Project Crystal Ball Earth: Earthquake Travel-Time Prediction

Mahamadou Dagnoko[1], Cassandra De Leon[1], Muhammad Rehan[1], Michelle Wheatley[2], Christopher Donan[3]

[1]University of Houston Main; [2]University of Houston Clear Lake; [3]Sam Houston State University

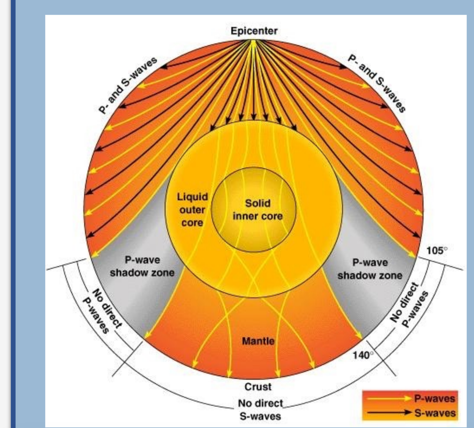
Abstract

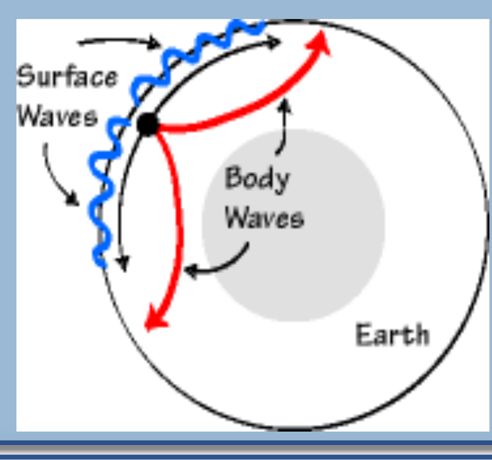
Predicting the travel time of seismic waves from an earthquake to a receiver is challenging due to the labyrinthine path of waves through the Earth's layers. This complexity is evident in the limitations of our current formulas and models. By selecting and testing different features, this study was able to narrow down the best predictors to build and train different models. We then proceeded by tuning our hyper-parameters to find the best bias-variance trade off for all the models. Overall, the best models were able to outperform the current formula in predicting the real travel time of seismic waves. This highlights the need for refined machine-learning tools that can improve seismic waves prediction.

Problem Statement

As seismic waves travels through earth's layers, their travel path changes depending on each layer they go trough as you can see below, this makes predicting their travel time before reaching the surface difficult.

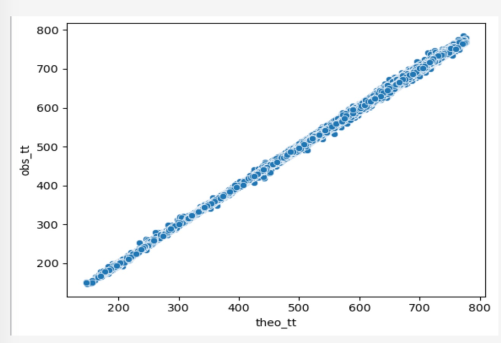
Solving this problem will provide valuable information for understanding the Earth's structure, tectonic plate movements, and earthquake mechanisms. This knowledge can help us get a deeper understanding of Earth's geology. This in turn can be crucial to the development of new technologies to reduce the effects of earthquakes. In sum, an accurate prediction of seismic waves can save lives, reduce property damage, improve building design, allocate resources effectively, and advance scientific research.



