Ensemble Methods

Zander Bonnet

September 25, 2024

In [59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')
import seaborn as sns
from sklearn.ensemble import GradientBoostingRegressor, BaggingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from catboost import CatBoostRegressor
from sklearn.tree import DecisionTreeRegressor
```

In [60]:
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
bike_sharing = fetch_ucirepo(id=275)

# data (as pandas dataframes)
X = bike_sharing.data.features
y = bike_sharing.data.targets

# metadata
print(bike_sharing.metadata)

# variable information
print(bike_sharing.variables)
```

{'uci_id': 275, 'name': 'Bike Sharing', 'repository_url': 'https://archive.ics.uci.edu/dataset/275/bike+sharing+dataset', 'data_url': 'https://archive.ics.uci.edu/static/public/275/data.csv', 'abstract': 'This dataset contains the hourly and daily count of rental bikes between years 2011 and 2012 in Capital bikeshare system with the corresponding weather and seasonal information.', 'area': 'Social Science', 'tasks': ['Regression'], 'characteristics': ['Multivariate'], 'num_instances': 17389, 'num_features': 13, 'feature_types': ['Integer', 'Real'], 'demographics': [], 'target_col': ['cnt'], 'index_col': ['instant'], 'has_missing_values': 'no', 'missing_values_symbol': None, 'year_of_dataset_creation': 2013, 'last_updated': 'Sun Mar 10 2024', 'dataset_doi': '10.24432/C5W894', 'creators': ['Hadi Fanaee-T'], 'intro_paper': {'ID': 422, 'type': 'NATIVE', 'title': 'Event labeling combining ensemble detectors and background knowledge', 'authors': 'Hadi Fanaee-T, João Gama', 'venue': 'Progress in Artificial Intelligence', 'year': 2013, 'journal': None, 'DOI': '10.1007/s13748-013-0040-3', 'URL': 'https://www.semanticscholar.org/paper/bc42899f599d31a5d759f3e0a3ea8b52479d6423', 'sha': None, 'corpus': None, 'arxiv': None, 'mag': None, 'acl': None, 'pmid': None, 'pmcid': None}, 'additional_info': {'summary': 'Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. Through these systems, user is able to easily rent a bike from a particular position and return back at another position. Currently, there are about over 500 bike-sharing programs around the world which is composed of over 500 thousands bicycles. Today, there exists great interest in these systems due to their important role in traffic, environmental and health issues. \r\n\r\nApart from interesting real world applications of bike sharing systems, the characteristics of data being generated by these systems make them attractive for the research. Opposed to other transport services such as bus or subway, the duration of travel, departure and arrival position is explicitly recorded in these systems. This feature turns bike sharing system into a virtual sensor network that can be used for sensing mobility in the city. Hence, it is expected that most of important events in the city could be detected via monitoring these data.', 'purpose': None, 'funded_by': None, 'instances_represent': None, 'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'Both hour.csv and day.csv have the following fields, except hr which is not available in day.csv\r\n\t\r\n\t- instant: record index\r\n\t- dteday : date\r\n\t- season : season (1:winter, 2:spring, 3:summer, 4:fall)\r\n\t- yr : year (0: 2011, 1:2012)\r\n\t- mnth : month (1 to 12)\r\n\t- hr : hour (0 to 23)\r\n\t- holiday : weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)\r\n\t- weekday : day of the week\r\n\t- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.\r\n\t+ weathersit : \r\n\t\t- 1: Clear, Few clouds, Partly cloudy, Partly cloudy\r\n\t\t- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist\r\n\t\t- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds\r\n\t\t- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog\r\n\t- temp : Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)\r\n\t- atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-16, t_max=+50 (only in hourly scale)\r\n\t- hum: Normalized humidity. The values are divided to 100 (max)\r\n\t- windspeed: Normalized wind speed. The values are divided to 67 (max)\r\n\t- casual: count of casual users\r\n\t- registered: count of registered users\r\n\t- cnt: count of total rental bikes including both casual and registered\r\n', 'citation': None}}

```
         name     role         type demographic  \
0      instant       ID      Integer        None
```

```
1       dteday  Feature        Date        None
2       season  Feature  Categorical       None
3           yr  Feature  Categorical       None
4         mnth  Feature  Categorical       None
5           hr  Feature  Categorical       None
6      holiday  Feature       Binary       None
7      weekday  Feature  Categorical       None
8   workingday  Feature       Binary       None
9   weathersit  Feature  Categorical       None
10        temp  Feature   Continuous       None
11       atemp  Feature   Continuous       None
12         hum  Feature   Continuous       None
13   windspeed  Feature   Continuous       None
14      casual    Other      Integer       None
15  registered    Other      Integer       None
16         cnt   Target      Integer       None

                                          description units missing_values
0                                        record index  None             no
1                                                date  None             no
2                1:winter, 2:spring, 3:summer, 4:fall  None             no
3                             year (0: 2011, 1: 2012)  None             no
4                                     month (1 to 12)  None             no
5                                     hour (0 to 23)  None             no
6   weather day is holiday or not (extracted from ...  None             no
7                                     day of the week  None             no
8   if day is neither weekend nor holiday is 1, ot...  None             no
9   - 1: Clear, Few clouds, Partly cloudy, Partly ...  None             no
10  Normalized temperature in Celsius. The values ...     C             no
11  Normalized feeling temperature in Celsius. The...     C             no
12  Normalized humidity. The values are divided to...  None             no
13  Normalized wind speed. The values are divided ...  None             no
14                                 count of casual users  None             no
15                             count of registered users  None             no
16  count of total rental bikes including both cas...  None             no
```

This data set is a collection of data about the amount of rental bikes being used. The dataset contains data such as the time of day, the type of weather, the day of the week, and so on. The description of the complete data set is printed above. With this data, I hope to accurately predict the number of rental bikes being used at a given time.

