In [1]:
```
# Determine if random walk
# Import packages
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
```

In [2]:
```
# Data preparation
m_list = ['Persistence Model', 'Decision Tree Model', 'Random Forest Model', '
mse_list = []
df = pd.read_csv(r'C:\Users\huang\OneDrive\Documents\ECO481\Bone Prices\merged
# We must make the training and test split, keep in mind it is timeseries data
x_train = df[['Lagged Total Trade Volume', 'Lagged Price', 'Lagged Total Trade
              'Lagged Price Substitute 1', 'Lagged Total Trade volume Substitut
              'Lagged Total Trade volume Substitute 3', 'Lagged Price Substitut
              'Lagged Price Substitute 4']].loc[1: 4238]
x_test = df[['Lagged Total Trade Volume', 'Lagged Price', 'Lagged Total Trade
              'Lagged Price Substitute 1', 'Lagged Total Trade volume Substitut
              'Lagged Total Trade volume Substitute 3', 'Lagged Price Substitut
              'Lagged Price Substitute 4']].loc[4239: 5298]
y_train = df[['Average Price']].loc[1: 4238]
y_test = df[['Average Price']].loc[4239: 5298]
```

In [3]:
```
# Scale the data:
scaler = preprocessing.StandardScaler().fit(x_train)
x_training_scaled = scaler.transform(x_train)
x_testing_scaled = scaler.transform(x_test)
```

In [4]:
```
# Create a persistence model ~ naive predictor, should be shit but if it has a
# perhaps we are dealing with something akin to random selection...
def model_persistence(x):
    return x
# walk-forward validation
predictions = list()
for x in x_test['Lagged Price']:
    yhat = model_persistence(x)
    predictions.append(yhat)
test_score = mean_squared_error(y_test, predictions)
print('Test MSE: %.3f' % test_score)
mse_list.append(test_score)
```

```
Test MSE: 928.501
```

In [5]:
```
# decision tree regressor
tree_reg = DecisionTreeRegressor()
d_tree = DecisionTreeRegressor(max_depth = 5)
tree_reg.fit(x_training_scaled, y_train)
rt_pred = tree_reg.predict(x_testing_scaled)
# MSE:
test_score_rt = mean_squared_error(y_test, rt_pred)
print('Test MSE: %.3f' % test_score_rt)
mse_list.append(test_score_rt)
```

```
Test MSE: 1655.198
```

In [6]:
```
# We now go on to run a random forest
forest = RandomForestRegressor()
forest.fit(x_training_scaled, y_train)
# Print the RF MSE
rf_pred = forest.predict(x_testing_scaled)
test_score_rf = mean_squared_error(y_test, rf_pred)
print('Test MSE: %.3f' % test_score_rf)
# OH SHIT ITS RANDOM WAAAAAAALK
# WAAAAAAAAAAAAAAAAGH
# But to be fair ~ have we tried GRADIENT BOOSTED DECISION TREES?
mse_list.append(test_score_rf)
# We determined it to be random walk as the other methods seem to have
# a higher MSE than the persistence MSE
```

```
C:\Users\huang\AppData\Local\Temp\ipykernel_24464\596940956.py:3: DataConvers
ionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  forest.fit(x_training_scaled, y_train)

Test MSE: 1204.756
```

In [7]:
```
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBClassifier

GBC = GradientBoostingRegressor(n_estimators = 100, learning_rate = 0.1,
                                max_leaf_nodes = None, criterion = "squared_e
# Make the fit?
GBC.fit(x_training_scaled, y_train)
gbc_pred = GBC.predict(x_testing_scaled)
test_score_gbc = mean_squared_error(y_test, gbc_pred)
print('Test MSE: %.3f' % test_score_gbc)
mse_list.append(test_score_gbc)
```

```
C:\Users\huang\anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:494: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

Test MSE: 1315.572
```

In [8]:
```
# knn model
# Create a KNN model, hyperparameter tuning ~ we want to find the optimal # of
from sklearn.model_selection import cross_val_score
k_list = []
cv_l = []
mse_l = []
ind_knn = 1
knn_df1 = scaler.transform(df[['Lagged Total Trade Volume', 'Lagged Price', 'L
             'Lagged Price Substitute 1', 'Lagged Total Trade volume Substitut
             'Lagged Total Trade volume Substitute 3', 'Lagged Price Substitut
             'Lagged Price Substitute 4']])
knn_predicted = df[['Average Price']]
while ind_knn in range(11):
    # neighbor list
    k_list.append(ind_knn)
    KNN = KNeighborsRegressor(n_neighbors = ind_knn)
    KNN.fit(x_training_scaled, y_train)
    # cross validation
    cross_val_scores = cross_val_score(KNN, knn_df1, knn_predicted, cv = 5)
    cross_val_mean = cross_val_scores.mean()
    cv_l.append(cross_val_mean)
    # MSE
    knn_pred = KNN.predict(x_testing_scaled)
    test_score_knn = mean_squared_error(y_test, knn_pred)
    mse_l.append(test_score_knn)
    ind_knn += 1
```

In [9]:
```
knn_outcomes_df = pd.DataFrame(list(zip(k_list, cv_l, mse_l)), columns = ['Num
                                                                          'MSE'

knn_outcomes_df
# We want the maximum CV score of it, along with requisite MSE:
```

Out[9]:

	Number of Neighbors	Cross Validation Score	MSE
0	1	-0.084885	2140.521513
1	2	0.116399	1755.548753
2	3	0.191548	1603.406411
3	4	0.204153	1478.253437
4	5	0.188064	1435.853168
5	6	0.179343	1470.942421
6	7	0.159486	1520.109223
7	8	0.143001	1543.086383
8	9	0.109816	1569.521246
9	10	0.089514	1575.464873

In [10]:
```
ind_knn1 = 0
max_cv = knn_outcomes_df['Cross Validation Score'][ind_knn1]
while ind_knn1 in range(len(knn_outcomes_df['Cross Validation Score'])):
    if knn_outcomes_df['Cross Validation Score'][ind_knn1] > max_cv:
        max_cv = knn_outcomes_df['Cross Validation Score'][ind_knn1]
        max_ind = ind_knn1
    else:
        pass
    ind_knn1 += 1
mse_list.append(knn_outcomes_df['MSE'][max_ind])
```

In [11]:
```
outcomes_df = pd.DataFrame(list(zip(m_list, mse_list)), columns = ['Model', 'T
outcomes_df
```

Out[11]:

	Model	Test MSE
0	Persistence Model	928.500905
1	Decision Tree Model	1655.198362
2	Random Forest Model	1204.756096
3	Gradient Boosted Tree	1315.572298
4	KNN	1478.253437

In [ ]:
```

```