Predicting Car Stopping Distance

The goal of this project is to predict a car's stopping distance by constructing a predictive model that will utilize the car's speed and the corresponding stopping distance.

Data

The source of the dataset is Ezekiel, M. (1930) Methods of Correlation Analysis. Wiley. They note the dataset is from the decade of the last century - the 1920's.

The dataset has two numerical attributes:

- speed - speed of the car in mph
- dist - stopping distance in feet

In [ ]:
```
#basic libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#train test split, grid search
from sklearn.model_selection import train_test_split, GridSearchCV

#regression models
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

#evaluation
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

In [ ]:
```
#loading the data
data = pd.read_csv('cars.csv', index_col=0)
```

Data Exploration and Visualization

In [ ]:
```
data.head()
```

Out[ ]:

	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16

In [ ]:
```
#basic statistics
data.describe()
```

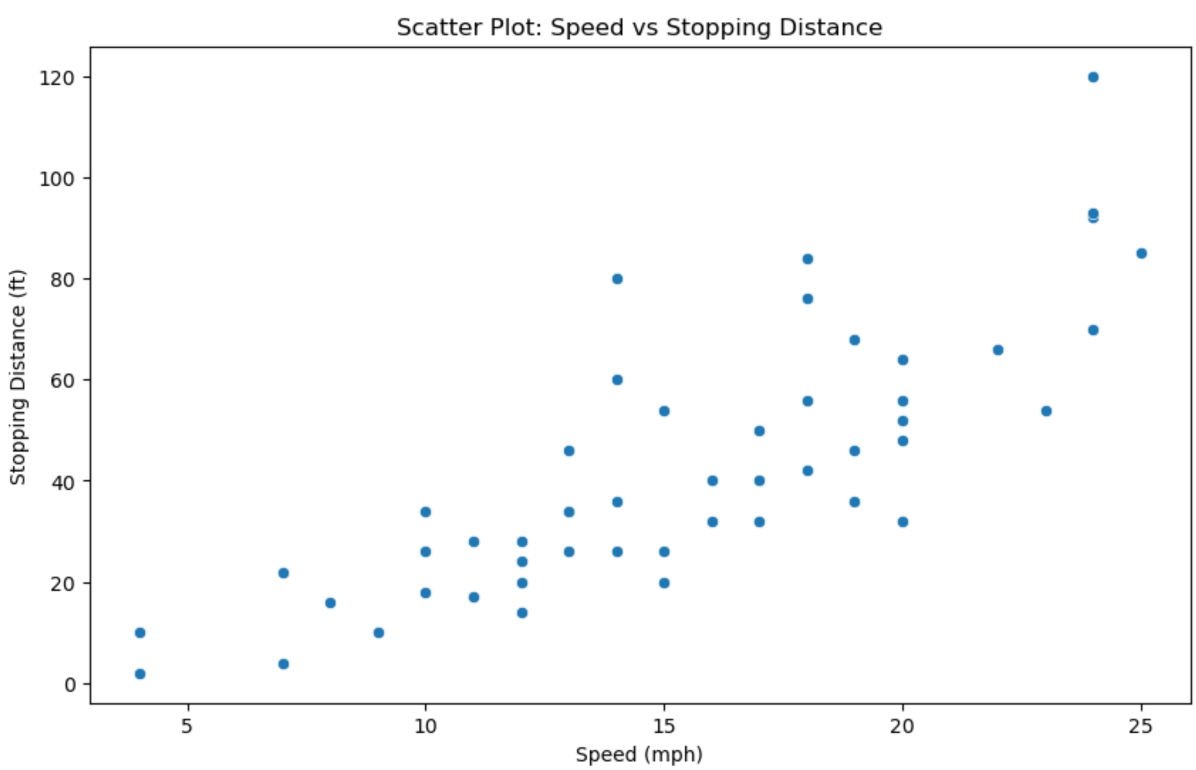
Out[ ]:

	speed	dist
count	50.000000	50.000000
mean	15.400000	42.980000
std	5.287644	25.769377
min	4.000000	2.000000
25%	12.000000	26.000000
50%	15.000000	36.000000
75%	19.000000	56.000000
max	25.000000	120.000000

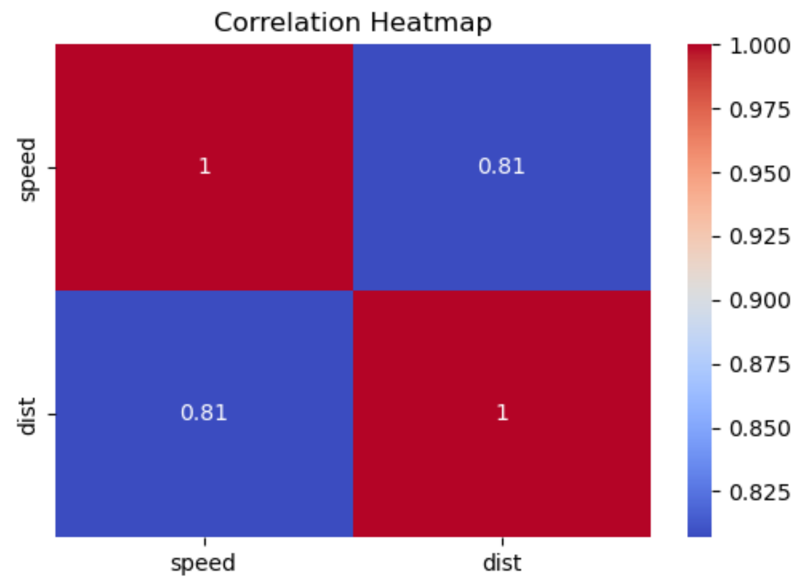
In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 1 to 50
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   speed   50 non-null     int64
 1   dist    50 non-null     int64
dtypes: int64(2)
memory usage: 1.2 KB
```