I will utilize various ensemble methods to find the most effective approach to accomplish this.

How accurately can I predict the number of rentals at a given time based on the factors in the dataset?

What ensemble method will prove to be the most accurate and efficient?

In [61]:
```
#drop unwanted column
X = X.drop('dteday', axis = 1)
X.dtypes
```

Out[61]:
```
season          int64
yr              int64
mnth            int64
hr              int64
holiday         int64
weekday         int64
workingday      int64
weathersit      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
dtype: object
```

In [62]:
```
#cast the categorical variables to categorical data type
cat = ['season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'we
for col in cat:
    X[col] = X[col].astype('category',copy=False)
X.dtypes
```

Out[62]:
```
season        category
yr            category
mnth          category
hr            category
holiday       category
weekday       category
workingday    category
weathersit    category
temp           float64
atemp          float64
hum            float64
windspeed      float64
dtype: object
```

In [63]:
```
X.head()
```

Out[63]:

	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	h
0	1	0	1	0	0	6	0	1	0.24	0.2879	C
1	1	0	1	1	0	6	0	1	0.22	0.2727	0
2	1	0	1	2	0	6	0	1	0.22	0.2727	0
3	1	0	1	3	0	6	0	1	0.24	0.2879	0
4	1	0	1	4	0	6	0	1	0.24	0.2879	0

In [64]:
```
X.describe()
```

Out[64]:

	temp	atemp	hum	windspeed
count	17379.000000	17379.000000	17379.000000	17379.000000
mean	0.496987	0.475775	0.627229	0.190098
std	0.192556	0.171850	0.192930	0.122340
min	0.020000	0.000000	0.000000	0.000000
25%	0.340000	0.333300	0.480000	0.104500
50%	0.500000	0.484800	0.630000	0.194000
75%	0.660000	0.621200	0.780000	0.253700
max	1.000000	1.000000	1.000000	0.850700

In [65]:

```
y.head()
```

Out[65]:

	cnt
0	16
1	40
2	32
3	13
4	1

In [66]:

```
y.describe()
```

Out[66]:

	cnt
count	17379.000000
mean	189.463088
std	181.387599
min	1.000000
25%	40.000000
50%	142.000000
75%	281.000000
max	977.000000

In [67]:

```
#split the data to a train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30)
```

Base Decision Tree

This will serve as a baseline as many of these ensemble models utilize decision trees under the hood.

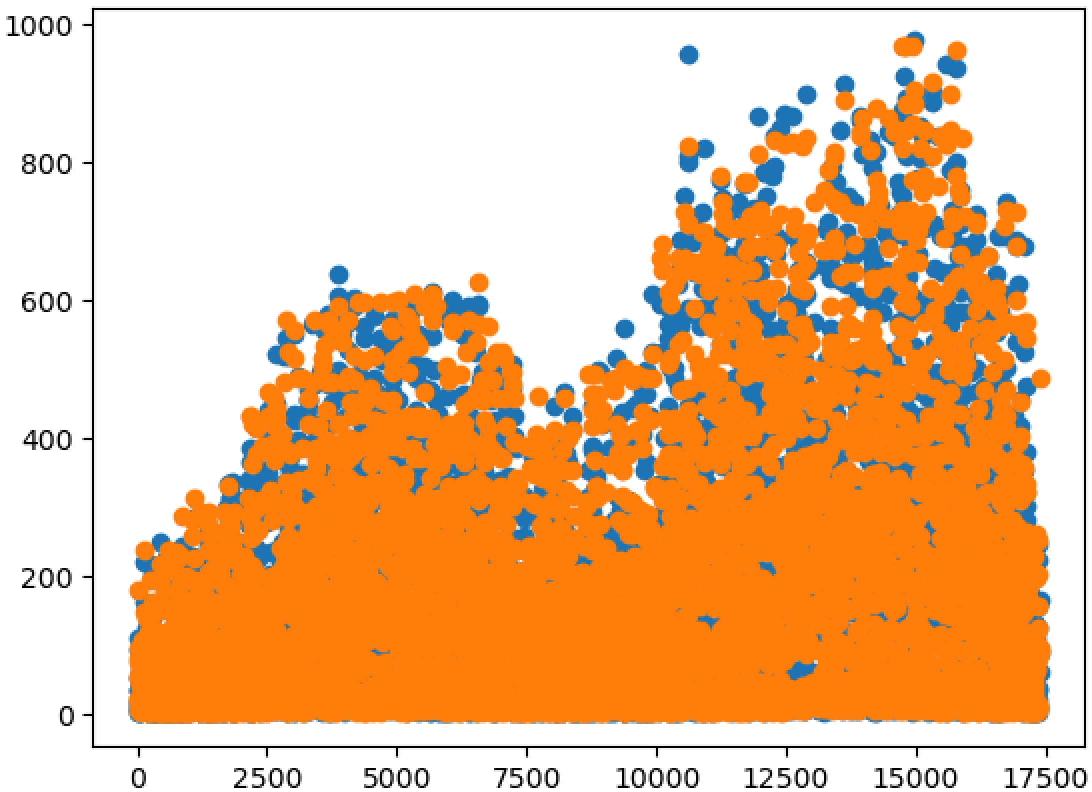
In [68]:
```
mod = DecisionTreeRegressor()
mod.fit(X_train, np.ravel(y_train))
pred = mod.predict(X_test)
print(np.sqrt(mean_squared_error(y_test, pred)))
```

```
57.900987948798196
```