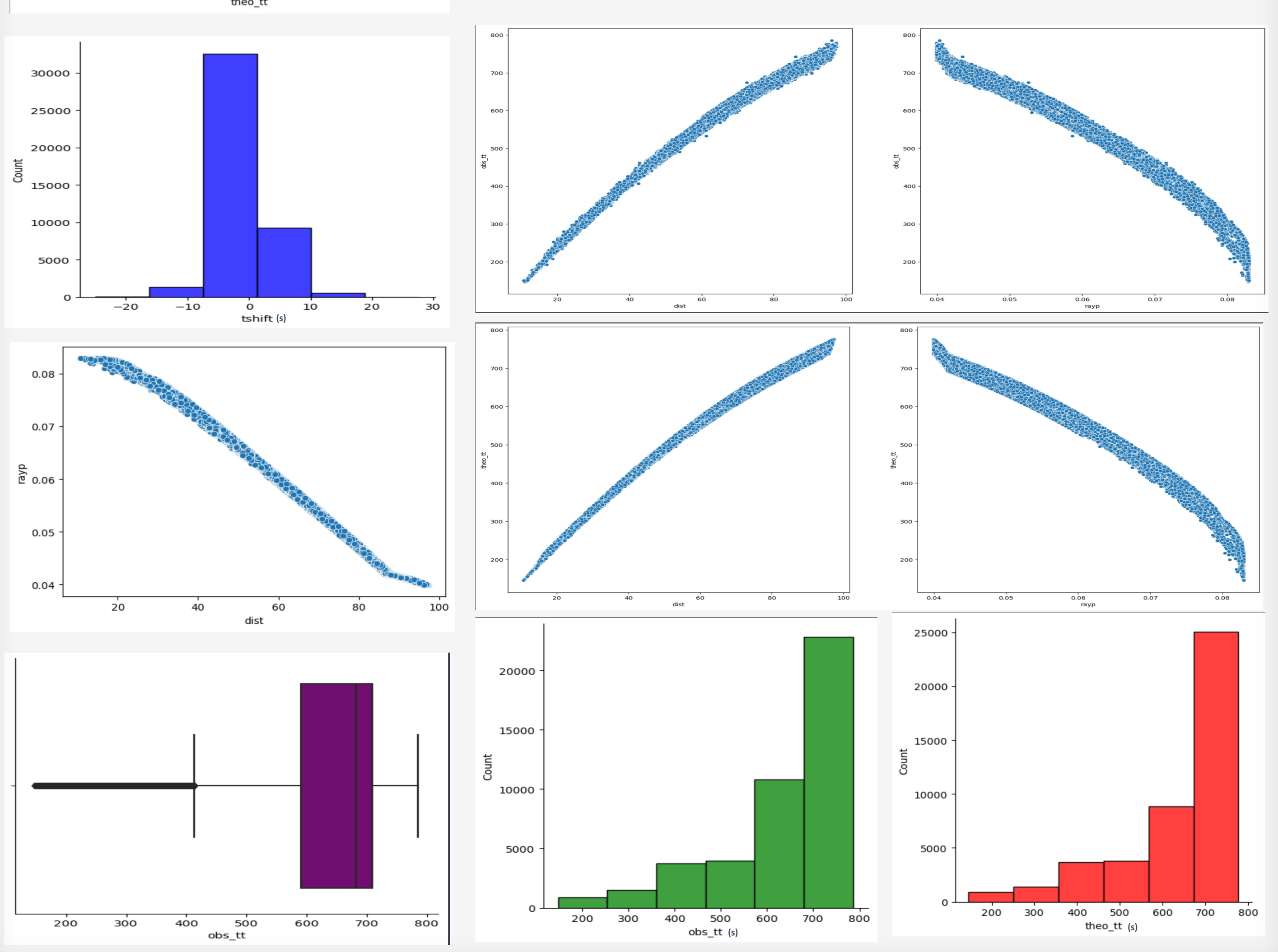


The initial columns are Filename, theo_tt, tshift, obs_tt, polarity, stnm,rayp, stla, stel, evla, evlo, evdp, dist, az, baz. Obs_tt is the target variable.

Data Visualization

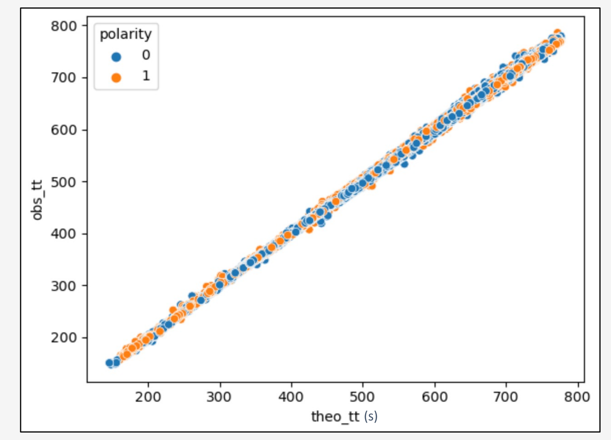


The correlation between dist and rayp, theo_tt and dist, obs_tt and dist are respectively -0.997, 0.9925, 0.992500. Those are correlated among themselves, so some of them must be dropped. theo_tt is the most relevant with a correlation of 0.9997 and has a linear relation wih obs_tt.