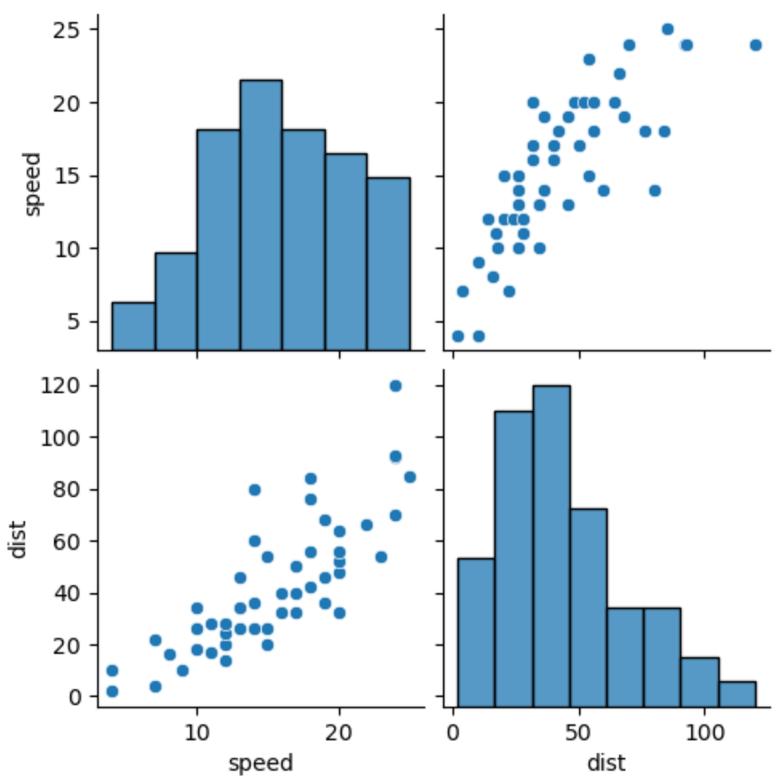
In [ ]:
```
#speed and stopping distance scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='speed', y='dist', data=data)
plt.title('Scatter Plot: Speed vs Stopping Distance')
plt.xlabel('Speed (mph)')
plt.ylabel('Stopping Distance (ft)')
plt.show()
```