In [69]:
```
mod.score(X_test,y_test)
```

Out[69]:
```
0.897945498484886
```

In [70]:
```
plt.scatter(y_test.index,np.ravel(y_test))
plt.scatter(X_test.index, pred)
plt.show()
```



This baseline model performs fairly well with an RMSE of 57.86 and an r_2 score of .897. This means the model is about 89% accurate and on average is 57.8 rentals off.

Bagging

Bagging works by bootstrapping the data into smaller subsets and running models on these individual subsets. These models all run at the same time. After the completion of the smaller models, the bagging algorithm averages the responses to create the final output.

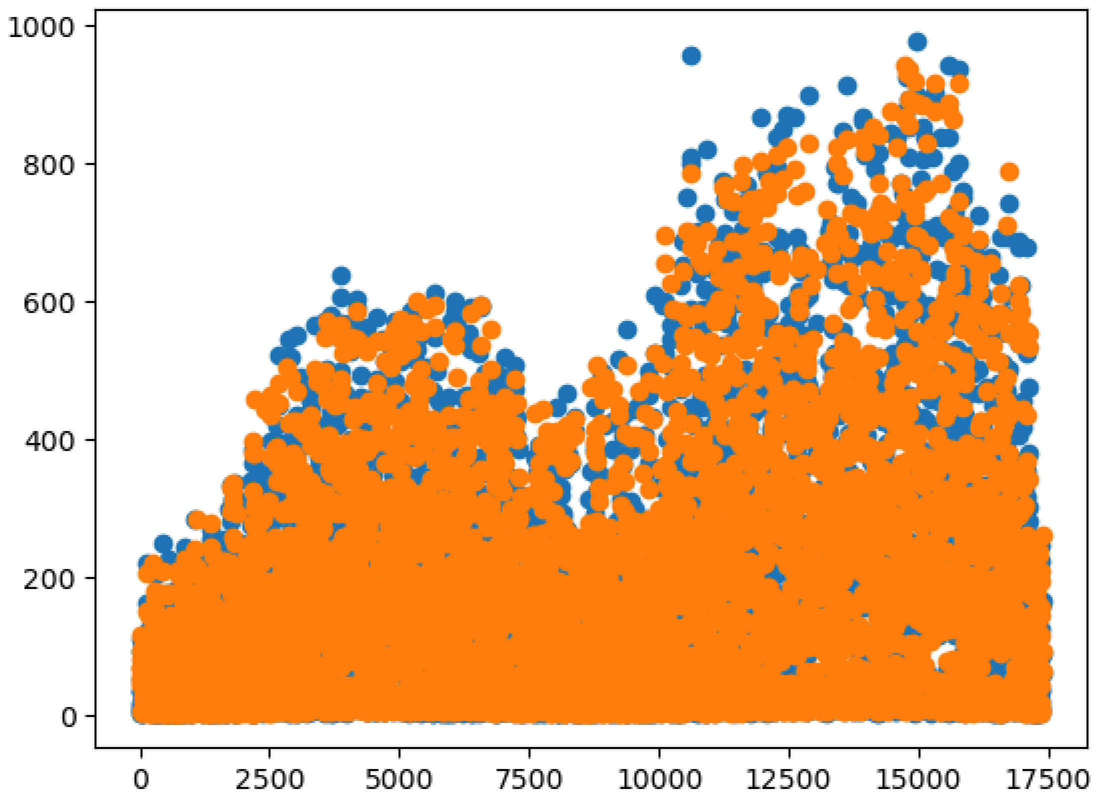
In [71]:
```
model0 = BaggingRegressor(DecisionTreeRegressor())
model0.fit(X_train, np.ravel(y_train))
pred_final0 = model0.predict(X_test)
print(np.sqrt(mean_squared_error(y_test, pred_final0)))
```

```
44.988698022038534
```

In [72]:
```
model0.score(X_test,y_test)
```

Out[72]:
```
0.9383877502359166
```

In [73]:
```
plt.scatter(y_test.index,np.ravel(y_test))
plt.scatter(X_test.index, pred_final0)
plt.show()
```



Comparing the bagging algorithm to the baseline we can see that there was an overall improvement in the ability to predict the data. The r2_score increased to .936 and the RMSE is now as low as 45. The bagging algorithm was able to reduce the variation in the data by averaging over multiple iterations of models. This will mean that models that are heavily impacted by more extreme data points will be outweighed by the more accurate models.

Gradient Boosting

GBM is a boosting algorithm, meaning it sequentially creates models that build upon each other. To accomplish this the model creates a base model and then calculates the residuals to the target. There is then a subsequent model created to minimize the residuals. Once a prediction is made it is added to the original prediction and the residuals are recalculated. This continues for the given number of iterations.

