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

# 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]:

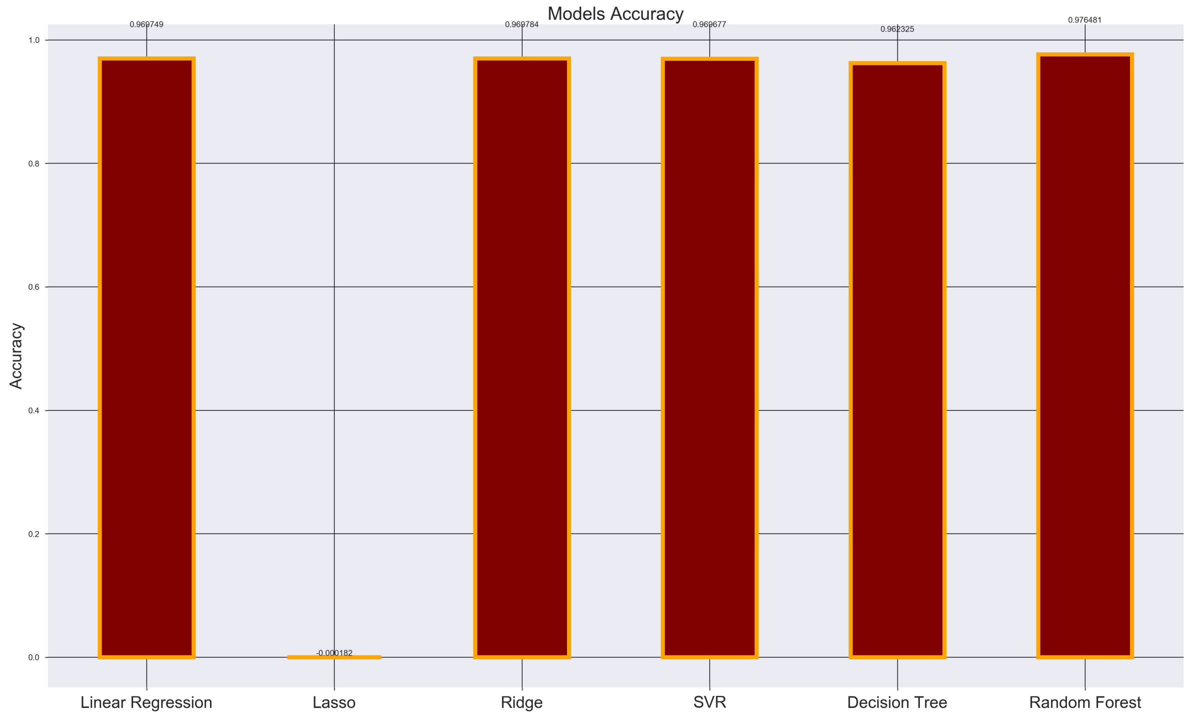
```
# Instantiating baseline models
models = [
    ("Linear Regression", linear_model.LinearRegression()),
    ("Lasso", linear_model.Lasso()),
    ("Ridge", linear_model.Ridge()),
    ("SVR", svm.LinearSVR()),
    ("Decision Tree", tree.DecisionTreeRegressor()),
    ("Random Forest", ensemble.RandomForestRegressor()),
]

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 Lasso -0.00018218953746829136
Akosomba_Data Ridge 0.9697837468881153
Akosomba_Data SVR 0.9696773290140956
Akosomba_Data Decision Tree 0.962324877287925
Akosomba_Data Random Forest 0.9764811116031492
```

Visualizing Models' Accuracy with Bar Charts

In [7]:

```python
# Size in inches (width, height) & resolution(DPI)
plt.figure(figsize=(25, 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="maroon", edgecolor="orange", linewidth=5,
)

plt.title("Models Accuracy", fontsize=22)
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()
```