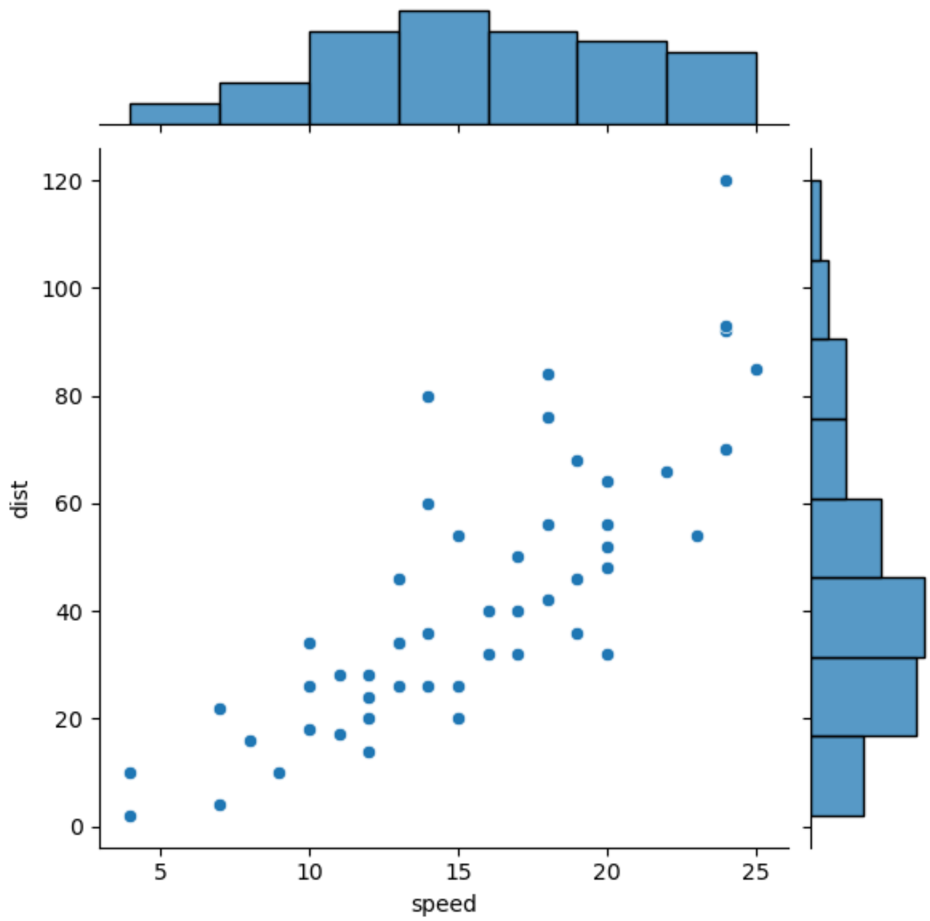
In [ ]:
```
#correlation
plt.figure(figsize=(6, 4))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



In [ ]:
```
#pairplot
sns.pairplot(data)
plt.show()
```



In [ ]:
```
#jointplot
sns.jointplot(x='speed', y='dist', data=data, kind='scatter')
plt.show()
```



Data Modeling

In [ ]:
```
#splitting the data into a train and test set
X = data[['speed']]
y = data['dist']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fitting different regression models