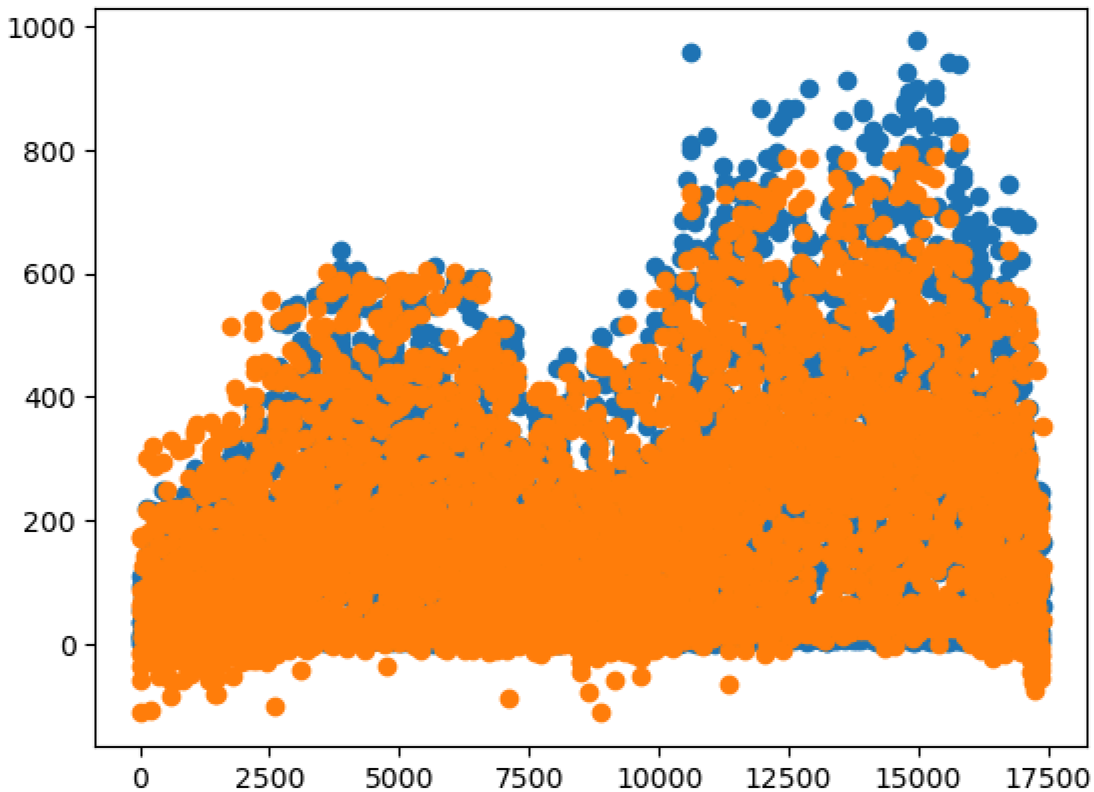
In [74]:
```
model = GradientBoostingRegressor(n_estimators=500)
model.fit(X_train, np.ravel(y_train))
pred_final = model.predict(X_test)
print(np.sqrt(mean_squared_error(y_test, pred_final)))
```

```
51.79883825916279
```

