Loading all necessary Packages/Libraries

In [1]:

```
# Loading the iconic trio 🔥
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing model_selection to get access to some dope functions like GridSearchCV()
from sklearn import model_selection

# from sklearn.externals import joblib

# Loading models
from sklearn import linear_model
from sklearn import svm
from sklearn import tree
from sklearn import ensemble
from sklearn import neighbors

# custom
import helper

# Loading black for formatting codes
%load_ext blackcellmagic
```

Styling Tables

In [2]:

```
%%HTML
<style type='text/css'>
table.dataframe th, table.dataframe td{
    border: 3px solid purple !important;
    color: solid black !important;
}
</style>
```

Loading the Dataset

In [3]:

```
# Loading dataset
filename = "Clean_Akosombo_data.csv"
akosombo = helper.load_csv_data(filename)
```

```
Successfully loaded!
```

Splitting the Dataset

In [4]:

```
# Splitting dataset
target_variable = "generation"
X, y, X_train, X_test, y_train, y_test = helper.split_data(akosombo, target_variable)
```

```
Data is splitted into X, y, X_train, X_test, y_train, y_test.

Shape Info of Features Training Set:
Number of datapoints (rows): 10001
Number of features (columns): 2

Shape Info of Features Test Set:
Number of datapoints (rows): 2501
Number of features (columns): 2
```

Scaling the Dataset

In [5]:

```
# Data Scaling
X_train, X_test = helper.scale(X_train, X_test)
```

Chosing Baseline Models and Training Models

In [6]:

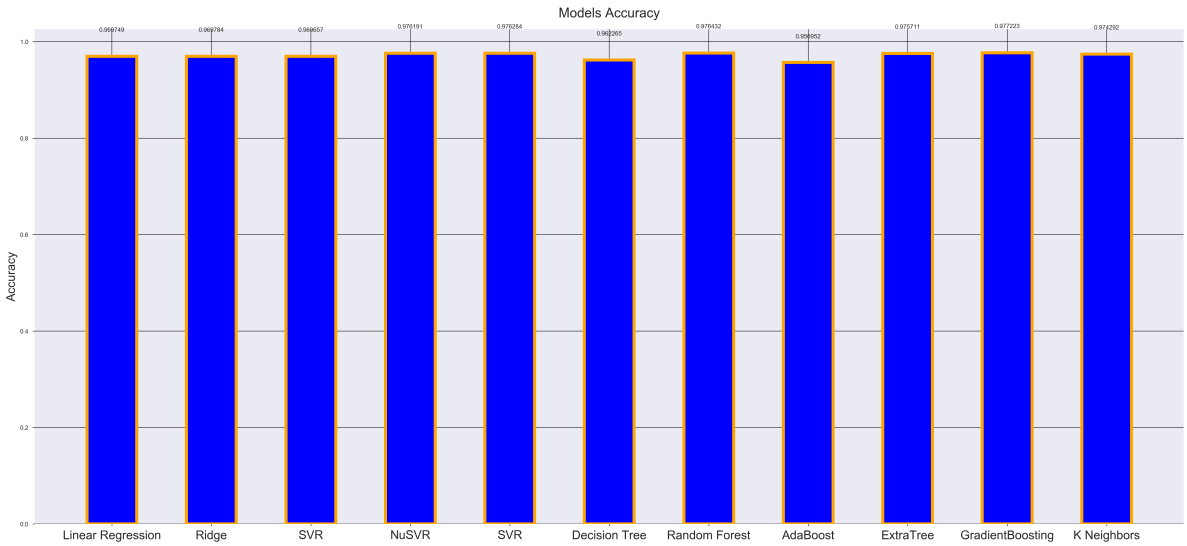
```
# Instantiating baseline models
models = [
    ("Linear Regression", linear_model.LinearRegression()),
#     ("Lasso", linear_model.Lasso()),
    ("Ridge", linear_model.Ridge()),
#     ("SDG", linear_model.SGDRegressor()),
    ("SVR", svm.LinearSVR()),
    ("NuSVR", svm.NuSVR()),
    ("SVR", svm.SVR()),
    ("Decision Tree", tree.DecisionTreeRegressor()),
    ("Random Forest", ensemble.RandomForestRegressor()),
    ("AdaBoost", ensemble.AdaBoostRegressor()),
    ("ExtraTree", ensemble.ExtraTreesRegressor()),
    ("GradientBoosting", ensemble.GradientBoostingRegressor()),
    ("K Neighbors", neighbors.KNeighborsRegressor()),
]

model_names = []
accuracies = []

# Fitting models to Training Dataset and Scoring them on Test set
for dataset_name, dataset in [("Akosomba_Data", akosombo)]:
    for model_name, model in models:
        regressor_model = model
        regressor_model.fit(X_train, y_train)

        accuracy = regressor_model.score(X_test, y_test)
        print(dataset_name, model_name, accuracy)

        model_names.append(model_name)
        accuracies.append(accuracy)
```

```
Akosomba_Data Linear Regression 0.9697494269312791
Akosomba_Data Ridge 0.9697837468881153
Akosomba_Data SVR 0.9696567718462318
Akosomba_Data NuSVR 0.9761908456257131
Akosomba_Data SVR 0.9762842898104876
Akosomba_Data Decision Tree 0.9622653656060981
Akosomba_Data Random Forest 0.9764322782401315
Akosomba_Data AdaBoost 0.9569520639911224
Akosomba_Data ExtraTree 0.9757105686833952
Akosomba_Data GradientBoosting 0.9772234044519424
Akosomba_Data K Neighbors 0.9742917147982414
```

Visualizing Models' Accuracy with Bar Charts

In [7]:

```
# Size in inches (width, height) & resolution(DPI)
plt.figure(figsize=(34, 15), dpi=200)

x_loc = np.arange(len(models))  # the x locations for the groups
width = 0.5  # bar width

# plotting the graphs with bar chart
models_graph = plt.bar(
    x_loc, accuracies, width, color="blue", edgecolor="orange", linewidth=5,
)

plt.title("Models Accuracy", fontsize=22, pad=20)
plt.xticks(x_loc, model_names, fontsize=20)
plt.ylabel("Accuracy", fontsize=20)
plt.grid(b=True, which="both", axis="both", color="black", linewidth=0.8)

# adding model accuracy on top of every bar
def addLabel(models):
    for model in models:
        height = model.get_height()
        plt.text(
            model.get_x() + model.get_width() / 2.0,
            1.05 * height,
            "%f" % height,
            ha="center",
            va="bottom",
        )


addLabel(models_graph)

plt.savefig('Bar_Charts_of_Models_and_their_Accuracy.png', dpi=300, transparent=True)

plt.show()
```