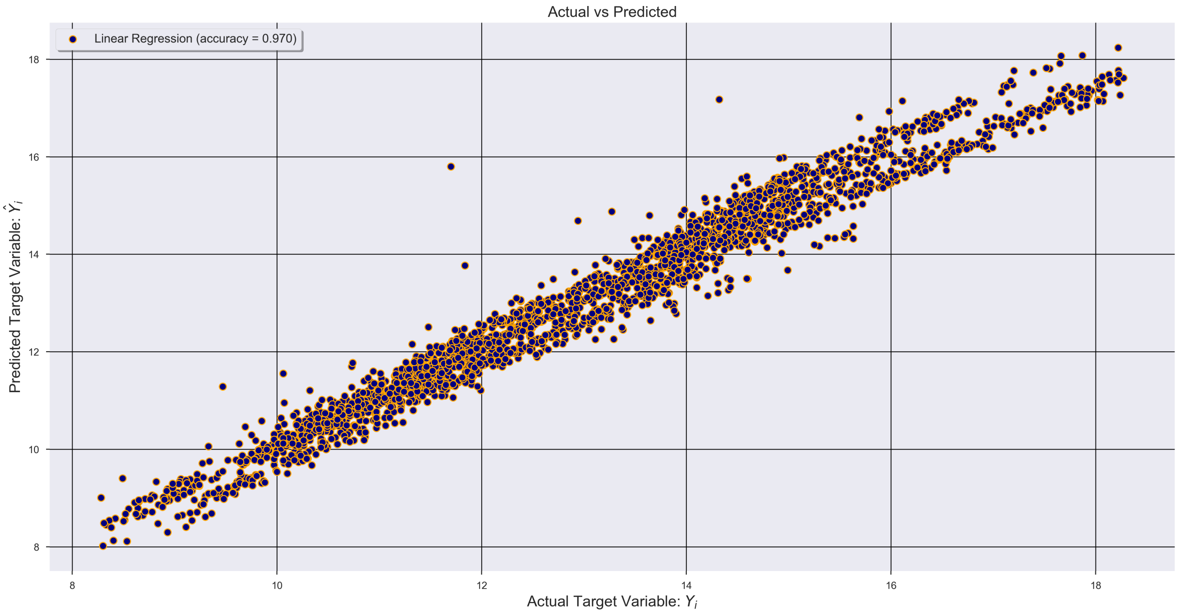


Evaluating Models

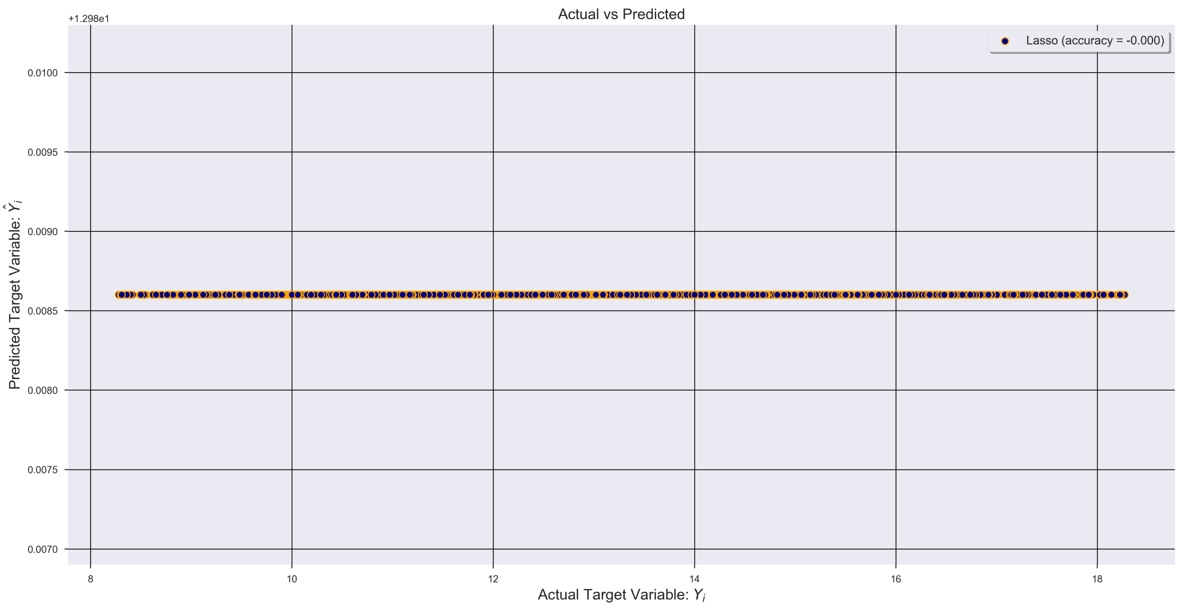
In [8]:

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

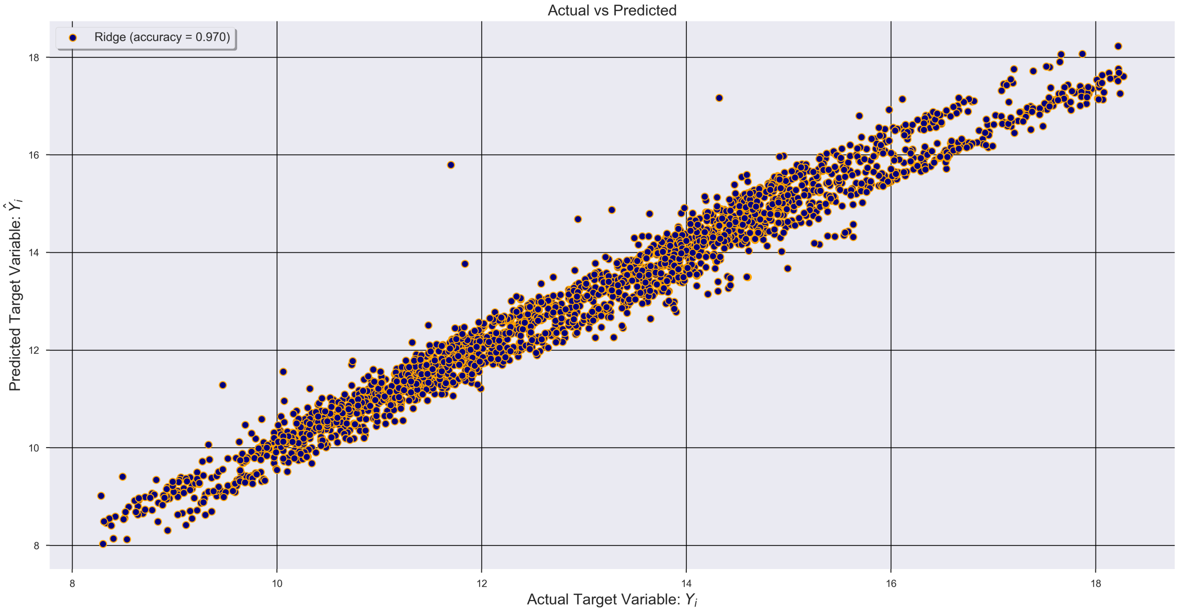
```
Linear Regression Mean Squared Error: 0.14625233556901102
Linear Regression Root Mean Squared Error: 0.38242951712571954
Linear Regression R2 Score: 0.969749426931279
Linear Regression Explained Variance Score: 0.969772234576705
Linear Regression Mean Absolute Error: 0.29263695465956613
Linear Regression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



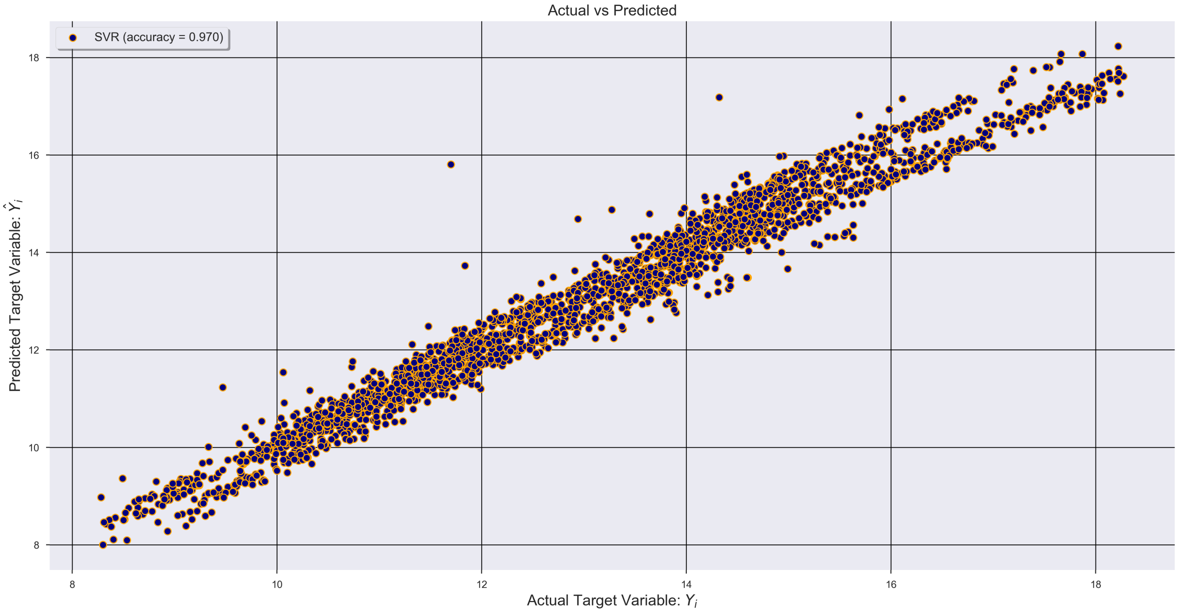