In [75]:
```
model.score(X_test,y_test)
```

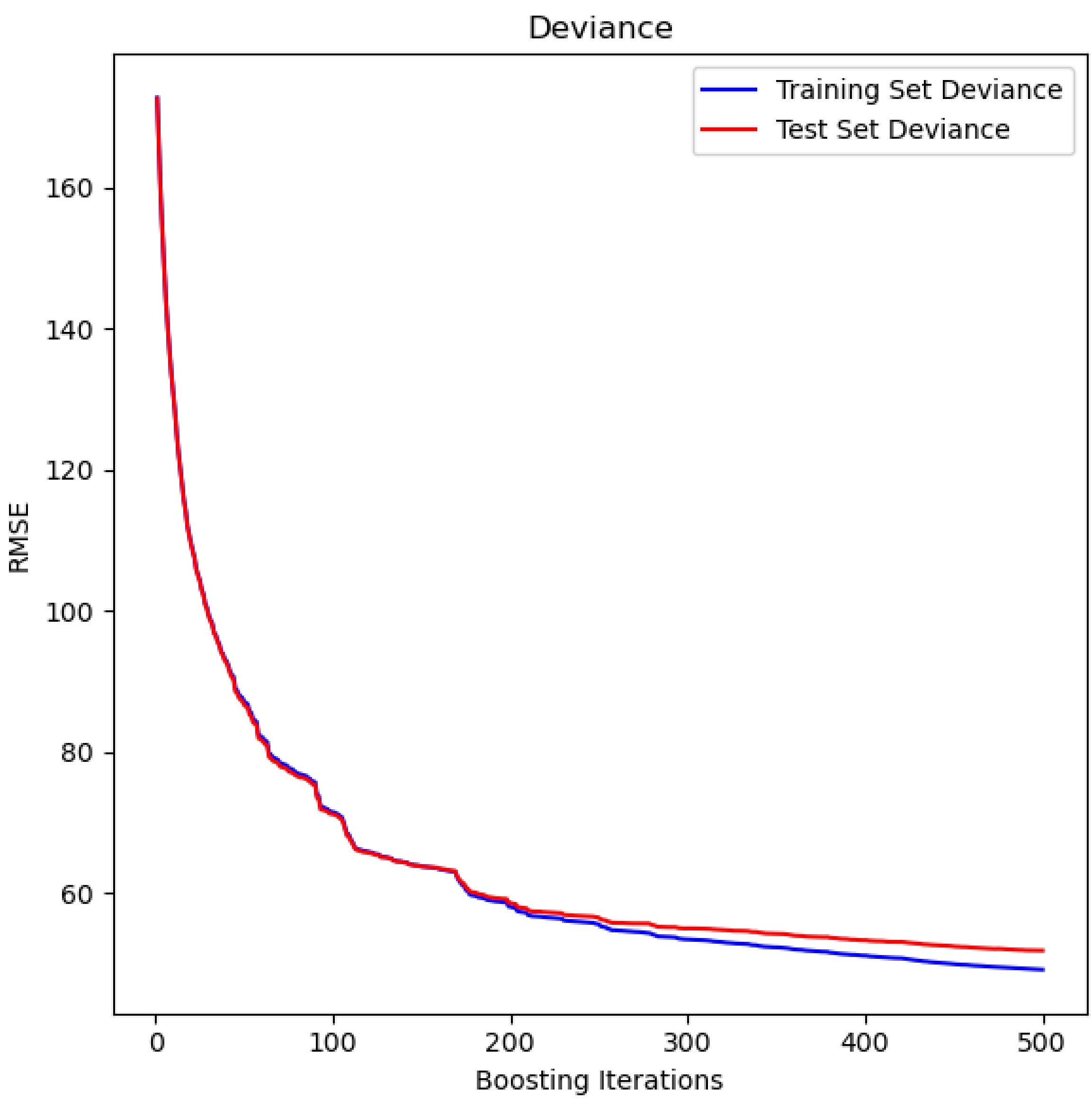
Out[75]:
```
0.9183229099150216
```

In [76]:
```
plt.scatter(y_test.index,np.ravel(y_test))
plt.scatter(X_test.index, pred_final)
plt.show()
```



In [77]:
```
test_score = np.zeros(500, dtype=np.float64)
for i, y_pred in enumerate(model.staged_predict(X_test)):
    test_score[i] = np.sqrt(mean_squared_error(y_test, y_pred))

fig = plt.figure(figsize=(6, 6))
plt.subplot(1, 1, 1)
plt.title("Deviance")
plt.plot(
    np.arange(500) + 1,
    np.sqrt(model.train_score_),
    "b-",
    label="Training Set Deviance",
```

```
)
plt.plot(np.arange(500) + 1, test_score, "r-", label="Test Set Deviance")
plt.legend(loc="upper right")
plt.xlabel("Boosting Iterations")
plt.ylabel("RMSE")
fig.tight_layout()
```



This model performs better than the original model, but not as well as the bagging model. With an r_2 score of .918 and an RMSE of 51.8, the model is just slightly worse. Looking at the Deviance we can see that as the iterations pass the RMSE begins to flatten and stabilize. This is a sign that adding more iterations will begin to become less and less beneficial. This model can minimize the variation in predictions by focusing on reducing the residuals.

XGBoost

XGBoost is a direct improvement of the previous method. It can do this by introducing regularization to the data, tree pruning, and built-in cross-validation. These additions make it so the algorithm doesn't spend as much time on tree nodes that do not improve the performance of the model.

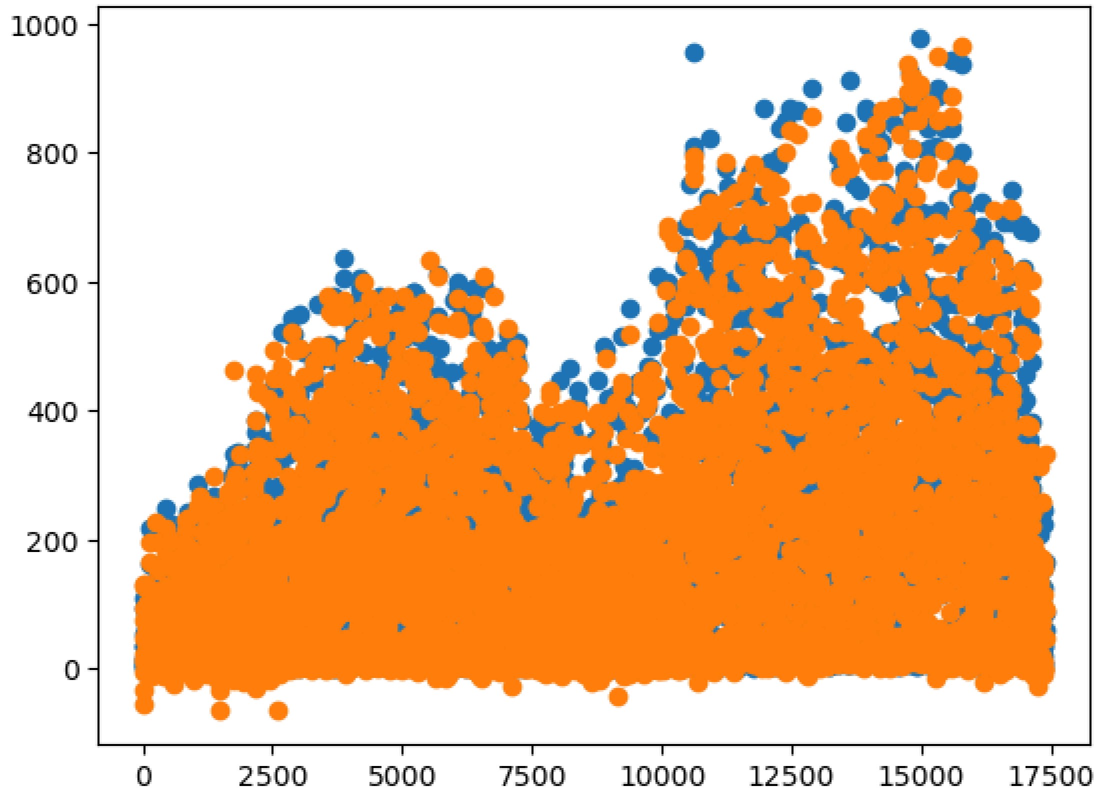
In [78]:
```
model2=xgb.XGBRegressor(enable_categorical = True)
model2.fit(X_train, y_train, eval_set = [(X_test, y_test)], verbose=10)
pred_final2 = model2.predict(X_test)
print(np.sqrt(mean_squared_error(y_test, pred_final2)))
```

```
[0]	validation_0-rmse:138.60938
[10]	validation_0-rmse:48.91669
[20]	validation_0-rmse:43.90393
[30]	validation_0-rmse:42.72042
[40]	validation_0-rmse:42.12085
[50]	validation_0-rmse:42.02758
[60]	validation_0-rmse:41.74164
[70]	validation_0-rmse:41.62703
[80]	validation_0-rmse:41.59106
[90]	validation_0-rmse:41.44882
[99]	validation_0-rmse:41.44781
41.44781393322742
```