Selected Features

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(0,)	[-10.669564464818311]	-10.669564	(theo_tt,)	NaN	0.0	NaN
2	(0, 6)	[-10.396257167400291]	-10.396257	(theo_tt, evdp)	NaN	0.0	NaN
3	(0, 1, 6)	[-10.2319281560013]	-10.231928	(theo_tt, stla, evdp)	NaN	0.0	NaN
4	(0, 1, 5, 6)	[-10.19400930133311]	-10.194009	(theo_tt, stla, evlo, evdp)	NaN	0.0	NaN
5	(0, 1, 3, 5, 6)	[-10.158479634380443]	-10.15848	(theo_tt, stla, stel, evlo, evdp)	NaN	0.0	NaN
6	(0, 1, 3, 4, 5, 6)	[-10.131353793823584]	-10.131354	(theo_tt, stla, stel, evla, evlo, evdp)	NaN	0.0	NaN
7	(0, 1, 2, 3, 4, 5, 6)	[-10.117085585950939]	-10.117086	(theo_tt, stla, stlo, stel, evla, evlo, evdp)	NaN	0.0	NaN



The feature polarity is either a positive or negative state, relative to the original waveform. We can flip the polarity and since it is controllable, it is not relevant as a predictor as we can observe in the graph above. So, the remaining relevant features are 'theo_tt', 'stla', 'stlo', 'stel', 'evla', 'evlo', 'evdp.' We passed these features to a forward selection algorithm and all these features were deemed relevant to predict 'obs_tt.'

Machine Learning Models

Procedure

After finding out our best subset, the next step was to find the best model. To do that, we split the data in three (50%, 25%, 25%). 50% of the data is used for training, 25% for testing and the remaining 25 % is used as new observation to test our models on unseen data.

What are we trying to beat?

We are trying to beat 10.9022 secinds which is the mean squared error between theo_tt and obs_tt. The error in prediction is the time shift (tshift). The distribution of the error can be seen in Data Visualization.

Linear Regression

```
lm_score = cross_val_score(LinearRegression(), dataset.drop('obs_tt', axis=1),
                           dataset['obs_tt'], cv = 3, scoring = 'neg_mean_squared_error')
np.mean(lm_score)

-10.120434378630415
```

Lasso Regression

```
mean_squared_error(dataset['theo_tt'], dataset['obs_tt'])

10.902170189379193
lasso_model = linear_model.Lasso(alpha=0.001, max_iter=100, tol=0.0001)
lasso_score = cross_val_score(lasso_model,  dataset.drop('obs_tt', axis=1),
                              dataset['obs_tt'], cv = 3, scoring = 'neg_mean_squared_error')
np.mean(lasso_score)

-10.120435359102094
```

Decision Tree

```
x = dataset.drop('obs_tt', axis = 1)
y = dataset['obs_tt']
tree_model = DecisionTreeRegressor (random_state=1)
model_evaluate(x, y, tree_model)

6.043171377191967
```

	0
stel	0.000028
stla	0.000048
stlo	0.000073
evla	0.000106
evlo	0.000112
evdp	0.000214
theo_tt	0.999419

Random Forest

```
rf_model  = RandomForestRegressor (random_state=1)
model_evaluate(x, y, rf_model)

3.312576821148712
rf_score = cross_val_score(RandomForestRegressor(), dataset.drop(['obs_tt'], axis = 1),
                           dataset['obs_tt'], cv = 3, scoring = 'neg_mean_squared_error')
np.mean(rf_score)

-3.028738040262948
```