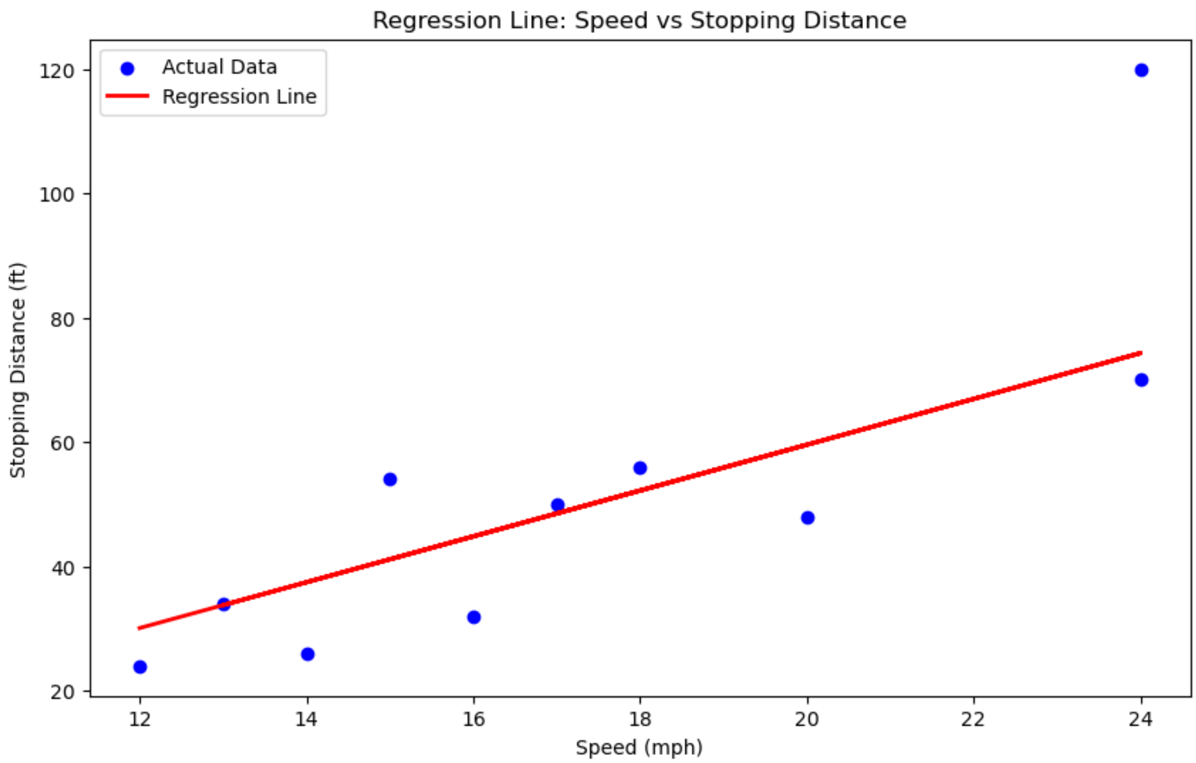
In [ ]:
```python
#fitting models

#Linear Regression
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)

#Polynomial
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train)
model_poly = LinearRegression()
model_poly.fit(X_poly, y_train)

#Random Forest
model_rf = RandomForestRegressor(oob_score=True, random_state=42)
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}
grid_rf = GridSearchCV(model_rf, param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)

#decicion tree
model_dt = DecisionTreeRegressor(random_state=42)
param_grid_dt = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10]
}
grid_dt = GridSearchCV(model_dt, param_grid_dt, cv=5)
grid_dt.fit(X_train, y_train)

#support vector machines
model_svr = SVR(kernel='rbf')
model_svr.fit(X_train, y_train)

#gradient boost
model_gb = GradientBoostingRegressor(random_state=42)
param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2]
}
grid_gb = GridSearchCV(model_gb, param_grid_gb, cv=5)
grid_gb.fit(X_train, y_train)
```

Out[ ]:
```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(random_state=42),
             param_grid={'learning_rate': [0.01, 0.1, 0.2],
                         'max_depth': [3, 5, 7],
                         'n_estimators': [100, 200, 300]})
```

Model Evaluation

In [ ]:
```
#evaluation instances
y_pred_lr = model_lr.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

X_poly_test = poly.transform(X_test)
y_pred_poly = model_poly.predict(X_poly_test)
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

y_pred_rf = grid_rf.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

y_pred_svr = model_svr.predict(X_test)
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)

y_pred_gb = grid_gb.predict(X_test)
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

y_pred_dt = grid_dt.predict(X_test)
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)
```

In [ ]:
```
#linreg regression line and actual data points
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Data')
plt.plot(X_test, y_pred_lr, color='red', linewidth=2, label='Regression Line')

plt.title('Regression Line: Speed vs Stopping Distance')
plt.xlabel('Speed (mph)')
plt.ylabel('Stopping Distance (ft)')
plt.legend()
plt.show()
```



In [ ]:
```
#evaluation dataframe
models = ['Linear Regression', 'Polynomial Regression', 'Random Forest', 'SVR', 'Gradient Boosting', 'Decision Tree']
y_preds = [y_pred_lr, y_pred_poly, y_pred_rf, y_pred_svr, y_pred_gb, y_pred_dt]
mse_scores = [mse_lr, mse_poly, mse_rf, mse_svr, mse_gb, mse_dt]
r2_scores = [r2_lr, r2_poly, r2_rf, r2_svr, r2_gb, r2_dt]

# Create a DataFrame to store the evaluation scores
evaluation_df = pd.DataFrame({
    'Model': models,
    'Mean Squared Error (MSE)': mse_scores,
    'R-squared (R2) Score': r2_scores
})

evaluation_df
```