```
Lasso Mean Squared Error: 4.83557719326759
Lasso Root Mean Squared Error: 2.198994586911844
Lasso R2 Score: -0.00018218953746829136
Lasso Explained Variance Score: 0.0
Lasso Mean Absolute Error: 1.8671255680309624
Lasso Meadian Abosulute Error: 1.7113991700829914
Lasso Mean Squared Log Error: 0.0253423068715663
```



```
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
```

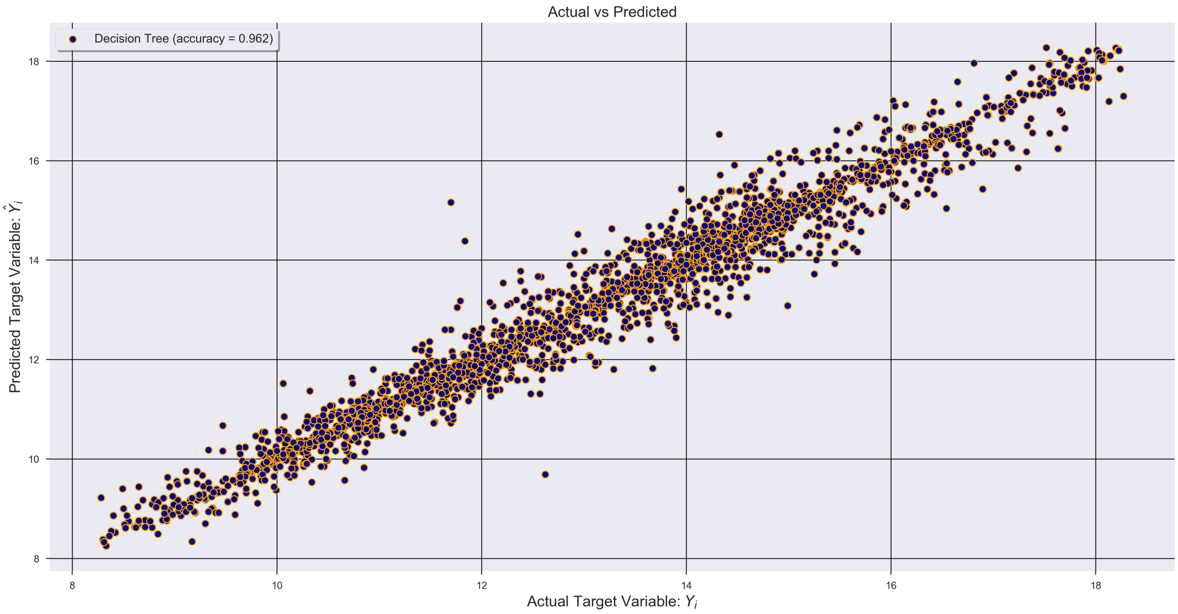



