PREDICTING HOUSE PRICES

AMES HOUSING DATA

SUBMITTED BY:TEAM 3B

DATA SUMMARY

Training Data

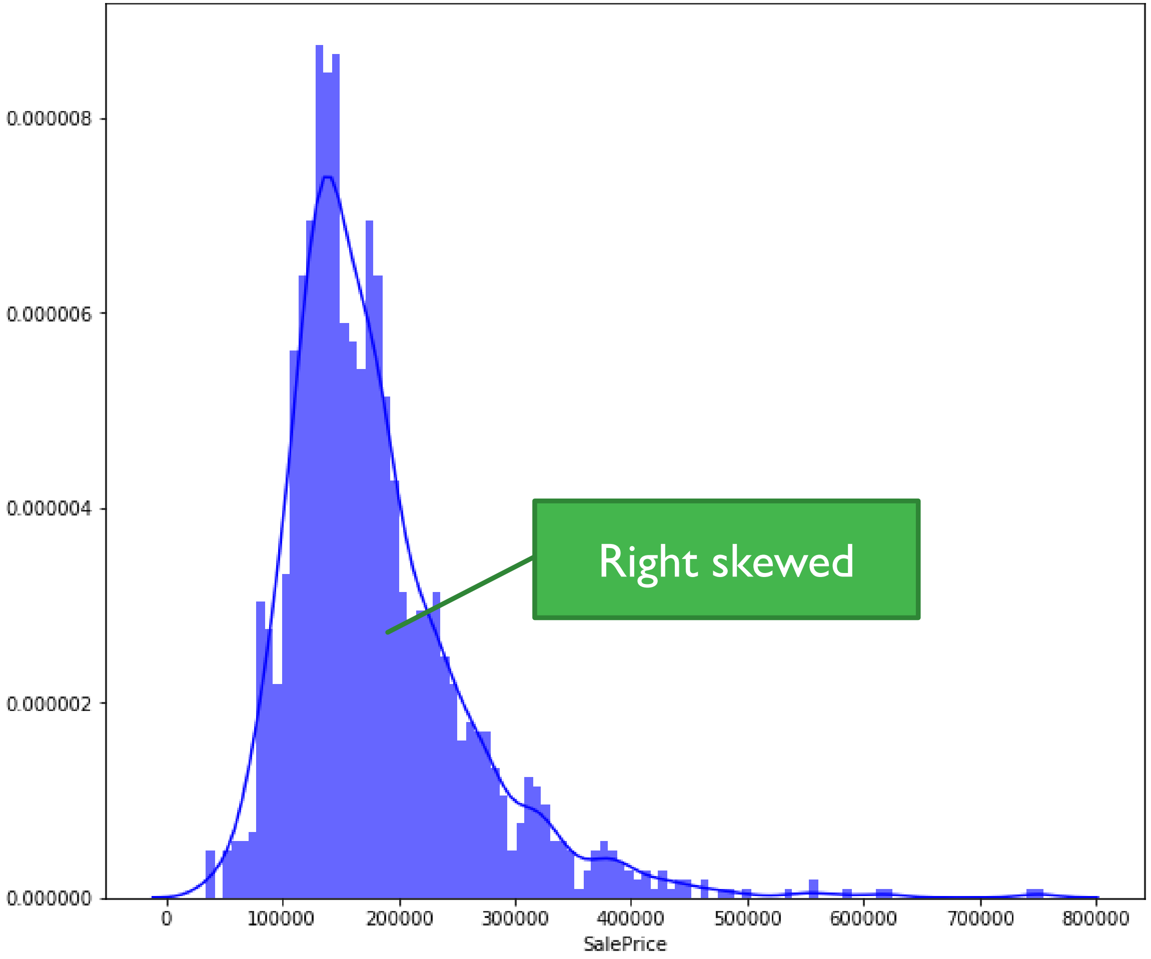
- 1460 data points
- 81 features
- Includes Sales price column

Test Data

- 1459 data points
- 80 features
- No Sales price
  - Need to predict

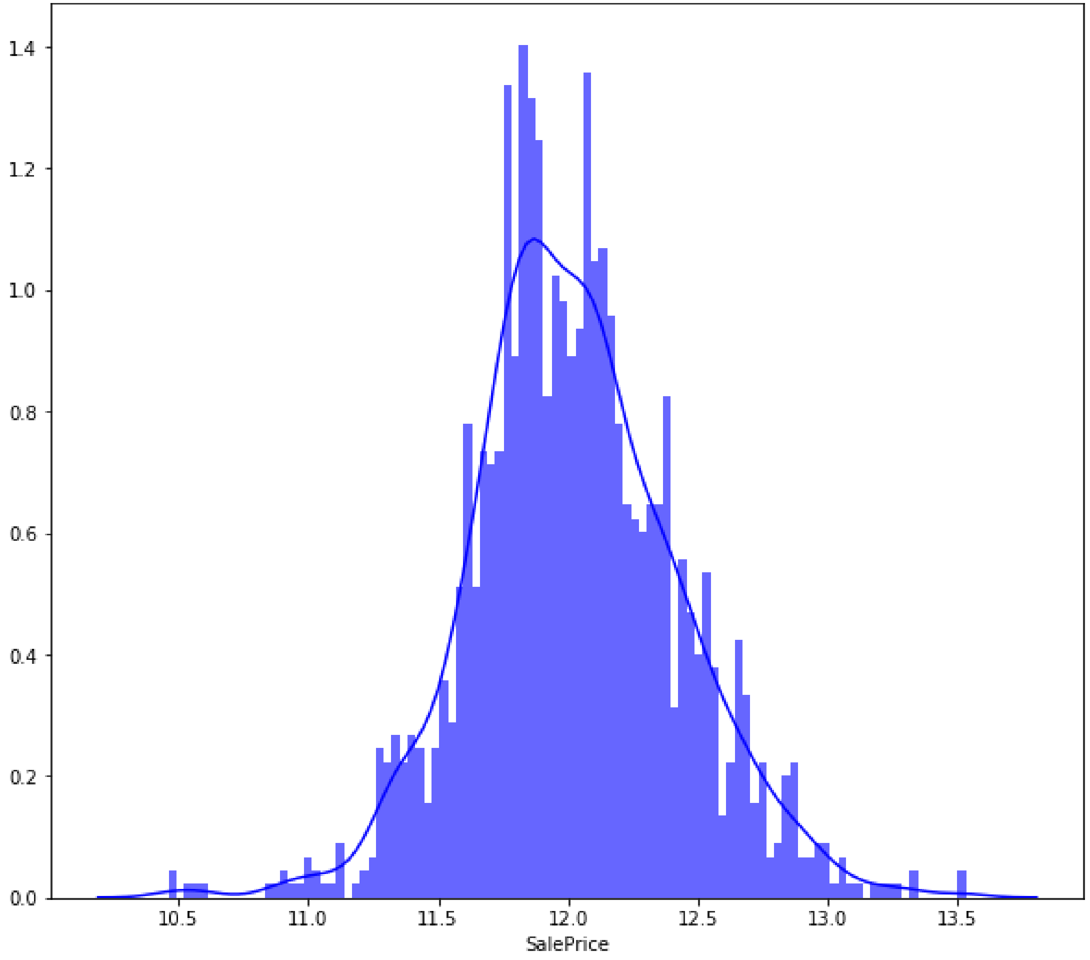
EDA– TARGET VARIABLE

Sales price



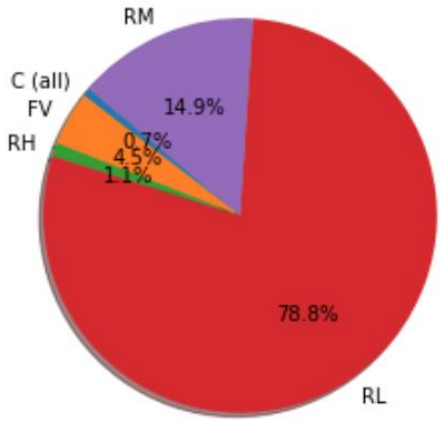
Right skewed

Log Sales price

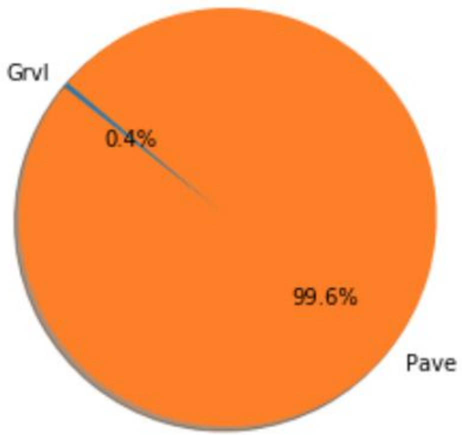


EDA - CATEGORICAL VARIABLES

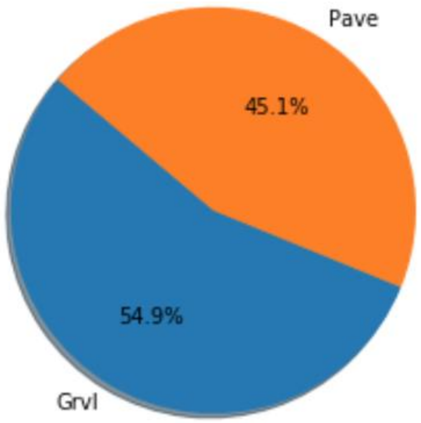
MS ZONING



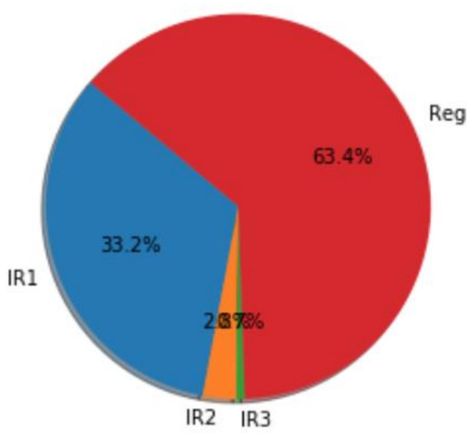
STREET



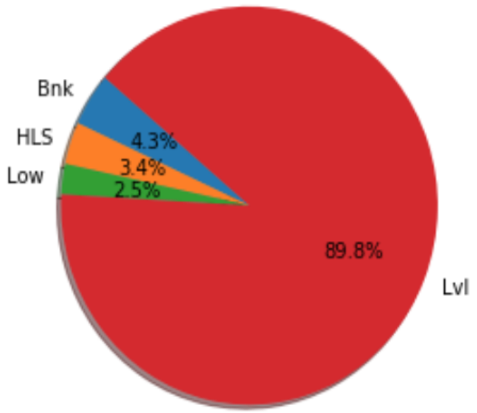
ALLEY



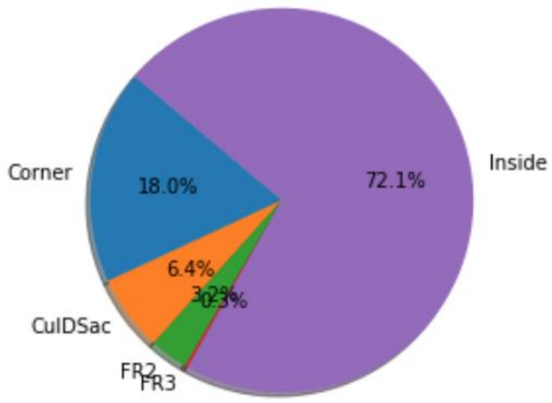
LOTSHAPE



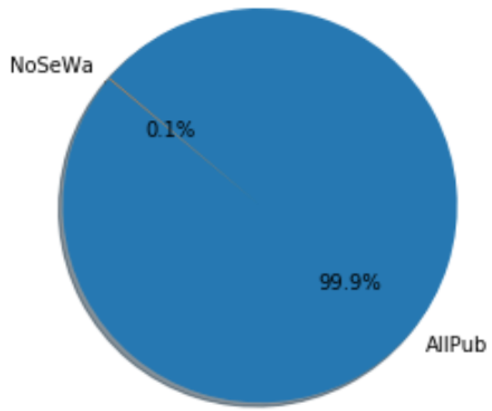
LANDCONTOUR



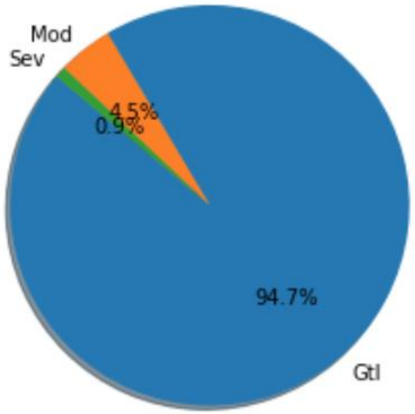
LOTCONFIG



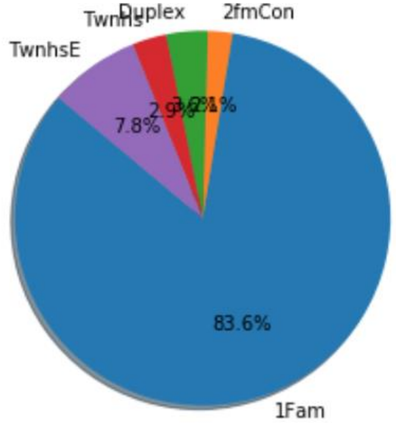
UTILITIES



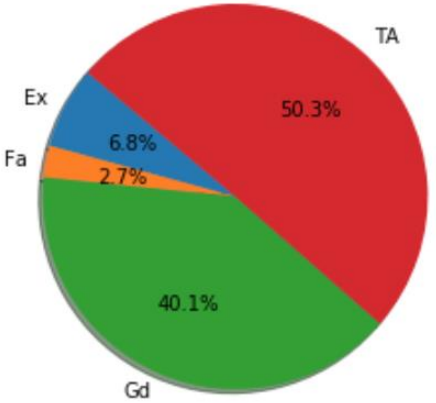
LANDSLOPE



BLDGTYPE



KITCHEN QUAL



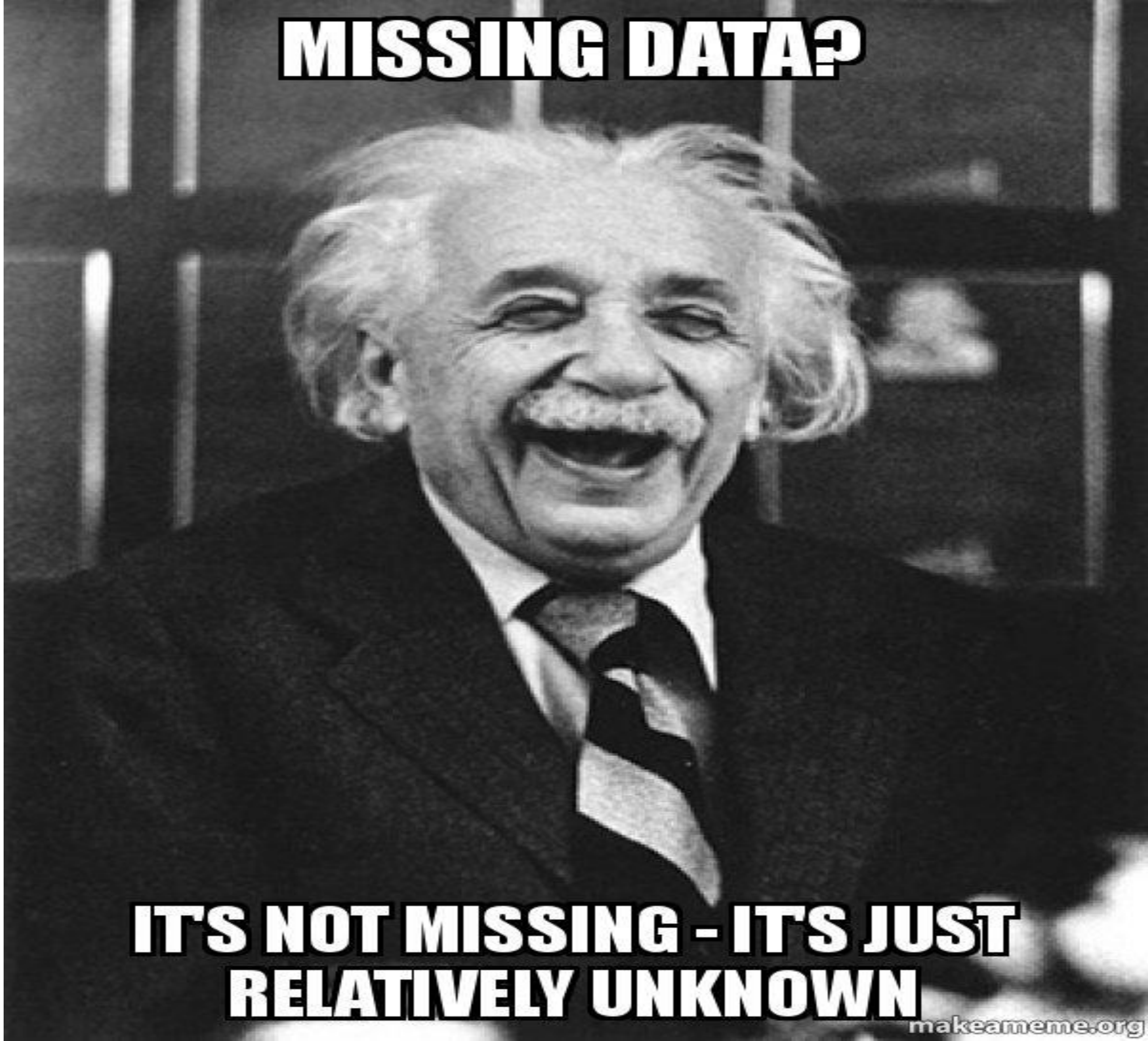
MISSING DATA

TRAIN DATA

6965

TEST DATA

7000



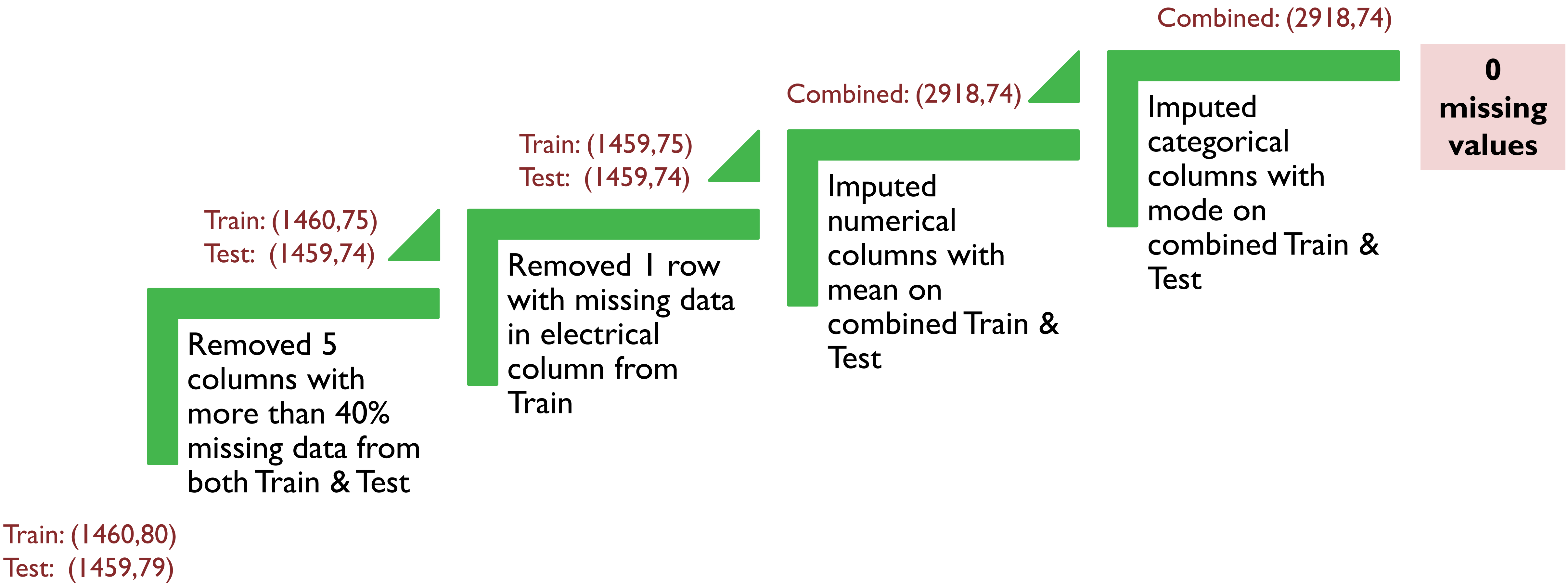
MISSING TRAIN DATA

Feature	# of data points missing	% of data missing
PoolQC	1453	99.52
MiscFeature	1406	96.3
Alley	1369	93.76
Fence	1179	80.75
FireplaceQu	690	47.26
LotFrontage	259	17.73
GarageYrBlt	81	5.54
GarageType	81	5.54
GarageFinish	81	5.54
GarageQual	81	5.54
GarageCond	81	5.54
BsmtFinType2	38	2.6
BsmtExposure	38	2.6
BsmtFinType1	37	2.53
BsmtCond	37	2.53
BsmtQual	37	2.53
MasVnrArea	8	0.54
MasVnrType	8	0.54
Electrical	1	0.06

MISSING TEST DATA

Feature	# of data points missing	% of data missing
PoolQC	1456	99.79
MiscFeature	1408	96.5
Alley	1352	92.66
Fence	1169	80.12
FireplaceQu	730	50.03
LotFrontage	227	15.5
GarageYrBlt	78	5.34
GarageCond	78	5.34
GarageQual	78	5.34
GarageFinish	78	5.34
GarageType	76	5.20
BsmtCond	45	3.08
BsmtExposure	44	3.01
BsmtQual	44	3.01
BsmtFinType1	42	2.87
BsmtFinType2	42	2.87
MasVnrType	16	1.09
MasVnrArea	15	1.02
Others		

MISSING VALUE TREATMENT



Train: (1460,80)
Test: (1459,79)