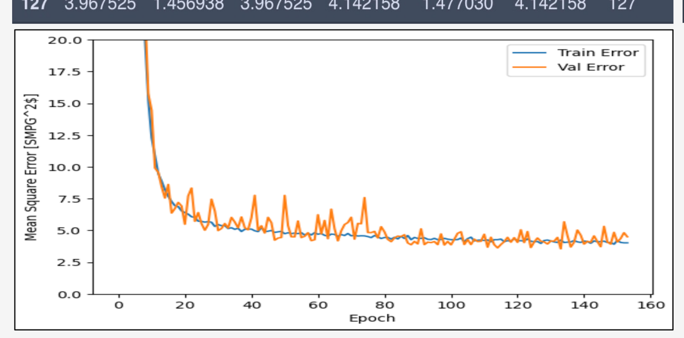
Gradient Boosting

params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
{'learning_rate': 0.5, 'n_estimators': 100}	-3.808205	-3.559288	-3.575564	-3.647686	0.113699	4
{'learning_rate': 0.5, 'n_estimators': 1000}	-1.666918	-1.757664	-1.621635	-1.682072	0.056558	1
{'learning_rate': 1, 'n_estimators': 100}	-3.271593	-3.701815	-3.422985	-3.465464	0.178187	3
{'learning_rate': 1, 'n_estimators': 1000}	-2.128162	-2.395794	-2.153529	-2.225828	0.120629	2

Neural Network (optimizer = Adam(0.001))

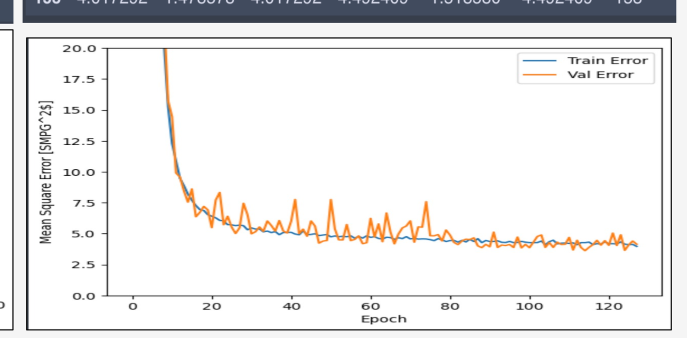
```
optimizer = optimizers.Adam(0.001)
neural_model.compile(loss='mse', optimizer = optimizer, metrics= ['mae','mse'])

history = neural_model.fit(x, y, epochs = 128, validation_split=0.334)
```

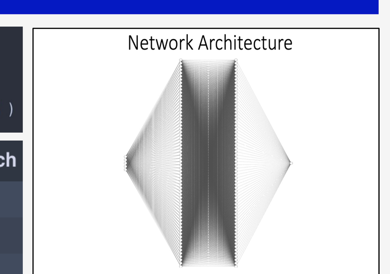
	loss	mae	mse	val_loss	val_mae	val_mse	epoch
123	4.327561	1.544589	4.327561	4.905277	1.613286	4.905277	123
124	4.142728	1.498574	4.142728	3.643636	1.366613	3.643636	124
125	4.102790	1.489783	4.102790	4.081650	1.468914	4.081650	125
126	4.126263	1.493542	4.126263	4.399500	1.554043	4.399500	126
127	3.967525	1.456938	3.967525	4.142158	1.477030	4.142158	127




```
early_stop = callbacks.EarlyStopping(monitor= 'val_loss', patience = 20)
history = neural_model.fit(x, y, epochs = 500, validation_split=0.334,
                           callbacks = [early_stop, PrintDot()], verbose=0 )
```

	loss	mae	mse	val_loss	val_mae	val_mse	epoch
149	3.906017	1.450457	3.906017	4.842895	1.683928	4.842895	149
150	4.208082	1.519250	4.208082	4.085391	1.483590	4.085391	150
151	4.047722	1.487720	4.047722	4.358139	1.544688	4.358139	151
152	4.017493	1.478220	4.017493	4.816047	1.625726	4.816047	152
153	4.017292	1.478578	4.017292	4.492409	1.518580	4.492409	153