```
SVR Mean Squared Error: 0.14660090710693777
SVR Root Mean Squared Error: 0.3828849789518228
SVR R2 Score: 0.9696773290140956
SVR Explained Variance Score: 0.9696893927526878
SVR Mean Absolute Error: 0.291008593476515 4
SVR Meadian Abosulute Error: 0.26332581829105983
SVR Mean Squared Log Error: 0.0007128432992722564
```

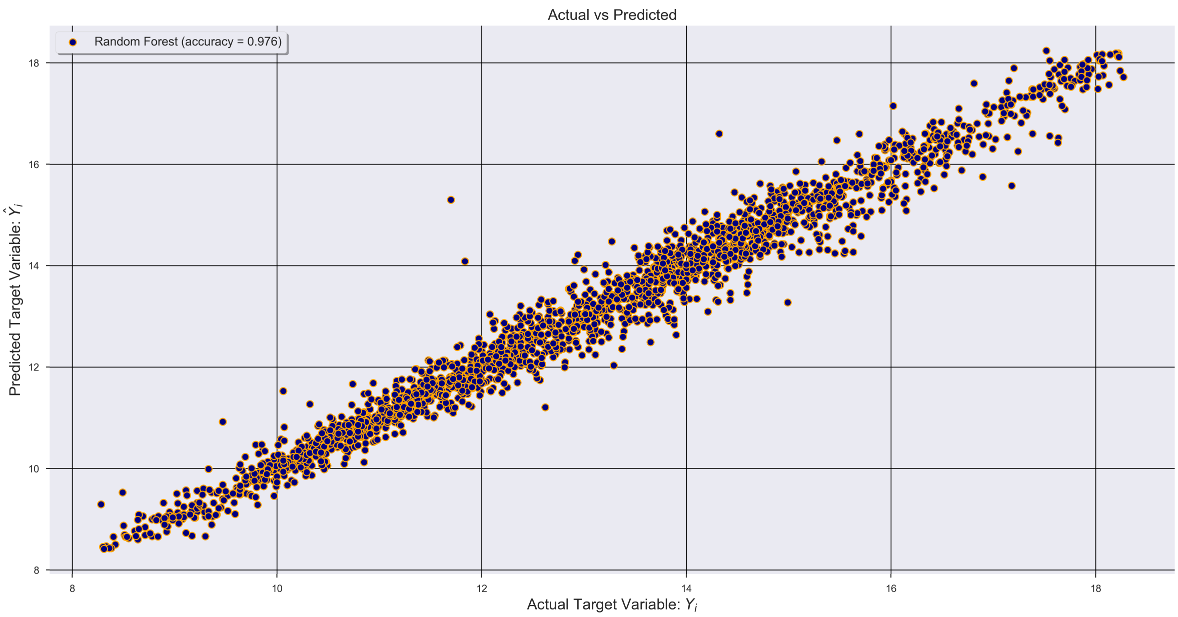



```
Decision Tree Mean Squared Error: 0.18214777872051172
Decision Tree Root Mean Squared Error: 0.4267877443419758
Decision Tree R2 Score: 0.962324877287925
Decision Tree Explained Variance Score: 0.9623848910919239
Decision Tree Mean Absolute Error: 0.2797403438624543
```

```
Decision Tree Meadian Abosulute Error: 0.15500000000000114
Decision Tree Mean Squared Log Error: 0.0009010063568695894
```




```
Random Forest Mean Squared Error: 0.11370668417462113
Random Forest Root Mean Squared Error: 0.337204217314406
Random Forest R2 Score: 0.9764811116031491
Random Forest Explained Variance Score: 0.9765071315241614
Random Forest Mean Absolute Error: 0.23050171260067406
Random Forest Meadian Abosulute Error: 0.15107999999999855
Random Forest Mean Squared Log Error: 0.0005612420264805661
```



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.
[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)
Lasso Accuracy: 0.95 (+/- 0.00)
Ridge Accuracy: 0.97 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    1.8s remaining:
1.2s
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    2.6s 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: 0.84 (+/- 0.11)
Decision Tree Accuracy: 0.96 (+/- 0.01)

[Parallel(n_jobs=-1)]: Done   6 out of  10 | elapsed:    3.5s remaining:
2.3s

Random Forest Accuracy: 0.98 (+/- 0.00)

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    6.0s 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
```