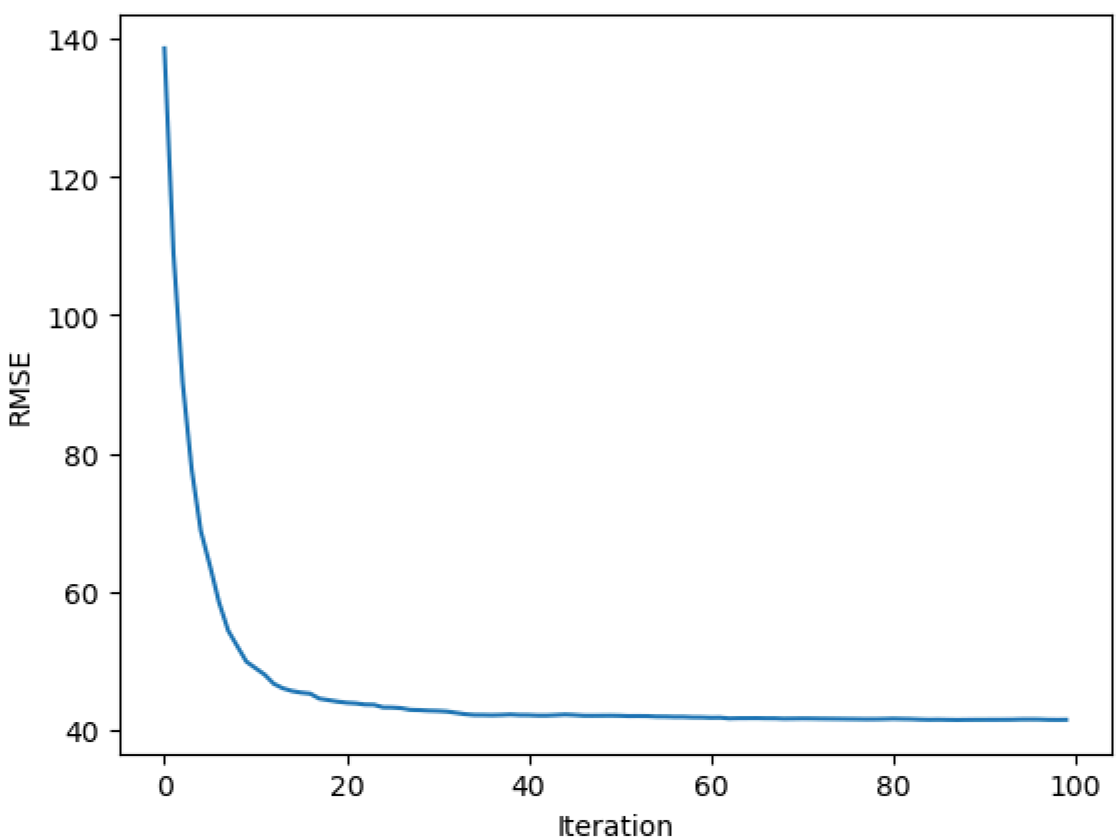
In [79]:
```
model2.score(X_test,y_test)
```

Out[79]:
```
0.9477046132087708
```

In [80]:
```
plt.scatter(y_test.index,np.ravel(y_test))
plt.scatter(X_test.index, pred_final2)
plt.show()
```



In [81]:
```
plt.plot(range(100),model2.evals_result_['validation_0']['rmse'])
plt.ylabel('RMSE')
plt.xlabel('Iteration')
plt.show()
```



This model performs the best so far with an RMSE of 41.3 and a r2_score of .947. This shows that the model performs very well. Looking at the RMSE Vs. Iterations plot we can also see that the model converged very early. There was very minimal improvement in RMSE after about 25 iterations.

