# # 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))

# # 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')