predictor
 fig = plt.figure()
 ax1 = fig.add_subplot()
 ax1.set_xlabel("Test Set Prediction")
 ax1.set_ylabel('Residuals from Test Set')
 ax1.set_title('Residual Graph: Random Forest')
 ax1.scatter(test_predictions, y_test - test_predictions)
 plt.show()
```

```
SUBSET = -1
```

Fire Size Prediction using Linear Regression and Ridge Regression

Here, we predict the size of a fire using linear regression

```
FIRE_SIZE = numerical["FIRE_SIZE"]
FIRE_SIZE.hist()
```

```
log_fire_size = pd.DataFrame(np.log(FIRE_SIZE))
log_fire_size = log_fire_size.replace([np.inf, -np.inf], np.nan)
log_fire_size = log_fire_size.fillna(0)
log_fire_size = (log_fire_size - log_fire_size.mean()) / log_fire_size.std()
log_fire_size.hist()
plt.show()

# run_full_linear_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
run_full_ridge_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
```

LINEAR REGRESSION and RIDGE REGRESSION for Summer Months

```
# LINEAR REGRESSION FOR SUMMER MONTHS
log_fire_size = pd.DataFrame(np.log(FIRE_SIZE))
log_fire_size = log_fire_size.replace([np.inf, -np.inf], np.nan)
log_fire_size = log_fire_size.fillna(0)
log_fire_size = (log_fire_size - log_fire_size.mean()) / log_fire_size.std()
log_fire_size.hist()
plt.show()

#x = 11
# x = 6
# x = 1
summer_fires = ((cleaned['DISCOVERY_MONTH'] >= 5) & (cleaned['DISCOVERY_MONTH'] <= 8))
summer_fires_log_size = log_fire_size[summer_fires]
summer_fires_df = fire_size_prediction_df[summer_fires]
print(len(summer_fires_log_size))

run_full_linear_regression_with_accuracy(summer_fires_df, summer_fires_log_size)
run_full_ridge_regression_with_accuracy(summer_fires_df, summer_fires_log_size)
```

Fire Duration Prediction using Linear Regression and Ridge Regression

Here, we predict the duration of a fire using linear regression

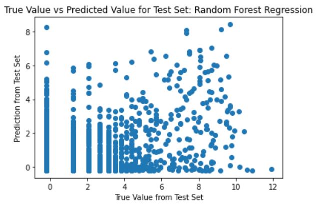
```
FIRE_DURATION = numerical["DURATION"]
FIRE_DURATION.hist()
```

```
log_fire_duration = pd.DataFrame(np.log(FIRE_DURATION))
log_fire_duration = log_fire_duration.replace([np.inf, -np.inf], np.nan)
log_fire_duration = log_fire_duration.fillna(0)
log_fire_duration = (log_fire_duration - log_fire_duration.mean()) / log_fire_duration.std()
log_fire_duration.hist()
plt.show()

```

```
# run_full_linear_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])
run_full_ridge_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])

```

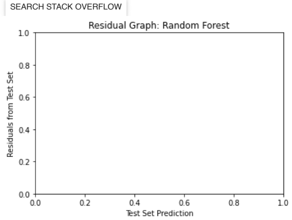
```
run_full_random_forest_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])
```

```
Response Minimum: DURATION   -0.214565
dtype: float64
Response Maximum: DURATION    11.92416
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:154: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
Training: mean absolute error:  0.11754670591465904
Test mean absolute error:  0.32449745615517356
Training average error rate:  1.2490009027033011e-15
Test average error rate:  0.04173831202560002
Explained variance in training set is:  0.8867669076109592
Explained variance in test set is:  0.22634597507339815
```



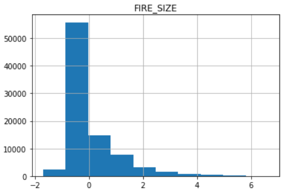

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-31-f1646410162a> in <module>()
----> 1 run_full_random_forest_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])

                                   3 frames
/usr/local/lib/python3.7/dist-packages/pandas/core/ops/__init__.py in to_series(right)
    464             if len(left.columns) != len(right):
    465                 raise ValueError(
--> 466                     msg.format(req_len=len(left.columns), given_len=len(right))
    467                 )
    468             right = left._constructor_sliced(right, index=left.columns)

ValueError: Unable to coerce to Series, length must be 1: given 17421
```