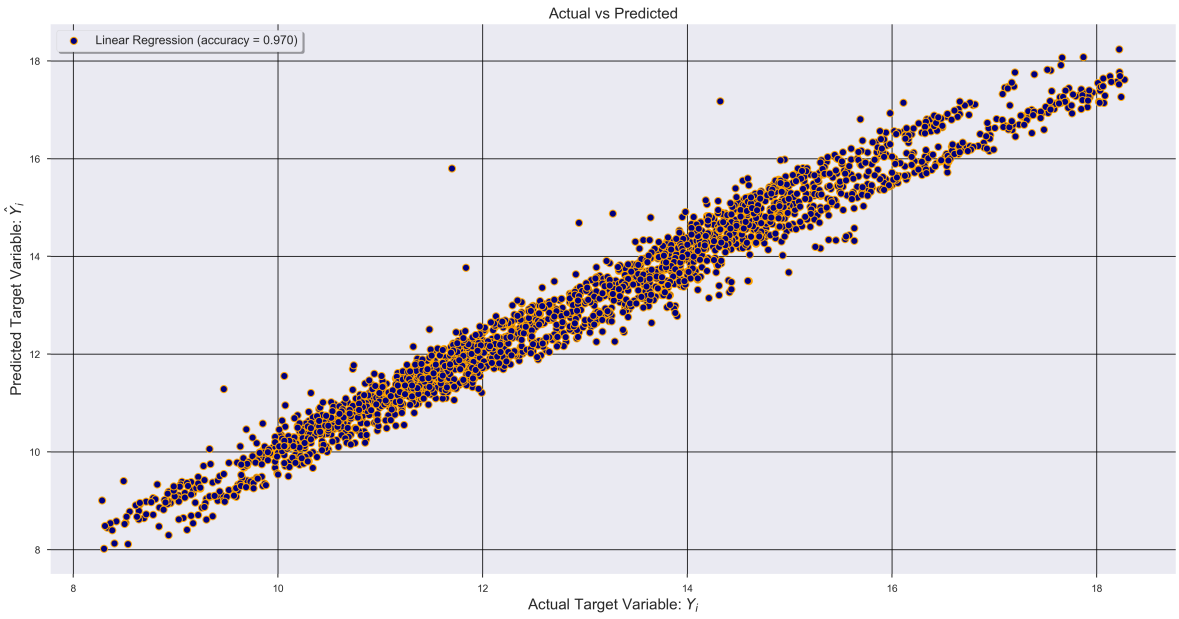


Evaluating Models

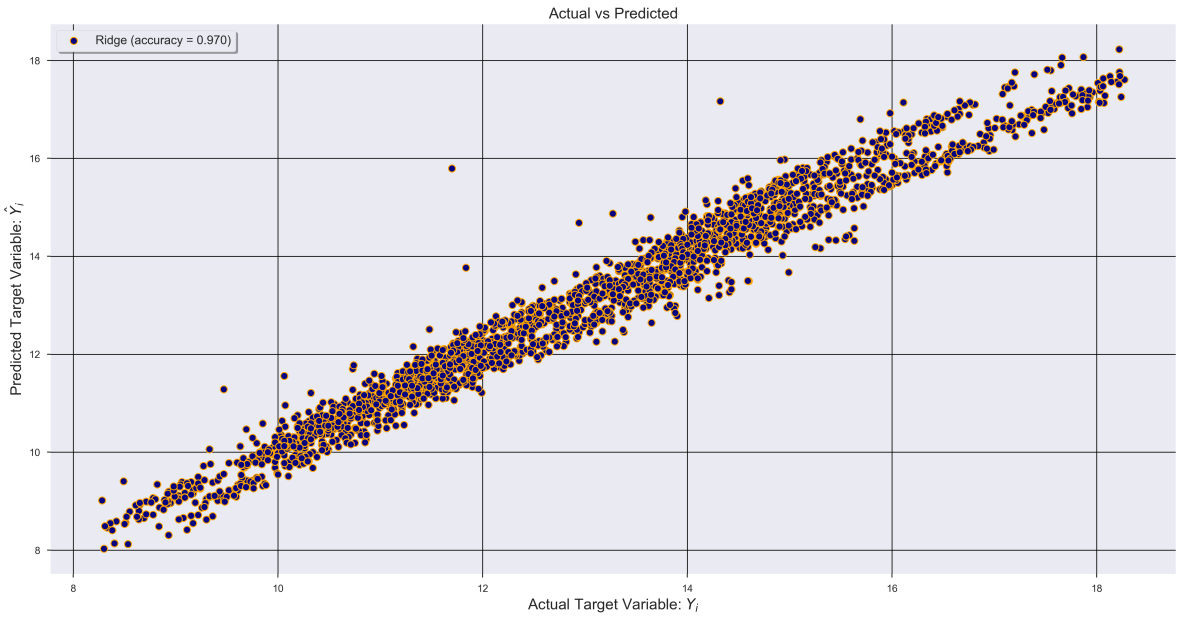
In [8]:

```
# Model Evaluation
for model_name, model in models:
    helper.evaluate(X_test, y_test, model_name, model)
```

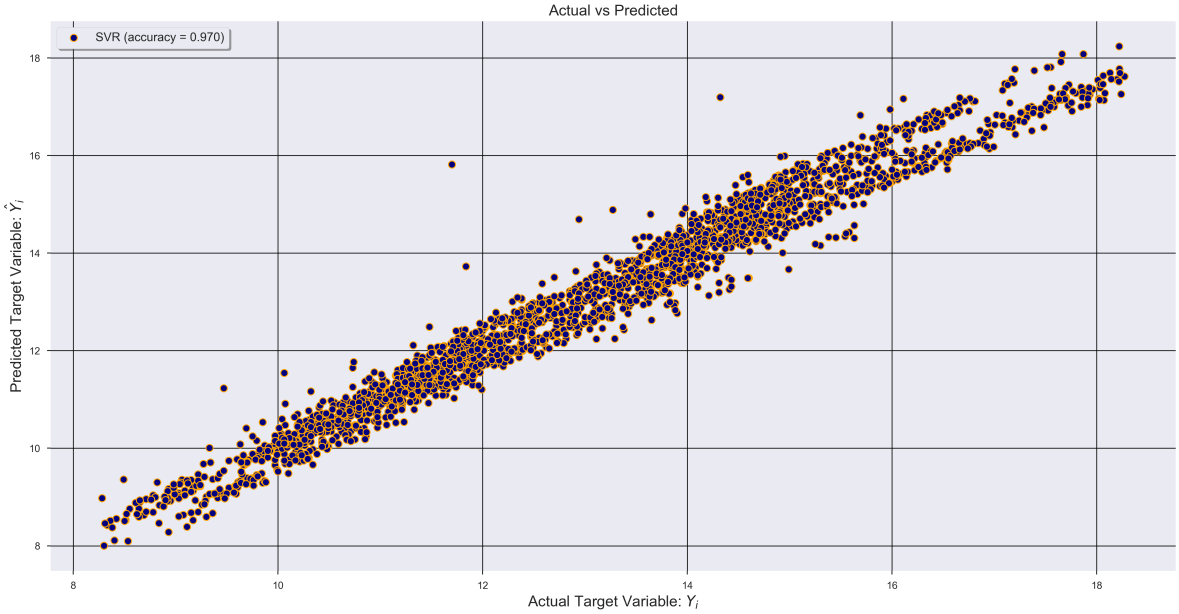
```
Linear Regression Mean Squared Error: 0.14625233556901102
Linear Regression Root Mean Squared Error: 0.38242951712571954
Linear Regression R2 Score: 0.969749426931279
Linear Regression Explained Variance Score: 0.969772234576705
Linear Regression Mean Absolute Error: 0.29263695465956613
Linear Regression Meadian Abosulute Error: 0.2677789934798991
Linear Regression Mean Squared Log Error: 0.0007141004739016008
```