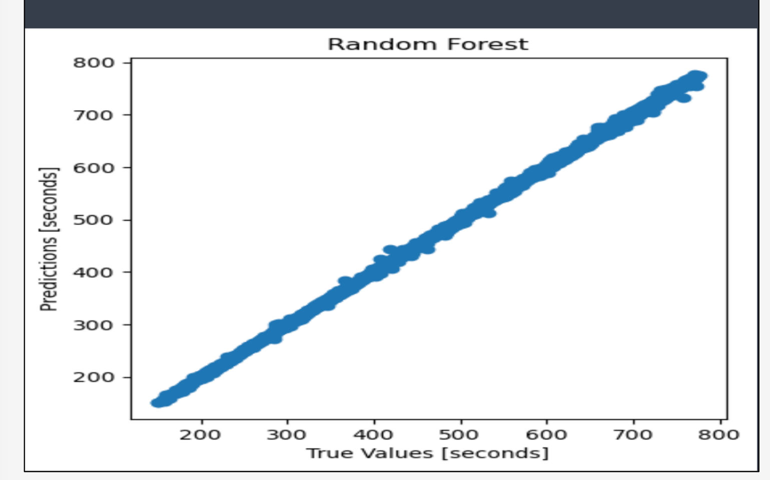


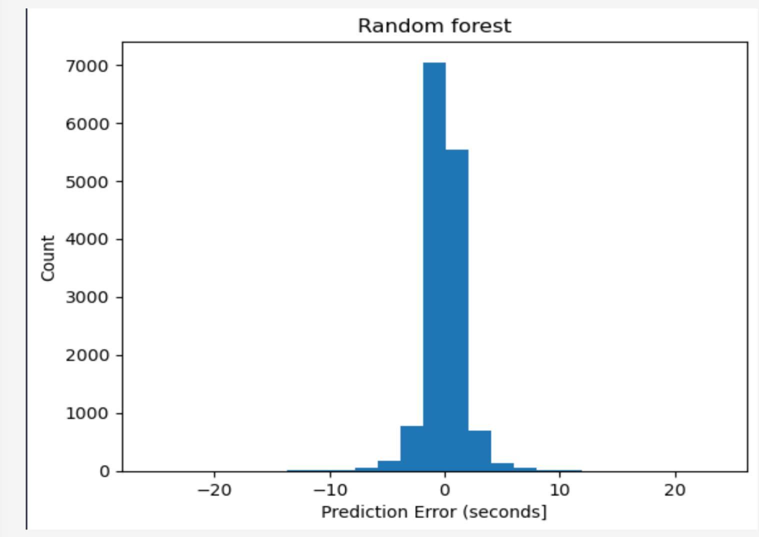
We started by building an initial model. Then the models have been optimized using an early stop algorithm.

TESTING BEST MODELS ON NEW OBSERVATIONS

Random Forest

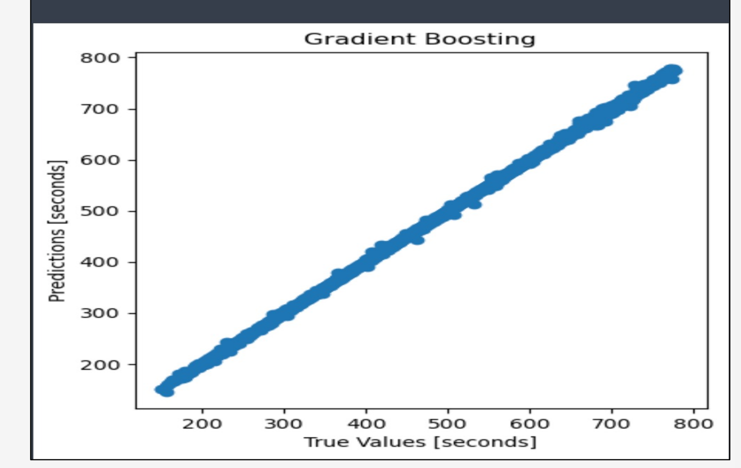
```
testing set Mean Squared Error: 2.9199 seconds
```