SEARCH STACK OVERFLOW



Random Forest Regression FIRE_SIZE

```
log_fire_size = pd.DataFrame(np.log(FIRE_SIZE))
log_fire_size = log_fire_size.replace([np.inf, -np.inf], np.nan)
log_fire_size = log_fire_size.fillna(0)
log_fire_size = (log_fire_size - log_fire_size.mean()) / log_fire_size.std()
log_fire_size.hist()
plt.show()

# run_full_SVM_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
run_full_random_forest_regression_with_accuracy(fire_size_prediction_df[:-1], log_fire_size[:-1])

```



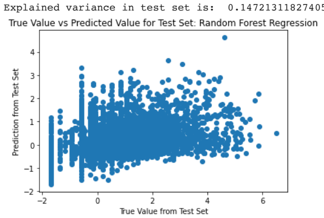

```
Response Minimum: FIRE_SIZE   -1.698422
dtype: float64
Response Maximum: FIRE_SIZE    6.638909
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:173: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-30-f49f4398cab1> in <module>()
      7
      8 # run_full_SVM_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
----> 9 run_full_random_forest_regression_with_accuracy(fire_size_prediction_df[:-1], log_fire_size[:-1])
```

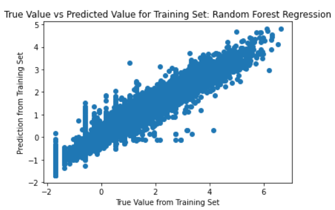
Double-click (or enter) to edit

```
/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py in fit(self, X, y, sample_weight, check_input, X_idx_sorted)
```

```
# Random forests for summer months
run_full_random_forest_regression_with_accuracy(summer_fires_df, summer_fires_log_size)
```

```
Response Minimum: FIRE_SIZE   -1.698422
dtype: float64
Response Maximum: FIRE_SIZE    6.638909
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:154: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
Training: mean absolute error:  0.25446062326206703
Test mean absolute error:  0.6681863760662583
Training average error rate:  0.1714524020813536
Test average error rate:  0.4603202742169611
Explained variance in training set is:  0.8693374232241285
Explained variance in test set is:  0.14721311827405048
```





Model 2: Random Forest Classifier

```
# Model 2:
stat = numerical['STAT_CAUSE_CODE']
stat.value_counts()

cleaned_stat1 = numerical[numerical.STAT_CAUSE_CODE != 13.0]
cleaned_stat = cleaned_stat1[cleaned_stat1.STAT_CAUSE_CODE != 9.0]

col_index = cleaned_stat.columns.get_loc('STAT_CAUSE_CODE')
#print(col_index)
cleaned_stat.head()

# use either a random forest or neural network to predict the
```

```
from sklearn.ensemble import RandomForestClassifier
# Instantiate model with 135 decision trees (figured out in the next few blocks of code that this is optimal)
forest = RandomForestClassifier(n_estimators = 135, oob_score = True)
```

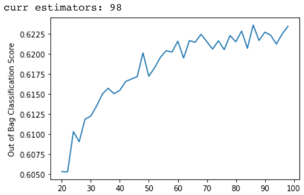
-0.155696 -0.232580 0.429544 0.364579 0.480207

```
x_train_mod2 = np.array(cleaned_stat.iloc[:, col_index+1:])
y_mod2 = cleaned_stat.iloc[:, [col_index]]
y_train_mod2 = np.array(y_mod2)

forest.fit(x_train_mod2, y_train_mod2.ravel())
print(forest.score(x_train_mod2, y_train_mod2))
print(forest.oob_score_)
```

```
1.0
0.6247818786422151
```

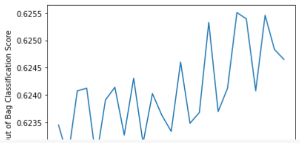
```
oob_list = []

for i in range(20, 100, 2):
    print("curr estimators:", i)
    forest = RandomForestClassifier(n_estimators = i, oob_score = True)
    forest.fit(x_train_mod2, y_train_mod2.ravel())
    oob_list.append(forest.oob_score_)
plt.plot(list(range(20, 100, 2)), oob_list)
plt.xlabel('# Trees')
plt.ylabel('Out of Bag Classification Score')
plt.show()
```

```
curr estimators: 20
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 22
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 24
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 26
curr estimators: 28
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 30
curr estimators: 32
curr estimators: 34
curr estimators: 36
curr estimators: 38
curr estimators: 40
curr estimators: 42
curr estimators: 44
curr estimators: 46
curr estimators: 48
curr estimators: 50
curr estimators: 52
curr estimators: 54
curr estimators: 56
curr estimators: 58
curr estimators: 60
curr estimators: 62
curr estimators: 64
curr estimators: 66
curr estimators: 68
curr estimators: 70
curr estimators: 72
curr estimators: 74
curr estimators: 76
curr estimators: 78
curr estimators: 80
curr estimators: 82
curr estimators: 84
curr estimators: 86
curr estimators: 88
curr estimators: 90
curr estimators: 92
curr estimators: 94
curr estimators: 96
curr estimators: 98
```