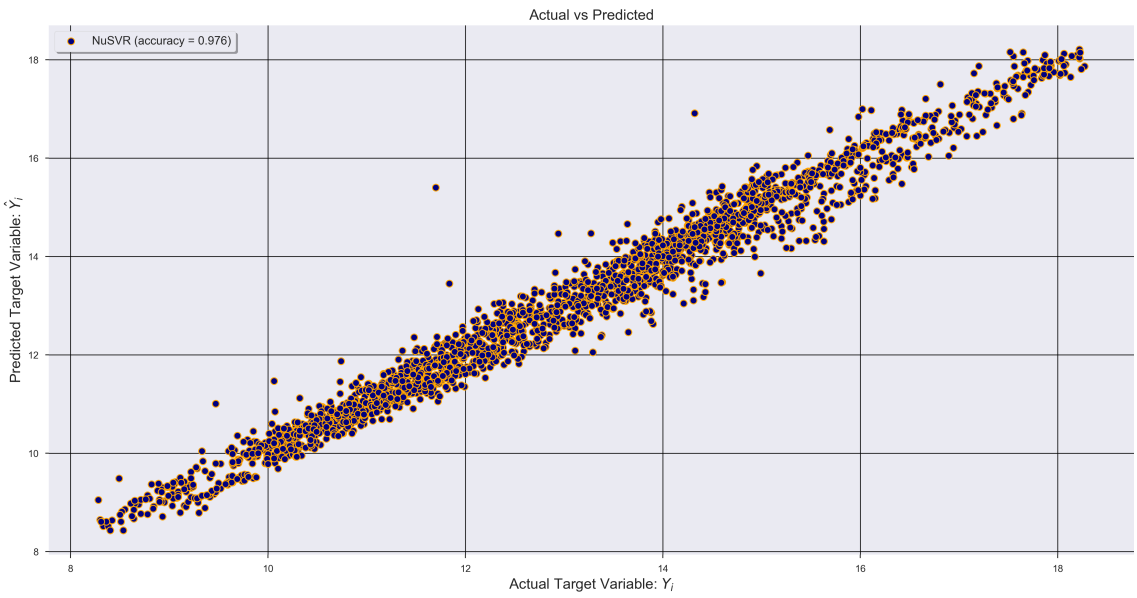

```
Ridge Mean Squared Error: 0.14608640899854475
Ridge Root Mean Squared Error: 0.3822125181081132
Ridge R2 Score: 0.9697837468881153
Ridge Explained Variance Score: 0.9698066436041364
Ridge Mean Absolute Error: 0.2926293335519423
Ridge Meadian Abosulute Error: 0.26566972718727655
Ridge Mean Squared Log Error: 0.0007123218460580986
```



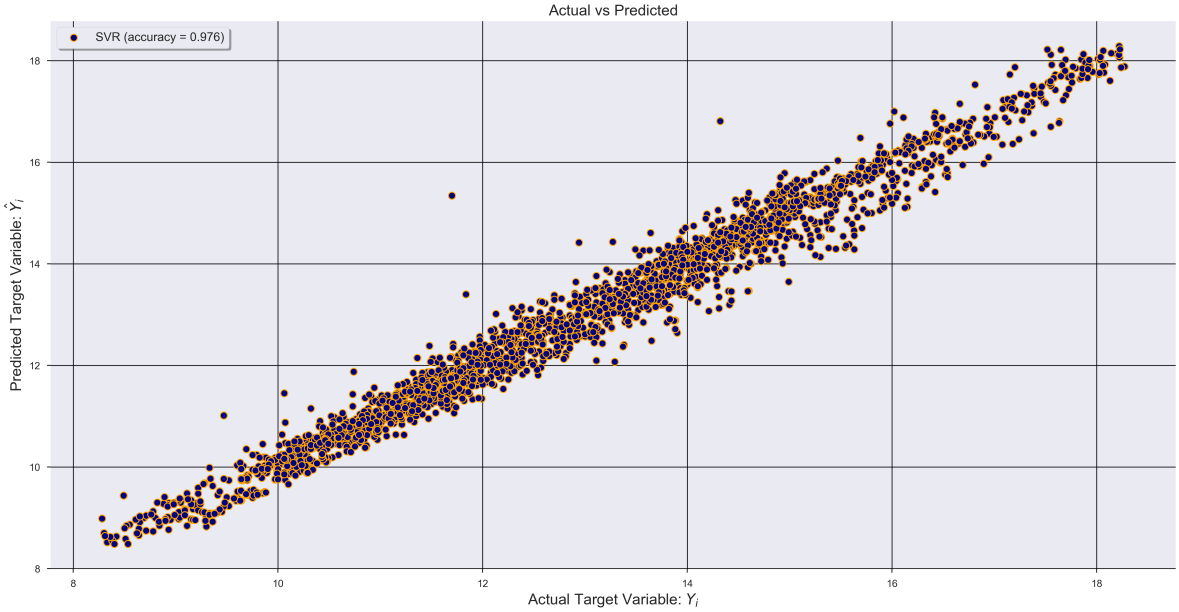
```
SVR Mean Squared Error: 0.14670029477162583
SVR Root Mean Squared Error: 0.383014744848845
SVR R2 Score: 0.9696567718462318
SVR Explained Variance Score: 0.969658812471311
SVR Mean Absolute Error: 0.2912638394608893
SVR Meadian Abosulute Error: 0.2636070991213728
SVR Mean Squared Log Error: 0.0007125399126440713
```