CatBoost

CatBoost is another boosting algorithm, but this one excels in datasets with large amounts of categorical features. It can do this by converting categorical factors to numerical values using a variety of statistical methods.

In [82]:

```
model3=CatBoostRegressor(verbose = 50)
model3.fit(X_train, y_train, cat_features=np.where(X.dtypes == 'category')[0
pred_final3 = model3.predict(X_test)
print(np.sqrt(mean_squared_error(y_test, pred_final3)))
```

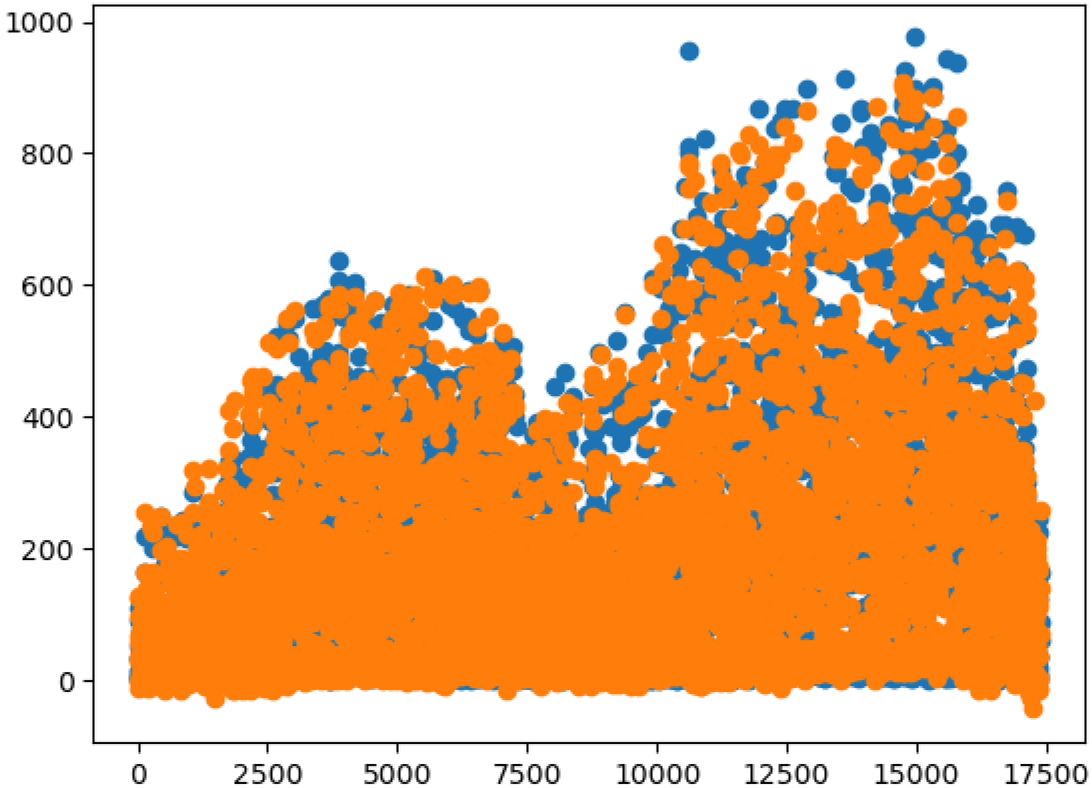
```
Learning rate set to 0.07538
0:	learn: 172.9089452	test: 172.4738129	best: 172.4738129
(0)	total: 5.11ms	remaining: 5.1s
50:	learn: 70.2833412	test: 61.6242359	best: 61.6242359 (5
0)	total: 186ms	remaining: 3.46s
100:	learn: 61.2717337	test: 53.1365734	best: 53.1365734 (10
0)	total: 337ms	remaining: 3s
150:	learn: 57.3629442	test: 50.3866016	best: 50.3866016 (15
0)	total: 521ms	remaining: 2.93s
200:	learn: 54.8220014	test: 48.9598002	best: 48.9598002 (20
0)	total: 669ms	remaining: 2.66s
250:	learn: 52.6467134	test: 47.7721806	best: 47.7721806 (25
0)	total: 834ms	remaining: 2.49s
300:	learn: 50.9642809	test: 46.9484068	best: 46.9484068 (30
0)	total: 988ms	remaining: 2.29s
350:	learn: 49.0840582	test: 46.0318651	best: 46.0318651 (35
0)	total: 1.13s	remaining: 2.08s
400:	learn: 47.8470091	test: 45.4162815	best: 45.4162815 (40
0)	total: 1.26s	remaining: 1.88s
450:	learn: 47.0441712	test: 45.0592163	best: 45.0592163 (45
0)	total: 1.38s	remaining: 1.68s
500:	learn: 46.0683988	test: 44.7082869	best: 44.7082869 (50
0)	total: 1.51s	remaining: 1.51s
550:	learn: 45.1930652	test: 44.2556508	best: 44.2556508 (55
0)	total: 1.66s	remaining: 1.35s
600:	learn: 44.5278151	test: 43.9904833	best: 43.9904833 (60
0)	total: 1.79s	remaining: 1.19s
650:	learn: 43.9830670	test: 43.7702176	best: 43.7702176 (65
0)	total: 1.91s	remaining: 1.02s
700:	learn: 43.3379794	test: 43.5027646	best: 43.5027646 (70
0)	total: 2.03s	remaining: 866ms
750:	learn: 42.7272985	test: 43.3653891	best: 43.3653891 (75
0)	total: 2.17s	remaining: 718ms
800:	learn: 42.0503314	test: 43.1342819	best: 43.1342819 (80
0)	total: 2.29s	remaining: 569ms
850:	learn: 41.2505193	test: 42.8686796	best: 42.8686796 (85
0)	total: 2.42s	remaining: 424ms
900:	learn: 40.6203609	test: 42.6511590	best: 42.6511590 (90
0)	total: 2.55s	remaining: 280ms
950:	learn: 40.1073898	test: 42.4619685	best: 42.4605777 (94
9)	total: 2.67s	remaining: 138ms
999:	learn: 39.6903829	test: 42.3803977	best: 42.3803977 (99
9)	total: 2.79s	remaining: 0us

bestTest = 42.38039774
bestIteration = 999

42.38039774008865
```