Removed 5 columns with more than 40% missing data from both Train & Test

Train: (1460,75)
Test: (1459,74)

Removed 1 row with missing data in electrical column from Train

Train: (1459,75)
Test: (1459,74)

Imputed numerical columns with mean on combined Train & Test

Combined: (2918,74)

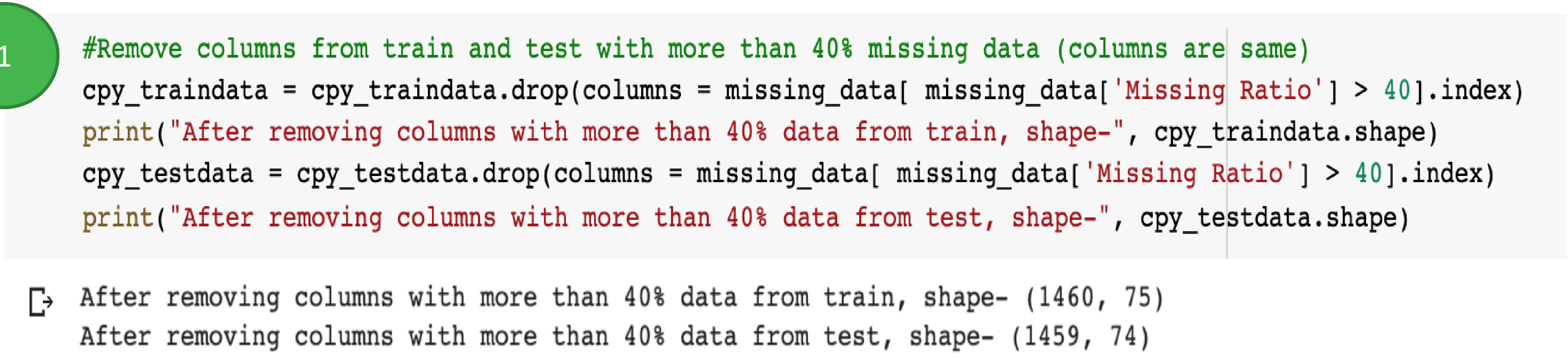
Imputed categorical columns with mode on combined Train & Test

Combined: (2918,74)

0 missing values

MISSING VALUE TREATMENT CODE

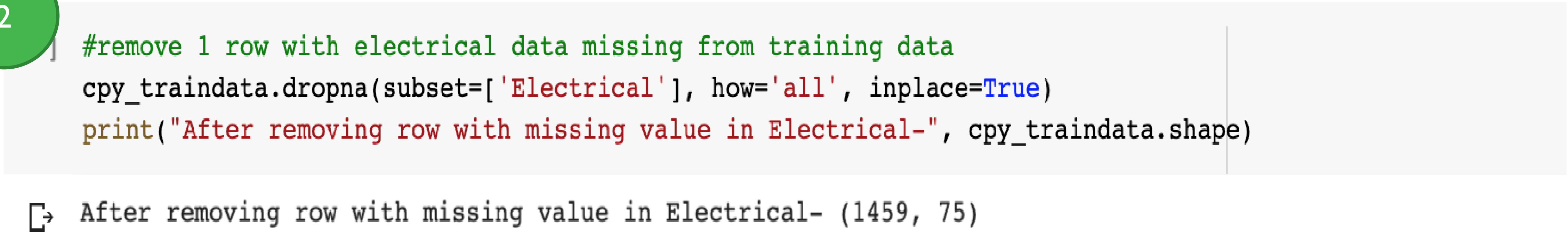
1



```
#Remove columns from train and test with more than 40% missing data (columns are same)
cpy_traindata = cpy_traindata.drop(columns = missing_data[ missing_data['Missing Ratio'] > 40].index)
print("After removing columns with more than 40% data from train, shape-", cpy_traindata.shape)
cpy_testdata = cpy_testdata.drop(columns = missing_data[ missing_data['Missing Ratio'] > 40].index)
print("After removing columns with more than 40% data from test, shape-", cpy_testdata.shape)
```

```
After removing columns with more than 40% data from train, shape- (1460, 75)
After removing columns with more than 40% data from test, shape- (1459, 74)
```

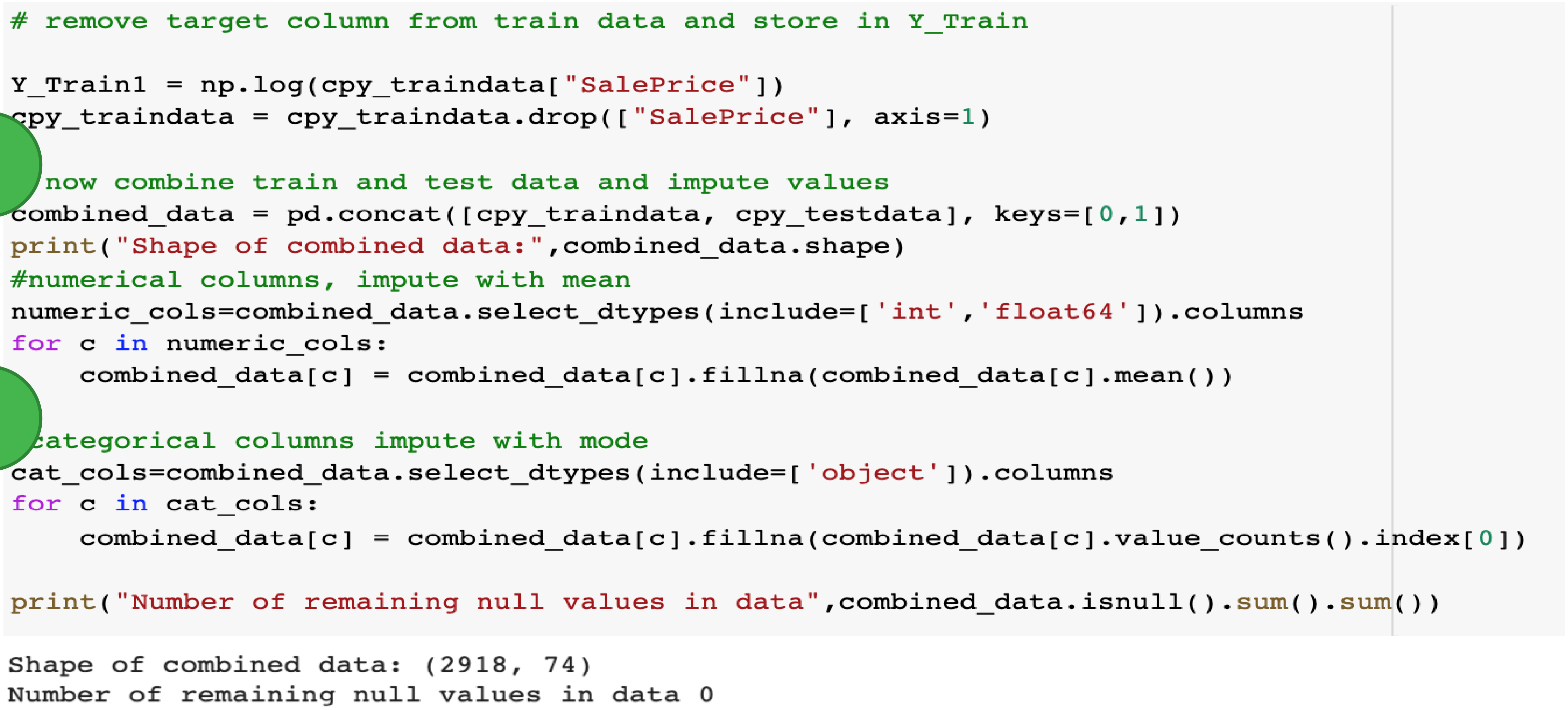
2



```
#remove 1 row with electrical data missing from training data
cpy_traindata.dropna(subset=['Electrical'], how='all', inplace=True)
print("After removing row with missing value in Electrical-", cpy_traindata.shape)
```

```
After removing row with missing value in Electrical- (1459, 75)
```

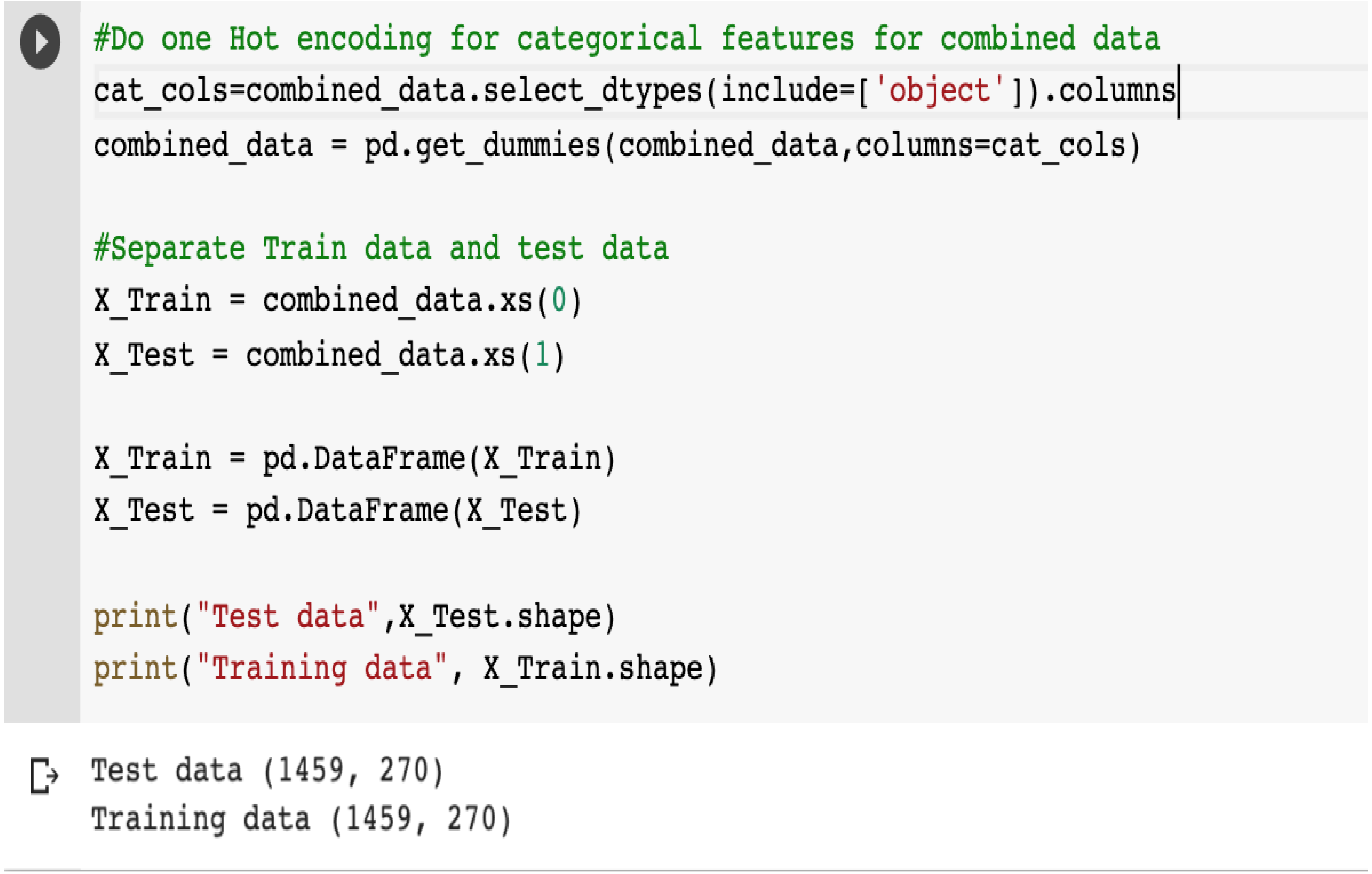
MISSING VALUE TREATMENT CODE



```
# remove target column from train data and store in Y_Train

Y_Train1 = np.log(cpy_traindata["SalePrice"])
cpy_traindata = cpy_traindata.drop(["SalePrice"], axis=1)

# now combine train and test data and impute values
combined_data = pd.concat([cpy_traindata, cpy_testdata], keys=[0,1])
print("Shape of combined data:",combined_data.shape)
#numerical columns, impute with mean
numeric_cols=combined_data.select_dtypes(include=['int','float64']).columns
for c in numeric_cols:
    combined_data[c] = combined_data[c].fillna(combined_data[c].mean())

#categorical columns impute with mode
cat_cols=combined_data.select_dtypes(include=['object']).columns
for c in cat_cols:
    combined_data[c] = combined_data[c].fillna(combined_data[c].value_counts().index[0])

print("Number of remaining null values in data",combined_data.isnull().sum().sum())
```

```
Shape of combined data: (2918, 74)
Number of remaining null values in data 0
```

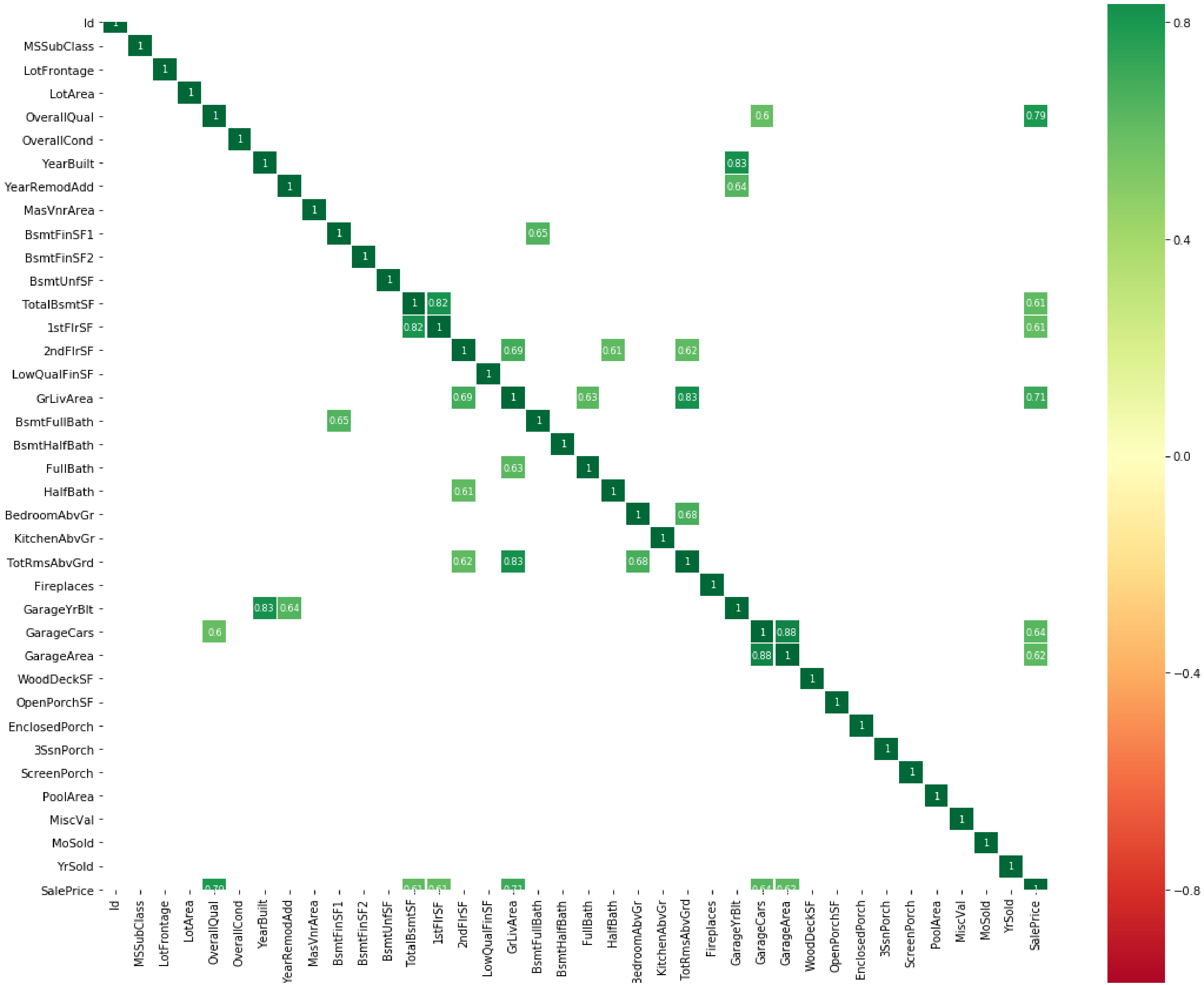
ONE HOT ENCODING



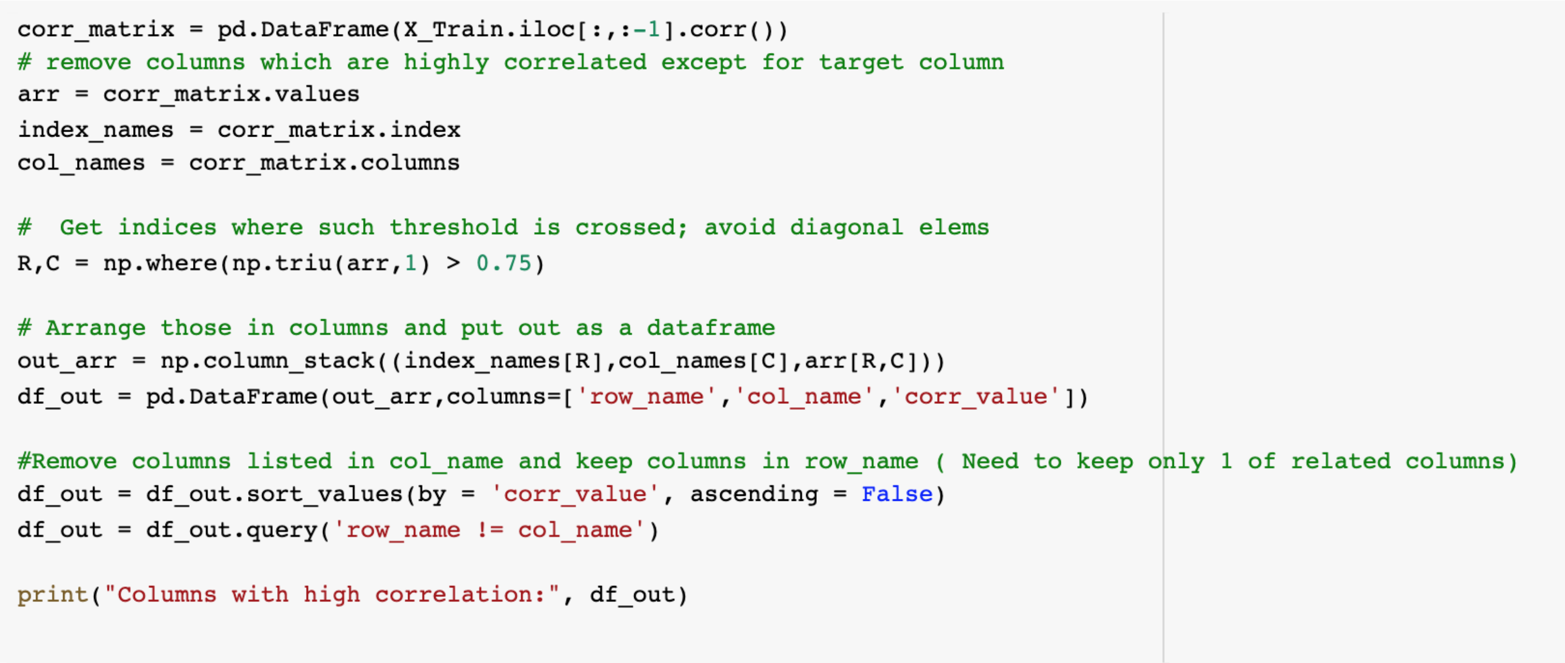
```
#Do one Hot encoding for categorical features for combined data
cat_cols=combined_data.select_dtypes(include=['object']).columns
combined_data = pd.get_dummies(combined_data,columns=cat_cols)

#Separate Train data and test data
X_Train = combined_data.xs(0)
X_Test = combined_data.xs(1)

X_Train = pd.DataFrame(X_Train)
X_Test = pd.DataFrame(X_Test)

print("Test data",X_Test.shape)
print("Training data", X_Train.shape)
```

```
Test data (1459, 270)
Training data (1459, 270)
```