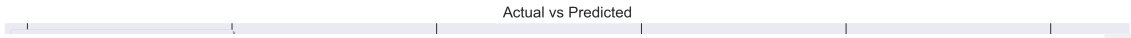

```
NuSVR Mean Squared Error: 0.1151100320397934
NuSVR Root Mean Squared Error: 0.3392786937604444
NuSVR R2 Score: 0.9761908456257131
NuSVR Explained Variance Score: 0.9761920945378169
NuSVR Mean Absolute Error: 0.24964070901547153
NuSVR Meadian Abosulute Error: 0.1964009749299347
NuSVR Mean Squared Log Error: 0.000575800094914912
```



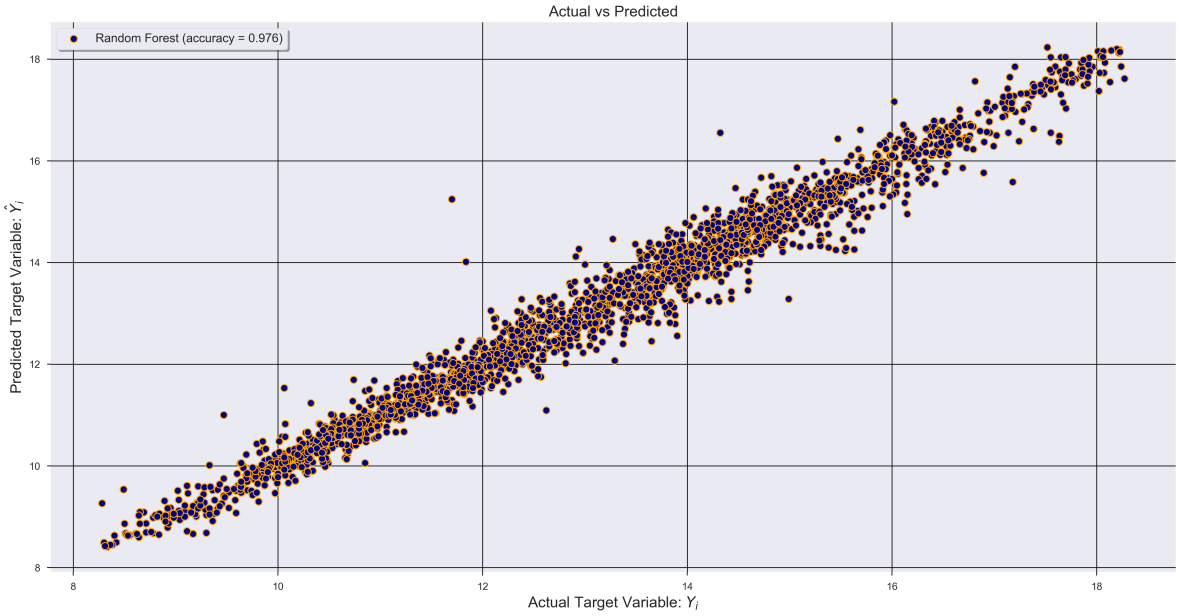
```
SVR Mean Squared Error: 0.1146582577795977
SVR Root Mean Squared Error: 0.338612252849181
SVR R2 Score: 0.9762842898104876
SVR Explained Variance Score: 0.9762930859445609
SVR Mean Absolute Error: 0.2442590031041782
SVR Meadian Abosulute Error: 0.18377625009494558
SVR Mean Squared Log Error: 0.000572986557719098
```



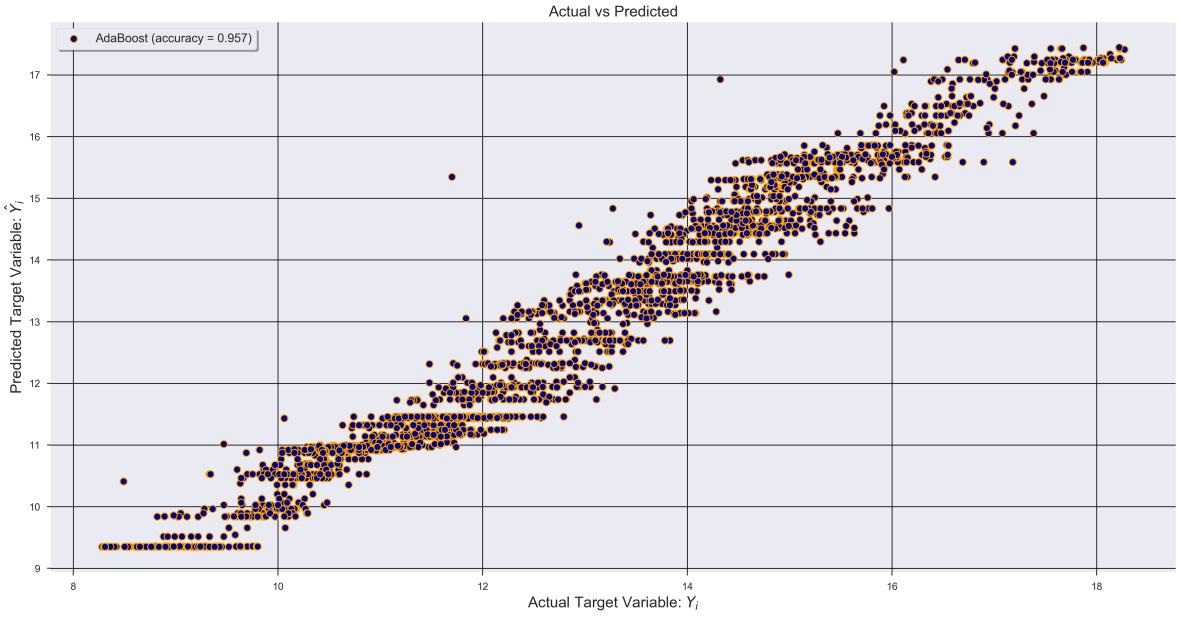

```
Decision Tree Mean Squared Error: 0.18243549963214703
Decision Tree Root Mean Squared Error: 0.4271246886239978
Decision Tree R2 Score: 0.9622653656060981
Decision Tree Explained Variance Score: 0.9623485228821076
Decision Tree Mean Absolute Error: 0.2804296681327466
Decision Tree Meadian Abosulute Error: 0.16000000000000014
Decision Tree Mean Squared Log Error: 0.0009007957414350518
```




```
Random Forest Mean Squared Error: 0.11394277865715507
Random Forest Root Mean Squared Error: 0.3375541121911494
Random Forest R2 Score: 0.9764322782401315
Random Forest Explained Variance Score: 0.9764535491742466
Random Forest Mean Absolute Error: 0.2311272736810036
Random Forest Meadian Abosulute Error: 0.15338000000000207
Random Forest Mean Squared Log Error: 0.0005630956765314753
```