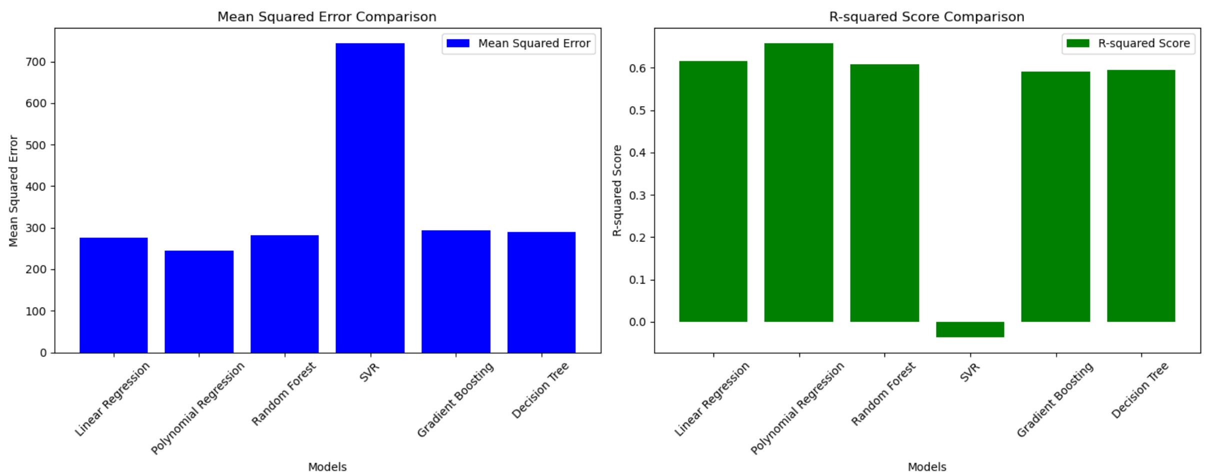
Out[ ]:

	Model	Mean Squared Error (MSE)	R-squared (R2) Score
0	Linear Regression	275.428983	0.615773
1	Polynomial Regression	244.521644	0.658890
2	Random Forest	281.244514	0.607661
3	SVR	743.360370	-0.036996
4	Gradient Boosting	293.433486	0.590657
5	Decision Tree	290.146785	0.595242

Model Comparison and Visualizations

In [ ]:

```
#plotting MSE and R2
plt.figure(figsize=(15, 6))

#MSE
plt.subplot(1, 2, 1)
plt.bar(models, mse_scores, color='blue', label='Mean Squared Error')
plt.title('Mean Squared Error Comparison')
plt.xlabel('Models')
plt.ylabel('Mean Squared Error')
plt.xticks(rotation=45)
plt.legend()

#R2
plt.subplot(1, 2, 2)
plt.bar(models, r2_scores, color='green', label='R-squared Score')
plt.title('R-squared Score Comparison')
plt.xlabel('Models')
plt.ylabel('R-squared Score')
plt.xticks(rotation=45)
plt.legend()

plt.tight_layout()
plt.show()
```