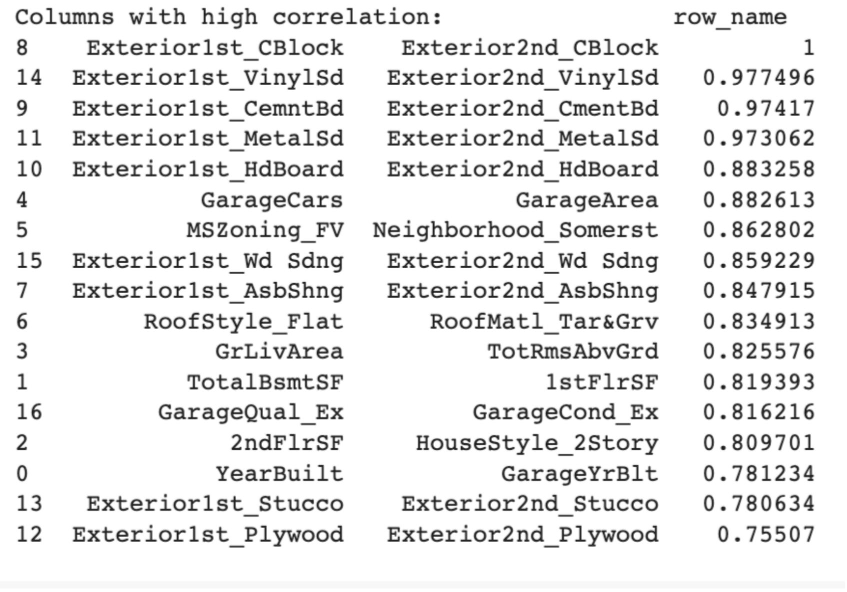
REMOVED 17 HIGHLY CORRELATED FEATURES



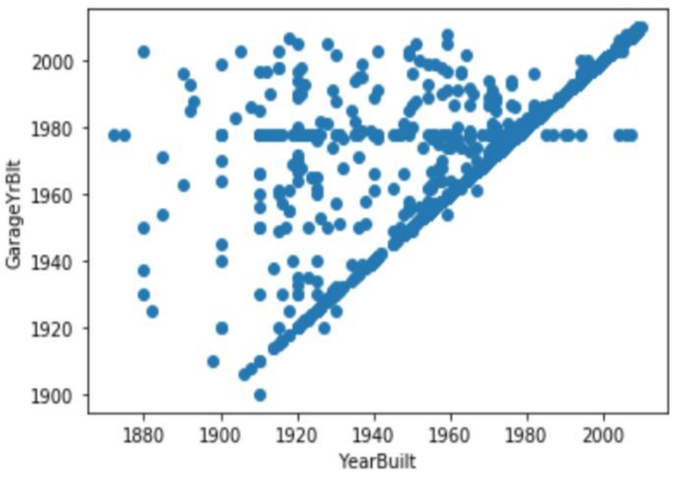
FINDING CORRELATED COLUMNS



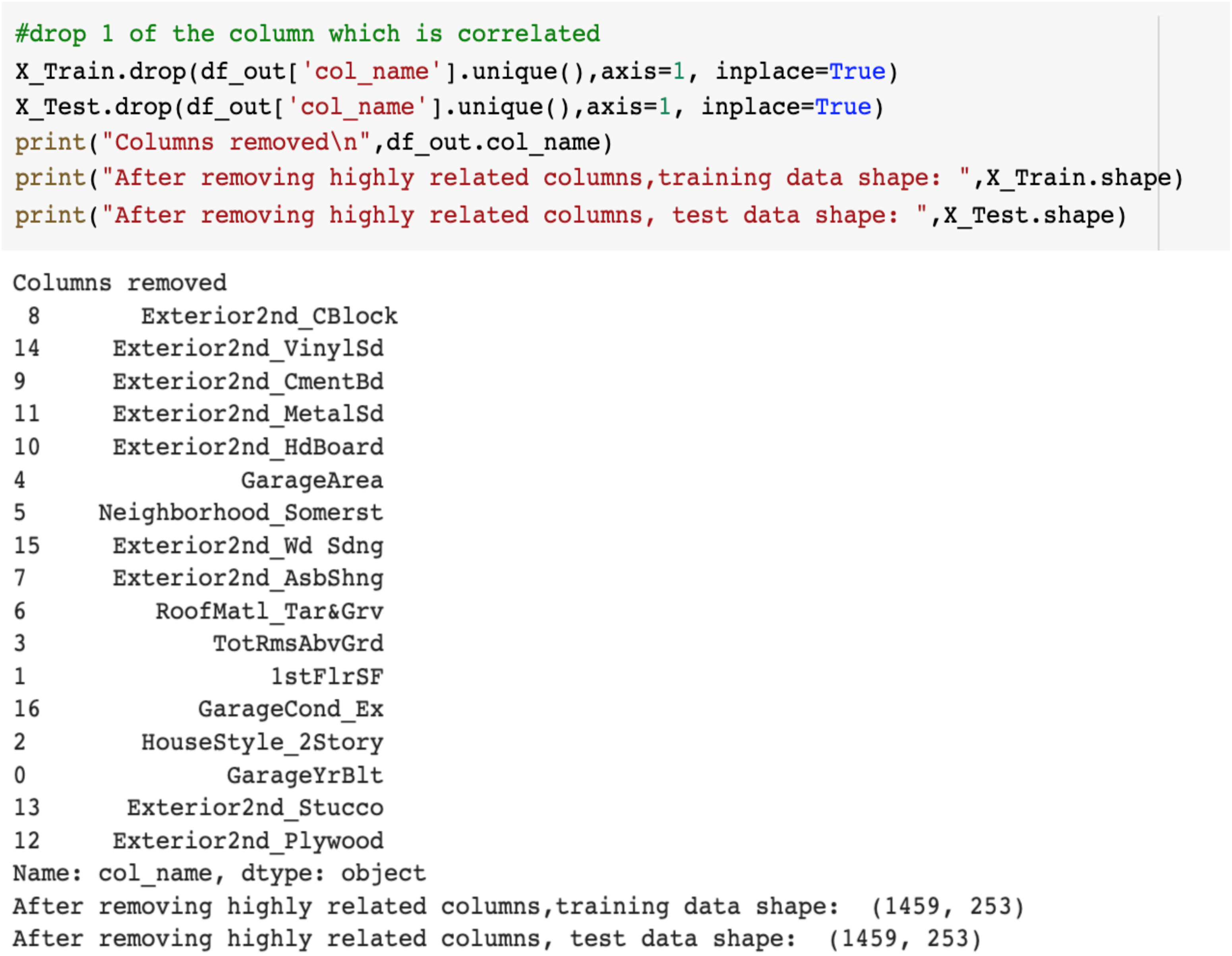
```
corr_matrix = pd.DataFrame(X_Train.iloc[:,:-1].corr())
# remove columns which are highly correlated except for target column
arr = corr_matrix.values
index_names = corr_matrix.index
col_names = corr_matrix.columns

#  Get indices where such threshold is crossed; avoid diagonal elems
R,C = np.where(np.triu(arr,1) > 0.75)

# Arrange those in columns and put out as a dataframe
out_arr = np.column_stack((index_names[R],col_names[C],arr[R,C]))
df_out = pd.DataFrame(out_arr,columns=['row_name','col_name','corr_value'])

#Remove columns listed in col_name and keep columns in row_name ( Need to keep only 1 of related columns)
df_out = df_out.sort_values(by = 'corr_value', ascending = False)
df_out = df_out.query('row_name != col_name')

print("Columns with high correlation:", df_out)
```



```
Columns with high correlation:                 row_name                col_name corr_value
8     Exterior1st_CBlock    Exterior2nd_CBlock          1
14  Exterior1st_VinylSd   Exterior2nd_VinylSd   0.977496
9   Exterior1st_CemntBd   Exterior2nd_CmentBd    0.97417
11  Exterior1st_MetalSd   Exterior2nd_MetalSd   0.973062
10  Exterior1st_HdBoard   Exterior2nd_HdBoard   0.883258
4            GarageCars            GarageArea   0.882613
5           MSZoning_FV  Neighborhood_Somerst   0.862802
15  Exterior1st_Wd Sdng   Exterior2nd_Wd Sdng   0.859229
7   Exterior1st_AsbShng   Exterior2nd_AsbShng   0.847915
6        RoofStyle_Flat      RoofMatl_Tar&Grv   0.834913
3             GrLivArea          TotRmsAbvGrd   0.825576
1           TotalBsmtSF              1stFlrSF   0.819393
16        GarageQual_Ex         GarageCond_Ex   0.816216
2              2ndFlrSF     HouseStyle_2Story   0.809701
0             YearBuilt           GarageYrBlt   0.781234
13   Exterior1st_Stucco    Exterior2nd_Stucco   0.780634
12  Exterior1st_Plywood   Exterior2nd_Plywood    0.75507
```

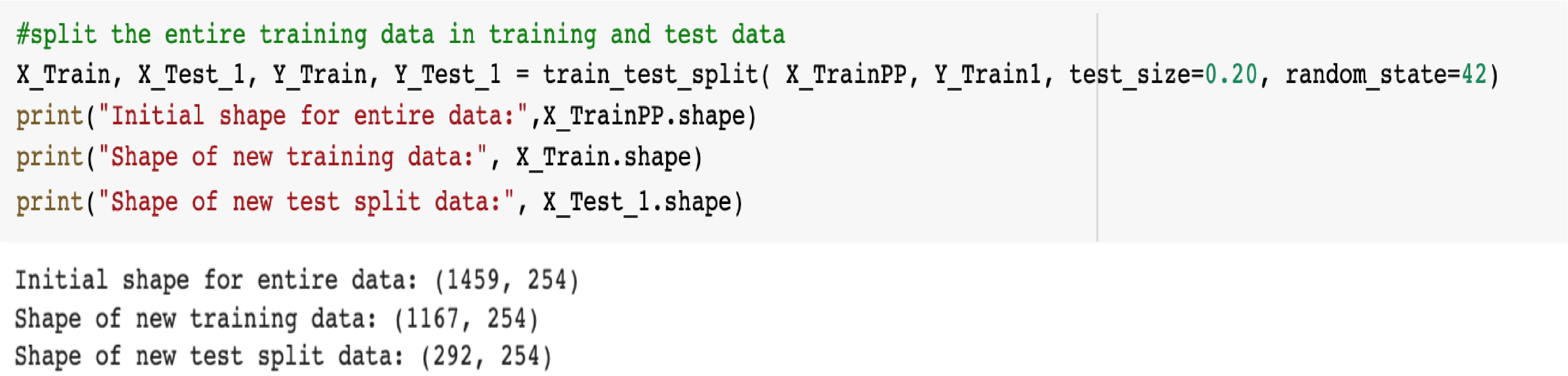


REMOVING 1 CORRELATED COLUMNS



```
#drop 1 of the column which is correlated
X_Train.drop(df_out['col_name'].unique(),axis=1, inplace=True)
X_Test.drop(df_out['col_name'].unique(),axis=1, inplace=True)
print("Columns removed\n",df_out.col_name)
print("After removing highly related columns,training data shape: ",X_Train.shape)
print("After removing highly related columns, test data shape: ",X_Test.shape)
```

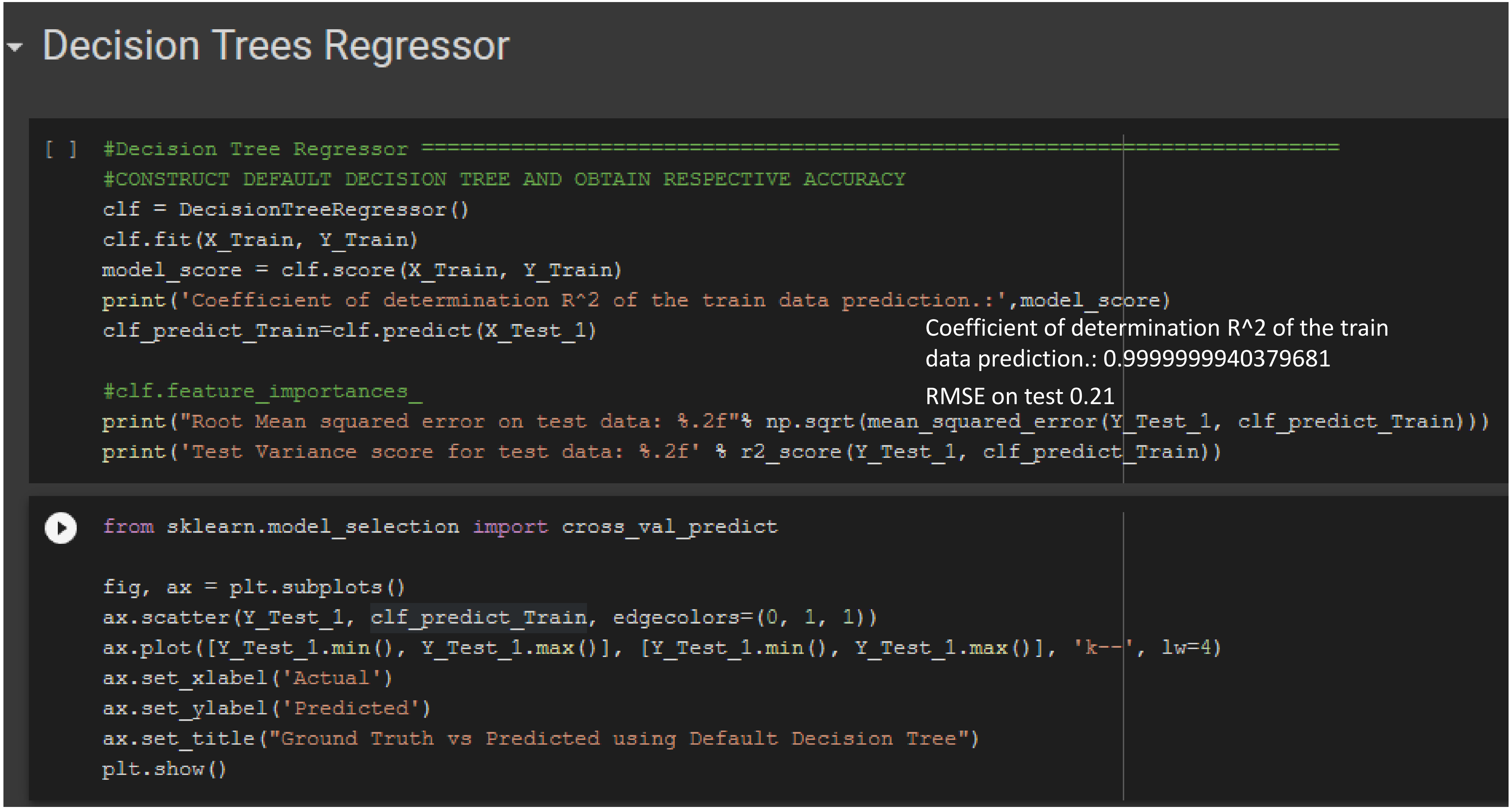
```
Columns removed
 8        Exterior2nd_CBlock
14       Exterior2nd_VinylSd
9        Exterior2nd_CmentBd
11       Exterior2nd_MetalSd
10       Exterior2nd_HdBoard
4                 GarageArea
5       Neighborhood_Somerst
15       Exterior2nd_Wd Sdng
7        Exterior2nd_AsbShng
6           RoofMatl_Tar&Grv
3               TotRmsAbvGrd
1                   1stFlrSF
16             GarageCond_Ex
2          HouseStyle_2Story
0                GarageYrBlt
13        Exterior2nd_Stucco
12       Exterior2nd_Plywood
Name: col_name, dtype: object
After removing highly related columns,training data shape:  (1459, 253)
After removing highly related columns, test data shape:  (1459, 253)
```



```
#split the entire training data in training and test data
X_Train, X_Test_1, Y_Train, Y_Test_1 = train_test_split( X_TrainPP, Y_Train1, test_size=0.20, random_state=42)
print("Initial shape for entire data:",X_TrainPP.shape)
print("Shape of new training data:", X_Train.shape)
print("Shape of new test split data:", X_Test_1.shape)
```

```
Initial shape for entire data: (1459, 254)
Shape of new training data: (1167, 254)
Shape of new test split data: (292, 254)
```

TRAIN TEST SPLIT – 80:20



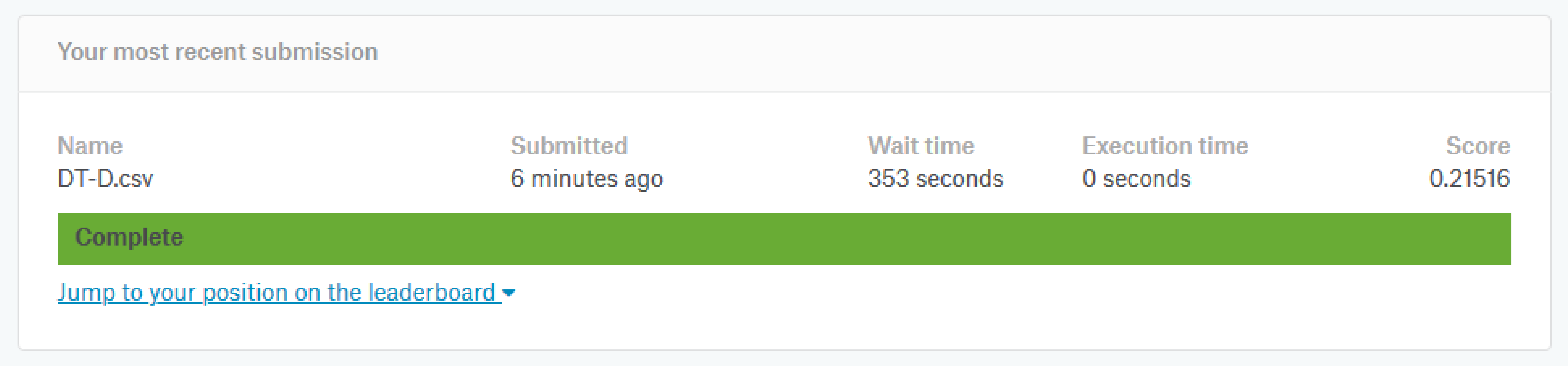
Decision Trees Regressor

```
#Decision Tree Regressor ==========================================================================
#CONSTRUCT DEFAULT DECISION TREE AND OBTAIN RESPECTIVE ACCURACY
clf = DecisionTreeRegressor()
clf.fit(X_Train, Y_Train)
model_score = clf.score(X_Train, Y_Train)
print('Coefficient of determination R^2 of the train data prediction.:',model_score)
clf_predict_Train=clf.predict(X_Test_1)

#clf.feature_importances_
print("Root Mean squared error on test data: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, clf_predict_Train)))
print('Test Variance score for test data: %.2f' % r2_score(Y_Test_1, clf_predict_Train))
```

Coefficient of determination R^2 of the train data prediction.: 0.9999999940379681

RMSE on test 0.21

```
from sklearn.model_selection import cross_val_predict

fig, ax = plt.subplots()
ax.scatter(Y_Test_1, clf_predict_Train, edgecolors=(0, 1, 1))
ax.plot([Y_Test_1.min(), Y_Test_1.max()], [Y_Test_1.min(), Y_Test_1.max()], 'k--', lw=4)
ax.set_xlabel('Actual')
ax.set_ylabel('Predicted')
ax.set_title("Ground Truth vs Predicted using Default Decision Tree")
plt.show()
```



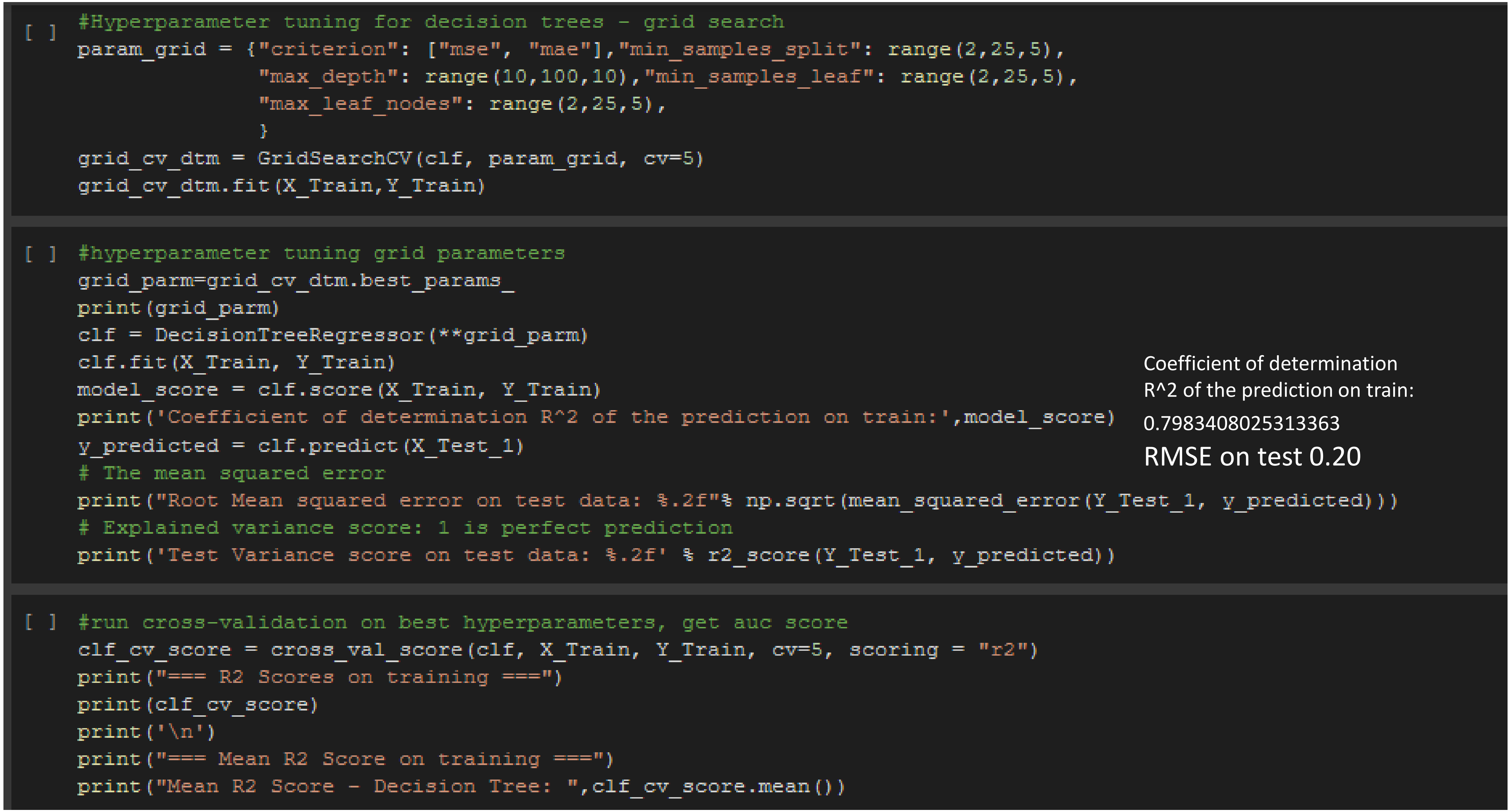
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
DT-D.csv	6 minutes ago	353 seconds	0 seconds	0.21516