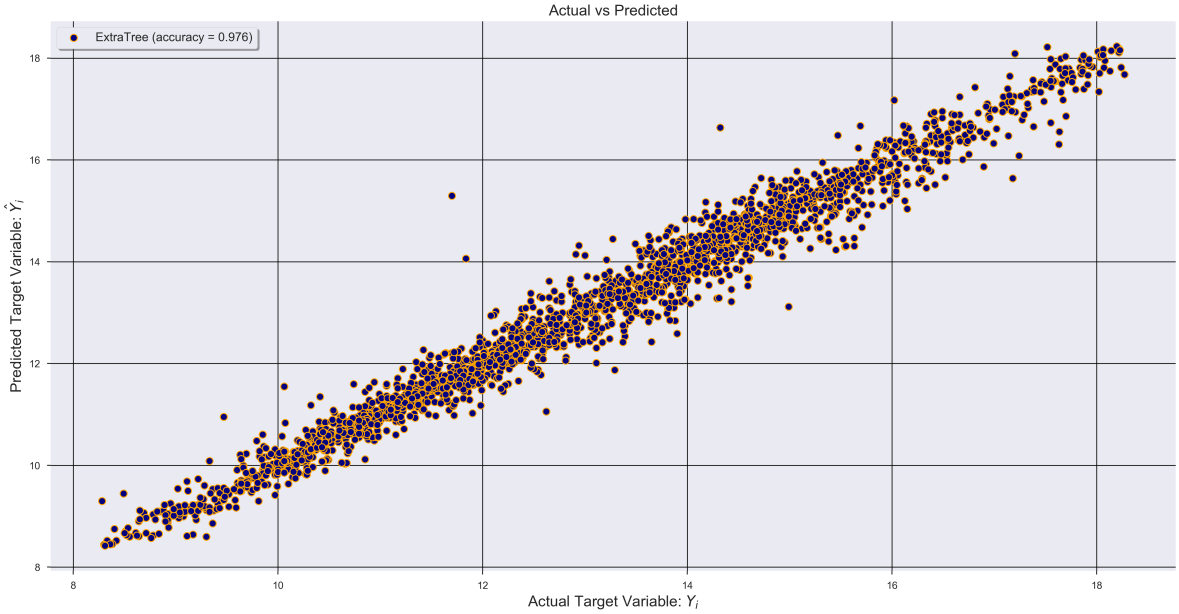



```
AdaBoost Mean Squared Error: 0.20812369962120095
AdaBoost Root Mean Squared Error: 0.4562057645637558
AdaBoost R2 Score: 0.9569520639911224
AdaBoost Explained Variance Score: 0.9569776365109394
AdaBoost Mean Absolute Error: 0.3599828662011853
AdaBoost Meadian Abosulute Error: 0.3025436447166907
AdaBoost Mean Squared Log Error: 0.0011126190000955328
```

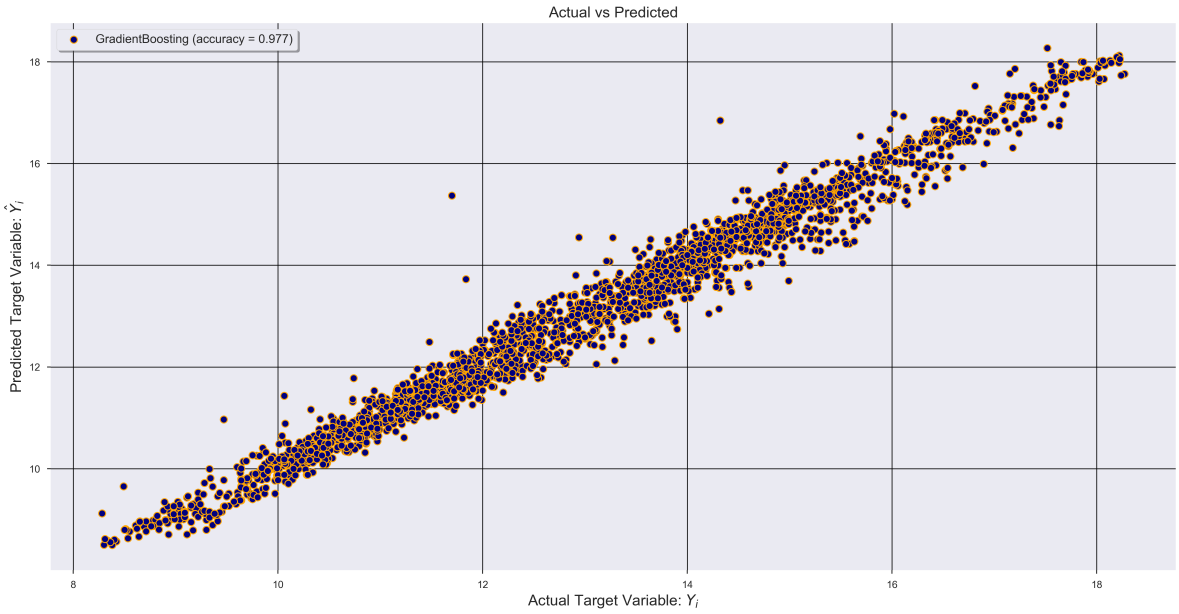



```
ExtraTree Mean Squared Error: 0.11743202522565402
ExtraTree Root Mean Squared Error: 0.3426835642770952
ExtraTree R2 Score: 0.9757105686833952
```

