MODEL BUILDING

```
fd.corr()

sns.pairplot(fd)

corrmat=fd.corr()

top_corr_features=corrmat.index

plt.figure(figsize=(20,20))

g=sns.heatmap(fd[top_corr_features].corr(),annot=True,cmap="RdYlGn")


fd.head()


#independent feature and dependent feature

x=fd.iloc[:,1:]

y=fd.iloc[:,0]


x.head()

y.head()


##feature importance

from sklearn.ensemble import ExtraTreesRegressor

model=ExtraTreesRegressor()

model.fit(x,y)


print(model.feature_importances_)


feat_importances=pd.Series(model.feature_importances_,index=x.columns)

feat_importances.nlargest(5).plot(kind='barh')

plt.show()


from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

from sklearn.ensemble import RandomForestRegressor
regressor=RandomForestRegressor()
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
print(n_estimators)

from sklearn.model_selection import RandomizedSearchCV
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

```
print(random_grid)


# Use the random grid to search for best hyperparameters

# First create the base model to tune

rf = RandomForestRegressor()


# Random search of parameters, using 3 fold cross validation,

# search across 100 different combinations

rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
random_grid,scoring='neg_mean_squared_error', n_iter = 10, cv = 5, verbose=2,
random_state=42, n_jobs = 1)


rf_random.fit(x_train,y_train)


rf_random.best_params_


rf_random.best_score_

predictions=rf_random.predict(x_test)

sns.distplot(y_test-predictions)

plt.scatter(y_test,predictions)


from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))

print('MSE:', metrics.mean_squared_error(y_test, predictions))

print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))


import pickle

# open a file, where you want to store the data

file = open('random_forest_regression_model.pkl', 'wb')

pickle.dump(rf_random, file)
```