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE



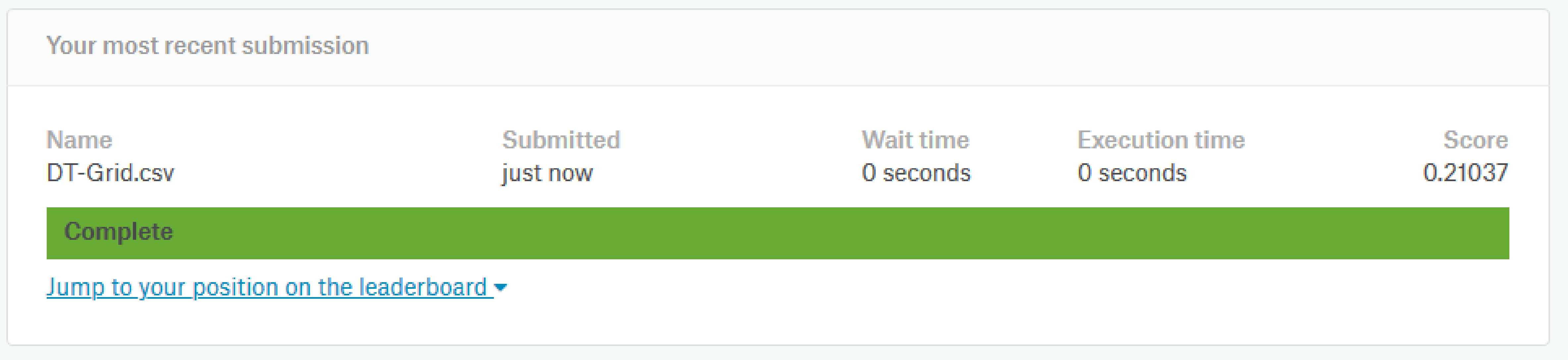
```
[ ]  #Hyperparameter tuning for decision trees - grid search
     param_grid = {"criterion": ["mse", "mae"],"min_samples_split": range(2,25,5),
                   "max_depth": range(10,100,10),"min_samples_leaf": range(2,25,5),
                   "max_leaf_nodes": range(2,25,5),
                   }
     grid_cv_dtm = GridSearchCV(clf, param_grid, cv=5)
     grid_cv_dtm.fit(X_Train,Y_Train)
```

```
[ ]  #hyperparameter tuning grid parameters
     grid_parm=grid_cv_dtm.best_params_
     print(grid_parm)
     clf = DecisionTreeRegressor(**grid_parm)
     clf.fit(X_Train, Y_Train)
     model_score = clf.score(X_Train, Y_Train)
     print('Coefficient of determination R^2 of the prediction on train:',model_score)
     y_predicted = clf.predict(X_Test_1)
     # The mean squared error
     print("Root Mean squared error on test data: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
     # Explained variance score: 1 is perfect prediction
     print('Test Variance score on test data: %.2f' % r2_score(Y_Test_1, y_predicted))
```

```
[ ]  #run cross-validation on best hyperparameters, get auc score
     clf_cv_score = cross_val_score(clf, X_Train, Y_Train, cv=5, scoring = "r2")
     print("=== R2 Scores on training ===")
     print(clf_cv_score)
     print('\n')
     print("=== Mean R2 Score on training ===")
     print("Mean R2 Score - Decision Tree: ",clf_cv_score.mean())
```

Coefficient of determination
R^2 of the prediction on train:
0.7983408025313363

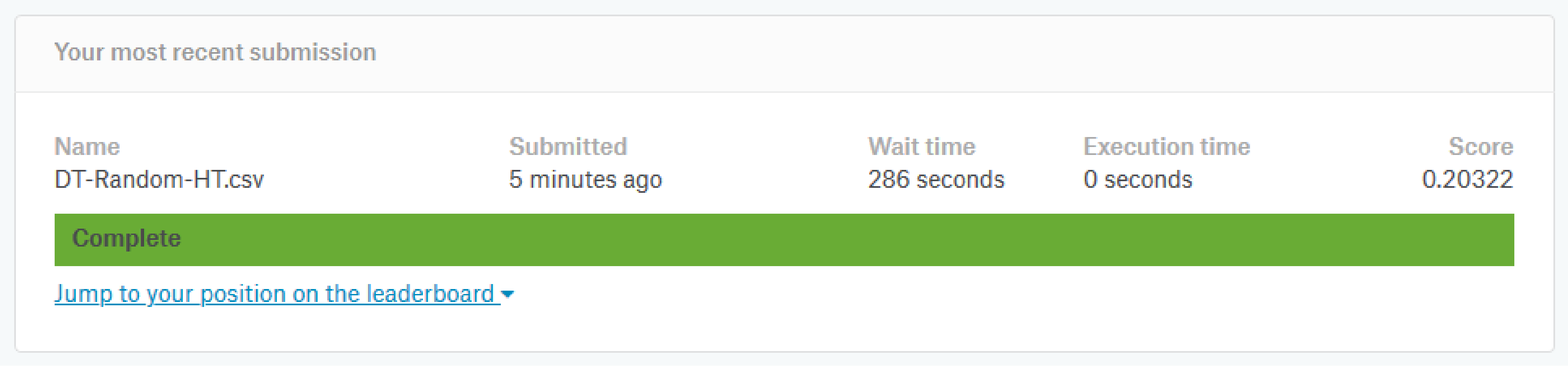
RMSE on test 0.20



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
DT-Grid.csv	just now	0 seconds	0 seconds	0.21037

KAGGLE SCORE

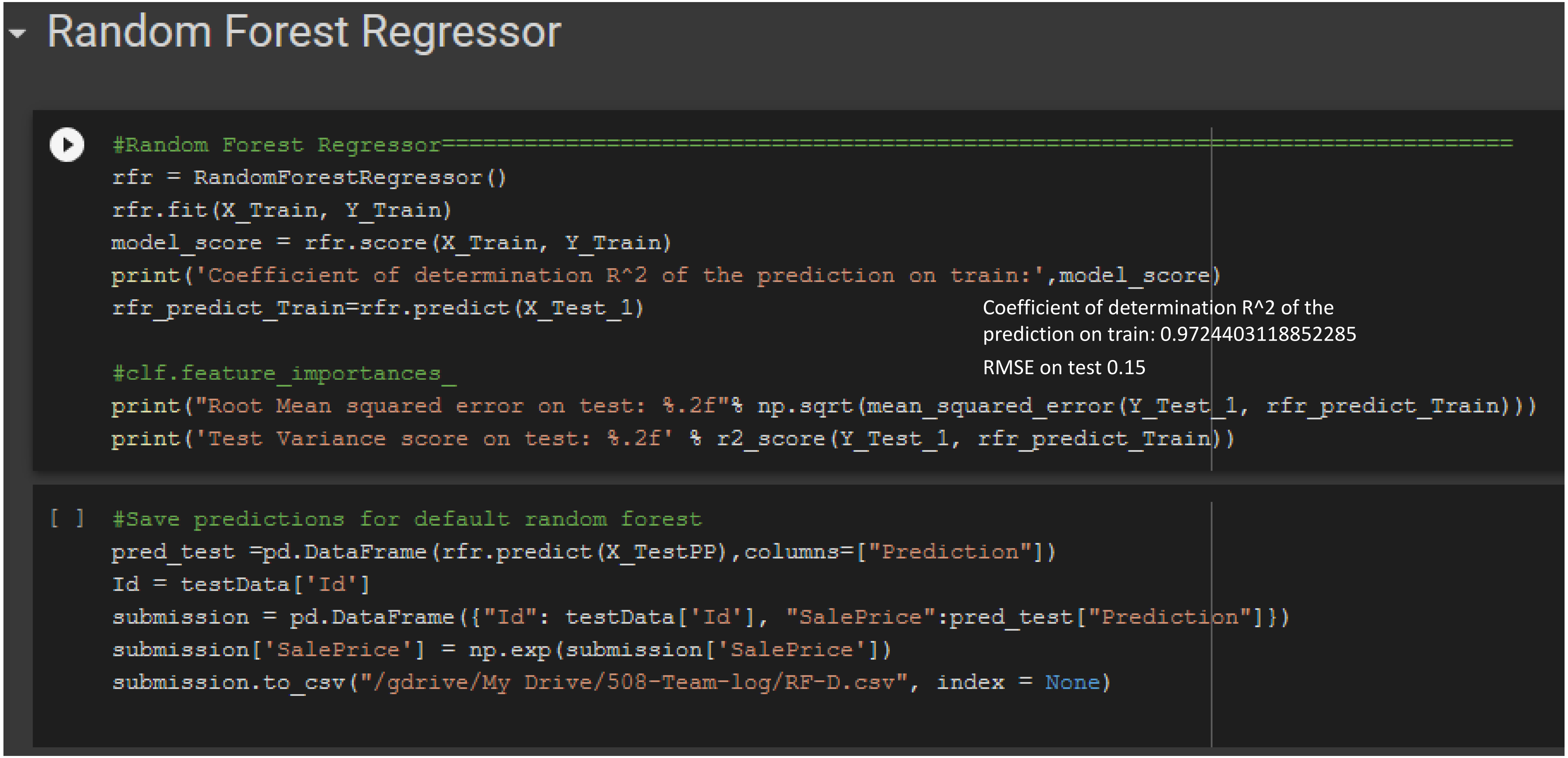


Your most recent submission

Name	Submitted	Wait time	Execution time	Score
DT-Random-HT.csv	5 minutes ago	286 seconds	0 seconds	0.20322

Complete

KAGGLE SCORE

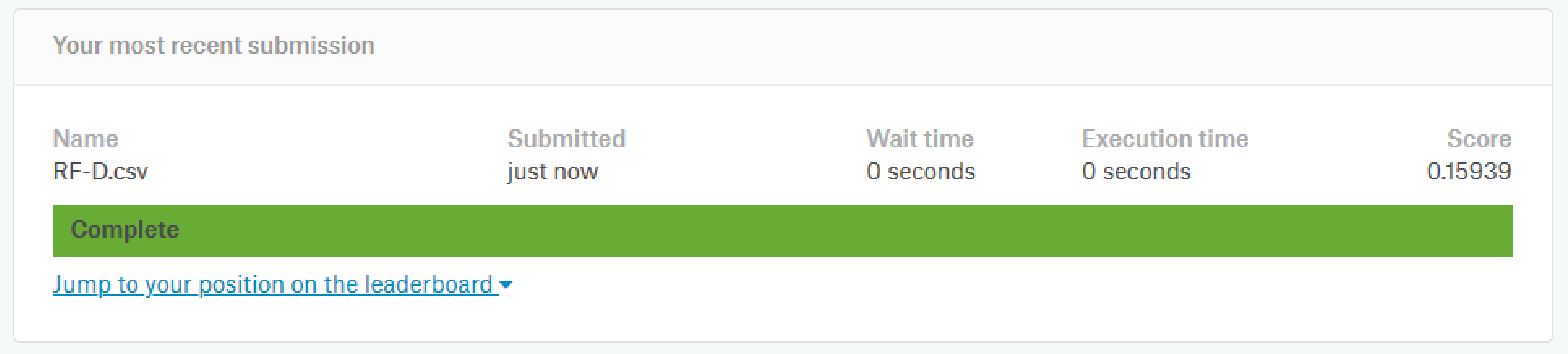


Random Forest Regressor

```
#Random Forest Regressor==============================================================================
rfr = RandomForestRegressor()
rfr.fit(X_Train, Y_Train)
model_score = rfr.score(X_Train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train:',model_score)
rfr_predict_Train=rfr.predict(X_Test_1)

#clf.feature_importances_
print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, rfr_predict_Train)))
print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, rfr_predict_Train))
```

```
Coefficient of determination R^2 of the
prediction on train: 0.9724403118852285

RMSE on test 0.15
```

```
#Save predictions for default random forest
pred_test =pd.DataFrame(rfr.predict(X_TestPP),columns=["Prediction"])
Id = testData['Id']
submission = pd.DataFrame({"Id": testData['Id'], "SalePrice":pred_test["Prediction"]})
submission['SalePrice'] = np.exp(submission['SalePrice'])
submission.to_csv("/gdrive/My Drive/508-Team-log/RF-D.csv", index = None)
```



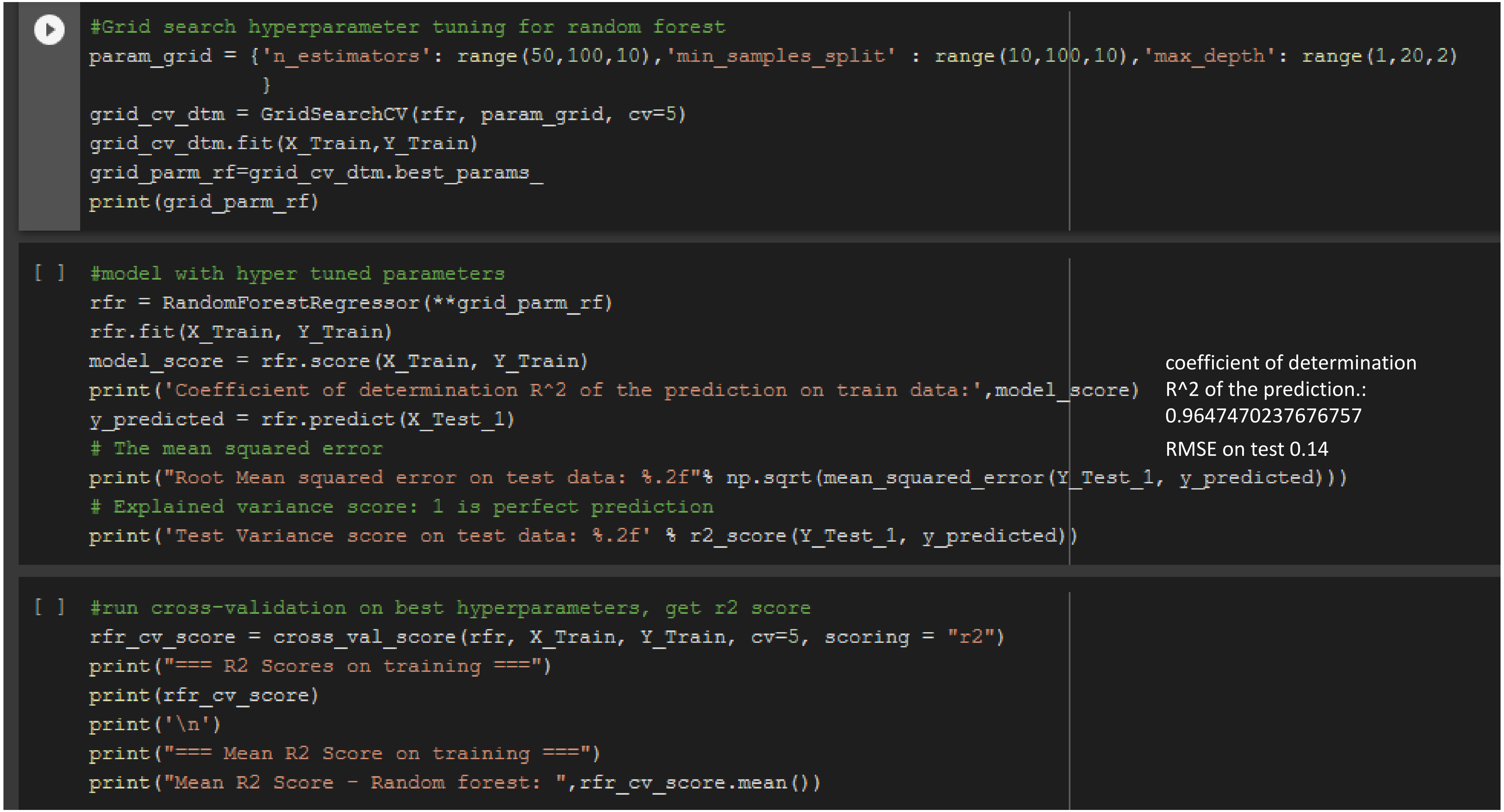
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
RF-D.csv	just now	0 seconds	0 seconds	0.15939

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE



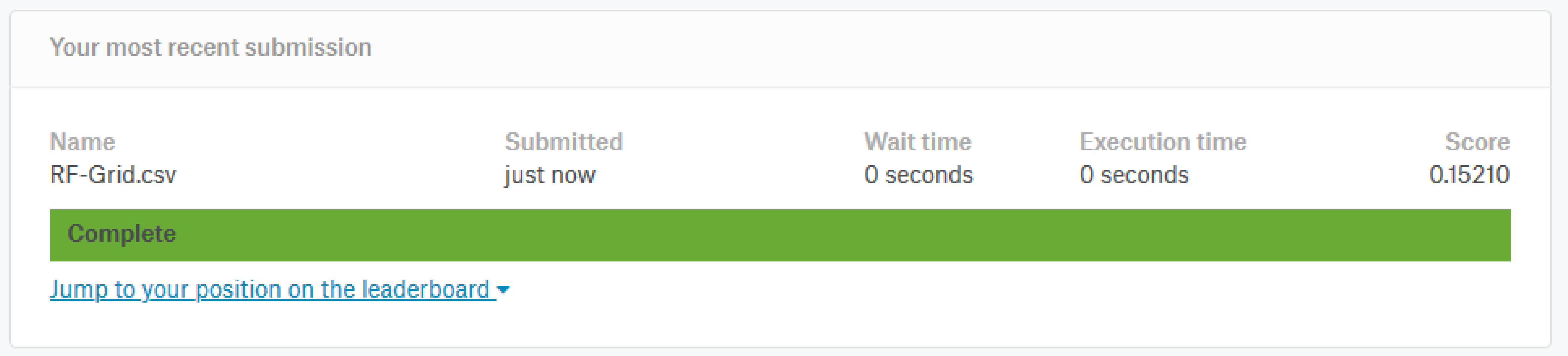
```
#Grid search hyperparameter tuning for random forest
param_grid = {'n_estimators': range(50,100,10),'min_samples_split' : range(10,100,10),'max_depth': range(1,20,2)
              }
grid_cv_dtm = GridSearchCV(rfr, param_grid, cv=5)
grid_cv_dtm.fit(X_Train,Y_Train)
grid_parm_rf=grid_cv_dtm.best_params_
print(grid_parm_rf)
```

```
#model with hyper tuned parameters
rfr = RandomForestRegressor(**grid_parm_rf)
rfr.fit(X_Train, Y_Train)
model_score = rfr.score(X_Train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train data:',model_score)
y_predicted = rfr.predict(X_Test_1)
# The mean squared error
print("Root Mean squared error on test data: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score on test data: %.2f' % r2_score(Y_Test_1, y_predicted))
```

coefficient of determination
R^2 of the prediction.:
0.9647470237676757

RMSE on test 0.14

```
#run cross-validation on best hyperparameters, get r2 score
rfr_cv_score = cross_val_score(rfr, X_Train, Y_Train, cv=5, scoring = "r2")
print("=== R2 Scores on training ===")
print(rfr_cv_score)
print('\n')
print("=== Mean R2 Score on training ===")
print("Mean R2 Score - Random forest: ",rfr_cv_score.mean())
```



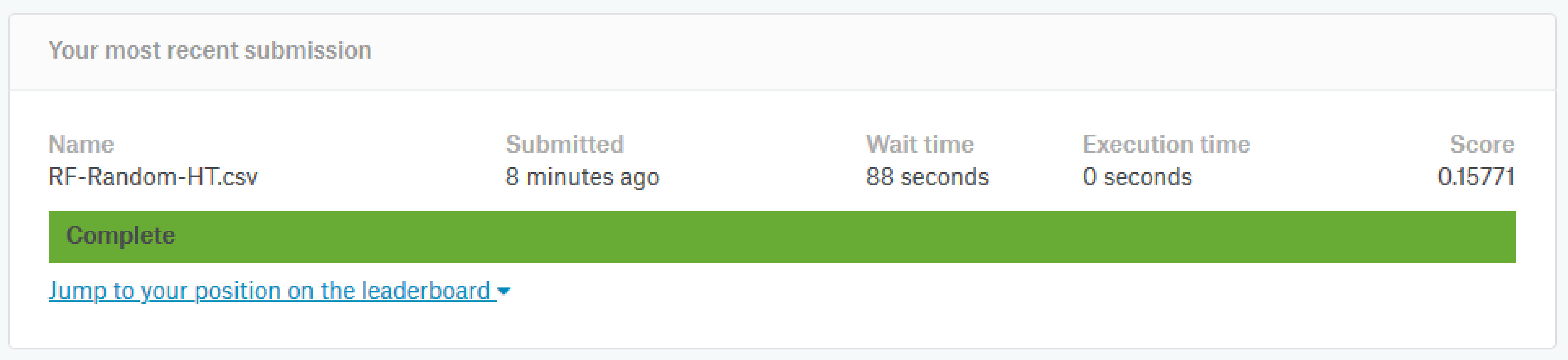
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
RF-Grid.csv	just now	0 seconds	0 seconds	0.15210