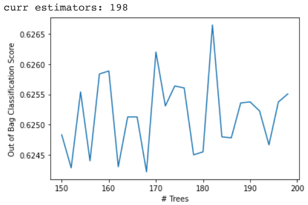

```
oob_list2 = []

for i in range(100, 150, 2):
    print("curr estimators:", i)
    forest = RandomForestClassifier(n_estimators = i, oob_score = True)
    forest.fit(x_train_mod2, y_train_mod2.ravel())
    oob_list2.append(forest.oob_score_)
plt.plot(list(range(100, 150, 2)), oob_list2)
plt.xlabel('# Trees')
plt.ylabel('Out of Bag Classification Score')
plt.show()
```

```
curr estimators: 100
curr estimators: 102
curr estimators: 104
curr estimators: 106
curr estimators: 108
curr estimators: 110
curr estimators: 112
curr estimators: 114
curr estimators: 116
curr estimators: 118
curr estimators: 120
curr estimators: 122
curr estimators: 124
curr estimators: 126
curr estimators: 128
curr estimators: 130
curr estimators: 132
curr estimators: 134
curr estimators: 136
curr estimators: 138
curr estimators: 140
curr estimators: 142
curr estimators: 144
curr estimators: 146
curr estimators: 148
```




```
oob_list3 = []

for i in range(150, 200, 2):
    print("curr estimators:", i)
    forest = RandomForestClassifier(n_estimators = i, oob_score = True)
    forest.fit(x_train_mod2, y_train_mod2.ravel())
    oob_list3.append(forest.oob_score_)
plt.plot(list(range(150, 200, 2)), oob_list3)
plt.xlabel('# Trees')
plt.ylabel('Out of Bag Classification Score')
plt.show()
```

```
curr estimators: 150
curr estimators: 152
curr estimators: 154
curr estimators: 156
curr estimators: 158
curr estimators: 160
curr estimators: 162
curr estimators: 164
curr estimators: 166
curr estimators: 168
curr estimators: 170
curr estimators: 172
curr estimators: 174
curr estimators: 176
curr estimators: 178
curr estimators: 180
curr estimators: 182
curr estimators: 184
curr estimators: 186
curr estimators: 188
curr estimators: 190
curr estimators: 192
curr estimators: 194
curr estimators: 196
curr estimators: 198
```




```
forest = RandomForestClassifier(n_estimators = 182, oob_score = True)
forest.fit(x_train_mod2, y_train_mod2.ravel())
val_list = forest.feature_importances_
idx_list = np.argsort(val_list)[::-1]

print('From high to low:')
for idx in idx_list:
    print('Feature %d: %f' % (idx, val_list[idx]))
```

```
From high to low:
Feature 0: 0.215777
Feature 4: 0.187571
Feature 3: 0.174209
Feature 1: 0.166072
Feature 2: 0.162683
Feature 5: 0.050670
Feature 6: 0.043017
```

```
x_train_mod3 = np.array(cleaned_stat.iloc[:, col_index+1:])
y_mod3 = cleaned_stat.iloc[:, [col_index]]
y_train_mod3 = np.array(y_mod3)
```

```
forest.fit(x_train_mod3, y_train_mod3.ravel())
print(forest.score(x_train_mod3, y_train_mod3))
print(forest.oob_score_)
```

```
1.0
0.6264445395581603
```

```
forest = RandomForestClassifier(n_estimators = 182, oob_score = True)

x_train_mod2 = np.array(cleaned_stat.iloc[:, col_index+1:])
y_mod2 = cleaned_stat.iloc[:, [col_index]]
y_train_mod2 = np.array(y_mod2)

X_train, X_test, y_train, y_test = train_test_split(cleaned_stat, y_train_mod2, test_size=0.2, random_state=2020)

forest.fit(X_train, y_train.ravel())
print(forest.score(X_test, y_test))
print(forest.oob_score_)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  "X does not have valid feature names, but"
0.8585185185185186
0.8519837023623343
```