Additional Statistics

In [ ]:

```
#linear regression
mae_lr = mean_absolute_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))

#random forest
y_pred_rf_train = grid_rf.predict(X_train)
oob_error_rf = 1 - grid_rf.best_estimator_.oob_score_


#gradient boosting
y_pred_gb_train = grid_gb.predict(X_train)  # Predictions on the training set for learnin
g curve
train_errors_gb = [mean_squared_error(y_train, y_pred) for y_pred in grid_gb.best_estimat
or_.staged_predict(X_train)]
validation_errors_gb = [mean_squared_error(y_test, y_pred) for y_pred in grid_gb.best_est
imator_.staged_predict(X_test)]

#some general metrics
adjusted_r2_lr = 1 - (1 - r2_lr) * (len(y_test) - 1) / (len(y_test) - X_test.shape[1] -
1)
mpe_lr = np.mean((y_test - y_pred_lr) / y_test) * 100
mape_lr = np.mean(np.abs((y_test - y_pred_lr) / y_test)) * 100

#displaying additional statistics
print("Linear Regression Additional Statistics:")
print(f'Mean Absolute Error (MAE): {mae_lr:.2f}')
print(f'Root Mean Squared Error (RMSE): {rmse_lr:.2f}')
print(f'Adjusted R-squared: {adjusted_r2_lr:.2f}')
print(f'Mean Percentage Error (MPE): {mpe_lr:.2f}%')
print(f'Mean Absolute Percentage Error (MAPE): {mape_lr:.2f}%')

print("\nRandom Forest Additional Statistics:")
print(f'Out-of-Bag (OOB) Error: {oob_error_rf:.4f}')

print("\nGradient Boosting Additional Statistics:")
print(f'Learning Curve - Train Errors: {train_errors_gb}')
print(f'Learning Curve - Validation Errors: {validation_errors_gb}')
```

```
Linear Regression Additional Statistics:
Mean Absolute Error (MAE): 11.03
Root Mean Squared Error (RMSE): 16.60
Adjusted R-squared: 0.57
Mean Percentage Error (MPE): -6.77%
Mean Absolute Percentage Error (MAPE): 21.21%

Random Forest Additional Statistics:
Out-of-Bag (OOB) Error: 0.3785

Gradient Boosting Additional Statistics:
Learning Curve - Train Errors: [602.5176744999999, 593.5896804023646, 584.839353387272, 5
76.2631578797798, 567.857628662887, 559.6193694774099, 551.5450516497241, 543.63141274680
91, 535.8752552580622, 528.2734453033414, 520.8229113667195, 513.5206430554364, 506.36368
988354786, 499.3491600797797, 492.4742194191067, 485.73609007758114, 479.1320495099518, 4
72.65942934961834, 466.31561433047546, 460.0980412302136, 454.00419783464696, 448.0316219
22652, 442.17790027130593, 436.44066768082155, 430.81760601888766, 425.3064432840264, 41
9.90495268758895, 414.6109517540205, 409.4223014390301, 404.3369052653079, 399.3527084754
429, 394.46769720169624, 389.67989765229703, 384.98737531393095, 380.38823417009826, 375.
8806159350279, 371.46269930283546, 367.1326992116236, 362.8888661222269, 358.729485311309
13, 354.6528761785287, 350.6573915674906, 346.7414171002121, 342.8974481149634, 338.56330
3262721, 334.3154078930382, 330.6830281336423, 326.58520347630355, 322.5689255296458, 31
9.1360186755885, 315.2613670043186, 311.463820901307, 308.2189743869377, 304.555090983658
2, 300.96411886010407, 297.4446070818086, 294.4317630424823, 291.0358932017795, 287.70760
117090674, 284.85874112385875, 281.6471488798278, 278.4994673214532, 275.80524025905123,
272.7677110309575, 269.7906286345029, 267.2422258869935, 264.3691204164917, 261.553189744
8527, 259.14231333997526, 256.42453490729224, 253.76084026541963, 251.47967313767768, 24
8.90863666146873, 246.38876381113636, 244.22994175252342, 241.79754466454014, 239.4135522
7860777, 237.37013752891548, 235.06873232094313, 232.8131250766095, 230.87858145740665, 2
28.70094980387694, 226.56665302025255, 224.64811355782822, 222.58769597193776, 220.568280
6960066, 218.7480529957871, 216.79740500946065, 215.04142583465463, 213.15707761537297, 2
11.31022792565494, 209.64257116322443, 207.63123686185222, 205.65992811307729, 203.944871
38823828, 202.04634082438102, 200.18559101874445, 198.3618701342401, 196.57444129533738,
194.82258229032882, 193.10558527951986, 191.61253079283082, 190.25089856243804, 188.62264
080771132, 187.02678538230367, 185.64014154068832, 184.37450406441866, 182.8029632467968
8, 181.26269609144575, 179.9190137790307, 178.43568937528386, 176.98188312717156, 175.557
00762339671, 174.16048714214693, 172.7917574184741, 171.45026541630236, 170.2791807511720
8, 169.2540321810393, 167.98184115263246, 166.73496672569092, 165.6469007512894, 164.4458
308294549, 163.2686621990649, 162.11491922441968, 160.98413573496987, 159.87585483696012,
158.9083977994606, 158.0608529634801, 157.04690494531872, 156.01593701181213, 155.1161805
835638, 154.32946083612185, 153.35162975976914, 152.3932575218358, 151.55712287013552, 15
0.63393319679352, 149.72911499795106, 148.84230268126552, 147.93212371465452, 147.1715669
225582, 146.2917401660947, 145.46653161483368, 144.6183403558278, 143.9106925126208, 143.
09073999031338, 142.28710452319987, 141.6156342898104, 140.83873457056657, 140.1339027297
3257, 139.38434673277513, 138.75888621486575, 138.03422795367314, 137.431013375641, 136.7