Complete

Jump to your position on the leaderboard

KAGGLE SCORE



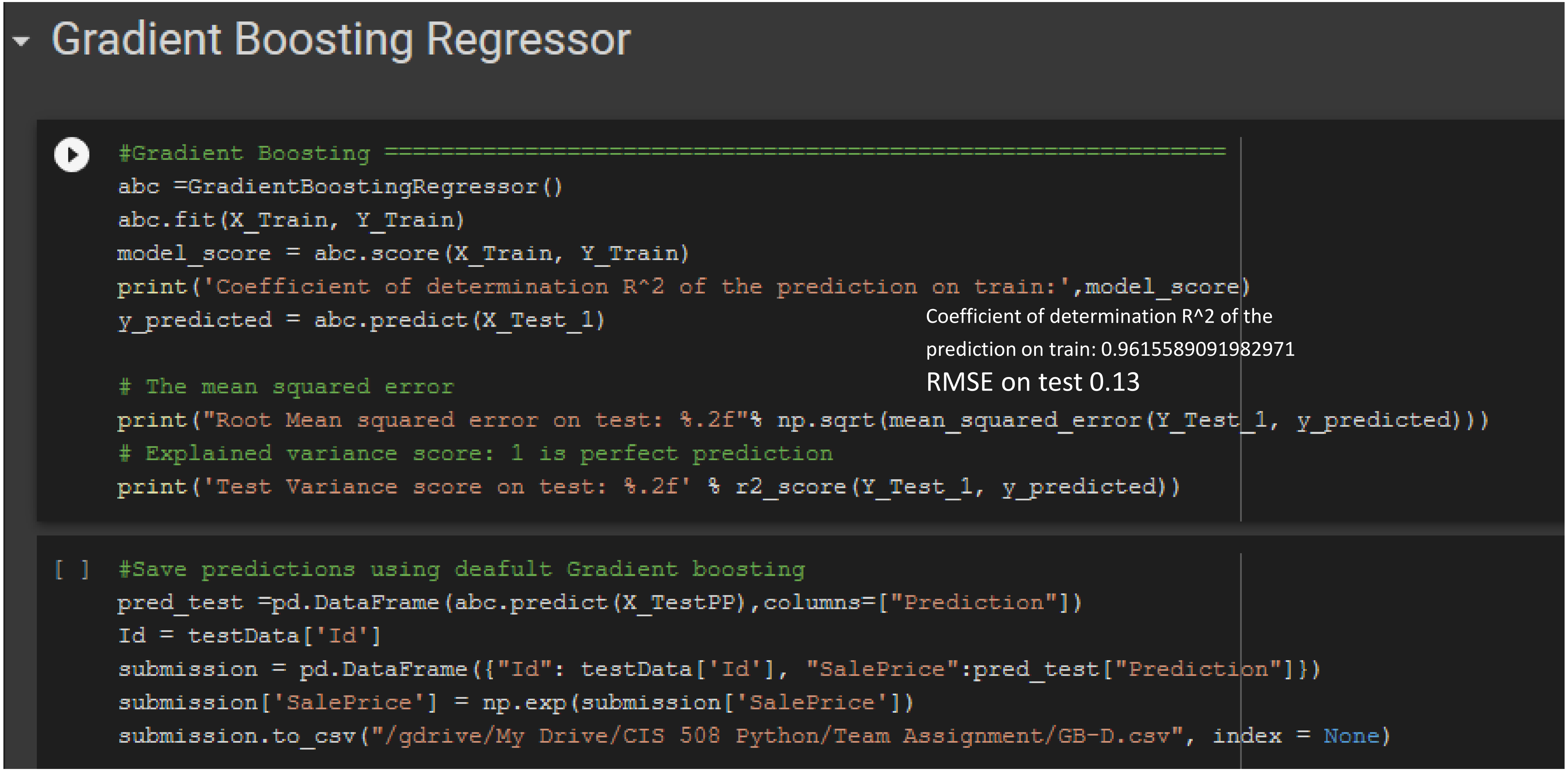
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
RF-Random-HT.csv	8 minutes ago	88 seconds	0 seconds	0.15771

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE

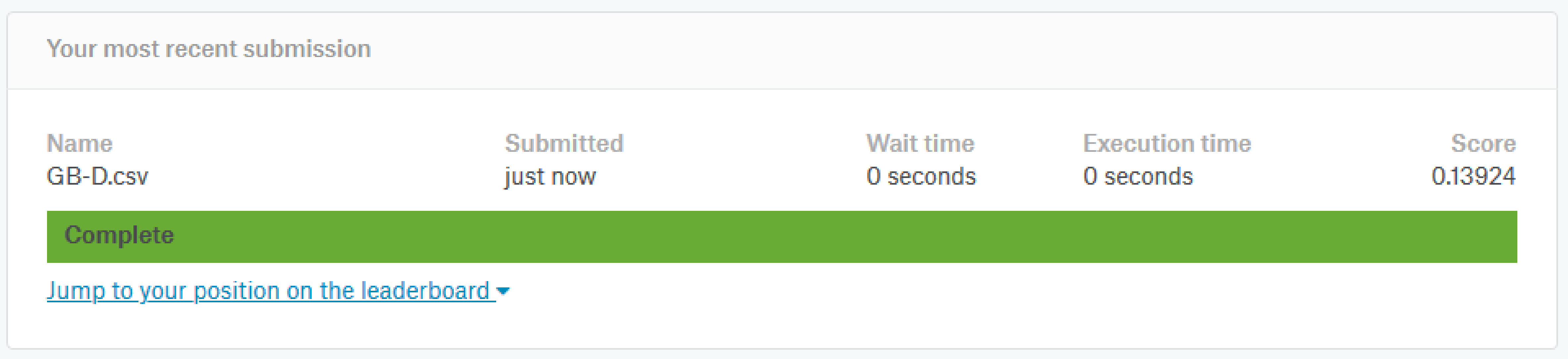


Gradient Boosting Regressor

```
#Gradient Boosting ============================================================
abc =GradientBoostingRegressor()
abc.fit(X_Train, Y_Train)
model_score = abc.score(X_Train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train:',model_score)
y_predicted = abc.predict(X_Test_1)

# The mean squared error
print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, y_predicted))
```

```
Coefficient of determination R^2 of the
prediction on train: 0.9615589091982971
RMSE on test 0.13
```

```
[ ]  #Save predictions using deafult Gradient boosting
pred_test =pd.DataFrame(abc.predict(X_TestPP),columns=["Prediction"])
Id = testData['Id']
submission = pd.DataFrame({"Id": testData['Id'], "SalePrice":pred_test["Prediction"]})
submission['SalePrice'] = np.exp(submission['SalePrice'])
submission.to_csv("/gdrive/My Drive/CIS 508 Python/Team Assignment/GB-D.csv", index = None)
```



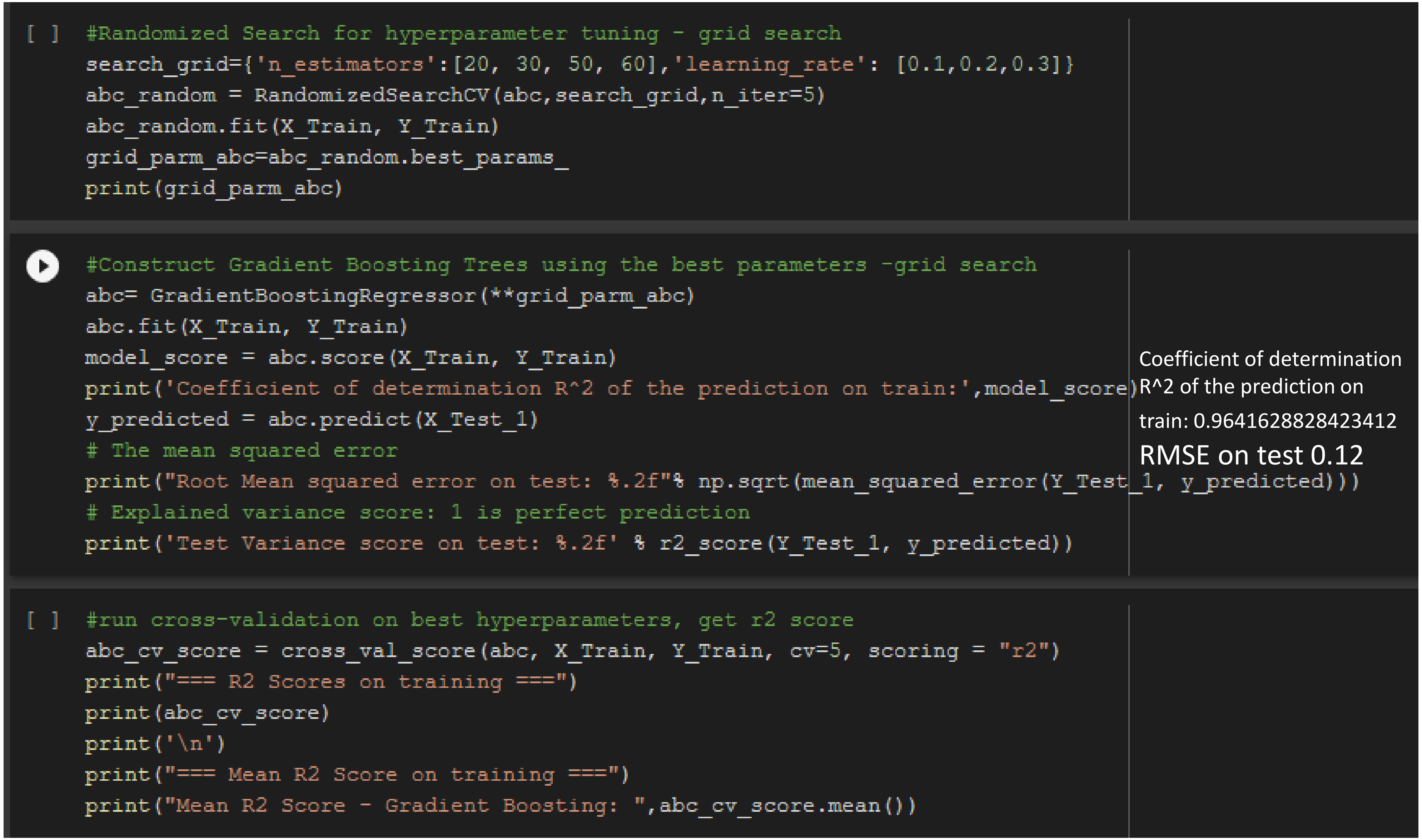
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
GB-D.csv	just now	0 seconds	0 seconds	0.13924

Complete

Jump to your position on the leaderboard

KAGGLE SCORE



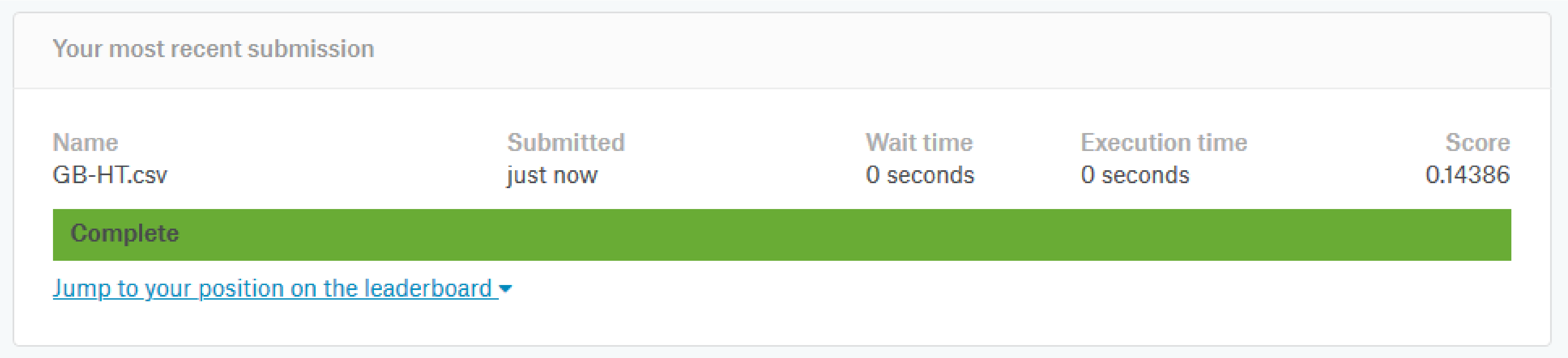
```
[ ]  #Randomized Search for hyperparameter tuning - grid search
     search_grid={'n_estimators':[20, 30, 50, 60],'learning_rate': [0.1,0.2,0.3]}
     abc_random = RandomizedSearchCV(abc,search_grid,n_iter=5)
     abc_random.fit(X_Train, Y_Train)
     grid_parm_abc=abc_random.best_params_
     print(grid_parm_abc)
```

```
     #Construct Gradient Boosting Trees using the best parameters -grid search
     abc= GradientBoostingRegressor(**grid_parm_abc)
     abc.fit(X_Train, Y_Train)
     model_score = abc.score(X_Train, Y_Train)
     print('Coefficient of determination R^2 of the prediction on train:',model_score)
     y_predicted = abc.predict(X_Test_1)
     # The mean squared error
     print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
     # Explained variance score: 1 is perfect prediction
     print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, y_predicted))
```

Coefficient of determination R^2 of the prediction on train: 0.9641628828423412

RMSE on test 0.12

```
[ ]  #run cross-validation on best hyperparameters, get r2 score
     abc_cv_score = cross_val_score(abc, X_Train, Y_Train, cv=5, scoring = "r2")
     print("=== R2 Scores on training ===")
     print(abc_cv_score)
     print('\n')
     print("=== Mean R2 Score on training ===")
     print("Mean R2 Score - Gradient Boosting: ",abc_cv_score.mean())
```



Your most recent submission

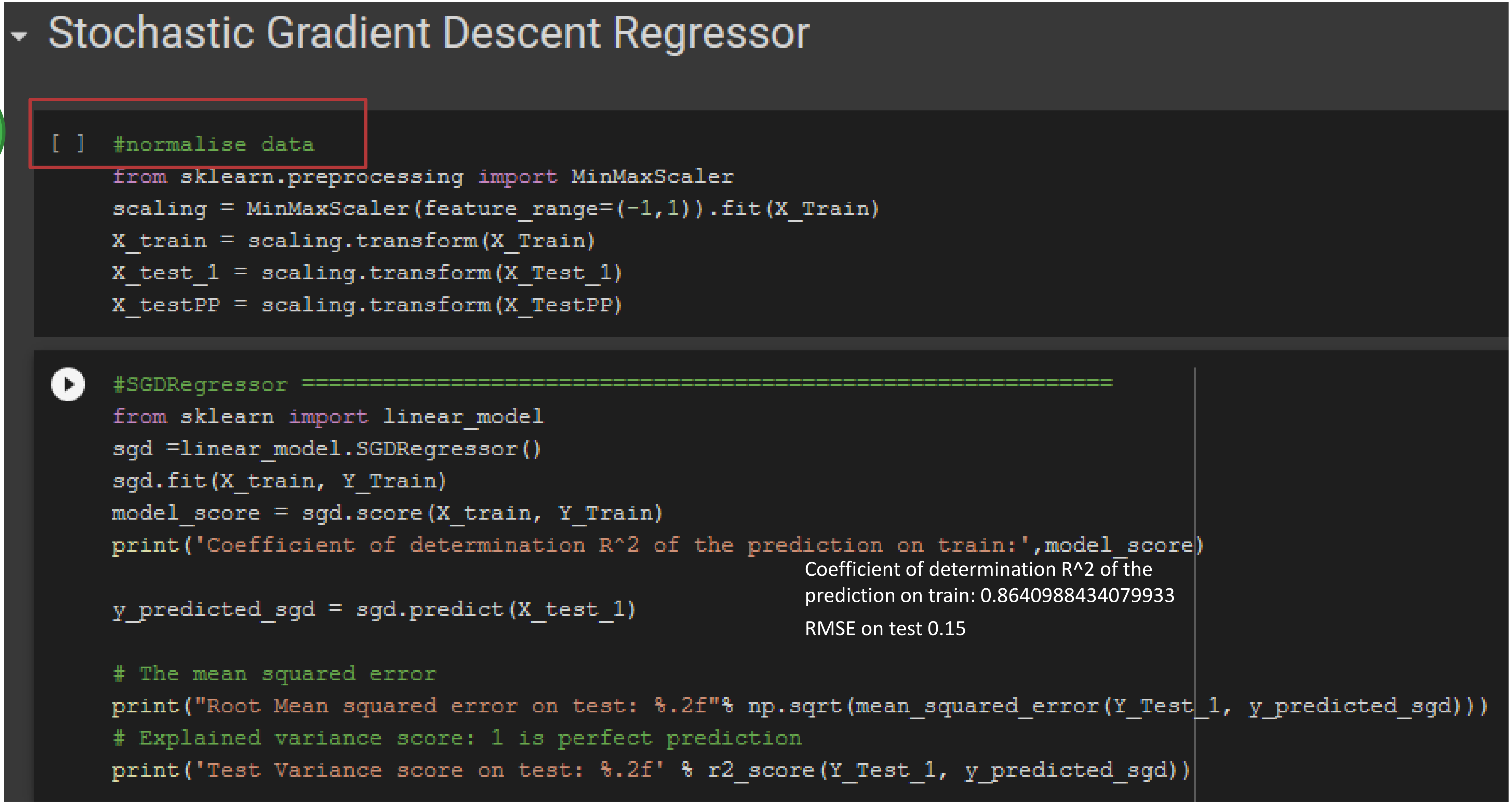
Name	Submitted	Wait time	Execution time	Score
GB-HT.csv	just now	0 seconds	0 seconds	0.14386

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE

1

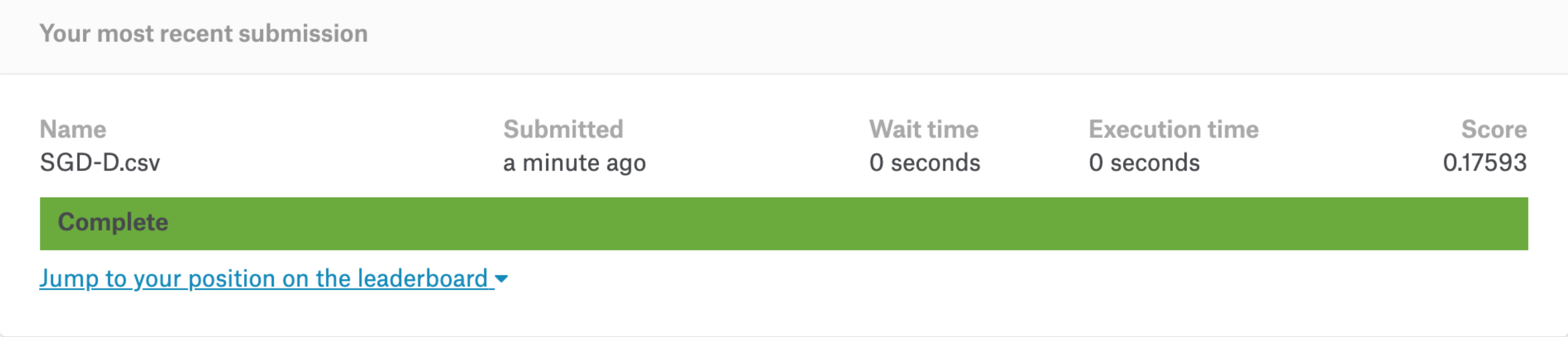


Stochastic Gradient Descent Regressor

```
[ ]  #normalise data
     from sklearn.preprocessing import MinMaxScaler
     scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_Train)
     X_train = scaling.transform(X_Train)
     X_test_1 = scaling.transform(X_Test_1)
     X_testPP = scaling.transform(X_TestPP)
```

```
#SGDRegressor ============================================================
from sklearn import linear_model
sgd =linear_model.SGDRegressor()
sgd.fit(X_train, Y_Train)
model_score = sgd.score(X_train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train:',model_score)

y_predicted_sgd = sgd.predict(X_test_1)

# The mean squared error
print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted_sgd)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, y_predicted_sgd))
```

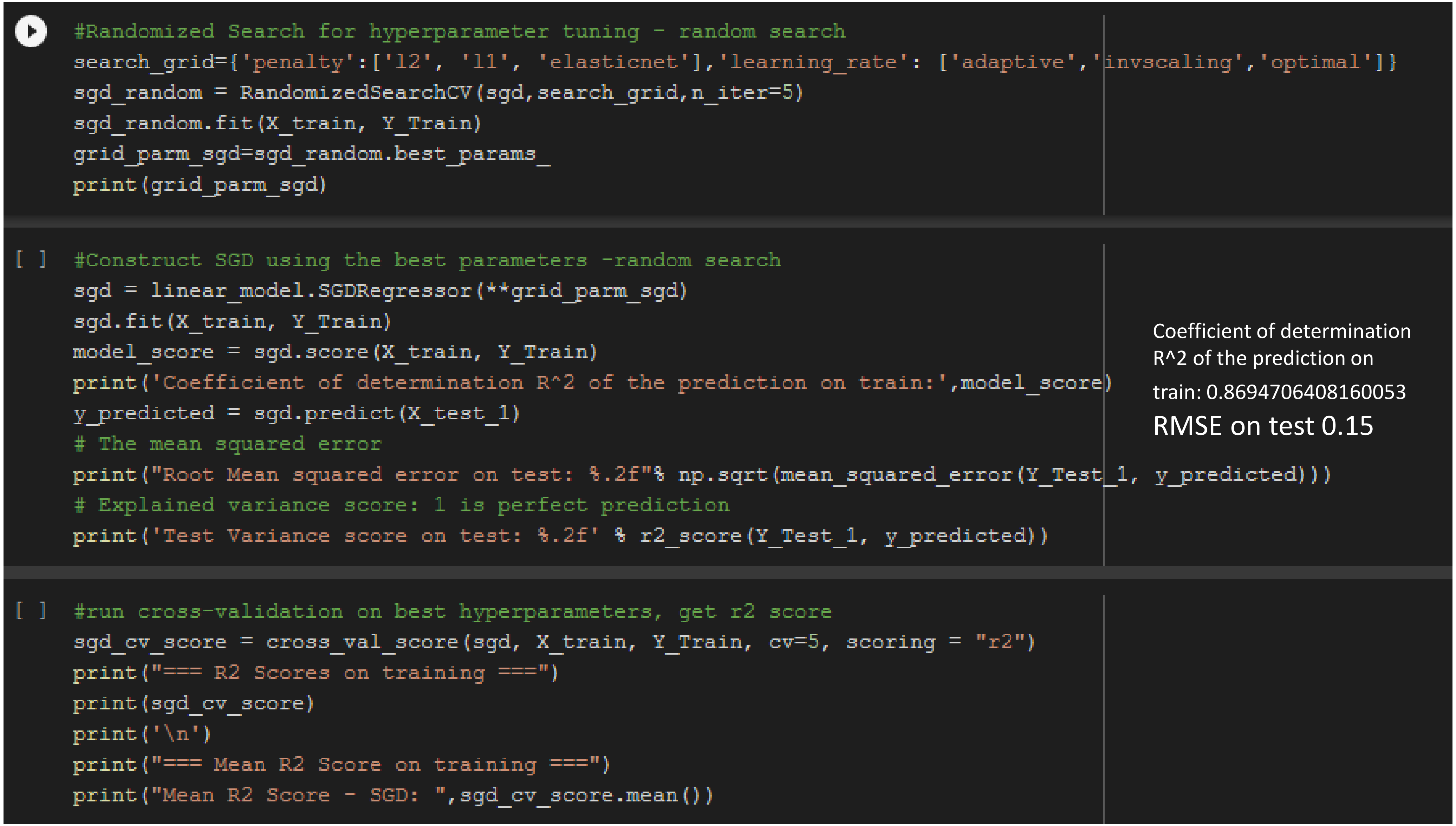
```
Coefficient of determination R^2 of the
prediction on train: 0.8640988434079933

RMSE on test 0.15
```



