# 7. Reference

https://dev.to/arepp23/how-to-write-to-a-csv-file-in-c-1l5b

https://towardsdatascience.com/how-to-build-your-first-machine-learning-model-in-python-e70fd1907cdd

https://towardsdatascience.com/how-to-use-random-seeds-effectively-54a4cd855a79

# sorted\_ps = sorted(psutil.process\_iter(['name', 'cpu\_times']), key=lambda p: sum(p.info['cpu\_times'][:2]) if p.info['cpu\_times'] is not None else 0)

# for p in sorted\_ps:

# print(p.pid, p.info['name'])

# if p.info['cpu\_times'] is not None:

# print(sum(p.info['cpu\_times']))

# else:

# print(0)

# y\_pred = lr.predict(X\_test)

# r2\_score = lr.score(X\_test, y\_test)

# df=pd.DataFrame({'Actual':y\_test, 'Predicted':y\_pred})

# Rr = RandomForestRegressor(n\_estimators=50, max\_features=None, random\_state=0)

# r2\_score = Rr.score(X\_test, y\_test)

# # predicting value

# new\_prediction = Rr.predict((np.array([[700, 256, 2000, 0, 1, 1]])))

# print("Prediction performance:", float(new\_prediction))

# corrprocessData['PerformanceRating'].sort\_values(ascending=False)

# corrprocessData['PerformanceRating'].sort\_values(ascending=False).index[:-4:-1]

# # EmpEnvironmentSatisfaction, EmpLastSalaryHikePercent is having high Corr with PerformanceRating

# processData[corrprocessData['PerformanceRating'].sort\_values(ascending=False).index[0:4]].head()

# processData[corrprocessData['PerformanceRating'].sort\_values(ascending=False).index[[0,-1,-2,-3]]].head()

# ######################################### Decision Tree ###############################################

# from sklearn.tree import DecisionTreeClassifier

# dtc = DecisionTreeClassifier(criterion="entropy", max\_depth=8, min\_samples\_split=20, random\_state=99)

# features = [col for col in processData.columns if col not in objTypeCols.columns if col != "PerformanceRating"]

# dtc.fit(train[features], train["PerformanceRating"])

# preds = dtc.predict(test[features])

# dtc.score(test[features], test["PerformanceRating"]) # Accuracy

# # Comparing actual and predicted values using CrossTab function

# pd.crosstab(test["PerformanceRating"], preds, rownames=['Actual'], colnames=['Predictions'])

# dtc.feature\_importances\_

# tmp = pd.DataFrame(zip(features, dtc.feature\_importances\_), columns = ["Feature","Importance"])

# tmp = tmp.sort\_values(by='Importance', ascending=False)

# ## Applying k-fold cross validation

# from sklearn.cross\_validation import KFold

# crossvalidation = KFold(n=train[features].shape[0], n\_folds=10, shuffle=True, random\_state=12)

# # Finding Accuracy using K-Fold

# from sklearn.cross\_validation import cross\_val\_score

# score = np.mean(cross\_val\_score(dtc,train[features],train['PerformanceRating'],scoring='accuracy',cv=crossvalidation,n\_jobs=1))

# print(score)

# # Plotting the Decision Tree

# from sklearn.externals.six import StringIO

# from IPython.display import Image

# from sklearn.tree import export\_graphviz

# import pydotplus

# dot\_data = StringIO()

# export\_graphviz(dtc, out\_file="Employee\_Perf\_Analysis/decisionTree.dot", feature\_names=features,

# filled=True, rounded=True, special\_characters=True)

# # graph = pydotplus.graph\_from\_dot\_data("dt.dot")

# # graph.write\_jpg("dtree2.jpg")

# ######################################## Random Forest ############################################

# from sklearn.ensemble import RandomForestClassifier

# from sklearn.metrics import confusion\_matrix,accuracy\_score

# rfc = RandomForestClassifier(n\_estimators=300, random\_state=123)

# rfc.fit(train[features], train["PerformanceRating"])

# preds = rfc.predict(test[features])

# # pd.crosstab(test["PerformanceRating"], preds, rownames=['Actual'], colnames=['Predictions'])

# print(confusion\_matrix(test["PerformanceRating"], preds))

# print(accuracy\_score(test["PerformanceRating"], preds))

# # Feature importance

# featImp = pd.DataFrame(data=rfc.feature\_importances\_\*100.0, columns=["GiniValue"])

# featImp.index = features

# featImp.sort\_values(['GiniValue'], axis=0, ascending=False, inplace=True)

# print(featImp.head())

# """

# From above lines we can infer that the Feature which has the Highest Percentage affects the Performance Rating.

# 1)EmpLastSalaryHikePercent---21.24%

# 2)EmpEnvirnmentSatisfaction---20.21%

# 3)YearsSinceLastPromotion---10.06%

# are the three factors affecting the Performance Rating

# Obeservations from the above line:

# 1) Employee who has the Highest Performance rating has more Environment Satisfaction and Employee

# who has the low arting has less Environment Satisfaction

# 2) Employee who has Highest performance rating has higest Salary hike Percentage

# 3) Employee who has Lowest Performance rating is the Current role for more number of years,

# employee who has the highest Performance rating has lowest number of Years in the Current Role.

# This implies that there is no career growth

# """

# # Feature Selection

# from sklearn.feature\_selection import RFECV

# clf\_rf\_4 = RandomForestClassifier()

# rfecv = RFECV(estimator=clf\_rf\_4, step=1, cv=5, scoring='accuracy') #5-fold cross-validation

# rfecv = rfecv.fit(train[features], train["PerformanceRating"])

# print('Optimal number of features :', rfecv.n\_features\_)

# print('Best features :', train[features].columns[rfecv.support\_])

# # Visualising the Random Forest Regression Results

X\_grid = np.arange(min(X), max(X), 0.01)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X\_test, y\_test, color = 'red')

plt.scatter(X\_test, y\_pred, color = 'green')

plt.title('Random Forest Regression')

plt.xlabel('Temperature')

plt.ylabel('Revenue')

plt.show()

plt.plot(X\_grid, regressor.predict(X\_grid), color = 'black')

plt.title('Random Forest Regression')

plt.xlabel('Temperature')

plt.ylabel('Revenue')

plt.show(

Random forest regressor

# Check the training error

np.sqrt(np.mean((ylog\_pred - ylog1p\_train)\*\*2)) # about 0.37 (if you use 100 trees)

np.sqrt(np.mean((Model.oob\_prediction\_ - ylog1p\_train)\*\*2)) # 0.47 slightly better than a simple tree.

# Create a dataframe of the variable importances

df\_ = pd.DataFrame(df\_all.columns, columns = ['feature'])

df\_['fscore'] = Model.feature\_importances\_[:, ]

In [23]:

# Plot the relative importance of the top 10 features

df\_['fscore'] = df\_['fscore'] / df\_['fscore'].max()

df\_.sort\_values('fscore', ascending = False, inplace = True)

df\_ = df\_[0:10]

df\_.sort\_values('fscore', ascending = True, inplace = True)

df\_.plot(kind='barh', x='feature', y='fscore', legend=False, figsize=(6, 10))

plt.title('Random forest feature importance', fontsize = 24)

plt.xlabel('')

plt.ylabel('')

plt.xticks([], [])

plt.yticks(fontsize=20)

plt.show()

#plt.gcf().savefig('feature\_importance\_xgb.png')