3039616508444, 136.04372123701805, 135.47122380396166, 134.80723847979453, 134.2037519553
7397, 133.56250479602323, 133.0482192913906, 132.42719267234344, 131.8639784522224, 131.2
641687972416, 130.67629535439494, 130.20311765940303, 129.63375248860748, 129.17667918471
3, 128.62521328751683, 128.12450962226035, 127.59182994266389, 127.16501235102726, 126.64
903986847489, 126.14333523832536, 125.7368997030912, 125.24704400712373, 124.801682373137
17, 124.3284652193, 123.94882626037084, 123.49040502431299, 123.12360408672143, 122.67949
466746325, 122.24422302564831, 121.93077679633775, 121.50828948150256, 121.2051304278121
5, 120.79502869067987, 120.46709480870227, 120.06975324358805, 119.78598506848763, 119.40
02548375624, 119.03490072858581, 118.76443065803039, 118.39671762034268, 118.135077207069
53, 117.77806826871517, 117.52496150448903, 117.19135942948105, 116.91670776660828, 116.5
8011528542268, 116.25022099461266]
Learning Curve - Validation Errors: [815.4540733034722, 803.2233951121406, 791.2339412242
88, 779.4809329382002, 767.9596864317089, 756.6656108762523, 745.5942065884494, 734.74106
32184332, 724.1018579742147, 713.6723538813635, 703.4483980772975, 693.4259201394963, 68
3.6009304469633, 673.9695185742748, 664.5278517175667, 655.2721731518229, 646.19880071884
9, 637.3041253453092, 628.5846095902409, 620.0367862214509, 611.6572568202248, 603.442690
4137832, 595.3898221349322, 587.495451908372, 579.756443163127, 572.1697215705804, 564.73
2273807607, 557.4411463442999, 550.2934442558031, 543.2863300577759, 536.417022565008, 52
9.6827957727381, 523.0809777602122, 516.6089496160475, 510.2641443849649, 504.04404603546
357, 497.94618844802505, 491.9681544234357, 486.1075747108259, 480.36212705503857, 474.72
953526293725, 469.2075682882813, 463.79403933479597, 459.03796810712794, 454.192687064269
5, 449.4583748363975, 444.9956031850662, 440.4624884906868, 436.0338471372227, 431.847194
9856065, 427.6079528222332, 423.467058029367, 419.540349921038, 415.5777373045744, 411.70
```

```
769457711276, 407.9282424588826, 404.3234652391339, 400.7083507436495, 397.1784891573076,
393.79955454395486, 390.4243593656057, 387.1293831750009, 383.96305625806656, 380.8136338
731865, 377.7396830751378, 374.7735024107613, 371.83651918883254, 368.9705304018911, 366.
1927627109368, 363.4556512704019, 360.7853120206283, 358.18490954343406, 355.635824796012
86, 353.14953043772783, 350.71609078922154, 348.34386851701396, 346.0306816359023, 343.75
44100144504, 341.54852783315886, 339.39813998797996, 337.26981359992794, 335.22035417006
2, 333.22304983757857, 331.39418721987903, 329.48930699625663, 327.6334716055565, 325.893
65882040926, 324.12509724056633, 322.46442703494506, 320.779769743717, 319.1394527546493
7, 317.59133440143563, 316.2599572976127, 314.9712509344142, 313.49257459862963, 312.2856
978680401, 311.11858393533583, 309.9902840613296, 308.899869961339, 307.8464338239573, 3
06.82908568902644, 305.61198510972804, 304.42842133803526, 303.5093514143708, 302.6232382
9449895, 301.53349621009625, 300.46250778357, 299.734847602034, 299.03514127824496, 298.1
2669319630703, 297.4810673584458, 296.86140716681916, 296.2670647281465, 295.697406353992
73, 295.1518122649739 6, 294.629676300981, 293.8879070384465, 293.07291708320014, 292.6169
3394468523, 292.182252024354 6, 291.52773123738257, 291.13481710531335, 290.7616261502259,
290.4076468390407, 290.0723790086052, 289.75533362752776, 289.21933999954655, 288.5903332
52008, 288.2686748829274, 288.0208374087342, 287.5550541889462, 286.99115826841046, 286.7
9085616886744, 286.60541975949167, 286.2034247312078, 286.04758417957004, 285.90543739673
99, 285.77660595129976, 285.7461699109772, 285.4160379365505, 285.41298339724335, 285.325
7774785258, 285.3478792278552, 285.06959441494064, 285.1164947902494, 285.17540953432456,
284.9329217723656, 285.0147906100988, 284.96719392349866, 285.07012820488455, 284.8720186
236062, 284.99566496834086, 284.8179704692564, 284.9612895677708, 285.113850547710 3, 284.
9652525207347 4, 285.2684781341407, 285.3043893981035, 285.61876312529773, 285.45891770651
43, 285.7849490809457, 285.849446496081, 286.1852062491285, 286.52517538776567, 286.40250
254501063, 286.7524607705087, 286.6439550791102, 287.003295081684, 287.10999069355773, 28
7.47683777001913, 287.394868585129, 287.7699443456419, 288.14776903054207, 288.0829880742
2733, 288.46823032162877, 288.60964822643723, 289.0004907270212, 288.95825840415375, 289.
35552441147354, 289.32371301823457, 289.7269401410368, 290.13171950487657, 289.9583078034
0714, 290.3694688991508, 290.2077399776789, 290.624850926916, 290.62726778107066, 291.048
83217111706, 290.90920370343366, 291.3359068681746, 291.639097936047, 291.51635422814286,
291.94904708019396, 291.8359359972502, 292.2728040269813, 292.16891915314045, 292.4944044
684464, 292.5458479980315, 292.9897974611398, 293.4334861162174]
```