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
SGD-D.csv	a minute ago	0 seconds	0 seconds	0.17593

KAGGLE SCORE



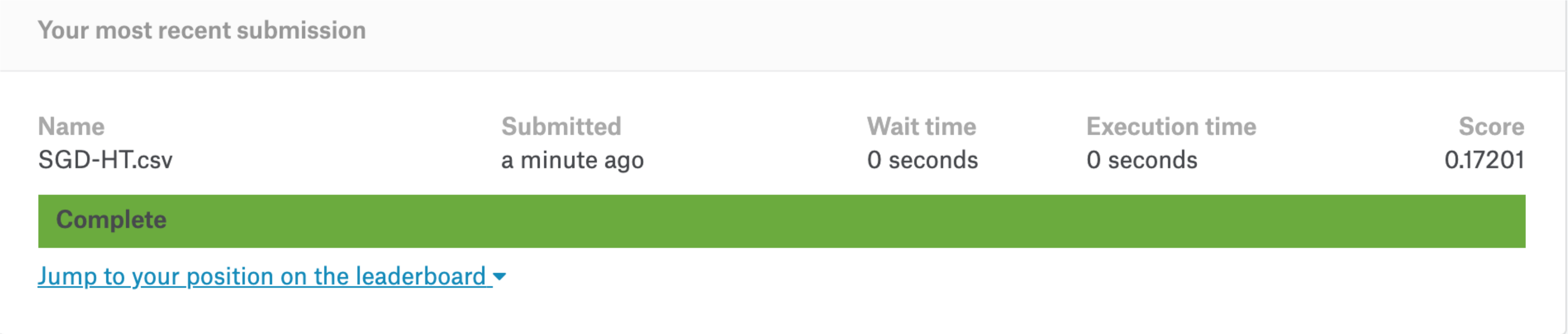
```
#Randomized Search for hyperparameter tuning - random search
search_grid={'penalty':['l2', 'l1', 'elasticnet'],'learning_rate': ['adaptive','invscaling','optimal']}
sgd_random = RandomizedSearchCV(sgd,search_grid,n_iter=5)
sgd_random.fit(X_train, Y_Train)
grid_parm_sgd=sgd_random.best_params_
print(grid_parm_sgd)
```

```
#Construct SGD using the best parameters -random search
sgd = linear_model.SGDRegressor(**grid_parm_sgd)
sgd.fit(X_train, Y_Train)
model_score = sgd.score(X_train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train:',model_score)
y_predicted = sgd.predict(X_test_1)
# The mean squared error
print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, y_predicted))
```

Coefficient of determination R^2 of the prediction on train: 0.8694706408160053

RMSE on test 0.15

```
#run cross-validation on best hyperparameters, get r2 score
sgd_cv_score = cross_val_score(sgd, X_train, Y_Train, cv=5, scoring = "r2")
print("=== R2 Scores on training ===")
print(sgd_cv_score)
print('\n')
print("=== Mean R2 Score on training ===")
print("Mean R2 Score - SGD: ",sgd_cv_score.mean())
```



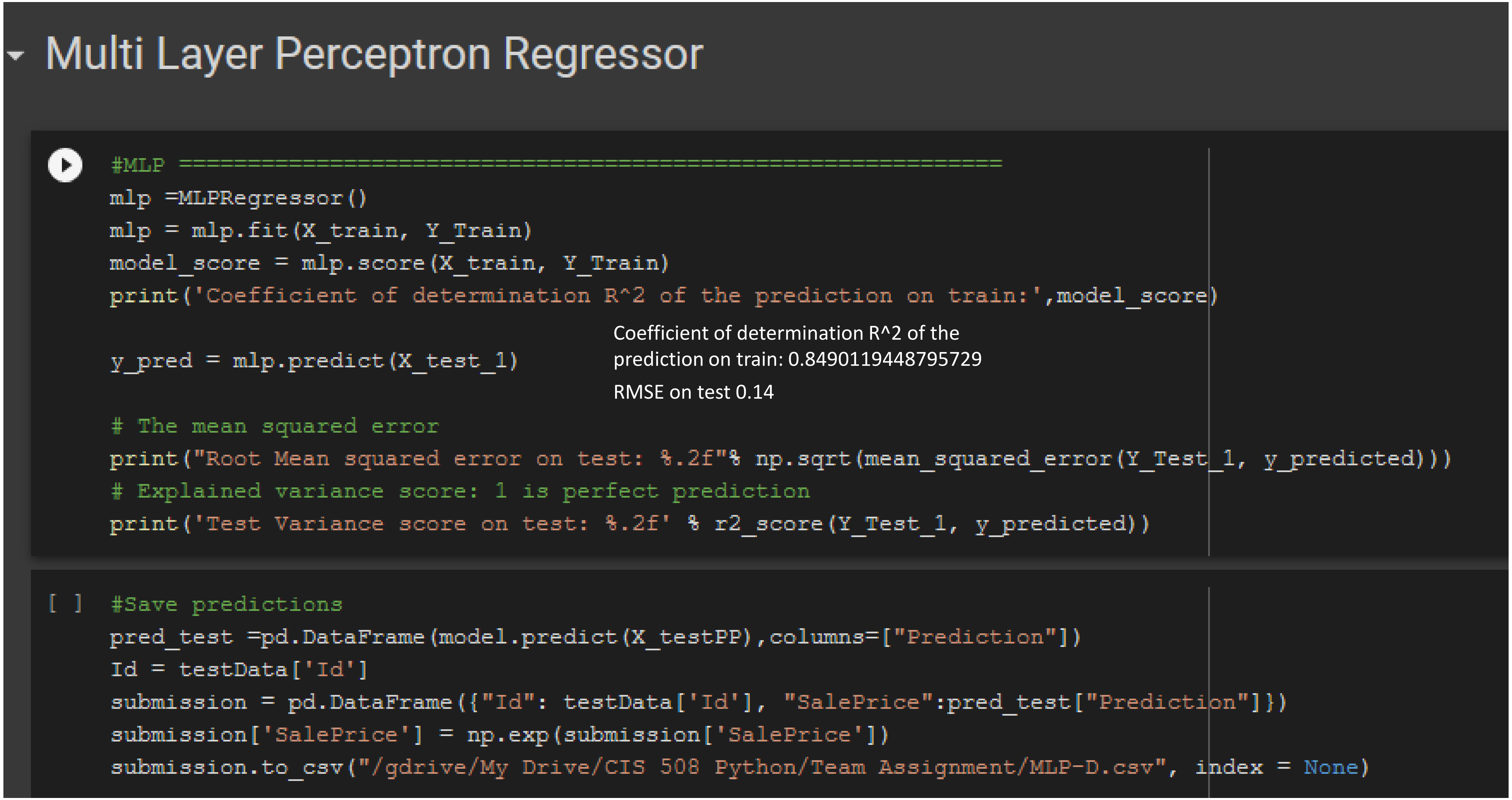
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
SGD-HT.csv	a minute ago	0 seconds	0 seconds	0.17201

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE

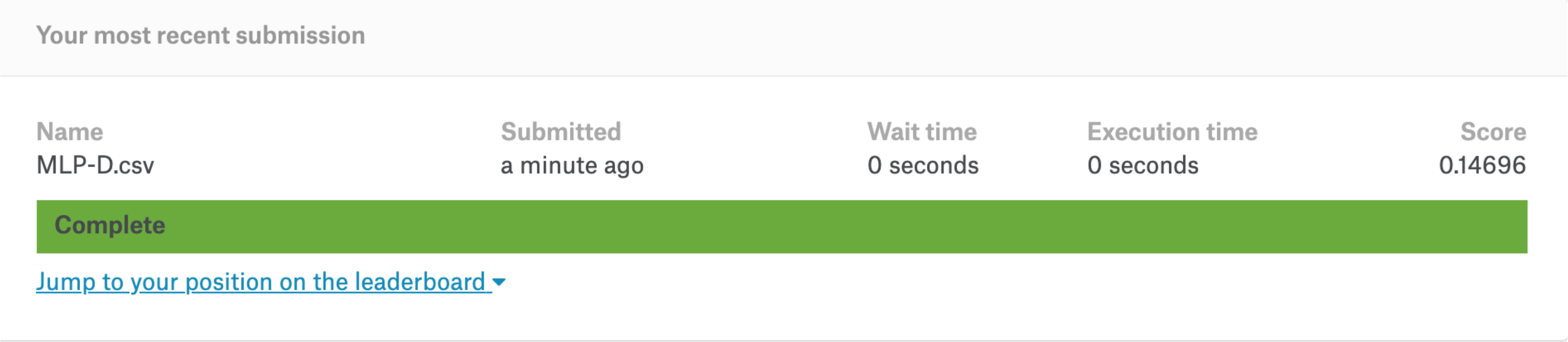


Multi Layer Perceptron Regressor

```
#MLP ===========================================================
mlp =MLPRegressor()
mlp = mlp.fit(X_train, Y_Train)
model_score = mlp.score(X_train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train:',model_score)

y_pred = mlp.predict(X_test_1)

# The mean squared error
print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, y_predicted))
```

```
Coefficient of determination R^2 of the
prediction on train: 0.8490119448795729

RMSE on test 0.14
```

```
#Save predictions
pred_test =pd.DataFrame(model.predict(X_testPP),columns=["Prediction"])
Id = testData['Id']
submission = pd.DataFrame({"Id": testData['Id'], "SalePrice":pred_test["Prediction"]})
submission['SalePrice'] = np.exp(submission['SalePrice'])
submission.to_csv("/gdrive/My Drive/CIS 508 Python/Team Assignment/MLP-D.csv", index = None)
```



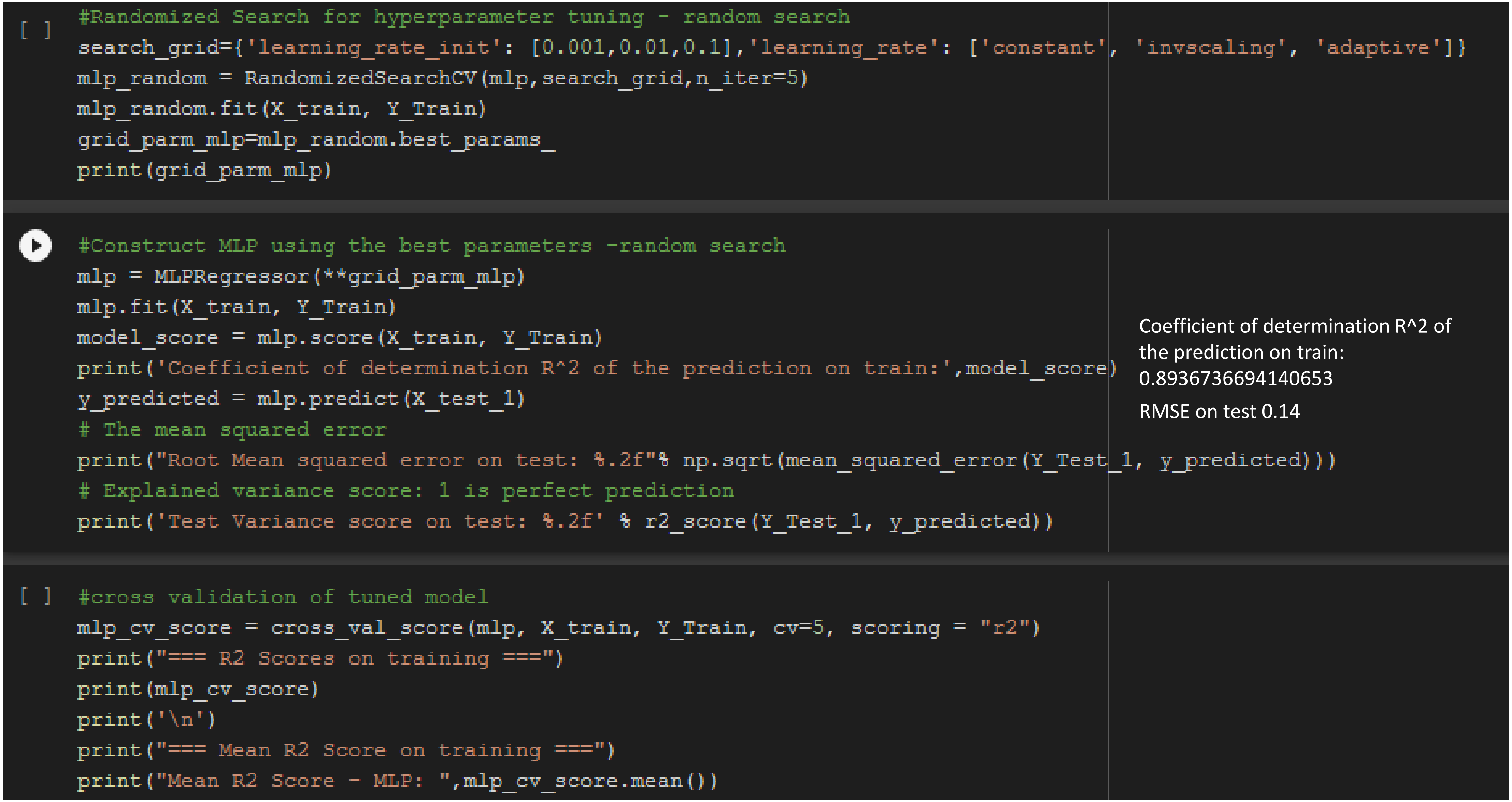
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
MLP-D.csv	a minute ago	0 seconds	0 seconds	0.14696

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE



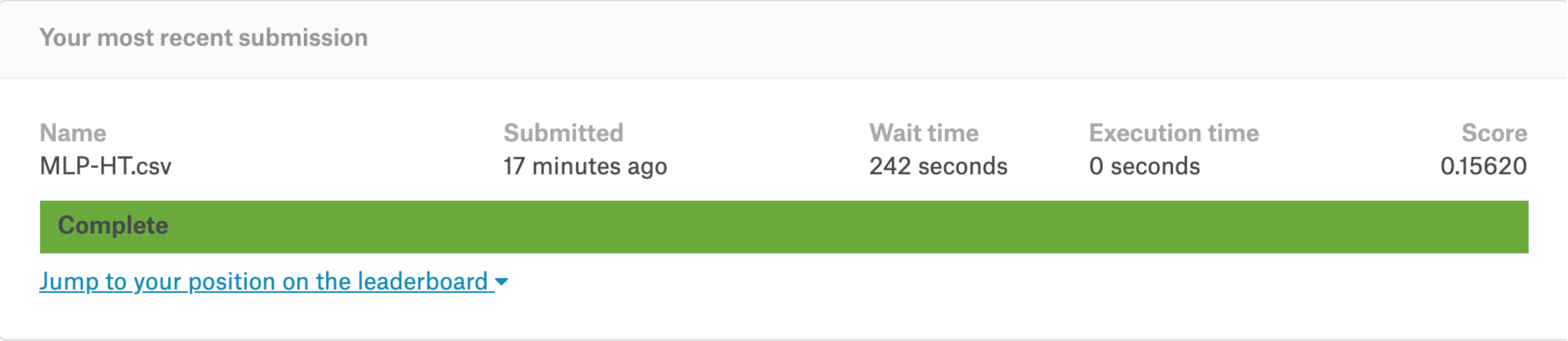
```
#Randomized Search for hyperparameter tuning - random search
search_grid={'learning_rate_init': [0.001,0.01,0.1],'learning_rate': ['constant', 'invscaling', 'adaptive']}
mlp_random = RandomizedSearchCV(mlp,search_grid,n_iter=5)
mlp_random.fit(X_train, Y_Train)
grid_parm_mlp=mlp_random.best_params_
print(grid_parm_mlp)
```

```
#Construct MLP using the best parameters -random search
mlp = MLPRegressor(**grid_parm_mlp)
mlp.fit(X_train, Y_Train)
model_score = mlp.score(X_train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train:',model_score)
y_predicted = mlp.predict(X_test_1)
# The mean squared error
print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, y_predicted))
```

Coefficient of determination R^2 of the prediction on train: 0.8936736694140653

RMSE on test 0.14

```
#cross validation of tuned model
mlp_cv_score = cross_val_score(mlp, X_train, Y_Train, cv=5, scoring = "r2")
print("=== R2 Scores on training ===")
print(mlp_cv_score)
print('\n')
print("=== Mean R2 Score on training ===")
print("Mean R2 Score - MLP: ",mlp_cv_score.mean())
```



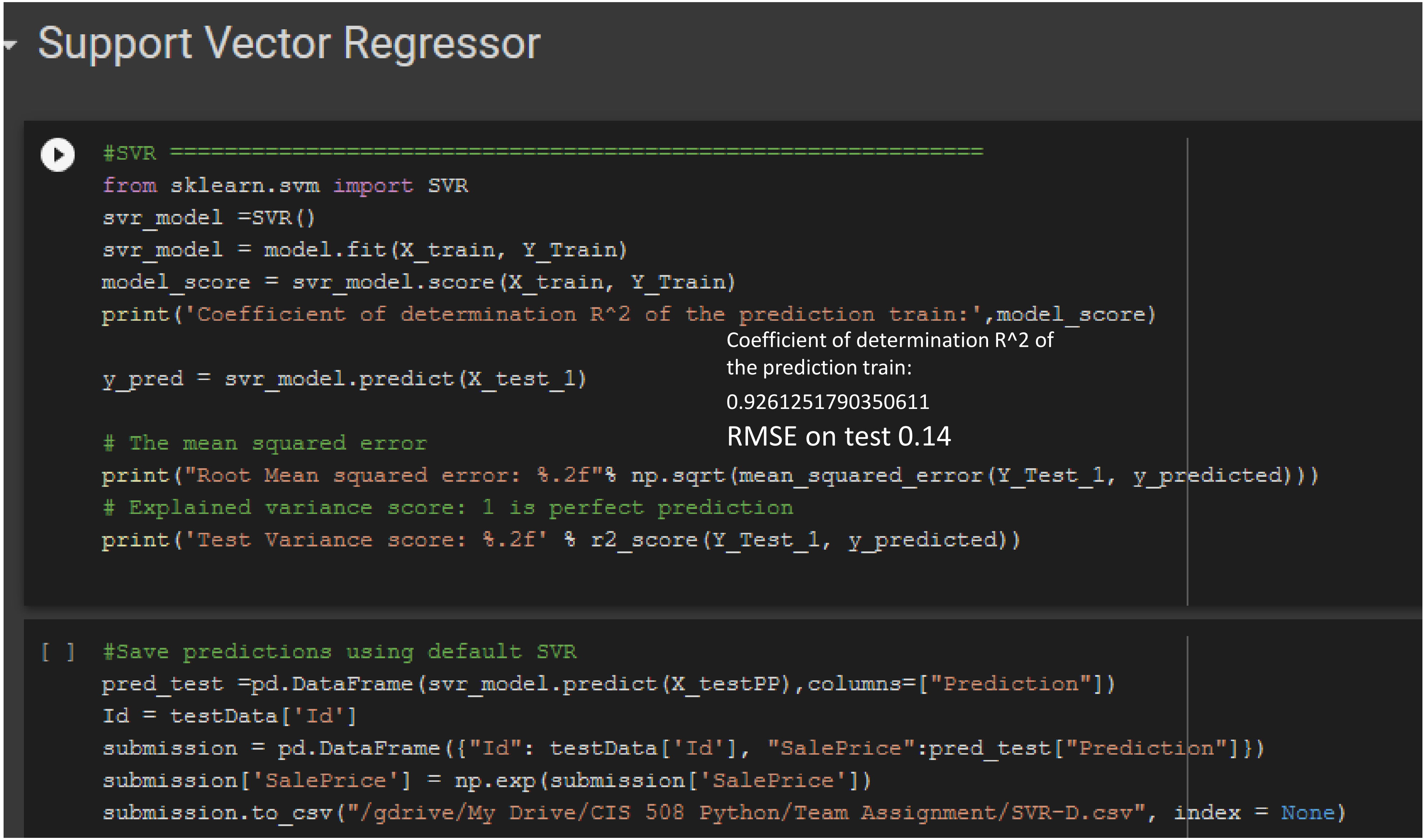
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
MLP-HT.csv	17 minutes ago	242 seconds	0 seconds	0.15620

Complete

Jump to your position on the leaderboard

KAGGLE SCORE



Support Vector Regressor

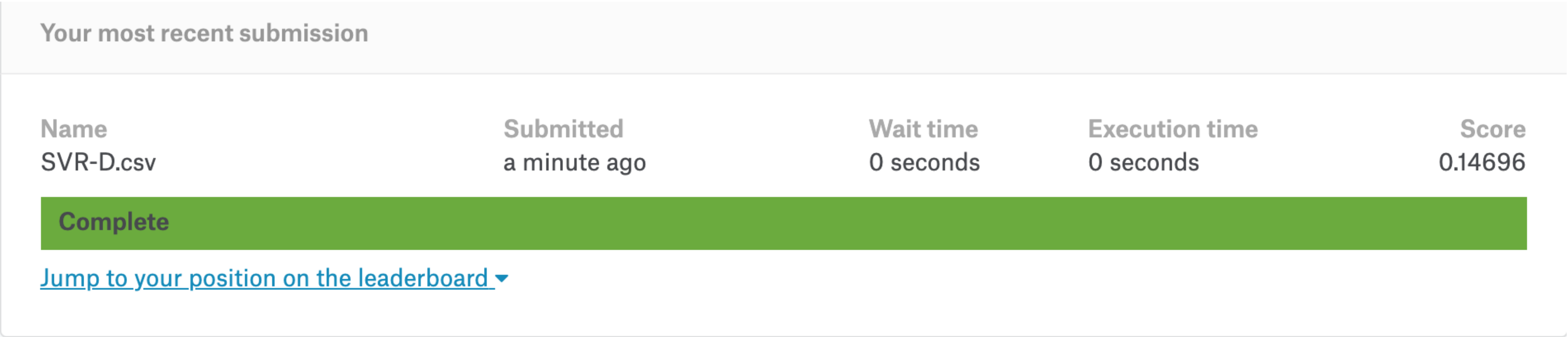
```
#SVR ============================================================
from sklearn.svm import SVR
svr_model =SVR()
svr_model = model.fit(X_train, Y_Train)
model_score = svr_model.score(X_train, Y_Train)
print('Coefficient of determination R^2 of the prediction train:',model_score)

y_pred = svr_model.predict(X_test_1)

# The mean squared error
print("Root Mean squared error: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(Y_Test_1, y_predicted))
```