In [83]:
```
model3.score(X_test,y_test)
```

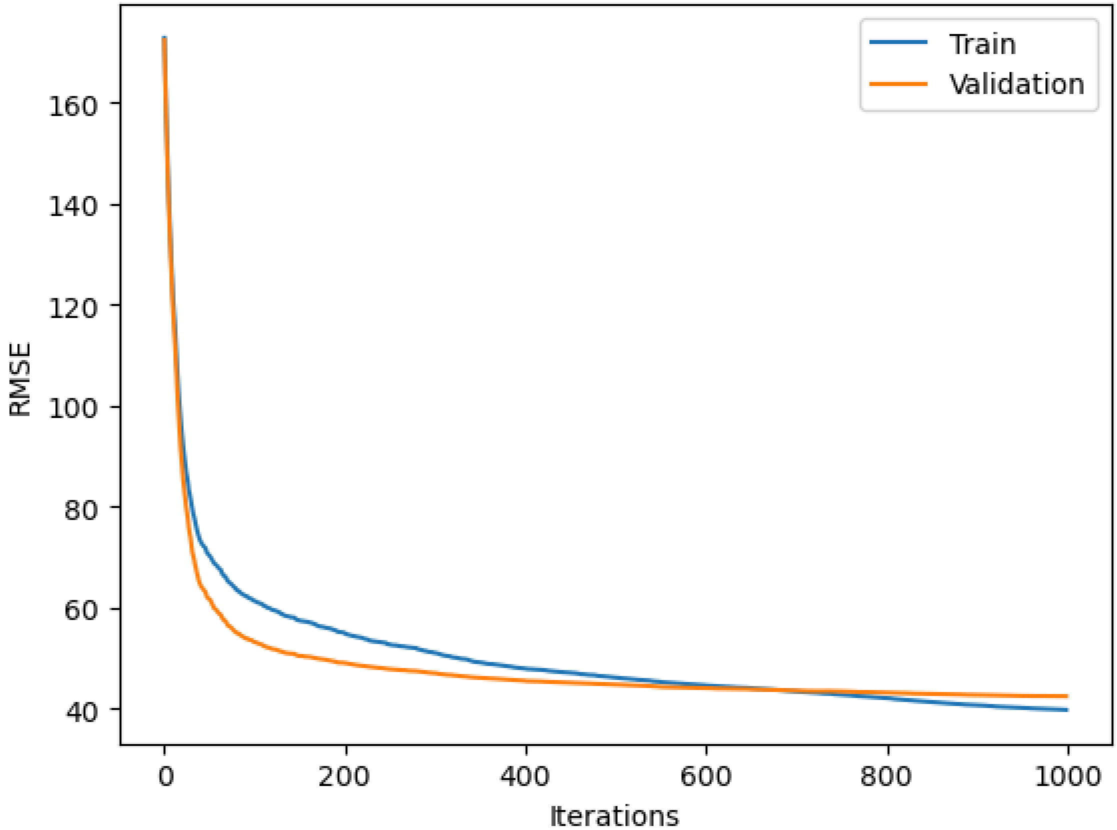
Out[83]:
```
0.9453248133685026
```

In [84]:
```
plt.scatter(y_test.index,np.ravel(y_test))
plt.scatter(X_test.index, pred_final3)
```

```
plt.show()
```



In [85]:
```
plt.plot(range(0,1000),model3.evals_result_['learn']['RMSE'], label = 'Trair
plt.plot(range(0,1000),model3.evals_result_['validation']['RMSE'], label = '
plt.ylabel('RMSE')
plt.xlabel('Iterations')
plt.legend()
plt.show()
```



This method performs very similarly to XGBoost. It does perform just slightly worse with an RMSE of 42.4 and a r_2 score of .945. This model does take much longer to execute as well. Looking at the RMSE Vs. Iterations plot we can also see that the validation set converged much quicker than the training data. This shows that the model was good at predicting external data at a small number of iterations.

Overall, the best model was XGBoost as it provided the most accurate predictions, and ran relatively fast compared to similar performers.

We were able to answer both of our questions. We found that XGBoost is the most accurate and efficient method to model this dataset, and it has an accuracy of about 94.7%.

Reference

Fanaee-T, H. (2013). Bike Sharing [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5W894.