In [ ]:

```python
#dataframe with additional statistics (without gradient boosting)
additional_stats_df = pd.DataFrame({
    'Metric': ['Mean Absolute Error (MAE)',
               'Root Mean Squared Error (RMSE)',
               'Adjusted R-squared',
               'Mean Percentage Error (MPE)',
               'Mean Absolute Percentage Error (MAPE)',
               'Out-of-Bag (OOB) Error'],
    'Linear Regression': [mae_lr, rmse_lr, adjusted_r2_lr, mpe_lr, mape_lr, '-'],
    'Random Forest': ['-', '-', '-', '-', '-', oob_error_rf]
})

additional_stats_df
```

Out[ ]:

	Metric	Linear Regression	Random Forest
0	Mean Absolute Error (MAE)	11.031431	-
1	Root Mean Squared Error (RMSE)	16.596053	-
2	Adjusted R-squared	0.567745	-
3	Mean Percentage Error (MPE)	-6.767803	-
4	Mean Absolute Percentage Error (MAPE)	21.213804	-
5	Out-of-Bag (OOB) Error	-	0.378481

In [ ]:

```
#using statsmodel to display linreg coefficients, and p-values
X = sm.add_constant(X)

#fitting the model
model = sm.OLS(y, X).fit()

#getting values
coefficients_lr = model.params
p_values = model.pvalues

#dataframe with values
coeff_pval_df = pd.DataFrame({
    'Coefficient': coefficients_lr,
    'P-value': p_values
})

print('Linear Regression:')
coeff_pval_df
```

```
Linear Regression:
```

Out[ ]:

	Coefficient	P-value
const	-17.579095	1.231882e-02
speed	3.932409	1.489836e-12