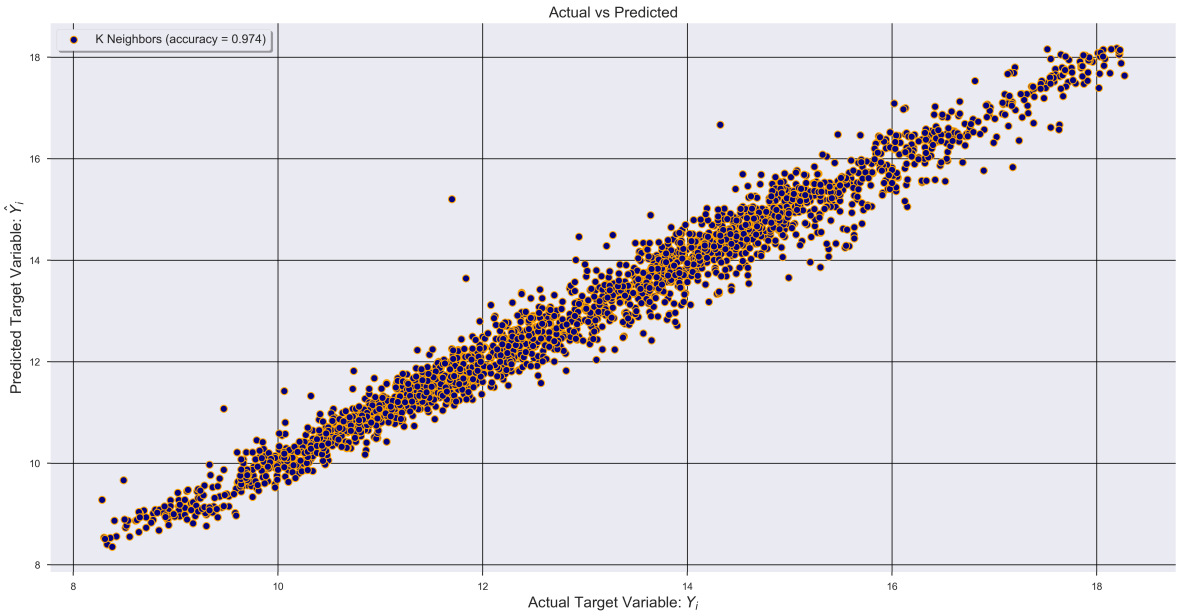
```
ExtraTree Explained Variance Score: 0.9757317960090854
ExtraTree Mean Absolute Error: 0.2303212874850074
ExtraTree Meadian Abosulute Error: 0.14989999999998638
ExtraTree Mean Squared Log Error: 0.0005826065648577845
```




```
GradientBoosting Mean Squared Error: 0.11011792363889294
GradientBoosting Root Mean Squared Error: 0.33184020799007
GradientBoosting R2 Score: 0.9772234044519423
GradientBoosting Explained Variance Score: 0.9772262384355532
GradientBoosting Mean Absolute Error: 0.24210040978092848
GradientBoosting Meadian Abosulute Error: 0.18397069087037288
GradientBoosting Mean Squared Log Error: 0.0005519263607088787
```




```
K Neighbors Mean Squared Error: 0.1242917529426631
K Neighbors Root Mean Squared Error: 0.3525503551872599
K Neighbors R2 Score: 0.9742917147982414
K Neighbors Explained Variance Score: 0.9743001723319055
K Neighbors Mean Absolute Error: 0.25122119152339034
K Neighbors Meadian Abosulute Error: 0.1819999999999986
K Neighbors Mean Squared Log Error: 0.0006201934569084686
```



Cross Validating Models

Cross Validating with a single metric

In [9]:

```
# Splitting data into 10 folds
cv_kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=23)
scorer = "r2"

model_names = []
cv_mean_scores = []
cv_std_scores = []

for model_name, model in models:
    regressor_model = model
    model_scores = model_selection.cross_val_score(
        regressor_model, X, y, cv=cv_kfold, scoring=scorer, n_jobs=-1, verbose=1,
    )

    print(
        f"{model_name} Accuracy: %0.2f (+/- %0.2f)"
        % (model_scores.mean(), model_scores.std() * 2)
    )

    model_names.append(model_name)
    cv_mean_scores.append(model_scores.mean())
    cv_std_scores.append(model_scores.std())
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    1.1s remaining:
0.7s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    1.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

Linear Regression Accuracy: 0.97 (+/- 0.00)
Ridge Accuracy: 0.97 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    1.9s remaining:
1.3s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    2.5s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

SVR Accuracy: 0.86 (+/- 0.21)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    9.0s remaining:
6.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   14.4s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

NuSVR Accuracy: 0.93 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    7.9s remaining:
5.2s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   12.7s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

SVR Accuracy: 0.94 (+/- 0.00)
```

```
Decision Tree Accuracy: 0.96 (+/- 0.00)
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    3.5s remaining:
2.3s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    6.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

Random Forest Accuracy: 0.98 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    0.6s remaining:
0.3s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    1.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

AdaBoost Accuracy: 0.95 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    2.5s remaining:
1.7s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    4.1s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

ExtraTree Accuracy: 0.98 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    1.1s remaining:
0.7s

GradientBoosting Accuracy: 0.98 (+/- 0.00)
K Neighbors Accuracy: 0.98 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    1.7s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    0.0s finished
```

In [10]:

```
cv_results = pd.DataFrame({"model_name": model_names, "mean_score": cv_mean_scores, "st
cv_results.sort_values("mean_score", ascending=False, inplace=True,)
cv_results.to_csv("cross_validation_results.csv", index=True)
cv_results
```

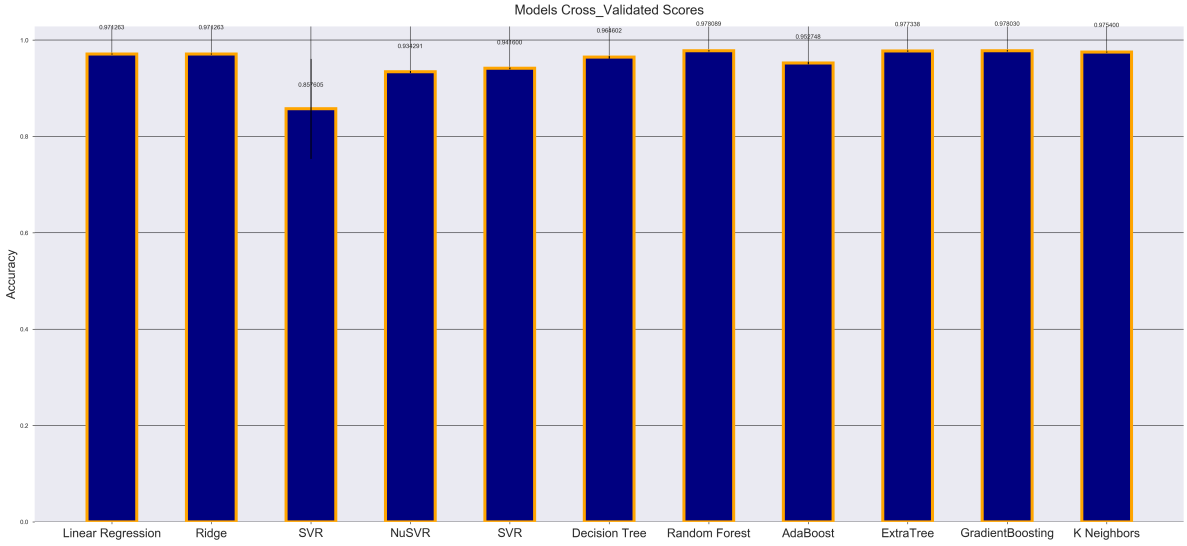
Out[10]:

	model_name	mean_score	std_score
6	Random Forest	0.978089	0.001374
9	GradientBoosting	0.978030	0.001234
8	ExtraTree	0.977338	0.001383
10	K Neighbors	0.975400	0.001267
1	Ridge	0.971263	0.001156
0	Linear Regression	0.971263	0.001156
5	Decision Tree	0.964602	0.002421
7	AdaBoost	0.952748	0.002287
4	SVR	0.941600	0.001811
3	NuSVR	0.934291	0.002104
2	SVR	0.857605	0.103771