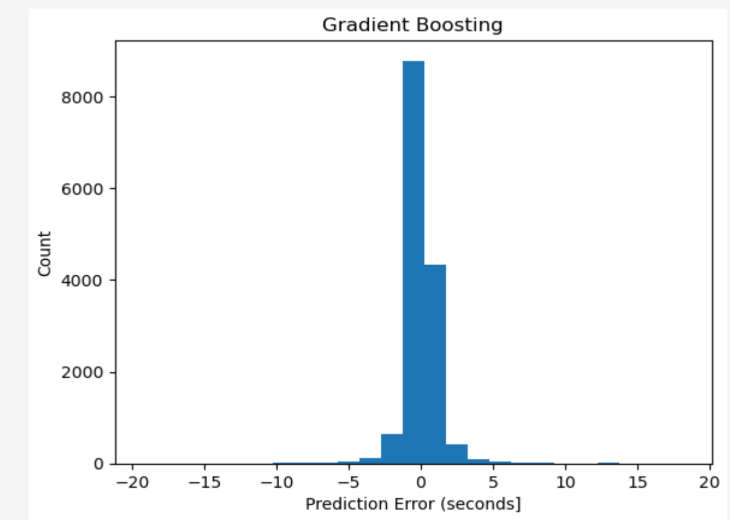




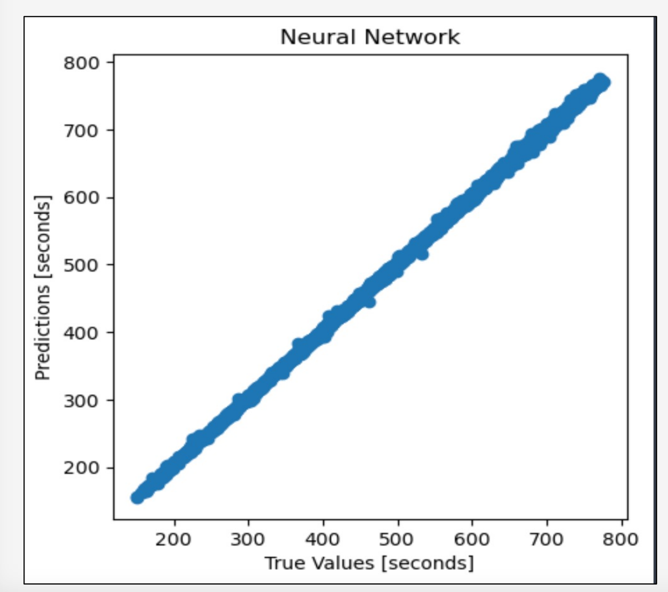
Gradient Boosting

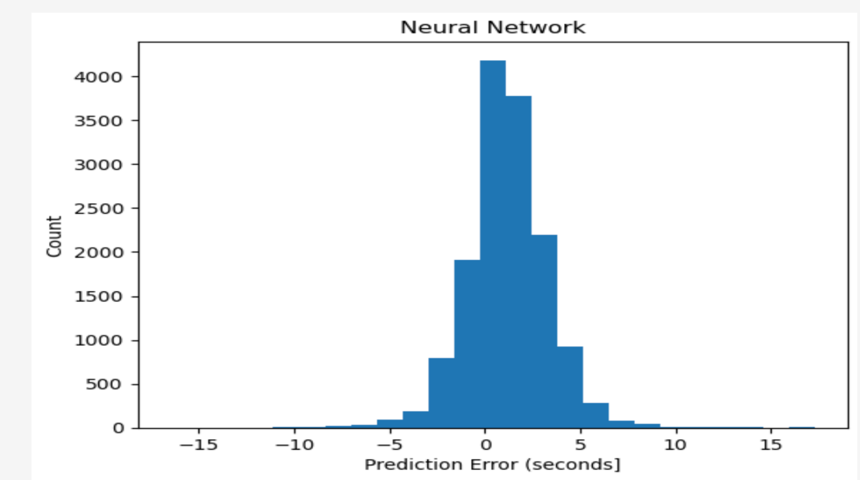
```
testing set Mean Squared Error: 1.5586 seconds
```





Neural Network






```
testing set Mean Squared Error: 5.9800 seconds
```

Conclusion

The Exploratory Data Analysis helped us select the best features for this task. These features are 'theo_tt', 'stla', 'stlo', 'stel', 'evla', 'evlo', 'evdp.'

Then, we tested different machine learning models to predict waves travel times and our best machine learning algorithm is the Gradient Boosting with a mean squared error of 1.56 seconds. Machine learning Boosting builds an initial model to fit the data and follows that by building a second model while correcting the inaccuracies of the first model. By doing that multiple time, the combination of these models produce a stronger and better model.

The next best model is the Random Forest model with a mean squared error of 2.92 second. A Random Forest combines the output of multiple decision trees to output a single result. In the case of regression, it uses the average prediction of all the trees making it more accurate and thus usually does better than a single decision tree.

The third best model is the Neural Network with a mean squared error of 5.98 seconds. A Neural Network uses interconnected nodes that works like neurons. Using algorithms, these nodes can learn patterns, cluster, classify, and improve overtime.

In sum, our best models did better than the theoretical values which has a mean squared error of 10.90 seconds. Therefore, using machine learning methods is the better way to predict waves travel time.