Coefficient of determination R^2 of the prediction train:

0.9261251790350611

RMSE on test 0.14

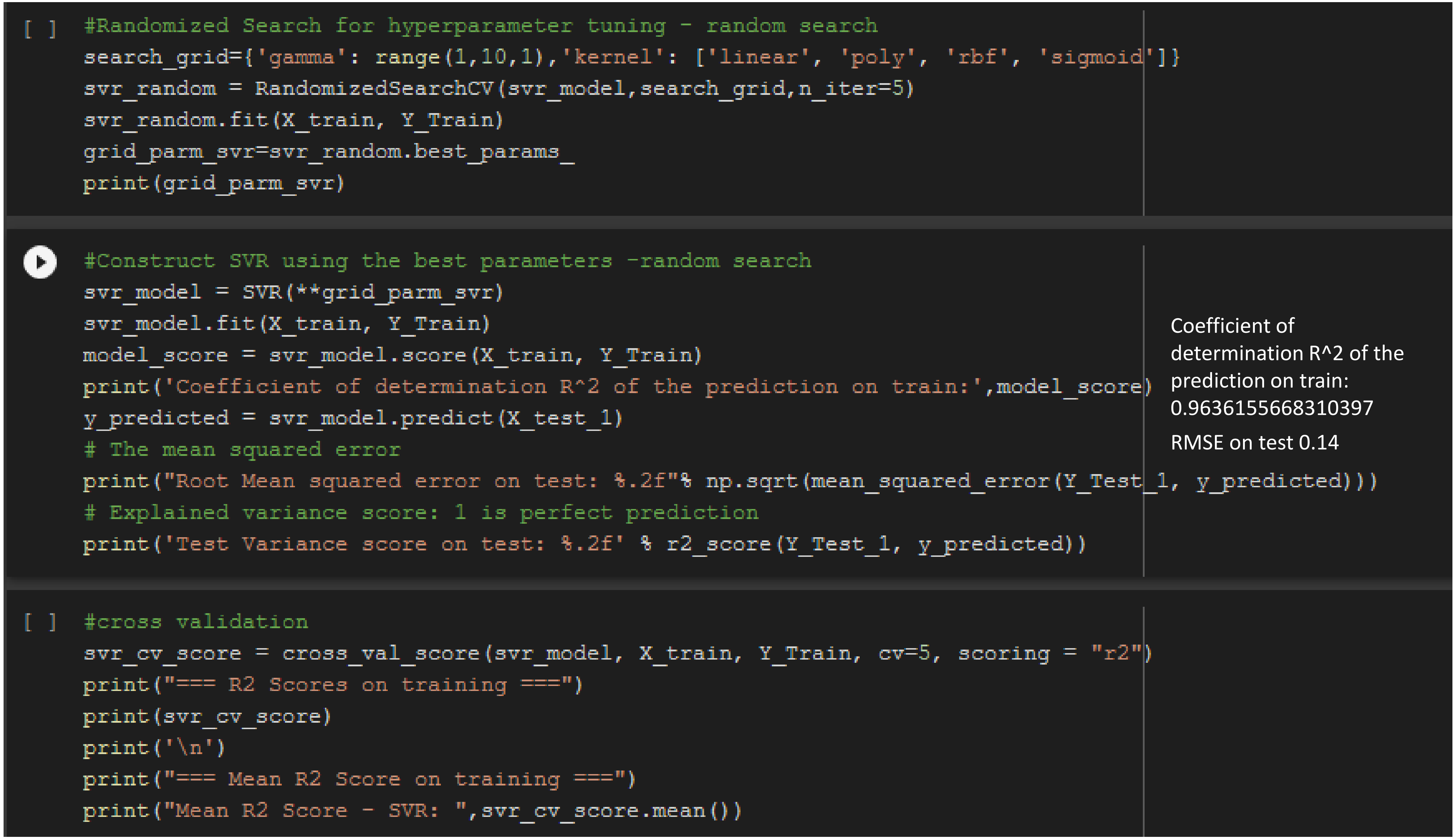
```
[ ]  #Save predictions using default SVR
pred_test =pd.DataFrame(svr_model.predict(X_testPP),columns=["Prediction"])
Id = testData['Id']
submission = pd.DataFrame({"Id": testData['Id'], "SalePrice":pred_test["Prediction"]})
submission['SalePrice'] = np.exp(submission['SalePrice'])
submission.to_csv("/gdrive/My Drive/CIS 508 Python/Team Assignment/SVR-D.csv", index = None)
```



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
SVR-D.csv	a minute ago	0 seconds	0 seconds	0.14696

KAGGLE SCORE



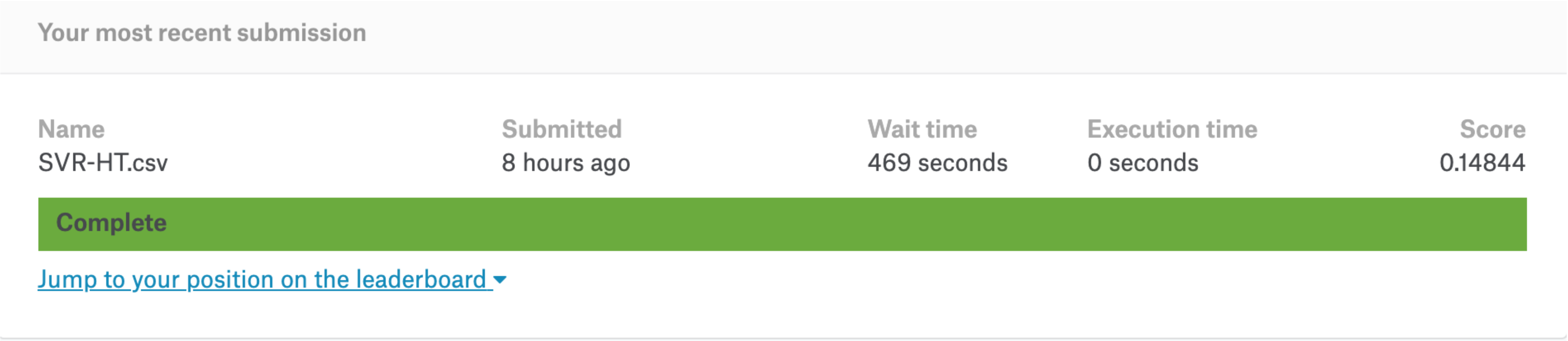
```
#Randomized Search for hyperparameter tuning - random search
search_grid={'gamma': range(1,10,1),'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}
svr_random = RandomizedSearchCV(svr_model,search_grid,n_iter=5)
svr_random.fit(X_train, Y_Train)
grid_parm_svr=svr_random.best_params_
print(grid_parm_svr)
```

```
#Construct SVR using the best parameters -random search
svr_model = SVR(**grid_parm_svr)
svr_model.fit(X_train, Y_Train)
model_score = svr_model.score(X_train, Y_Train)
print('Coefficient of determination R^2 of the prediction on train:',model_score)
y_predicted = svr_model.predict(X_test_1)
# The mean squared error
print("Root Mean squared error on test: %.2f"% np.sqrt(mean_squared_error(Y_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score on test: %.2f' % r2_score(Y_Test_1, y_predicted))
```

Coefficient of determination R^2 of the prediction on train: 0.9636155668310397

RMSE on test 0.14

```
#cross validation
svr_cv_score = cross_val_score(svr_model, X_train, Y_Train, cv=5, scoring = "r2")
print("=== R2 Scores on training ===")
print(svr_cv_score)
print('\n')
print("=== Mean R2 Score on training ===")
print("Mean R2 Score - SVR: ",svr_cv_score.mean())
```



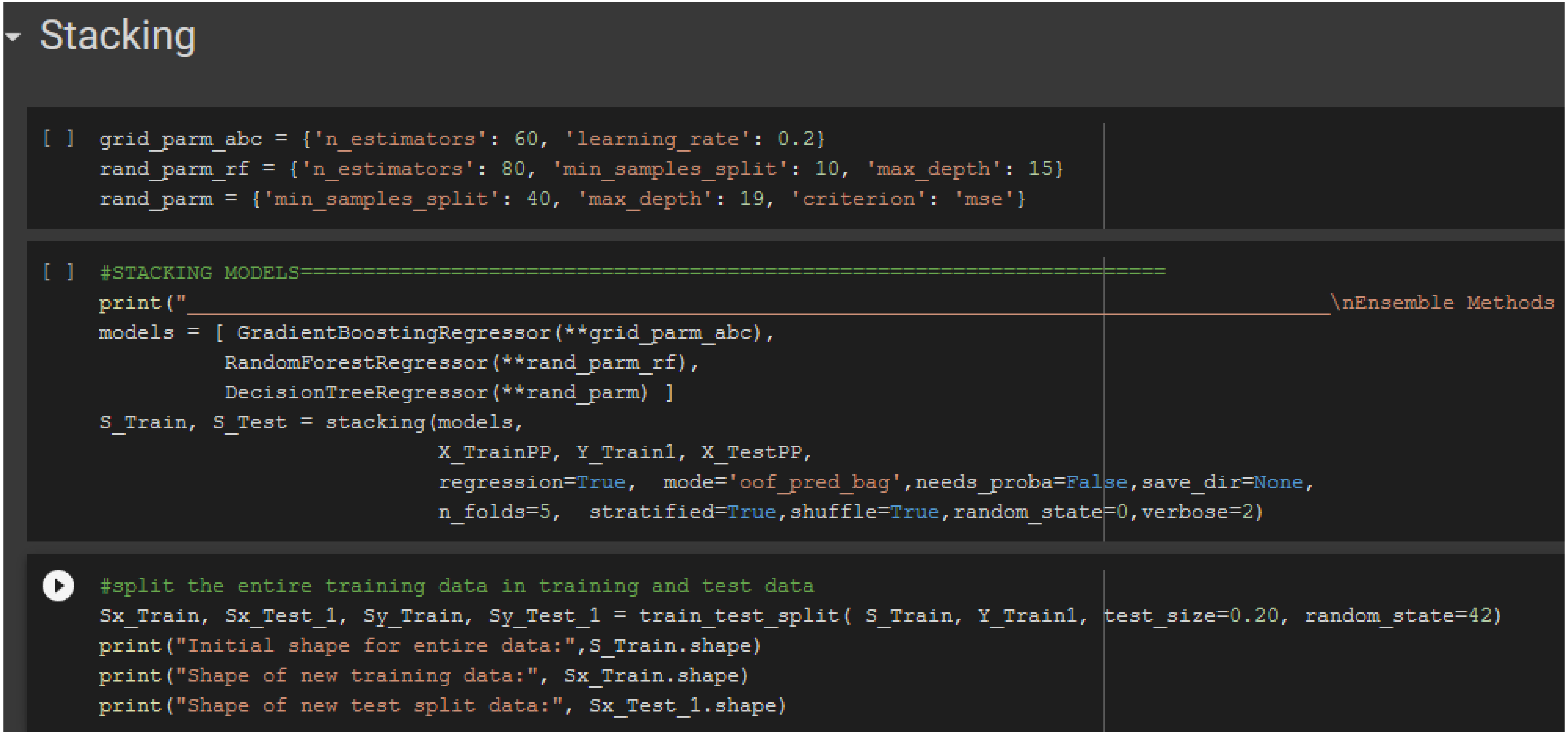
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
SVR-HT.csv	8 hours ago	469 seconds	0 seconds	0.14844

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE

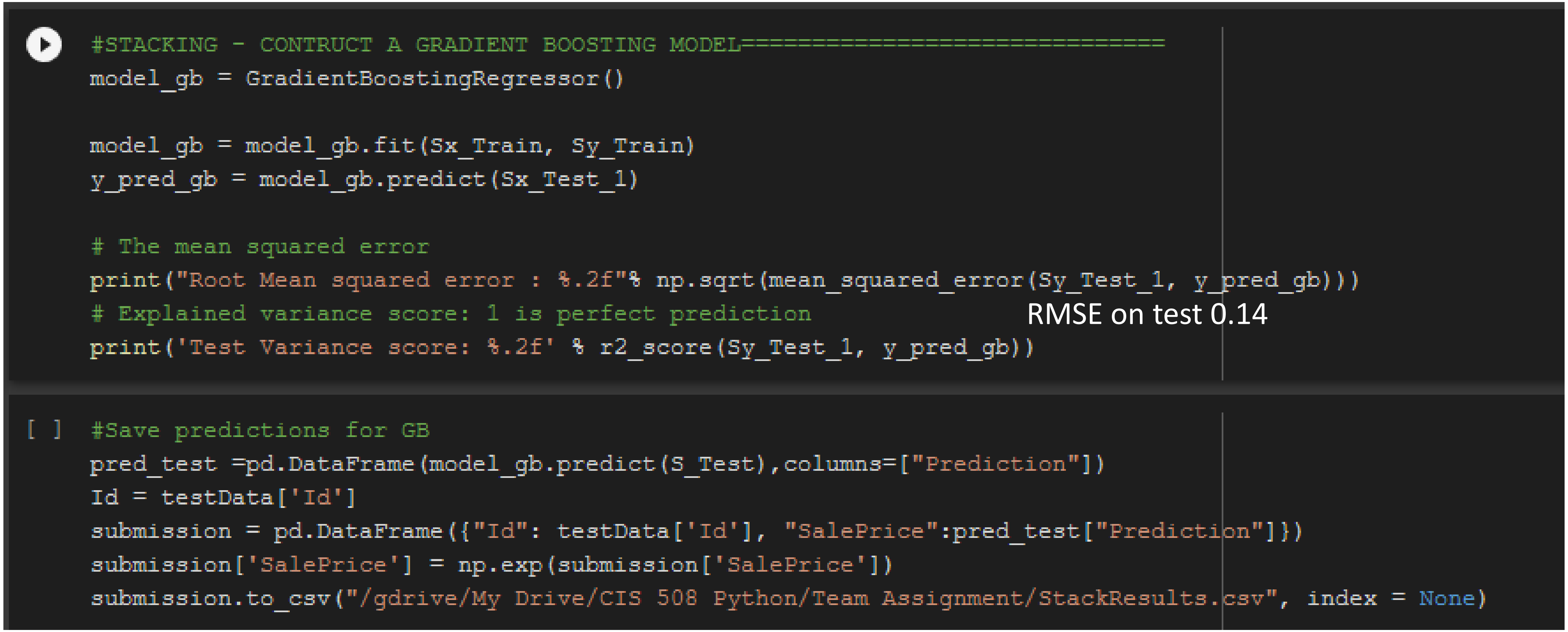


Stacking

```
[ ]  grid_parm_abc = {'n_estimators': 60, 'learning_rate': 0.2}
     rand_parm_rf = {'n_estimators': 80, 'min_samples_split': 10, 'max_depth': 15}
     rand_parm = {'min_samples_split': 40, 'max_depth': 19, 'criterion': 'mse'}
```

```
[ ]  #STACKING MODELS=====================================================================
     print("___________________________________________________________________________________________\nEnsemble Methods
     models = [ GradientBoostingRegressor(**grid_parm_abc),
                RandomForestRegressor(**rand_parm_rf),
                DecisionTreeRegressor(**rand_parm) ]
     S_Train, S_Test = stacking(models,
                                X_TrainPP, Y_Train1, X_TestPP,
                                regression=True,  mode='oof_pred_bag',needs_proba=False,save_dir=None,
                                n_folds=5,  stratified=True,shuffle=True,random_state=0,verbose=2)
```

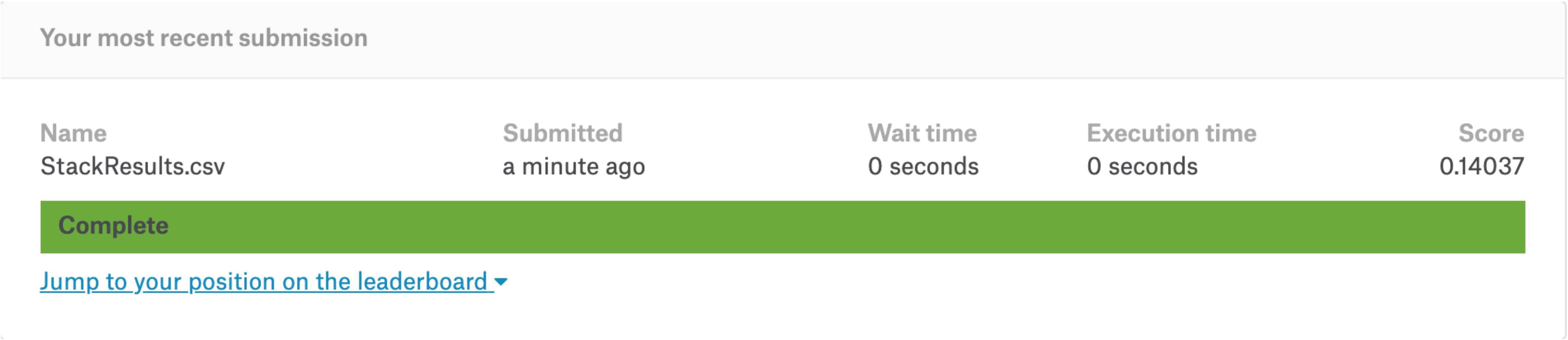
```
     #split the entire training data in training and test data
     Sx_Train, Sx_Test_1, Sy_Train, Sy_Test_1 = train_test_split( S_Train, Y_Train1, test_size=0.20, random_state=42)
     print("Initial shape for entire data:",S_Train.shape)
     print("Shape of new training data:", Sx_Train.shape)
     print("Shape of new test split data:", Sx_Test_1.shape)
```



```
#STACKING - CONTRUCT A GRADIENT BOOSTING MODEL==============================
model_gb = GradientBoostingRegressor()

model_gb = model_gb.fit(Sx_Train, Sy_Train)
y_pred_gb = model_gb.predict(Sx_Test_1)

# The mean squared error
print("Root Mean squared error : %.2f"% np.sqrt(mean_squared_error(Sy_Test_1, y_pred_gb)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(Sy_Test_1, y_pred_gb))
```

RMSE on test 0.14

```
#Save predictions for GB
pred_test =pd.DataFrame(model_gb.predict(S_Test),columns=["Prediction"])
Id = testData['Id']
submission = pd.DataFrame({"Id": testData['Id'], "SalePrice":pred_test["Prediction"]})
submission['SalePrice'] = np.exp(submission['SalePrice'])
submission.to_csv("/gdrive/My Drive/CIS 508 Python/Team Assignment/StackResults.csv", index = None)
```



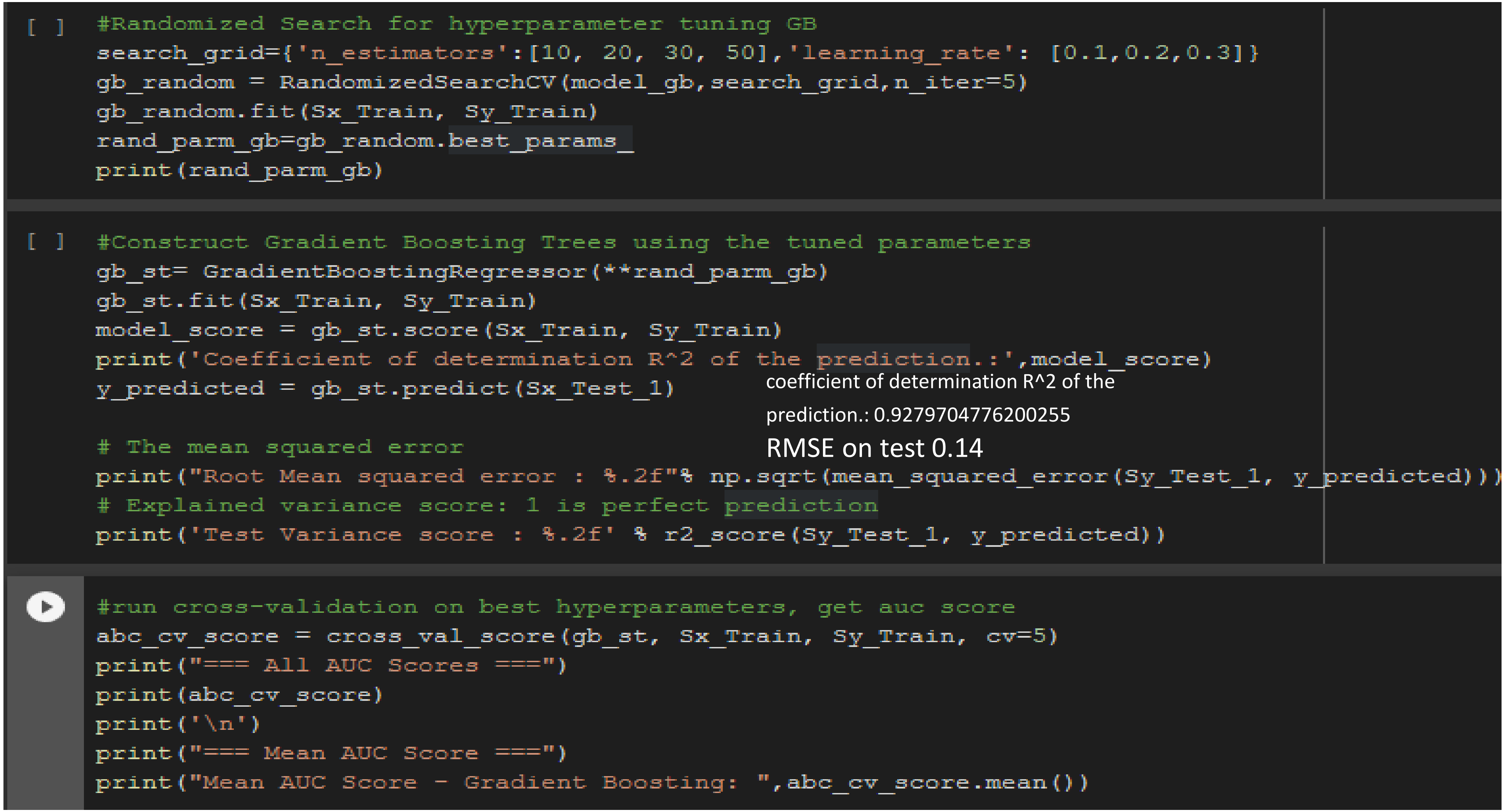
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
StackResults.csv	a minute ago	0 seconds	0 seconds	0.14037