Visualizing Cross Validated Models with Bar Charts

In [11]:

```
plt.figure(figsize=(34, 15), dpi=200)

x_loc = np.arange(len(models))
width = 0.5

models_graph = plt.bar(
    x_loc, cv_mean_scores, width, yerr=cv_std_scores, color="navy", edgecolor="orange",
)
plt.title("Models Cross_Validated Scores", fontsize=22, pad=20)
plt.xticks(x_loc, model_names, fontsize=20)
plt.ylabel("Accuracy", fontsize=20)
plt.grid(b=True, which="both", axis="both", color="black", linewidth=0.8)

addLabel(models_graph)

plt.savefig('Bar_Charts_of_Cross_Validated_Models_and_their_Accuracy.png', dpi=300, tra

plt.show()
```



Training the Model with the Highest Score with Default Hyperparameters

In [12]:

```
# Instantiating model object
high_score_model = ensemble.GradientBoostingRegressor()

# Fitting the model on Train set
high_score_model.fit(X_train, y_train)

# Scoring the model on Test set
high_score_model_accuracy = high_score_model.score(X_test, y_test)

print(
    f"Model without tuned hyperparameters has an accuracy of {high_score_model_accuracy
)
```

```
Model without tuned hyperparameters has an accuracy of 0.9772167213704891
```

Predicting with the Trained Model and Saving Predicted Results as `csv`

In [13]:

```
y_pred = high_score_model.predict(X_test)

data = pd.DataFrame({"generation": list(y_test), "predicted_generation": list(y_pred),}
data.to_csv("model_predicted_values.csv", index=True)
```

In [14]:

```
data.head(10)
```

Out[14]:

	generation	predicted_generation
0	17.596	17.663987
1	15.630	15.226165
2	10.850	11.354292
3	14.520	14.603583
4	8.420	8.569025
5	13.640	14.509512
6	14.010	13.864608
7	9.699	9.489637
8	13.900	12.741841
9	11.150	11.255741

Optimizing the Hyperparameter of the Best Model with `GridSearchCV`

In [15]:

```
# Kfold with with n_splits = 5 to split the Dataset into 5-folds
kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=23)

# Dictionary of parameters to tune
parameters = {
    "loss" : ['ls', 'lad', 'huber', 'quantile'],
    "n_estimators" : [120, 500, 800, 1200],
    "max_depth" : [15, 25, 30, None],
    "min_samples_split" : [5, 10, 15, 100],
    "min_samples_leaf" : [1, 2, 5, 10],
    "max_features" : ["log2", "sqrt", None],
}

scorer = "r2"

# Instantiating Search object
grid = model_selection.RandomizedSearchCV(
    estimator=high_score_model,
    param_distributions=parameters,
    scoring=scorer,
    cv=kfold,
    n_jobs=-1,
    verbose=1,
)

# Fit the grid object on Training Dataset
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 10.0min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 11.2min finished
```

Out[15]:

```
RandomizedSearchCV(cv=KFold(n_splits=5, random_state=23, shuffle=True),
                   error_score=nan,
                   estimator=GradientBoostingRegressor(alpha=0.9, ccp_alph
a=0.0,
                                                       criterion='friedman
_mse',
                                                       init=None,
                                                       learning_rate=0.1,
                                                       loss='ls', max_dept
h=3,
                                                       max_features=None,
                                                       max_leaf_nodes=Non
e,
                                                       min_impurity_decrea
se=0.0,
                                                       min_impurity_split=
None,
                                                       min_samples_leaf=1,
                                                       min_samples_split=
2,
                                                       min...
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'loss': ['ls', 'lad', 'huber',
```

```
                                          'quantile'],
                                 'max_depth': [15, 25, 30, None],
                                 'max_features': ['log2', 'sqrt', N
one],
                                 'min_samples_leaf': [1, 2, 5, 10],
                                 'min_samples_split': [5, 10, 15, 1
00],
                                 'n_estimators': [120, 500, 800, 12
00]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='r2', verbose=1)
```

In [16]:

```
results = pd.DataFrame(grid.cv_results_)[["params", "mean_test_score", "std_test_score"
results.sort_values("rank_test_score", inplace=True)
results.to_csv("hyperparameter_tuning_results.csv", index=True)
results
```

Out[16]:

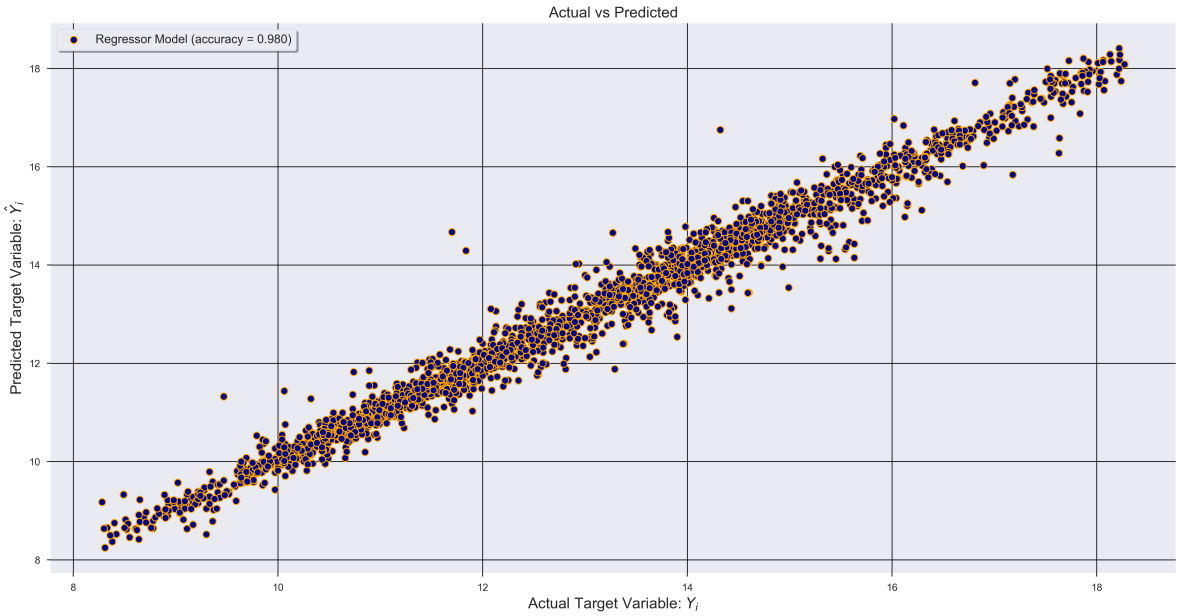
	params	mean_test_score	std_test_score	rank_test_score
9	{'n_estimators': 500, 'min_samples_split': 100...	0.980954	0.001212	1
6	{'n_estimators': 800, 'min_samples_split': 100...	0.980632	0.000622	2
7	{'n_estimators': 500, 'min_samples_split': 100...	0.979432	0.000919	3
4	{'n_estimators': 120, 'min_samples_split': 10,...	0.978602	0.000493	4
2	{'n_estimators': 120, 'min_samples_split': 5, ...	0.978561	0.000687	5
0	{'n_estimators': 1200, 'min_samples_split': 10...	0.977948	0.000639	6
3	{'n_estimators': 1200, 'min_samples_split': 10...	0.977863	0.000600	7
5	{'n_estimators': 800, 'min_samples_split': 10,...	0.976270	0.000749	8
8	{'n_estimators': 800, 'min_samples_split': 5, ...	0.974772	0.000898	9
1	{'n_estimators': 1200, 'min_samples_split': 10...	0.968720	0.001231	10