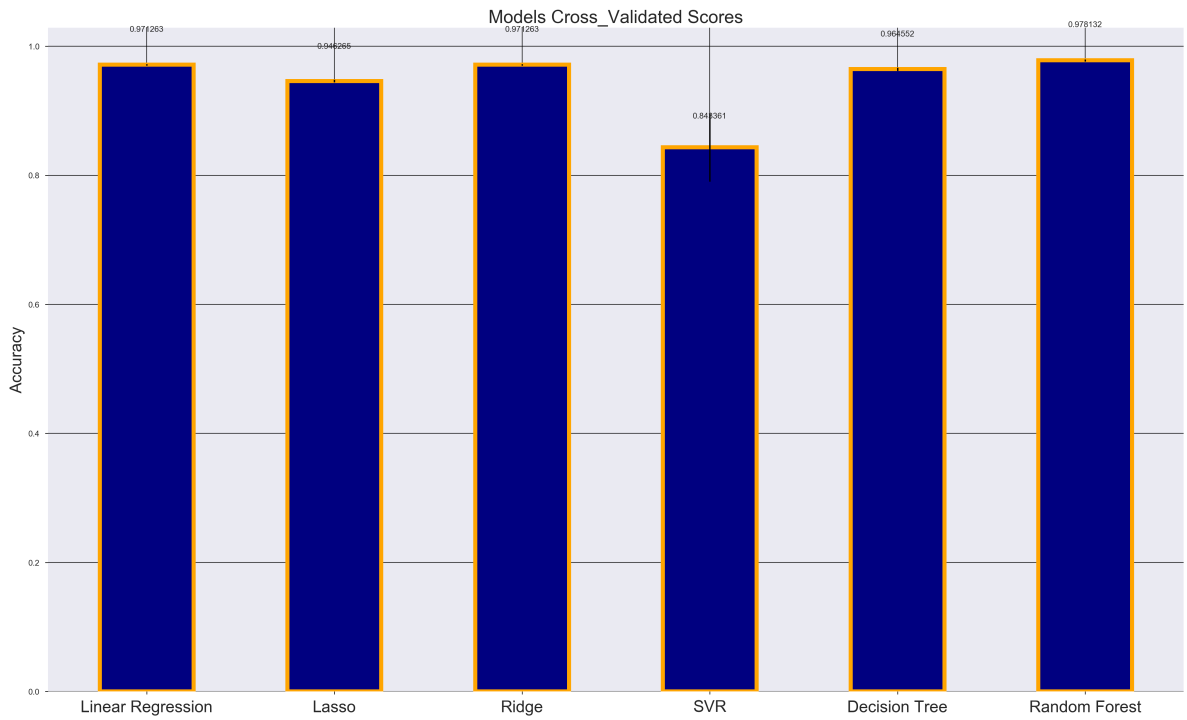
Out[10]:

	model_name	mean_score	std_score
5	Random Forest	0.978132	0.001437
2	Ridge	0.971263	0.001156
0	Linear Regression	0.971263	0.001156
4	Decision Tree	0.964552	0.002598
1	Lasso	0.946265	0.001854
3	SVR	0.843361	0.052938

Visualizing Cross Validated Models with Bar Charts

In [11]:

```
plt.figure(figsize=(25, 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)
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.RandomForestRegressor()

# 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 0.9763555356961289
```

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

In [13]:

```
y_pred = high_score_model.predict(X_test)

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

In [14]:

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

Out[14]:

	actual_generation	predicted_generation
0	17.596	17.875130
1	15.630	15.209890
2	10.850	11.478600
3	14.520	14.756200
4	8.420	8.510460
5	13.640	14.302863
6	14.010	13.693800
7	9.699	9.551900
8	13.900	12.544700
9	11.150	10.977800

Optimizing the Hyperparameter of the Best Model with `GridSearchCV`

In [16]:

```
# 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 = {
    "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:   47.0s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  1.4min finished
```

Out[16]:

```
RandomizedSearchCV(cv=KFold(n_splits=5, random_state=23, shuffle=True),
                   error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.
0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=
0....
                                                   random_state=None, verbos
e=0,
                                                   warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [15, 25, 30, None],
                                        'max_features': ['log2', 'sqrt', Non
e],
                                        'min_samples_leaf': [1, 2, 5, 10],
```