Complete

Jump to your position on the leaderboard ▾

KAGGLE SCORE WITH GRADIENT BOOSTING

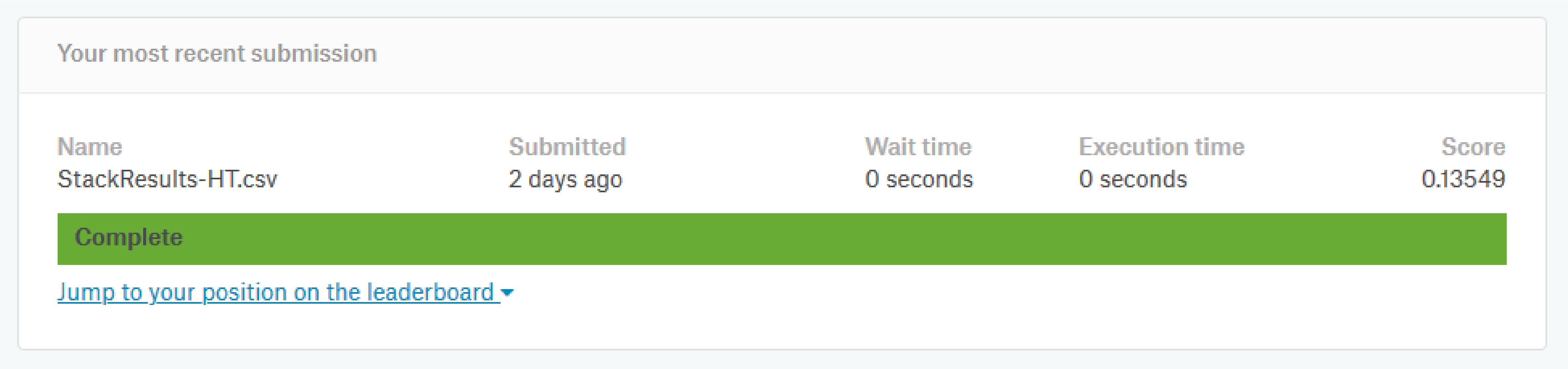


```
#Randomized Search for hyperparameter tuning GB
search_grid={'n_estimators':[10, 20, 30, 50],'learning_rate': [0.1,0.2,0.3]}
gb_random = RandomizedSearchCV(model_gb,search_grid,n_iter=5)
gb_random.fit(Sx_Train, Sy_Train)
rand_parm_gb=gb_random.best_params_
print(rand_parm_gb)
```

```
#Construct Gradient Boosting Trees using the tuned parameters
gb_st= GradientBoostingRegressor(**rand_parm_gb)
gb_st.fit(Sx_Train, Sy_Train)
model_score = gb_st.score(Sx_Train, Sy_Train)
print('Coefficient of determination R^2 of the prediction.:',model_score)
y_predicted = gb_st.predict(Sx_Test_1)

# The mean squared error
print("Root Mean squared error : %.2f"% np.sqrt(mean_squared_error(Sy_Test_1, y_predicted)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score : %.2f' % r2_score(Sy_Test_1, y_predicted))
```

```
coefficient of determination R^2 of the
prediction.: 0.9279704776200255
RMSE on test 0.14
```

```
#run cross-validation on best hyperparameters, get auc score
abc_cv_score = cross_val_score(gb_st, Sx_Train, Sy_Train, cv=5)
print("=== All AUC Scores ===")
print(abc_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Gradient Boosting: ",abc_cv_score.mean())
```



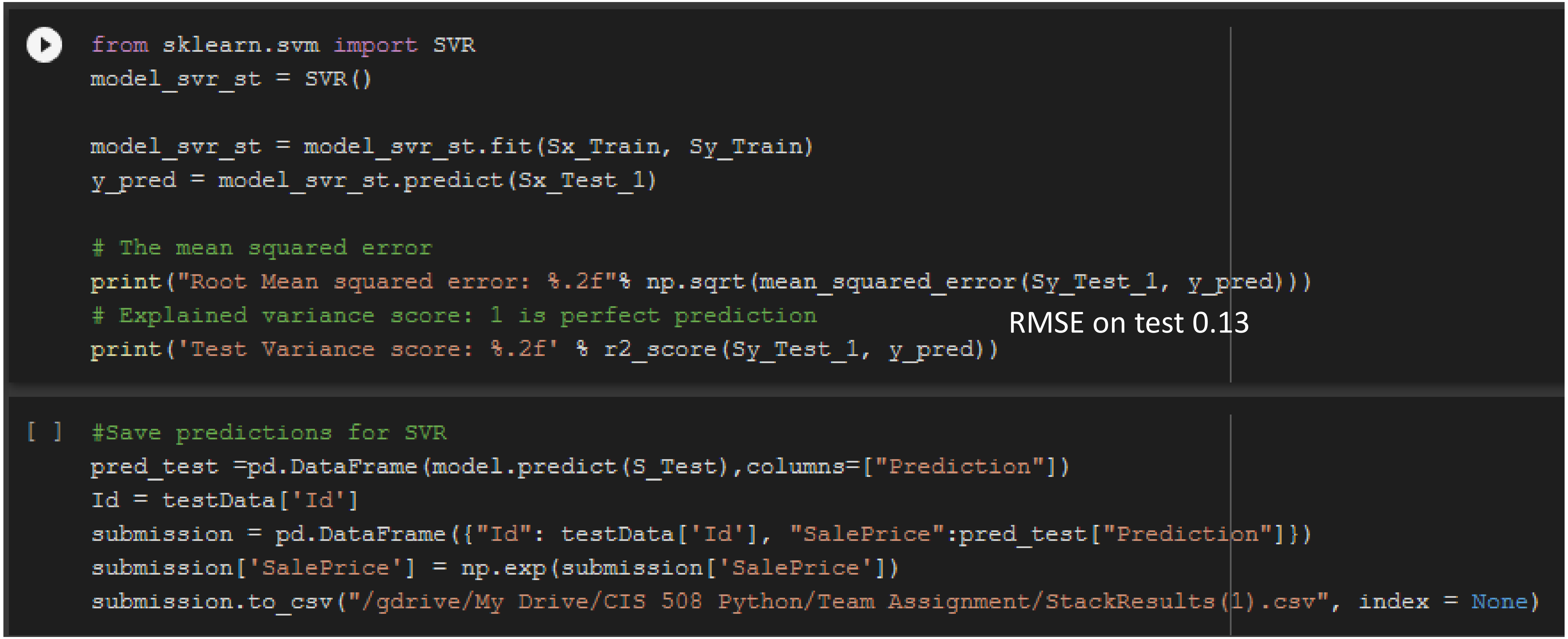
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
StackResults-HT.csv	2 days ago	0 seconds	0 seconds	0.13549

Complete

Jump to your position on the leaderboard

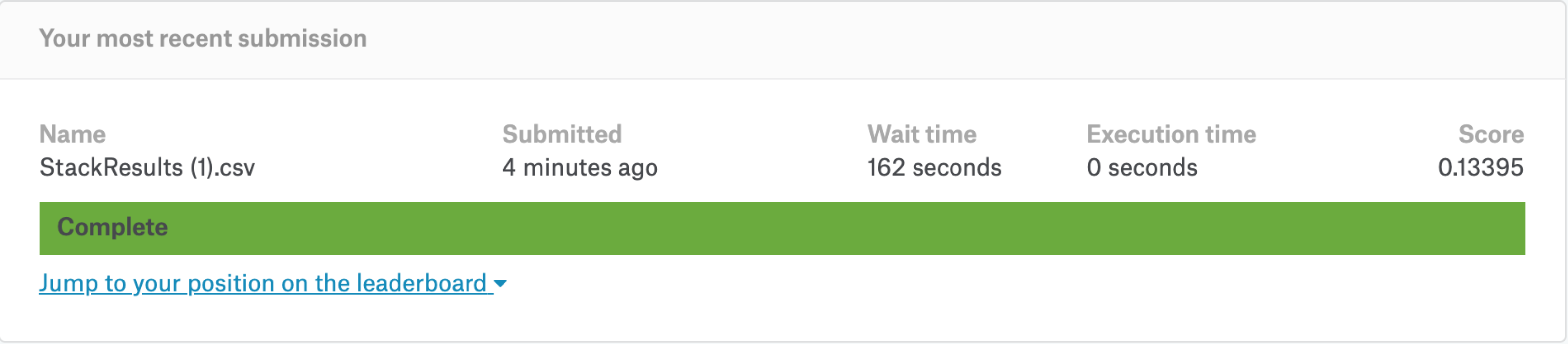
KAGGLE SCORE WITH GRADIENT BOOSTING HT



```
from sklearn.svm import SVR
model_svr_st = SVR()

model_svr_st = model_svr_st.fit(Sx_Train, Sy_Train)
y_pred = model_svr_st.predict(Sx_Test_1)

# The mean squared error
print("Root Mean squared error: %.2f"% np.sqrt(mean_squared_error(Sy_Test_1, y_pred)))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(Sy_Test_1, y_pred))
```

RMSE on test 0.13

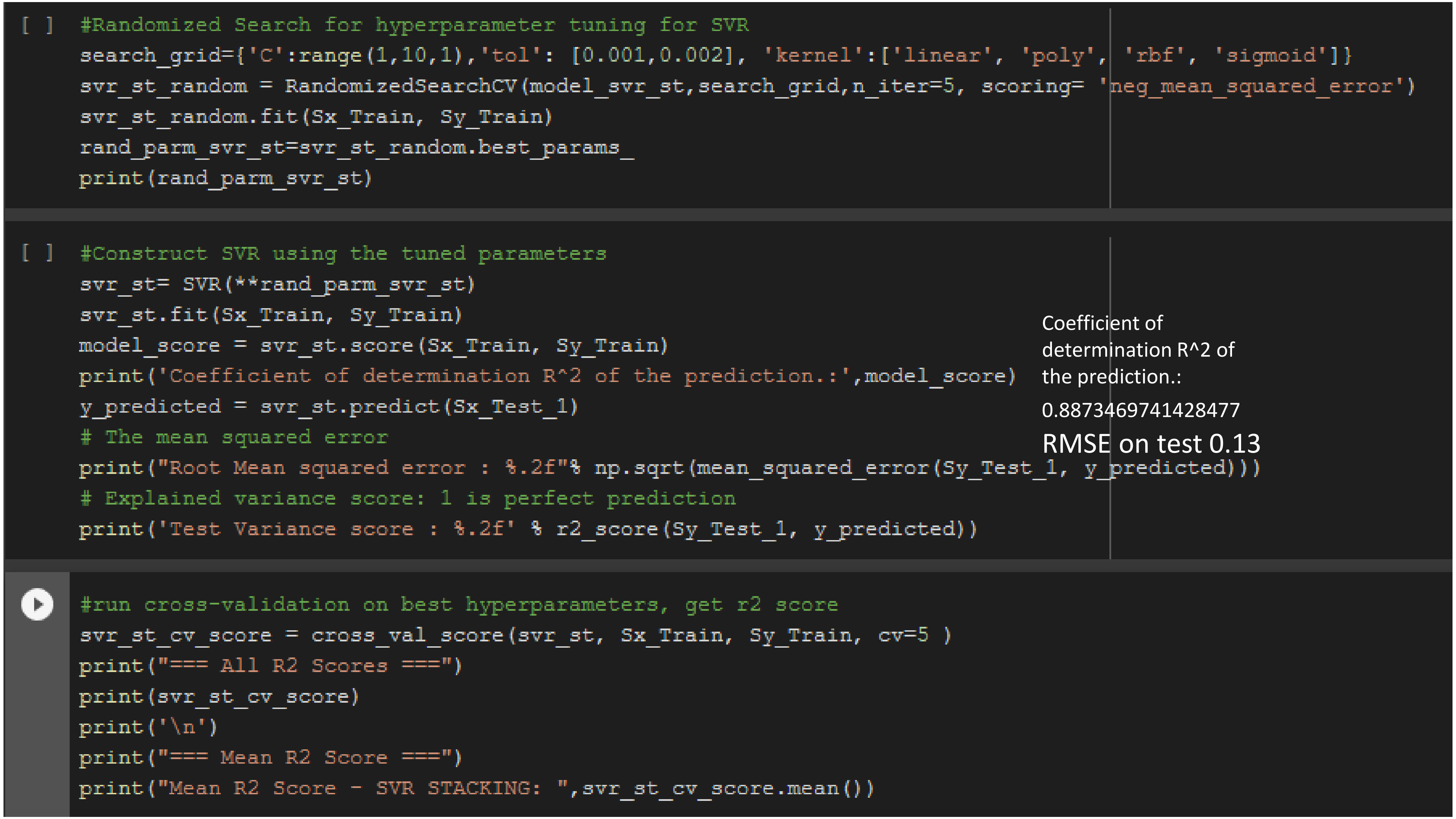
```
#Save predictions for SVR
pred_test =pd.DataFrame(model.predict(S_Test),columns=["Prediction"])
Id = testData['Id']
submission = pd.DataFrame({"Id": testData['Id'], "SalePrice":pred_test["Prediction"]})
submission['SalePrice'] = np.exp(submission['SalePrice'])
submission.to_csv("/gdrive/My Drive/CIS 508 Python/Team Assignment/StackResults(1).csv", index = None)
```



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
StackResults (1).csv	4 minutes ago	162 seconds	0 seconds	0.13395

KAGGLE SCORE WITH SVR META MODEL



```
[ ] #Randomized Search for hyperparameter tuning for SVR
    search_grid={'C':range(1,10,1),'tol': [0.001,0.002], 'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
    svr_st_random = RandomizedSearchCV(model_svr_st,search_grid,n_iter=5, scoring= 'neg_mean_squared_error')
    svr_st_random.fit(Sx_Train, Sy_Train)
    rand_parm_svr_st=svr_st_random.best_params_
    print(rand_parm_svr_st)
```

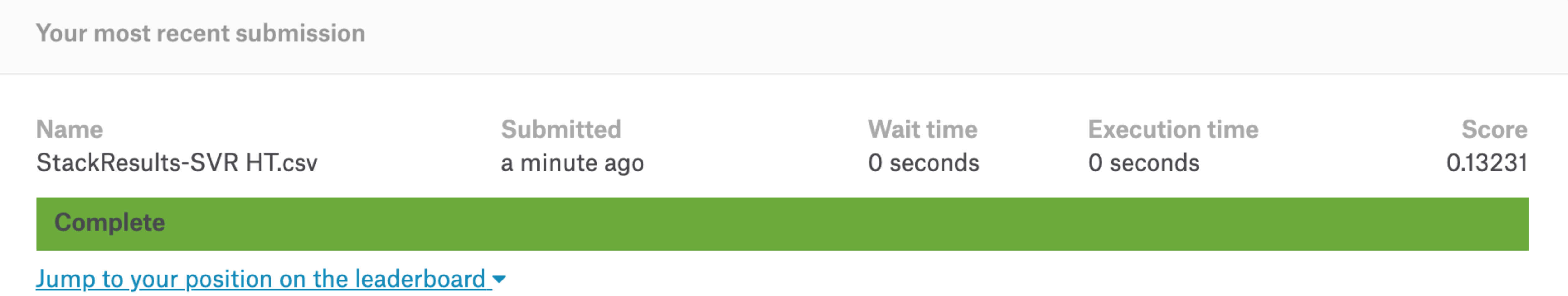
```
[ ] #Construct SVR using the tuned parameters
    svr_st= SVR(**rand_parm_svr_st)
    svr_st.fit(Sx_Train, Sy_Train)
    model_score = svr_st.score(Sx_Train, Sy_Train)
    print('Coefficient of determination R^2 of the prediction.:',model_score)
    y_predicted = svr_st.predict(Sx_Test_1)
    # The mean squared error
    print("Root Mean squared error : %.2f"% np.sqrt(mean_squared_error(Sy_Test_1, y_predicted)))
    # Explained variance score: 1 is perfect prediction
    print('Test Variance score : %.2f' % r2_score(Sy_Test_1, y_predicted))
```

```
#run cross-validation on best hyperparameters, get r2 score
svr_st_cv_score = cross_val_score(svr_st, Sx_Train, Sy_Train, cv=5 )
print("=== All R2 Scores ===")
print(svr_st_cv_score)
print('\n')
print("=== Mean R2 Score ===")
print("Mean R2 Score - SVR STACKING: ",svr_st_cv_score.mean())
```

Coefficient of determination R^2 of the prediction.:

0.8873469741428477

RMSE on test 0.13



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
StackResults-SVR HT.csv	a minute ago	0 seconds	0 seconds	0.13231

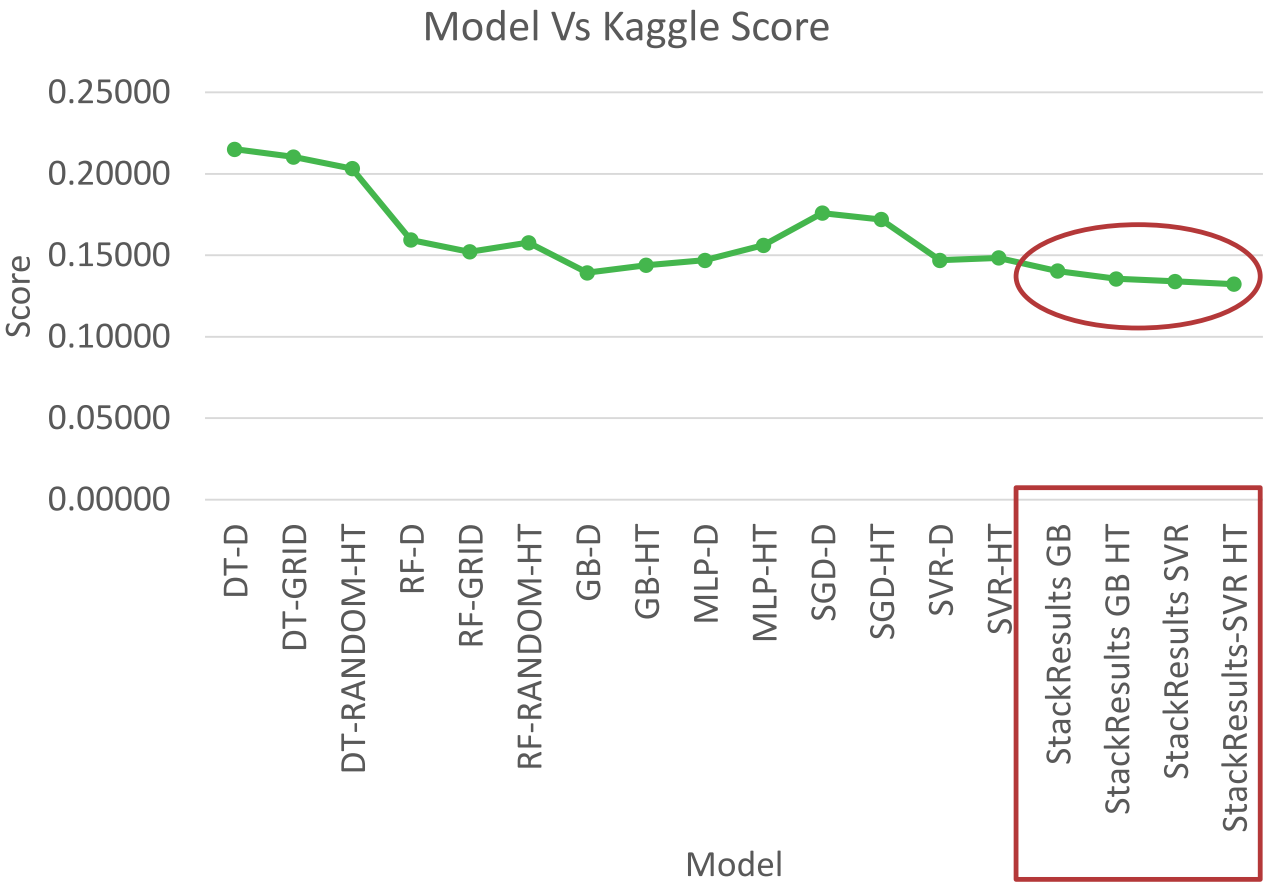
Complete

Jump to your position on the leaderboard

KAGGLE SCORE WITH SVR META MODEL HT

KAGGLE SCORE COMPARISON

Model	Kaggle Score
DT-D	0.21516
DT-GRID	0.21037
DT-RANDOM-HT	0.20322
RF-D	0.15939
RF-GRID	0.15210
RF-RANDOM-HT	0.15771
GB-D	0.13924
GB-HT	0.14386
MLP-D	0.14696
MLP-HT	0.15620
SGD-D	0.17593
SGD-HT	0.17201
SVR-D	0.14696
SVR-HT	0.14844
StackResults GB	0.14037
StackResults GB HT	0.13549
StackResults SVR	**0.13395**
StackResults-SVR HT	**0.13231**



KEY TAKE-AWAYS

- Hyperparameter tuning helped in improving the model.
- Stacking helped in significant improvement of the model.
- Surprisingly, Gradient Boosting with default parameters is performing at par with stacked models.
- ✓ Data normalization improved MLP, SGD, SVR models way high.

- THANK YOU