Evaluating the Best Estimator from the `GridSearch`

In [17]:

```
1 best_estimator = grid.best_estimator_
2 helper.evaluate(X_test, y_test, "Regressor Model", best_estimator)
```

```
Regressor Model Mean Squared Error: 0.09690800378247133
Regressor Model Root Mean Squared Error: 0.31130050398685727
Regressor Model R2 Score: 0.9799557207892775
Regressor Model Explained Variance Score: 0.9799816608782863
Regressor Model Mean Absolute Error: 0.20714495307769037
Regressor Model Meadian Abosulute Error: 0.13735171170935345
Regressor Model Mean Squared Log Error: 0.0004856029538959505
```



Predicting with the Best Estimator and Saving Predicted Results as `csv`

In [18]:

```
tune_y_pred = best_estimator.predict(X_test)

hyp_tune_data = pd.DataFrame(
    {"generation": list(y_test), "predicted_generation": list(tune_y_pred),}
)

hyp_tune_data.to_csv("tune_model_predicted_values.csv", index=True)

hyp_tune_data.head(10)
```

Out[18]:

	generation	predicted_generation
0	17.596	17.759893
1	15.630	15.338581
2	10.850	11.194873
3	14.520	14.460236
4	8.420	8.525759
5	13.640	14.225518
6	14.010	13.788593
7	9.699	9.588924
8	13.900	12.536513
9	11.150	10.931286

Saving the Best Estimator with `joblib`

In [19]:

```
import joblib

joblib.dump(best_estimator, "GradientBoosting.joblib")
```

Out[19]:

```
['GradientBoosting.joblib']
```

Feature Importance

In [20]:

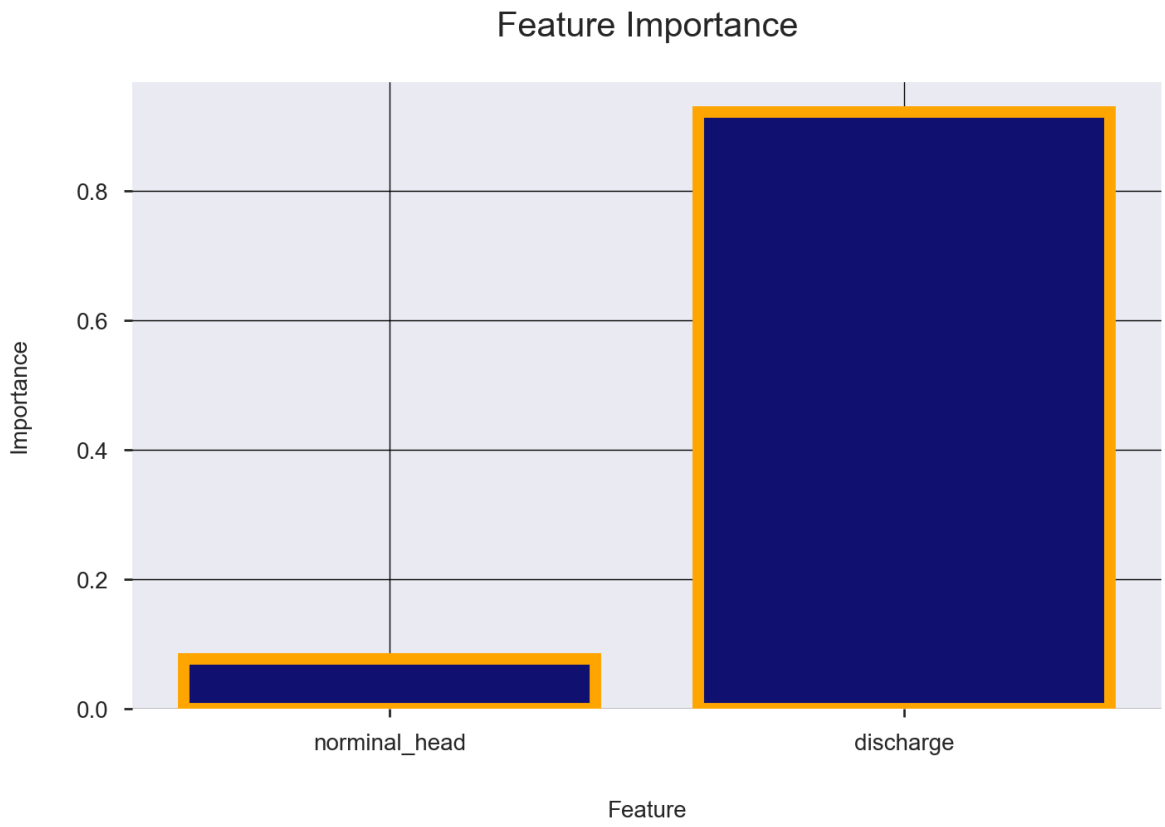
```
features = ['norminal_head', 'discharge']
importances = best_estimator.feature_importances_

feature_importance = pd.DataFrame({
    'feature': features,
    'importance': importances,
})

feature_importance.sort_values("importance", inplace=True, ascending=False)
feature_importance.to_csv("feature_importance.csv", index=True)
feature_importance
```

Out[20]:

	feature	importance
1	discharge	0.92209
0	norminal_head	0.07791

In [21]:

```
import seaborn as sns

plt.figure(figsize=(8, 5), dpi=200)

sns.barplot(x=features, y=importances, color="navy", edgecolor="orange", linewidth=5)
plt.title("Feature Importance", size=15, pad=20)
plt.xlabel("Feature", fontsize=10, labelpad=20)
plt.ylabel("Importance", fontsize=10, labelpad=20)

plt.grid(b=True, which="both", axis="both", color="black", linewidth=0.5)

plt.savefig('feature_importance.png', dpi=300, transparent=True)

plt.show()
```



In [ ]:

```

```

In [ ]:

```

```