```
                                           'min_samples_split': [5, 10, 15, 10
0],
                                           'n_estimators': [120, 500, 800, 120
0]},
                     pre_dispatch='2*n_jobs', random_state=None, refit=True,
                     return_train_score=False, scoring='r2', verbose=1)
```

In [17]:

```
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[17]:

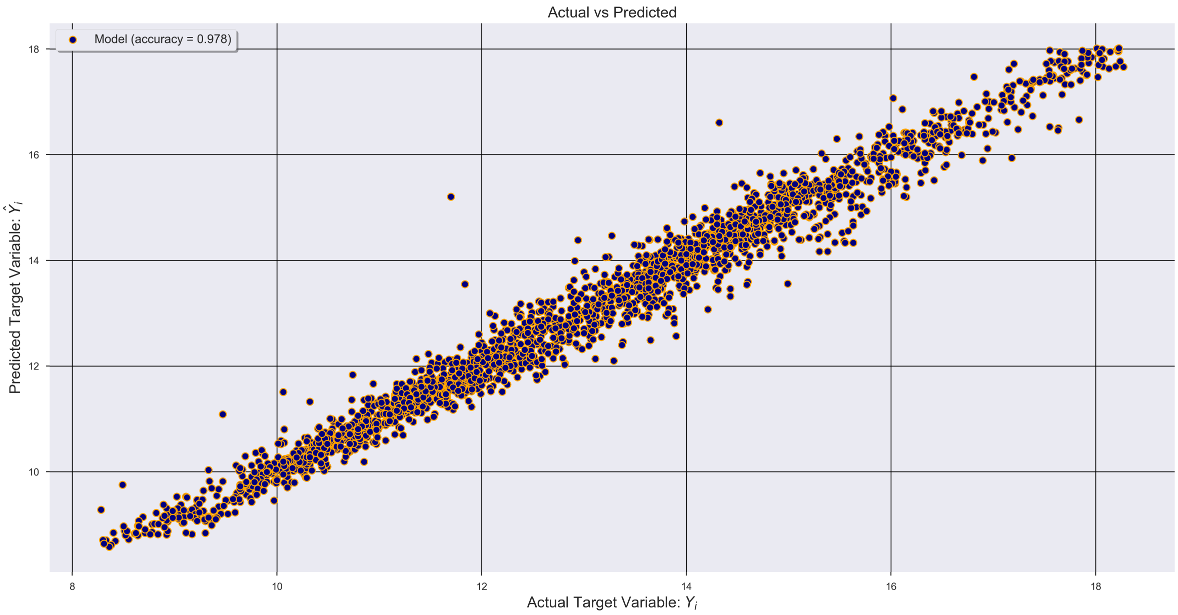
	params	mean_test_score	std_test_score	rank_test_score
7	{'n_estimators': 1200, 'min_samples_split': 10...	0.978835	0.000849	1
8	{'n_estimators': 1200, 'min_samples_split': 15...	0.978787	0.000782	2
4	{'n_estimators': 800, 'min_samples_split': 15,...	0.978703	0.000840	3
0	{'n_estimators': 1200, 'min_samples_split': 5,...	0.978700	0.000827	4
9	{'n_estimators': 1200, 'min_samples_split': 15...	0.978694	0.000824	5
2	{'n_estimators': 800, 'min_samples_split': 10,...	0.978583	0.000835	6
1	{'n_estimators': 1200, 'min_samples_split': 10...	0.978556	0.000854	7
6	{'n_estimators': 120, 'min_samples_split': 5, ...	0.978369	0.000830	8
3	{'n_estimators': 500, 'min_samples_split': 10,...	0.977691	0.001001	9
5	{'n_estimators': 800, 'min_samples_split': 100...	0.971914	0.001033	10

Evaluating the Best Estimator from the `GridSearch`

In [18]:

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

```
Model Mean Squared Error: 0.1083768208580254
Model Root Mean Squared Error: 0.32920634996613507
Model R2 Score: 0.9775835310556501
Model Explained Variance Score: 0.9775842437878877
Model Mean Absolute Error: 0.23278235862104127
Model Meadian Abosulute Error: 0.1657286307725485
Model Mean Squared Log Error: 0.0005422388176224308
```



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

In [19]:

```
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[19]:

	generation	predicted_generation
0	17.596	17.742404
1	15.630	15.212326
2	10.850	11.358161
3	14.520	14.700633
4	8.420	8.687071
5	13.640	14.385984
6	14.010	13.830590
7	9.699	9.565270
8	13.900	12.565679
9	11.150	11.021706

Saving the Best Estimator with `joblib`

In [20]:

```
import joblib

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

Out[20]:

```
['Random_Forest_Regressor.joblib']
```

Feature Importance

In [21]:

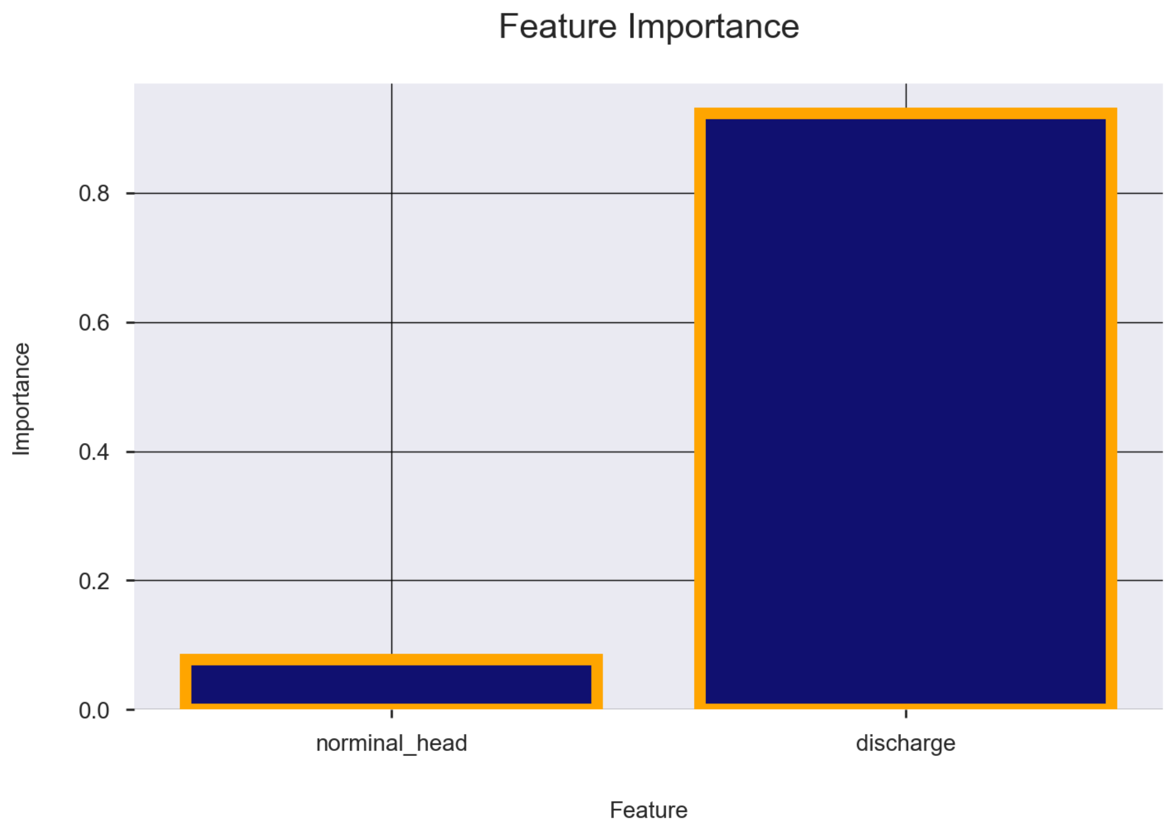
```
features = ['norminal_head', 'discharge']
importances = best_estimator.feature_importances_

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[21]:

	feature	importance
1	discharge	0.922701
0	norminal_head	0.077299

In [22]:

```python
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 [ ]:

```python

```